МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Кемеровский государственный университет»
Институт фундаментальных наук
Кафедра ЮНЕСКО по информационным вычислительным технологиям

Степанов Иван Юрьевич

Выпускная квалификационная работа
магистерская диссертация

Наукоемкий проблемно-ориентированный сервис вычислительного портала КемГУ

по направлению подготовки 02.04.03 Математическое обеспечение и администрирование информационных систем
направленность (профиль) подготовки «Разработка программного обеспечения и способов администрирования информационных систем»

Научный руководитель:

доктор техн. наук, профессор кафедры ЮНЕСКО по ИВТ

А.М. Гудов

Кемерово 2020

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 3

ОБЗОР ЛИТЕРАТУРЫ 5

ГЛАВА 1. ОБЗОР И СРАВНЕНИЕ ИМЕЮЩИХСЯ РЕШЕНИЙ ДЛЯ СОЗДАНИЯ ВЫЧИСЛИТЕЛЬНЫХ ПОРТАЛОВ 6

1.1. СИСТЕМА УПРАВЛЕНИЯ КОНТЕЙНЕРАМИ KUBERNETES 6
1.2. НАБОР СВОБОДНО РАСПРОСТРАНЯЕМЫХ ПРИЛОЖЕНИЙ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАЗРАБОТКИ ПО 13
1.3. СРАВНЕНИЕ СРЕДСТВ РАЗРАБОТКИ ВЫЧИСЛИТЕЛЬНЫХ ПОРТАЛОВ 15

ГЛАВА 2. ДЕКОМПОЗИЦИЯ ПРОЦЕССА РАЗРАБОТКИ ТЕСТОВЫХ ВЫЧИСЛИТЕЛЬНОГО ПОРТАЛА И СЕРВИСА И СОПУТСТВУЮЩИХ МЕТОДОВ 18

2.1. ОПИСАНИЕ ОБЪЕКТА, ИСПОЛЬЗУЕМОГО В РАБОТЕ 18
2.2. ОПИСАНИЕ АРХИТЕКТУРЫ ТЕСТОВОГО ПОРТАЛА 20
2.3. ОПИСАНИЕ МЕХАНИЗМА ВЗАИМОДЕЙСТВИЯ КОНТЕЙНЕРОВ ПОРТАЛА И ПРОБЛЕМНО-ОРИЕНТИРОВАННОГО СЕРВИСА 23

ГЛАВА 3. РЕАЛИЗАЦИЯ 26

3.1. ОПИСАНИЕ СРЕДСТВ РЕАЛИЗАЦИИ СЕРВИСОВ 26
3.2. СЕРВИС «ВЫЧИСЛИТЕЛЬНЫЙ ПОРТАЛ» 26
3.3. СЕРВИС «ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЙ НАУКОЁМКИЙ СЕРВИС» 28
3.4. РАЗВОРАЧИВАНИЕ ЛОКАЛЬНОГО КЛАСТЕРА 29
3.5. КОНТЕЙНЕРИЗАЦИЯ РАЗРАБОТАННЫХ СЕРВИСОВ И РАЗВОРАЧИВАНИЕ ИХ НА ЛОКАЛЬНОМ КЛАСТЕРЕ 31

ЗАКЛЮЧЕНИЕ 37

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 38

ВВЕДЕНИЕ

В настоящее время все сильнее прослеживается тенденция к переходу от классической модели распространения ПО к облачным технологиям. Облачные сервисы позволили уравнять в ИТ-возможностях и малый бизнес, и крупные компании. Сегодня облачные сервисы и приложения — идеальный вариант для большинства начинаний в сфере IT.

Однако, для некоторых задач, использование облачных технологий всё же не нашло повсеместного распространения. Одной из сфер, в которой распространение через облако весьма затруднительно, является вычислительные порталы.

Казалось бы, что именно они и должны были стать самым популярным объектом в сфере облачных технологий, предоставляя пользователям ресурсы для решения различных ресурсоемких задач, но, при более детальном рассмотрении, становится очевидно, что имеется ряд мелких задач, без решения которых становится невозможным реализовать корректный pipeline использования конечным продуктом с целью получения результата, например:

- отсутствие унифицированного интерфейса не позволит воспользоваться инструментами пакетного размещения решателей;
- размещение ресурсов под каждый отдельный экземпляр задачи должен размещаться либо вручную, либо специализированным ПО, способным балансировать нагрузку без вмешательства администратора;
- межпользовательское взаимодействие реализуется лишь на уровне форумов, но не на уровне подготовки моделей, что может сказаться на временных затратах решения ресурсоемкой задачи, и т.д.

Но помимо концептуальных проблем, связанных с такими порталами, возникают и локальные проблемы, которые могут привести к отсутствию положительного результата:

- отсутствие сериализации при передаче данных между модулями;
- сбои внутри модуля (например, утечки памяти), которые приводят к потере контроля над всем кластером.

Целью работы является разработка наукоемкого проблемно-ориентированного сервиса, который может встраиваться на различных вычислительных порталах.

Актуальность исследуемой темы заключается в том, что на момент написания ВКР не существует общедоступных специализированных инструментариев по созданию универсальных сервисов, которые не зависели бы от используемых CMS, систем управления задач и т.д.

Реализация поставленной цели потребует решения следующих задач:

1. провести сравнение имеющихся решений для создания вычислительных порталов;
2. создать тестовую платформу для развертки сервисов;
3. провести обзор доступных технологий, используемых для создания унифицированных самоизолированных сервисов;
4. рассмотреть способы межмодульного взаимодействия;
5. произвести тестирование развернутого модуля

К объектам исследования относятся программное обеспечение для масштабирования контейнеризированных приложений, автоматизации развёртывания и управления Kubernetes, постановка задачи «Математическое моделирование процессов переноса воздушных масс».

Методы исследования определяются сущностью теоретических и практических проблем. Используются: теория и методы разработки программного обеспечения, математическое моделирование, компьютерное моделирование.

ОБЗОР ЛИТЕРАТУРЫ

При написании магистерской диссертации были использованы книги и пособия по программному продукту Kubernetes, научные статьи по выбранной тематике, а также различные электронные порталы, расположенные в сети Интернет, по тематике магистерской диссертации и смежных с ней.

Анализ представленных в открытом доступе источников литературы показал, что по выбранной тематике отсутствует какая-либо литература, она имеется в наличии лишь по смежным темам, например: по проектированию моделей в пакете вычислительной гидродинамики PHOENICS, по способам распознавания объектов на изображениях, по отдельным классификаторам, и т.д.

На основе работ «Введение в технологию контейнеров и Kubernetes» Маркелов, А. А [1] «Kubernetes в действии» Лукша, М [2], подробно изучен механизм развертки контейнеров на любом подходящем абстрактном устройстве, в котором установлена система управления контейнерами Kubernetes.

Такие электронные ресурсы как «Инженерный вычислительный портал для решения экологических задач» [3], «Один из подходов к созданию вычислительных сервисов для инженерного вычислительного портала» [4] и «Механизм создания вычислительных сервисов для инженерного вычислительного портала» [5] позволили освоить основные принципы разработки вычислительных порталов и сервисов для них.

Используя такие работы, как «Теория информационных процессов и систем» Доррер, Г.А [6], «Системный анализ и моделирование информационных процессов и систем» Горлушкиной, Н. Н [7] был проведен подробный разбор способов построения диаграмм информационных процессов.

Работы «Python. Подробный справочник, 4-е издание» Бизли Д. [8], «Программирование на Python, 4-е издание, I и II том» Лутц М [9]. легли в основу реализации вспомогательных модулей, используемых для дальнейшей работы.

ГЛАВА 1. ОБЗОР И СРАВНЕНИЕ ИМЕЮЩИХСЯ РЕШЕНИЙ ДЛЯ СОЗДАНИЯ ВЫЧИСЛИТЕЛЬНЫХ ПОРТАЛОВ

1.1. СИСТЕМА УПРАВЛЕНИЯ КОНТЕЙНЕРАМИ KUBERNETES

Kubernetes (K8s) — это система с открытым исходным кодом для автоматизации развертывания, масштабирования и управления контейнерными приложениями.

Он группирует контейнеры, которые составляют приложение, в логические единицы для удобного управления и обнаружения. Kubernetes опирается на 15-летний опыт управления производственными нагрузками в Google в сочетании с лучшими в своем классе идеями и практиками сообщества. Использование данного программного средства представляется возможным в широком спектре различных сценариев, таких как:

- обнаружение сервисов и балансировка нагрузки - нет необходимости изменять приложение, чтобы использовать незнакомый механизм обнаружения сервисов. Kubernetes предоставляет модулю свои собственные IP-адреса и одно DNS-имя для набора модулей и может распределять нагрузку между ними.
- топология обслуживания - маршрутизация сервисного трафика на основе кластерной топологии.
- оркестрирование хранения – поддерживает различные системы хранения, будь то локальное хранилище, поставщик общедоступного облака, такой как GCP или AWS, или сетевая система хранения, такая как NFS, iSCSI, Gluster, Ceph, Cinder или Flocker.
- самовосстановление - Kubernetes перезапускает контейнеры, которые выходят из строя, заменяет и перепланирует контейнеры, когда узлы умирают, убивает контейнеры, которые не отвечают на вашу пользовательскую проверку работоспособности, и не объявляет их клиентам, пока они не будут готовы к обслуживанию.

- Автоматические развертывания и откаты - Kubernetes постепенно внедряет изменения в ваше приложение или его конфигурацию, одновременно отслеживая работоспособность приложения, чтобы убедиться, что оно не уничтожает все его экземпляры одновременно. Если что-то пойдет не так, Kubernetes отменит изменения.
- Управление ключами и конфигурацией - развертывайте и обновляйте секреты и конфигурацию приложения, не перестраивая образ и не раскрывая секреты в конфигурации стека.
- Автоматическая упаковка бинов - автоматически размещает контейнеры на основе их требований к ресурсам и других ограничений, не жертвуя при этом доступностью. Комбинируйте критические и максимально эффективные рабочие нагрузки, чтобы повысить эффективность использования и сэкономить еще больше ресурсов.
- Пакетное исполнение - Kubernetes может управлять пакетными рабочими нагрузками и рабочими нагрузками CI, заменяя отказавшие контейнеры, если это необходимо.
- Горизонтальное масштабирование – упрощенное управление масштабированием вверх и вниз с помощью команд, пользовательского интерфейса или автоматически в зависимости от загрузки процессора. [10].

Для развертывания Kubernetes, нужно иметь или физический, или логический кластер. Кластер Kubernetes состоит из набора рабочих машин или совокупности виртуальных машин, называемых узлами, которые запускают контейнерные приложения. В каждом кластере есть хотя бы один рабочий узел.

Рабочие узлы содержат узлы, которые являются компонентами рабочей нагрузки приложения. Плоскость управления управляет рабочими узлами и модулями в кластере. В производственных средах плоскость управления обычно выполняется на нескольких компьютерах, а кластер обычно работает на нескольких узлах, обеспечивая отказоустойчивость и высокую доступность.

