|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |



ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.03 Прикладная информатика**

Оглавление

[Индивидуальное задание 4](#_Toc33683600)

[Введение 5](#_Toc33683601)

[Основная часть 6](#_Toc33683602)

[1 Высоконагруженные системы 6](#_Toc33683603)

[1.1 Понятие высоконагруженной системы 6](#_Toc33683604)

[1.2 Качества высоконагруженной системы 6](#_Toc33683605)

[1.3 Особенности высоконагруженных систем 6](#_Toc33683606)

[1.4 Использование микросервисов в высоконагруженных системах 8](#_Toc33683607)

[2 Технологии контейнеризации 10](#_Toc33683608)

[2.1 Контейнер как метод виртуализации 10](#_Toc33683609)

[2.2 Сравнение виртуальных машин и контейнеров 11](#_Toc33683610)

[2.3 Сравнение различных технологий контейнеризации 13](#_Toc33683611)

[2.4 Docker 14](#_Toc33683612)

[3 Система оркестровки контейнеров Kubernetes 15](#_Toc33683613)

[3.1 Архитектура кластера Kubernetes 16](#_Toc33683614)

[3.2 Процесс развертывания компонентов программной системы в Kubernetes 18](#_Toc33683615)

[3.3 Преимущества использования Kubernetes 19](#_Toc33683616)

[4 Проектирования программной системы с использованием Kubernetes и Docker 20](#_Toc33683617)

[4.1 Выбор базовых программных компонентов проектируемой системы 20](#_Toc33683618)

[4.2 Простейший вариант программной архитектуры проектируемой системы 21](#_Toc33683619)

[4.3 Масштабируемый вариант архитектуры проектируемой системы 22](#_Toc33683620)

[4.4 Реализация проектируемой системы в кластере Kubernetes 23](#_Toc33683621)

[Заключение 26](#_Toc33683622)

[Список использованных источник 27](#_Toc33683623)

[Приложение А – Техническое задание 28](#_Toc33683624)

Индивидуальное задание

* изучить понятие высоконагруженной системы и критерии их оценки;
* изучить современные подходы к проектированию высоконагруженные систем;
* исследовать современные подходы к поддержанию бесперебойной работоспособности высоконагруженных сервисов;
* ознакомиться с наиболее популярными и часто используемыми технологиями контейнеризации;
* изучить возможные способы разворачивания приложения с использованием технологии контейнеризации в облачной инфраструктуре;
* исследовать систему оркестровки контейнерами «Kubernetes», выявить её основные достоинства и недостатки, определить перечень ситуаций для оправданного использования;
* спроектировать программную систему с использованием облачной реализации Kubernetes «Google Kubernetes Engine»;
* выявить потенциальные «узкие» места в проектируемой системе;
* обеспечить отказоустойчивость проектируемой системы;
* обеспечить возможность горизонтальной масштабируемости компонентов проектируемой системы;
* обеспечить автоматичесую масштабируемость отдельных компонентов системы в зависимости от текущей нагрузки с помощью средств, которые предоставляет Kubernetes.

Введение

Основной целью практики является получение целостного представления об работе современных высоконагруженных систем и способах их проектирования и поддержания их в рабочем состоянии.

Задачи практики в ООО «Мэйл.Ру»:

* исследовать существующие подходы к проектированию высоконагруженные систем;
* изучить современные подходы к поддержанию бесперебойной работоспособности высоконагруженных сервисов;
* ознакомиться с наиболее популярными и часто используемыми технологиями контейнеризации;
* опробовать на практике возможные способы разворачивания приложения с использованием технологии контейнеризации в облачной инфраструктуре;
* исследовать системы оркестровки контейнерами «Kubernetes», выявить её основные достоинства и недостатки, определить перечень ситуаций для оправданного использования;
* спроектировать программную систему с использованием облачной реализации Kubernetes «Google Kubernetes Engine»;
* найти потенциальные «узкие» места в проектируемой системе;
* обеспечить горизонтальную масштабируемость компонентов проектируемой системы;
* обеспечить отказоустойчивость проектируемой программной системы.

Основным результатом прохождения практики является: индивидуальное задание, выполненное по поручению руководителя практики от кафедры и отчет, в котором отражены все результаты деятельности за период прохождения практики.

