КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2026 Т. 18 № 1 С. 133–147
DOI: 10.20537/2076-7633-2026-18-1-133-147

Ки&М

МОДЕЛИ В ФИЗИКЕ И ТЕХНОЛОГИИ

УДК: 004.9:004.5:61

Использование сервис-контейнеров Docker для создания систем поддержки принятия врачебных решений (СППВР) на базе веб-браузера

Г. В. Копытов[1,2,a], А. Н. Дроздов[1,b]

[1]ИВМ РАН,
Россия, 119333, г. Москва, ул. Губкина, д. 8
[2]Балтийский федеральный университет,
Россия, 236041, г. Калининград, ул. Александра Невского, д. 14

E-mail: [a] gkopytov@yandex.ru, [b] andrei.drozdov1@mail.ru

Получено 11.11.2025, после доработки — 28.11.2025.
Принято к публикации 29.11.2025.

В статье представлена технология построения систем поддержки принятия врачебных решений (СППВР), основанная на сервис-контейнерах с использованием Docker и веб-интерфейсе, работающем непосредственно в браузере без установки специализированного программного обеспечения на рабочую станцию врача. Предложена модульная архитектура, где для каждого прикладного модуля формируется самостоятельный сервис-контейнер, объединяющий мини-веб-сервер, пользовательский интерфейс и вычислительные компоненты обработки медицинских изображений. Взаимодействие между браузером и серверной частью реализовано через постоянное двунаправленное соединение по веб-сокетам с бинарной сериализацией сообщений в формате MessagePack, что обеспечивает малые задержки и эффективную передачу больших объемов данных. Для локального хранения изображений и результатов анализа применены средства браузера (IndexedDB) с оболочкой Dexie.js, что ускоряет повторный доступ к данным. Трехмерная визуализация и базовые операции с DICOM-данными реализованы с использованием библиотек Three.js и AMI.js: такая связка обеспечивает интеграцию интерактивных элементов, возникающих в контексте задачи (аннотации, ориентиры, метки, 3D-модели), в трехмерные медицинские изображения.

Серверные компоненты и функциональные модули собраны в виде набора взаимодействующих контейнеров, управляемых средствами Docker. Рассмотрены выбор базовых образов, подходы к минимизации контейнеров до уровня исполняемых файлов без внешних зависимостей, организация многоступенчатой сборки, включающей отдельный «сборочный» контейнер. Описан сервис-хаб, выполняющий запуск прикладных контейнеров по обращению пользователя, проксирование запросов, управление сессиями и перевод контейнера из общего режима в монопольный при начале вычислений. Приведены примеры прикладных модулей (оценка фракционного резерва кровотока, расчет количественного отношения потока, расчет смыкания створок аортального клапана), показана интеграция React-интерфейса и трехмерной сцены, а также представлены политика версионирования, автоматизированные проверки воспроизводимости результатов и порядок развертывания на целевой площадке.

Продемонстрировано, что контейнеризация обеспечивает переносимость и воспроизводимость программной среды, изоляцию зависимостей и масштабирование, а браузерный интерфейс — доступность, снижение требований к инфраструктуре и интерактивную визуализацию медицинских данных в реальном времени. Отмечены технические ограничения (зависимость от версий библиотек визуализации и форматов данных) и представлены практические меры их преодоления.

Ключевые слова: системы поддержки принятия врачебных решений (СППВР), zero-footprint-приложения, сервис-контейнеры, веб-приложение

Работа выполнена при поддержке Отделения Московского центра фундаментальной и прикладной математики в ИВМ РАН (Соглашение с Минобрнауки России № 075-15-2025-347).

COMPUTER RESEARCH AND MODELING
2026 VOL. 18 NO. 1 P. 133–147
DOI: 10.20537/2076-7633-2026-18-1-133-147

Ки&М

MODELS IN PHYSICS AND TECHNOLOGY

UDC: 004.9:004.5:61

Using Docker service containers to build browser-based clinical decision support systems (CDSS)

G. V. Kopytov[1,2,a], A. N. Drozdov[1,b]

[1]INM RAS,
8 Gubkin st., Moscow, 119333, Russia
[2]Baltic Federal University,
14 Alexander Nevsky st., Kaliningrad, 236041, Russia

E-mail: [a] gkopytov@yandex.ru, [b] andrei.drozdov1@mail.ru

Received 11.11.2025, after completion — 28.11.2025.
Accepted for publication 29.11.2025.

The article presents a technology for building clinical decision support systems (CDSS) based on service containers using Docker and a web interface that runs directly in the browser without installing specialized software on workstation of a clinician. A modular architecture is proposed in which each application module is packaged as an independent service container combining a lightweight web server, a user interface, and computational components for medical image processing. Communication between the browser and the server side is implemented via a persistent bidirectional WebSocket connection with binary message serialization (MessagePack), which provides low latency and efficient transfer of large data. For local storage of images and analysis of results, browser facilities (IndexedDB with the Dexie.js wrapper) are used to speed up repeated data access. Three-dimensional visualization and basic operations with DICOM data are implemented with Three.js and AMI.js: this toolchain supports the integration of interactive elements arising from the task context (annotations, landmarks, markers, 3D models) into volumetric medical images.

Server components and functional modules are assembled as a set of interacting containers managed by Docker. The paper discusses the choice of base images, approaches to minimizing containers down to runtime-only executables without external utilities, and the organization of multi-stage builds with a dedicated build container. It describes a hub service that launches application containers on user request, performs request proxying, manages sessions, and switches a container from shared to exclusive mode at the start of computations. Examples of application modules are provided (fractional flow reserve estimation, quantitative flow ratio computation, aortic valve closure modeling), along with the integration of a React-based interface with a three-dimensional scene, a versioning policy, automated reproducibility checks, and the deployment procedure on the target platform.

It is demonstrated that containerization ensures portability and reproducibility of the software environment, dependency isolation and scalability, while the browser-based interface provides accessibility, reduced infrastructure requirements, and interactive real-time visualization of medical data. Technical limitations are noted (dependence on versions of visualization libraries and data formats) together with practical mitigation measures.

