МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование института полностью)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Моделирование процесса развертывания микросервисных приложений с использованием облачной инфраструктуры

Студент А.С. Ефремов
(И.О. Фамилия) (личная подпись)

Руководитель канд. пед. наук, доцент, О.М. Гущина
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема выпускной квалификационной работы – «Моделирование процесса развертывания микросервисных приложений с использованием облачной инфраструктуры».

Процесс первоначальной настройки сервера является трудоемкой задачей, которую можно упростить с помощью автоматически выполняемых скриптов.

Цель работы – разработка проекта моделирования процесса развертывания микросервисных приложений с использованием облачной инфраструктуры.

Объектом исследования бакалаврской работы является облачная инфраструктура.

Предметом исследования выпускной квалификационной работы является подготовка облачной инфраструктуры к развертыванию приложений на стороне сервера.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- проанализировать инструменты, решающие задачу развертывания клиентских приложений на сервере;
- разработать модель автоматизации развертывания облачного окружения;
- определить компоненты, использующие автоматизацию развертывания клиентских приложений в микросервисной архитектуре;
- определить средства выполнения пользовательских сценариев;
- разработать схемы взаимодействия с операционной системой и развернутыми автоматически приложениями для дальнейшего тестирования корректности установки;

Выпускная работа состоит из 2 таблиц, 36 рисунков и 48 страниц.

Abstract

The title of the graduation work is *Modeling the process of deploying microservice applications using a cloud infrastructure*.

The research is also related to a cloud platform.

The goal of the work is to give some information about the microservice architecture in a cloud platform.

The object of the graduation work is cloud architecture using microservices.

The subject of the investigation is preparation of cloud infrastructure to deploy applications on servers.

The issues of continuous integration, continuous delivery and orchestration server installation are highlighted in the work's general part.

To choose a method for automating server deployment, two alternatives are considered, and the one that is more independent on the modules is selected on the basis of the analysis.

In these environments, it is proposed to create a space for calculating the tasks assigned to the control tools to develop and integrate modules, as well as to update the server applications.

When choosing software, Openshift, Ansible, and Jenkins are preferred.

The research also deals with the deployment project and configuration of settings.

The special part of the graduation work gives full coverage to the microservices application technology on the basis of the cloud infrastructure conditions.

The development is provided by software for tracking the changes in any set of files and by an orchestration tool.

The investigation explains the effectiveness of the decisions made as well.

The technology used corresponds to the high availability regulations.

Оглавление

Введение

В настоящее время рынок облачных решений растет очень быстро, становится сложно предсказать темп его роста. Аналитические компании фиксируют ежегодно одни и те же тенденции развития: рост расходов на облачные вычисления, развитие рынка сервисов, дата-центров и увеличение трафика в облачных системах. В 2020 году был зафиксирован рост рынка публичных облаков на 11% по сравнению с 2019, что составляет около 270 миллиардов долларов.

Облачные технологии используются в различных сферах, например:

- в банковской сфере для вычислений, необходимых для построения различных отчетов в режиме реального времени;
- в расчетных организациях для автоматизированных систем расчета или биллинга;
- в телекоммуникационных компаниях для обеспечения возможности передачи сигнала;
- в поисковых системах для предсказания запросов пользователей.

Заказчиком данной работы выступил ООО «НетКрэкер» – компания, специализирующаяся на создании, внедрении и сопровождении систем эксплуатационной поддержки (OSS) для операторов связи, крупных предприятий и государственных учреждений. Компания преимущественно работает с зарубежными клиентами, от клиентов часто поступают задания по внедрению и сопровождению серверов, предоставляющих облачные технологии для вычислений. Процесс развертывания сервера включает:

- настройку виртуального окружения;
- установку операционной системы;
- настройку серверного подключения;
- настройку взаимодействия с базой или базами данных;
- настройку связующего программного обеспечения и промежуточных обработчиков;

- установку и настройку пользовательских приложений.

Таким образом, процесс первоначальной настройки сервера является трудоемкой задачей, которую можно упростить с помощью автоматически выполняемых скриптов. Скрипт-сценарий, описывающий последовательность команд, которые необходимо выполнить для достижения той или иной задачи. В нашем случае – развертывание сервера.

Как видим, работа, связанная с облачными технологиями, является актуальной в силу повсеместного их использования и роста количества задач от реальных клиентов компании «НетКрэкер».

Цель данной работы – разработка проекта моделирования процесса развертывания микросервисных приложений с использованием облачной инфраструктуры. Для достижения поставленной цели необходимо выполнить следующие задачи:

- проанализировать инструменты, решающие задачу развертывания клиентских приложений на сервере;
- разработать схему автоматизации развертывания облачного окружения;
- определить компоненты, использующие автоматизацию развертывания клиентских приложений в микросервисной архитектуре;
- определить средства выполнения пользовательских сценариев;
- разработать схемы взаимодействия с операционной системой и развернутыми автоматически приложениями для дальнейшего тестирования корректности установки;
- протестировать корректность развертывания с помощью схем взаимодействия.

В первой главе будет рассмотрен процесс развертывания сервера для облачных вычислений в ручном режиме, в рамках данной бакалаврской работы будет рассмотрен вариант автоматизации первоначальной настройки окружения. Затем будет проведен анализ на преимущества и недостатки методов монолитной и микросервисной архитектуры. Далее будет описано

техническое задание для проекта моделирование процесса развертывания микросервисных приложений с использованием облачной инфраструктуры, также будет приведено описание используемых технологий в разработке согласно техническому заданию.

Во второй главе будет описано проектирование схемы инструментария, а также составляющие инструментария, взаимодействие микросервисов с базой данных. Затем в данном параграфе будет описан уровень проектирования доступа к данным. Далее будет описан слой интеграции.

В третьей главе будет выполнена базовая настройка виртуальной машины для работы с облачными технологиями. Затем будет выполнена установка Jenkins, будет создан hosts файл с описание переменных окружения. Далее будет проведена установка и настройка микросервисного приложения. Затем будут проведены предварительные и опытно-промышленные испытания.

В процессе выполнения выпускной квалификационной работы продемонстрировано обладание такими компетенциями, как готовность использовать навыки проектирования, реализации и анализа эффективности программного обеспечения для решения задач в различных предметных областях, готовность к созданию сценариев и автоматизации их.

Глава 1 Характеристика объекта автоматизации

1.1 Описание объекта автоматизации

Заказчиком инструмента для моделирования процесса развертывания микросервисных приложений с использованием облачной инфраструктуры является общество с ограниченной ответственностью «НетКрэкер», являющееся подразделением международной компании Netcracker Technology. Компания специализируется на создании, внедрении и сопровождении различных информационных систем. В том числе она занимается и поддержкой серверов, работа Netcracker с серверами включает в себя развертывание окружения, настройку кластеров серверов, работу с обращениями клиентов. Логотип компании представлен на рисунке 1.

Рисунок 1 – Логотип Netcracker Technology

Процесс работы с проектной задачей первоначальной работой сервера выглядит следующим образом:

- клиент обращается в компанию с задачей первоначальной настройки сервера для облачных вычислений;
- бизнес-аналитик уточняет условия, при которых будет использоваться серверное оборудование, затем он составляет вводные требования к проекту;
- затем к работе над задачей подключается технический менеджер, по описанным бизнес-процессам строит модель работы для дальнейшей работы технических специалистов, а проектный менеджер анализирует техническое задание и принимает решение о том, что берется ли он за задачу;

- проектный менеджер подбирает себе команду с учетом технических требований проекта и бюджета;
- рабочая группа приступает к выполнению задачи: знакомится с техническим заданием и планом работы, согласно разбивке на подзадачи, которые выполнил проектный менеджер, каждый приступает к реализации к поставленной ему подзадаче;
- рабочая группа завершает выполнение задачи и передает выполненный проект для тестирования;
- тестировщики согласно требованиям технического задания разрабатывают тестировочные сценарии и выполняют по ним тестирование;
- далее происходит сдача готового проекта клиенту;
- затем проект переходит в стадию поддержки если клиент согласен на обслуживание от Netcracker.

Сам процесс первоначальной настройки сервера для it-инженера выглядит следующим образом:

- ознакомление со структурой проекта и взаимодействия данных приложения;
- настройка пользовательского окружения для установки первоначального программного обеспечения;
- изменение конфигурационных параметров для установки окружения проекта;
- установка окружения и подготовка приложений;
- установка приложений.

Детализировано развертывание сервера с точки зрения архитектуры выглядит так: запускается мастер-сервер или DVM. DVM представляет собой сервер с установленным оркестратором непрерывной интеграции и инструментами удаленного выполнения сценариев. Структура DVM сервера представлена на рисунке 2.

DVM сервер состоит из start service to start all and install patch scripts – это программное обеспечение и патчи необходимое для установки

дальнейших компонентов. Jenkins – программная система с открытым исходным кодом на Java, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения. Позволяет автоматизировать часть процесса разработки программного обеспечения, в котором не обязательно участие человека, обеспечивая функции непрерывной интеграции. Работает в сервлет-контейнере, например, Apache Tomcat. Поддерживает инструменты системы управления версиями, включая AccuRev, CVS, Subversion, Git, Mercurial, Perforce, Clearcase и RTC. Может собирать проекты с использованием Apache Ant и Apache Maven, а также выполнять произвольные сценарии оболочки и пакетные файлы Windows.

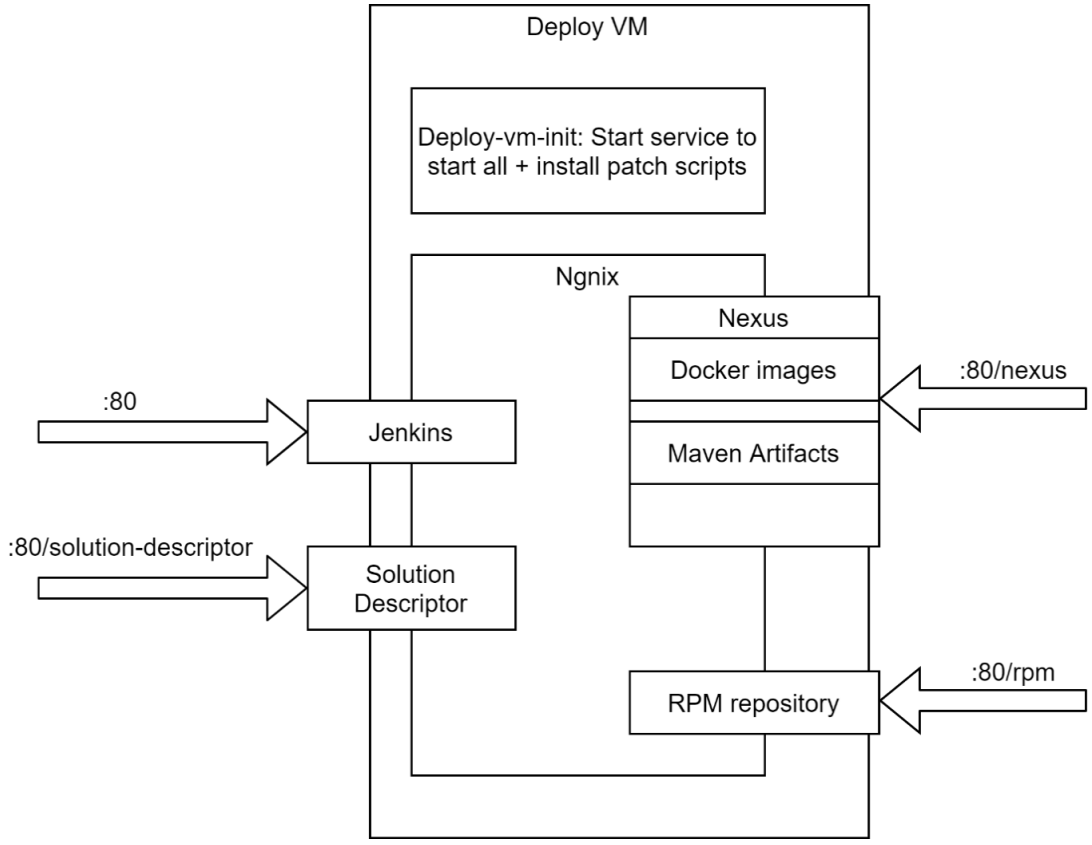


Рисунок 2 – Структура DVM

Сборка может быть запущена разными способами, например, по событию фиксации изменений в системе управления версиями, по расписанию, по запросу на определённый URL, после завершения другой

сборки в очереди. Возможности Jenkins можно расширять с помощью плагинов. Пример отображения установки Jenkins показан на рисунке 3.