Основная часть

# Высоконагруженные системы

## Понятие высоконагруженной системы

Понятие высоконагруженной системы является расплывчатым, нет конкретных цифр, благодаря которым ту или иную программную системы можно отнести к этому классу.

Высоконагруженная система – это совокупность программных компонентов, обеспечивающая бесперебойный доступ клиентам, предоставляя им набор сервисом, которая постоянно масштабируется для поддержания работоспособности.

Из определения следует, что основной признак высоконагруженной системы – это постоянная необходимость в ее улучшения (масштабировании и добавлении ресурсов) ввиду постоянного роста нагрузки.

## Качества высоконагруженной системы

Нельзя точно сказать, начиная с каких количественных показателей система может считаться высоконагруженной, но можно выявить некоторые качества, которые могут помочь отнести систему к классу высоконагруженных

* обслуживает большое количество клиентов;
* системы является распределенной;
* постоянно растущая по количеству клиентов система;
* развивающаяся система;
* система, обладающая большим количеством ресурсов и растущей потребностью в них;

## Особенности высоконагруженных систем

Разработка высоконагруженной системы не так сильно отличается от просто грамотной разработки веб-приложения (большинство представителей современных высоконагруженных систем – это веб-приложения, ввиду того, что Интернет обладает громадной аудиторией, где каждый пользователь может быть клиентом высоконагруженного сервиса).

Ключевым элементом в такой разработке является правильное проектирование архитектуры приложения, т. к. повышения нагрузки на разрабатываемую систему не должно вызывать необходимость в изменение логики работы компонентов, добавлению новых, удалению старых.

Обеспечение горизонтальной масштабируемости компонентов системы на всех уровнях и предварительное выявление и ликвидация узких мест – наиболее эффективный шаг для обеспечения эффективного роста и легкой модернизации разрабатываемой системы в отдаленном будущем, когда рост нагрузки приведет к дефициту ресурсов при текущей конфигурации оборудования.

На данный момент для реализации больших систем с большим количеством различных компонентов (сейчас почти любая высоконагруженная система обладает большими размерами и большим разнообразием используемых компонентов) используются технологии виртуализации.

Независимо от того, сколько отдельных компонентов разрабатывается и развертывается, одна из самых больших проблем, с которой всегда приходится сталкиваться разработчикам и системным администраторам, – это различия в окружениях, в которых они выполняют свои приложения. Мало того, что существует огромная разница между окружением разработки и рабочим окружением, различия даже существуют между отдельными машинами в рабочем окружении. Еще одним неизбежным фактом является то, что окружение одной рабочей машины будет меняться с течением времени.

Виртуализация позволяет абстрагироваться от конфигурации реальных физических машин и запускать компоненты системы в отдельных виртуальных машинах, тем самым делая их выполнение независимым от ОС, инсталированной на физическом сервере.

## Использование микросервисов в высоконагруженных системах

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. Изменения в одной части приложения требуют новой выкладки всего приложения, и со временем отсутствие жестких границ между частями приводит к увеличению сложности и последующему ухудшению качества всей системы из-за неограниченного роста взаимосвязей между этими частями.

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 (так называемое масштабирование вверх), 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 (масштабирование вширь). Хотя масштабирование вверх обычно не требует каких-либо изменений в приложении, оно относительно быстро становится дорогостоящим и на практике всегда имеет верхний предел.

Масштабирование вширь, с другой стороны, является относительно дешевым аппаратно, но может потребовать больших изменений в программном коде приложения и не всегда возможно – некоторые части приложения с большим трудом поддаются горизонтальному масштабированию или почти невозможны для него (например, реляционные базы данных). 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 начать разбиение сложных монолитных приложений на небольшие независимые развертывания компонентов, называемых микросервисами. Каждый микросервис выполняется как независимый процесс (рис. 1) и взаимодействует с другими микросервисами через простые, четко определенные интерфейсы (API).

