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 *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ

«Развертывание и масштабирование многопользовательского приложения, разработанного на Unreal Engine 4»

Студент РК6-11М — Д. В. Боженко
(Группа) (подпись, дата) (инициалы и фамилия)

Руководитель — Ф. А. Витюков
(подпись, дата) (инициалы и фамилия)

Москва, 2023 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой РК6
А.П. Карпенко

«_____» _____ 2023 г.

ЗАДАНИЕ
на выполнение научно-исследовательской работы

по теме: Развертывание и масштабирование многопользовательского приложения, разработанного на Unreal Engine 4

Студент группы __РК6-11М___

_____Боженко Дмитрий Владимирович_____
(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.) _учебная___
Источник тематики (кафедра, предприятие, НИР) _кафедра_____

График выполнения НИР: 25% к 5 нед., 50% к 11 нед., 75% к 14 нед., 100% к 16 нед.

Техническое задание: проанализировать возможные варианты хостинга на виртуальных выделенных серверах многопользовательского приложения, разработанного на Unreal Engine 4. После проведения анализа выбрать наиболее подходящее решение и описать характерные для него особенности процесса развертывания выделенного сервера.

Оформление научно-исследовательской работы:
Расчетно-пояснительная записка на 26 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):
_____
_____
_____
_____

Дата выдачи задания «24» октября 2023 г.

Руководитель НИР _____ **Витюков Ф.А.**
(Подпись, дата) И.О. Фамилия

Студент _____ **Боженко Д. В.**
(Подпись, дата) И.О. Фамилия

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедр

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развертывание приложения любого вида – это один из важных этапов разработки любого продукта. От качества развертывания приложения зависит такие параметры как отказоустойчивость, скорость и масштабируемость приложения, а также в целом дальнейшее качество его использования конечным пользователем.

Для того, чтобы развертывание приложения получилось качественным, необходимо провести предварительный анализ и расчет, а именно подсчитать максимально возможное количество одновременных подключений клиентов, максимальное количество требуемых вычислительных ресурсов, а также будет ли приложение использовать кластерную архитектуру или же использовать сервер в единственном экземпляре.

После проведенного анализа, используя подсчитанные данные, необходимо выбрать вид хостинга. Для больших проектов с серьезной нагрузкой речь идет о хостинге на физических выделенных серверах или же на виртуальных выделенных серверах.

Хотя развертывание на физическом аппаратном программном обеспечении на данный момент до сих популярно и в некоторых случаях незаменимо, большое количество крупных приложений разворачиваются и запускаются в облачных сервисах, так как там запуск приложения, как правило, более доступный и быстрый, чем настройка целого выделенного сервера. При этом хостинг приложения в облаке имеет лишь незначительные ограничения по сравнению с более привычным и традиционным способом развертывания.

В рамках данной работы была поставлена задача проанализировать возможные варианты хостинга на виртуальных выделенных серверах многопользовательского приложения, разработанного на Unreal Engine 4. Также необходимо изучить процесс и способы масштабирования приложения. После проведения анализа выбрать наиболее подходящее решение и описать

характерные для него особенности процесса развертывания выделенного сервера.

1. ОСНОВНЫЕ ВИДЫ ХОСТИНГА

1.1. Listen-server

Listen-server – один из способов запустить приложение. Его особенность заключается в том, что любой узел или группа узлов, находящихся в одноранговой сети, способны выполнять как роль сервера, так и роль клиента, что соответствует типу архитектуры сети Peer-to-peer.

Явным плюсом такого вида хостинга является его простота. Для обеспечения многопользовательской игры в Unreal Engine необходимо, чтобы один из клиентов был запущен в режиме Listen-сервера, и тогда он сможет принимать запросы клиентов и управлять их подключениями.

К минусам такого вида хостинга можно отнести:

1. Ограниченность вычислительных ресурсов: роль хостинга берет на себя один из клиентов, вычислительные ресурсы которого сильно ограничены. Из-за этого возникает проблема малого количества возможных подключений в пределах одной игровой сессии.
2. Непостоянность хостинга: узел, который выполняет роль сервера, может намеренно остановить работу Unreal Engine приложения или же потерять подключение к сети. При этом клиенты уже не могут продолжать находиться в данной сессии, так как не кому больше выполнять роль сервера и обрабатывать запросы.
3. Ограничение топологией сети: пользователи, которые подключены к одной игровой сессии должны находиться в пределах одной локальной сети или же между ними должно быть настроено постоянное туннельное соединение, если взаимодействие между ними происходит по сети Интернет.

