Санкт-Петербургский Государственный Университет

Кафедра системного программирования
09.04.04 «Программная инженерия»
21.М07-мм

Ким Анатолий Александрович

Отчет по учебной практике

«Разработка и реализация прототипа Kubernetes-кластера для РЦ ВЦ СПбГУ»

Научный руководитель:
доцент кафедры ВФ, к. ф.-м. н. Степанова М. М.

Санкт-Петербург
2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ....3
1 КОНТЕЙНЕРИЗАЦИЯ ПРИЛОЖЕНИЙ И ОБОСНОВАННОСТЬ ЕЕ ИСПОЛЬЗОВАНИЯ....5
1.1 Зонная виртуализация. Обзор и характеристика технологии....5
1.2 Отличие контейнера от виртуальной машины....8
2 ОПИСАНИЕ ИНФРАСТРУКТУРНЫХ КОМПОНЕНТОВ ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА....11
2.1 Основы Kubernetes- стандарта для разработки высоконагруженных проектов....11
2.2 Развертывание сервера с помощью технологии Docker....16
2.3 Grafana как инструмент для визуализации метрик мониторинга....20
ЗАКЛЮЧЕНИЕ....22
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ....23

ВВЕДЕНИЕ

Впервые такое направление, как контейнеризация приложений появилось достаточно давно, но востребованным оно стало только после 2013 года, благодаря Docker. И с тех пор технология успешно развивается, открывает для себя новые возможности, расширяет области применения. Согласно статистическим данным, уже в следующем году свыше 75% мировых компаний перейдут на контейнеры.

На данный момент в структурных подразделениях Санкт-Петербургского государственного университета отсутствует платформа, позволяющая студентам, сотрудникам и иным заинтересованным лицам, использовать вычислительные мощности Ресурсного центра «Вычислительный центр СПбГУ» для образовательных целей, научно-исследовательских работах с использованием удаленного доступа к готовому к использованию рабочего пространства.

Таким образом, цель данной работы – разворачивание инфраструктуры по предоставлению доступа к изолированной среде со всеми необходимыми компонентами для задач, определяемых самим пользователем. Для данных целей РЦ ВЦ СПбГУ выделил сервера и предоставил удаленный доступ к ним. Основным инструментом развертывания контейнеризированных сред будет выступать Docker, а в качестве инструмента оркестровки данными контейнерами – Kubernetes. Для мониторинга серверов и контейнеризизированного кластера, а также отображения всевозможных циклических метрик будет использовать open-source проект Grafana.

Задачи:

1. Изучить основные направления контейнеризации приложений и обосновать необходимость ее использования.

2. Провести сравнительный анализ контейнера и виртуальной машины.

3. Изучить основы Kubernetes.

4. Изучить способы развертывания сервера с помощью технологии Docker.

5. Изучить инструмент Grafana в разрезе визуализации метрик мониторинга.

1 КОНТЕЙНЕРИЗАЦИЯ ПРИЛОЖЕНИЙ И ОБОСНОВАННОСТЬ ЕЕ ИСПОЛЬЗОВАНИЯ

1.1 Зонная виртуализация. Обзор и характеристика технологии

Контейнеризация (виртуализация на уровне операционной системы, контейнерная или зонная виртуализация) – это метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. С точки зрения пользователя эти экземпляры (контейнеры или зоны) полностью идентичны отдельной операционной системе. Ядро обеспечивает полную изолированность контейнеров, поэтому приложения из разных контейнеров не могут воздействовать друг на друга.

Технология контейнеризации далеко не новая и не новаторская технология. Она используется уже довольно длительное время. Тем не менее, данная технология стала набирать значительную популярность с приходом облачных технологий. Недостатки традиционных виртуальных машин стали катализатором роста популярности контейнеров. Поставщики инструментов для работы с контейнерами, например, Docker, в значительной степени упростили технологию контейнеризации, что способствовало внедрению данной технологии в широкие массы. Популярность DevOps и 42 микросервисной архитектуры также ускорила перерождение технологии контейнеризации.

