|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РФ | | | | | | |
|  | |  | | |  | |
| Пермский государственный национальный  исследовательский университет | | | | | | |
|  | |  | | |  | |
|  | | ОТЧЕТ  по дисциплине «Технологии разработки распределенных приложений»  по теме «Разработка распределенной системы» | | |  | |
|  | |  | | |  | |
|  | Работу выполнили  студенты гр. ПМИ – 1,2  Никитас Николай,  Миков Роман,  Поселенцева Диана  29 апреля 2020 г. | |  | Проверил  ассистент кафедры МОВС  Постаногов И.С.  «\_\_\_» \_\_\_\_\_\_\_\_ 2020 г. | |  |
|  |  | |  |  | |  |
| Пермь, 2020 | | | | | | |

**1 Постановка задачи**

***Цель***: изучение методов коммуникации процессов в сети, а также средств динамического конфигурирования распределенных приложений.

***Проверяемые компетенции***: способность работы с информацией из различных источников, включая сетевые ресурсы сети Интернет, для решения профессиональных задач; способность применять на практике теоретические основы и общие принципы разработки распределенных систем; уверенное знание теоретических и практических основ построения распределенных баз данных; способность использовать на практике стандарты сетевого взаимодействия компонент распределенной системы.

Задание выполняется в группе (не более трех студентов) или индивидуально. Каждый студент отчитывается по каждому пункту задания индивидуально.

Требования к выполнению работы:

* + Приложение должно обеспечивать параллельную работу нескольких клиентов и серверов. Дополнительное требование: возможность запуска нескольких серверов на одном компьютере;
  + Клиентские приложения должны автоматически находить серверы для обслуживания и выполнения заданных функций;
  + Серверы системы могут выполнять различные функции;
  + При разрыве сеанса приложения должны автоматически восстанавливать свою работоспособность;
  + Для хранения данных и доступа к ним применить ADO и/или ADO.NET (или их аналоги);
  + Приложения должны поддерживать возможность взаимодействия в различных режимах;
  + Для организации взаимодействия нужно использовать различные средства коммуникации (именованные каналы, мейлслоты, сокеты, MSMQ, .Net Remoting, web-сервисы, WCF-сервисы), сравнив их возможности.

В программе **необходимо** использовать облачные технологии.

По окончании выполнения задания каждая группа студентов должна подготовить отчет.

Отчет по выполнению задания должен включать:

1. Общее описание приложения. Постановка задачи, введение в предметную область.
2. Архитектура системы. Обоснование выбора данного типа архитектуры распределенного приложения. Алгоритм работы приложения в целом.
3. Архитектура каждого из логических компонент системы (серверы, клиенты, диспетчеры). Подходы к реализации. Алгоритмы работы. Многопоточность, обоснование.
4. Методы коммуникаций компонентов системы (клиент→сервер, сервер→клиент и т.д.). Обоснование выбора этих методов коммуникации.
5. Способ передачи данных (синхронная / асинхронная, однонаправленная / двунаправленная и т.д.). Обоснование.
6. Структура передаваемых данных. Вид протоколов, обоснование выбора.
7. Отказоустойчивость системы. Как система поведет себя, если «исчезнет» один или несколько ее компонент. Что произойдет с системой, если «исчезнувший» компонент будет восстановлен на другом узле сети.
8. Работа с базой данных (если используется). Обоснование.
9. Исходный код приложений с комментариями.

**2 Описание приложения. Введение в предметную область**

Предметная область – данные о сетевых интерфейсах.

В настоящее время все больше и больше устройств подключаются к различным сетям, но пользователи редко задумываются о том, какая информация об устройстве становится доступна окружающим в этот момент и как она может повлиять на дальнейшую жизнь пользователя. Например, информация об открытых портах на одном из сетевых интерфейсов пользователя.

На первый взгляд, знания о том, кем и на каком порту слушаются запрос – бесполезна, однако, в руках специалистов, эти знания могут облегчить получение несанкционированного доступа к компьютеру, либо дать информацию о запущенных на компьютере программах злоумышленнику.

Если же все службы, слушающие порты, запущенные на устройстве хорошо защищены от атак, то пользователь может не бояться атак, но хорошая защита всех открытых интерфейсов в жизни встречается далеко не всегда: люди держат открытый порт для SSH с доступом по стандартному паролю, держат открытым FTP сервер с доступом в корневую папку и т.д.