1.2. Выделенный сервер (dedicated server)

Dedicated server – способ хостинга, при котором роль сервера выполняет отдельная мощная вычислительная машина, обрабатывающая запросы всех подключенных клиентов и производящая расчёт всех переменных состояния

виртуального пространства. Подключение всех участников сети образует клиент-серверную топологию. При выделенном хостинге можно арендовать все физическое аппаратное программное обеспечивание и использовать его в своих нуждах.

К явным плюсам такого вида хостинга можно отнести:

1. Эксклюзивность: поставщик хостинга предоставляет полный и эксклюзивный доступ ко всему физическому серверу.
2. Вычислительная мощность: сервер – это мощная машина, обладающая большими вычислительными ресурсами. Наличие такой машины целиком позволяет обрабатывать большое количество запросов в единицу времени, следовательно к одной сессии может подключиться большое количество клиентов. Также хостинг Unreal Engine приложения будет постоянен, так как выделенный сервер находится в дата центре.

К минусам можно отнести:

1. Высокая стоимость: организовать хостинг с выделенным сервером на физическом аппаратном программном обеспечении финансового дорого.
2. Сложность эффективного использования: для работы Unreal Engine приложения может не понадобиться столько ресурсов, сколько предоставляет арендованные выделенный сервер. При этом, неиспользуемые вычислительные ресурсы будут простаивать и не приносить никакой пользы.

1.3. Virtual Private Server

Virtual Private Server (VPS) – это выделенный виртуальный сервер, который потребляет только часть физических ресурсов физического сервера, на котором он запущен. При аренде VPS клиент получает полный доступ к виртуальной машине с выделенной для нее ресурсами физического аппаратного обеспечения в соответствии с тарифом клиента и его нуждами, даже при

совместном использовании ресурсов физического аппаратного обеспечения сервера другими пользователями.

Хостинг с использованием VPS содержит все плюсы и характеристики выделенного сервера, но также имеет ряд своих преимуществ. К ним можно отнести:

1. Динамическое выделение ресурсов: если на VPS предустановлено программное обеспечение, которое позволяет каким-либо образом масштабировать приложение, то он сможет осуществить предоставление ресурсов по мере необходимости. Если все выделенные ресурсы уже использованы к определенному моменту времени, можно выполнить запрос на предоставление новых для увеличения масштабов приложения. Например, для создания очередной сессии и ее хостинга необходимы дополнительные ресурсы для подключения новых пользователей.
2. Невысокая стоимость использования: использование хостинга на VPS помогает эффективно использовать столько ресурсов, сколько потребуется для работы приложения, из-за чего снижается стоимость аренды хостинга и повышается эффективность использования доступных вычислительных ресурсов.
3. Более простой процесс развертывания приложения – клиент, который использует VPS-хостинг может получить виртуальную машину с подготовленными для его нужд ресурсами и виртуальным окружением. Следовательно, нет необходимости иметь знания по сложному процессу запуска физического сервера и настройки его виртуального окружения и тратить большое количество времени на данную процедуру.

К недостатку VPS можно отнести отсутствие универсальности и возможности быстро перейти на хостинг другого поставщика. Как правило, для развертывания приложения на хостинге провайдера необходимо интегрировать

в исходный код API, которое он для этого предоставляет. При решении перейти на другого поставщика потребуется изменить исходный код приложения и интегрировать уже другое API, что может занять довольно много времени и вызвать трудности.

Также к недостаткам можно отнести то, что на долгосрочной перспективе аренда облачных серверов может обойтись значительно дороже, чем аренда выделенного физического сервера.

Ниже представлена схема, более подробно поясняющая различия принципа работы VPS и выделенного сервера (Рисунок 1).

Рисунок 1 – Схема принципа работы VPS и выделенного сервера

Анализируя рисунок, можно заметить, что каждый VPS имеет свою операционную систему и использует общие ресурсы физического сервера. Это позволяет создать множество виртуальных машин, каждая из которых будет настроена под определенные задачи и будет работать независимо от других экземпляров. Такое совместное использование также удобно, так как каждый экземпляр VPS имеет свои строго-выделенные ограниченные ресурсы, что не позволит ему забирать больше вычислительных ресурсов у других экземпляров.

Приложение, запущенное на выделенном физическом сервере, полностью забирает себе все вычислительные ресурсы. Также, это может быть не единственный вариант приложения, а кластер, где существует множество экземпляров приложения. На VPS, также можно реализовать кластерную архитектуру и использовать несколько экземпляров приложения на одном экземпляре сервера.