Keywords: clinical decision support systems (CDSS), zero-footprint applications, service containers, web application

Citation: *Computer Research and Modeling*, 2026, vol. 18, no. 1, pp. 133–147 (Russian).

This work was supported by the Moscow Center of Fundamental and Applied Mathematics at INM RAS (Agreement with the Ministry of Education and Science of the Russian Federation No. 075-15-2025-347).

Используемые термины

API-сервер — серверное приложение, которое принимает и обрабатывает запросы по API (интерфейсу прикладного программирования).

Document Object Model (DOM) — программный интерфейс и структура представления HTML-документов в виде дерева объектов, где каждый элемент, атрибут и текстовый блок представлены отдельными узлами.

Docker — программная платформа для упаковки приложений с зависимостями в переносимые контейнеры и автоматизации их развертывания и запуска.

Docker Hub (докер-хаб) — облачный сервис и публичный репозиторий образов Docker-контейнеров.

HTTP-запрос — сообщение, отправляемое клиентом на сервер для получения ресурса или выполнения действия по протоколу HTTP.

UI (User Interface, пользовательский интерфейс) — визуальная и интерактивная часть приложения или сайта, через которую пользователь взаимодействует с продуктом.

Zero-footprint-приложение — веб-приложение, работающее без установки на устройство пользователя и доступное через браузер, без необходимости локального ПО.

Бэкенд — серверная часть приложения или сайта, где происходит обработка данных.

Веб-интерфейс — совокупность средств, позволяющая пользователю взаимодействовать с сайтом, программой или сервисом через браузер.

Веб-сокет — сетевой протокол для постоянного двустороннего обмена данными между браузером и сервером в реальном времени.

Деплой — процесс переноса и запуска готового программного обеспечения на сервере (развертывание).

Парсинг — автоматизированный процесс извлечения, сбора и структурирования данных из источников (например, веб-сайтов) с помощью специальных программ — парсеров.

Проксирование — передача сетевых запросов и данных через промежуточный сервер (прокси), который выступает посредником между клиентом и целевым ресурсом.

Рендеринг — процесс отображения содержимого сайта или приложения в браузере.

Сервис-контейнер — Docker-контейнер с приложением СППВР.

Сервис-хаб — специализированный веб-сервер, который управляет запуском приложений СППВР.

Фронтенд — клиентская часть сайта или приложения, отвечающая за внешний вид, интерфейс и взаимодействие с пользователем.

Введение

В статье представлена технология построения систем поддержки принятия врачебных решений (СППВР), основанная на сервис-контейнерах с использованием Docker и веб-интерфейсе, работающем непосредственно в браузере без установки специализированного программного обеспечения на рабочую станцию врача (zero-footprint-приложение). Отсутствие необходимости установки программного обеспечения на устройство дает ряд преимуществ. Можно использовать приложение с любого устройства, имеющего доступ в интернет, без привязки к конкретному компьютеру или мобильному устройству. Отсюда легкость доступа: приложения доступны сразу после открытия браузера. Такие приложения позволяют экономить ресурсы локального компьютера, так как часть вычислений и обработка данных могут происходить на серверах провайдера услуги; это существенно, если в процессе обработки используются ресурсоемкие алгоритмы, как, например, сегментация аорты в трехмерном объеме DICOM-данных. Обработка на удаленном сервере, а не локальном компьютере, позволяет масштабировать zero-footprint-приложения,

а также использовать аппаратные возможности удаленного сервера, например многоядерность. Еще одним преимуществом zero-footprint-приложений является легкость обновления программного обеспечения: провайдер автоматически обновляет версию приложения, пользователю не нужно ничего делать для обновления приложения.

Концепция zero-footprint-приложений, не требующих установки на локальное устройство и запускаемых непосредственно в веб-браузере, известна не первый год и активно развивается в различных областях. К числу наиболее известных решений этого класса относятся облачные офисные сервисы (например, Google Docs, Microsoft Office 365), онлайн-редакторы кода (CodePen, JSFiddle), а также платформы для совместной работы (Trello, Asana).

Аналогичные подходы постепенно внедряются и в медицинскую визуализацию. В обзоре [Pereira et al., 2025] анализируются 16 веб-просмотрщиков, оцениваются их возможности 2D- и 3D-рендеринга, особенности интерфейса и производительность в разных браузерах. В статье [AboArab et al., 2024] обсуждаются подходы построения прогрессивных веб-приложений для визуализации DICOM и MPR (многоплоскостной реконструкции). В другом исследовании [Hostetter et al., 2018] рассматривается интеграция zero-footprint облачных PACS и веб-форм для клинической работы с изображениями. Среди существующих zero-footprint-решений для медицины можно выделить следующие системы.

- **OHIF DICOM Viewer** [OHIF] — браузерное приложение, которое позволяет загружать и просматривать 2D- и 3D-DICOM-изображения, проводить мультипланарную реконструкцию, осуществлять сегментацию и оконтуривание и многое другое. Оно использует свою библиотеку UI-компонентов, а для DICOM-рендеринга — низкоуровневую библиотеку Cornerstone.js [Cornerstone.js] и VTK.js [VTK.js]. В качестве транспилятора и сборщика кода используются Babel [Babel] и Webpack [Webpack].

- **Med3Web** [Med3Web] позволяет просматривать в браузере объемные данные в форматах DICOM и NIfTI. Приложение использует React для построения пользовательского интерфейса и Three.js для 3D-рендеринга. Отмечается собственная реализация 3D-текстур, так как их нативная поддержка в Three.js появилась позже.

- **DWV (DICOM Web Viewer)** [DWV] — просмотрщик медицинских изображений на чистом JavaScript без сторонних библиотек для 2D-рендеринга, на возможностях HTML5 canvas; легко интегрируется в Angular, React и Vue.

