

Санкт-Петербургский государственный университет

Хажоян Левон Владимирович

Выпускная квалификационная работа

Повышение стабильности и рефакторинг CI/CD краудсорсинговой платформы

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование
информационных систем»*

Основная образовательная программа *СВ.5162.2021 «Технологии программирования»*

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н. Д.В. Луцев

Консультант:
руководитель группы, ООО «Яндекс.Технологии» А.С. Писарьков

Рецензент:
руководитель группы, ООО «Яндекс.Технологии» А.А. Петров

Санкт-Петербург
2025

Saint Petersburg State University

Levon Khazhoyan

Bachelor's Thesis

Improving stability and refactoring CI/CD of the crowdsourcing platform

Education level: bachelor

Speciality *02.03.03 «Software and Administration of Information Systems»*

Programme *CB.5162.2021 «Programming Technologies»*

Scientific supervisor:
C.Sc., docent CS D.V. Lutcv

Consultant:
Group leader at “Yandex.Technologies” A.S. Pisarkov

Reviewer:
Group leader at “Yandex.Technologies” A.A. Petrov

Saint Petersburg
2025

Оглавление

1. Введение	4
2. Постановка задачи	6
3. Обзор используемых технологий и «9999»	7
3.1. Используемые технологии	7
3.2. Текущее состояние платформы и требования «9999» . . .	7
4. Метрики работоспособности	13
4.1. Главный пайплайн исполнителя и заказчика	13
4.2. RED метрики	14
4.3. USE метрики	17
5. Формализация улучшений	18
5.1. Проблемы с CI/CD	19
5.2. Требование выдержать потерю дата-центра	21
6. Реализация	23
6.1. CI/CD	23
6.2. Учения по закрытию дата-центра	26
7. Анализ соответствия требованиям	28
8. Заключение	29
Список литературы	30

1. Введение

Краудсорсинг (Crowdsourcing) [2] – это привлечение к решению тех или иных проблем инновационной производственной деятельности широкого круга лиц для использования их творческих способностей, знаний и опыта по типу субподрядной работы на добровольных началах. В последние годы краудсорсинг стал важным инструментом для решения множества задач, связанных с обработкой данных. Он позволяет крупным компаниям делегировать конкретные задачи широкой аудитории, тем самым увеличивая масштаб деятельности и скорость её выполнения.

На текущий момент главной задачей краудсорсинга в Яндексе является обеспечение процесса пользовательской разметки данных - о товарах, магазинах, дорожных маршрутах, вариантах перевода с одного языка на другой и пр. Эти размеченные данные используются нейросетями для улучшения работы алгоритмов целевых сервисов Яндекса - Яндекс.Маркет [10], Яндекс.Такси, Яндекс.Карты. «ЯНГ» и «Яндекс Задания» — два сервиса, которые в совокупности представляют собой краудсорсинговую платформу компании Яндекс. Первый сервис используется внутренними заказчиками, второй – внешними. Платформа предоставляет общий (единый) интерфейс для сервисов Яндекса, предназначенный для работы с разметкой. Кроме того, платформа должна обеспечивать корректное хранение, ухаживание и доступ к данным о разметке, что является непростой задачей ввиду сложной логической модели данных и их большого объема.

Краудсорсинговая платформа компании Яндекс существует и используется почти 10 лет. Однако на момент моего вступления в этот проект (2 года назад) резко сменилась команда разработки этой платформы. И перед новой командой встали задачи поддержки работоспособности текущей функциональности, реализация новых требований, перепроектирование отдельных подсистем платформы, значительному улучшению быстродействия, надёжности и стабильности платформы. В частности, в начале работы новой команды приходило до нескольких

тысяч жалоб пользователей в 1 день, в настоящий же момент ситуация значительно улучшилась.

Команда разработки платформы получает значительное количество жалоб от пользователей на всевозможные проблемы и ошибки (до нескольких тысяч жалоб за один день). Эти жалобы в основном связаны со сбоями, которые препятствуют процессам разметки и обработки обращений в поддержке. Данные процессы являются критическими для компании, поэтому необходимо оперативное восстановление привычной работы сервиса и повышение стабильности платформы.

Внутренний фреймворк компании Яндекс под названием «9999» предназначается для измерения пользовательской доступности тех или иных сервисов. Кроме того, он включает концепцию надёжности сервиса с помощью так называемых тиров надёжности (от А до D). У бизнеса сложились требования по тире доступности сервиса, уровень которого определяется уровнем доступности в процентах и дополнительными требованиями, которые зависят от тира.

В рамках работы будет сказано, как автор добивался тира С у сервисов ЯНГ и Яндекс Заданий, а именно сосредоточился на многих аспектах повышения стабильности системы, включая выведение ключевых технических метрик, улучшение системы сигналов и алёртов, а также разработку и внедрение оптимизированного и прозрачного для пользователей релизного процесса. Кроме того, акцент будет сделан на автоматизации, сокращении времени на деплой и тестирование, а также на улучшении взаимодействия с командами тестирования и поддержки, повышении доступности платформы и достижении переживания отключения целого дата-центра.