При выборе провайдера облачного хостинга для начала необходимо определиться, какой вид выделенного сервера необходимо будет запускать. Существуют два основных вида выделенных серверов, а также их смешения:

1. Постоянные выделенные серверы – это серверы, экземпляры которых создаются на продолжительное время. В таких экземплярах создается виртуальное пространство, которое также существует продолжительное время и состояние которого все время сохраняется. При этом должен существовать некий период, при котором состояние виртуального пространства перезаписывается или же создается абсолютно новое виртуальное пространство. В рамках приложения, которое запущено в экземплярах постоянных выделенных серверов, как правило представлен ограниченный список таких серверов с подробной о них информации, где пользователь может выбрать, к какому серверу он хочет и имеет возможность подключиться.
2. Сессионные выделенные серверы – это серверы, экземпляры которых создаются на короткий период времени для хостинга созданной сессии. Как правило, в рамках созданной сессии в виртуальном пространстве проходит матч, куда подключается ограниченное количество пользователей. После того, как матч завершается, завершается сама сессия, а экземпляр выделенного сервера, отвечающий за ее хостинг, удаляется. В рамках такого приложения, у пользователя, как правило, нет необходимости видеть список

запущенных экземпляров. Ему достаточно заявить о желании найти матч, и система должна сама автоматически выбрать наиболее подходящий под его параметры экземпляр сервера и отправить его структуру для подключения.

2. ОСНОВНЫЕ СПОСОБЫ МАСШТАБИРОВАНИЯ

Каждое развернутое приложение имеет свою пропускную способность или, выражаясь по-другому, имеет ограниченное количественно клиентских соединений, которое оно может обработать одновременно. Количество таких запросов и является мерой масштабируемости приложения.

Масштабируемость – это способность системы адаптироваться под изменяющиеся нагрузки. При росте одновременных подключений и исчерпывании текущего количества вычислительных ресурсов, приложение должно выделять новые, при падении числа одновременных подключений – убирать неиспользуемые ресурсы.

Масштабирование приложений подразделяется на два вида: горизонтальное и вертикальное. Каждое, несомненно имеет свои плюсы и минусы.

Вертикальное масштабирование – это разновидность масштабирования, при котором производительность системы улучшается за счет добавления дополнительных вычислительных мощностей на машину, на которой оно развернуто. Такой вид масштабирования прост с точки зрения администрирования, так как все множество экземпляров приложения может быть запущено на одной машине. Из ограничений такого подхода можно выделить:

1. Жесткое ограничение по ресурсам, так как машина, на которой развернуто приложение, может вместить в себя ограниченное количество памяти и вычислительных ядер.
2. Проблема отказоустойчивости. В случае, если машина, на которой развернута система выйдет из строя, ни один из ее экземпляров также не будет доступен.

Горизонтальное масштабирование – это разновидность масштабирования, при котором производительность системы улучшается за счет добавления новых машин, которые выполняют такую же задачу, как уже созданные. К плюсам

горизонтального масштабирования можно отнести отсутствие жесткого ограничения на вычислительные ресурсы, так как можно выделить довольно дополнительных ресурсов в соответствии с текущими нуждами. К минусам такого подхода можно отнести высокую стоимость, более сложный процесс администрирования и необходимость в большом количестве сетевого оборудования.

Вертикальное масштабирование, несомненно, может подойти при развертывании приложения на выделенном физическом аппаратном обеспечении, где есть возможность самостоятельно производить увеличение вычислительных мощностей машины. Горизонтальное масштабирование является более предпочтительным методом масштабирование. Подавляющее большинство провайдеров, которые предоставляют в аренду VPS, используют именно горизонтальный вид масштабирования.

2.1. Масштабирование с помощью предустановленных фреймворков

Некоторые провайдеры облачного сервиса могут предоставлять в аренду VPS, на которых уже предустановлено программное обеспечение, способствующее автоматизированному масштабированию приложения. Ярким и самым известным провайдером такого рода является Amazon Web Services.

Amazon Web Services (AWS) предоставляют отдельный вид хостинга, рассчитанный специально для хостинга выделенных серверов, в том числе и для Unreal Engine.

В AWS существует два вида инфраструктуры, которые могут быть использованы для развертывания выделенного сервера Unreal Engine: Elastic Compute Cloud (EC2) и GameLift.

EC2 – это основная инфраструктура AWS, которая предоставляет пользователю VPS в аренду. На них можно выполнять любую необходимую нагрузку.

Серверы GameLift – это те же самые EC2 серверы, на которых предустановлен фреймворк GameLift. Он берет на себя ответственность за администрирование всех запущенных экземпляров выделенных серверов приложения, а также позволяет автоматически масштабировать их, в случае, если количество текущих подключений превысило определенный порог и необходимо запустить новые экземпляры для новых пользователей. Таким образом GameLift идеально подходит для приложений с непродолжительными сессиями.

GameLift можно как интегрировать в проект в виде плагина и использовать его API, так и даже использовать интерфейс в виде командной строки для управления экземпляр.