Технология контейнеризации предоставляет приватное окружение в операционной системе. Данная технология также называется виртуализация операционной системы [8]. В данном подходе, ядро операционной системы предоставляет изолированное виртуальное пространство. Каждое из виртуальных пространств называется контейнером. Контейнеры позволяют процессам создавать изолированное окружение на операционной системе,

содержащие эти контейнеры. На рисунке 1.1 изображены различные слои, вовлеченные в процесс контейнеризации.

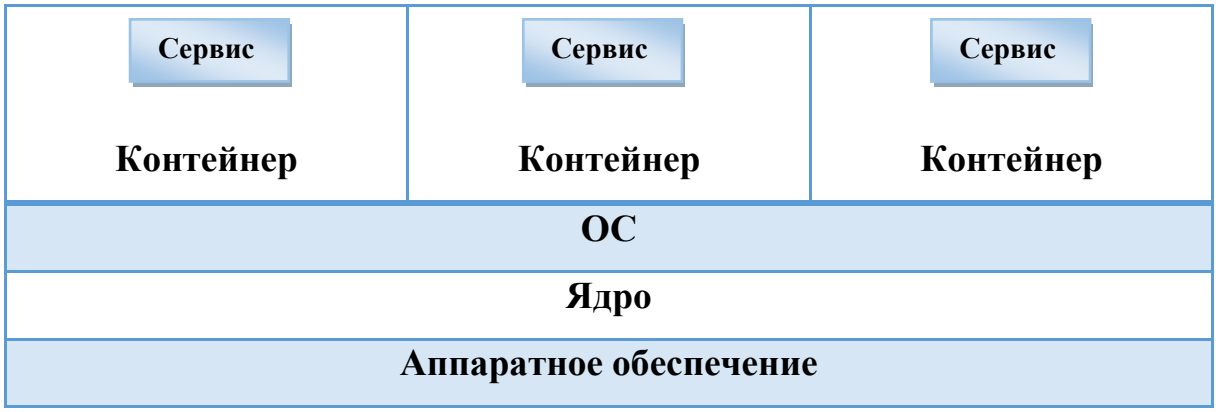

Рисунок 1.1 – Слои контейнеризации

Контейнеры, это простой механизм для сборки и поставки слабосвязанных компонентов программного обеспечения. В общем случае, контейнеры упаковывают все выполняемые файлы и библиотеки, которые необходимы для запуска приложения. Контейнеры полностью изолируют следующие элементы:

- файловую систему;
- IP адрес;
- сетевые интерфейсы;
- внутренние процессы;
- пространства имен;
- библиотеки операционной системы;
- бинарные файлы приложения;
- зависимости;
- файлы конфигурации приложения.

Все инструменты контейнеризации основаны на функциональности ядра Linux. Основные компоненты ядра Linux для контейнеризации перечислены ниже:

- пространства имен;
- управляющие группы.

Пространства имен (namespaces) используются для создания изолированного окружения, которое и называется контейнер. Когда запускается контейнер, докер создает новые пространства имен для этого контейнера. Пространство имен предоставляют из себя слой изоляции. Каждый аспект работы контейнера выполняется в отдельном пространстве.

Ниже перечислены группы пространств имен:

- PID;
- сетевое взаимодействие;
- межпроцессное взаимодействие;
- точки монтирования файловой системы;
- изолированное ядро и идентификаторы версий.

Управляющие группы (control groups) также, как и пространства имен содержатся в ядре Linux. Управляющие группы ограничивают ресурсы, потребляемые приложением. Также они позволяют движку докера делить доступные аппаратные ресурсы между контейнерами и ограничивать или расширять эти ресурсы по необходимости. Например, можно ограничить использование памяти для какого-то определённого контейнера.