Такой подход позволит не только повысить стабильность и надёжность платформы, но и значительно сократить время реакции на инциденты, что в итоге приведет к значительной экономии ресурсов и повышению удовлетворенности пользователей. В данной работе будет подробно рассмотрен процесс внедрения улучшений и достигнутые результаты.

2. Постановка задачи

Целью данной учебной практики является повышение стабильности и рефакторинг CI/CD краудсорсинговой платформы.

1. Провести обзор используемых технологий и требований фреймворка «9999».
2. Выработать и внедрить процесс сбора метрик работоспособности главной функциональности платформы.
3. Сформулировать предложения по улучшениям на основе анализа собранных метрик и других требований.
4. Реализовать и внедрить улучшения в платформу.
5. Проанализировать соответствие платформы требованиям фреймворка «9999».

3. Обзор используемых технологий и «9999»

3.1. Используемые технологии

Краудсорсинговая платформа – это две крупные высоконагруженные распределённые системы с общей кодовой базой. Первая – ЯНГ, внутренняя платформа для сотрудников Яндекса, т.е. только внутренних заказчиков и исполнителей, которые подписывают трудовой договор с компанией. Вторая – Яндекс Задания, внешняя платформа для внешних и внутренних заказчиков, т.е. не только заказчики из Яндекса, исполнителем может быть любой самозанятый человек в РФ. Платформа реализована с помощью микросервисной архитектуры, имеющей 32 общих микросервиса, с 33 и 35 микросервисами соответственно. Её серверная часть представляет собой REST API¹, но некоторые микросервисы также общаются между собой через RPC². В качестве основных языков программирования используются Java и Kotlin. Среди СУБД, используемых серверной частью платформы, можно найти PostgreSQL, MongoDB, Redis, YDB, S3, ClickHouse. Yandex Planner – система управления ресурсами кластера, ближайшим аналогом которой является Kubernetes[4], только без деплоя. Деплой реализован в сервисе Yandex Deploy. Данная платформа разрабатывалась более семи лет другой командой разработчиков и досталась команде, в которой участвует автор данной работы, на момент начала её написания.

3.2. Текущее состояние платформы и требования «9999»

Тип D – это ожидаемый план, но также бизнесу очень нужен тип C, как минимум на следующие полгода, потому что тип C – это общее

¹REST API – архитектурный стиль разработки API веб-приложений или компонентов распределённого приложения, используя протокол HTTP.

²RPC (Remote Procedure Call) – это метод, позволяющий программам вызывать функции или процедуры, которые выполняются на удалённом сервере, как если бы они выполнялись локально в рамках одного процесса.

требование для сервиса, после которого на него можно выставить более конкретные ограничения. Поэтому автор нацелился сразу на тир С, т.к. более приоритетных технических проектов перед платформой не стояло.

Рассмотрим данные требования и сформулируем общий план достижения:

3.2.1. Сбор метрики доступности сервиса

Существует множество методологий для сбора метрики доступности. В крупных компаниях, таких как Google, Amazon или Microsoft, практики и принципы расчёта этой метрики в облачных сервисах довольно схожи. Из всех методологий сбора метрики доступности[?], представленных в источнике, на реализованном в Яндексе фреймворке 9999 идентично рассчитывается временная метрика, которая рассчитывается как отношение $u * 100 / (u + d) \%$, где u – время, когда сервис был доступен, а d – время, когда сервис был недоступен. Как альтернатива выступает совокупная доступность, которая рассчитывается как отношение s/t , где s – количество успешных запросов, а t – количество всех запросов, но, как и отмечено в источнике, не все запросы должны эквивалентно влиять на метрику, отсутствие одной функциональности может сказываться на пользователе намного хуже, чем другой. Данная метрика рассчитывается по системным метрикам, наложенным на всю функциональность с весами, которые выбирает сам продукт. Первым этапом являются сами базовые системные метрики.

Для сбора метрик нужно внедрить систему мониторинга и систему оповещений с эскалациями, а также сформулировать метрики работоспособности главной функциональности платформы. Целевыми метриками для распределённых систем с микросервисной архитектурой являются метрики, собранные по методологиям RED[3] и USE[8].

RED (Rate Errors Duration) — эта методология описывает три основных метрики, предоставляющие информацию о запросах в сервисе. RED выделяет следующие метрики:

- Rate — количество запросов в единицу времени. Данная метрика хороша тем, что по ней можно отслеживать, что сервис находится в рабочем состоянии, то есть что приходят запросы и уходят какие-то ответы.
- Errors — количество ошибок в единицу времени. Данная метрика также очень важна. Она позволяет выявить и предупредить ошибки при развертывании новых версий приложения.
- Duration — средняя продолжительность обработки одного запроса. Также одна из главных метрик, так как резкое изменение значения данной метрики, говорит, о том, что возникли проблемы с производительностью и нужно смотреть на другие метрики (например физических ресурсов), чтобы выявить причины.