AWS также предлагает несколько опций хостинга сервера с помощью фреймворка GameLift:

1. Хостинг для пользовательских серверов: позволяет развернуть полностью пользовательский сервер Unreal Engine с автоматическим масштабированием.
2. Realtime-хостинг: предлагает возможность разворачивать приложения, которые не требуют пользовательского сервера. Вместо этого AWS предлагает свое легковесное решение, которое можно конфигурировать под собственные нужды.
3. FleetIQ хостинг: предоставляет более расширенный и гибкий вариант хостинга, чем Realtime-хостинг, основанный на серверах AWS EC2.
4. Matchmaking хостинг: позволяет определить собственный набор правил для организации подбора многопользовательских матчей. Такой вариант можно использовать, если в приложении еще нет собственной системы рейтинга.
5. Anywhere хостинг: позволяет выполнять подбор игроков и игровых сессий на серверах AWS и при этом запускать игровые сессии на собственном оборудовании.

2.2. Масштабирование с помощью Kubernetes

Примерами провайдеров, которые предлагают в аренду VPS с предустановленным Kubernetes является Microsoft Azure, Google Cloud, DigitalOcean. Они так же, как и AWS, предоставляют экономичные и масштабируемые решения для развертывания любого серверного приложения, в том числе и для Unreal Engine. На их VPS предустановлен набор инструментов (Kubernetes, Kafka), который необходим для масштабирования приложения.

Kubernetes является одной из самых используемых и популярных платформ для развертывания и дальнейшего масштабирования приложения. Главной задачей Kubernetes является распределение контейнеров и запуск приложений в контейнерах в пределах одного кластера. Приложения в контейнерах более гибки и доступны, так как нет необходимости формировать, например, deb- и rpm-пакеты приложения, тесно связанные с определенным хостом.

Контейнер представляет собой изолированное адресное пространство внутри операционной системы, которое позволяет настроить виртуальное окружение, необходимое для работы приложения. Ниже продемонстрирована схема, на которой поясняется принцип устройства кластера в Kubernetes (Рисунок 2).

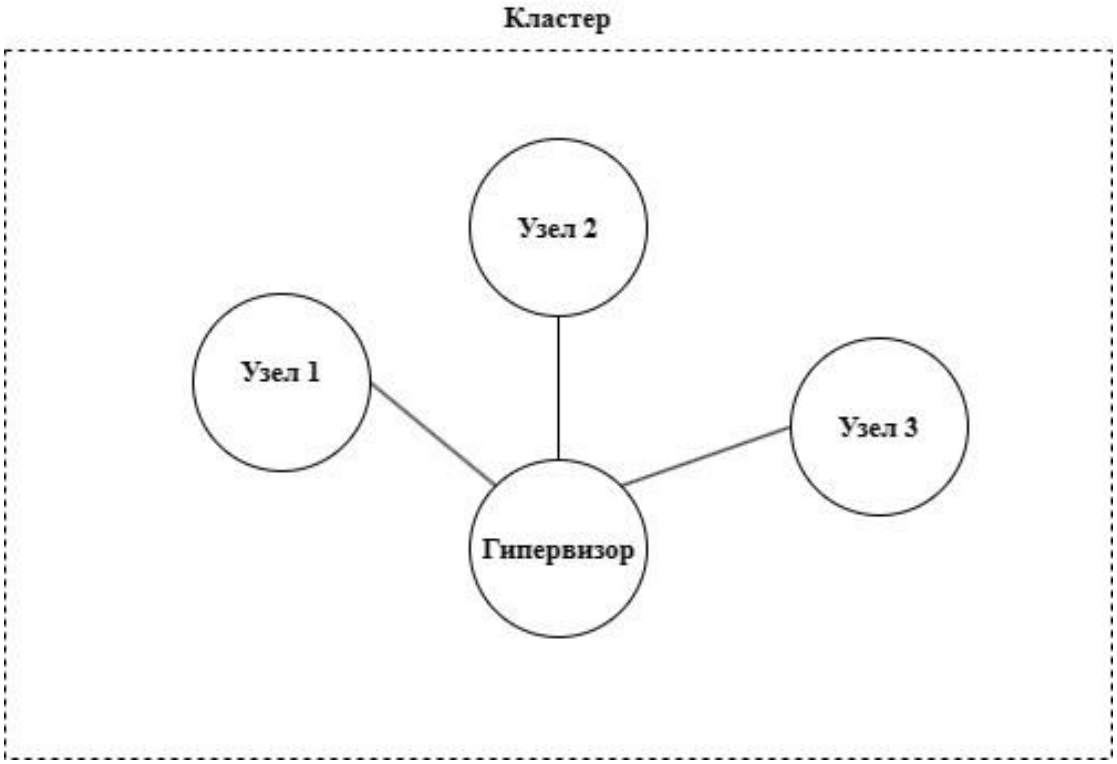

Рисунок 2 – Упрощенная схема простейшего кластера в Kubernetes

Первоначальным этапом развертывание приложения с помощью Kubernetes является создание кластера. В Kubernetes кластер должен состоять как минимум из трех узлов. Далее перечислены основные компоненты кластера Kubernetes и их описание.