Различные организации используют миллиарды контейнеров. Кроме этого, много крупных компаний инвестируют в технологию контейнеризации. Docker значительно опережает конкурентов и поддерживается многими крупными поставщиками операционных систем, а также облачными 44

провайдерами. Также в текущее время разрабатывается открытая спецификация контейнеров.

1.2 Отличие контейнера от виртуальной машины

В отличие от аппаратной виртуализации, когда эмулируется аппаратное окружение и может быть запущено множество гостевых ОС, в контейнере запускается экземпляр ОС только с тем же ядром, что и у исходной (хостовой) операционной системы. Таким образом, все контейнеры узла используют общее ядро. При этом отсутствуют дополнительные накладные расходы ресурсов на эмуляцию виртуального оборудования и запуск полноценного экземпляра ОС, что характерно аппаратной виртуализации. Существуют реализации для создания практически полноценных экземпляров ОС (Solaris Containers, контейнеры Virtuozzo, OpenVZ), а также варианты изоляции отдельных сервисов с минимальным операционным окружением (jail, Docker) [1], которые наиболее часто используются в Big Data системах с учетом роста популярности микросервисной архитектуры.

Таким образом, контейнеры позволяют уместить гораздо больше приложений на одном физическом сервере, чем любая виртуальная машина, которая занимает гораздо больше системных ресурсов (оперативной памяти и процессорных циклов). В отличие от виртуальной машины, где эмулируется ОС и необходимое виртуальное оборудование, в контейнере размещается только приложение и необходимый минимум системных библиотек. Благодаря этому можно запустить в 2–3 раза больше приложений на одном сервере. Также контейнеризация позволяет создавать портативное и целостное окружение для разработки, тестирования и последующего развертывания, что соответствует подходу DevOps

С точки зрения использования контейнеризации приложений в Big Data наиболее значимы следующие достоинства этой технологии:

– стандартизация – благодаря базе открытых стандартов контейнеры могут работать во всех основных дистрибутивах Linux, Microsoft и других популярных ОС;

– независимость контейнера от ресурсов или архитектуры хоста, на котором он работает, облегчает переносимость образа контейнера из одной среды в другую, обеспечивая непрерывный конвейер DevOps-процессов от разработки и тестирования к развертыванию (CI/CD pipeline);

– изоляция – приложение в контейнере работает в изолированной среде и не использует память, процессор или диск хостовой ОС, что гарантирует изолированность процессов внутри контейнера и обеспечивает некоторый уровень безопасности. Подробнее про безопасность в контейнерных технологиях читайте в нашей новой статье;

– возможность повторного использования – все компоненты, необходимые для запуска приложения, упаковываются в один образ, который может запускаться многократно;

– быстрота развертывания – на создание и запуск контейнера требуется гораздо меньше времени, чем на экземпляр виртуальной машины или настройку полноценного рабочего окружения;

– повышение производительности труда – если каждый микросервис сложной системы упаковать в отдельный контейнер, за который отвечает один разработчик, то можно распараллелить рабочие задачи без взаимных зависимостей и конфликтов;

– упрощение мониторинга – управление версиями контейнерных образов позволяет отслеживать обновления, избегая проблем с синхронизацией.

Тем не менее, при всех вышеотмеченных достоинствах, контейнерной виртуализации свойственны следующие недостатки:

– «упаковка» в контейнер гораздо большего количества ресурсов, чем требуется приложению, что приводит к разрастанию образа и большому размеру контейнера;

– проблема с обеспечением информационной безопасности – несмотря на то, что многие системы работы с контейнерами (Kubenetes, Docker и пр.) включают средства защиты от взлома и потери данных, например, изоляция пространства имен, гибкие механизмы сетевых политик и ролевой авторизации пользователей по ключу, а также прочие инструменты, о которых мы подробнее рассказываем в следующей статье, использование контейнеров не всегда безопасно. Например, некоторые важные подсистемы ядра операционной системы Linux функционируют вне контейнера (переферийные устройства, SELinux, Cgroups и вся файловая система внутри системного раздела /sys). Благодаря этому можно взломать операционную систему, если клиент (пользователь или приложение) контейнера имеет root-права;

– проблема обеспечения качества при увеличении числа контейнеров, включая распределение рабочей нагрузки – для решения этой задачи используются системы оркестрации контейнеров, о которых мы поговорим далее.