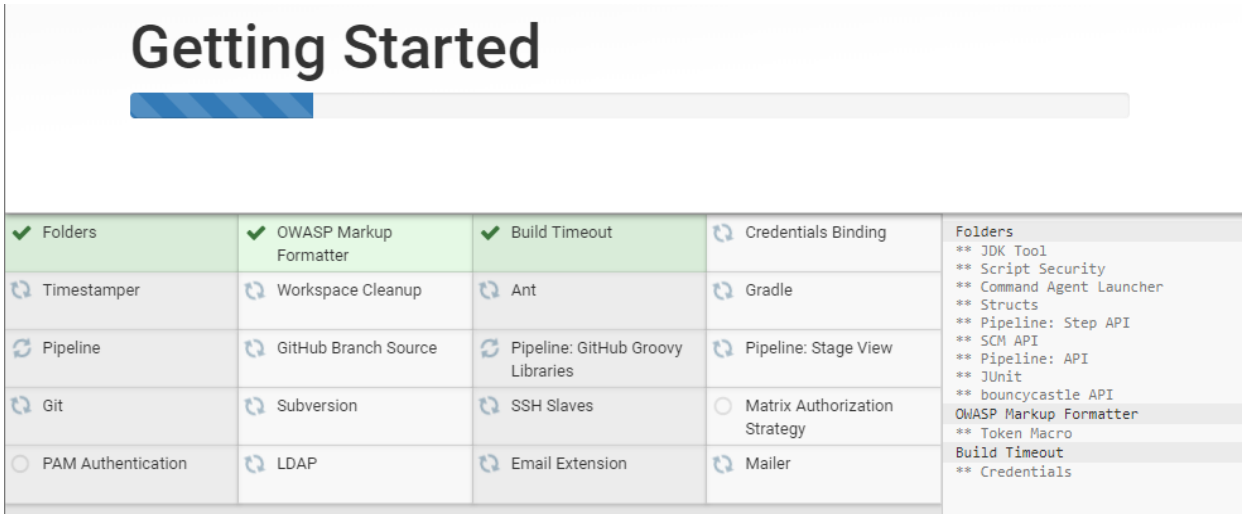


Рисунок 3 – Пример установки Jenkins

На рисунке 2 solution description – это описание устройства приложения. RPM repository – это подключение к серверу удаленного хранения данных. Nexus – интегрированная платформа, с помощью которой разработчики могут проксировать, хранить и управлять зависимостями Java (Maven), образами Docker, Python, Ruby, NPM, Bower, RPM-пакетами, gitlfs, Apt, Go, Nuget, а также распространять свое программное обеспечение. Nginx – это HTTP-сервер и обратный прокси-сервер. Основная функциональность:

- обслуживание неизменяемых запросов, индексных файлов, автоматическое создание списка файлов, кэш дескрипторов открытых файлов;
- акселерированное проксирование без кэширования, простое распределение нагрузки и отказоустойчивость;
- поддержка кеширования при акселерированном проксировании и FastCGI;
- акселерированная поддержка FastCGI и memcached-серверов, простое распределение нагрузки и отказоустойчивость;

- модульность, фильтры, в том числе сжатие (gzip), byte-ranges (докачка), chunked-ответы, HTTP-аутентификация, SSI-фильтр;
- несколько подзапросов на одной странице, обрабатываемые в SSI-фильтре через прокси или FastCGI, выполняются параллельно;
- поддержка SSL;
- поддержка PSGI, WSGI;
- экспериментальная поддержка встроенного Perl.

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений. В Docker все основано на изображениях. Docker images представляет собой комбинацию файловой системы и параметров.

Maven – инструмент для автоматизации сборки проектов. Maven артефакт является результатом вывода сборки maven, как правило, jar или war или другого исполняемого файла. Maven Artifacts идентифицируются с помощью системы координат groupId, artifactId и версии. Maven использует groupId, artifactId и version для определения зависимостей, необходимых для создания и запуска кода.

Таким образом, в данном параграфе был описан объект автоматизации, рассмотрен процесс выполнения задачи клиента от момента её возникновения до решения и её передачи окружения на сопровождение. Также в данном параграфе был рассмотрен процесс развертывания сервера для облачных вычислений в ручном режиме, в рамках данной бакалаврской работы будет рассмотрен вариант автоматизации первоначальной настройки окружения.

1.2 Анализ возможных проектных решений

Существуют разработанные решения для автоматизации развертывания сервера, используемого для облачных вычислений. В данном параграфе

описаны эти решения и проведен сравнительный анализ. Существующие решения чаще всего используются в организациях в которых они были разработаны и не передаются в открытый доступ. Однако основной алгоритм их работы без деталей конструкции можно найти в открытых источниках и сравнить их между собой.

Одним из таких решений является последовательная установка компонентов сервера с применением технологий оркестраторов и инструментов управление конфигурацией топологий push и pull. Оркестровка – автоматическое размещение, координация и управление сложными компьютерными системами и службами. Оркестровка описывает то, как сервисы должны взаимодействовать между собой, используя для этого обмен сообщениями и последовательность действий.

Управление конфигурациями выделяет две топологии push и pull. В pull топологии клиентские машины устанавливают приложения для централизованного обновления. В этой системе клиенты обращаются к серверу не зависимо друг от друга. В push топологии клиентские машины не требуют установки дополнительного программного обеспечения. Установка и выполнения сценариев поступает с сервера на клиентские машины по списку подключений что способствует масштабированию подключений и централизованной системе работы серверов.

Следующий метод решения исходной задачи базируется на монолитной архитектуре сервера. В данном решении используются инструменты базового развертывания приложения в рабочем окружении. На рисунке 4 структура монолитного сервера.

Kernel Space – пространство ядра операционной системы. Fundamental process management – основные концепции. Каждый процесс запускает одну программу и изначально имеет один поток управления. I/O and device managment – управление и устройствами ввода-вывода. Inter process communication – межпроцессное взаимодействие, обмен данными между потоками одного или разных процессов. Реализуется посредством

механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС. File system – порядок, определяющий способ организации, хранения и именования данных. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов.

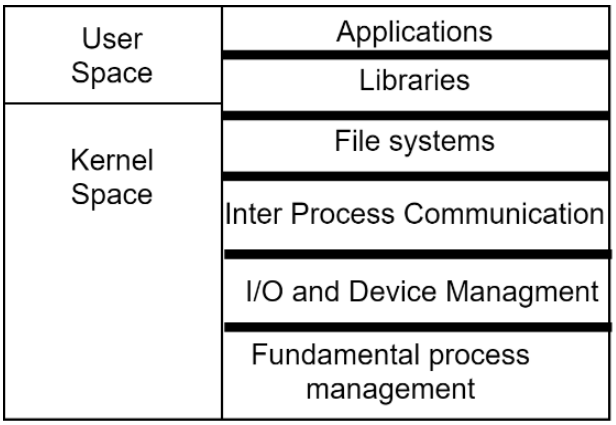


Рисунок 4 – Структура монолитного сервера

User space – адресное пространство виртуальной памяти операционной системы, отводимое для пользовательских программ. Libraries – библиотеки для работы приложений. Applications – приложения пользователя на сервере.

Перейдем к сравнению двух описанных технологий решения задачи развертывания сервера для облачных вычислений. Перед этим введем и опишем понятие «микросервисная архитектура».

В общем случае микросервисная архитектура – подход к разработке приложения, в котором приложение рассматривается, как набор небольших сервисов, каждый из которых работает в своем собственном окружением и взаимодействует другими механизмами приложения посредством http-запросов через API-подключение. Такие сервисы построены на бизнес-возможностях и могут независимо использоваться с помощью полностью автоматизированного оборудования для развертывания. Для управления этими сервисами существуют централизованные системы управления

микросервисными службами. Они могут быть написаны на системных языках программирования и использовать разные технологии хранения данных.

Теперь перейдем к сравнению существующих методов развертывания окружения для облачных вычислений в автоматизированном режиме.

При использовании решения основывающегося на монолитной архитектуре преимущества заключаются в простоте развертывания. К недостаткам данного подхода относится сложность поддержки. Также нет возможности изоляции модулей, обновление архитектуры затруднено в виду сложности сохранения ссылочной целостности. Придерживаясь решения основывающегося на микросервисной архитектуре преимущества заключаются в использовании модулей для написания, это облегчает процесс дальнейшей поддержки. Возможность запуска и развертывания микросервисов независимо друг от друга. Высокая отказоустойчивость является главным преимуществом. Недостатком метода является то, что транзакции и построение проекта усложняется. По результатам сравнения для разработки автоматизированного метода развертывания сервера для облачных вычислений можно базироваться на методе микросервисной архитектуры стараясь избежать выявленных недостатков.

Таким образом, в данном параграфе были рассмотрены существующие решения задачи автоматизированного развертывания сервера, который используется для облачных вычислений. Такими решениями является метод, базирующийся на монолитной архитектуре, и метод, базирующийся на микросервисной архитектуре.

Для данных методов был выполнен анализ на их преимущества и на их недостатки. По результатам проведенного анализа в качестве метода, на который будем полагаться для данной бакалаврской работы, был выбран метод, основанный на микросервисносной архитектуре.

1.3 Техническое задание на разработку

В приложении А представлено разработанное техническое задание для данной бакалаврской работы, в ходе которой была реализована задача, поставленная ООО «НетКрэкер» в лице отдела инженерии. Рассмотрим подробнее требования, изложенные в техническом задании.

Требования к инструментарию выглядят следующим образом:

- название виртуальной машины;
- операционная система;
- набор устанавливаемых приложений;
- количество требуемых процессоров;
- количество минимальной памяти на жестком диске в гигабайтах.

Технические требования для разрабатываемого инструментария представлены в таблице 1.

Таблица 1 – Технические требования

Название виртуальных машин	Выбранная операционная система	Приложения	Количество требуемых процессоров	Количество минимальной оперативной памяти в гигабайтах	Количество минимальной памяти на жестком диске в гигабайтах
DVM	CentOS 7	DNS, NTP, Ansible, Jenkins	1	1	20
OKD	CentOS 7	OpenShift (master, infra, compute) Microservices Demo	2	6	20

Для виртуальных машин были заданы характеристики и доменные имена виртуальных машин. В таблице 2 представлены характеристики интернет-протокола четвертой версии и доменные имена.

Для инструментария в обязательном порядке использовать протокол сетевого времени europe.pool.ntp.org с часовым поясом GMT+4.

Для понимания работы, которую необходимо выполнить согласно техническому заданию, опишем некоторые понятия, изложенные выше.

DNS (Domain Name System «система доменных имен») – компьютерная распределенная система для получения информации о доменах. Данный инструмент используется в качестве средства преобразования доменных имен в IP-адреса в момент отправки запроса на сервер.