Гипервизор – это главный узел в кластере, на котором не размещаются контейнеры с запущенным приложением и который координирует все процессы в кластере, а именно масштабирование, сохранение состояний приложений и их обновление. Другими словами, гипервизор “принимает” решение, в какой узел какой контейнер разместить. Например, если все ресурсы одного из узлов уже заняты, то он будет автоматически размещать новые экземпляры контейнеров уже на более свободных узлах.

Узлами в кластере могут выступать как VPS, так и физические серверы. На них размещаются и запускаются контейнеры с приложениями. У каждого узла есть свой Kublet. Kublet – это агент, который управляет узлом и отвечает за его взаимодействие с гипервизором. Узлы взаимодействуют с гипервизором с

помощью API Kubernetes. На узле также расположены инструменты контейнеризации, такие как, например, Docker.

Docker – это инструмент контейнеризации. Его основными задачами является создание образа контейнера. Образ представляет собой шаблон виртуального окружения. Ссылка на образ записывается в Docker-файле. Также Docker-файл содержит инструкцию по установке образа со всеми зависимостями, копированию файлов контейнерезированного приложения и команды для запуска самого контейнера. При наличии образа и инструкций к этому образу можно запустить экземпляр контейнерезированного приложения. Для начала работы с Kubernetes можно использовать, например, Minikube. Minikube – это упрощенная реализация Kubernetes, которая позволяет развернуть упрощенный кластер, состоящий из одного узла, на локальной виртуальной машине, которую он сам же и создает. Как только упрощенный кластер был запущен, необходимо создать деплоймент, чтобы была возможность развернуть в кластере контейнерезированные приложения. Деплоймент – это инструмент, который определяет, как создавать и обновлять экземпляры приложения. После того, как создается деплоймент, гипервизор начинает запуск контейнеров на отдельных доступных узлах в кластере.

Далее необходимо привести более подробное описание устройства узла в кластере Kubernetes. На самом деле, неделимой частью в узле является не контейнер, а под. Под представляет собой абстрактный объект в Kubernetes, который помимо одного или нескольких контейнеров одного или разного типа может содержать в себе общее хранилище, сетевые структуры и информацию о том, как запускать каждый из контейнеров. Все контейнеры внутри пода имеют один и тот же IP-адрес и пространство портов. В рамках одного узла может быть размещено несколько подов, а в рамках одного пода – несколько контейнерезированных приложений. Ниже приведена схема, поясняющее устройство узла в кластере Kubernetes (Рисунок 3).

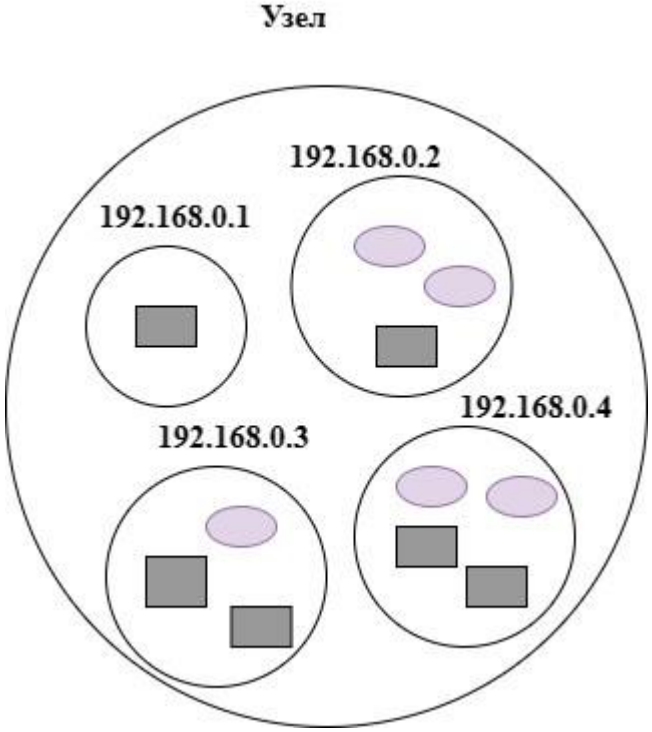

Рисунок 3 – Устройство пода в кластере Kubernetes

На рисунке окружностями с IP-адресами обозначены поды. Можно заметить, что каждый под имеет свой номер, как компьютер, имеющий свой определенный номер в подсети. В каждом поде серым прямоугольником обозначен сам контейнер, в котором запущено приложение. В рамках одного пода их может быть несколько. Фиолетовым цветом обозначено хранилище или базы данных. В поде их также может быть несколько.