2 ОПИСАНИЕ ИНФРАСТРУКТУРНЫХ КОМПОНЕНТОВ ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

2.1 Основы Kubernetes- стандарта для разработки высоконагруженных проектов

Kubernetes была задумана корпорацией Google как платформа контейнерной оркестрации с открытым исходным кодом, предназначенная для развертывания, масштабирования и управления контейнерными приложениями. Kubernetes стала де факто общепринятым стандартом контейнерной оркестрации и флагманским проектом Фонда развития облачных вычислений (англ. Cloud Native Computing Foundation), который поддержали такие ключевые игроки рынка как Google, AWS, Microsoft, IBM, Intel, Cisco, и Red Hat.

Kubernetes позволяет легко внедрять и использовать приложения в микросервисной архитектуре, создавая уровень абстракции поверх группы хостов. В результате команды разработчиков могут развертывать приложения и позволяют платформе Kubernetes:

- Контролировать, сколько ресурсов потребляет приложение или команда
- Равномерно распределять нагрузку приложений по инфраструктуре хоста
- Автоматически отправлять запросы на балансировку нагрузки между разными экземплярами приложения
- Отслеживать потребление ресурсов и их лимиты, чтобы автоматически останавливать и затем перезапускать приложения, потребляющие слишком много ресурсов.
- Перемещать экземпляр приложения с одного хоста на другой, если хосту не хватает ресурсов или если он упал.

– Автоматически задействовать дополнительные ресурсы при добавлении нового хоста в кластер

Чем больше организаций переходит на микросервисные и заточенные под облако архитектуры с контейнеризацией, тем больше растет спрос на надежные и проверенные платформы. Специалисты переходят на Kubernetes по четырем основным причинам:

1. Kubernetes ускоряет работу. С Kubernetes вы получаете платформу как услугу (PaaS) с возможностью самообслуживания и можете создавать на ней уровень аппаратных абстракций для разработчиков. Ваши команды разработчиков смогут быстро и легко запрашивать и получать как необходимые, так и дополнительные ресурсы (например, для обработки дополнительной нагрузки), так как все ресурсы предоставляются из общей инфраструктуры, доступной всем вашим командам.

2. Kubernetes – это экономично. Kubernetes и контейнеры гораздо рациональнее расходуют ресурсы по сравнению с гипервизорами и виртуальными машинами (ВМ); они мало весят и поэтому требуют меньше ресурсов ЦП и памяти.

3. Kubernetes не зависит от конкретных облачных платформ. Kubernetes работает не только на Amazon Web Services (AWS), Microsoft Azure или Google Cloud Platform (GCP), но еще и в локальной инфраструктуре. При переносе рабочих нагрузок не придется перепроектировать приложения или полностью перекраивать инфраструктуру, что позволяет стандартизировать работу на платформе и избежать привязки к конкретному вендору.

4. на радость пользователям.

Основной элемент Kubernetes – это кластер. Кластеры формируются из множества виртуальных или физических машин, каждая из которых выполняет определенную функцию – узла (node) или ведущего узла (master). На каждом узле хранятся группы из одного или нескольких контейнеров

(в которых находятся ваши приложения), а ведущий узел сообщает остальным узлам, когда создавать и удалять контейнеры и как изменить маршрут трафика в зависимости от нового места размещения контейнера.

Ведущий узел Kubernetes – это точка доступа (или панель управления), из которой администратор и другие пользователи взаимодействуют с кластером и управляют развертыванием контейнеров и планировщиком. В кластере всегда есть как минимум один ведущий узел, но их может быть и больше в зависимости от шаблона репликации кластера.