Выставить мониторинги на работоспособность общей инфраструктуры по методологии USE, которая выделяет следующую информацию о ресурсах, которую нужно отражать:

- Utilization (утилизация) — среднее время или процент использования ресурса (CPU, память, сеть). Таким образом мы можем понять, насколько загружен ресурс.
- Saturation (насыщение) — степень перегруженности ресурса, то есть количество отложенной или поставленной в очередь работы. Данная информация помогает выявить и предупредить нехватку ресурсов.
- Errors (ошибки) — количество ошибок, связанных с использованием данного ресурса (например, количество ошибок при записи информации на диск).

Собраным метрикам функциональностей нужно выставить нужные веса того, как же они должны влиять на общую метрику доступности сервиса. Такое определяют менеджеры и сам продукт. Для того чтобы команда могла повышать данную метрику, автор завёл регулярные дежурства по инцидентам и план действий, по которому дежурный

должен следовать во время инцидента, опираясь на практики управления инцидентами в крупных компаниях[7]. Примерные принципы и практики, которые автор реализовал в платформе:

- Остановите поломку, восстановите сервис и сохраните доказательства первопричины;
- Заранее разработайте и задокументируйте свои процедуры управления инцидентами в консультации с участниками инцидента;
- Предоставьте полную автономию в рамках назначенной роли всем участникам инцидента;
- По окончании инцидента сформулировать задачи по предотвращению его повторения, обсудить с частью команды;
- Приоритезировать задачи по предотвращению повторения инцидентов.

3.2.2. Выдержать потерю одного дата-центра

Дата-центр в Яндексе де-факто является зоной доступности, потому что не существует зоны доступности с двумя дата-центрами, и условие выдержать потерю одного дата-центра это и предполагает. В популярных компаниях существует множество популярных практик, которые нацелены на повышение доступности сервиса и помогают выдержать потерю одного дата-центра. Среди них:

- балансировка нагрузки;
- репликация серверных компонентов;
- управление перегрузкой.

Есть много видов алгоритмов балансировки, каждое усложнение которого и взятие в уравнение расчёта пути запроса дополнительной переменной обычно положительно влияет на равномерность распределения запросов между машинами. Dynamic Weight Adjustment[9] – выступает

одним из самых востребованных алгоритмов балансировки для сервисов с большим числом серверов. Его принцип состоит в том, чтобы динамически перекидывать веса серверов в зависимости от нагрузки. В Яндексе используется свой алгоритм нагрузки, который является дополнением данного и использует внутри себя не только утилизацию, но и проверки работоспособности.

Политика балансировки нагрузки не решает все проблемы с нагрузкой, в конечном итоге какая-то часть вашей системы станет перегруженной[6]. Грамотное управление условиями перегрузки имеет основополагающее значение для работы надежной обслуживающей системы. Одним из вариантов обработки перегрузки является обслуживание ухудшенных ответов: ответов, которые не так точны или содержат меньше данных, чем обычные ответы, но которые легче вычислить. Например, можно опираться на локальную копию результатов, которая может быть не полностью актуальной, но которая будет дешевле в использовании, чем обращение к главному хранилищу. Также одной из самых частых практик является квотирование. Рассмотрим одни из самых частых групп квотирования:

- общие лимиты запросов на балансере;
- лимиты запросов на конкретного пользователя;
- лимиты конкретных запросов на конкретного пользователя.

При выборе количества и расположения реплик в базе данных нужно думать о следующих факторах:

- проживание пользователей и расположение доступных дата-центров;
- плата за использование машин и поддержку;
- консистентность данных;
- задержка ответов.

На практике, мы скорее никогда не можем посчитать, сколько же нам реплик будет нужно, потому что это неэффективно. Обычно начинают с малого, а именно один главный узел и одна реплика, а дальше увеличивают количество и проводят тестирование по потере зоны доступности, чтобы понять, что данного количества достаточно.

Как же провести тестирование потери дата-центра[1]? Нужно максимально правдоподобно с точки зрения всех клиентов отключить доступ к любой инфраструктуре в одном дата-центре. Из инфраструктуры в платформе представлены базы данных, различные виды балансеров, поды в Yandex Planner. Физически выключать сервера, на которых в данный момент развёрнуты виртуальные машины, у автора нет и быть не может, поэтому самый явный способ, как имитировать потерю питания – отключить сетевой трафик с помощью файрвола на всей инфраструктуре. Вся инфраструктура платформы размещена в своих подсетях, что делает возможным отключить только хосты, находящиеся в определённом дата-центре и принадлежащие платформе. Процесс также должен быть регулярным.

