## На AWS произвести деплой всего приложения сборки по dist, в том числе прогнать JMeter-тесты (с возможной доработкой) чата приложения на базе Fargate 1.40

Крупный проект – на стадии mvp релиза с ветками для деплоя на github (предоставляется код для изучения темплейтов, баш-скриптов и прочего, так как devops часть реализована и требуется ее доработка, при необходимости и на усмотрение специалиста devops – так как есть автоматизация сборки через сервисы AWS типа: балансировщика, пайплайна и так далее, за исключением сборки через Travis CI), реализованный через паттерн: MVC на базе фреймворка laravel с доработками по модулям php, согласно prs стандарта – для отдельных функциональных мощностей сервиса на базе mvp части проекта. Данный проект необходимо запустить в dev-деплое с проведением нагрузочного тестирования отдельного модуля приложения: чата, через предоставленные готовые конфиги jmeter – которые также можно доработать при необходимости. Тесты должны быть выполнены с условиями покрытия 300к и 500к сообщений/сек – при наличии 300 тасок фаргейта и инстансов для БД: redis (rds), mongo, mysql – типов m5/r5 – далее приводится принципиальная каноническая схема проекта, которая может быть детализирована при совместном обсуждении. Общий смысл работы чата: SSE через WS на базе чата – далее приводится app flow чата, который и рассматривается в данной задаче. Проект не монолитный, построен через принципы облачного решения, включая горизонтальное/вертикальное скалирование через кластер. Политики ИБ реализованы и предопределены по ролям IAM в политиках групп, включая настройки на стороне темплейтов.

Все необходимые ресурсы имеются на руках для выполнения условий задачи, все условия со стороны кода бекенда, фронтенда по оптимизации и доработки- включаются после проведения нагрузочного тестирования в последующие версии проекта. То есть на сейчас имеется конечный продукт, который требует проверки по типам инстансов и получения результатов, согласно базы чата.

Обращаю внимание, что необходим человек, кто умеет работать с AWS и знаком с технологией fargate, то есть работал ранее и понимает как идет работа с EC, ECS2, тасок fargate, cloudwatch, pipeline, loadbalancer и etc. Технология применения идет с марта 2020 года. То есть специалист, который готов разобраться в крупном проекте, помочь в решении вопроса с ним и при успехе взяться за дальнейшие задачи, которые будут ставятся в нем. Отмечу, что данное решение можно будет переработать далее и/или оптимизировать для следующих задач.

Вся работа проекта построена через темплейты, они имеются в репозитории на GitHub для изучения в отдельной ветке: aws_deploy_v3, где идет отдельная ветка коммитов для девопс части – несвязанная с основной веткой: crossdev. Отмечу, что: терроформирования на данный момент практически нет - так как интересовала срочность выполнения задачи самого devops lead, поэтому были применены другие скрипты автоматизации.

На базе действующего проекта реализуется деплой приложения на стороне http и https – где все составные части проекта работают стабильно и корректно, но под нагрузкой (в зависимости от типа инстанса m5/r5 для БД и после конечного числа нагрузки отвалилось сокетное соединение) происходят определённые проблемы - которые надо будет устранить методами devops – за счет железа, проект крутится и скалируется в соответствие с ожидаемыми нормами в 30% мощностей, но могут возникают проблемы с подкруткой инстансов по самим мощностям взависимости от выставленной нагрузки в самом jmeter-конфиге, как мы видели решение вопроса: 300 секунд поднимается нагрузка и 900 секунд она держится, но на усмотрение специалиста можно провести корректировки, при должном обосновании- где приходится "прибивать" задачи и переподнимать кластера с перепроверкой хелсчеков (с ними не возникали проблемы), инстансы БД - что бы подкрутить мощности железа под необходимые значения для проведения нагрузочного тестирования.

**Также:** реализовано 2 очереди в проекте – для нагрузочного тестирования, для проверки отправки сообщения в момент нагрузочного тестирования – которую надо перепроверить по

работоспособности и в момент отправки сообщения устранить проблему с рефрешем страницы, который приходится делать, что бы увидеть отправленное/доставленое сообщение.