Таблица 2 – Параметры виртуального окружения

Название виртуальных машин	Доменное имя виртуальных машин	IPv4 (внешний) виртуальных машин	IPv4(внутренний) виртуальных машин
DVM	DVM	ip: 10.0.2.100/24 gw: 10.0.2.1 dns: 10.0.2.1	ip: 192.168.56.100 gw: none dns: 192.168.56.100
OKD	OKD	ip: 10.0.2.150/24 gw: 10.0.2.1 dns: 10.0.2.1	ip: 192.168.56.150 gw: none dns: 192.168.56.100

Для получения результата от веб-страницы компьютер посылает запрос на сервер доменных имен. Если найдена информация, соответствующая запросу, сервер отвечает IP-адресом сайта, к которому была осуществлена попытка получить доступ. Условный вид IP-адреса: 123.123.123.123. Пример показан на рисунке 5.

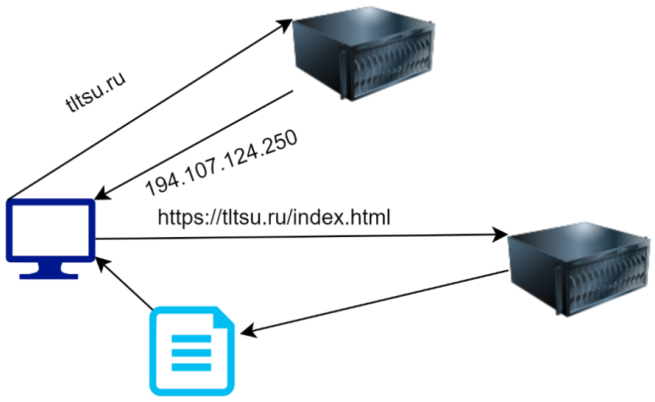


Рисунок 5 – Схема получения IP-адреса

В данном примере клиент отправляет запрос на сервер. Сервер находит страницу по запросу пользователя и отправляет страницу клиенту.

NTP (англ. Network Time Protocol – протокол сетевого времени) – сетевой протокол для синхронизации внутренних часов компьютера с использованием сетей с переменной латентностью.

Ansible – система управления конфигурациями. Она используется для автоматизации настройки и развертывания программного обеспечения. Наряду с Chef, Puppet и SaltStack считается одной из наиболее популярных систем управления конфигурациями для Linux. Главное отличие Ansible от аналогов заключается в том, что она не требует установки агента или клиента на целевые системы. Пользователь Ansible создаёт определённые Playbook в формате YAML с описанием требуемых состояний управляемой системы. Playbook – это описание состояния ресурсов системы, в котором она должна находиться в конкретный момент времени, включая установленные пакеты, запущенные службы, созданные файлы и многое другое. Пример Playbook показан на рисунке 6.

```
---
- name: install and start apache
  hosts: web
  become: yes
  vars:
    http_port: 80

  tasks:
  - name: httpd package is present
    yum:
      name: httpd
      state: latest
```

Рисунок 6 – Пример playbook файла

Ansible проверяет, что каждый из ресурсов системы находится в ожидаемом состоянии и пытается исправить состояние ресурса, если оно не

соответствует ожидаемому. Для выполнения задач используется система модулей. Каждая задача представляет собой ее имя, используемый модуль и список параметров, характеризующих данную задачу. Пример Inventory файла показан на рисунке 7.

Jenkins – программная система с открытым исходным кодом на Java, предназначенная для обеспечения процесса непрерывной интеграции программного обеспечения. Позволяет автоматизировать часть процесса разработки программного обеспечения, в котором не обязательно участие человека, обеспечивая функции непрерывной интеграции. Сборка может быть запущена разными способами, например, по событию фиксации изменений в системе управления версиями, по расписанию, по запросу на определённый URL, после завершения другой сборки в очереди. Возможности Jenkins можно расширять с помощью плагинов.

```
[control]
control ansible_host=10.42.0.2

[web]
node-[1:3] ansible_host=10.42.0.[6:8]

[haproxy]
haproxy ansible_host=10.42.0.100

[all:vars]  ansible_user=vagrant
ansible_ssh_private_key_file=~/.vagrant.d/insecure_private_key
```

Рисунок 7 – Пример Inventory файла

OpenShift – это облачная платформа с открытым исходным кодом, используемая для создания, тестирования и запуска приложений и их развертывания в облаке. OpenShift способен управлять приложениями, написанными на разных языках программирования, таких как Node.js, Ruby, Python, Perl и Java. Одной из ключевых особенностей OpenShift является его расширяемость, которая помогает пользователям поддерживать приложение, написанное на других языках.

Таким образом в данном параграфе было описано составленное техническое задание для проекта автоматизации подготовки облачной инфраструктуры для развертывания микросервисных приложений, также было приведено описание используемых технологий в разработке согласно техническому заданию.

Глава 2 Проектирование архитектуры облачной инфраструктуры для развертывания микросервисных приложений

2.1 Проектирование схемы инструментария для автоматической установки облачного окружения

Инструментарий – совокупность инструментов, с помощью которых достигается некая поставленная задача. В контексте данной работы инструментарий – совокупность сценариев, назначение которых – автоматизация процесса развертывания сервера с приложениями, построенными на микросервисной архитектуре. Инструментарий разрабатывается для клиент-серверной архитектуры, сама архитектура представлена в приложении В.

Процесс настройки DVM и OKD является основным составляющим процесса первоначальной настройки сервера и его компонентов. Данный процесс можно описать, используя нотацию BPMN. Эта нотация является универсальной для описания процессов работы. Декомпозируемая задача основного процесса работы представлена на рисунке 8.

Рисунок 8 – Декомпозированная задача подготовки

Вышеупомянутый процесс включает в себя следующие подпроцессы:

- произвести базовую настройку VM;
- настоить Ansible и Jenkins;
- настроить DNS-сервера;
- создать задачи в Jenkins;

- установить OpenShift;
- установить демонстрационное приложение Microservices Demo.

Декомпозиция процесса представлена на рисунке 9.

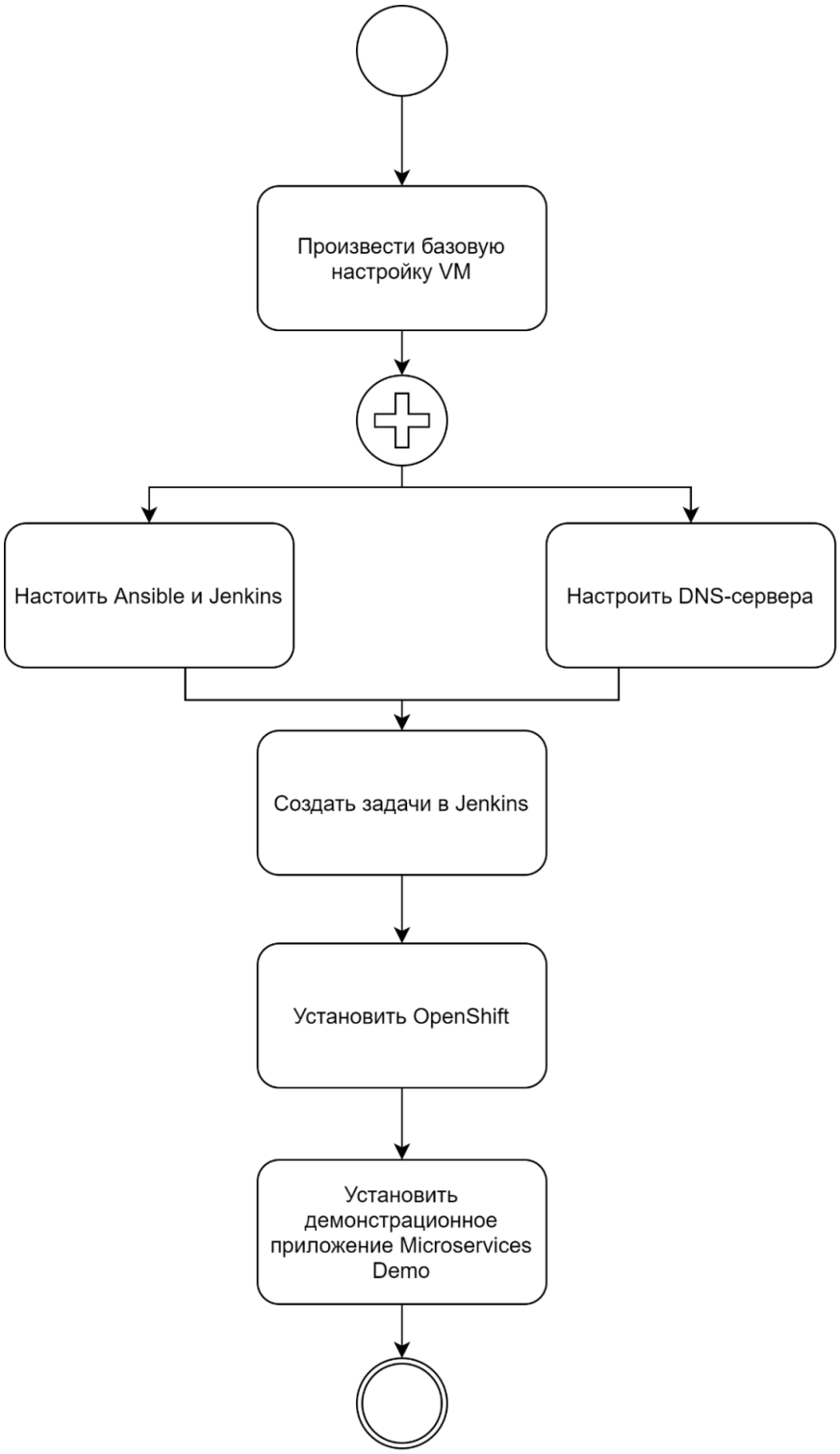


Рисунок 9 – Декомпозиция процесса подготовки и настройки

Схема взаимодействия DVM и OKD представлена на рисунке 10.

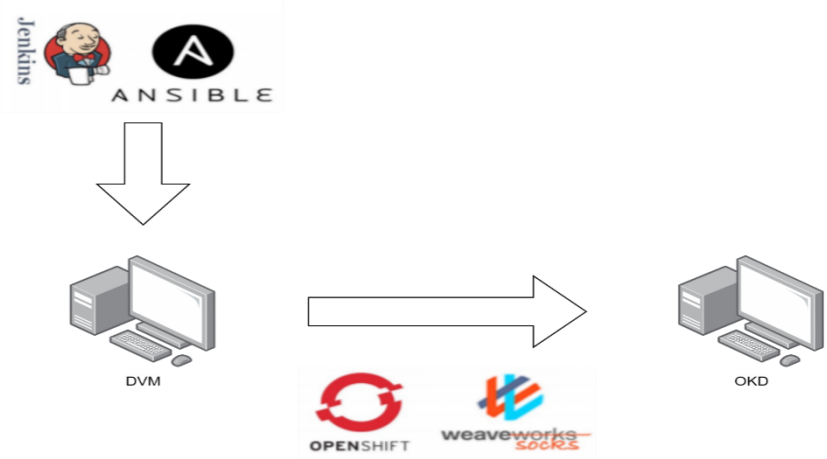


Рисунок 10 – Схема взаимодействия DVM и OKD

DVM ориентирован на технологиях оркестратора Jenkins и системе управления конфигураций Ansible. OKD несет в себе облачную платформу OpenShift и демонстрационное приложение «weavesocks».

Инструментарий должен включать в себя инструмент непрерывной инттеграции за основу взят оркестратор Jenkins. Для выполнения сценариев и отслеживания статуса конфигураций выбран Ansible. С помощью инструмента управления конфигурациями на OKD должен будет установлена облачная платформа OpenShift для запуска, тестирования, и установки пользовательского приложения «weavesocks».