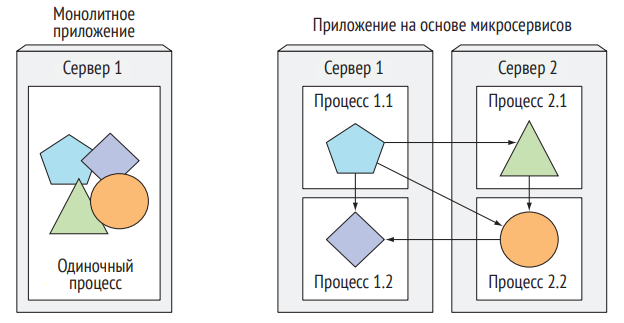


Рисунок – Компоненты программной системы внутри монолитного приложения и независимых микросервисов

Микросервисы взаимодействуют через синхронные протоколы, такие как HTTP, используя которые, они обычно предоставляют RESTful, или через асинхронные протоколы, такие как AMQP (Advanced Message Queueing Protocol, расширенный протокол организации очереди сообщений). Эти протоколы просты, хорошо понятны большинству разработчиков и не привязаны к какому-либо конкретному языку программирования. Каждый микросервис может быть написан на языке, который наиболее целесообразен для реализации конкретных микросервисов.

Поскольку каждый микросервис представляет собой автономный процесс с относительно статическим внешним API, существует возможность разрабатывать и развертывать каждый микросервис отдельно. Изменение одной из них не требует изменений или повторного развертывания какого-либо другого сервиса, при условии, что API не изменяется или изменяется только обратно совместимым образом.

# Технологии контейнеризации

## Контейнер как метод виртуализации

Когда приложение состоит лишь из небольшого количества крупных компонентов, вполне допустимо предоставить каждому компоненту выделенную виртуальную машину и изолировать их среды, предоставив каждому из них собственный экземпляр операционной системы.

Однако, когда эти компоненты начинают уменьшаться в объеме и их количество начинает расти, становится невозможным предоставлять каждому из них свою собственную виртуальную машину.

Поскольку каждая виртуальная машина обычно должна настраиваться и управляться индивидуально, увеличение количества виртуальных машин также приводит к трате человеческих ресурсов, поскольку они значительно увеличивают рабочую нагрузку на системных администраторов.

Вместо того чтобы использовать виртуальные машины для изоляции сред каждого микросервиса (или программных процессов в целом), можно использовать контейнерные технологии 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.

Контейнеризация - метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного.

Эти экземпляры (обычно называемые контейнерами) с точки зрения пользователя полностью идентичны отдельному экземпляру операционной системы.

Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

При контейнеризации отсутствуют дополнительные ресурсные накладные расходы на эмуляцию виртуального оборудования и запуск полноценного экземпляра операционной системы, характерные при аппаратной виртуализации.

## Сравнение виртуальных машин и контейнеров

По сравнению с виртуальными машинами, контейнеры гораздо облегченнее, что позволяет запускать большее количество программных компонентов на одном и том же оборудовании, главным образом потому, что каждая виртуальная машина должна запускать свой собственный набор системных процессов, который требует еще вычислительных ресурсов в дополнение к тем, которые потребляются собственным процессом компонента.

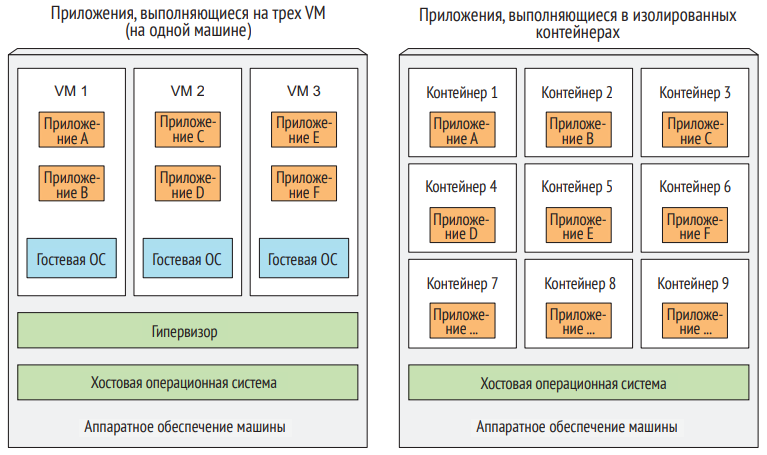


Рисунок – Сравнение виртуальных машин для изоляции групп приложения с изоляцией отдельных приложения с помощью контейнеров

Контейнеры выполняют системные вызовы на одном и том же ядре, работающем в хостовой ОС. Это единственное ядро, выполняющее инструкции на процессоре хоста. ЦП не нужно делать какой-либо виртуализации, как он делает с виртуальными машинами.