Диаграмма кластера Kubernetes со всеми связанными компонентами представлена на рисунке 1.

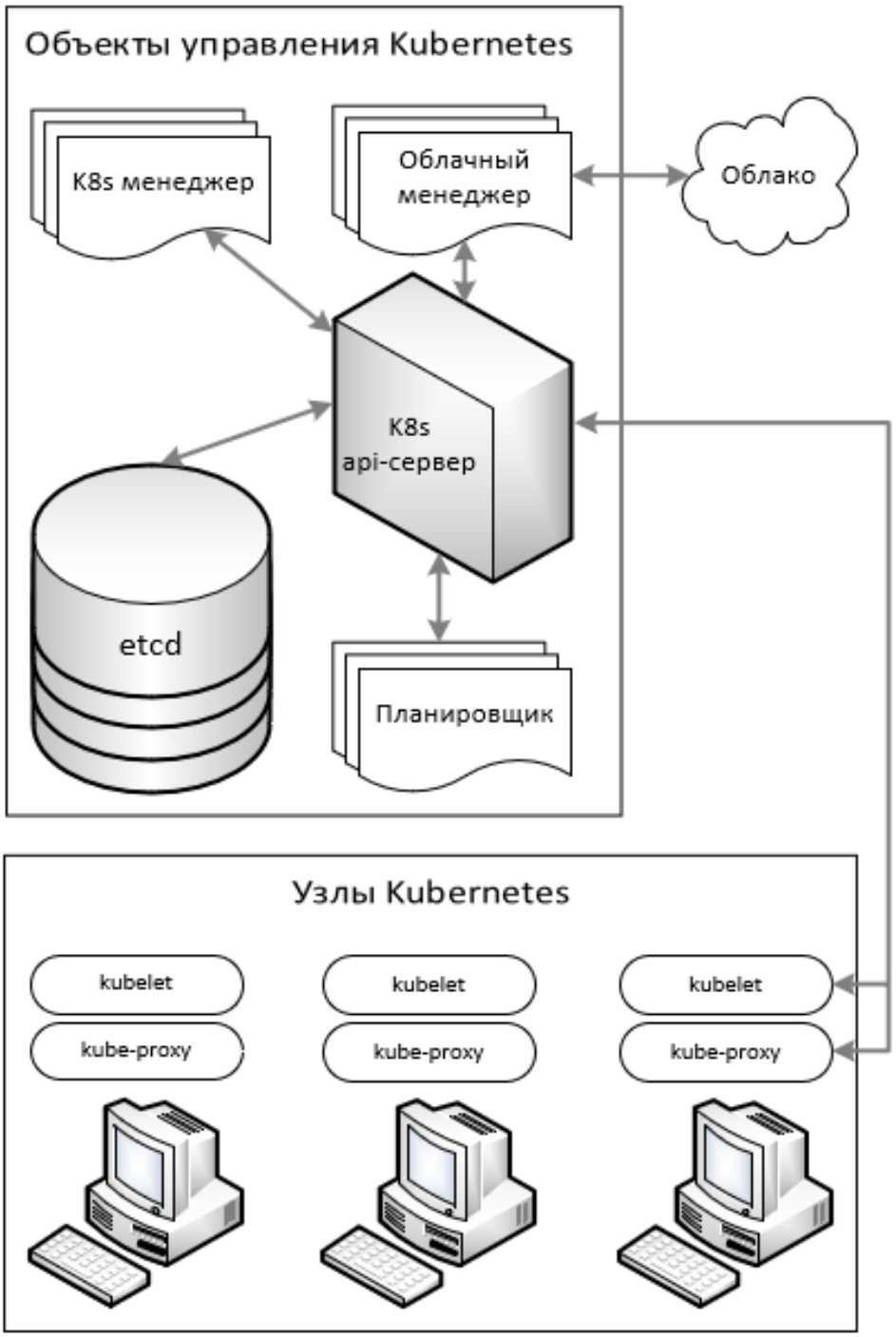

Рис.1. Диаграмма взаимодействия модулей Kubernetes

Для того чтобы понимать, как работает Kubernetes, необходимо немного более детально рассмотреть его компоненты.

1.1.1. Объекты управления Kubernetes

Объекты управления Kubernetes принимают глобальные решения о кластере (например, планирование), а также обнаруживают и реагируют на события кластера (например, запускают новый модуль, когда поле реплик развертывания не удовлетворено имеющимися репликами).

Такие объекты могут быть запущены на любой машине в кластере. Однако для простоты сценарии настройки обычно запускают все объекты управления Kubernetes на одном компьютере и не запускают пользовательские контейнеры на этом компьютере.

1.1.2. K8s api-сервер

Сервер API — это компонент управления Kubernetes, который предоставляет API Kubernetes. Сервер API является внешним интерфейсом для объектов управления Kubernetes.

Основной реализацией сервера API Kubernetes является kube-apiserver. Он предназначен для горизонтального масштабирования, то есть для масштабирования путем развертывания большего количества экземпляров. Можно запустить несколько экземпляров kube-apiserver и распределить трафик между этими экземплярами.

1.1.3. etcd

Согласованное и высокодоступное хранилище пар «ключ - значение», используемое в качестве резервного хранилища Kubernetes для всех данных кластера. Если кластер Kubernetes использует etcd в качестве резервного хранилища, убедитесь, что у вас есть план резервного копирования этих данных.

1.1.4. Планировщик

Компонент управления, который наблюдает за вновь созданными модулями без назначенного узла и выбирает узел для их запуска.

Факторы, принимаемые во внимание при планировании решений, включают в себя: индивидуальные и коллективные требования к ресурсам, аппаратные/программные/политические ограничения, спецификации соответствия и анти-соответствия, местоположение данных, помехи между нагрузками и сроки.

1.1.5. K8s-менеджер

Компонент области управления, который запускает процессы контроллера. Логически каждый контроллер — это отдельный процесс, но для уменьшения сложности все они скомпилированы в один бинарный файл и выполняются в одном процессе.

Эти контроллеры включают в себя:

- Контроллер узла. Отвечает за уведомление и реагирование при выходе из строя узлов.
- Контроллер репликации: отвечает за поддержание правильного количества модулей для каждого объекта контроллера репликации в системе.
- Контроллер конечных точек: заполняет объект конечных точек (то есть присоединяется к службам и модулям).
- Контроллеры учетных записей и токенов службы: создание учетных записей по умолчанию и токенов доступа API для новых пространств имен.

1.1.6. Облачный менеджер

Компонент управления Kubernetes, который внедряет логику управления для конкретного облака. Менеджер облачного контроллера позволяет связать кластер с API облачного провайдера и отделить компоненты, которые взаимодействуют с этой облачной платформой, от компонентов, которые просто взаимодействуют с кластером.

Облачный менеджер запускает только те контроллеры, которые характерны для облачного провайдера. Если Kubernetes используется в собственном помещении или в учебной среде на личном ПК, в кластере может отсутствовать диспетчер облачных контроллеров.

Как и в случае с K8s менеджером, облачный менеджер объединяет несколько логически независимых контуров управления в один двоичный файл, который запускается как один процесс. Поддерживается масштабирование по горизонтали (запуск более одной копии), чтобы повысить производительность или помочь справляться со сбоями

Следующие контроллеры могут иметь зависимости облачного провайдера:

- Контроллер узла: для проверки облака, чтобы определить, был ли удален узел в облаке после того, как он перестал отвечать;
- Контроллер маршрутизации: для настройки маршрутов в базовой облачной инфраструктуре;

- Сервисный контроллер: для создания, обновления и удаления балансировщиков нагрузки облачного провайдера.

1.1.7. Узлы Kubernetes

Компоненты узла работают на каждом узле, поддерживая работающие модули и обеспечивая среду выполнения Kubernetes.[11]

1.1.8. kubelet

Агент, который работает на каждом узле в кластере. Это гарантирует, что контейнеры работают в Pod.

Модуль Pod - это базовая исполнительная единица приложения Kubernetes - наименьшая и самая простая единица в объектной модели Kubernetes, которая создается или разворачивается. Модуль представляет процессы, запущенные в кластере.

Модуль инкапсулирует контейнер приложения (или, в некоторых случаях, несколько контейнеров), ресурсы хранения, уникальный сетевой идентификатор (IP-адрес), а также параметры, определяющие порядок работы одного или нескольких контейнеров. Модуль представляет собой единицу развертывания: один экземпляр приложения в Kubernetes, который может состоять из одного контейнера или небольшого числа контейнеров, которые тесно связаны между собой и совместно используют ресурсы.

Docker - наиболее распространенная среда выполнения контейнеров, используемая в модуле Kubernetes, но модули поддерживают и другие среды выполнения контейнеров.

Kubelet принимает набор PodSpecs, предоставляемых с помощью различных механизмов, и обеспечивает работоспособность и работоспособность контейнеров, описанных в этих PodSpecs. Kubelet не управляет контейнерами, которые не были созданы Kubernetes [12].

1.1.9. kube-proxy

Kube-proxy - это сетевой прокси, который работает на каждом узле в кластере и реализует часть концепции сервиса Kubernetes. Kube-proxy поддерживает сетевые правила на узлах. Эти сетевые правила разрешают сетевое взаимодействие с модулями из сетевых сеансов внутри или вне кластера.

kube-proxy использует уровень фильтрации пакетов операционной системы, если он есть, и он доступен. В противном случае, Kube-прокси пересылает сам трафик.

1.1.10. Среда выполнения контейнера

Среда выполнения контейнера — это программное обеспечение, которое отвечает за запуск контейнеров.

Kubernetes поддерживает различные среды выполнения контейнеров, такие как: Docker, containerd, CRI-O и любую реализацию интерфейса Kubernetes CRI (Container Runtime Interface) [11].

К преимуществам использования Kubernetes, так же можно отнести поддержку различных утилит, которые можно использовать для расширения возможностей:

- Кластерный DNS — это DNS-сервер, в дополнение к другим DNS-серверам в среде, который обслуживает записи DNS для служб Kubernetes. Контейнеры, запущенные Kubernetes, автоматически включают этот DNS-сервер в свои поиски DNS.
- Dashboard — это универсальный веб-интерфейс для кластеров Kubernetes. Это позволяет пользователям управлять приложениями, работающими в кластере, и устранять неполадки, а также сам кластер.
- Чтобы масштабировать приложение и предоставлять надежный сервис, необходимо понимать, как приложение ведет себя при его развертывании. Kubernetes предоставляет подробную информацию об использовании ресурсов приложения на каждом из этих уровней. В Kubernetes мониторинг приложений не зависит от одного решения для мониторинга. В новых кластерах вы можете использовать метрики ресурсов или конвейеры полных метрик для сбора статистики мониторинга.
- Механизм ведения журнала на уровне кластера отвечает за сохранение журналов контейнеров в центральном хранилище журналов с интерфейсом поиска / просмотра. [13]

1.2. НАБОР СВОБОДНО РАСПРОСТРАНЯЕМЫХ ПРИЛОЖЕНИЙ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАЗРАБОТКИ ПО

В качестве средств, которые могут быть использованы для разработки вычислительных порталов, с таким же успехом может выступать набор свободно распространяемого ПО.