Принцип автоматизации состоит в последовательном запуске следующих скриптов:

- установка и настройка Jenkins;
- создание пользователя OpenShift;
- запуск скрипта установки микросервисного приложения «weavesocks».

Скрипт получает список виртуальных машин с заданными кофигурациями, а также получает переменные окружения системы. Структура конфигурационного файла выглядит следующим образом:

доменное имея систем, группы и роли виртуальных машин. Затем ключевые слова и переменные групп. Используя методы функционального моделирования, опишем процесс настройки сервера. Перед тем как перейти непосредственно к моделированию опишем последовательность действий при развёртывании окружения. Для первоначальной настройки сервера с учётом требований клиента необходимо выполнить следующее:

– получить требования для развёртывания серверного окружения;
– установить базовое ПО;
– настроить платформу развёртывания приложений Weblogic;
– установить пользовательские приложения;
– актуализировать конфигурации приложений для взаимодействия с другими приложения в системе посредством патчей;
– протестировать функционал окружения;
– перенести конфигурацию на клиентский сервер.

Функциональная модель получения требований для развертывания серверного окружения в нотации IDEF0 представлена на рисунке 11.

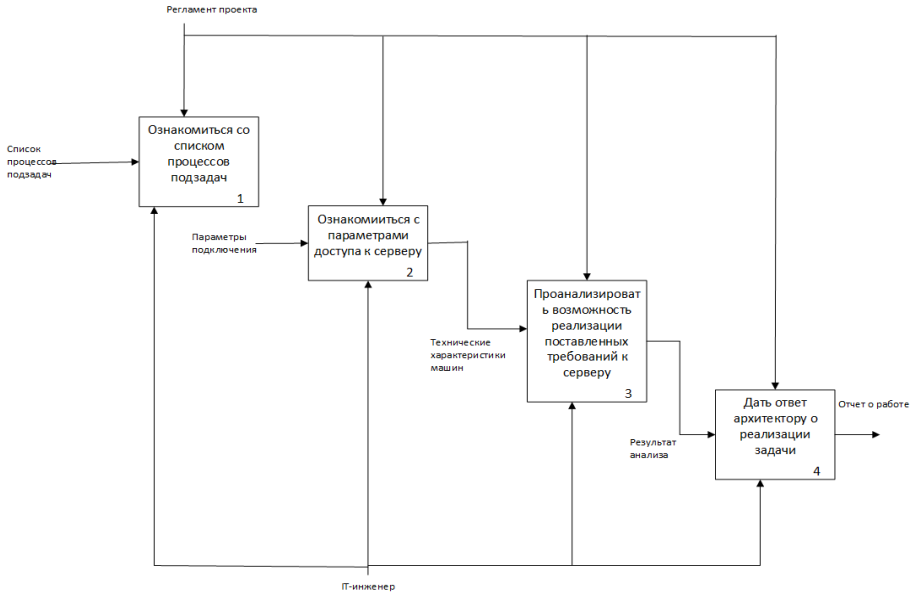


Рисунок 11 – Функциональная модель получения требований для развертывания серверного окружения

Функциональная модель установки базового программного обеспечения, необходимого для функционирования системы, представлена на рисунке 12.

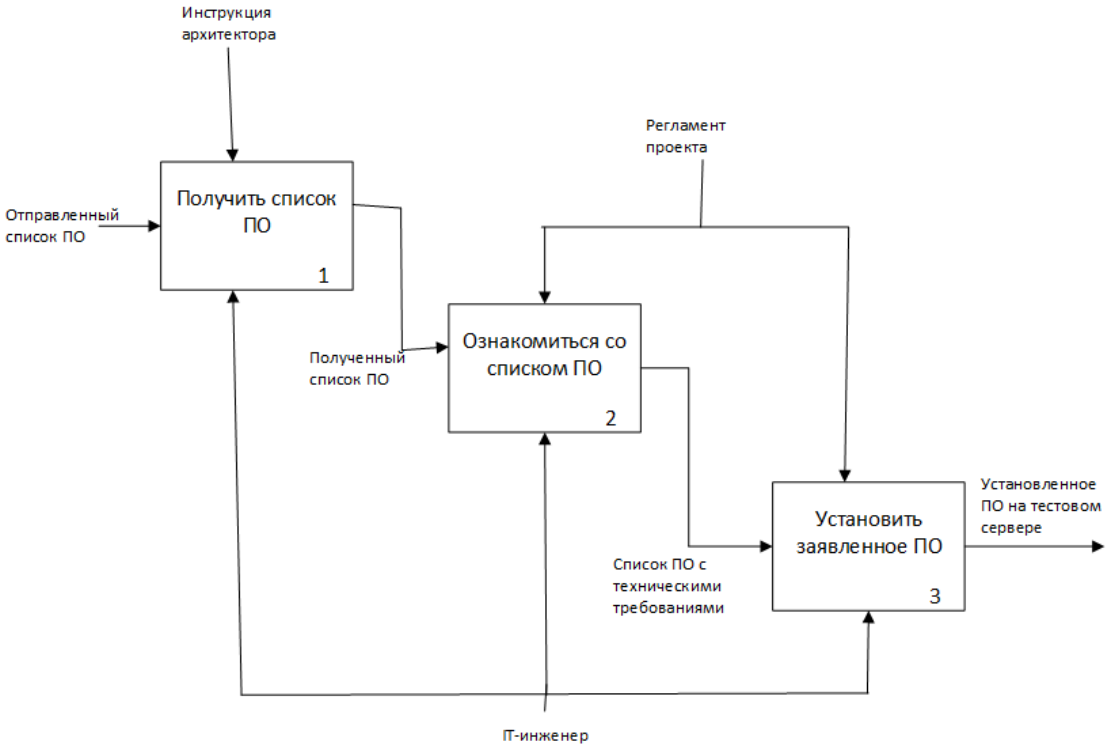


Рисунок 12 – Функциональная модель установки базового ПО

Список базового программного обеспечения влияет на кластер и средства связи приложения. От версий программного обеспечения зависит функционал кластера. Функциональная модель развертывания приложения с использованием кластера Weblogic, представлена на рисунке 13.

Конфигурация Weblogic может отличаться от ряда задач. При объединении окружений в кластер следует учитывать и момент оркестрации кластера. Функциональная модель установки пользовательского приложения представлена на рисунке 14. Оркестратор контейнеров способен запускать, выключать и разворачивать контейнеры с разными конфигурациями. Устанавливая пользовательское приложение на платформу возможно создать копию контейнера отвечающий за процесс в управляемой платформе.

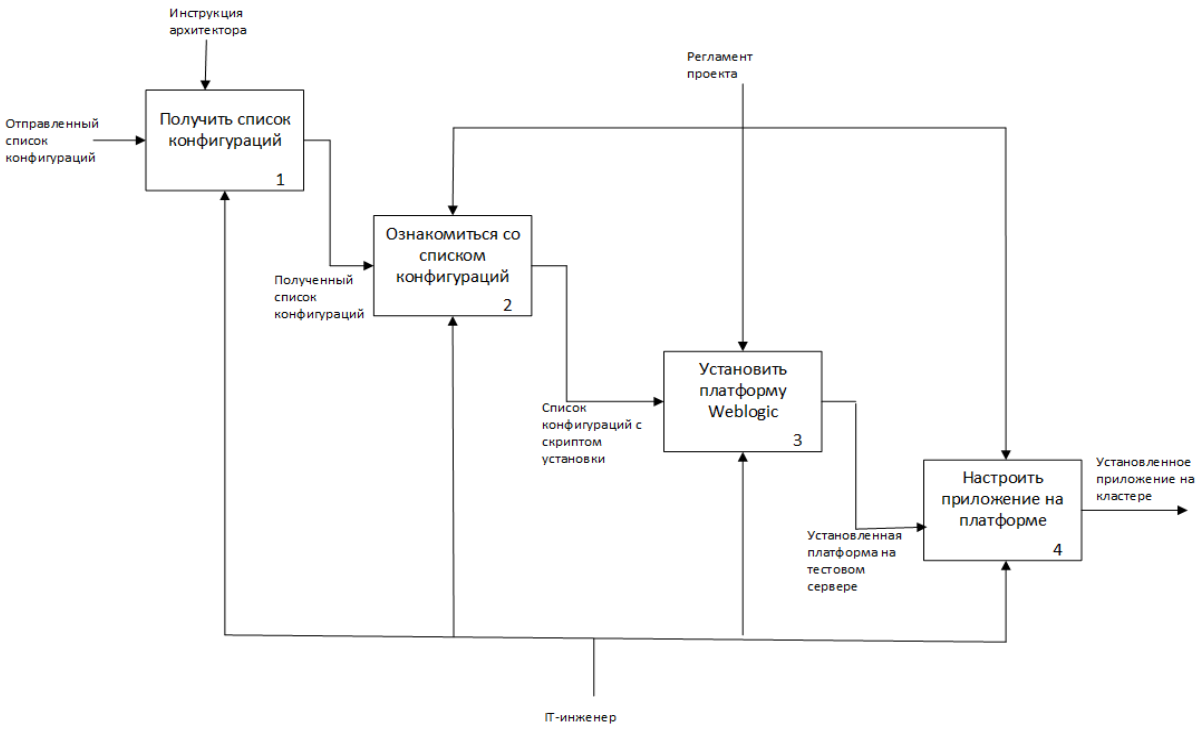


Рисунок 13 – Функциональная модель развертывания приложения с использованием кластера Weblogic

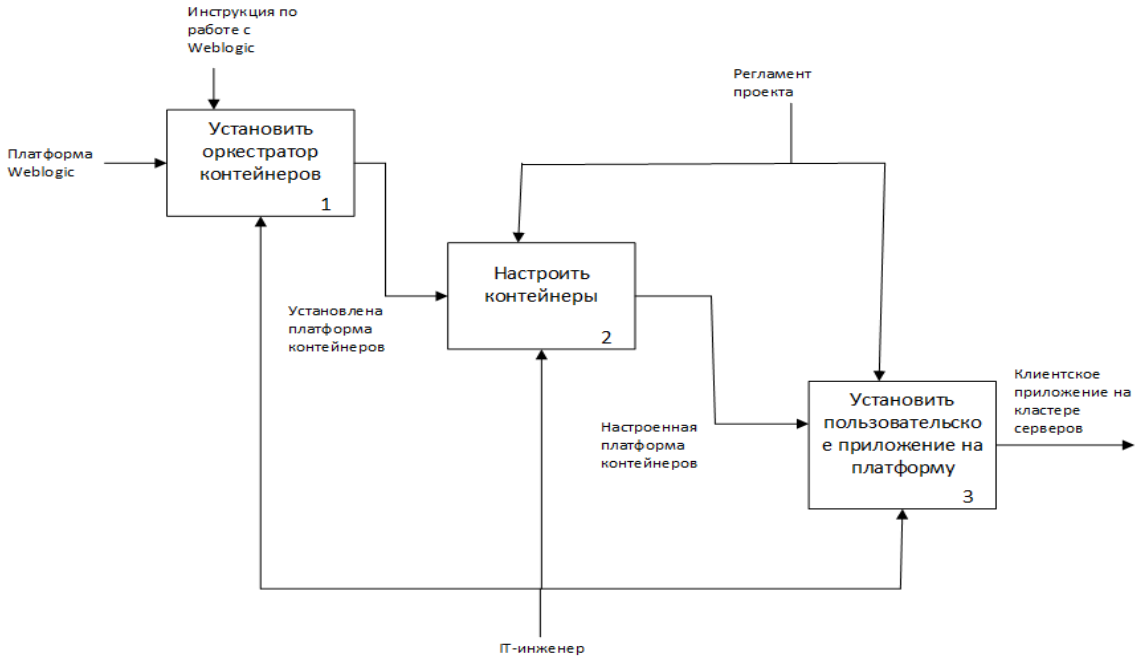


Рисунок 14 – Функциональная модель установки пользовательского приложения

Блок-схема актуализации конфигураций приложений, представлена на рисунке 15.