**Суть общей проблемы:** надо получить необходимые значения при проведении нагрузочного тестирования (описаны выше), подкрутить инстансы БД и необходимое железо на базе fargate (повторение: тасков выдано 300), что бы получить конечные нужные результаты проведения нагрузочного тестировния. Цель достижения результата 300с/с и самый оптимальный вариант 500с/с.

**Цель:** запустить проект - оттестировать вторую очередь в чате, которая реализована не через приоритетность, а как отдельный механизм - через обработку в Redis – описание идет в app flow. Исправить проблемы с safety протоколом для второй очереди – если она возникнет, так как по нашем предположениям из-за нее происходит вынужденный рефреш страницы для рекконекта. Далее надо провести нагрузочное тестирование ресурса, для необходимых нам цифр - которые достигаются путем скалирования железа, где надо выкрутить под определенные значения сами инстансы БД (Redis, RDS, Mongo, MySql). Повторюсь, что имеется репозиторий с конфигами для проекта, отладочными темплейтами и необходимыми материалами, сами тестировочные материалы jmeter-конфига. Нужна оценка специалиста, рекомендации и непосредственное решение вопроса. Вопрос по сдаче будет закрыт, когда будет получено необходимое значение показателя Redis на метриках, Jmeter, совокупых метрик на AWS, в том числе по хитам/суммам и проверки через mongo connect (где CLI не нужно ставить в отдельное окружение), что обработались сообщений и записались из редис в монго, в том числе сообщения в рабочем варианте второй очереди, включая корректные таймстампы. *Фактически:* проект построен по такой топологии со стороны серверной архитектуры, проект не монолитный: http://joxi.ru/eAOQXMlIpN6p5m - то есть работа через таски и воркеры, на базе сонетного соединения. Вся остальная необходимая информация будет предоставлена в личном диалоге, при небходимости могут быть в беседе backlead, frontlead, teamlead, CTO и они ответят на все интересующие вас вопросы.

**Краткий повтор:** Сам хайлоад выдает мощности за счет скейлинга инстансов со стороны железа. Задача - разобрать с действующими лидами соответствующие конфиги, в том числе самого JMeter и провести нагрузочное тестирование - предоставив репорты на базе уже готового материала, метрик AWS и так далее. Весь необходимый материал будет выдан на руки по факту приступления к задаче самого исполнителя. Вся работа идет в лайв режиме с репортами промежуточных результатов.

**Условия:**

- работа с крупным и перспективным проектом;

- компетентное отношение к работе;

- соблюдение сроков выполнения задач, дааемое после анализа самим работником;

- оплата по факту выполнения задач

**Общие требования:**

- Умение вести адекватный структурированный диалог;
- Стрессоустойчивость;
- Соответствие заявленному стеку по CV (резюме);
- Регламентированное соблюдение сроков по реализации;
- Исполнительность;
- Умение самостоятельного решения поставленных задач, также под руководством teamlead и/или lead своих подразделений;
- Навыки работы с CI/CD, как пример: Travis;
- Умение работы с Git;
- ведение работы через Jira, Trello с использованием воркспейса: SLACK EMM;
- итерационные коммиты на git.

Схема работы чата (Application flow) в упрощенном виде:

- Приложение в браузере (далее Клиент) отправляет сообщение на URL-адрес, который привязан (alias на AWS Route53) на Load Balancer (Load Balancer распределяет нагрузку между серверами).
- - Приложение на frontend части браузера (далее Клиент) отправляет запрос сообщения части на url-адрес, который привязан (alias на AWS Route53) на load balancer (который реализован в виде AWS Application Load Balancer, Elastic Load Balancing V2)
-
- Application Load Balancer определяет свободный контейнер серверного обработчика (далее Сервер), который может обработать запрос, используя методологию Round Robin, выдает IP адрес для установки подключения, а также Cookie для формализации соединения с конкретным инстансом (Сервером), которое будет использоваться в дальнейшем.
- Клиент устанавливает WebSocket-соединение с сервером, используя механизм Sticky Session через HTTPS -протокол для отправки-получения сообщений (для этого привязывается Cookie, выданный Application Load Balancer для поддержания соединения через конкретный инстанс (Сервер))
- Сервер «говорит» Клиенту, что нужно обновить соединение, и Клиент делает это, с помощью JavaScript-библиотеки Autobahn (она же используется для отправки-получения сообщений) используя необходимые параметры, которые запрашивает сервер, далее на Клиенте соединение обслуживается библиотекой Autobahn.
- При получении сообщения серверный обработчик проводит все необходимые операции, далее помещает сообщение в очередь Redis (AWS ElastiCache), что позволяет реализовать асинхронное соединение между клиентом и сервером (это позволяет избавиться от задержки в соединении и обработке сообщений).
- При появлении сообщения в очереди Redis, выделенный серверный обработчик проводит следующие операции:
- «Достает» сообщение из очереди, обрабатывает его содержимое, в зависимости от типа содержимого - обрабатывает данные из определенных хранилищ (файлы и картинки - AWS S3, данные пользователя - AWS RDS), а также помещает запись о сообщении в базу данных MongoDB (AWS DocumentDB).
- Сообщение запрашивается из базы данных MongoDB, проводятся все необходимые операции (сериализация, добавление ссылок на вложения и т.п.), выполняется отправка сообщения конечным Клиентам пользователей (браузерам) посредством WebSocket-соединения.
- Рассылает оповещение о получении нового сообщения в определенном чате всем участникам чата (для чего используется ZMQ на стороне сервера, на стороне клиента (в браузере) для реализации используется Autobahn).

В Redis (база данных для хранения очереди) отсутствует техническая возможность вставлять сообщения не в конец очереди.

Ввиду описанного механизма работы серверной части приложения, невозможно обработать сообщение от определенного пользователя вне очереди отправки сообщений, т.к. очередь работает по принципу FIFO (First-In First-Out).

Т.е. если пользователь отправляет сообщение в некий чат, во время проведения нагрузочного тестирования, его сообщение попадает в очередь, и будет обработано в рамках позиции данного сообщения в очереди, ввиду того, что скрипты нагрузочного тестирования создают сообщения с высокой плотностью, в большом количестве, время отображения сообщения (отправленного пользователем) сдвигается до момента обработки всех сообщений, которые были отправлены скриптом нагрузочного тестирования ДО момента отправки сообщения пользователем. Эта проблема решена через вторую очередь.

## Схема работы чата (Application flow детальнее):

- Приложение на frontend части (далее Клиент) отправляет запрос сообщения части на url-адрес, который привязан (alias на AWS Route53) на load balancer (который реализован в виде AWS Application Load Balancer, Elastic Load Balancing V2)

- Application load balancer определяет свободный контейнер серверного обработчика (далее Сервер), который может обработать запрос (находится в статусе healthy), используя методологию round robin, выдает ip адрес для установки websocket подключения, а также cookie (sticky session) для формализации соединения с конкретным инстансом, которое будет использоваться в дальнейшем.

- Клиент устанавливает websocket соединение с сервером, используя механизм sticky session (привязывается cookie, который был выдан application load balancer для поддержания соединения через конкретный инстанс серверного обработчика) через протокол http, для отправки-получения сообщений.

- Сервер отвечает на клиент заголовком upgrade required, клиент обновляет соединение (используя библиотеку autobahn, также используется для поддержания соединения и отправки-получения сообщений) используя необходимые параметры, которые запрашивает сервер, далее соединение обслуживается библиотекой.

- При получении сообщения в websocket серверный обработчик проводит все необходимые операции по сериализации/десериализации, помещает сообщение в очередь Redis (AWS ElastiCache), что позволяет реализовать асинхронное соединение между клиентом и сервером, а также избавиться от задержки в соединении и обработке сообщений

- При появлении сообщения в очереди Redis, выделенный серверный обработчик проводит следующие операции:

  - Достает сообщение из очереди, обрабатывает его содержимое, в зависимости от типа содержимого - обрабатывает данные из определенных хранилищ (файлы и картинки - AWS S3, данные пользователя - AWS RDS) - а также помещает запись о сообщении в MongoDB (AWS DocumentDB)

  - Рассылает оповещение о получении нового сообщения в определенном чате всем членам чата, для реализации возможности broadcasting сообщения посредством ZMQ/autobahn.

  - Сообщение запрашивается из базы MongoDB, проводятся все необходимые message processing операции, в т.ч. сериализация, добавление ссылок на вложения и т.п., выполняется отправка сообщения конечным клиентам пользователей посредством websocket соединения.