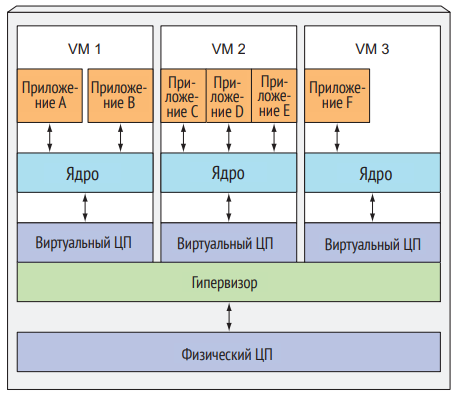


Рисунок – Приложения, выполняющиеся на множестве виртуальных машин

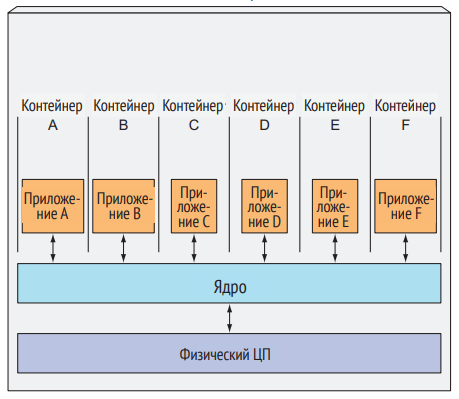


Рисунок – Приложения, выполняющиеся в изолированных контейнерах

## Сравнение различных технологий контейнеризации

Существуют реализации, ориентированные на создание практически полноценных экземпляров операционных систем (Solaris Containers, контейнеры Virtuozzo, OpenVZ), так и варианты, фокусирующиеся на изоляции отдельных сервисов с минимальным операционным окружением (jail, Docker).

Таблица – Сводная таблица технологий контейнеризации



По таблице 1 можно увидеть, что существует большое количество реализаций технологии контейнеризации с различными особенностями, однако самая популярная на данный момент – это Docker.

## Docker

Хотя контейнерные технологии существуют уже давно, они стали более широко известны с появлением контейнерной платформы Docker.

Docker была первой контейнерной системой, которая сделала контейнеры легко переносимыми на разные машины. Это упростило процесс упаковки не только приложения, но и всех его библиотек и других зависимостей, даже всей файловой системы ОС, в простой, переносимый пакет, который может использоваться для подготовки приложения к работе на любой другой машине, на которой работает Docker.

При выполнении приложения, упакованного с помощью Docker, оно видит точное содержимое файловой системы, поставляемое вместе с ним. Оно видит одни и те же файлы, независимо от того, работает ли оно на машине, предназначенной для разработки, или же на машине из рабочего окружения, даже если на рабочем сервере запущена совершенно другая ОС Linux.

Процесс создания образа и запуска контейнера проиллюстрирован на рисунке ниже.

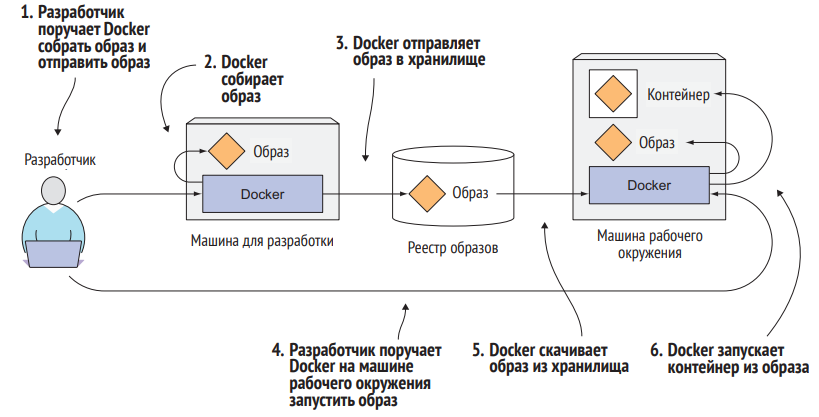


Рисунок – Процесс создания докер-образа и запуска контейнера на рабочем узле программной системы

# Система оркестровки контейнеров Kubernetes

По мере роста программной системы ею становится все сложнее управлять из-за количества разнородных разворачиваемых компонентов. Для решения данной проблемы была разработана такая система, как Kubernetes.