Помимо классических решений, ориентированных на визуализацию томографических изображений (CT, MRI), zero-footprint-технологии находят применение и в смежных областях медицинской визуализации. Так, в проекте **VICTORIA** реализовано браузерное средство анализа дозовых распределений в радиационной терапии, поддерживающее интерактивную работу с воксельными полями доз и трехмерное оконтуривание зон интереса [Badun et al., 2021]. В системе **SLIM** (Slide Microscopy Viewer and Annotation Tool) разработан веб-интерфейс для визуализации микроскопических изображений в формате DICOMweb и их аннотирования в реальном времени [Gorman et al., 2023]. Оба проекта демонстрируют расширение применения zero-footprint-архитектур за пределы стандартных сценариев визуализации.

Zero-footprint-приложения в медицинской области, такие как системы поддержки принятия врачебных решений (СППВР), имеют ряд особенностей, связанных с приватностью медицинских данных и потенциально большим объемом обрабатываемых данных, если речь идет об обработке DICOM-данных различной модальности. Кроме того, из-за возможных ограничений на доступ в интернет приложение должно работать внутри периметра медицинского учреждения и без какого-либо выхода в интернет.

Исследования в рамках разрабатываемой технологии по использованию анимированной компьютерной 3D-графики в браузере начались в 2016–2017 годах в работах по моделированию биоимпедансных исследований [Danilov et al., 2017]. В то время на базе технологии WebGL появились фреймворки и библиотеки Babylon.js [Babylon.js], Three.js [Three.js] и другие, которые упрощали использование 3D-графики в веб-приложениях. Наши первоначальные опыты по созданию браузерных zero-footprint-приложений велись с использованием Google Closure Library [Closure library] как библиотеки UI-компонентов и оказались не очень успешными из-за трудоемкого процесса программирования веб-интерфейса с использованием этой библиотеки. Дальнейшее развитие передвинулось на библиотеку React [R3F] и различные библиотеки UI-компонентов на его основе; в качестве основы для 3D-рендеринга была взята библиотека Three.js [Three.js]. Для обработки и рендеринга DICOM-изображений была использована библиотека AMI.js [AMI.js].

Статья организована следующим образом: в первом разделе дается общая архитектура системы, далее описывается процесс разработки компонентов сервис-контейнера, описываются процесс проксирования HTTP- и WebSocket-запросов в сервис-хабе и процесс сборки и развертывания сервис-контейнера СППВР. В заключение дается оценка временных затрат при использовании данной технологии для создания СППВР.

Общая архитектура

Разрабатываемая технология нашла свое воплощение при разработке веб-приложений для расчета фракционного резерва кровотока (FFR) и расчета количественного отношения потока (QFR), востребованных при диагностике коронарных сосудов, а также веб-приложения Ozaki для расчета диастолического состояния аортального клапана, реконструированного по методу С. Озаки. Все три приложения являются системами поддержки принятия врачебных решений (СППВР), принадлежащих категории zero-footprint-приложений.

Общая технологическая схема представлена на рис. 1, она может служить ориентиром для дальнейших разделов. Показанные на рис. 1 СППВР скомпонованы внутри Docker-контейнера вместе с бэкендом, выполняющим обработку DICOM-данных различной модальности. Скомпонованные в одном контейнере бэкенд и фронтенд образуют полноценное приложение, доступное из браузера после запуска сервис-контейнера. Сервис-хаб в этой архитектуре берет на себя функции запуска контейнеров СППВР и проксирования пользовательских запросов в нужные контейнеры. Более подробно работа хаба описана в разделе «Сервис-хаб».

Веб-интерфейс, как правило, реализован в виде Single Page Application на React или на другом фреймворке. Сервис-контейнер одновременно выступает мини-веб-сервером, обслуживающим статический HTML-контент веб-приложения, и API-сервером, который по своей природе может быть разным (например, REST API или PHP API).

На практике REST API оказался неудобен для СППВР: по определению он предполагает отсутствие состояния (stateless), тогда как разрабатываемые СППВР являются приложениями с состоянием (stateful). По этой причине API всех СППВР реализован поверх веб-сокетов с использованием библиотеки LWS [Libwebsockets]. Использование веб-сокетов дает возможность эффективной передачи двоичных данных, составляющих основной объем в системах поддержки принятия врачебных решений. Главным образом это DICOM-данные, объем которых может достигать сотен мегабайт. Веб-сокеты позволяют реализовать не только традиционный механизм RPC (запрос – ответ), но и потоковую передачу данных со стороны клиента, используемую для передачи DICOM.

Для сериализации двоичных данных применяется MessagePack [MessagePack, 2025], который обеспечивает компактное кодирование и доступен на многих языках программирования, включая JavaScript и C++. Таким образом, формат сериализации MessagePack является общим