Ведущий узел хранит данные о состоянии и конфигурации для всего кластера в постоянном и распределенном хранилище ключей/значений ectd. У каждого узла есть доступ к хранилищу ectd, и через него узлы узнают, как поддерживать конфигурацию своих контейнеров. Хранилище etcd можно запустить на ведущем узле Kubernetes или в автономной конфигурации.

Ведущие узлы сообщают остальным узлам в кластере через API сервер Kubernetes (kube-apiserver), где находится главная точка доступа к панели управления. Например, kube-apiserver контролирует, чтобы конфигурации в хранилище etcd и в контейнерах, развернутых в кластерах, соответствовали друг другу.

Менеджер контроллеров Kubernetes (kube-controller-manager) взаимодействует с контурами регулирования, которые управляют состоянием кластера, через API сервер Kubernetes. Кроме того, этот сервис управляет контроллерами развертывания, реплик и узлов. Например, контроллер узла регистрирует узел и отслеживает его рабочее состояние в течение всего жизненного цикла.

За отслеживание рабочих нагрузок на узле в кластере и управление ими отвечает планировщик Kubernetes (kube-scheduler). Этот сервис отслеживает производительность и ресурсы узлов и распределяет работу по узлам, исходя из их доступности.

Менеджер облачных контроллеров (cloud-controller-manager) – это один из сервисов в Kubernetes, обеспечивающий независимость от конкретных облачных платформ. Он реализован в виде уровня абстракции, который отделяет API и инструменты поставщика облачных услуг (например, тома хранилища или балансировщики нагрузки) от их визави в Kubernetes.

Все узлы в кластере Kubernetes должны быть настроены в среде выполнения контейнера, как правило, в Docker. Среда выполнения контейнера запускает контейнеры и управляет ими после того, как Kubernetes развернет их на узлах в кластере. Ваши приложения (веб-сервисы, базы данных, серверы API и т.д.) выполняются внутри контейнеров.

Каждый узел Kubernetes запускает процесс-агент, так называемый kubelet, который отвечает за управление состоянием узла (запуск, остановку и поддержание контейнеров приложений), выполняя команды с панели управления. kubelet собирает информацию о производительности и рабочем состоянии с узлов, подов и контейнеров под его управлением и передает ее на панель управления, чтобы затем можно было распределить задачи.

kube-proxy – это сетевой прокси, который выполняется на узлах в кластере и еще работает как балансировщик нагрузки для сервисов, запущенных на узле.

Под (pod) подразумевается базовая единица диспетчеризации, которая состоит из одного или нескольких контейнеров, гарантированно соразмещенных на хост-машине, и может делиться ресурсам. Каждому поду присваивается уникальный в пределах кластера IP-адрес, чтобы приложения не конфликтовали при использовании портов.

Необходимые характеристики контейнеров можно описать в поде через объект, написанный на языке YAML или JSON и называемый Pod Spec. Эти объекты попадут в kubelet через API сервер.

Pod может определять один или несколько томов (volumes), таких как локальный или сетевой диск, а также раскрывать их другим контейнерам в поде – таким образом разные контейнеры делят между собой пространство хранения. Например, тома можно задействовать, когда один контейнер загружает содержимое, а другой выгружает его куда-то еще.

Поскольку контейнеры внутри подов часто эфемерны, Kubernetes предлагает тип балансировщика нагрузки (service), который упрощает отправку запросов группе подов. Сервис целенаправленно отбирает логическое множество подов на основе меток (labels) (подробности ниже). По умолчанию сервисы доступны в пределах конкретного кластера, но вы можете сделать их общедоступными, если хотите, чтобы они получали запросы извне кластера.

Развертывание (deployment) – это YAML-объект , определяющий поды и количество экземпляров контейнеров, так называемых реплик, для каждого пода. Вы задаете необходимое количество реплик для выполнения в кластере через набор реплик (ReplicaSet), который входит в объект развертывания. Так, например, если узел, на котором работал под, падает, то ReplicaSet распределит его нагрузку на другой под, работающий на другом доступном узле.