Однако, необходимо принципиально разобрать то, как такой портал должен функционировать. Рассмотрим отдельные компоненты такого портала:

- Client System – некоторая внешняя система клиента, позволяющая производить взаимодействие с web-сервисами/бизнес-процессами портала и/или с LDAP-каталогом. Включает в себя совокупность как аппаратных (некоторый стационарный\портативный ПК), так и программных средств (веб-браузер или любое другое ПО, позволяющее в режиме графического интерфейса взаимодействовать с таким порталом).
- Portal Server — компьютер-сервер, на котором развернут портал.
- Прокси-сервер, который передает запросы другим компонентам (контейнеру сервлетов, веб-серверу, менеджеру процессов и т.д.). Исключение составляют только запросы на получение статичных файлов, например, изображений, скриптовых-файлов и т.д., ввиду того, что большинство таких серверов не оптимизированы для такого рода запросов.
- Web-сервер, на котором работают сервисы разработки и запуска сервисов.
- Менеджер процессов FastCGI, применяемый для генерации динамического PHP-содержимого. Используется ввиду явного использования модульной структуры сервера.
- Контейнер сервлетов — это сервер web-приложений, реализующий поддержку спецификаций сервлетов. На нем разворачиваются платформы разворачивания порталов, движки веб-сервисов и т.д.
- Набор специфических портлетов, необходимых для поддержания работоспособности, создания более дружелюбного интерфейса пользователя и расширения функционала разработчика портала:

- портлет, генерирующий web-интерфейс сервисов портала;
- портлет управления доступом к вычислительным ресурсами, учета использованных ресурсов и балансировки запущенных портлетов;
- портлет проверки доступа пользователей к web-сервисам и т.д.

- Некоторый web-сервис для взаимодействия с высокопроизводительными вычислительными ресурсами.
- Web-сервис или локальный сервис с внешним API для взаимодействия с хранилищем файлов пользователей.

Это лишь краткий перечень необходимых компонентов, позволяющих реализовать полноценный портал, на котором можно производить решения вычислительных задач. [5]

Безусловно, использование такого подходя для разработки имеет ряд преимуществ:

- Практически полная модульность позволяет использовать различные компоненты.
- Поскольку модули независимы, для различных задач может использоваться как разные версии, так и различные реализации одной библиотеки. Это уменьшает время, которое проходит между началом и завершением добавлением вычислительного модуля на портал.
- Кроме того, модули могут быть радикально изменен, не затрагивая другие модули, если его первоначальная функция остается неизменной.

Но есть и очевидные недостатки:

- Разные модули могут иметь разные зависимости, что будет увеличивать количество ресурсов, требуемых для корректной работы.
- При конфликте модулей, может стать краеугольным камнем при расширении функционала портала: добавить больше модулей нарушив общую работоспособность всего портала или оставить всё как есть.
- У каждого модуля имеется свой API, для которых приходится разрабатывать соединяющие модули утилиты, что усложняет разработку.

1.3. СРАВНЕНИЕ СРЕДСТВ РАЗРАБОТКИ ВЫЧИСЛИТЕЛЬНЫХ ПОРТАЛОВ

В качестве критериев сравнения средств разработки вычислительных порталов будут выступать:

- наличие официальной документации по используемым средствам;
- обособленность (можно ли обойтись имеющимися модулями или необходимо разрабатывать их самостоятельно);
- горизонтальная масштабируемость;
- поддержка балансировки нагрузки
- унифицированность разработки отдельных сервисов для портала.

Официальная документация для системы автоматизации развертывания, масштабирования и управления контейнерными приложениями Kubernetes представлена на официальном сайте, причем для этой системы есть не только описание внутренних модулей, но также и примеры использование и даже обучение, что позволяет практически каждому человеку опробовать работу в виртуальном окружении, не разворачивая всю систему на локальном кластере.

Что касается документация различных модулей, используемых для самостоятельной разработки, то нельзя сразу однозначно ответить на данный вопрос. Как правило, документация имеется в открытом доступе, но её полнота может варьироваться от исчерпывающей, как, например, у прокси-сервера nginx, так и иметь лишь описательный характер, как, к примеру, у PHP-FPM.

Kubernetes – полностью обособленная система, несмотря на то, что она очень хорошо расширяется дополнительными модулями. Она имеет достаточное количество дополнений, которые можно добавить, но при всём этом, даже чистая, система предоставляет все возможные инструменты.

Систему, реализованную с использованием ансамбля различных модулей трудно назвать обособленной. Безусловно, опытным путем можно подобрать модули, которые будут работать, нативно поддерживая друг друга, но необходимое количество человеко-часов не представляется возможным оценить, т.к. данный процесс невозможно автоматизировать каким-либо способом [].

Масштабируемость по горизонтали и по вертикали поддерживается Kubernetes сразу «из коробки» ввиду того, что она разрабатывалась с идеей на то, что будет работать на кластерах различных конфигураций, что подразумевает отсутствие привязки к каким-либо аппаратным конфигурациям.

Если говорить о разрабатываемой вручную системе, то масштабирование возможно реализовать через конфигурационные файлы, но только вертикальную. Горизонтальная масштабируемость не возможна в полной мере, т.к. настройка отдельных узлов должна будет производиться вручную от установки ОС и до финального подключения в пул узлов.

Т.к. Kubernetes – кластерная система, обязательным требованием для таких систем является наличие балансировщика нагрузки, поэтому, без сомнений, у Kubernetes нет с этим проблем.

Зато задача балансировки нагрузки становится очень актуальной в самописной системе ввиду того, что от архитектуры разрабатываемой системы будет зависеть многое. Но в целом, существуют расширения для отдельных модулей, которые могут заниматься балансировкой нагрузки, но делать это в автоматическом режиме не представляется возможным.

Краеугольным камнем при разработке портала, безусловно, является решение задачи дальнейшего расширения путем встраивание сторонних сервисов, которые позволяет производить решение численных задач.

Говоря о Kubernetes, каждый такой сервис – это обособленное приложение, запакованное в контейнер, которое обязательно должно иметь собственный интерфейс и решать задачу сопоставления зависимостей путем развертки дополнительных контейнеров со вспомогательными приложениями. Если же говорить о вручную разрабатываемом портале, то тут нужно учесть несколько аспектов:

1) сам портал должен иметь какой-либо интерфейс, значит и другие сервисы должны вписываться в общую канву, не стилистически, но ввод данных, необходимых для решения какой-либо задачи с помощью этого сервиса, должен быть реализован на самом портале;

2) у каждого сервиса свои входные и выходные данные, а значит для работы с ними могут потребоваться дополнительные модули, которые скажутся на производительности не только отдельного сервиса, но и всей системы;

3) обновление самого портала требуют и обновления интерфейса (как графического, так и программного) всех сервисов, что становится большой проблемой при их большом количестве.

Таким образом, добавление каждого из модулей – процесс крайне ресурсоёмкий с точки зрения человеко-часов, поэтому ни о какой унифицированности разработки и добавления сервисов в систему речи не может идти, за исключением той ситуации, когда эти сервисы не запаковываются в контейнер, но это возвращает разработчика или к использованию Kubernetes, или к использованию связки “Docker – Django”, что хоть является не полной альтернативой Kubernetes.

Результаты проведенного сравнения приведены в таблице 1.

Таблица 1. Результаты сравнения средств разработки порталов

Критерии	Kubernetes	Самодельная реализация
Официальная документация	+	+-
Обособленность	+	+-
Масштабируемость	+	+-
Балансировка нагрузки	+	+-
Унифицированность разработки отдельных сервисов	+	-

На основании полученных результатов можно сделать вывод о том, что система автоматизации развертывания, масштабирования и управления контейнерными приложениями Kubernetes удовлетворяет всем поставленным на этапе выбора средств разработки вычислительных порталов требованиям и оказался наиболее подходящим. Он обладает рядом особенностей, упрощающих процесс разворачивания как самой системы, так и сервисов в ней.

ГЛАВА 2. ДЕКОМПОЗИЦИЯ ПРОЦЕССА РАЗРАБОТКИ ТЕСТОВЫХ ВЫЧИСЛИТЕЛЬНОГО ПОРТАЛА И СЕРВИСА И СОПУТСТВУЮЩИХ МЕТОДОВ

Прежде чем перейти к описанию объектов, которые используются в работе, необходимо обосновать, почему вместе с сервисом производится ещё и разворачивание тестового вычислительного портала.

Основной проблемой, которая сопровождает на всех этапах, является то, что действовавший до некоторых пор вычислительный портал КемГУ icp.kemsu.ru и его зеркало icp2.kemsu.ru более недоступны для внешнего доступа, что подтверждается доступными снимками с сайта web.archive.org, что демонстрируется на рисунках 2 и 3 соответственно.

Рис. 2. Скриншот с сайта web.archive.org

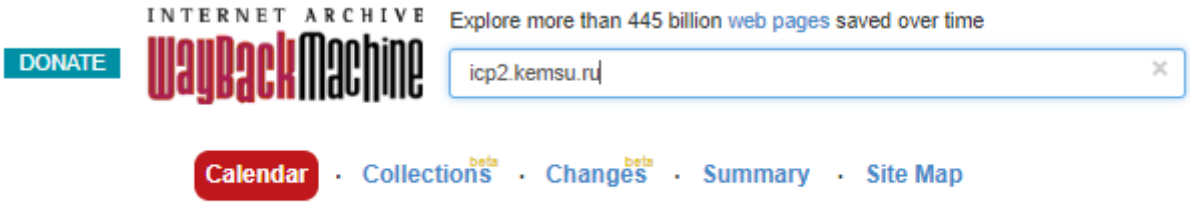

Рис. 3. Скриншот с сайта web.archive.org

Таким образом, ввиду отсутствующего для тестирования портала, возможно сразу выбрать концепцию, которая будет использоваться далее в разработке.

2.1. ОПИСАНИЕ ОБЪЕКТА, ИСПОЛЬЗУЕМОГО В РАБОТЕ

В качестве объектов, которые используются в работе, выступают абстрактные наукоемкие проблемно-ориентированные сервисы, пользование которыми распространяется согласно модели обслуживания AaaS – “ Application as a Service”.

Почему сервис распространяется по такой модели? Нужно обратиться к модели становления облачных вычислений Cloud Computing Maturity Model (CCMM), которая включает в себя пять уровней, представленных на рисунке 4.

Согласно ей выделяют облачные вычисления первого и второго поколений. Технологии первого поколения соответствуют уровням I—III и являются бизнес-моделями IaaS, PaaS, SaaS.