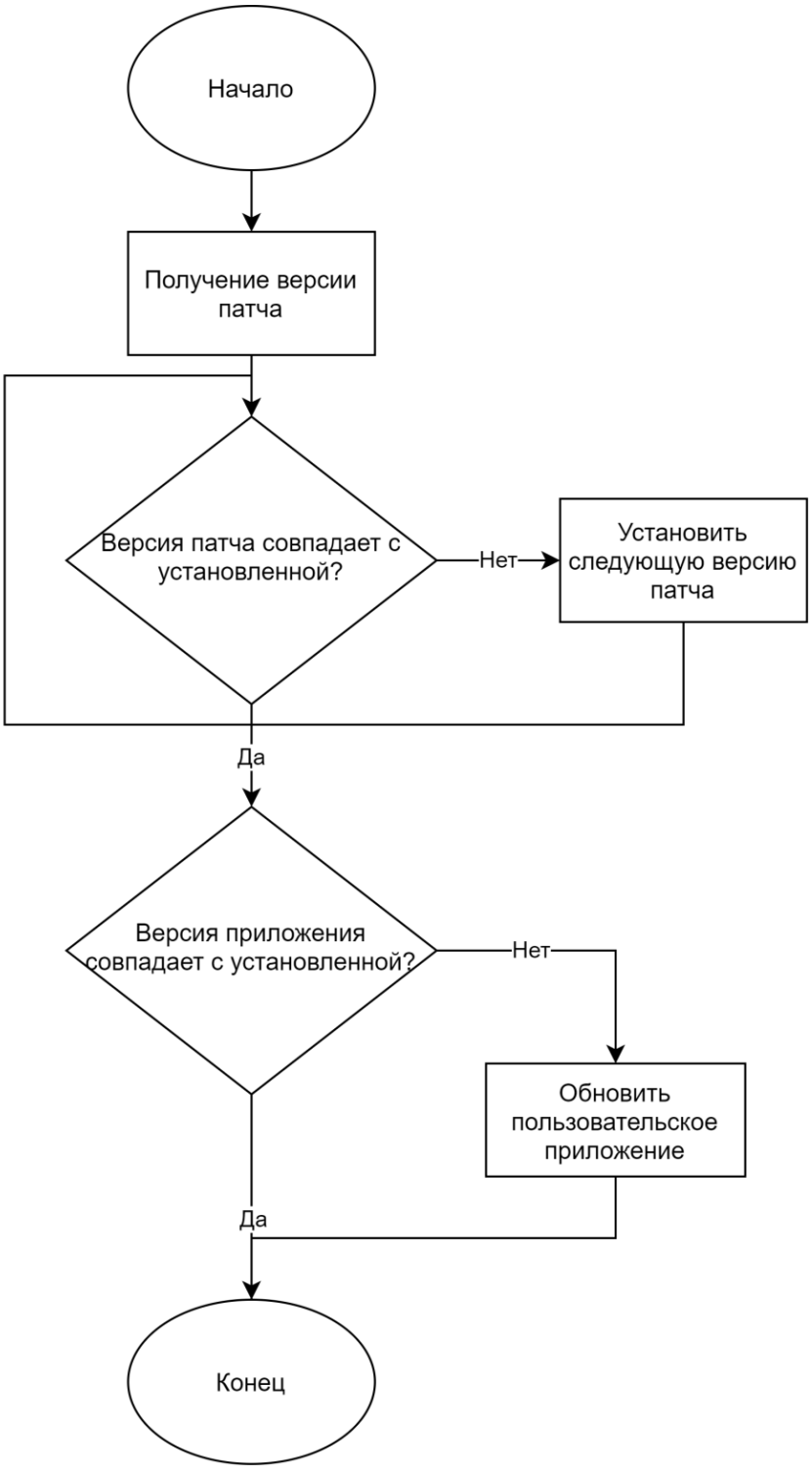


Рисунок 15 – Блок-схема актуализации конфигурации приложений для взаимодействия с другими приложения в системе посредством патчей

Соответствие версий приложения пользователя и патча окружения является важной задачей для производительности приложения.

Функциональная модель тестирования окружения, представлена на рисунке 16.

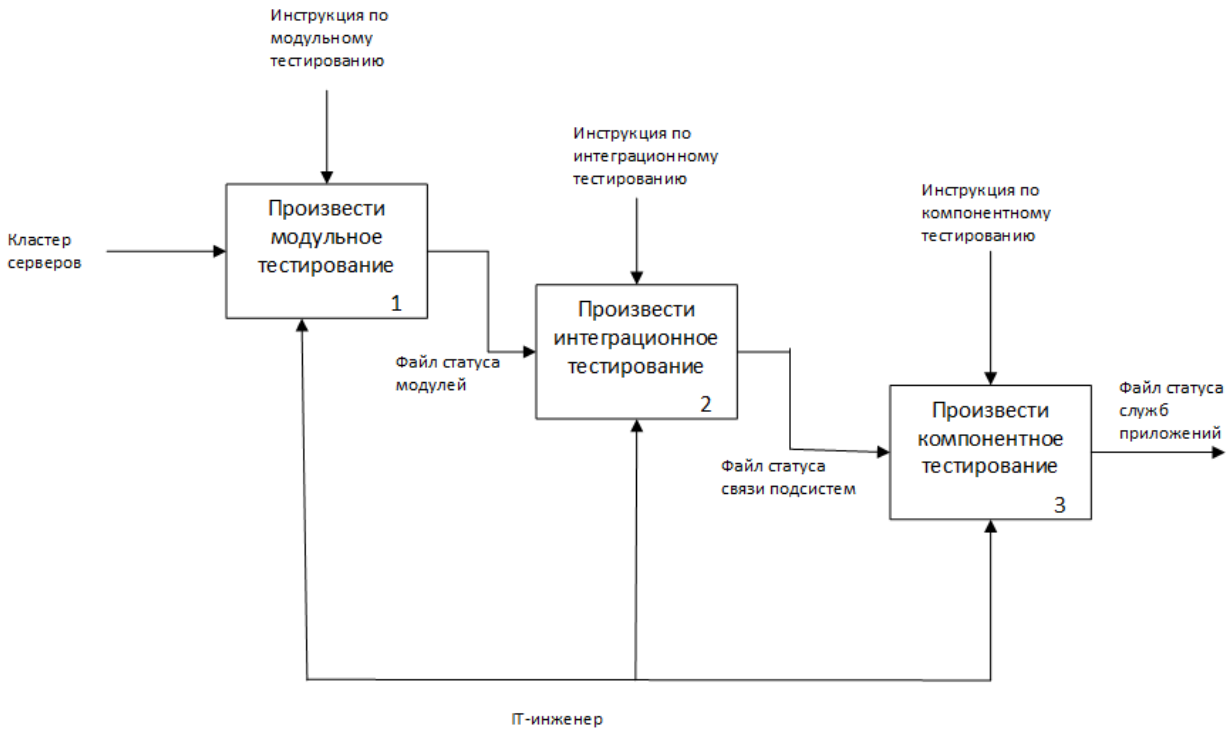


Рисунок 16 – Функциональная модель тестирования окружения

Модульное тестирование на этом этапе обрабатывается наибольшее количество отдельных тестовых единиц, что особенно важно при тестировании микросервисов.

Интеграционное тестирование для проверки связи между подсистемами, которые взаимодействуют с внешними компонентами, такими как хранилища данных и другие службы. Поскольку микросервисы обмениваются данными и выполняются вместе. Тесты должны проверять запросы, проходящие через сервисы, чтобы гарантировать, что каналы связи функционируют должным образом.

Компонентное тестирование проверяет микросервисы, формируя имитирующие сервисы, которые напоминают развернутые сервисы.

Это помогает проверить связь между распределенными службами и их зависимостями, такими как база данных и сторонние компоненты.

Функциональная модель переноса конфигурации на клиентский сервер, представлена на рисунке 17.

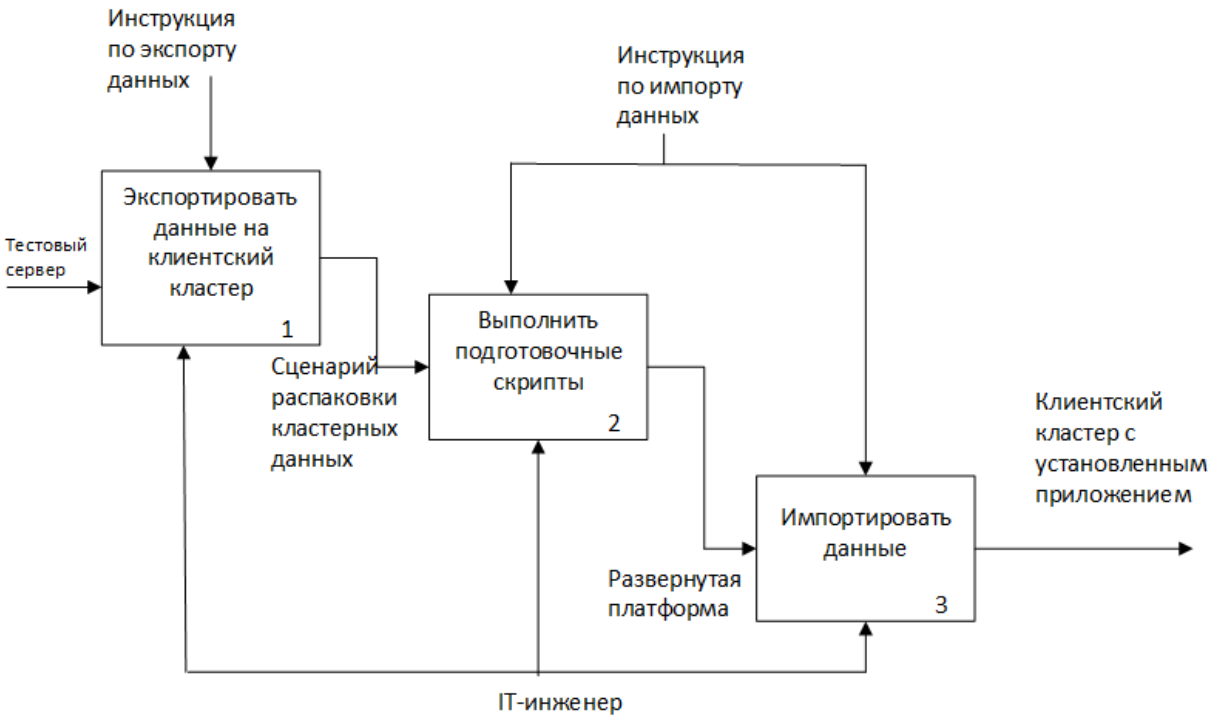


Рисунок 17 – Функциональная модель переноса конфигурации на клиентский сервер

Сделав экспорт данных, создаются и выполняются скрипты для развертывания окружения на клиентском сервере. Затем данные импортируются на клиентское приложение на пользовательском кластере.

Таким образом в данном параграфе было описано проектирование схемы инструментария, а также составляющие инструментария.

2.2 Проектирование уровня бизнес-логики

Бизнес-логика – правила, которые определяют каким образом данные могут быть созданы, сохранены, изменены. Процесс развертывания сервера для облачных вычислений также включает в себя развертывание базы данных.

Настройка базы данных для операционной системы в таком случае происходит по умолчанию в автоматизированном режиме. Инструмент, отвечающий за развертывание базы данных, – является скрипт, взаимодействующий с базой данных mongodb. Данный скрипт выполняет задачу импорта базы данных. Используя json- файлы для хранения, каждая таблица выделена для определенного микросервиса.