Набор фоновых процессов (DaemonSet) запускает и выполняет определенные фоновые процессы (в поде) на назначенных вами узлах. Чаще всего они применяются для обслуживания и поддержки подов. Например, с помощью набора фоновых процессов инфраструктура New Relic разворачивает агент инфраструктуры на всех узлах в кластере.

Пространства имен (Namespaces) позволяют создавать виртуальные кластеры поверх физического кластера и предназначены для многопользовательских сред, где пользователи распределены по разным командам или проектам. Они распределяют квоты ресурсов и логически изолируют ресурсы кластера.

Метки (Labels) – это пары ключ/значение, которые можно присвоить подам и другим объектам в Kubernetes. С помощью меток операторы Kubernetes упорядочивают и отбирают подгруппы объектов. Например, во время мониторинга объектов Kubernetes метки помогают быстро найти нужную информацию.

Наборы с сохранением состояния (StatefulSets) позволяют присвоить подам уникальные идентификационные номера, если понадобится переместить поды на другие узлы, поддерживать сетевое взаимодействие или сохранение данных между подами. Аналогично, постоянные тома хранения предоставляют ресурсы хранения кластеру, к которому поды могут запросить доступ при развертывании.

2.2 Развертывание сервера с помощью технологии Docker

Docker является самой популярной платформой использующей технологию контейнеризации. Платформа Docker решает три основные проблемы развертывания сервисов:

- доставка кода на сервер;
- запуск кода;
- единообразие окружения.

Docker позволяет изолировать сервисы от инфраструктуры, таким образом достигается возможность доставлять их гораздо быстрее. Docker позволяет управлять инфраструктурой с помощью тех же принципов, которые используются при управлении приложениями. При использовании инструментов Docker для доставки, тестирования и развертывания, значительно сокращается задержка между фиксацией нового кода в системе контроля версий и запуском его на сервере промышленной эксплуатации.

Docker предоставляет возможность упаковывать и запускать сервисы в изолированном окружении, которое и называется контейнером. Данная изолированность и безопасность позволяет запускать несколько контейнеров на одном хосте одновременно. Контейнеры легковесны поскольку не требуют работы гипервизора. Они запускаются непосредственно на ядре хост машины. Таким образом достигается возможность запуска большего количества контейнеров, чем виртуальных машин, на одном и том же физическом оборудовании. Также, Docker контейнеры можно запускать в самих виртуальных машинах.

Docker предоставляет следующие инструменты и платформу для управления жизненным циклом контейнеров. С помощью Docker происходит упаковка приложения и их компонентов в контейнер. В экосистеме Docker, контейнер становится атомарным юнитом для распространения и тестирования приложения. После разработки, приложение может быть развернуто на сервере промышленной эксплуатации вручную или с помощью оркестровщика контейнеров. Данная техника развертывания единообразна независимо от того где разворачивается приложение на промышленную эксплуатацию, будь то в локальном дата центре, облачном провайдере или в гибридном окружении.

Docker оптимизирует жизненный цикл разработки, позволяя разработчикам работать в стандартизированном окружении. При использовании докер, окружение при разработке ничем не отличается от окружения в промышленной эксплуатации.

Контейнеры идеально подходят для непрерывной интеграции или непрерывной поставки. Обычный сценарий разработки программного обеспечения сводится к получению некоего артефакта, который строится после любого изменения исходного кода приложения. При использовании докер, этим артефактом может быть контейнер. Разработчики или инженеры по тестированию используют докер контейнеры для развертывания приложений на

тестовое окружение и запускают автоматические или ручные тесты. Если во время проверок обнаруживается баг, данный контейнер может быть развернут в окружении разработки, исправлен и перенесен обратно в тестовое окружение, для повторной проверки. Если все проверки успешно завершены достаточно доставить контейнер с исправлениями в промышленное окружение.