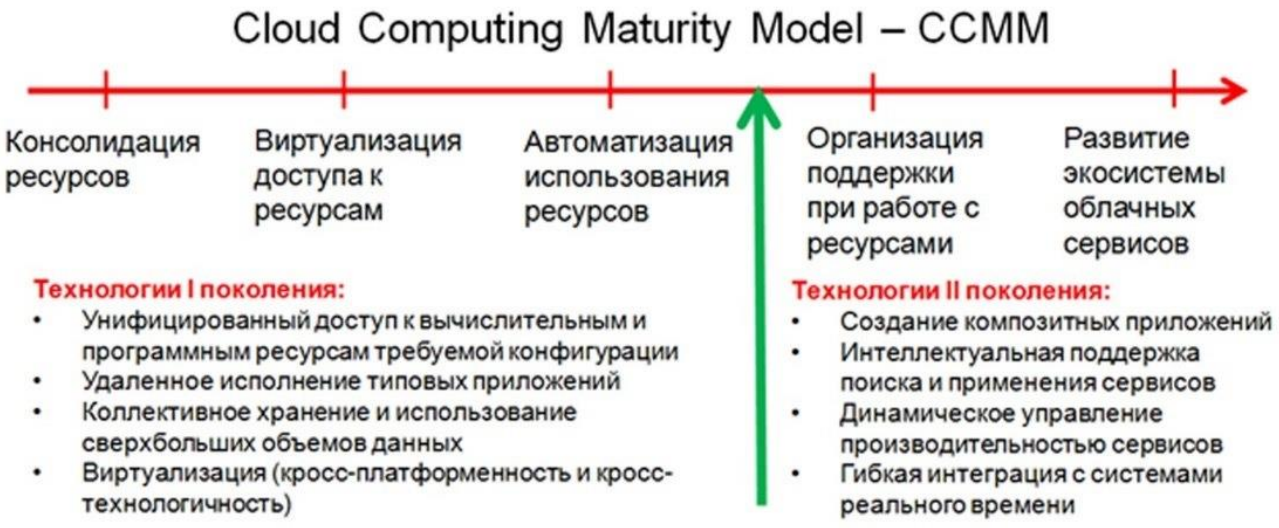

Рис. 4. Модель становления облачных вычислений

Технологии второго поколения включают в себя уровни IV и V и ориентированы не только на предоставление пользователям отдельных сервисов по сети интернет, но и на управление целой экосистемой вычислительных, программных и информационных ресурсов.

Это позволяет решать задачи, связанные с совместной деятельностью множества пользователей. Технологии облачных вычислений второго поколения реализуют перспективную модель облачных вычислений AaaS (Application as a Service), основной услугой в рамках которой оказывается разработка и использование вычислительных порталов (это совокупность взаимодействующих облачных сервисов, направленных на решение одной задачи) [14].

Что же такое «наукоемкий вычислительный сервис»? Это такой сервис, который решает некоторый ограниченный класс задач, который требует серьезных научных разработок: специализированных методов и подходов, направленных на получение некоторого численного решения. Это могут быть

задачи численного моделирования, компьютерного зрения, обработки Big Data и т.д.

На понятийном уровне становится ясно, что сервис – это некоторый абстрактный объект, работа которого для конечного пользователя согласуется с концепцией «черного ящика». Однако, если рассматривать такой сервис в экосистеме вычислительного портала, то сервис – это обособленное приложение, которое имеет:

- свой собственный текстово-графический интерфейс (пользователь может взаимодействовать с ним посредством командной строки или набора графических окон);
- свои собственные средства для организации как ввода входных данных, так и вывода результатов в каком-либо виде, пригодном для дальнейших взаимодействий
- некоторый встроенный ресолвер зависимостей, т.к. Kubernetes (а с его помощью будет развернут вычислительный портал) не может сопоставлять неизвестные ему зависимости автоматически, т.е. настройкой окружения должен заниматься сам сервис. [15]

2.2. ОПИСАНИЕ АРХИТЕКТУРЫ ТЕСТОВОГО ПОРТАЛА

Т.к. ранее было сказано, что в качестве основы будет использован Kubernetes, нужно разобраться, в чем заключается его преимущество.

Во-первых, Kubernetes не использует концепцию монолитных приложений, а использует микросервисы. Монолитные приложения состоят из компонентов, которые тесно связаны друг с другом и должны разрабатываться, развертываться и управляться как одна сущность, поскольку все они выполняются как один процесс ОС.

Для запуска монолитного приложения обычно требуется небольшое количество мощных серверов, которые могут предоставить достаточно ресурсов для запуска приложения. Для того чтобы справиться с растущей нагрузкой на систему, нужно либо масштабировать серверы вертикально, добавляя больше процессоров, оперативной памяти и других серверных компонентов, либо

масштабировать всю систему по горизонтали, настраивая дополнительные серверы и запуская несколько копий (или реплик) приложения. Если какая-либо часть монолитного приложения не масштабируется, то все приложение становится немасштабируемым, если только каким-то образом этот монолит не разделить.

Эти и другие проблемы заставили нас начать разбиение сложных монолитных приложений на небольшие независимые развертывания компонентов, называемых микросервисами. Каждый микросервис выполняется как независимый процесс и взаимодействует с другими микросервисами через простые, четко определенные интерфейсы (API).

Микросервисы взаимодействуют через синхронные протоколы, такие как HTTP, используя которые, они обычно предоставляют RESTful, или через асинхронные протоколы, такие как AMQP.

Компоненты в архитектуре микросервисов не только развертываются независимо, но и разрабатываются таким же образом. Из-за их независимости и того факта, что, как правило, каждый компонент разрабатывается отдельными командами, ничто не мешает каждой команде использовать разные библиотеки и заменять их всякий раз, когда возникает необходимость. Расхождение в библиотеках между компонентами приложения, где в любом случае требуются разные версии одной библиотеки, – это неизбежность, что демонстрируется на рисунке 5.

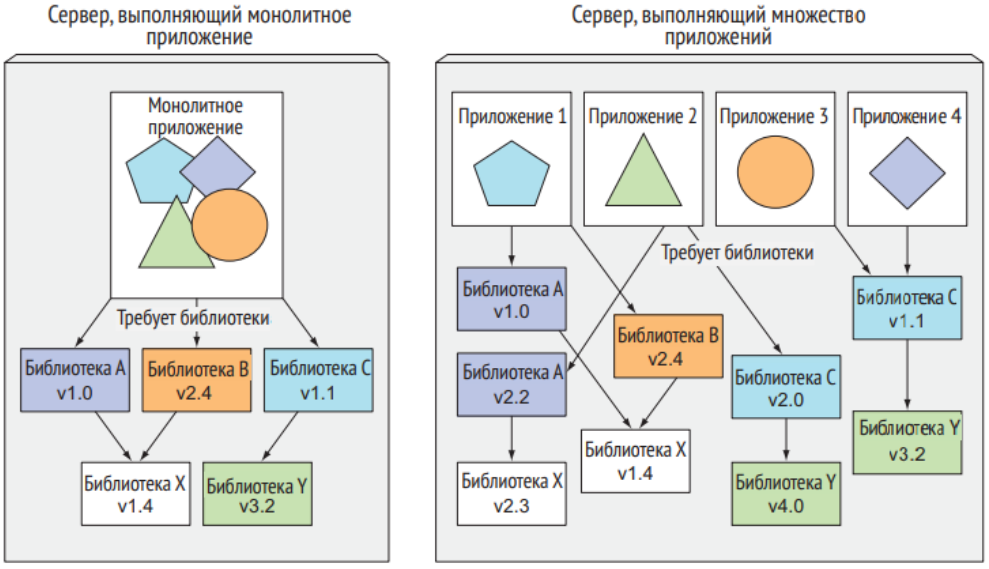

Рис. 5. Дерево зависимостей серверов с монолитом и микросервисами

Когда приложение состоит лишь из небольшого количества крупных компонентов, вполне допустимо предоставить каждому компоненту выделенную виртуальную машину (VM) и изолировать их среды, предоставив каждому из них собственный экземпляр операционной системы. Но когда эти компоненты начинают уменьшаться в объеме и их количество начинает расти, становится невозможным предоставлять каждому из них свою собственную виртуальную машину.

Вместо того чтобы использовать виртуальные машины для изоляции сред каждого микросервиса (или программных процессов в целом), разработчики обращаются к контейнерным технологиям Linux. Данные технологии позволяют запускать несколько сервисов на одной хост-машине, не только обеспечивая доступ к разным средам, но и изолируя их друг от друга, подобно виртуальным машинам, но с гораздо меньшими затратами.

Процесс, запущенный в контейнере, выполняется внутри операционной системы хоста, как и все другие процессы (в отличие от виртуальных машин, где процессы выполняются в отдельных операционных системах). Но процесс в контейнере по-прежнему изолирован от других процессов. Для самого процесса это выглядит, как если бы он был единственным работающим на машине и в ее операционной системе. [16].

Таким образом, сам портал, по сути, должен состоять как минимум из двух контейнеров:

1. контейнер, содержащий в себе приложение, хранящее в себе либо статичные ссылки для запуска проблемно-ориентированных сервисов, либо производящее поиск доступных сервисов для запуска автоматически;
2. контейнер, который является обособленным проблемно-ориентированным сервисом решения некоторого класса задач.

Как можно заметить, концептуально, архитектура всего создаваемого проекта весьма проста, что является неоспоримым преимуществом, ввиду того что такой подход упрощает процесс расширения функционала всей системы.

Поэтому остаётся рассмотреть лишь один вопрос: как организовывается взаимодействие между контейнерами внутри Kubernetes.

2.3. ОПИСАНИЕ МЕХАНИЗМА ВЗАИМОДЕЙСТВИЯ КОНТЕЙНЕРОВ ПОРТАЛА И ПРОБЛЕМНО-ОРИЕНТИРОВАННОГО СЕРВИСА

Kubernetes предлагает более высокий уровень абстракции в управлении контейнерами, в сравнении с Docker или rkt. Достигается это путем использования подов (Pod) - наименьшей единицей для развертывания в объектной модели Kubernetes.

Pod состоит обычно из одного или нескольких контейнеров, хранилища (storage), отдельного IP-адреса и опций, которые определяют, как именно контейнер(ы) должны запускаться. Также Pod представляет собой некий запущенный процесс в кластере Kubernetes. Kubernetes управляет контейнерами посредством управления через Pod, но не напрямую.

В кластере Kubernetes Pod используются двумя способами:

- Pod из одного контейнера - наиболее распространенный случай использования. В этом варианте Pod можно представить как обертку вокруг одного контейнера;
- Pod из нескольких контейнеров, работающих сообща - контейнеры в данном случае образуют отдельную единицу (сервис) и используют общие ресурсы - например, файлы из одного хранилища (storage). При этом, отдельный контейнер Pod'a может обновлять эти файлы. [1]

Каждый Pod предназначен для запуска одного экземпляра конкретного приложения. Если есть необходимость горизонтального масштабирования, то можно запустить несколько экземпляров Pod'a - в терминологии Kubernetes это называется репликацией. Реплицированные Pod'ы создаются и управляются как единая группа абстракцией, называемой контроллер (Controller).