Kubernetes – это программная система, которая позволяет легко развертывать контейнеризированные приложения и управлять ими. Она использует возможности контейнеров Linux для запуска разнородных приложений без необходимости знать какие-либо внутренние детали этих приложений и без необходимости вручную развертывать эти приложения на каждом хосте.

Поскольку данные приложения работают в контейнерах, они не влияют на другие приложения, работающие на том же сервере, что имеет решающее значение при запуске приложений для совершенно разных организаций на одном и том же оборудовании.

Это имеет первостепенное значение для облачных провайдеров, поскольку они стремятся к максимально возможной задействованности своего оборудования, сохраняя при этом полную изоляцию размещенных приложений.

Kubernetes позволяет выполнять программные приложения на тысячах компьютерных узлов, как если бы все эти узлы были одним огромным компьютером. Она абстрагируется от базовой инфраструктуры и тем самым упрощает разработку, развертывание и управление как для разработчиков, так и для системных администраторов.

Процедура развертывания приложений через Kubernetes всегда одинаковая, независимо от того, содержит ли кластер всего несколько узлов или тысячи. Размер кластера не имеет никакого значения. Дополнительные узлы кластера просто представляют собой дополнительный объем ресурсов, доступных для развернутых приложений.

Простейший вид системы Kubernetes представлен на рисунке ниже. Система состоит из ведущего узла (мастера) и любого количества рабочих узлов. Когда разработчик отправляет список приложений ведущему узлу, Kubernetes развертывает их в кластере рабочих узлов. То, на какой узел приземляется компонент, не имеет (и не должно иметь) значения ни для разработчика, ни для системного администратора.

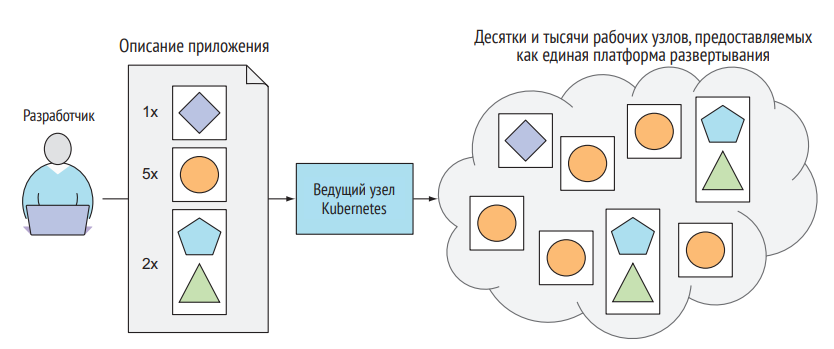


Рисунок – Общий вид системы Kubernetes

Kubernetes можно рассматривать как операционную систему для кластера. Она избавляет разработчиков приложений от необходимости внедрять в свои приложения определенные службы, связанные с инфраструктурой; вместо этого в вопросе предоставления этих служб они опираются на Kubernetes. Это включает в себя такие аспекты, как обнаружение службы, масштабирование, балансировка нагрузки, самовосстановление и даже выбор лидера. Поэтому разработчики приложений могут сосредоточиться на реализации реального функционала приложений и не тратить время на то, чтобы разбираться в том, как интегрировать их с инфраструктурой.

## Архитектура кластера Kubernetes

На аппаратном уровне кластер Kubernetes состоит из множества узлов, которые можно разделить на два типа:

* ведущий узел (мастер), на котором размещена плоскость управления (Control Plane) Kubernetes, контролирующая и управляющая всей системой Kubernetes;
* рабочие узлы, на которых выполняются развертываемые приложения.

На рисунке ниже показаны компоненты, работающие на этих двух типах узлов Kubernetes.

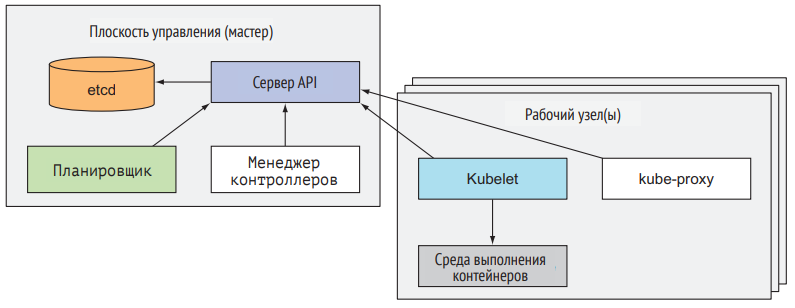


Рисунок 7 – Компоненты кластера Kubernetes

Плоскость управления – это то, что управляет кластером и заставляет его функционировать. Она состоит из нескольких компонентов, которые могут работать на одном ведущем узле либо быть распределены по нескольким узлам и реплицированы для обеспечения высокой доступности.