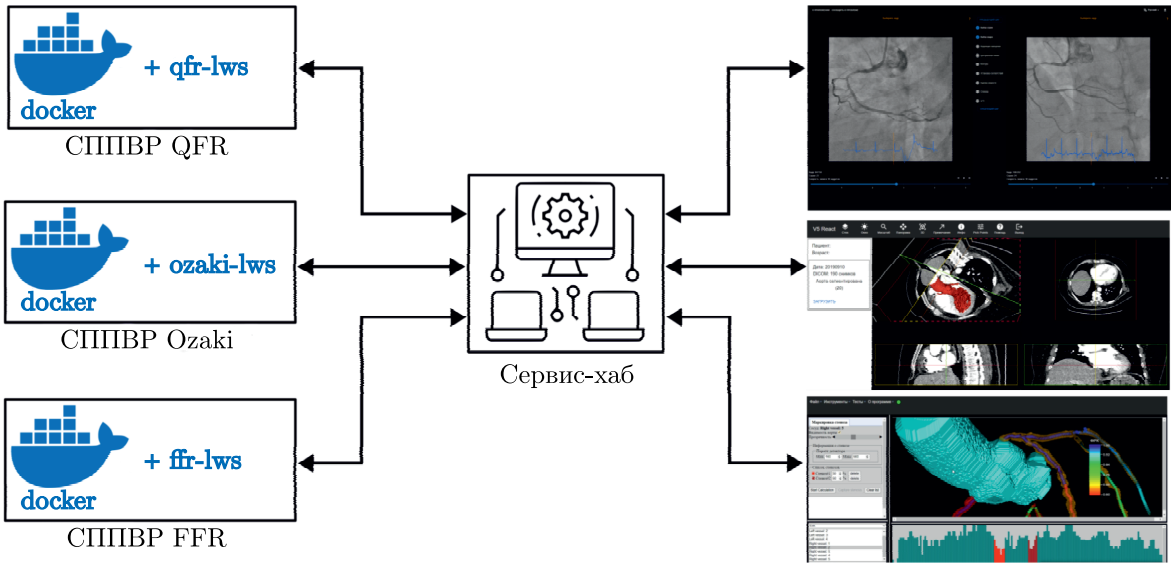

Рис. 1. Схема взаимодействия сервис-контейнеров СППВР (FFR, QFR, Ozaki) с сервис-хабом и примеры клиентских интерфейсов

для веб-приложения в браузере и бэкенда на C++. Спецификация формата MessagePack описывает низкие накладные расходы: например, срез DICOM размером 512×512 `short` (524 288 байт) передается как массив 524 293 байт — практически без накладных метаданных.

API-сервер

Компонент сервис-контейнера, показанный на рис. 1 как qfr-lws, является мини-веб-сервером, обслуживающим статический HTML-контент. Он также предоставляет конечные точки (endpoints) API для веб-приложения и реализован с помощью библиотеки LWS. Библиотека LWS берет на себя периодический опрос (polling) сетевых соединений, парсинг HTTP-заголовков и преобразование HTTP-соединения в WebSocket-соединение, то есть обработку HTTP-заголовка «Upgrade: WebSocket».

Для обработки входящих веб-сокет-соединений сервер должен определить обработчик, который будет вызываться в цикле обработки событий библиотеки LWS при обработке входящих запросов. В силу того что цикл обработки событий LWS является однопотоковым, вызов обработчика протокола не должен занимать слишком большое время, иначе это вызовет уменьшение времени реакции при обработке цикла событий. Между тем некоторые вызовы API требуют значительных вычислений, например, вычисление центральной линии сосуда занимает около минуты, а вычисление характеристик кровотока в сосуде может занимать несколько минут при решении уравнений динамики вязкой жидкости. Такие вычисления выполняются в сервис-контейнере отдельными процессами, запуск которых инициирует обработчик. Для отслеживания завершения запущенных процессов используется механизм опроса (polling), предоставляемый LWS. Каждые 5 секунд сервер опрашивает статус запущенных процессов и возвращает либо код ошибки, либо данные в ответ на API-вызов.

Веб-интерфейс

Веб-ориентированный интерфейс обеспечивает интерактивную визуализацию и анализ медицинских изображений непосредственно в браузере посредством одностраничного приложения

(SPA) с использованием React, JavaScript-библиотеки для построения пользовательских интерфейсов компонентным и декларативным способом.

Для 3D-рендеринга медицинских изображений используется библиотека Three.js (популярный инструментарий WebGL для 3D-графики в браузере) в сочетании с AMI.js (Medical Imaging Toolkit) с открытым исходным кодом для работы с DICOM-изображениями. Важным техническим решением является стратегия локального управления данными в браузере и организации сложных вычислений на стороне сервера.

Далее описаны ключевые технические решения и модули веб-интерфейса.

Визуализация медицинских изображений с помощью AMI.js и Three.js

Для отображения и манипуляции медицинскими изображениями (например, ангиограммами и объемными данными) в браузере используется AMI Medical Imaging Toolkit, JavaScript-библиотека, специально ориентированная на визуализацию двухмерных и трехмерных медицинских изображений, построенная поверх Three.js для рендеринга через WebGL. Она предоставляет высокоуровневые абстракции для распространенных задач медицинской визуализации (отображение срезов в разных проекциях, загрузчик различных форматов). Библиотека обеспечивает парсинг DICOM-файлов, добавление снимков на 3D-сцену и предоставление высокоуровневых абстракций для работы с геометрией снимков. Интеграция снимков непосредственно в трехмерный графический программный модуль нужна для того, чтобы инструменты с изначально трехмерной природой (аннотации, измерения, интерактивные векторы, центральные линии) могли работать в едином мировом пространстве со снимками и корректно между собой взаимодействовать. Это позволяет в прикладных задачах не только накладывать такие элементы на данные, но и интерактивно перемещать/фиксировать их, выполнять геометрические запросы (точки пересечения, расстояния, углы, сечения), а также получать обратную связь от рендеринга в реальном времени.

Основное препятствие использованию AMI Medical Imaging Toolkit заключается в том, что эта библиотека больше не поддерживается. Последний доступный релиз датируется примерно 2018 годом и может взаимодействовать со старыми версиями API Three.js. На практике удалось совместить версию AMI 0.33 лишь с Three.js версии r99, тогда как актуальной является версия Three.js 0.180. Адаптация AMI под современные версии Three.js требует глубоких изменений ядра библиотеки. Решение об использовании Three.js r99 совместно с AMI v0.33.0 с отказом от новых возможностей Three.js оказалось самым практичным вариантом для реализации рабочего модуля 3D-визуализации DICOM в ограниченные сроки.

Интеграция модуля 3D-визуализации в приложение React

Интеграция модуля 3D-визуализации, основанного на Three.js/AMI, в приложение на React столкнулась с рядом технических трудностей. Дело в том, что React управляет интерфейсом с помощью виртуальной объектной модели документа (Document Object Model, DOM) и жизненного цикла компонентов, в то время как Three.js напрямую наносит изображение на холст (HTML-элемент canvas) и имеет собственный цикл рендеринга и состояние (камера, сцена, контроллеры). Согласование работы жизненного цикла React (монтирование, обновление, размонтирование компонентов) и императивного цикла рендеринга Three.js — серьезный вызов. Попытка использования React Three Fiber [R3F], рендера React для Three.js с декларативным описанием 3D-сцен в виде React-компонентов оказалась неудачной: React Three Fiber поддерживает только современные версии Three.js и не работает со старой версией r99, а кроме того, AMI.js написан в монолитном стиле, предполагающем прямое управление структурами Three.js, что также усложняет интеграцию.