Каждый Pod должен быть спроектирован таким образом, чтобы иметь поддержку множества взаимодействующих процессов (например, контейнеров), которые образуют отдельный сервис (единицу обслуживания, не следует путать их с сервисами как концепции организации взаимодействия).

Контейнеры внутри Pod автоматически размещаются и управляются на одном и том же физическом (или виртуальной) узле кластера. Контейнеры в Pod

могут совместно использовать ресурсы и зависимости, взаимодействовать друг с другом и определять, когда и как они будут завершаться. [12]

К слову, группировка и запуск нескольких контейнеров в Pod считается “продвинутым” вариантом использования, и применять этот шаблон нужно только при реальной необходимости. Например, может быть контейнер с web-сервером, использующий файлы из общего (относительно Pod'a) хранилища, и отдельный контейнер, который обновляет эти файлы из удаленного источника.

Каждый Pod предоставляет запущенным внутри контейнерам два типа общих ресурсов:

- Сеть. Каждому Pod присваивается уникальный IP-адрес. Внутри Pod'a каждый контейнер использует общее пространство имен сети, включая IP-адрес и сетевые порты. Между собой внутри Pod'a контейнеры взаимодействуют через localhost. При взаимодействии с объектами, находящимися за пределами Pod'a, контейнеры используют встроенный в Kubernetes инструмент маршрутизации и координируют использование общих сетевых ресурсов (таких как порты).
- Хранилище. Pod может определить набор общих томов (volumes) для хранения данных. Контейнеры внутри Pod'a могут работать с этими томами и, таким образом, обмениваться данными между собой. Благодаря использованию томов, можно сохранить данные, если один из контейнеров Pod'a (которому нужны эти данные для корректной работы) должен быть перезапущен. Время жизни томов совпадает с временем жизни самого Pod'a. [2]

Что касается работы в кластере Kubernetes, то редко когда приходится создавать и запускать отдельные Pod вручную. Связано это с тем, что Pod изначально проектировались как эфемерные объекты, минимальные единицы, “строительные блоки” для сущностей более высокого уровня (например, контроллеров). Когда Pod создается (не важно, вручную или контроллером), он по плану (за это отвечает планировщик) запускается на одной из нод кластера Kubernetes. На этой ноде Pod остается до тех пор, пока:

- не завершится процесс внутри Pod'a (например, при разовой задаче);
- Pod не будет удален вручную;
- Pod не будет передислоцирован с узла из-за нехватки ресурсов;
- узел не выйдет из строя.

Сами по себе Pod не являются “самолечащимися” (self-healing) объектами. Если запуск Pod'a запланирован на узле, который вышел из строя, или сама операция запуска потерпела крах, то Pod удаляется. Точно также, при нехватке ресурсов на узле (когда Pod'ы передислоцируются) или при переводе узла в режим обслуживания, Pod'ы не будут запущены на других узлах и удалятся. Для обеспечения запуска на других узлах (и управления Pod в целом) в Kubernetes используются абстракции более высокого уровня, называемые контроллерами - в основном именно с ними и приходится иметь дело.

Контроллер может управлять несколькими экземплярами Pod'ов, в том числе обеспечивать репликацию и самовосстановление в пределах кластера (при выходе из строя узла, идентичный экземпляр будет запущен на другом узле). [17]

Таким образом, учитывая особенности оркестрирования, имеется два возможных сценария реализации портала и сервисов:

- Контейнер с порталом и контейнеры сервисов помещаются в отдельных Pod;
- Контейнер портала и контейнеры сервисов помещаются в один Pod.

Безусловно, с точки зрения упрощения реализации, второй сценарий гораздо более привлекателен, т.к.взаимодействие через все модули производится через localhost, но в то же время, будет наноситься большой удар по масштабируемости: при репликации Pod с контейнерами сервиса для балансировки нагрузки, будут создаваться и дополнительные контейнеры портала, что может привести к ошибкам, ввиду того, что контейнер с порталом должен быть только один на кластере.

Следовательно, возможный сценарий остается лишь один – иметь один Pod с контейнером, в котором будет размещаться портал, и некоторое множество Pod’ов, на которых будут размещаться наукоёмкие сервисы. [18]

ГЛАВА 3. РЕАЛИЗАЦИЯ

1.1. ОПИСАНИЕ СРЕДСТВ РЕАЛИЗАЦИИ СЕРВИСОВ

Для реализации как вычислительного портала, так и наукоемкого проблемно-ориентированного сервиса были использованы следующие средства реализации:

- Python 3.8 — это интерпретируемый высокоуровневый язык программирования общего назначения. Философия дизайна Python подчеркивает удобочитаемость кода, а объектно-ориентированный подход направлен на написание кода как для малых, так и крупных проектов. Он используется для написания всех сервисов.
- Visual Studio Code — это легкий и мощный редактор исходного кода, который доступен для операционных систем Windows, macOS и Linux. Он поставляется со встроенной поддержкой JavaScript, TypeScript и Node.js и имеет богатую экосистему расширений для других языков (например, Python).
- Flask — это облегченная среда веб-приложений WSGI. Он предназначен для быстрого и простого создания микросервисов с возможностью масштабирования до сложных приложений. Задумываясь простой оболочкой вокруг Werkzeug и Jinja, Klask стал одной из самых популярных платформ веб-приложений Python.
- SQLite — это автономная база данных без сервера SQL. Используется для хранения информации как о пользователях, так и для хранения ссылок до развернутых в Pod’ах проблемно-ориентированных сервисов.

1.2. СЕРВИС «ВЫЧИСЛИТЕЛЬНЫЙ ПОРТАЛ»

Реализация сервиса «Вычислительный портал» будет производиться на языке Python с использованием Flask и SQLite. Однако, прежде чем переходить к непосредственной реализации, необходимо определится с требованиями, которые будут выдвигаться данному сервису:

- возможность находить доступные сервисы без участия пользователя;
- позволять запускать новые Pod с другими сервисами без вмешательства в код;

- поддерживать неограниченное количество пользователей, при этом:
  - авторизованный пользователь может запускать любые сервисы, доступные на кластере;
  - сервис, запущенный конкретным пользователем, доступен только ему и никому больше;
  - пользователи, не прошедшие авторизацию, не имеют доступ даже к списку доступных сервисов.

Для хранения данных о пользователях и доступных сервисах, в качестве хранилища была выбрана база данных, инфологическая модель которой представлена на рисунке 6.

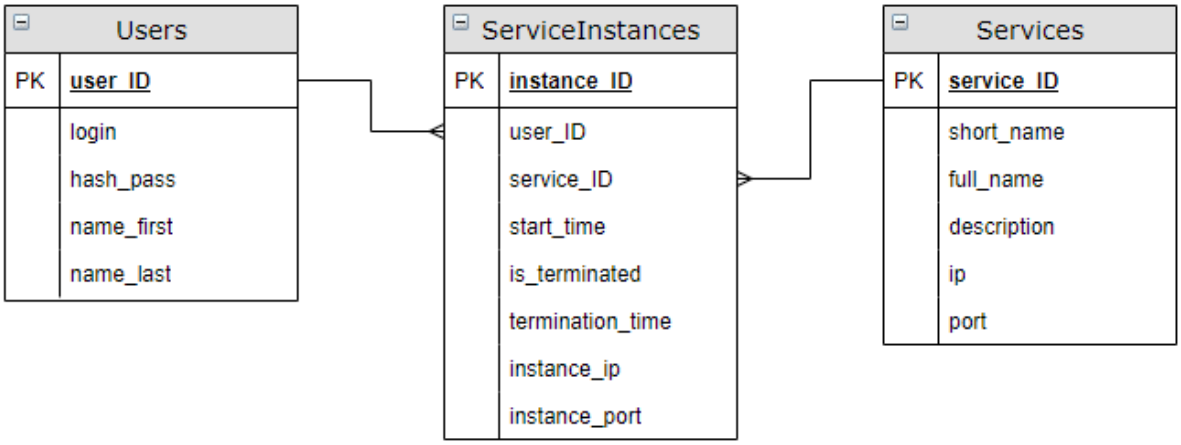

Рис. 6. Инфологическая модель используемого хранилища

Т.к. работа внутри сервисов будет производиться в отдельных контейнерах, то графический интерфейс портала весьма минималистичен:

- главная страница со списком доступных сервисов;
- страницы авторизации и регистрации пользователей.

Графический интерфейс главной страницы представлен на рисунке 7.

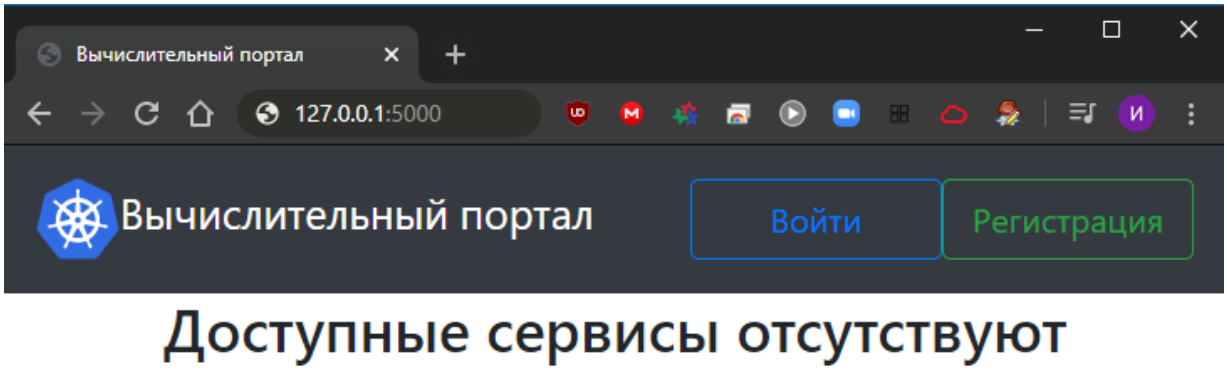

Рис. 7. Главная страница портала

Т.к. поиск сервисов должен производиться, когда сам вычислительный портал будет развернут внутри кластера, то во время тестирования всегда будет указываться надпись «Доступные сервисы отсутствуют» вне зависимости от того, был ли произведен вход или нет.

Для поддержки многопользовательской работы была использована библиотека Flask-Login, которая предоставляет методы для авторизации, хеширования паролей и т.д.

Для поиска доступных сервисов используются проверка на то, что клиент произвел вход под каким-либо пользователем и консольной команды обнаружения сервисов Kubernetes. Взаимодействие с консолью производится при помощи встроенной в Python библиотеки subprocess и её метода call.

Т.к. портал будет находиться в одном из Pod, следовательно ему будут доступны все переменные локального окружения, а также весь перечень команд, используемых Kubernetes для работы. Для поиска доступных Pod можно воспользоваться командой «printenv», которая возвращает список всех запущенных Pod’ов, или же поиском по DNS при помощи “nslookup”.