На рисунке 18 показано взаимодействие микросервисов с базой данных.

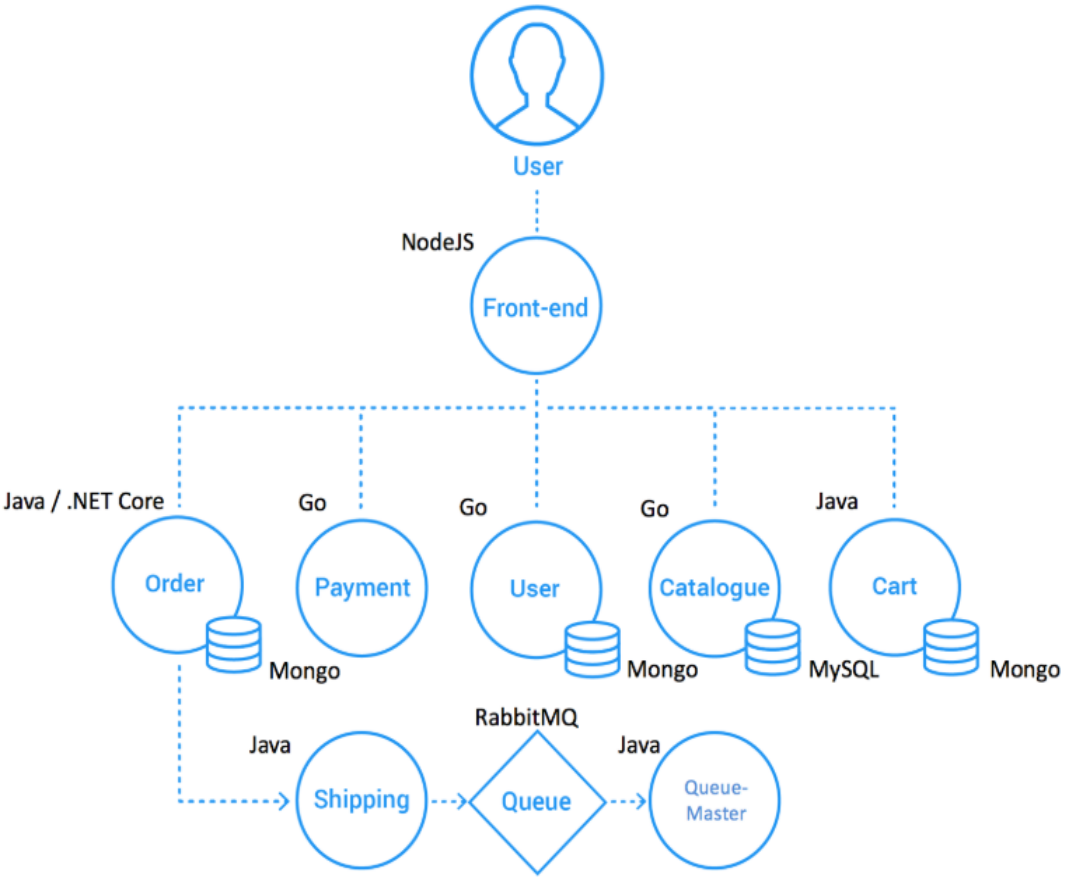


Рисунок 18 – Взаимодействие микросервисов с базой данных

Микросервис Front-end включает в себя выполнения вычислений и поддержу html страниц. Catalog отвечает за поддержку взаимодействия с базой данных. User используется для показа личного кабинета пользователя. Order отвечает за товары поставленные в корзине покупателя. Payment отвечает за вычисление стоимости покупателя. Cart включает в себя модуль запроса оплаты в банке. Shipping выполняет проверку транзакций. Queue уточняет приоритет. Queue-Master отслеживает очередь обращений к данным. Архитектура демонстрационного приложения микросервисов была специально разработана для предоставления как можно большего количества микросервисов. Все службы обмениваются данными с помощью REST через HTTP. Это было выбрано из-за простоты разработки и тестирования.

2.3 Проектирование уровня доступа к данным

В составе компонентов к установке на микросервисное окружение есть база данных. В которой лежит некоторые данные, для которых необходимо разграничить доступ согласно данных заказчика групп пользователей. Список групп пользователей:

- daemon – от этой группы и пользователя daemon запускаются сервисы, которым выполняют задачи в фоновом режиме;
- sys – группа имеет доступ к исходным данным ядра и файлам include сохраненным в системе;
- sync – позволяет синхронизировать данные на диске и данные в памяти;
- man – позволяет инициализировать или обновить кэши базы данных;
- lp – позволяет использовать устройства параллельных портов и выполнять настройку сетевой подсистемы;

- proxy – используется прокси серверами;
- www-data – с этой группой запускается веб-сервер, группа дает доступ на запись в каталог /var/www, где находятся файлы веб-документов;
- nogroup – группа используется для процессов, которые не могут создавать файлов на жестком диске, а только читать, обычно применяется вместе с пользователем nobody;
- adm – группа позволяет читать логи из директории /var/log;
- tty – все устройства /dev/vca разрешают доступ на чтение и запись пользователям из этой группы;
- disk – группа открывает доступ к жестким дискам, расположенным в каталогах /dev/sd* /dev/hd*;
- wheel – позволяет запускать утилиту sudo для повышения привилегий;
- shadow – разрешает чтение файла /etc/shadow.

Группы пользователей созданных для взаимодействия с базой данных включают в семя разрешения:

- администрирование пользовательских каталогов;
- ведение логов выполнения процедур;
- выполнения пользовательских сценариев;
- обращение к сетевым запросам;
- импортирование и экспортирование данных из файловой системы пользователя;
- изменение схемы базы данных;

База данных будет хранится в каталоге пользователя /u01/weavesocks. В каталоге /u01 должна быть организована структура каталогов, показанная на рисунке 19.

Структура каталогов:

- /u01/weavesocks – хранит данные приложения и микросервисы;
- /u01/temp – хранит исполняемые файлы подготовки к установке;

- /u01/share – каталог создан для объединения виртуальных машин в кластерную систему;
- /u01/lib – каталог для хранения пользовательских библиотек;
- /u01/app_compile – каталог для хранения программ использующих java;
- /u01/template_installed – каталог хранения шаблонных скриптов;
- /u01/cfg – каталог хранения конфигурационных файлов;
- /u01/scripts – каталог для хранения сценариев и пользовательских скриптов;
- /u01/data – каталог хранения файлов для импорта в базы данных.

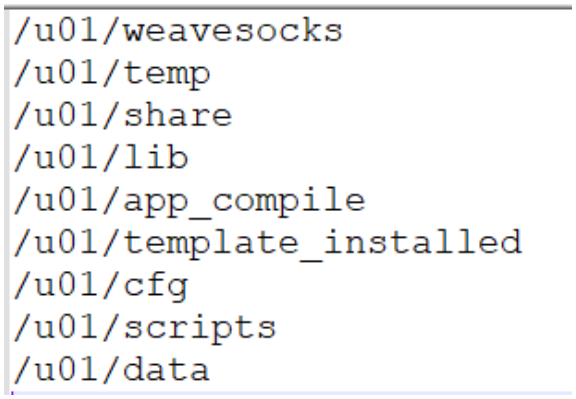


Рисунок 19 – Примерная структура хранения пользовательских файлов

Таким образом в данном параграфе был описан уровень проектирования доступа к данным.

2.4 Проектирование слоя интеграции

Интеграционный слой представляет собой структурированные обращения приложения к базе данных с применением унифицированного подхода связи виртуального окружения и API запросов. Программы, использующие инструментарий, способны управлять состояниями

микросервисов, способны выключать, создавать и перезапускать процессы и виртуальные окружения обеспечивая высокую доступность.

Высокая доступность – уровень, при котором система или приложение доступны в обозначенные требованиями дни и часы без незапланированных простоев, а о запланированных остановках в работе объявлено заранее.

Для данного проекта используется непрерывная интеграция в окружении. Непрерывная интеграция берет свое начало в XP (экстремальное программирование) и направлено на минимизацию конфликтов интеграции и ускорению самого процесса интеграции с помощью так называемого подхода «понемногу и часто».

В команде, практикующей CI все участники регулярно фиксируют свои изменения в основной ветке или ветке выделенном репозитории (обычно не реже одного раза в день). Каждая фиксация в мастере автоматически запускает сборку и набор тестов.

В качестве ключевого элемента непрерывной интеграции, выбран статус, согласно которому магистраль или центральная главная ветвь всегда должны быть готовы к отправке. В большинстве случаев это повышает вероятность того, что выполненная сборка будет успешной и программный код пройдет все этапы от отправки тестирования.

Если сборка потерпит неудачу: авто-тесты будут провалены, тогда приоритетной задачей для проекта будет задача исправления ошибки.

В то время как CI фокусируется на регулярных фиксациях выбранный нами метод непрерывной доставки заключается в автоматическом перемещении этих сборок по конвейеру развертывания для тестирования и отправки.

Чтобы использовать одну и ту же сборку для каждой среды, то необходимо переместить все переменные, относящиеся к среде, из основной базы в файлы конфигурации. Это включает в себя настройку флагов функций, которые можно использовать для некоторых тестовых сред или отключить их в производственной среде.

Использование файлов конфигурации – необходимая часть для автоматизации эффективного развертывания приложений с микросервисной архитектурой. Таким образом, устраняется необходимость в ручном вмешательстве, то есть минимизируется количество возможных ошибок.

При непрерывной доставке сборке автоматически перемещаются по конвейеру до последнего этапа: последний этап остается ручным.

Диаграмма развертывания в нотации UML представлена на рисунке 20.

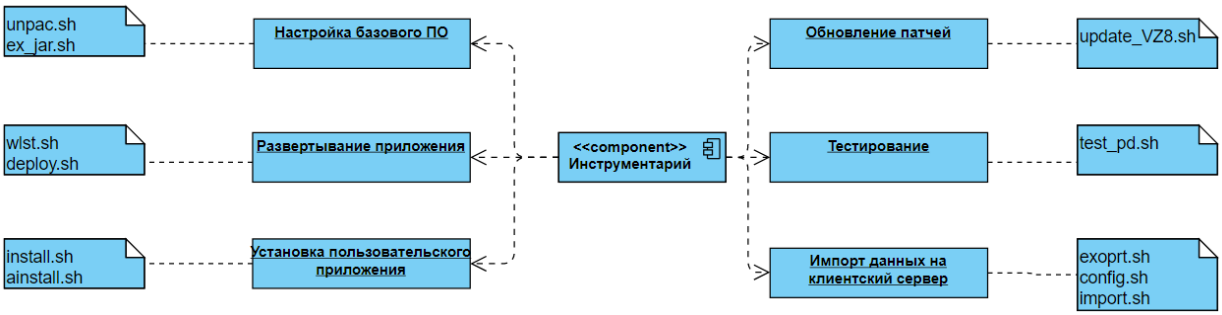


Рисунок 20 – Диаграмма развертывания

Таким образом, в данном параграфе был описан слой интеграции. В качестве интеграционного слоя был выбран метод непрерывной интеграции.