Решением стала собственная интеграция на базе React Portals: созданы React Context и система порталов, позволяющая встраивать 3D-сцену Three.js в дерево компонентов React в нужных местах, сохраняя при этом независимость рендеринга от управления React. Например, компоненты `Canvas2D` и `Canvas3D` инициализируют рендер Three.js для двумерных изображений и 3D-модели сосуда соответственно, а затем вставляют сцены в указанное место в JSX-разметке React. Таким образом, 3D-сцена встроена в архитектуру React-приложения как независимый компонент: ее можно разместить в любой части DOM, при этом логика рендеринга остается изолированной.

Для синхронизации состояния используется MobX [MobX], библиотека управления состоянием, основанная на принципах реактивного программирования. MobX-хранилища содержат общие данные приложения (например, загруженные DICOM-файлы, аннотации пользователя, выбранные инструменты), а React-компоненты и логика Three.js/AMI наблюдают за изменениями в этих хранилищах. При изменении состояния (например, выборе нового кадра или параметров визуализации) MobX автоматически обновляет сцену Three.js, обеспечивая согласованность интерфейса и визуализации.

Комбинация React Portals и MobX позволяет плавно встроить рендеринг 3D-графики в реактивный интерфейс. Данный подход может быть полезен не только для интеграции Three.js и AMI, но и в более широком контексте — при встраивании любых внешних графических систем в реактивную архитектуру веб-приложений. Помимо этого, внедрение не ограничивается библиотекой React: механизм порталов также присутствует и в других реактивных библиотеках.

Несмотря на использование устаревающих версий Three.js (r99) и AMI.js, долгосрочная поддержка веб-интерфейса будет осуществляться благодаря двум техническим решениям. Во-первых, архитектура текущего решения изолирует использование Three.js и AMI.js, разделяет логику 3D-визуализации и UI, а синхронизация между ними осуществляется через слой состояния на базе MobX, что позволяет либо продолжать эксплуатацию и развитие приложения на базе Three.js r99 либо при необходимости единовременно обновить модули, завязанные на Three.js с минимальными изменениями остального кода. Во-вторых, ведется разработка собственной библиотеки для работы с DICOM-данными на базе актуальных версий инструментов визуализации. Эта библиотека призвана со временем заменить AMI.js и стать целевой платформой для миграции существующих проектов; при таком переходе основное изменение затрагивает подсистему загрузки медицинских снимков, тогда как общая концепция работы с данными и пользовательские сценарии сохраняются. Миграция проектов будет включать замену DICOM-загрузчика и связанной с ним логики на новую библиотеку, а также обновление версии Three.js с внесением изменений во всех модулях, использующих его API до достижения стабильной работы проекта на обновленной версии. Предлагаемый подход одновременно оставляет возможность поддерживать и развивать текущую реализацию и формирует понятный путь перехода к более современным версиям без долгосрочной привязки к AMI.js и устаревшей версии Three.js.

Локальное хранение данных в IndexedDB

Стратегия развертывания и управления данными может базироваться на серверной системе PACS (Picture Archiving and Communication System) или другом специализированном сервисе, куда загружаются DICOM-изображения для последующего анализа. Однако поддержание серверной базы изображений влечет за собой дополнительные сложности, такие как поддержка учетных записей пользователей, синхронизации данных, безопасности и соответствия нормативным требованиям. Альтернативной стратегией является хранение данных для медицинских изображений и результатов анализа на стороне клиента. Целевой сценарий предполагает использование СППВР одним врачом на его собственном компьютере, поэтому естественно хранить данные визуализации локально, в веб-браузере пользователя через IndexedDB. API IndexedDB

позволяет браузерам хранить значительные объемы структурированных данных на стороне клиента. Клиент может загружать DICOM-файлы прямо со своего компьютера в приложение, и эти файлы вместе с необходимыми производными данными сохраняются в локальном хранилище IndexedDB браузера. Такой подход позволяет врачу, после первоначальной загрузки данных, повторно заходить в приложение даже без доступа к интернету и сразу получать доступ к ранее сохраненным исследованиям на этом устройстве без повторной загрузки файлов.

Использование Dexie.js, обертки над IndexedDB с удобным Promise-ориентированным API, существенно упрощает работу с IndexedDB: определение хранилищ объектов для разных типов данных и выполнение запросов и обновлений требуют гораздо меньшего объема шаблонного кода, чем при использовании нативных вызовов IndexedDB. Dexie также автоматически обрабатывает особенности работы разных браузеров и оптимизирует производительность, что ускорило разработку слоя хранения данных. Недостатком использования Dexie является привязка данных к конкретному браузеру и устройству (если их не экспортировать), но в целевом сценарии врач работает на одном рабочем месте, поэтому это ограничение оказалось приемлемым.

Стратегия локального хранения данных обладает важными преимуществами для СППВР: 1) данные остаются под контролем пользователя (что соответствует требованиям защиты персональных данных); 2) потребность в серверной инфраструктуре исчезает. При необходимости серверное или облачное хранилище можно добавить поверх текущей архитектуры (например, с периодической фоновой загрузкой новых исследований).

Аутентификация пользователя в системе осуществляется с использованием механизма логина и пароля. Конфиденциальность обработки данных обеспечивается полной анонимностью передаваемых в контейнер данных, в которых отсутствуют любые персональные метаданные о пациенте. Шифрование при передаче реализуется для всех поддерживаемых протоколов (HTTP/1, HTTP/2, WebSockets) через стандартные TLS/SSL-механизмы, что обеспечивает конфиденциальность и целостность передаваемых данных.