Попытке заставить таких пользователей задуматься о своей безопасности была посвящена данная работа.

В качестве распределенной информационной системы была разработана программа-шпион, которая запускается на устройстве пользователя, сканирует все сетевые интерфейсы, порты, слушающие эти порты службы, и отправляет их на сервер, где злоумышленник сможет их увидеть и воспользоваться в своих корыстных целях, в то время как само приложение закроется, и пользователь не заметит, как часть его данных утекла.

Кроме функций шпиона, в системе присутствует функция DDOS-атаки. Каждый зараженный компьютер может отправлять запросы на указанные владельцем системы адреса и, тем самым, мешать работе указанного владельцем сервера.

**3 Архитектура системы**

На рисунке 1 изображена архитектура рассматриваемой распределенной системы.

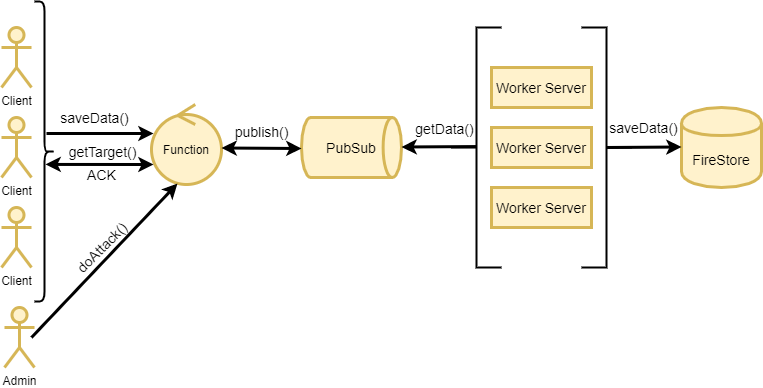


Рисунок 1 – Архитектура распределенной системы

Алгоритм работы данной распределенной системы состоит из следующих шагов:

1. Клиент выполняет сбор информации об устройстве.
2. Клиент отсылает собранную информацию диспетчеру.
3. Диспетчер перенаправляет данные от клиента в очередь сообщений Google Pub/Sub.
4. Исполнитель считывает данные из Google Pub/Sub и отправляет их в БД - Google Firestore.
5. Клиент подключается по WS к диспетчеру и ожидает адреса, на которые нужно слать запросы.
6. Администратор посылает запрос к диспетчеру, в котором содержится информация о целевом адресе, типе подключения, содержимом запроса и количестве запросов. Данное действие генерирует в очереди сообщений такое количество новых сообщений, какое указал пользователь в поле количества запросов.
7. Диспетчер считывает из очереди сообщения и передает их клиентам.
8. Клиенты совершают запросы.

**4 Архитектура клиента**

Клиент представляет собой коммуникационный модуль и модуль логики. Модуль логики собирает информацию о сетевых интерфейсах устройства, а коммуникационный модуль через REST API связывается с функцией-диспетчером и передает эту информацию. Затем клиент устанавливает соединение по WS, в котором ожидает адреса, на которые нужно слать запросы. Саму схему архитектуры клиента можно увидеть на рисунке 2.

WS

Модуль логики

Коммуникационный модуль

Клиент

Функция-диспетчер

HTTP/REST

Коммуникационный модуль

Рисунок 2 – Архитектура клиента

**5 Архитектура сервера**

Серверная часть ИС состоим из набора компонентов: диспетчер, очередь сообщений, исполнители и база данных.

Диспетчер содержит в себе 2 коммуникационных модуля, один отвечает за обмени данными с клиентами (по принципам REST API), а второй отправляет данные, полученные от клиентов, в очередь сообщений Google Pub/Sub.

Затем, из этой очереди сообщения считывают исполнителями и записываются в базу данных Google Cloud Firestore, из которой они могут быть извлечены злоумышленниками для осуществления коварных планов.

Кроме того, диспетчер позволяет создавать сообщения с целями атаки через REST API, сохраняя их в очередь сообщений и перенаправлять их из Google Pub/Sub клиентам по WS.

Архитектура сервера изображена на рисунке 3.

.