Контейнерная платформа Docker обеспечивает высокую переносимость. Контейнер может выполняться на рабочей станции разработчика, физических или виртуальных машинах в дата центре, в инфраструктуре облачного провайдера. Переносимость докера, а также его легковесность позволяют динамически менять загрузку, увеличивая или уменьшая количество приложений и сервисов, практически в реальном времени.

Docker легковесный и быстрый. Он предоставляет жизнеспособную, экономически выгодную альтернативу виртуальным машинам на основе гипервизора. С помощью докер можно использовать больше вычислительных мощностей, поскольку эти ресурсы не тратятся на обслуживание гипервизора.

Docker использует клиент-серверную архитектуру. Docker клиент взаимодействует с фоновым процессом – сервером Docker, который в свою очередь запускает контейнеры. Клиент и фоновый процесс могут выполняться в одной операционной системе, также есть возможность подключить клиента к удаленному фоновому процессу. Докер клиент и фоновый процесс взаимодействуют через REST API поверх сокетов или через сетевой интерфейс. Фоновый процесс Docker (dockerd) принимает запросы и управляет объектами Docker.

Фоновый процесс может также взаимодействовать с другими фоновыми процессам. Фоновый процесс управляет следующими объектами:

– образы;
– контейнеры;
– сетевое взаимодействие;

– тома.

Docker клиент основной способ взаимодействия с сервером. При выполнении команды клиент отправляет эту команду фоновому процессу, который в свою очередь ее выполняет. Docker клиент может взаимодействовать с множеством различных серверов.

Реестр контейнеров хранит образы. Данный реестр может быть, как публичным, так и приватным. Docker Hub является официальным открытым реестром, в котором хранятся официальные образы от различных провайдеров технологий, например, ubuntu или nginx. Добавлять образы в публичный реестр может любой желающий. По умолчанию Docker настроен на поиск образов в этом публичном реестре. Также имеется возможность создавать приватные реестры образов.

При использовании команд docker pull или docker run, необходимые образы будут скачаны из сконфигурированного реестра. При использовании команды docker push, указанный образ будет помещен в реестр.

При использовании докер создаются и используются образы, контейнеры, сетевое взаимодействие, тома, плагины и другие компоненты. Образ – это неизменяемый шаблон, содержащий инструкции для создания контейнера. В большинстве случаем образ базируется на другом образе, дополняя его новой функциональностью. Например, можно построить образ, который базируется на образе ОС Linux Ubuntu, расширив его установив веб-сервер Apache с разработанным приложением, а также сконфигурировав сервер для правильной работы приложения.

Имеется возможность создавать новые образы или же повторно использовать уже созданные и размещенные в реестре. Чтобы создать новый образ, требуется создать конфигурационный файл, который называется DockerFile. Данный файл содержит инструкции для создания и запуска образа. Каждая инструкция в конфигурационном файле создает слой в образе. Когда

конфигурационный файл меняется и запущен процесс повторной сборки образа, только те слои, которые изменились будут пересобраны. Именно эта технология делает контейнеры легковесными и быстрыми по сравнению с другими технологиями виртуализации.

Контейнер – это запущенный экземпляр образа. Контейнеры можно создавать, запускать, останавливать и удалять, используя Docker API или интерфейс командной строки. Контейнер может быть подключен к одной или несколько сетям. К нему может быть смонтирована часть файловой системы хост машины. Также имеется возможность создать новый образ из текущего состояния контейнера.

По умолчанию, контейнеры относительно хорошо изолированы от других контейнеров и хост машины. Имеется возможность управлять изоляцией сетевого взаимодействия контейнеров и подключёнными томами между другими контейнерами и хостовой системой. Контейнер создается из образа, а также содержит все конфигурационные параметры, которые предоставляются при его создании и запуске. При остановке контейнера, все изменения в его состоянии, которые не зафиксированы в постоянном хранилище пропадают.