4. Метрики работоспособности

4.1. Главный пайплайн исполнителя и заказчика

Перед тем как начать выделение метрик, стоит выразить главную функциональность платформы в двух пайплайнах. Первый из которых — пайплайн заказчика:

1. Заказчик загружает задания (датасет) для разметки в некоторый пул (множество заданий). Указывает различные шаблоны для их отрисовки, входные и выходные данные.
2. Настраивает различные настройки пула или конкретных заданий внутри них, среди которых может быть цена, перекрытие (сколько раз нужно разметить одно из заданий разными исполнителями). Отмечает ханипоты (задания с заранее известным ответом), которые сделаны для того, чтобы вычислять тех, кто размечает задания не вникая в них. Таким часто занимаются те, кто пытается воспользоваться ботами для прохождения заданий.
3. Также выставляет навыки, статистика активности или качества разметки и прочие параметры, которыми должен обладать исполнитель, чтобы иметь возможность начать выполнять задание. Банальный навык – пройти инструктаж. Чуть менее банальный навык – владение определённым языком. Более редкий – знаток некоторой предметной области.
4. Запускает пул.
5. Выкачивает размеченный датасет и выплаты исполнителям после разметки.

Пайплайн главной функциональности исполнителя:

1. Исполнитель регистрируется, загружая нам все свои личные данные в виде ИНН, СНИЛС, фото паспорта и номер телефона. Этого пункта нет в ЯНГе, потому что данные получаются на момент устройства в компанию, регистрация в этом случае проще.

2. После верификации личности пользователем, исполнитель заходя на главную страницу, которую принято называть столом исполнителя, получает доступные ему задания. В проверке доступности участвует множество дополнительных факторов, неупомянутых в пайплайне заказчика, т.к. это не самая важная функциональность заказчика. Пример: заказчик иногда помимо навыков выставляет исполнителям смену, в которой они должны работать и какие задания им в эту смену приоритетнее выдавать на стол.
3. После формирования стола, исполнитель видит задания вместе с ценой, содержанием, описанием и инструкцией по выполнению.
4. Берётся за выполнение определённого задания. В это время формируется кастомный html-шаблон, предоставленный заказчиком и исполнитель начинает видеть задание.
5. Исполнитель имеет возможность пропустить задание.
6. После прохождения задания, исполнитель должен получить деньги на свой баланс и уметь их вывести, что не является основной функциональностью исполнителя.

4.2. RED метрики

Основные жалобы на работоспособность платформы были на высокую задержку ответа или на огромное количество ошибок, что отлично подтверждает внедрение методологии RED. Сам Rate в методологии RED нам скорее нужно держать ненулевым, т.к. никаких бизнес-требований к обработчикам у сервиса не имеется, поэтому дальше про данную метрику будет сказано не слишком много. Эти метрики сервис всегда должен держать на минимальных значениях. Т.к. схема общения пользователя с сервисами устроена через балансир, то и метрики количества запросов в секунду, высокую задержку ответа или количество ошибок тоже нужно брать с них. Сами балансеры предоставляются

Awacs³, и сигналы времени ответа, количества запросов и кодов ответа доступны по включению соответствующей настройки. Но их недостаточно, т.к. нужно агрегировать сигналы как минимум по дата-центрам, а как максимум по всем экземплярам балансиров и вычислять, например, нужные перцентили.

Эти сигналы нужно направлять в некоторую систему количественного мониторинга. В качестве целевого примера во фреймворке 9999 представлен Yandex Monitoring. Т.е. Awacs предоставляет для каждого балансера в каждом датацентре массив со временем ответа бегущих запросов по каждой конкретной конечной точке в данную секунду, а в Yandex Monitoring-е требуется увидеть метрику вида: конечная точка – 95-ая перцентиль по всем временам ответа всех балансиров в данную секунду. Реализуется это некоторыми конфигами на специальных языках запросов или скриптов, предоставляемыми Yandex Monitoring-ом и Awacs-ом. Спецификации скрыты под NDA.

Самой основной функциональностью являются два основных пайплайна заказчика и исполнителя. Причём пайплайн заказчика частично повторяется в формате API, а не только в UI. Ниже представлены группы конечных точек, для которых требуется завести сигналы времени ответа и кодов ответа. Многие конечные точки переиспользуются в UI/API пайплайнах, а также благодаря большому времени развития платформы накопилось большое количество версий API. В качестве примера, для формирования стола нужно создать сигналы времени ответа и количества ошибок по трём разным конечным точкам:

- GET /api/task-suite-pool-groups;
- GET /api/i-v2/task-suite-executions;
- GET /api/i-v3/task-suite-pools.

Т.к. у, к примеру, получения стола есть разные версии, то нужно распределить веса каждой конечной точки относительно функциональности, к примеру, взять долю среднего RPS относительно всех конечных

³Awacs – система управления облачной балансировкой в Яндексе.

точек функциональности. Подобные метрики нужно считать как единую, т.к. «открытие стола исполнителя» в данном контексте – это одна из многих функциональностей, которым разработчиками сервиса выставляется вес p_i . Нужны они как минимум для инфраструктурных метрик, но также очевидно, что разные функциональности имеют разный вес относительно пользователя: пользователь не может получить стол исполнителя – не то же самое, что и пользователь не может вывести деньги. Также разработчиками команды выписываются пороги срабатывания конкретных функциональностей, обычно также экспертно смотря на графики данного сервиса и решая, в какую перцентиль хочет целиться команда. В контексте весов метрика медленного ответа нет смысла считать относительно того, насколько она замедлилась, т.к. если, например, пользователь получает стол исполнителя более 5-ти секунд – он, не дожидаясь, идёт жаловаться в поддержку.