Рабочие узлы – это машины, на которых выполняются контейнеризированные приложения.

Для запуска приложения в Kubernetes необходимо упаковать его в один или несколько образов контейнеров, отправить эти образы в хранилище образов, а затем опубликовать описание приложения на сервере API Kubernetes.

Описание содержит такие сведения, как образ или образы контейнеров, содержащие компоненты приложения, как эти компоненты связаны друг с другом и какие из них должны выполняться совместно, а какие нет.

## Процесс развертывания компонентов программной системы в Kubernetes

Когда сервер API обрабатывает описание приложения, планировщик назначает указанные группы контейнеров доступным рабочим узлам, исходя из вычислительных ресурсов, требуемых каждой группой, и нераспределенных ресурсов на каждом узле в данный момент. Агент Kubelet на этих узлах затем поручает среде выполнения контейнеров (например, Docker) извлечь из хранилища требуемые образы контейнеров и запустить контейнеры.

После отправки дескриптора в Kubernetes он запланирует использование указанного количества реплик каждого модуля на доступных рабочих узлах. Затем агенты Kubelet на узлах поручают Docker извлечь образы контейнеров из реестра образов и запустить контейнеры

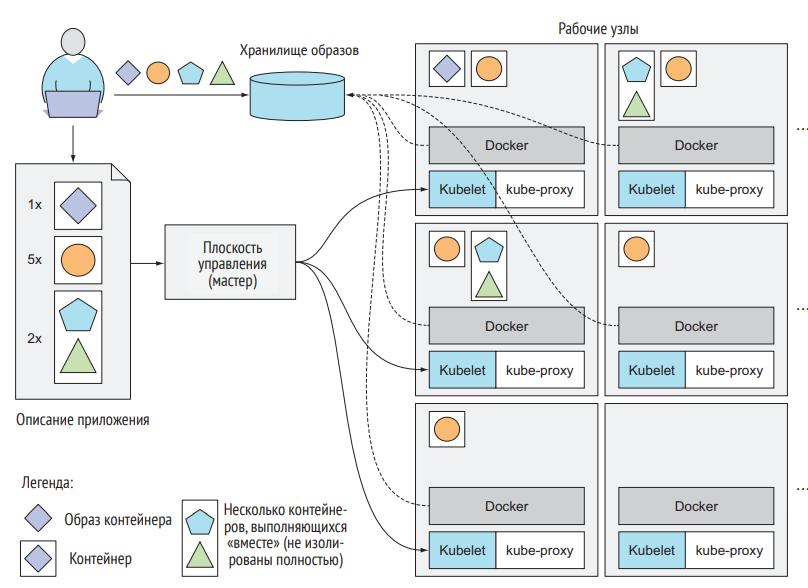


Рисунок – Процесс развертывания приложения в Kubernetes

## Преимущества использования Kubernetes

Упрощенное развертывание приложений. Kubernetes обеспечивает доступ ко всем своим рабочим узлам как к единой платформе развертывания. По сути, все узлы теперь представляют собой единую группу вычислительных ресурсов, которые ждут, когда приложения будут их потреблять.

Повышение эффективности задействования оборудования. Настройка кластера Kubernetes позволяет отделить программную систему от инфраструктуры. Компоненты программной системы могут свободно запускаться и перемещаться между доступными узлами в кластере, что позволяет эффективно использовать аппаратные ресурсы.

Проверка здоровья и самолечение. Kubernetes отслеживает компоненты приложения и узлы, на которых они выполняются, и автоматически переносит их на другие узлы в случае аварийного сбоя узла. Также в случае аварийного завершения работы одного из рабочих физических узлов Kubernetes, работавшие до этого на нем модули будут автоматически переназначены на другие узлы, тем самым обеспечивая бесперебойность работы всей системы.

Автоматическое масштабирование. Если Kubernetes работает в облачной инфраструктуре, где добавлять дополнительные узлы так же просто, как запрашивать их через API поставщика облака, Kubernetes даже может автоматически масштабировать размер всего кластера вверх или вниз в зависимости от потребностей развернутых приложений.