2.3 Grafana как инструмент для визуализации метрик мониторинга

Grafana – универсальная обертка для работы с аналитическими данными, которые хранятся в разных источниках. Она сама ничего не хранит и не собирает, а является лишь универсальным клиентом для систем хранения метрик. Например, с помощью нее можно ходить за цифрами как в традиционную базу PostgreSQL, так и в специализированные аналитические системы типа Prometheus или Influx.

Grafana можно подключать к любому хранилищу статистических данных. Разные отделы компании могут использовать разные СУБД и системы сбора

статистики. Так вот, Grafana умеет работать с любой популярной системой хранения данных. Конечно, делает она это не сама – первоначальную настройку и подключение к СУБД выполняют администраторы. Но на этом их работа заканчивается – дальше аналитики могут самостоятельно строить свои запросы.

Grafana может быть подключена к хранилищу и выполнять там определенные запросы. Запросы конструируются аналитиками в специальном удобном интерфейсе, помогающем сосредоточиться именно на данных, а не на правильности написания запросов в СУБД. Полученные результаты Grafana показывает в доступном виде. Это могут быть как простые таблицы, так и графики, распределения и десятки других форматов отображения данных.

Запросы отрисовываются на графиках, в таблицах или выводятся напрямую в абсолютных значениях. Сами отображения можно группировать между собой и собирать в интерактивные дашборды.

ЗАКЛЮЧЕНИЕ

В современном мире при разработке программного обеспечения все чаще и чаще возникает ситуация, в которой требования к системе меняются уже на конечных этапах разработки. К тому же, наблюдается постоянный прирост функциональности в уже существующих системах. Помимо этого, к современному ПО предъявляются крайне высокие требования по производительности и отказоустойчивости.

Классическая монолитная архитектура приложений не отвечала требованиям современного мира, и на смену ей пришел сервисно-ориентированный подход. Такой подход имеет значительные преимущества по сравнению с монолитными приложениями, такие, например, как лучшая масштабируемость, меньшая связанность между модулями, лучший контроль на этапах разработки, тестирования и развертывания.

Системы, построенные с помощью сервисно-ориентированой архитектуры, также являются распределенными. Взаимодействие между сервисами осуществляется с использованием различных протоколов.

На данный момент в РЦ ВЦ СПбГУ ведутся удаленные работы по настройке серверов для дальнейшего развертывания контейнеризированных сред и инструмента оркестровки контейнерами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Martin L. Abbott. The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise. – Addison-Wesley Professional, 2015. – 624 с.

2. Хабрахабр, Микросервисы (Microservices). – https://habrahabr.ru/post/249183/

3. Martin Fowler, Microservices. – https://martinfowler.com/articles/microservices.html

4. Vikram Murugesan, Microservices Deployment Cookbook. – Packt Publishing, 2017. – 378 с.

5. Sam Newman. Building Microservices: Designing Fine-Grained Systems. – O'Reilly Media, 2015. – 280 с.

6. Sourabh Sharma, Rajesh RV, David Gonzalez. Microservices: Building Scalable Software. – Packt Publishing Limited, 2017. – 919 с.

7. Rajesh RV, Spring Microservices. – Packt Publishing, 2016. – 436 с.

8. Pethuru Raj. Docker: Creating Structured Containers. – Packt Publishing, 2016. – 320 с.

9. Docker, Docker Documentation. – https://docs.docker.com/

10. Randall Smith. Docker Orchestration. – Packt Publishing, 2017. – 284 с.

11. Kubernetes, Kubernetes Documentation. https://kubernetes.io/docs/home/

12. Docker, Swarm mode overview. – https://docs.docker.com/engine/swarm/

13. Fabrizio Soppelsa, Chanwit Kaewkasi. Native Docker Clustering with Swarm. – Packt Publishing, 2016. – 280 с.

14. Хабрахабр, Основы Kubernetes. – https://habrahabr.ru/post/258443/

15. Platform9, Container Orchestration Tools: Compare Kubernetes vs Docker Swarm. – https://platform9.com/blog/compare-kubernetes-vs-docker-swarm/