Также метрику на количество ошибок конкретной конечной точки стоит рассчитывать как отношение ошибок ко всем остальным ответам данной конечной точки в рамках секунды. Вот некоторые группы конечных точек, функциональностей в которых очень много и указывать их нет практического смысла: получение следующего задания, отрисовка шаблона, отправка задания, всё для администрирования пулов заказчиком. Рассмотрим пример. Представим, что стол исполнителя в 95 перцентиле стал показываться дольше заданного порога в 2 секунды на протяжении 5 секунд, а именно конечной точки `/api/i-v3/task-suite-pools`. Конечная метрика недоступности данной функциональности содержит несколько параметров:

- $p = 0.2$ – вес функциональности открытия стола, который в сумме от всех конечных точек должен составлять 1, у открытия стола он очень большой, т.к. это одна из самых главных функциональностей сервиса, у остальных вес может составлять 0.01;
- $m = 0.3$ – доля RPS данной конечной точки ко всем RPS конечных точек, у конечной точки `/api/i-v3/task-suite-pools` в среднем RPS равен 60-ти, а в сумме у трёх перечисленных конечных точек

отвечающих за получение стола исполнителя средний RPS равен 200;

- $k = 5$ – период недоступности метрики.

Соответственно, умножив эти параметры, мы получим недоступность, равную 0,2 секундам. Целевая метрика, которую назовём метрикой доступности сервиса, рассчитывается как $(p - q) / p$, где p – количество секунд в рассматриваемом периоде, q – сумма $n * m * k$ каждой из функциональностей в рассматриваемом периоде. В расчёте метрики ошибок есть дополнительный вес l , равный отношению количества ошибок на все за период недоступности. В самом фреймворке 9999 расчёты немного посложнее, но общий принцип таков.

4.3. USE метрики

Представленная в качестве целевого примера система количественного мониторинга в фреймворке 9999 – Yandex Monitoring имеет заготовленные шаблоны метрик для каждого вида ресурса в почти всей инфраструктуре Яндекса. Основная инфраструктура: сами микросервисы, базы данных, очередь сообщений. А значит, может отлично подойти для выделения метрик в рамках методологии USE.

Для самописной очереди сообщений, точно такие же, как и для общих микросервисов, и в дополнение:

- алёрты на совокупный размер из списков топиков;
- алёрты на задержки на чтение из топиков реального времени.

Полученную целевую метрику доступности сервиса нужно держать как можно выше, в идеале равной 100%. Автор работы дежурил очень часто, разобрав около 90% инцидентов за рассматриваемый период в полгода. В рамках данной работы ввиду длины не получится описать всё сделанное, поэтому был выбран один крупный проект по удалению архивных данных и большинство рефакторинга с CI/CD.

5. Формализация улучшений

По мере разбора инцидентов были выявлены основные проблемы в системе, которые влияли на стабильность. Их можно разбить на две группы. Проблемы с CI/CD:

- непрозрачные для пользователя релизы;
- нестабильное взаимодействие с внешними и внутренними системами;
- долгая доставка хотфиксов;
- нерегулярные релизы.

В рамках борьбы с непрозрачными для пользователя релизами будет доработан внутренний алгоритм балансировки, поэтому в рамках требования выдерживать потерю одного дата-центра останется реализовать:

- репликация серверных компонентов;
- управление перегрузкой;
- регулярные учения по закрытию дата-центра.

Проблемы стабильности системы во время исполнения:

- нестабильное взаимодействие с внешними и внутренними системами;
- перезаполнение кластеров пользовательскими данными и их неисправная работа при практически или полностью забитых дисках;
- неоптимальная работа с ресурсами;
- нестабильная работа сервиса во время регламентных работ или учений в одном из дата-центров;
- неправильное распределение нагрузки.

Проблемы с CI/CD на регулярной основе инициировали работу алёртов и создавали инциденты. Далее рассмотрим перечисленные проблемы более подробно, как они на самом деле влияли на пользователей, разработаем и реализуем улучшения.

5.1. Проблемы с CI/CD

5.1.1. Непрозрачные для пользователя релизы

В платформе на момент заведения мною этих алёртов релизы проходили непрозрачно для пользователей, а именно во время релиза пользователь встречал 504 коды ответов. Для того чтобы понять, почему так было, отмечу, что деплой в Яндексе происходит с помощью сервиса Yandex Deploy⁴, алгоритм развертывания в котором состоит в том, что, не прекращая запросы к машинке, на которой была развёрнута прошлая ревизия сервиса, развернуть на неё новую версию. Далее будут использоваться термины, которые идентичны представленным в документации Kubernetes[4]. Это поведение регламентируется алгоритмами балансировки, которые при стандартном конфиге продолжают отправлять запросы во время перекачивания на под новой версии или даже после зафейленной проверки работоспособности, если ещё не успела сработать проверка на чтение.