Упрощение разработки приложений. Системы позволяет быстро разворачивать новые версии программных компонентов. Kubernetes позволяет также настроить процесс постепенного обновления программных компонентов, сделав его наиболее безопасным и плавным. В случае аварийной ситуации Kuberenetes может прервать процесс выкладки и вернуться к предыдущей стабильной конфигурации кластера.

# Проектирования программной системы с использованием Kubernetes и Docker

Для проектирования программной системы с использованием технологии контейнеризации и системы оркестровки контейнерами Kubernetes необходимо выделить компоненты разрабатываемой системы, а также взаимосвязи между ними, после чего интегрировать их в кластер Kubernetes, используя предоставляемые им инструменты и сервисы.

## Выбор базовых программных компонентов проектируемой системы

Поскольку разрабатываемая система представляет из себя веб-приложения, доступ к которому осуществляется посредством использования сети Интернет, то для отображения графического интерфейса будет использоваться браузер.

Минимальная реализация подобной программной системы должна включать в себя минимум три компонента: сервис для отдачи веб-приложения браузеру, сервис для обработки запросов от веб-приложения и хранилище данных.

Для реализации данных программных компонентов разрабатываемой системы были выбраны следующие языки программирования и технологии:

* для отдачи веб-приложения на браузер пользователя будет использоваться frontend-сервер Node.js;
* для обработки действия пользователя, отдачи данных и обработки запросов будет использоваться веб-сервис, реализованный на языке программирования GO;
* для хранения данных пользователей и служебных данных разрабатываемой системы, планируется использовать реляционную базу данных Postgres 10.1.

## Простейший вариант программной архитектуры проектируемой системы

Наиболее простой вариант программной архитектуры с использованием frontend-backend подхода представлен на рисунке 9.

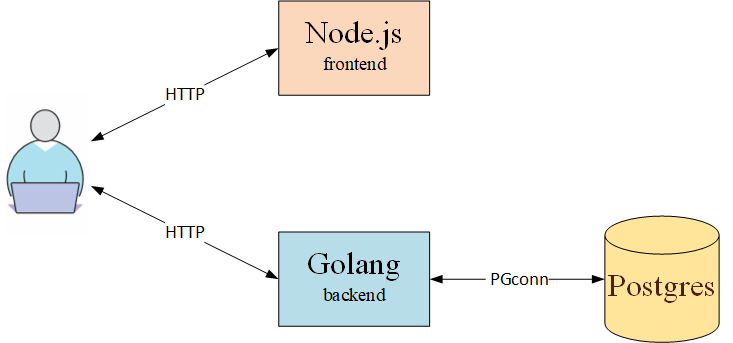


Рисунок – Простой вариант программной архитектуры веб-приложения

Данный подход к проектированию программной системы имеет как плюсы, так и недостатки.

Плюсы:

* при такой архитектуре количество разнородных программных компонентов не велико.
* нет накладных расходов на дополнительные сервисы, обеспечивающие инфраструктуру приложения (балансировки нагрузки, прокси-серверы, регистраторы сервисов и другие)

Минусы:

* отсутствует единая точка входа для пользователя в программную систему
* масштабируемость для каждого из компонентов системы требует изменение конфигурации других элементов системы
* компоненты системы представляют из себя большие монолитные программы, некоторый функционал из которых может быть вынесен в отдельные сервисы
* отсутствуют компоненты программной системы, позволяющие равномерно распределить нагрузки между однородными компонентами системы
* выход из строя одного из компонентов приводит к отказу всей системы

С учётом всех плюсов и недостатков описанная выше архитектура может быть использована в небольших проектах, но для высоконагруженных систем она не годится в связи с громоздкостью отдельных компонентов и потенциальными проблемами при масштабировании.

## Масштабируемый вариант архитектуры проектируемой системы

На рисунке 10 представлен вариант архитектуры программной системы с учетом необходимости в масштабировании отдельных компонентов и требованием к отказоустойчивости.



Рисунок – Программная система с масштабируемой архитектурой

Заметна существенна разница между первым вариантом архитектуры и новым. Можно также выделить в данной архитектуре список достоинств и недостатков

Достоинства:

* каждый компонент проектируемой системы может быть горизонтально отмасштабирован
* нет единой точки отказа, выход из строя одного из компонентов системы не ведет к потере ее работоспособности
* для добавления дополнительного frontend или backend сервера достаточно только изменить конфигурацию прокси-сервера nginx.