Настройка параметров безопасности, сертификатов и протоколов обмена, включая валидацию TLS-сертификатов, реализована через JSON-policy-файл. Для обеспечения аудита безопасности система поддерживает для каждой сессии лог-файлы, которые хранятся на сервере в процессе работы с пациентом и архивируются в долговременном хранилище через месяц после завершения работы.

Сервис-хаб

Сервис-хаб управляет запуском сервис-контейнеров при обращении браузера по URL СППВР. Например, если пользователь обращается в браузере по адресу http://zeroapp.ru/app/FFR, то запускается контейнер с приложением FFR, рассчитывающим фракционный резерв кровотока, а при обращении по адресу http://zeroapp.ru/app/Ozaki будет запущен сервис-контейнер СППВР Озаки. Сервис-хаб является обратным прокси-сервером, выполняющим при необходимости запуск нужного контейнера по URL и перенаправляющим запросы в уже запущенные контейнеры.

Пока веб-приложение использует лишь статический HTML-контент, запрос браузера может быть перенаправлен в любой контейнер соответствующего типа, причем один контейнер СППВР может обслуживать запросы многих пользователей, то есть является разделяемым. Как только одним из пользователей в контейнер переданы данные для обработки, он переходит в монопольное владение того пользователя, который послал данные. С этого момента все последующие запросы пользователя, получившего контейнер в монопольное пользование, направляются только в этот контейнер.

Для перенаправления запросов пользователя в правильные контейнеры используются session ID (SID), которые создаются при самом первом обращении пользователя по адресу СППВР. Новая сессия первоначально приписывается к разделяемому контейнеру по умолчанию. Такой контейнер создается при запуске сервис-хаба для каждой СППВР, зарегистрированной в сервис-хабе. Каждый URL ассоциирован с конкретным образом контейнера на докер-хабе; например, URL http://zeroap.ru/app/FFR ассоциирован с контейнером inmpea/ffr:1.3.1-076.5-dev.

На старте сервис-хаб запускает по одному экземпляру контейнера для каждого зарегистрированного URL, этот экземпляр для данного URL будет являться разделяемым контейнером, в который перенаправятся все запросы, не имеющие SID. Таким образом, разделяемый контейнер по умолчанию фактически обслуживает статический HTML-контент конкретной СППВР.

Разные СППВР реализуют разную политику при работе с API. Например, СППВР FFR предполагает, что API-вызовы происходят в рамках одного соединения, в то время как СППВР QFR и СППВР Озаки реализуют политику «одно соединение — один API-вызов». Первый способ характерен для приложений, где обработка данных происходит последовательно, а второй допускает несколько параллельных API-вызовов, например, в СППВР QFR для вычисления центральных линий сосудов на левой и правой ангиограммах.

Сервис-хаб производит захват контейнера в монопольное пользование при самом первом API-вызове. Логика при этом простая: уже при первом вызове в контейнер могут быть переданы данные, которые не позволяют его дальнейшее использование как разделяемого контейнера, и он должен быть закреплен за определенной сессией. Сессия, монопольно владеющая контейнером, будет продолжать работать только с этим контейнером, то есть все последующие API-вызовы пойдут в этот контейнер, сохраняя тем самым контекст обработки данных. При этом возможно, что другие сессии продолжают использовать тот же самый контейнер, который только что был монопольно захвачен, но только до тех пор, пока эти сессии продолжают использовать контейнер только для получения статического HTML-контента. Как только одна из таких сессий попытается сделать API-вызов, она тут же будет перекинута на свободный контейнер, который тут же будет монопольно захвачен. Эта техника напоминает механизм copy-on-write, используемый в операционных системах.

В случае сервис-контейнеров использование этого механизма подразумевает идемпотентность HTML-контента однотипных сервис-контейнеров. Иными словами, статический HTML-контент не должен зависеть от предыстории использования контейнера. Вообще говоря, это условие может и не выполняться, но на практике такие случаи не встречались.

Техника переключения «статических» сессий, то есть использующих только статический HTML-контент, на свободный контейнер в случае монопольного захвата текущего контейнера потребовала ввести пул резервных свободных контейнеров. Запуск сервис-контейнера является потенциально длинной операций, хотя обычно это занимает десятые доли секунды. Поэтому отсутствие под рукой свободного контейнера может привести к задержке ответа на HTTP-запрос конкретного пользователя.

Сборка сервис-контейнеров

На практике сборка сервис-контейнеров СППВР происходит в едином монорепозитории, в котором собраны подкаталоги отдельных СППВР. Но поскольку все сборки однотипны, то в качестве примера ниже взята конкретная СППВР QFR.

На рис. 2 показана структура репозитория кода типового СППВР на примере приложения QFR, которое оценивает степень сужения коронарных сосудов на снимках бипланарной контрастной ангиографии. При подключении новой СППВР к монорепозиторию возникает две задачи: выбор базового докер-образа для сервис-контейнера и создание сборочных контейнеров для модулей и бэкенда.

/QFR/

- api/
- src/
- tests/
- modules
  - qfr-frontend
  - qfr-sandbox
- Makefile

Рис. 2. Структура репозитория кода типового СППВР

В создании СППВР принимают участие, как правило, несколько групп разработчиков, что отражено в модульной структуре репозитория, где модули фронтенда и бэкенда (qfr-frontend и qfr-sandbox) разрабатывались в отдельных репозиториях. Они являются подмодулями git-репозитория QFR. Использование подмодулей позволяет вести независимую разработку разных частей СППВР, обеспечивая модульность и разделение кода.

Выбор базового контейнера определяется особенностями обработки данных; например, в СППВР QFR активно используется python, и поэтому базовым образом для сервис-контейнера стал образ python:3.11. В СППВР Озаки и СППВР FFR базовым образом служит debian:bookworm-slim. Типовая структура сервис-контейнера показана на рис. 3.