Но если сервис не был запущен, то такие проверки не увенчаются успехом, однако в таком случае единственным выходом будет являться запуск команды «kubectl get services». Использование kubectl так же допустимо и для поиска запущенных Pod’ов, достаточно лишь изменить её следующим образом: «kubectl get pods»

1.3. СЕРВИС «ПРОБЛЕМНО-ОРИЕНТИРОВАННЫЙ НАУКОЁМКИЙ СЕРВИС»

В целях упрощения разработки, можно воспользоваться уже реализованным приложением, которое позволяет произвести численное моделирование переноса загрязнителя в воздушной среде по заданной области и физическим параметрам, таким как скорость и направление ветра, температура окружаещего воздуха и т.д. [19]

Реализованное программное средство состоит из следующих модулей:

- модуль выбора входных геоданных – оконное приложение, позволяющее выбрать исследуемую область, используя сторонний

картографический web-сервис, и позволяющий пользователю указать примерное расположение объектов (загрязнителей, построек) с целью увеличения точности построения модели.

- модуль распознавателя объектов – модуль, в котором происходит процесс распознавания объектов на изображении, с последующей генерацией файла Q1;
- модуль обучения классификатора – используется для обучении классификатора с целью повышения точности распознавания и загрузки весов ранее обученного классификатора;
- главный модуль – связывает модули между собой и CFD PHOENICS.

У имеющегося решения есть ровно один момент, который усложняет использование – зависимость от внешнего приложения вычислительной гидродинамики PHOENICS. Ввиду того, что имеющаяся в наличии версия PHOENICS предназначена для ОС Windows, то имеются некоторые ограничения, которые будут обсуждаться в разделе контейнеризации приложений.

Однако, несмотря на это, разработанное приложение возможно использовать в качестве проблемноориентированного науко-емкого сервиса, т.к. получение данных производится извне, оно имеет свой собственный графический интерфейс, и, помимо PHOENICS, больше нет никаких специфических зависимостей.

1.4. РАЗВОРАЧИВАНИЕ ЛОКАЛЬНОГО КЛАСТЕРА

Для того, чтобы воспользоваться Kubernetes не обязательно иметь в распоряжении целый кластер. Достаточно воспользоваться таким инструментом, как Minikube, позволяющий запустить Kubernetes на локальной машине, запуская одноузловой кластер внутри виртуальной машины.

Minikube поддерживает следующие возможности Kubernetes:

- DNS;
- Сервисы NodePort;
- Словари конфигурации (ConfigMaps) и секреты (Secrets);
- Панель управления (Dashboard);

- Среда выполнения контейнера: Docker, CRI-O и containerd;
- Поддержка CNI (Container Network Interface);
- Ingress

Процесс установки подразуемевает сперва создание и настройки некоторого локального окружения, а именно установку утилиты kubectl и некоторого гипервизора (Hyper-V, VirtualBox, VMWare).

Установка kubectl предельно проста – достаточно загрузить исполняемый файл и поместить его по одному из путей переменной окружения PATH. Для того, чтобы убедиться в том, что установка произведена корректно, можно вызвать команду «kubectl», вызов которой продемонстрирован на рисунке 8.

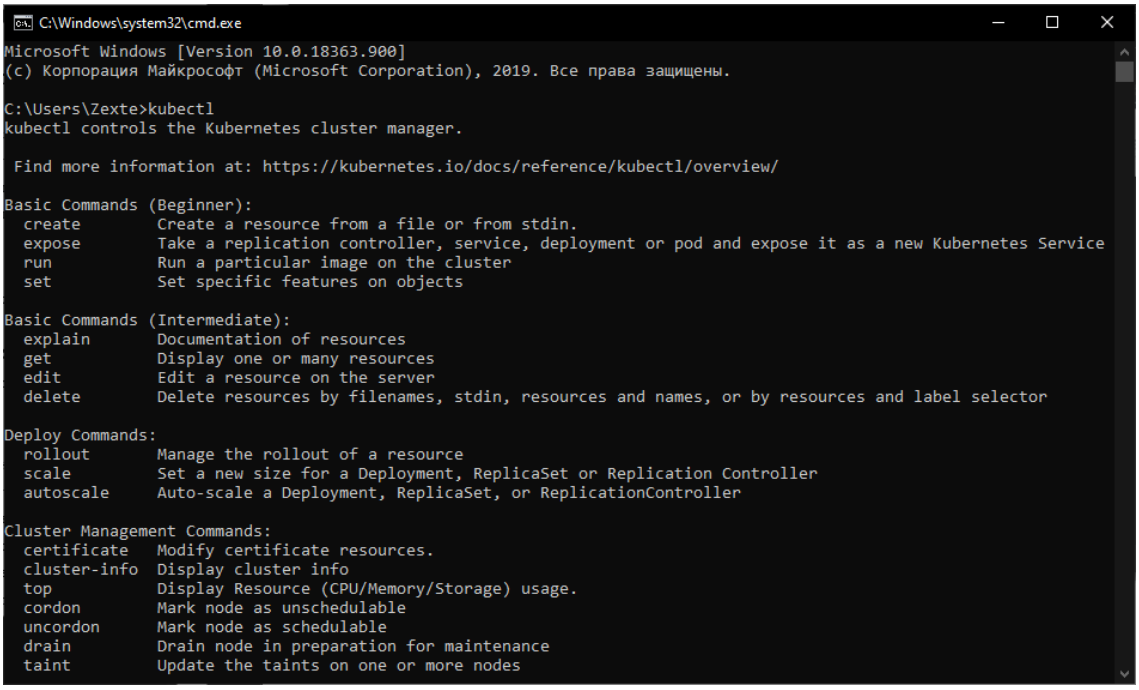

Рис. 8. Вызов команды «kubectl»

После этого необходимо установить утилиты окружения Minikube с официального сайта. Для удобства, все они собраны в едином исполняемом файле, который размещает их по указанному пользователем пути.

Для начала работы необходимо выполнить команду «minikube start --vm-driver=hyperv» (или любой другой драйвер в зависимости от установленного гипервизора)

В результате выполнения этой команды будет настроено локальное окружение при помощи специального файла настроек kubeconfig, выделены ресурсы и создана виртуальная машина, содержащая локальный кластер [20].

Проверить работоспособность можно при помощи команды «minikube status». Если установка произведена корректно, то в результате мы получим сведения, что все сервисы запущены, так же, как и убедимся, что ничего, кроме самого Kubernetes, более не запущено, что и проиллюстрировано на рисунке 9.

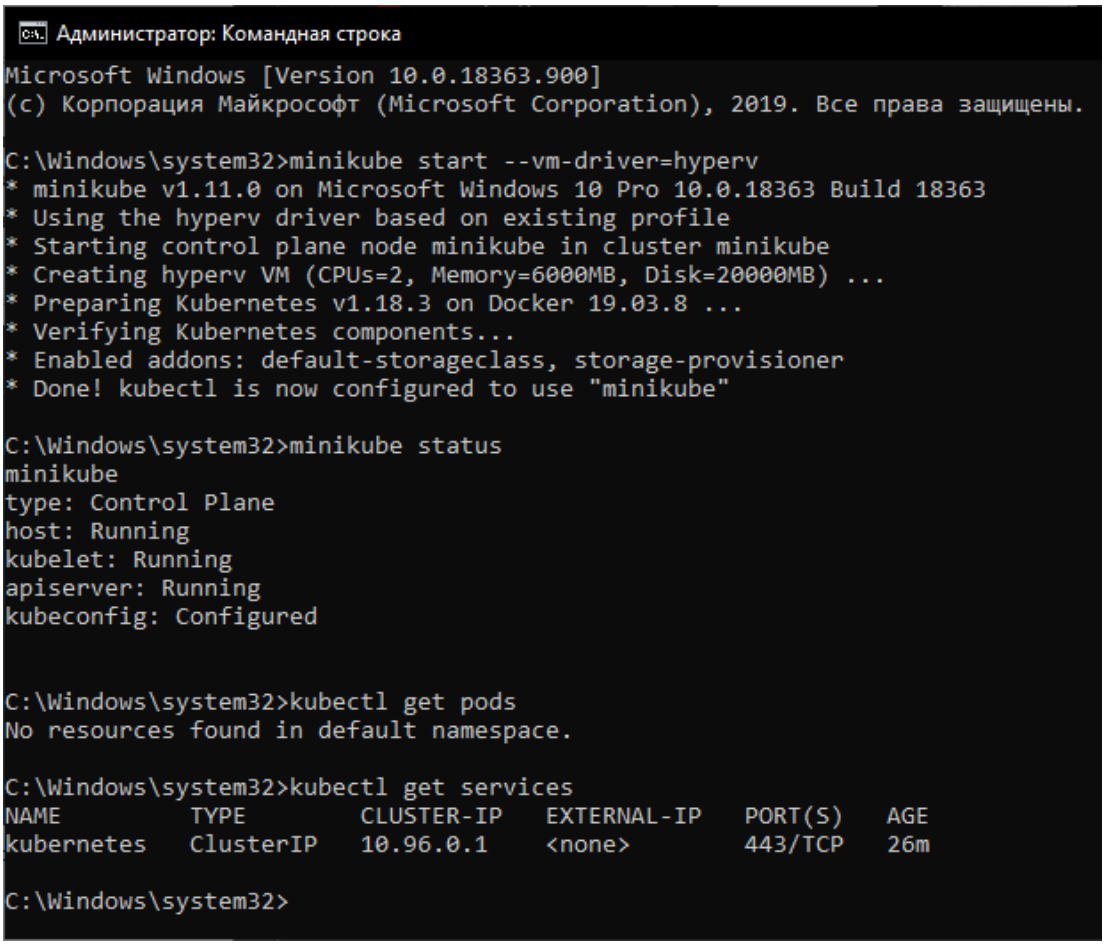

Рис. 9. Отчет о работоспособности Minikube

1.5. КОНТЕЙНЕРИЗАЦИЯ РАЗРАБОТАННЫХ СЕРВИСОВ И РАЗВОРАЧИВАНИЕ ИХ НА ЛОКАЛЬНОМ КЛАСТЕРЕ

Все подготовительные работы, необходимые для основного действия, а именно, контейнеризации совершены, однако сейчас можно столкнуться с проблемой, о которой упоминалось ранее – привязка PHOENICS к ОС Windows.

Когда речь заходит о контейнеризации, то нужно сперва разобрать как происходит процесс упаковки хотя-бы на концептуальном уровне.

Для сборки требуется всего два объекта – само приложение и dockerfile. Последний – файл, содержащий в себе параметры сборки. Параметры собраны по группам в количестве 12 штук:

- FROM — задаёт базовый образ.