Каждый под имеет свой цикл жизни. Это значит, что при выходе из строя узла, на котором был расположен под, он уничтожается. Чтобы все время поддерживать работоспособность приложения, Kubernetes отслеживает состояние кластера, и в случае необходимости, он создает уничтоженные поды и размещает их на работоспособных узлах. Чтобы приложение продолжало функционировать, необходимо, чтобы между подами одного типа была создана слабая связь. За это отвечает сервис. Сервис – это абстрактный объект в Kubernetes, который определяет набор подов, логически связанных между собой, а также задает политику доступа к ним. Также сервисы позволяют направлять

трафик через набор подов. Сервисы определяются с помощью файла с расширением *yaml*. Пример задания сервиса представлен ниже (листинг 1).

Листинг 1 – Задание сервиса в файле *.yaml*

```
kind: Service
apiVersion: v1
metadata:
  name: test-service
spec:
  selector:
    app: test
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: NodePort
```

Поле *type* позволяет указать, как должен быть открыт сервис:

- *ClusterIP* – делает сервис доступны только внутри кластера и открывает доступ к нему по внутреннему IP-адресу в кластере.
- *NodePort* – открывает сервис на том же порту каждого выбранного узла в кластере и делает сервис доступным вне кластера;
- *LoadBalancer* – создает внешний балансировщик нагрузки, назначает ему внешний постоянный IP-адрес.
- *External Name* – открывает доступ к сервису по заданному имени, содержащемуся в поле *externalName*.

Поле *selector* позволяют задать, какой класс подов будет использоваться в описываемом сервисе. За классификацию подов отвечают лейблы, которые позволяют задать класс, к которому принадлежит текущий под.

Схема работы сервиса проиллюстрирована ниже (Рисунок 4).

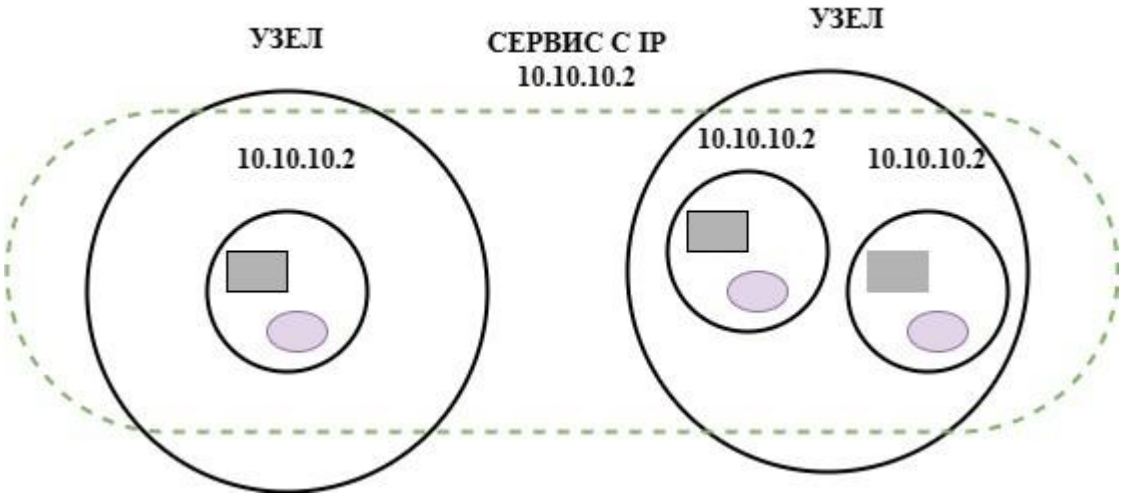

Рисунок 4 – Устройство сервиса в Kubernetes

На рисунке видно, что хотя поды и находятся на разных узлах, с помощью сервиса возможно логически объединить их в одно целое.

3. АЛГОРИТМ ЗАПУСКА ВЫДЕЛЕННОГО СЕРВЕРА UNREAL ENGINE

Для дальнейшей работы с развертыванием приложения на любом хостинге для начала необходимо скачать исходные файлы Unreal Engine с официального репозитория Epic Games и выполнить сборку проекта. Это необходимо для того, чтобы была возможность собрать серверный вариант Unreal Engine приложения, так как версия движка, скаченная из лаунчера Epic Games, позволяет собирать только приложения Unreal Engine клиентского типа.