5.1.2. Долгая доставка хотфиксов

Несколько минут раскатывался сервис не всегда. Как только я пришёл в сервис, деплой трёх дата-центров занимал около 2 часов. Из-за чего 2 часа на сервис ожидается повышенная нагрузка, а также нет возможности задеплоить какой-то багфикс в кратчайшие сроки. Поэтому данная проблема тоже повышает метрику недоступности сервиса, когда требуется какой-то хотфикс, а его невозможно выкатить. Ускорять старт довольно большого приложения довольно тяжело, оптимизировать его можно очень долгое время, поэтому командой было выделено

⁴Yandex Deploy – сервис для оркестрации контейнеризированных приложений.

фиксированное 3 дня на ускорение старта и вот какие проблемы отладил автор:

- пропускная способность диска на всех кластерах была 2-4 мб/с, что при jar-файлах весящих более 400 мб, а местами и 600 мб, могло занимать около 5 минут только на выкачивание jar файла на диск;
- однопоточные разогревы на старте приложения, с неоптимальными запросами в базу данных внутри;
- у многих микросервисов был неправильный disruption budget[5], а именно количество одновременно выкатывающихся подов во время перекачивания контейнеров;
- некоторые контейнеров, которые требовали обновления довольно редко перекачивались каждую выкладку.

5.1.3. Нерегулярные релизы

Дежурств в команде в какой-то момент не существовало, релизы происходили сборкой приложения вручную, а именно без отведения релизной ветки, а сборкой докер-контейнера на главной ветке в системе контроля версий. Деплой через UI происходил указанием нужного контейнера в нужный микросервис.

Ручные релизы практически невозможно сделать регулярными, т.к. это отнимает большое количество времени разработчика. Поэтому деплой происходил при надобности и только одного микросервиса. Тем самым, иногда микросервисы не выкатывались несколько месяцев, что делало очередную выкладку очень опасной. Также, возвращаясь к проблемам с хотфиксами, т.к. отсутствовало автоматизированное отведение релизной ветки, приходилось сначала отводить ветку от последнего коммита в релизе и только потом коммитить туда фикс. Также, чтобы снизить вероятность плохого релиза, вместе с командой тестирования было принято проводить регрессионное тестирование на каждом из релизов.

5.1.4. Медленные тесты

Также, с тестами была проблема, что они прогонялись около 3-х часов, что, разумеется, при наличии нестабильных тестов, могло задерживать релиз как минимум на день. Путём профилирования приложения во время прохода тестов выявились основные причины медленности модульных и интеграционных тестов. Ниже они представлены одновременно с решением:

1. Малая гарантия сети, а именно случайная, из-за которой загрузка контейнеров могла быть очень медленная.
2. Выполнение многих тестов с базой данных не в транзакциях, из-за чего каждый раз происходил полный дроп тестовых баз данных и наполнения их стартовыми данными, что намного медленнее отката транзакции.
3. Исполнение тестов в одном модуле.

5.2. Требование выдержать потерю дата-центра

5.2.1. Управление перегрузкой

Здесь речь не о управлении перегрузкой инфраструктуры, а скорее о том, что пользователям нельзя давать в свободное пользование очень сложные запросы. Механизм квотирования операций в платформе существует, только за время сбора метрик стало известно о том, что несколько тяжёлых операций, запущенных пользователем, сломали один из микросервисов, который связан с расчётом аналитики. Поэтому автор снова провёл ревизию операций в платформе, которые теоретически могут сломать сервис, и выделил ещё две операции, которые стоит квотировать. Критерий таких операций был таков: если большое количество операций когда-либо загружало большую часть ЦПУ, то стоит такую операцию квотировать.

Также нужно считать общий лимит запросов на балансере и лимиты конкретных запросов на конкретного пользователя. Второй вид квоти-

рования тоже можно реализовать с балансеров, смотря на заголовки HTTP-запросов, в которых хранится уникальный логин пользователя.

5.2.2. Репликации и учения по закрытию дата-центра

Первой итерацией реплик в каждом дата-центре стоит здесь 3. С двух начинать нет смысла, т.к. при потере одного дата-центра останется только мастер, а в платформе есть много логики, которая на мастере и реплике. Например, запросы, которые исполняются только на репликах. Или механизм подтверждения данных из MongoDB, который не может эффективно работать на мастере и теряются данные. Поэтому первым шагом становится написать скрипты добавления дополнительных реплик для YP Planner и Yandex Cloud, где находятся базы данных платформы. А вторым шагом становятся регулярные учения по закрытию дата-центра. Данные учения должны уметь закрывать файрвол подсети баз данных и серверных бэкендов на заданное время. Для этого нужно увезти все базы данных в отдельную сеть и выдать все нужные сетевые доступы серверам и разработчикам. Для ЯНГа и Яндекс Заданий отдельно, для безопасности. Также уметь быстро открывать подсеть в случае сбоя. Под подсетью подразумевается именно подсеть одного дата-центра из всей сети баз данных платформы, т.к. в инфраструктуре Яндекса используется HBF⁵. Это даёт гибкость не создавать отдельные сети для хостов в каждом дата-центре и закрывать сетевой трафик хостам в одной сети только по признаку названия дата-центра. Под регулярными в 9999 подразумеваются учения раз в неделю. Также, чтобы разом не проваливать все запросы, которые пришли на балансеры и не успели уйти на бэкенды, желательно перед закрытием сети проставить веса балансеров в этих датацентрах в 0, к примеру, за минуту до закрытия HBF. Немало важно перед началом учений провести оповещение дежурных сервиса, чтобы те были готовы к резкому повышению нагрузки.