- LABEL — описывает метаданные — сведения о том, кто создал, поддерживает образ, какие аргументы запуска имеются и т.д.
- ENV — устанавливает постоянные переменные среды.
- RUN — выполняет команду и создаёт слой образа. Используется для установки в контейнер пакетов.
- COPY — копирует в контейнер локальные файлы и папки.
- ADD — копирует файлы и папки в контейнер, в том числе и удаленные ресурсы. Может распаковывать tar-архивы.
- CMD — описывает команду с аргументами, которую нужно выполнить когда контейнер будет запущен. Аргументы могут быть переопределены при запуске контейнера. В файле может присутствовать лишь одна инструкция CMD.
- WORKDIR — задаёт рабочую директорию следующей инструкции.
- ARG — задаёт переменные Docker во время сборки образа.
- ENTRYPOINT — предоставляет команду с аргументами для вызова во время выполнения контейнера. Аргументы не переопределяются.
- EXPOSE — указывает на необходимость открыть порт.
- VOLUME — создаёт точку монтирования для работы с постоянным хранилищем. [21]

С классическими приложениями Windows дела обстоят не самым лучшим образом именно из-за параметра FROM. Всё дело в том, что при установке любого приложения некоторые его параметры записываются в реестр. Это же и происходит с PHOENICS – если скопировать рабочую директорию с компьютера, на котором он установлен, на компьютер, где его не было, то приложение будет завершаться с ошибкой спустя несколько секунд после запуска.

В настоящий момент имеется неразрешимая проблема с упаковкой такого приложения, но в качестве временного решения может выступать монтирование директории, с установленным PHOENICS и предоставление полных прав на взаимодействие контейнера с этой директорией.

Для приложений, которые собираются из исходного кода, или с универсальными приложениями Windows (UWP) никаких проблем не возникает. Наоборот, указывая в параметре FROM компактный образ или даже лишь один интерпретатор, можно получить приложение, которое запустится, причем где угодно, главное – чтобы сам контейнер можно было на той системе запустить.

Единственное решаемое неудобство – по умолчанию, ни Docker, ни Kubernetes не разрешает запускать X-Server (набор пакетов обеспечивающих приём от пользователя управляющих сигналов с устройств ввода и позиционирования, с последующей отрисовкой на устройстве вывода). Однако это можно решить как добавлением аргументов запуска приложения в контейнере, так и при запуске сервиса вызовом соответствующей последовательности команд из сервиса «Вычислительный портал».

Т.к. приложения «Вычислительный портал» и «Наукоемкий проблемно-ориентированный сервис» разработаны на языке Python, то процесс их запаковки в контейнер схожи, поэтому рассмотрим процесс на примере первого приложения.

Для запаковки понадобится корректный DockerFile. Сборка будет производиться под alpine, по двум причинам:

1) Популярный образ Ubuntu:latest, несмотря на рекомендации, с приложениями на языке Python работает гораздо медленнее, чем любые другие дистрибутивы.
2) Сборка контейнера на базе чистого интерпретатора python невозможна ввиду того, что процесс взаимодействия между Pod’ами будет сильно усложнен ввиду отсутствия поддержки терминальных команд.

Dockerfile, который используется для сборки приложения в контейнер представлен в листинге 1. Как видно из его содержания помимо основной системы, так же устанавливаются python и его зависимости, удобно собранные в файле requirements.txt. Для того, чтобы собрать финальный контейнер достаточно перейти в директорию, расположенную на один уровень выше каталога, содержащего Dockerfile и выполнить команду «docker build -t <название>:latest <директория>.

Листинг 1. Dockerfile приложения «Вычислительный портал»

```
FROM python:alpine3.7
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD python ./main.py
```

Конец Листинга 1

Для того, чтобы удостовериться в том, что контейнер успешно создан, достаточно выполнить команду «docker images». Если в списке имеется созданный контейнер, то можно попробовать запустить его в среде docker, путём вызова команды «docker run --name <синоним> -d -p 5000:5000 <имя_образа>».

Результат вызова всех вышеописанных команд продемонстрирован на изображении 10.

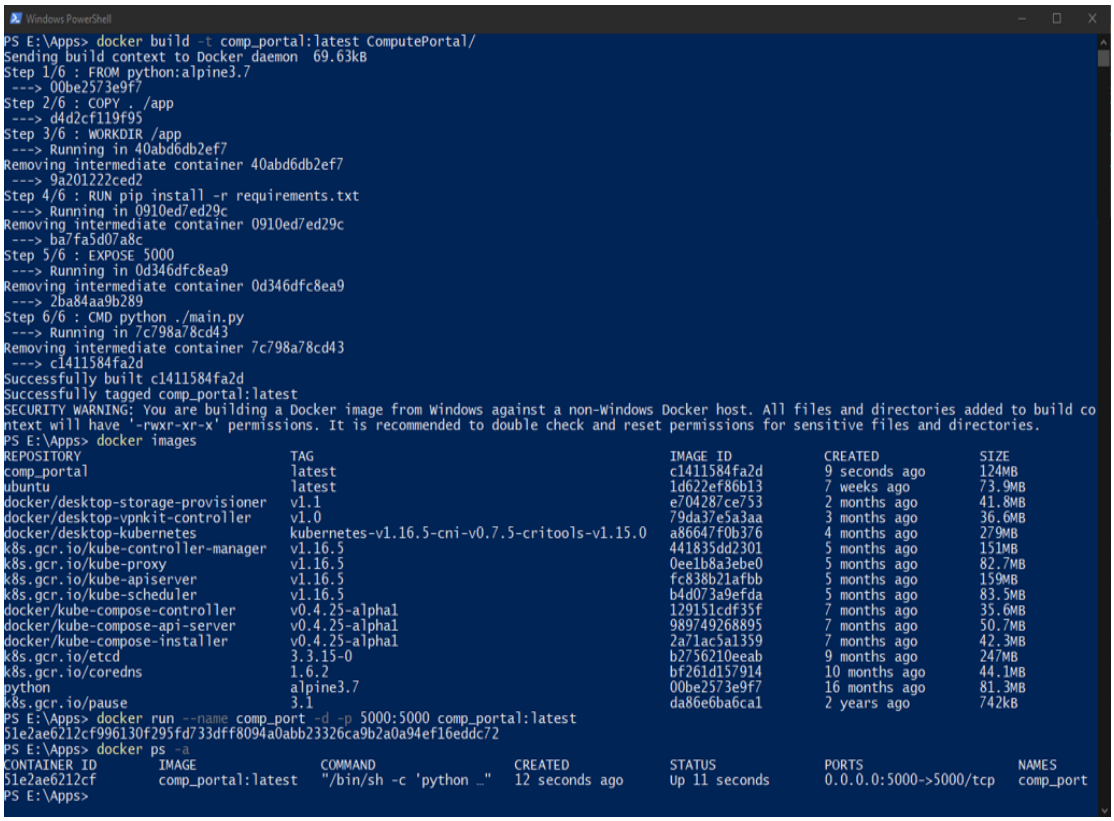

Рис. 10. Создание контейнера из приложения.

Dockerfile для сервиса «Наукоёмкий проблемно-ориентированный сервис» выглядит аналогичным образом, однако использует образ Ubuntu:latest поскольку требуется X-сервер и содержит дополнительные разделы VOLUME для связи с PHOENICS, CMD, в котором производится запуск X-сервера и

LABEL, в котором в ключах svc-name и svc-meta передаётся название сервиса и его описание в понятном для пользователя виде (используется на портале в карточке объекта). [22]

Теперь нужно создать pod, который будет содержать docker-контейнер. Для этого нужно создать специальный конфигурационный файл в формате YAML. Для каждого сервиса будет выделяться отдельный pod, поэтому нужно использовать несколько конфигурационных файлов. Содержимое конфигурационного файла для сервиса «Вычислительный портал» приведено в листинге 2.

Листинг 2. Dockerfile приложения «Вычислительный портал»

```
apiVersion: v1
kind: Pod
metadata:
  name: comp-port
spec:
  containers:
  - name: computational-portal
    image: computational_portal:latest
  hostNetwork: true
  dnsPolicy: Default
```

Конец Листинга 2

После создания конфигурационного файла достаточно просто запустить команду «kubectl run comp-port --image=computational_portal --port=5000 --image-pull-policy=Never» чтобы создать Pod с сервисом «Вычислительный портал». Результат работы данной команды продемонстрирован на рисунке 11.

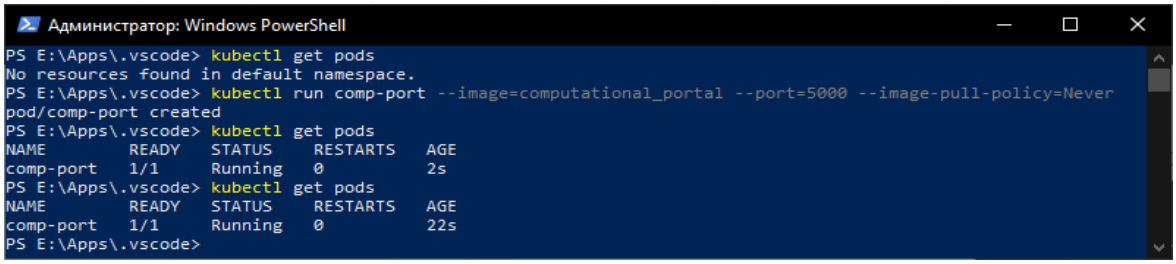

Рис. 11. Запуск Pod’а

Развертка Pod’а для сервиса «Наукоёмкий проблемно-ориентированный сервис» происходит аналогичным образом без каких-либо изменений.

Остаётся только проверить работоспособность обоих сервисов. Для этого необходимо запустить сервис «Вычислительный портал», проверить, был ли найден сервис из другого Pod’а и запустить его.

При подключении по локальному адресу Pod’а при помощи браузера открывается главная страница портала, изображенная на рисунке 12. А при нажатии на кнопки «Запустить» в карточке сервиса, открывается окно X-сервера, представленное на рисунке 13, что говорит о работоспособности приложения.