Модуль работы с БД

PubSub

Функция-диспетчер

Исполнитель

Модуль взаимодействия с Pub/Sub

Коммуникационный модуль

Модуль взаимодействия с Pub/Sub

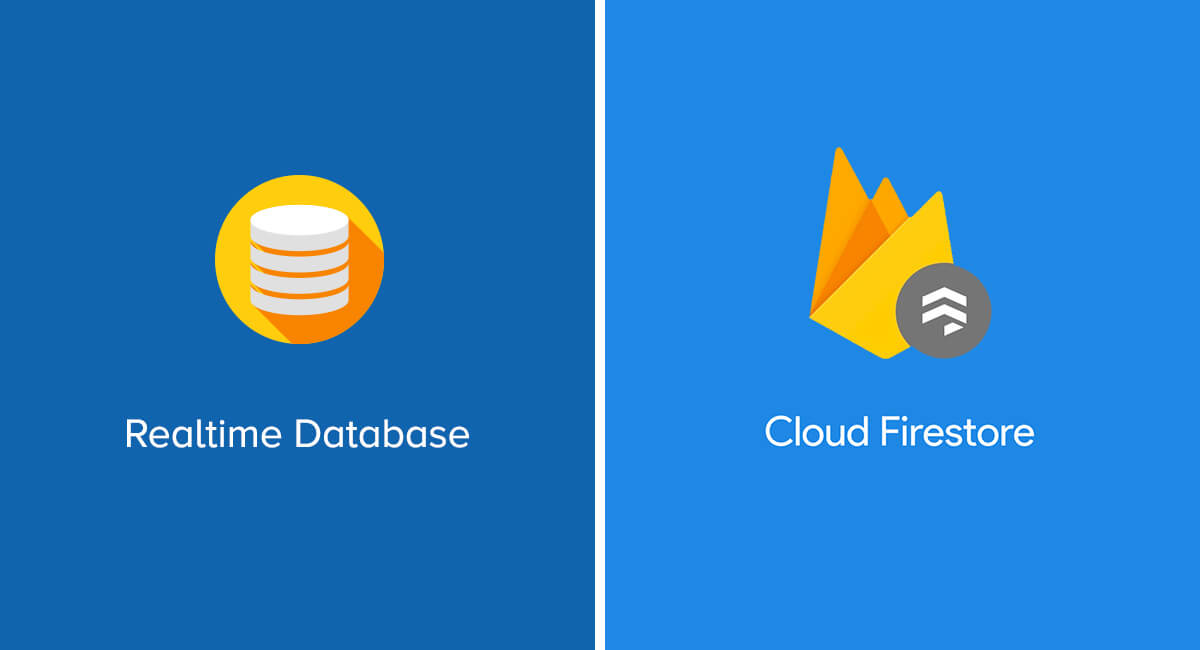


Рисунок 3 – Архитектура сервера

**6 Google Cloud Firestore**

Для хранения информации была использована облачная NoSQL база данных – Google Cloud FireStore. Взаимодействие БД с сервером происходит за счет REST API (протокол HTTP, JSON). Схема взаимодействия представлена на рисунке 4.