⁵Host Based Firewall – межсетевой экран, установленный на конкретном хосте, а не на всей сети

6. Реализация

6.1. CI/CD

Большинство решений с CI/CD выходило напрямую из проблем. Основной проблемой всегда выступала отладка, а также декомпозиция с состояния: «Деплой работает 2 часа» до «Деплой работает 5 минут». Чёткого регламента к скорости работы той или иной функциональности из CI не было, скорее только выделенное время на эту работу. Т.е. если до деплоя в 5 минут нужно было 2 недели работы, то до 3 минут могло выходить ещё 2 месяца. Так что работа была построена так, чтобы найти самые выгодные оптимизации, которые в дальнейшем реализовать, и если время выкладки перестанет регулярно влиять на пользователей, то можно оставить процесс. В свою очередь разработка ускорения тестов и оптимизации релизного процесса строились таким образом, чтобы удовлетворить команду. В отладке использовались инструменты Perf, FlameGraph и Yandex Planner. Также в реализации нового релизного процесса использовался инструмент Arcadia CI. Внутренняя документация закрыта.

6.1.1. Непрозрачные для пользователя релизы

Нужно научить балансировщик ретраить, если он получил 504 ошибку, с данной задачей и задачей пережить потерю дата-центра отлично справляется алгоритм балансировки Dynamic Weight Adjustment, а именно его модификация в Яндексе. Спецификации алгоритмов балансировки описаны в внутренней документации сервиса Awacs, используемый описывается довольно большим количеством параметров, вот самые важные из них в данном контексте: Сначала рассмотрим параметры алгоритма балансировки по дата-центрам:

- `weights_section_id`: `bygeo` – то, как дата-центры получают соответствующие им веса;
- `method`: `BY_DC_WEIGHT` – метод, который выбирает каждую

следующую попытку запроса по весам дата-центров;

- attempts: 3 – попытки выбрать нужный дата-центр, т.к. все сервисы жили в 3 дата-центрах, нужны 3 попытки на выбрать нужный.

Параметры алгоритма балансировки по подам в рамках дата-центра:

- attempts: 2 – количество стандартных попыток выбрать под в рамках одного дц;
- max_reattempts_share: 0.5000000029802323 – доля ретраев относительно всех попыток;
- fast_attempts_type: CONNECT_FAILURE_AND_503 – что считать быстрой попыткой, а именно попытка, которую можно ещё дополнительно ретраить в рамках выбранного датацентра;
- fast_attempts: 3 – максимальное количество быстрых попыток;
- retry_http_responses: codes: - «503» – какие коды ответов стоит ретраить;
- retry_non_idempotent: true – ретраить неидемпотентные запросы, что не соответствует RFC, но в текущей инфраструктуре единственное решение не фейлить запросы при перекачивании, разумеется, ретраить можно только те запросы, которые не создают новых сущностей.

Перекачивать при текущей схеме балансировки оптимально только по одному дата-центру, потому что если меньше – деплой релиза будет слишком медленным, если больше – деплой релиза будет слишком опасным, потому что может возникнуть ситуация:

- поды раскатились, но не прошли ещё все разогревы и проверки работоспособности не поднялись;
- т.к. поды раскатились, то начинаются выкатываться следующие;

- 2 дата-центра недоступны, что на самом деле уже довольно опасно.

Поэтому повышать уж точно нельзя, целевая картина – уметь жить при одном дата-центре хотя бы несколько минут, пока раскатывается сервис.

6.1.2. Долгая доставка хотфиксов

Как были решены отлаженные проблемы:

- пропускная способность диска на всех кластерах поднята до 20-40 мб/с через Yandex Planner скриптом;
- распараллелил разогревы на старте приложения, что ускорило поднятие контекста приложения вплоть до 10-ти раз, в зависимости от микросервиса;
- всем микросервисам увеличил disruption budget, а именно количество одновременно выкатывающихся подов во время перекачивания, чтобы он стал равным 1 дата-центру;
- добавил кеширование некоторых контейнеров, которые не менялись, в качестве примера, довольно редко менялся logbroker (модуль, который отвечал за то, чтобы поставлять в экспорт логи приложения), поэтому его не нужно было разворачивать и выкачивать снова.

После данных улучшений деплой начал проходить около 5 минут, что уже приемлемо для такого огромного сервиса. Дальше ускорять сильно тяжелее и неоправданно.

6.1.3. Ускоренные тесты

Ниже представлены решения ранее представленных проблем:

1. Была поднята гарантия сети через планировщик ресурсов, заранее рассчитав сколько понадобится.