Ввиду описанного механизма работы серверной части приложения, невозможно обработать сообщение от определенного пользователя вне очереди отправки сообщений, т.к. очередь работает по принципу FIFO (First-In First-Out).

Т.е. если пользователь отправляет сообщение в некий чат, во время проведения нагрузочного тестирования, его сообщение попадает в очередь, и будет обработано в рамках позиции данного сообщения в очереди, ввиду того, что скрипты нагрузочного тестирования создают сообщения с высокой плотностью, в большом количестве, время отображения сообщения (отправленного пользователем) сдвигается до момента обработки всех сообщений, которые были отправлены скриптом нагрузочного тестирования ДО момента отправки сообщения пользователем. Эта проблема решена через вторую очередь.

**Обязанности по позиции «DevOps Engineer»:**

1. Оценка объемов, масштаба и сроков выполнения поставленных и декларированных работ, в зависимости от разбиения поставленных задач на спринты в установленные сроки и итерации по ним.

2. Выполнение данных задач и их оценка, с непосредственным разбиением на демонстрацию результатов в итерациях, чтобы показывать реализацию выполненных заданий итерационно и последовательно в зависимости от выполненных задач, относительно собственной работы.

3. Ведение документации по реализованным задачам и следующим поставленным задачам в зависимости от БП, также ведение ежедневного отчета по выполненным задачам через «**Slack EMM»** и «**Trello»** – с непосредственным информированием руководителя.

4. Построение цикла разработки, а также планирования собственной нагрузки на реализацию проектных частей по основным и второстепенным БП – от поставленного проекта.

5. Обеспечение высокого качества продукта при разработке и вводе в промышленную эксплуатацию, через сервис «**GitHub»**, также тестирования и покрытия кода тестами, и проверки итераций в спринтах, в том числе от работы с багами, фризами и иными проблемами в момент выполнения разработки проектных частей.

6. Обеспечение технического качества кода проекта, посредством проведения «**Unit»**-тестов, структурированного кода, рефакторинга кода, подчинения непосредственному руководителю и вводом в промышленную эксплуатацию продукта.

7. Разработка кодовой базы, также непосредственное знание технологического стека, под который выделяется поставленная задача в зависимости от проектной реализации, выбор технологий и контроль стека по разработке проектов, согласно технического задания и технических требований.

8. Автоматизация цикла разработки с помощью работы в **«VCS»**, **«CI/CD»** и автоматизации «**GitHub»** релизов.

9. Менторство над подчиненными, направление их в необходимом русле для решения поставленной задачи, и формирование технической документации для них – также вместе с ними, по установленным требованиям организации, а также: обмен опытом между членами команды, с целью повышения эффективности, понимания и совершенствования навыков, - повышения квалификации. В том числе рассматривается вариант от обратного, где идет такое подчинение под вышестоящими лицами по иерархии в организации.

10. Контроль и итерационное выполнение поставленных лабораторных работ внутри организации, для повышения собственной квалификации.

11. Проведение технической экспертизы по собственной работе, посредством сервиса **GitHub** с применением смежных сторонних сервисов для выполнения оптимизационно-итерационного подхода при решении поставленных задач в проекте.

12. Проведение тестирования и контроля выполнения собственных задач, которые были делегированы от вышестоящего руководства и/или Заказчика.

13. Разработка и проектирование архитектуры, структуры проектных частей, сбор технических требований и выполнения в полной мере поставленных задач из технического задания к проектной части.

14. Коммуницировать с каждым сотрудником, мотивировать команду, в том числе на собственном примере.

15. Проведение регулярных внутрикомандных совещаний по методике **«Scrum».**

16. Контроль версионности и реализации проектной части от поставленной задачи внутри проекта.

17. Применение собственных навыков и знаний для разработки проекта в коллаборации с сотрудниками персонала.

18. Возможность вносить определенные коррективы в проекте при необходимости для оптимизации рисков и решения поставленных задач, для того, чтобы достичь поставленного

результата в конкретные сроки для проекта – с уведомления и разрешения вышестоящего руководства.

19.Производить контроль по исполнительности в конкретные временные рамки итерации в спринтах по поставленным задачам.

20.Выполнять поставленные формализованные требования организации.