/APP/bin

- qfr-lws (мини-веб-сервер и API-сервер)
- *.py (скрипты для вычисления контуров сосудов и пр.)
- solver1/
- solver2/
- solver3/
- mount-origin/
  - index.html
  - assets

Рис. 3. Структура типового сервис-контейнера СППВР

В качестве базового контейнера может служить даже образ docker scratch размером всего 77 байт, что позволяет создавать суперлегковесные контейнеры с минимальным развертыванием. Примером может служить сервис-контейнер, сегментирующий аорту в ранних реализациях СППВР FFR, построенный на базе docker scratch и имеющий размер 30 Мб.

Компоновка сервис-контейнеров СППВР близка по идеологии к подходу Distroless [Distroless], в котором компонуются максимально легковесные образы, содержащие только само приложение и его зависимости времени исполнения, но не содержащие пакетных менеджеров, командных оболочек и прочих утилит юникса. Для обеспечения легковесности обычно выбираются slim-образы с типичным размером 70–80 Мб.

Вторая стадия сборки — компиляция сервера — также придерживается принципа минимизации внешних зависимостей и, как правило, производит статически собранный исполняемый

файл без внешних зависимостей. Технически компиляция сервера происходит в сборочном контейнере, в котором установлены все необходимые пакеты и библиотеки и в который монтируется для компиляции дерево исходных кодов.

Содержимое сборочного контейнера целиком диктуется тем бэкендом, который будет работать в сервис-контейнере СППВР и выполнять фактические вычисления. Сначала выбирается подходящий образ докера, обычно ubuntu-slim или debian-slim, но могут быть и python-образы. В нем разворачиваются полноценная среда компиляции gcc, g++ или Fortran и все необходимые для приложения библиотеки. Это иногда приводит к значительным размерам сборочного контейнера: например, для одного из модулей СППВР Озаки сборочный контейнер занимал 5 Гб. Создание сборочного контейнера для нового СППВР — творческая, хотя и единоразовая задача, выходным артефактом которой является Docker-файл сборочного контейнера. Но как только Docker-файл создан, процесс сборки сервис-контейнера СППВР может быть автоматически воспроизведен на любой машине или в цепочке CI/CD.

Развертывание СППВР на сервис-хабе

Ниже будет описан процесс развертывания СППВР на сервис-хабе, работающем на гипотетическом сайте zeroapp.ru. В качестве примера взято упомянутое выше приложение QFR. Сборка происходит на специально выделенной build-машине, на которой установлены необходимые сборочные контейнеры для подмодулей и для компиляции исполняемых программ бэкенда. Если их нет, то они автоматически создаются из Docker-файлов, созданных при начальном подключении СППВР к монорепозиторию сборки. Build-машина имеет также прямой ssh-root-доступ к zeroapp.ru.

Развертывание состоит из четырех этапов и происходит в полуавтоматическом режиме.

Определение новой версии сервис-контейнера СППВР

В Makefile устанавливается версия сервис-контейнера, например inmpea/qfr:1.7.0-010.1-dev, и подтягиваются нужные изменения (commits) модулей. Какие именно изменения войдут в сборку, определяется целью сборки. Это может быть новая версия UI или бэкенда или API. Периодичность выпусков сервис-контейнеров 1–2 раза в месяц определяется накопленными функциональными изменениями. Никакой автоматической сборки при каждом изменении не предполагается: автоматизированный процесс сборки инициируется вручную.

В названии версии «inmpea/qfr:1.7.0-010.1-dev» inmpea означает логин на докер-хаб, qfr — имя СППВР, 1.7.0 — версия API, 010.1 — номер сборки, суффикс dev означает, что контейнер не минимизирован по размеру. Номер сборки, как правило, меняется при изменении major- или minor-версии API, то есть сборки 010 — это серии выпусков, соответствующих API 1.7. Такое правило является неформальным, соответствие версии API и номера сборки иногда носит чисто мнемонический характер.

Сборка контейнера

После того как определена версия нового сервис-контейнера, необходимо создать и скомпоновать в него артефакты сборки.

Для СППВР QFR это мини-сервер qfr-lws, скрипты из модуля qfr-sandbox, минимизированное веб-приложение и решатели solver1, solver2 и solver3. Решатели предоставляются сборщику извне как данность. Они созданы другими разработчиками, имеют командный интерфейс и поставляются вместе с набором конфигурационных и настроечных файлов. Это особенность СППВР QFR. Другие СППВР собираются каждый раз из исходных текстов. Все решатели не имеют внешних зависимостей и совместимы с версией glibc сервис-контейнера, в котором они выполняются.

Веб-приложение QFR является React-приложением и собирается стандартной командой «npm run build» в подмодуле qfr-frontend, артефакт сборки — справочник dist — копируется в контейнер в mount-origin, который выполняет функцию DOCUMENT ROOT для мини-веб-сервера qfr-lws. Сам мини-сервер qfr-lws компилируется в своем сборочном контейнере, давая на выходе статически собранную исполняемую программу. Таким образом, на втором этапе формируется образ контейнера «inmpea/qfr:1.7.0-010.1-dev».

Автоматизированные тесты

Новый сервис-контейнер сначала проходит ручное тестирование UI и полуавтоматическое тестирование функциональности. Автоматические тесты запускают расчеты, проведенные на предыдущих версиях СППВР, и сравнивают результаты с полученными в новой версии контейнера.

Подключение к сервис-хабу и деплой

После прохождения локальных тестов новый сервис-контейнер загружается на hub.docker.com, включается в таблицу маршрутизации сервис-хаба, и запускается автоматическое развертывание на zeroapp.ru.