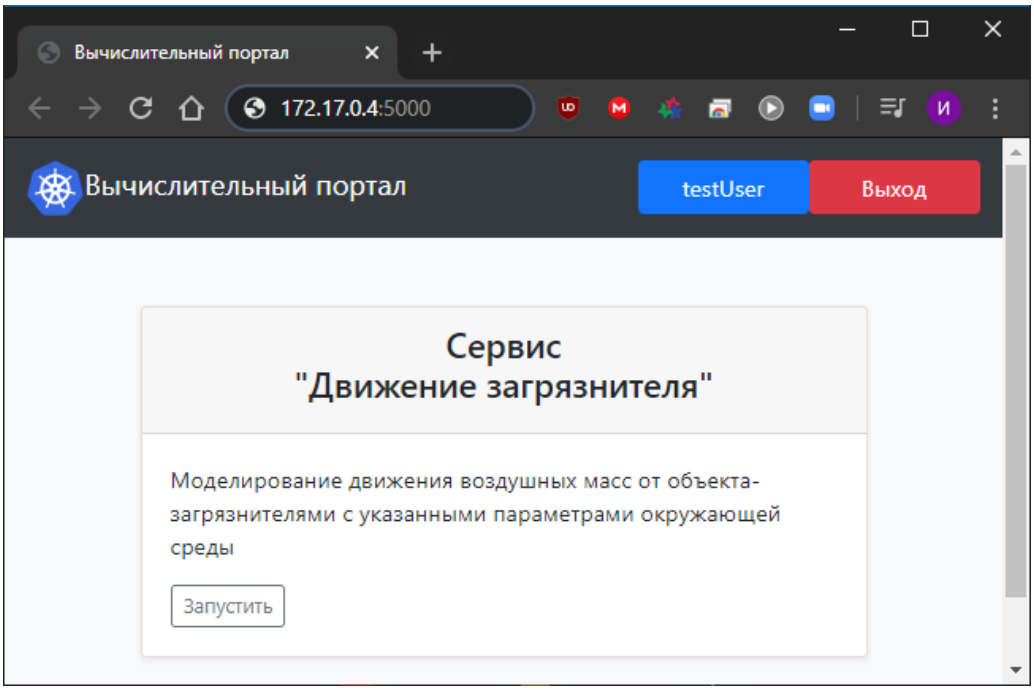

Рис. 12. Главная страница вычислительного портала

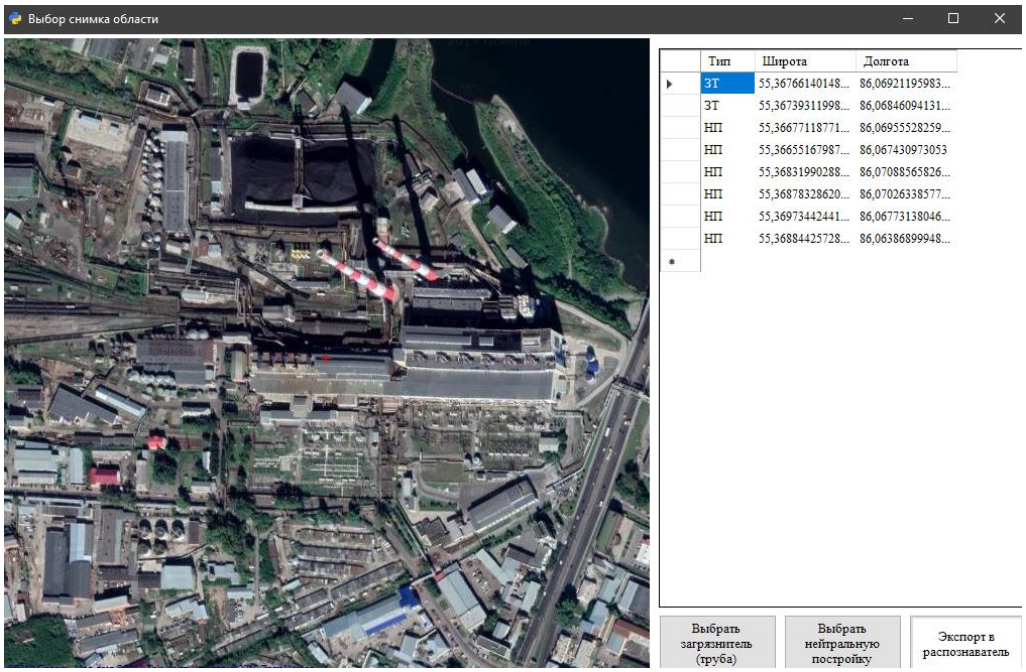

Рис. 13 Главное окно модуля выбора области

ЗАКЛЮЧЕНИЕ

В результате проведённой работы достигнуты следующие результаты:

- проведен анализ системы управления контейнерами Kubernetes в качестве хоста вычислительного портала, в ходе которого были выявлены основные функции и возможности системы, а так же проведено сравнение с другими решениями для создания вычислительных порталов;
- создана тестовая платформа для развертки сервисов на языке Python с использованием библиотеки для разработки web-приложений Flask, изолированной базы данных SQLite; данная платформа не использует большого количества ресурсов и поддерживает запаковку в изолированный контейнер, что вписывается в общую концепцию контейнеризации;
- проведен обзор доступных технологий, используемых для создания унифицированных самоизолированных сервисов; в качестве такой технологии была выбрана контейнеризация, как наиболее востребованная в современном мире и предоставляющая как простоту развертки, так и универсальность;
- рассмотрены способы межмодульного взаимодействия; в качестве такого способа выступает изоляция отдельных модулей в различных Pod’ах, т.к. передача данных между модулями не предусматривается, а для взаимодействия модулей со специфичным ПО используются общие дисковые тома, подключаемые непосредственно к контейнеру с сервисом;
- произвести тестирование развернутых сервисов на изолированном локальном кластере Kubernetes с использованием minikube в качестве основной тестирующей среды; тестирование продемонстрировало, что взаимодействие между сервисами и самим локальным окружением выстроено корректно, ввиду того, что сервисы «видимы» друг для друга, а сам Kubernetes занимается как выделением ресурсов для Pod’ов, равно как и занимается их сетевой маршрутизацией.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Маркелов, А. А. Введение в технологию контейнеров и Kubernetes [Электронный ресурс] / А. А. Маркелов. — Москва : ДМК Пресс, 2019. — 194 с. — ISBN 978-5-97060-775-6. — // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131702. — Режим доступа: для авториз. пользователей.

2. Лукша, М. Kubernetes в действии [Электронный ресурс] / М. Лукша; перевод с английского А. В. Логунов. — Москва : ДМК Пресс, 2019. — 672 с. — ISBN 978-5-97060-657-5. — // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/131688. — Режим доступа: для авториз. пользователей.

3. Гудов А. М. Инженерный вычислительный портал для решения экологических задач [Текст] / А. М. Гудов, С. Ю. Завозкин, И. Ю. Сотников // Экология и безопасность в техносфере: современные проблемы и пути решения : сборник трудов Всероссийской научно-практической конференции молодых ученых, аспирантов и студентов, г. Юрга, 23-25 ноября 2017 г. — Томск : Изд-во ТПУ, 2017. — [С. 455-460].

4. Гудов А. М. Один из подходов к созданию вычислительных сервисов для инженерного вычислительного портала [Текст] / А. М. Гудов, С. Ю. Завозкин, И. Ю. Сотников // Цифровые технологии в образовании, науке, обществе : материалы XII всероссийской науч.-практ. конф. (4 – 6 декабря 2018 года). — Петрозаводск, 2018. — 268 с. — [С. 85 - 89].

5. Гудов А. М. Механизм создания вычислительных сервисов для инженерного вычислительного портала [Текст] / А. М. Гудов, И. Ю. Сотников // Тезисы XIX Всероссийской конференции молодых учёных по математическому моделированию и информационным технологиям. г. Кемерово, Россия, 29 октября – 2 ноября 2018 г. — Новосибирск: ИВТ СО РАН, 2018. — 94 стр. — [С. 77 - 78].

6. Доррер, Г.А. Теория информационных процессов и систем [Электронный ресурс] : учеб. пособие / Г.А. Доррер .— Красноярск : СибГТУ, 2008 .— 196 с. — Режим доступа: https://rucont.ru/efd/206257

7. Горлушкина, Н. Н. Системный анализ и моделирование информационных процессов и систем [Текст] : учебное пособие / Н. Н. Горлушкина. — Санкт-Петербург : НИУ ИТМО, 2016. — 120 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/110469. — Режим доступа: для авториз. пользователей.

8. Бизли, Д. Python. Подробный справочник [Электронный ресурс] / Д. Бизли. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 864 с. – Режим доступа: http://prog.tversu.ru/library/Python.pdf.

9. Лутц, М. Программирование на Python, 4-е издание в 2-х томах [Текст] / М. Лутц. – М.: Символ-Плюс, 2011. – 992 с.

10. Что такое Kubernetes [Электронный ресурс] / Авторы Kubernetes, 2020. – Режим доступа: kubernetes.io/ru/docs/concepts/overview/what-is-kubernetes/

11. Компоненты Kubernetes [Электронный ресурс] / Авторы Kubernetes, 2020. – Режим доступа: kubernetes.io/ru/docs/concepts/overview/components/

12. Сидоров А. Так что же такое pod в Kubernetes? [Электронный ресурс] / А. Сидоров. – Хабр, 2020. – Режим доступа: https://habr.com/ru/company/flant/blog/427819/

13. Что вас ждёт до, после и во время перехода на Kubernetes [Электронный ресурс] / Новосибирск : ООО «Тру Инжиниринг», 2020. — Режим доступа: https://trueengineering.ru/ru/cases/k8s_for_business

14. Облачные вычисления второго поколения: Система CLAVIRE [Электронный ресурс] / Университет ИТМО. – Хабр, 2017. – Режим доступа: https://habr.com/ru/company/spbifmo/blog/319688/

15. Белоножко П.П. Свободные облачные аппаратно-программные платформы. Аналитический обзор [Электронный ресурс] // Белоножко П. П., Белоус В. В., Куцевич Н. А., Храмов Д. А. Вестник евразийской науки.

2016. №6 (37). Режим доступа: https://cyberleninka.ru/article/n/svobodnye-oblachnye-apparatno-programmnye-platformy-analiticheskiy-obzor

16. Руководство по Kubernetes, часть 1: приложения, микросервисы и контейнеры [Электронный ресурс] / ru_vds. – Хабр, 2019. – Режим доступа: https://habr.com/ru/company/ruvds/blog/438982/

17. Изучение подов и узлов [Электронный ресурс] / Авторы Kubernetes, 2020. – Режим доступа: https://kubernetes.io/ru/docs/tutorials/kubernetes-basics/explore/explore-intro/

18. Три уровня автомасштабирования в Kubernetes: как их эффективно использовать [Электронный ресурс] / pxeno, Блог компании Mail.ru Group, 2020. – Режим доступа: https://habr.com/ru/company/mailru/blog/484344/

19. Степанов И.Ю. Применение ГИС-технологий в задачах моделирования процессов загрязнения окружающей среды : магистерская диссертация [Текст] / И.Ю. Степанов. Кемерово, КемГУ. – 2019 г. – 52 с.

20. Procida C. Getting Started with Kubernetes via Minikube [Электронный ресурс] / C. Procida, Medium. – 2016. – Режим доступа: https://medium.com/@claudiopro/getting-started-with-kubernetes-via-minikube-ada8c7a29620

21. Hale J. Learn Enough Docker to be Useful [Электронный ресурс] / J. Hale, Towards Data Science. – 2019. – Режим доступа: https://towardsdatascience.com/learn-enough-docker-to-be-useful-b0b44222eef5

22. Шпаргалка по kubectl [Электронный ресурс] / Авторы Kubernetes, 2020. – Режим доступа: https://kubernetes.io/ru/docs/reference/kubectl/cheatsheet/