Исполнитель

REST API

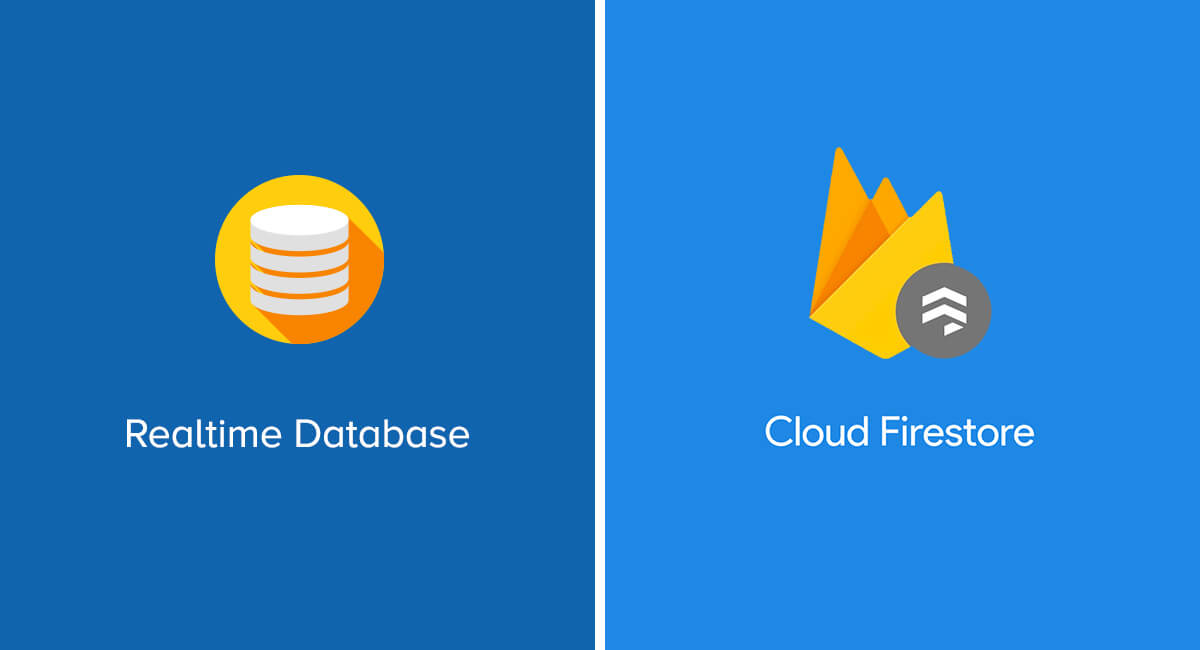


Рисунок 4 – Схема взаимодействия сервера с Cloud Firestore

**7 Структура передаваемых данных и работа с БД**

Сама БД не имеет жесткой структуры, т.к. используется NoSQL подход. Пример структуры передаваемых данных:

{"ExternalIP":"172.17.0.5:14558","ID":"24007ed2-2b90-4d6d-adc2-e55b7df96c7d","Ifaces":[{"Iface":"lo","Addresses":[{"IP":"127.0.0.1","OpenedPorts":[{"Port":631,"Service":"cups"},{"Port":902,"Service":"\u003cunknown\u003e"}]}],"Err":null},{"Iface":"enp2s0","Addresses":[{"IP":"192.168.31.27","OpenedPorts":[{"Port":902,"Service":"\u003cunknown\u003e"}]}],"Err":null},{"Iface":"virbr0","Addresses":[{"IP":"192.168.122.1","OpenedPorts":[{"Port":902,"Service":"\u003cunknown\u003e"}]}],"Err":null},{"Iface":"docker0","Addresses":[{"IP":"172.17.0.1","OpenedPorts":[{"Port":902,"Service":"\u003cunknown\u003e"}]}],"Err":null},{"Iface":"tun0","Addresses":[{"IP":"10.115.206.74","OpenedPorts":[{"Port":902,"Service":"\u003cunknown\u003e"}]}],"Err":null}],"Err":null}

Структура содержит информацию об открытых портах одного сетевого интерфейса пользователя, которые нашла система.

ExternalIP – IP-адрес клиента, ID – ID клиента, Iface – сетевой интерфейс пользователя, IP – IP-адрес устройства, Port – один из открытых портов сетевого интерфейса, который система смогла найти, Service – служба, которая слушает данные порты.

**8 Отказоустойчивость**

Отказоустойчивость системы гарантируется, прежде всего, делегированием ответственности за решение проблем, связанных с выходом из строя компонентов, на Google Cloud и те средства, которые используются данной платформой для защиты от неожиданного поведения системы. Рассмотрим подробнее отказоустойчивость каждого компонента.

Клиент гарантирует свою корректную работу за счет системы поллинга диспетчера, в случае отсутствия от того ответа, процесс повторяется каждые 10 секунд. При разрыве WS соединения попытки повторное подключения выполняются с тем же интервалом.

Так как диспетчер представляет собой экземпляр Cloud App Engine, то его работоспособность гарантируется платформой Google Cloud. При падении приложения, либо отсутствия связи, контейнер с приложением перезапускается на другом узле, доступном по тому же адресу, что и раньше. Сообщения, отправленные диспетчеру для атаки, хранятся в Google Pub/Sub и регулируются механизмом подтверждения приема, описанным ниже.

В работе с очередью Google Pub/Sub, отказоустойчивость так же гарантируется платформой Google Cloud и ее встроенными средствами.

Корректность работы исполнителя выполняется за счет использования механизма подтверждения приема сообщения из очереди (так называемого ACK). Исполнитель считывает сообщение из очереди и записывает его в БД. Если в какое-то из этих действий происходит ошибка, то подтверждение приема сообщения не происходит, и оно обрабатывается следующим исполнителем.