Глава 3 Разработка архитектуры облачной инфраструктуры для развертывания микросервисных приложений

3.1 Развертывание элементов инфраструктуры

Первым шагом подготовки и настройки DVM И OKD является базовая настройка VM. Первым шагом происходит настройка сетевого менеджера. Конфигурационный файл лежит по пути /etc/sysconfig/network-scripts/ifcfg

Содержимое конфигурационного файла предоставлено на рисунке 21.

```
DEVICE=eth0
HWADDR=00:0C:29:46:6A:95
TYPE=Ethernet
UUID=25a7bad9-616a-40a0-ace5-52aa0af9fdb7
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=static
IPADDR=192.168.90.1
NETMASK=255.255.255.0
GATEWAY=192.168.90.254
```

Рисунок 21 – Содержание файла ifcfg

Файл настройки библиотеки распознавателя имен DNS, находится в /etc/resolv.conf. Содержимое файла представлено на рисунке 22.

```
search nc.proba
nameserver 192.168.56.100
```

Рисунок 22 – Содержание файла resolv.conf

Файл /etc/named.conf хранит основные конфигурации DNS сервера. Содержание файла представлено на рисунке 23.

```
listen-on port 53 { any; };     // listen for requests on port 53 from any hosts
allow-query { any; };       // allow requests from any networks/hosts
forwarders { 10.0.2.3; };       // forward unresolved addresses to provider's DNS
zone "nc.proba" in {        // forward zone nc.proba
type master;
file "/var/named/proba.zone";   // file with zone nc.proba
};
zone "56.168.192.in-addr.arpa" in { // reverse zone 192.168.56.x
type master;
file "/var/named/192.168.56.db";     // file with reverse zone 192.168.56.x
};
```

Рисунок 23 – Содержание файла named.conf

Файл хранящий конфигурацию зон DNS располагается в /etc/named/proba.zone. Показан на рисунке 24.

```
$TTL 604800             // this zone timings
@   IN SOA dvm.nc.proba. admin.nc.proba. (
1;
604800;
86400;
2419200;
604800;
);
@   IN NS   dvm.nc.proba.       // DNS server
    IN A    192.168.56.100      // IPv4 address of DNS server
dvm     IN A 192.168.56.100 // IPv4 address of DVM host
okd     IN A 192.168.56.150 // IPv4 address of OKD host
```

Рисунок 24 – Содержание файла proba.zone

Таким образом в данном параграфе было выполнено базовая настройка виртуальной машины для работы с облачными технологиями, а также была выполнена настройка DNS сервера на примере операционной системы CentOS 7.

3.2. Настройка и администрирование облачного окружения

Следующим шагом в настройке DVM и OKD является установка Ansible и Jenkins. Для установки Ansible используются следующие команды. Вызов этой команды представлен на рисунке 25.

```
yum install epel-release
yum install ansible
```

Рисунок 25 – Команды установки ansible

Создадим Playbook для установки Jenkins. Файл показан на рисунке 26.

```
---
- name: Install Jenkins
  hosts: dvm
  tasks:
    - name: Java Install
      yum:
        name:
        - wget
        - java-1.8.0-openjdk
  - name: Ensure Jenkins Repository is Installed
    yum_repository:
      name: Jenkins
      state: present
      description: Official Jenkins Yum Repo
      baseurl: http://pkg.jenkins.io/redhat
      gpgkey: https://jenkins-ci.org/redhat/jenkins-ci.org.key
      gpgcheck: yes
      enabled: yes
  - name: Ensure Jenkins is Installed
    yum:
      name: jenkins
      update_cache: yes
      state: present
  - name: Enable and Start the Jenkins Service
    service:
      name: jenkins
      enabled: yes
      state: started
  - name: Open Firewall Port
    firewalld:
      zone: public
      port: 8080/tcp
      permanent: true
      state: enabled
      immediate: true
...
```

Рисунок 26 – Playbook установки Jenkins

Для нашего приложения был установлен Jenkins с удаленного репозитория. Получены ключи приложения. Затем был запущен сервис Jenkins и изменена политика firewalld.

После установки Jenkins был создан Hosts файл с внутренними переменными окружения, показан на рисунке 27.

```
dvm
okd

[OpenShift]
okd

[OSEv3:children]
etcd
masters
nodes

[masters]
okd.nc.proba openshift_ip=192.168.56.150

[etcd]
okd.nc.proba etcd_ip=192.168.56.150

[nodes]
okd.nc.proba openshift_ip=192.168.56.150 openshift_node_group_name="node-config-all-in-one"

[OSEv3:vars]
ansible_ssh_user=jenkins
ansible_become=true
deployment_type=origin
openshift_disable_check = docker_image_availability,disk_availability,docker_storage,memory_availability
openshift_generate_no_proxy_hosts = True
containerized=True
openshift_release="3.11"
openshift_public_hostname=okd.nc.proba
openshift_master_default_subdomain=apps.okd.nc.proba
```

Рисунок 27 – Hosts файл

Hosts файл содержит группы серверов, ключевые слова и переменные окружения.

Playbook по подготовке пользователя OpenShift. В данной конфигурации его права повысили до суперпользователя и создали публичный ключ подключения. Playbook показан на рисунке 28.

```
---
- hosts: OpenShift
  tasks:
  - name: create user
    user:
      name: jenkins
  - name: add to sudoers
    lineinfile:
      path: /etc/sudoers.d/jenkins
      line: 'jenkins ALL=(ALL) NOPASSWD: ALL'
      state: present
      mode: 0440
      create: yes
  - name: put pubkey
    authorized_key:
      user=jenkins
      key="{{ lookup('file', '/var/lib/jenkins/.ssh/id_rsa.pub') }}"
...
```

Рисунок 28 – Playbook по подготовке пользователя OpenShift

После конфигурации пользователя был создан скрипт по установке микросервисного приложения. На рисунке 29 представлен скрипт установки.

```
#!/bin/bash

oc login -u system:admin -n default

oc create user adm
oc adm policy add-cluster-role-to-user cluster-admin adm --as=system:admin

oc new-project sock-shop

oc project sock-shop

oc create sa socks-shop
oc adm policy add-cluster-role-to-user cluster-admin -z socks-shop
oc adm policy add-scc-to-user privileged -z socks-shop
oc adm policy add-scc-to-user anyuid -z default

oc apply -f ./complete-demo.yaml

oc expose svc/front-end --hostname=demo-sock-shop.apps.okd.nc.proba
```

Рисунок 29 – Скрипт по установке микросервисного приложениями

В скрипте было выполнено подключение пользователем к клиентской виртуальной машине. Затем заданы политики кластера. И после описания

ролей было запущено выполнение установочного скрипта. Результат выполнения установки микросервисного приложения показан на рисунке 30.

```
        "deployment.extensions/queue-master created",
        "service/queue-master created",
        "deployment.extensions/rabbitmq created",
        "service/rabbitmq created",
        "deployment.extensions/shipping created",
        "service/shipping created",
        "deployment.extensions/user-db created",
        "service/user-db created",
        "deployment.extensions/user created",
        "service/user created",
        "route.route.openshift.io/front-end exposed"
    ]
}

PLAY RECAP *********************************************************************
okd                        : ok=3    changed=1    unreachable=0    failed=0

Finished: SUCCESS
```

Рисунок 30 – Установка микросервисного приложения

Установка микросервисного приложения прошла успешно. Клиентская машина прошла этапы установки и настройки микросервисного приложения.

Таким образом в данном параграфе была выполнена установка Jenkins, был создан hosts файл с описание переменных окружения, проведена установка и настройка микросервисного приложения.

3.3 Тестирование разработанного решения

Следующим шагом в автоматизации подготовки развертывании облачной инфраструктуры является тестирование. Согласно разработанному техническому заданию необходимо подготовить и провести тестирование в два этапа: предварительные испытания и эксплуатационные испытания.

Подготовка к предварительным испытаниям выключает в себя: проверку на работоспособность решения. Результат проверки представлен на рисунке 31.

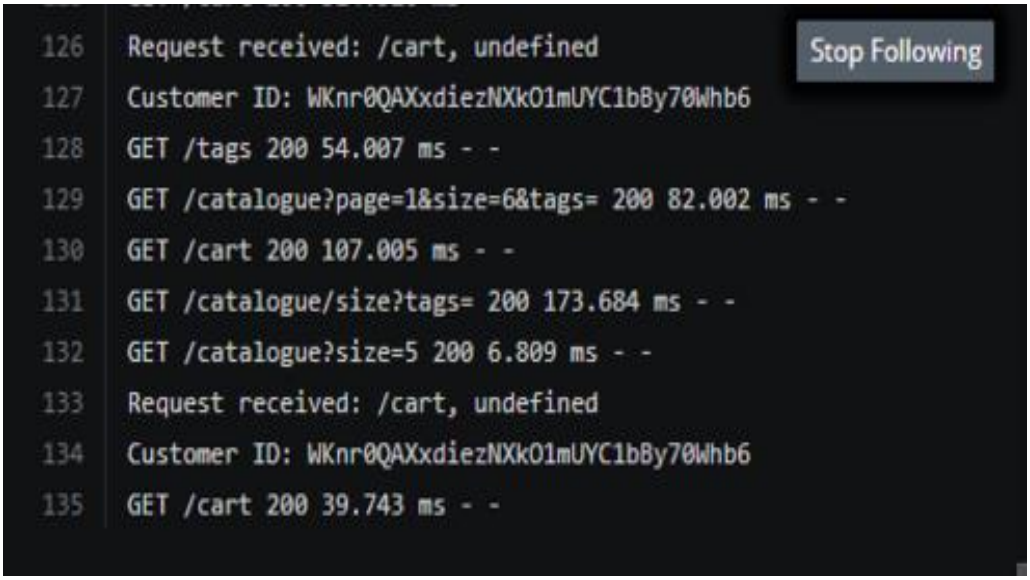


Рисунок 31 – Результат проверки

На рисунке представлен ответ, полученный от сервера. На сервере развернут некий сайт интернет-магазина. С пользовательской точки зрения успешность работы сервера состоит в том, что открывается веб-страница за определенное время или нет. Правилом хорошего тона считается время отклика сервера до 100 мс. Тестовый сайт интернет-магазина, который был размещен на автоматически собранном виртуальном окружении, был открыт за 70 мс. Таким образом были проведены предварительные испытание по сдаче работы заказчику. Совместно с заказчиком так же были произведены опытно-промышленные испытания, которые включают в себя нагрузочное тестирование, корректность структуры сервера согласно поставленным требованиям заказчика.

Нагрузочное тестирование было произведено успешно: для нагрузочного тестирования системы было задействовано 50 пользователей,

одновременно выполняющих тестирование с помощью сторонних средств. Нагрузка на сервер представлена на рисунке 32.

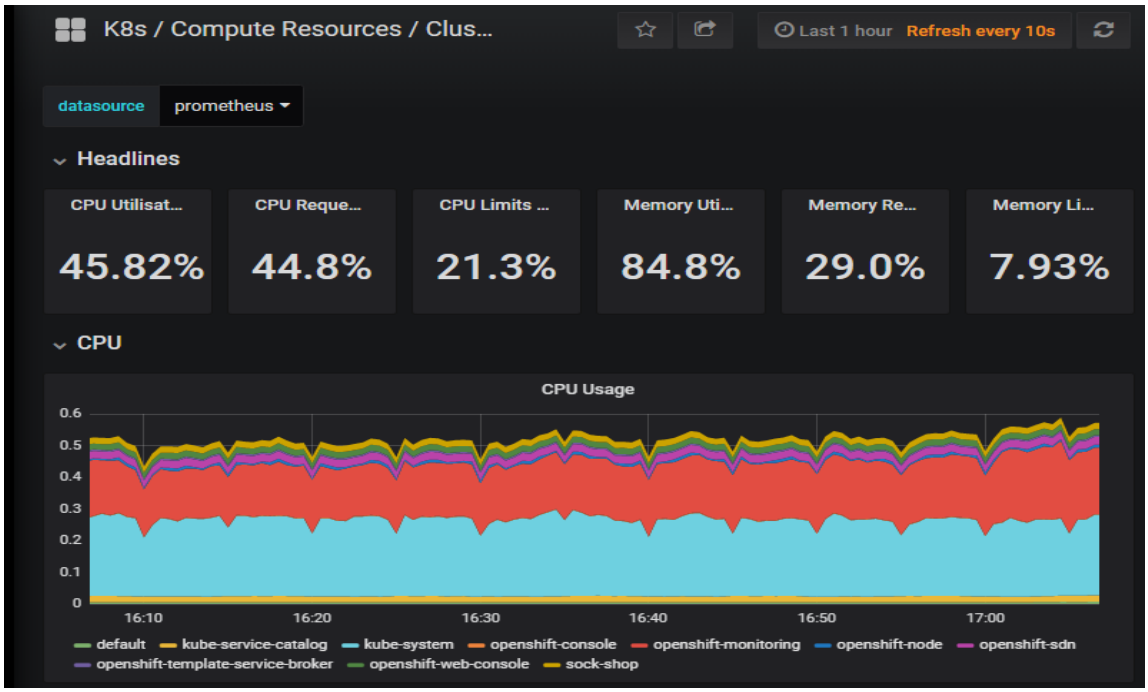


Рисунок 32 – Мониторинг нагрузки

Тестирование прошло успешно. Система работает согласно поставленной задаче.