Оценка масштабируемости системы

Для оценки масштабируемости и реактивности системы были произведены эксперименты по измерению времени холодного старта и реакции на пиковую нагрузку 100 пользователей. Для эксперимента были рассмотрены СППВР QFR с размером образа 1,46 Гб и СППВР FFR с размером образа 115 Мб. Измерения проводились на 4-ядерном процессоре AMD Ryzen 5 3550H с оперативной памятью 16 Гб. Время холодного старта обеих СППВР, то есть от момента запуска сервера до готовности СППВР обработать первый запрос, составило около 0,3 секунды. Развертывание дополнительных 100 экземпляров СППВР для горячего резерва заняло 3,0505 и 3,0207 секунды соответственно. Это подтверждает очень высокую горизонтальную масштабируемость предложенной системы на основе докер-контейнеров. Интересно отметить, что время запуска 100 дополнительных контейнеров не зависит от размера запускаемого образа. Это связано с тем, что при запуске контейнер не копирует файловую систему образа, из которого создан, а создает дополнительный слой с пустой файловой системой поверх образа. Этим объясняются высокая скорость запуска и отсутствие дополнительных расходов на оперативную память при многократном запуске однотипных контейнеров. Эксперименты по масштабируемости системы больше 100 одновременных пользователей не проводились, так как в типичном сценарии использования вряд ли можно предположить более 10 одновременно работающих клиницистов.

Реактивность СППВР, то есть время отклика на типичные задачи, решаемые в рамках СППВР, сильно зависит от характера обработки данных и индивидуально для каждой системы поддержки принятия решений. Например, в СППВР FFR выдача результата производится через 3–5 минут, а в СППВР QFR — менее чем за минуту.

Заключение

Основная цель представленной технологии — ужать цикл разработки браузерного приложения СППВР до 6–12 месяцев от замысла до завершения. Практика разработки трех приложений СППВР показала, что достижение такой цели возможно при условии хорошо организованной технологической цепочки, а также при использовании стандартных фреймворков и библиотек для UI-компонентов и 3D-рендеринга. Продемонстрирована возможность эффективной разработки zero-footprint-приложений СППВР, основанных на сервис-контейнерах с использованием

Docker и веб-интерфейсе, работающем непосредственно в браузере без установки специализированного программного обеспечения на рабочую станцию врача.

Список литературы (References)

AboArab M. A., Potsika V., Theodorou A., Vagena S., Gravanis M., Sigala F., Fotiadis D. I. Advancing progressive web applications to leverage medical imaging for visualization of digital imaging and communications in medicine and multiplanar reconstruction: software development and validation study // JMIR Medical Informatics. — 2024. — Vol. 12. — e63834. — DOI: 10.2196/63834

AMI.js — medical imaging toolkit for the web. — [Electronic resource]. — https://github.com/FNNDSC/ami (accessed: 09.10.2025).

Babel — JavaScript compiler. — [Electronic resource]. — https://babeljs.io/ (accessed: 09.10.2025).

Babylon.js — 3D engine for the web. — [Electronic resource]. — https://www.babylonjs.com/ (accessed: 09.10.2025).

Badun E., Tessier F., Townson R., Mainegra-Hing E., Storey M.-A., Bazalova-Carter M. VICTORIA — voxel interactive contour tool for online radiation intensity analytics. — [Electronic resource]. — https://web.uvic.ca/~bazalova/dose_viewer/ (accessed: 09.10.2025).

Google closure library. — [Electronic resource]. — https://github.com/google/closure-library (accessed: 09.10.2025).

Cornerstone.js — medical imaging library. — [Electronic resource]. — https://www.cornerstonejs.org/ (accessed: 09.10.2025).

Danilov A., Rudnev S., Vassilevski Y. Numerical basics of bioimpedance measurements // Simini F., Bertemes-Filho P. (eds.) Bioimpedance in biomedical applications and research. — Springer International Publishing, 2018. — P. 117–135. — DOI: 10.1007/978-3-319-74388-2_8

Danilov A. A., Kopytov G. V., Vassilevski Y. V. Online simulator for bioimpedance measurements. — [Electronic resource]. — http://dodo.inm.ras.ru/bia/ (accessed: 10.11.2017).

Google distroless containers. — [Electronic resource]. — https://github.com/GoogleContainerTools/distroless (accessed: 09.10.2025).

DWV — DICOM web viewer. — [Electronic resource]. — https://ivmartel.github.io/dwv/ (accessed: 09.10.2025).

Gorman C., Punzo D., Octaviano I., Pieper S., Longabaugh W. J. R., Clunie D. A., Kikinis R., Fedorov A. Y., Herrmann M. D. Interoperable slide microscopy viewer and annotation tool for imaging data science and computational pathology // Nature Communications. — 2023. — Vol. 14. — Article 1572. — DOI: 10.1038/s41467-023-37224-2

Hostetter J., Khanna N., Mandell J. C. Integration of a zero-footprint cloud-based picture archiving and communication system with customizable forms for radiology research and education // Academic Radiology. — 2018. — Vol. 25, No. 6. — P. 811–818. — DOI: 10.1016/j.acra.2018.01.031

Libwebsockets — lightweight C web server library. — [Electronic resource]. — https://libwebsockets.org/ (accessed: 09.10.2025).

Med3Web project. — [Electronic resource]. — https://github.com/quantori/med3web (accessed: 09.10.2025).

MessagePack — home. — [Electronic resource]. — https://msgpack.org/ (accessed: 09.10.2025).

MobX — documentation. — [Electronic resource]. — https://mobx.js.org/ (accessed: 09.10.2025).

OHIF viewer — open health imaging foundation. — [Electronic resource]. — https://ohif.org/ (accessed: 09.10.2025).

Pereira H., Romero L., Faria P. Web-based DICOM viewers: a survey and a performance classification // Journal of Imaging Informatics in Medicine. — 2025. — Vol. 38. — P. 1304–1322. — DOI: 10.1007/s10278-024-01216-5

React three fiber — documentation. — [Electronic resource]. — https://r3f.docs.pmnd.rs/ (accessed: 09.10.2025).

Three.js — JavaScript 3D library. — [Electronic resource]. — https://threejs.org/ (accessed: 09.10.2025).

VTK.js — visualization toolkit for the web. — [Electronic resource]. — https://github.com/Kitware/vtk-js (accessed: 09.10.2025).

Webpack — module bundler. — [Electronic resource]. — https://webpack.js.org/ (accessed: 09.10.2025).