Недостатки:

* в данной системе добавление компонентов при масштабировании все еще требует изменению конфигурации других компонентов
* отсутствует инфраструктура для автоматического масштабирования в зависимости от текущей нагрузки
* данная архитектура требует от администратора системы дополнительных временных затрат времени из-за в целом более высокой сложности по сравнению с предыдущим вариантом

Всё выявленные недостатки в данной архитектуре можно исправить при помощи использования системы оркестровки контейнеров Kubernetes.

## Реализация проектируемой системы в кластере Kubernetes

Для переноса программных компонентов системы необходимо прежде всего использовать технологию контейнеризации Docker. За счет этого каждый программный компонент будет работать изолированно от других запущенных на физической машине контейнеров и независимо от самой ОС и конфигурации этой самой машины.

Для построения инфраструктуры для проектируемой программной системы в кластере Kubernetes потребуются перечень инструментов Kubernetes, представленный в таблице 2.

Таблица – Используемые средства Kubernetes

|  |  |  |
| --- | --- | --- |
| Название средства Kubernetes | Тип | Выполняемая функция |
| Deployment | Контроллер развертывания | Развертывание экземпляров одного из компонентов программной системы |
| Service | Сервис | Обеспечение доступа между компонентами программной системы и из вне |
| Vault | Хранилище | Создание временных и постоянных хранилищ данных |
| Secret | Чувствительная информация | Хранения паролей и секретной информации для обеспечения контроля доступа |

Итоговая архитектура представлена на рисунке ниже.



Рисунок – Архитектура программной системы в Kubernetes

Такие средства Kubernetes, как vault и secret создаются и настраиваются посредством конфигурации соответствующих опций в ресурсах deployment. Кластер Kubernetes самостоятельно определит машины для размещения хранилища данных (vault) и паролей (secret).

Ресурсы deployment позволяют в автоматическом режиме разворачивать экземпляры программных компонентов и увеличивать/уменьшать их количество в зависимости от текущей нагрузки (если соответствующих критерии были определены администратором).

В случае использования такого ресурса kubernetes, как service, больше нет необходимости менять конфигурацию компонентов программной системы в случае изменения количества работающих элементов. Сервисы самостоятельно отслеживают запуск новых экземпляров программных компонентов и обеспечивают равномерное распределение нагрузки.

Kubernetes использует декларативный подход. Это значит, что администратору достаточно только описать желаемое состояние кластера, а дальше сама система уже сделает все необходимое, чтобы прийти к заданному состоянию. Это значительно облегчает расходы на администрирование и обслуживание подобных систем

Заключение

В результате проделанной работы была спроектирована программная системы с использованием технологий виртуализации (с использованием технологии контейнеризации Docker) и системы оркестровки контейнерами Kubernetes.

В ходе работы были достигнуты следующие результаты:

* изучено понятие высоконагруженной системы и критерии их оценки;
* изучены современные подходы к проектированию высоконагруженные систем;
* исследованы современные подходы к поддержанию бесперебойной работоспособности высоконагруженных сервисов;
* было проведено ознакомление с наиболее популярными и часто используемыми технологиями контейнеризации;
* изучены возможные способы разворачивания приложения с использованием технологии контейнеризации в облачной инфраструктуре;
* исследована система оркестровки контейнеров «Kubernetes», выявить её основные достоинства и недостатки, определить перечень ситуаций для оправданного использования;
* спроектирована программная система с использованием Kubernetes;
* обеспечена отказоустойчивость проектируемой системы;
* обеспечена возможность горизонтальной масштабируемости компонентов проектируемой системы;
* обеспечена автоматическая масштабируемость отдельных компонентов системы.

Список использованных источник

1. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. — СПб.: Питер, 2018. — 640 с.
2. Марко Лукша. Kubernetes в действии / пер. с англ. А. В. Логунов. – М.: ДМК Пресс, 2019. – 672 с.
3. Сайфан Джиджи. Осваиваем Kubernetes. Оркестрация контейнерных архитектур. — СПб.: Питер, 2019. — 400 с.
4. Раджпут Динеш. Spring. Все паттерны проектирования. — СПб.: Питер, 2019. — 320 с.

Приложение А – Техническое задание

**Листов 11**