Для того, чтобы получить доступ к репозиторию Epic Games и склонировать оттуда исходный код Unreal Engine нужной версии, для начала необходимо привязать свой аккаунт на GitHub к учетной записи Epic Games (Рисунок 5).

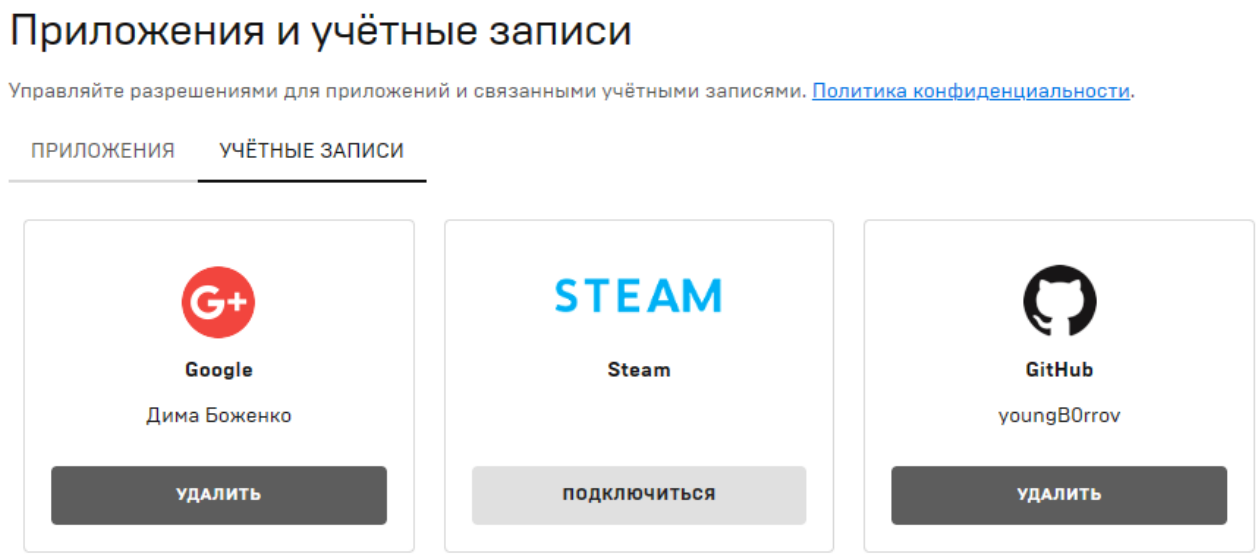

Рисунок 5 – Привязка аккаунта GitHub к аккаунту Epic Games

После клонирования исходных файлов движка из репозитория необходимо собрать новую версию движка из данных исходников. Для этого необходимо выполнить скрипт Setup, далее сгенерировать файлы для Visual Studio (должен появиться файл с разрешением .sln). После необходимо открыть решение в Visual Studio и собрать проект в конфигурации Development Editor. Сборка редактора Unreal Engine может занять довольно долгое время.

Далее после клонирования необходимо создать в уже существующем проекте новую цель сборки. Цель сборки – это файл с расширением .cs, который определяет, как в дальнейшем будет собираться приложения. Всего необходимо иметь две цели сборки: одна для клиентской части приложения, другая – для серверной. Пример содержания файла с целью сборки представлен ниже (листинг 2).

Листинг 2 – Содержание файла с целью сборки проекта

```
using UnrealBuildTool;
using System.Collections.Generic;

public class MultiplayerShooterServerTarget : TargetRules
{
    public TestProjectServerTarget(TargetInfo Target) :
base(Target)
    {
        Type = TargetType.Server;
        DefaultBuildSettings = BuildSettingsVersion.V2;
        ExtraModuleNames.Add("MultiplayerShooter");
    }
}
```

После копирования файла с целью сборки в проект, необходимо перегенерировать Visual Studio файлы проекта, выбрав в качестве версии уже не бинарную версию из лаунчера, а версию только что собранного движка из исходников.

После перегенерации файлов необходимо открыть проект в среде разработки и произвести сборку проекта с выбранной конфигурацией *Development Server Win x64*.

После успешной сборки проекта сгенерируются бинарные файлы для серверной части проекта.

Далее в среде разработки необходимо выбрать конфигурацию *Development Editor Win x64* и выполнить сборку. После удачно выполненной сборки откроется редактор, в котором уже можно будет настроить карты по умолчанию для сервера и для клиента. Также в редакторе необходимо будет по отдельности выполнить сборку клиентской и серверной части приложения.

При успешной сборке обеих частей создадутся исполняемые файлы для клиентской и серверной части, которые в дальнейшем необходимо будет запускать по отдельности.