2. Выполнение всех тестов с базой данных в транзакциях, для чего также были переработаны общие конфигурации для тестов.
3. Разделение тестов на модули и запуск этих модулей параллельно.

Также были устранены проблемы с нестабильными тестами, что в среднем ускорило итерацию тестов с 2-3 часов до 10 минут.

6.1.4. Новый релизный процесс

Ниже будет описан флоу нового релиза:

1. Отведение релизной ветки.
2. Прохождение модульных и интеграционных тестов и сборка всех микросервисов параллельно.
3. Создание релизного тикета с координацией дежурного тестировщика и разработчика.
4. Деплой в препродакшн окружение всех микросервисов параллельно.
5. В специально отведённое время и при наличии подтверждения от команды тестирования, дежурный нажимает ещё одну кнопку на деплой всех микросервисов параллельно в продакшн окружение, параллельно с чем отправляются во все сервисы координации сообщения о начале и конце релиза.

6.2. Учения по закрытию дата-центра

В рамках самих учений, чтобы смоделировать потерю дата-центра, поставим веса балансеров в тестируемом дата-центре в 0 с помощью инструмента Awacs – инструмент для управления балансировкой в Яндексе, чтобы не слишком мешать пользователям искусственными проблемами. Далее нужно закрыть сетевой трафик во всей инфраструктуре в этом дата-центре. Для этого сначала нужно вывести всё из ком-

мунальных сетей. Достаточно одной сети на один тип инфраструктуры. Потом закрыть в этой сети хосты в определённом дата-центре. Это возможно из-за того, что межсетевые экраны в Яндексе установлены на хосты, т.е. можно налить новые сетевые правила вида «Не принимай сетевой трафик на хост, если сам хост в тестируемом дата-центре». Требование состоит в реальных хостах, поэтому нужно написать скрипты, которые регулярно запускаются. Процесс должен быть достаточно быстро отменяемым, на случай если учения начали влиять на пользователей, т.е. также нужно написать скрипты на обратную выкатку сетевых правил. Во избежание дополнительной нагрузки на кластера, нужно изменить деплой таким образом, чтобы он был заблокирован во время учений. Управление сетевыми правилами реализовано через сервис Yandex Topka.

7. Анализ соответствия требованиям

По окончании отчётного периода в «9999» достигнуты следующие результаты:

1. Сформулированы и внедрены меры предотвращения по 53 инцидентам.
2. Введены регулярные учения по потере дата-центра, по которым отсутствуют или решены инциденты.
3. Доступность обоих сервисов входящих в платформу составила более чем 99.9%.
4. Оба сервиса в платформе получили статус тира С.

Доступность обоих сервисов платформы достигла уровня тира В, но данный тир также требует других требований, которые не были удовлетворены. Тем не менее, за один отчётный период платформа сразу достигла тира С, что было целью двух отчётных периодов, т.е. работа превысила свои ожидания и платформа стала намного отказоустойчивее и стабильнее, чем раньше. Пользователи стали меньше жаловаться, и новых требований к стабильности на второй отчётный период не было выставлено.

8. Заключение

В рамках выполнения данной работы были достигнуты следующие результаты.

1. Проведён обзор используемых технологий и требований фреймворка «9999».
2. Выработан и внедрен процесс сбора метрик измерения работоспособности главной функциональности платформы. Данные метрики отправлены в фреймворк «9999» для расчета уровня доступности платформы.
3. Сформулированы предложения по улучшениям на основе анализа собранных метрик и других требований.
4. Реализованы и внедрены улучшения в существующие процессы разметки данных и поддержки.
5. Получен статус тира С для всех сервисов платформы.

Код работы находится в закрытом репозитории.

Список литературы

- [1] Chaos Engineering: System Resiliency. Datacenter operations. — P. 65–78.
- [2] Crowdsourcing explained. — URL: <https://www.investopedia.com/terms/c/crowdsourcing.asp> (online; accessed: 2024-11-20).
- [3] Grafana Labs about RED. — URL: <https://grafana.com/blog/2018/08/02/the-red-method-how-to-instrument-your-services/> (online; accessed: 2024-11-05).
- [4] Kubernetes. — URL: <http://kubernetes.io> (online; accessed: 2024-11-30).
- [5] Kubernetes How-to Disruption Budget Analogy to Yandex Deploy. — URL: <https://kubernetes.io/docs/tasks/run-application/configure-pdb/> (online; accessed: 2024-01-03).
- [6] Site Reliability Engineering: How Google Runs Production Systems. Handling Overload. — P. 226–242.
- [7] Site Reliability Engineering: How Google Runs Production Systems. Incident Management. — P. 146–152.
- [8] USE explanation from Brendan Gregg. — URL: <https://www.brendangregg.com/usemethod.html> (online; accessed: 2024-11-04).
- [9] WCSE; Dynamic Weight Adjustment. — URL: https://www.wcse.org/WCSE_2018/W110.pdf (online; accessed: 2024-11-20).
- [10] Yandex Market; Product Website. — URL: <https://market.yandex.ru/> (online; accessed: 2024-10-25).