Таким образом в данном параграфе были проведены предварительные и опытно-промышленные испытания. Реализованный проект прошел успешно данные испытания.

Заключение

Микросервисная архитектура – благодаря своим преимуществам находит все больше применений. В любом проекте есть возможности разбить задачи на модули и обеспечить независимость исполнения микросервисов.

Если проект требует возможности масштабирования, то продуманная микросервисная архитектура станет лучшим решением для построения кластера серверов.

Способность быстро организовать настроенное окружение для установки клиентского программного обеспечения влияет на разработку микросервисных приложений. Автоматизируя процесс настройки пользовательского окружения, уменьшается вероятность ошибки, связанная с человеческим фактором. При использовании технологии непрерывной интеграции микросервисное приложение способно проходить автоматическое тестирование, а при использовании технологии непрерывной доставки весь цикл обновления микросервисного приложения и его окружения будет автоматизирован. Данные обновлений поступают на сервер из инструментов контроля версий.

В заключении отметим система, созданная для автоматизации подготовки облачной инфраструктуры для развертывания микросервисных приложений, способна подготавливать клиентские машины для объединения их в кластер или задач, поставленных клиентами заказчика.

Практическая значимость бакалаврской работы состоит в автоматизации процессов развертывания облачного окружения и установки пользовательских приложений, имеющих микросервисную архитектуру.

Таким образом можно прийти к выводу, что для автоматизации эффективного развертывания приложений с микросервисной архитектурой необходимо использование файлов конфигурации, что позволит устранить необходимость в ручном вмешательстве, то есть минимизирует количество возможных ошибок.

Список используемой литературы

1. Арти З. Homo Futurus. Облачный Мир: эволюция сознания и технологий. М. : Издательство АСТ, 2019. 380 с.

2. Арундел Д. Kubernetes для DevOps. Развертывание, запуск и масштабирование в облаке. П. : Питер, 2019. 384 с.

3. Докука О. Практика реактивного программирования в Spring 5. Э. : Черно-белое, 2018. 510 с.

4. Клашанов Ф. К. Вычислительные системы и сети, облачные технологии. М. : НИУ МГСУ, 2020. 40 с.

5. Максимов К. В. Эффективность использования облачных вычислений: методы и модели оценки. М. : Синергия, 2016. 113 с.

6. Маркелов А. OpenStack. Практическое знакомство с облачной операционной системой. М. : ДМК Пресс, 2017. 270 с.

7. Хасс Р. Паттерны Kubernetes. Шаблоны разработки собственных облачных приложений. П. : Питер, 2019. 320 с.

8. Хохштейн Л. Запускаем Ansible. М. : ДМК Пресс, 2018. 384 с.

9. Bahga A. Cloud Computing Solutions Architect: A Hands-On Approach: A Competency-based Textbook for Universities and a Guide for AWS Cloud Certification and Beyond. В. : VPT, 2019. 826 с.

10. Duncan C. E. Winn. Cloud Foundry: The Definitive Guide: Develop, Deploy, and Scale 1st Edition, Kindle Edition. С. : O'Reilly Media, 2017. 478 с.

11. Gilbert J. Cloud Native Development Patterns and Best Practices: Practical architectural patterns for building modern, distributed cloud-native systems. Б. : Packt Publishing, 2018. 316 с.

12. Hoff T. Explain the Cloud Like I'm 10. LG. : Possibility Outpost Inc, 2018. 239 с.

13. Jackson K. OpenStack Cloud Computing Cookbook: Over 100 practical recipes to help you build and operate OpenStack cloud computing, storage, networking, and automation. Б. : Packt Publishing, 2018. 398 с.

14. Kim G. The Phoenix Project. П. : IT Revolution Press, 2018. 432 с.

15. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems Paperback. С. : O'Reilly Media, 2017. 616 с.

16. Laszewski T. Cloud Native Architectures: Design high-availability and cost-effective applications for the cloud. Б. : Packt Publishing, 2018. 358 с.

17. Morris K. Infrastructure as Code: Managing Servers in the Cloud 1st Edition. С. : O'Reilly Media, 2016. 362 с.

18. Mulholland A. Enterprise Cloud Computing: A Strategy Guide for Business and Technology Leaders Perfect Paperback. ЛА. : Meghan-Kiffer Press, 2010. 260 с.

19. Passmore E. Migrating Large-Scale Services to the Cloud. НЮ. : Apress, 2016. 110 с.

20. Vadapalli S. Hands-on DevOps: Explore the concept of continuous delivery and integrate it with data science concepts Kindle Edition. Б. : Packt Publishing, 2017. 424 с.

Приложение А

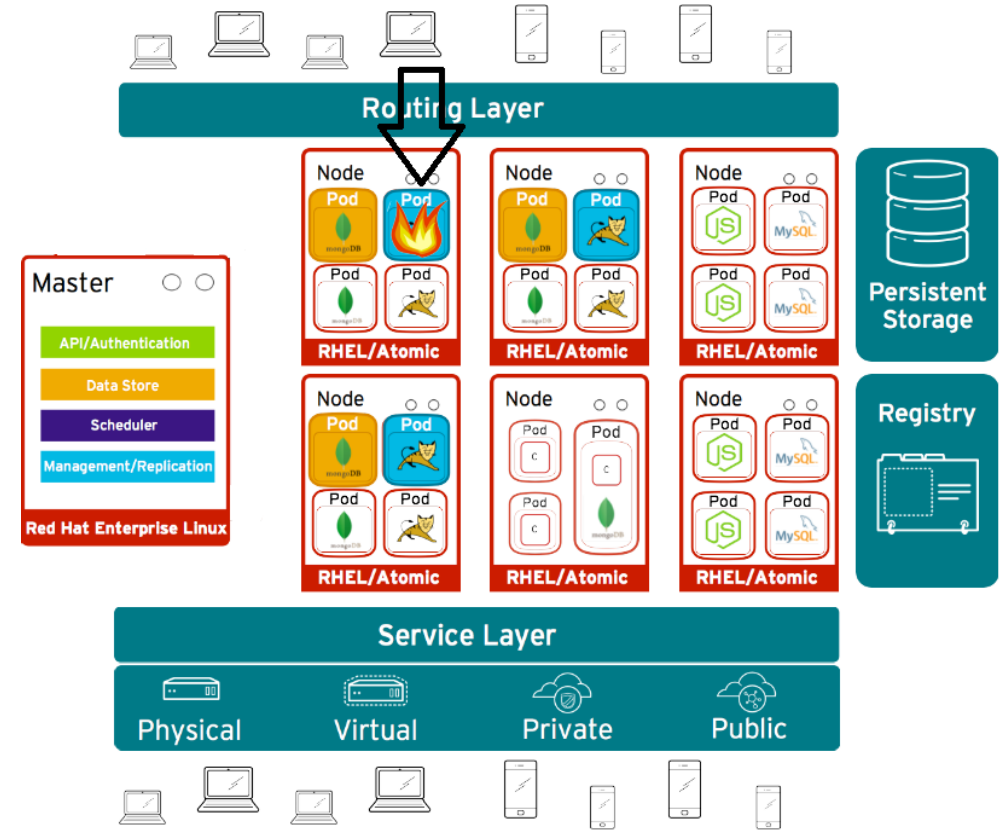
Playbook установки Jenkins

Playbook установки Jenkins. Показан на рисунке 33.

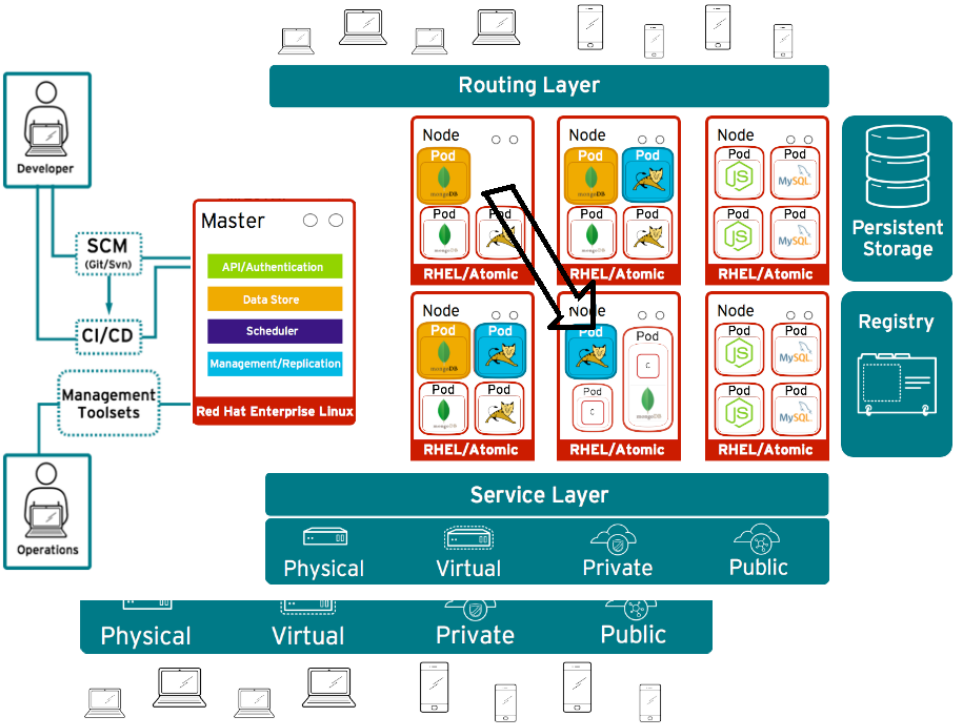
```
---
- name: Install Jenkins
  hosts: dvm
  tasks:
    - name: Java Install
      yum:
        name:
        - wget
        - java-1.8.0-openjdk
  - name: Ensure Jenkins Repository is Installed
    yum_repository:
      name: Jenkins
      state: present
      description: Official Jenkins Yum Repo
      baseurl: http://pkg.jenkins.io/redhat
      gpgkey: https://jenkins-ci.org/redhat/jenkins-ci.org.key
      gpgcheck: yes
      enabled: yes
  - name: Ensure Jenkins is Installed
    yum:
      name: jenkins
      update_cache: yes
      state: present
  - name: Enable and Start the Jenkins Service
    service:
      name: jenkins
      enabled: yes
      state: started
  - name: Open Firewall Port
    firewalld:
      zone: public
      port: 8080/tcp
      permanent: true
      state: enabled
      immediate: true
…
```

Приложение Б

Работа OpenShift



А) Прекращение работы Pod



Б) Перезапуск работы Pod

Приложение В

Клиент-серверная архитектура