При запуске уже собранного выделенного сервера можно указывать параметры запуска сервера в исполняемом файле, например, в файле с расширением *bat* для *Windows*. Файл должен содержать команду вида

<EXECUTABLE> [URL_PARAMETERS] [ARGUMENTS]

где вместо *EXECUTABLE* необходимо прописать путь к исполняемому файлу, например *UnrealEditor.exe* или *MultiplayerShooter.exe*. *URL_PARAMETERS* опционально, вместо него можно указать, какой уровень с какими параметрами можно загрузить, например, */Game/Maps/MainMap.umap?game=LobbyGameMode*. *ARGUMENTS* также опционально. Вместно них можно указать флаги запуска. Unreal Engine позволяет указывать как зарезервированные флаги, так и свои собственные. Очевидно, что для собственных флагов необходимо сделать обработку. Это можно выполнить с помощью статических функций *FParse::Param* и *FParse::Value* из заголовочного файла Engine\Source\Runtime\Core Public\Misc прямо в коде проекта. Ниже приведен пример, как можно задать параметры запуска отдельно для сервера и отдельно для клиента (листинг 3).

Листинг 3 – Задание параметров запуска выделенного сервера и клиента

```
MultiplayerShooter.exe MultiplayerShooter.uproject
/Game/Maps/MainMap.umap -server -port=7777 -log

UnrealEditor.exe MyGame.uproject 127.0.0.1:7777 -game -log
```

Первая строка предназначена для запуска выделенного сервера. В инструкции написано, что необходимо с помощью исполняемого файла *MultiplayerShooter.exe* запустить проект *MultiplayerShooter.uproject* в режиме выделенного сервера на карте */Game/Maps/MainMap.umap*. В качестве дополнительных параметров указывается, что необходимо запустить сервер на порту 7777 и выводить логи сервера в консоль.

Вторая строка предназначена для запуска клиента. В ней указано, что необходимо запустить проект в режиме клиента и сразу же подключиться по указанному адресу *127.0.0.1:7777*.

ЗАКЛЮЧЕНИЕ

В результате работы были рассмотрены два основных способа развертывания и масштабирования выделенного сервера, разработанного на Unreal Engine 4. Также хорошим дополнением к созданию развернутого и масштабируемого кластера будет созданный менеджер серверов Unreal Engine, с помощью которого можно будет отслеживать метрики каждого сервера и следить за его состоянием.

Из вышеописанных способов развертывания приложения способ, который предлагает AWS, является как наиболее простым для старта, так и надежным и отказоустойчивым. Это обоснованно тем, что основной задачей является внедрение плагина AWS в проект, что в дальнейшем позволяет запускать и управлять выделенными серверами проекта с помощью API плагина.

Способ с созданием масштабируемого кластера на Kubernetes является более предпочтительным, так как при разработке масштабируемо кластера с помощью Kubernetes появляется наиболее полный контроль над всем приложением. К минусам данного метода, можно, пожалуй, отнести, обязательное использование контейнеризации для дальнейшей работы с кластером.

Для более полного понимания специфики разработки и развертывания приложения в дальнейшем будут рассмотрены и реализованы оба метода развертывания. Также планируется создание простого менеджера выделенных серверов.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. What is a VPS. Amazon Documentation [Электронный ресурс] // Режим доступа: https://aws.amazon.com/what-is/vps/?nc1=h_ls (дата обращения 06.11.2023);
2. Learn Kubernetes basics. Kubernetes Documentation [Электронный ресурс] // Режим доступа: https://kubernetes.io/ru/docs/tutorials/kubernetes-basics/explore/explore-intro/ (дата обращения 6.12.2023);
3. Setting Up Dedicated Servers. Unreal Engine Documentation [Электронный ресурс] // Режим доступа: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Networking/HowTo/DedicatedServers/ (дата обращения 12.12.2023);
4. Dedicated server guide (Windows). Unreal Engine Community Wiki [Электронный ресурс] // Режим доступа: https://www.unrealcommunity.wiki/dedicated-server-guide-(windows)-cny2avzu (дата обращения 19.12.2023);
5. Containers Quick Start. Unreal Engine Documentation [Электронный ресурс] // Режим доступа: https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/Containers/ContainersQuickStart/ (дата обращения 20.12.2023);
6. Viewing pods and nodes. Kubernetes Documentation [Электронный ресурс] // Режим доступа: https://kubernetes.io/ru/docs/tutorials/kubernetes-basics/explore/explore-intro/ (дата обращения 21.12.2023);
7. Command-line Arguments. Unreal Engine Documentation [Электронный ресурс] // Режим доступа: https://docs.unrealengine.com/5.2/en-US/command-line-arguments-in-unreal-engine/#additionalparameters (дата обращения 21.12.2023).