

Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПРОГРАММНЫЙ ПРОЕКТ НА ТЕМУ
"ПРИМЕНЕНИЕ ОБЛАЧНЫХ ТЕХНОЛОГИЙ ДЛЯ ПРОВЕРКИ РЕШЕНИЙ УЧЕБНЫХ
ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ"

Выполнил студент группы 206, 4 курса,
Олигер Никита Александрович

Руководитель ВКР:
Старший преподаватель
Департамент больших данных и информационного поиска
Куренков Владимир Вячеславович

Москва 2024

Содержание

1	Введение	4
1.1	Предметная область	4
1.2	Цель и требования проекта	4
1.3	Ожидаемые результаты работы	5
1.4	Структура работы	5
2	Существующие решения и подходы	5
2.1	Яндекс Контест	6
2.1.1	Архитектурные особенности платформы	6
2.1.2	Недостатки платформы	6
2.2	Ejudge	7
2.2.1	Обзор платформы	7
2.3	Сравнительная характеристика решений	8
3	Выбор подхода к решению	9
3.1	Модели решения	9
3.2	Концепция 'Serverless'	10
3.3	Выбор провайдера	11
3.3.1	Удобство использования сервиса	11
3.3.2	Тарификация	11
4	Архитектура решения	13
4.1	Алгоритм исполнения задачи	14
4.2	Слой API	14
4.3	Слой исполнения	16
4.4	Базы данных	17
4.5	Структура теста	18
5	Веб интерфейс и API	19
5.1	Веб интерфейс	19
5.2	API	19
6	Администрирование платформы	20
6.1	Инструкция по установке	20
6.2	Мониторинг системы	20
7	Эксперименты и производительность платформы	20
8	Актуальность и значимость	21
8.1	Сценарии использования	21
9	Дальнейшее развитие платформы	22
10	Полученные результаты	22

Abstract

Platforms for grading programming assignments allow one to perform various tests. Such platforms have a purpose of verifying the correctness of tasks related to sports programming or backend development.

This paper presents the possibilities of using cloud technologies for software code testing. Besides, the paper provides a comparative analysis of using this approach in comparison with classical testing on dedicated servers. The result of the work is a platform built with the concept of CaaS (Cloud as a Service) [48], supporting automated testing on C++ and Python languages.

To implement the platform, instructions for its configuration as well as administration are added. The interaction interface is described in detail for its use. The source code is available on [GitHub](#).

Keywords

Cloud services, serverless applications, lambda functions, automated testing

Аннотация

Платформы для проверки решений учебных задач по программированию позволяют проводить различные тестирования программного кода. Такие платформы преимущественно направлены на проверку корректности выполнения задач в рамках изучения спортивного программирования и бэкенд-разработки.

Данная работа представляет возможности применения облачных технологий для тестирования программного кода. Кроме того, в работе проводится сравнительный анализ использования такого подхода в сравнении с классическим тестированием на выделенных серверах. Результатом выполнения работы является платформа, построенная с концепцией CaaS (Cloud as a Service), поддерживающая автоматизированное тестирование для языков C++ и Python.

Для внедрения платформы добавлена инструкция по её настройке, а также администрированию. Для её использования подробно описан интерфейс взаимодействия. Исходный код размещён на [GitHub](#).

Ключевые слова

Облачные платформы, бессерверные вычисления, лямбда-функции, автоматизированное тестирование

1 Введение

1.1 Предметная область

Тренировочные задачи по программированию являются неотъемлемой частью любого курса, нацеленного на ознакомление студентов с языками программирования и алгоритмами. Однако составление таких заданий оказывается гораздо более простым в сравнении с проверкой решений к предложенным задачам.

Если обратиться к истории создания учебных курсов по программированию, можно заметить, что написание кода на бумажном носителе и последующая проверка преподавателем вручную имели много недостатков: такой процесс был очень затратен по времени на всех его этапах, а точность проверки напрямую зависела от человеческого фактора. Было понятно: этот процесс было необходимо упростить и автоматизировать.

Сейчас большинство тренировочных онлайн-платформ, таких как Leetcode [2], Coursera [3] и прочие, имеет собственные системы тестирования решений тренировочных задач. Такие системы позволяют проверять код решения задачи на наборе тестов на корректность, покрывая при этом всевозможные корнер кейсы. Эффективность каждого конкретного решения зависит от скорости выполнения проверки, а также от широты функционала: наличия выбранного студентом языка программирования, полноты документации и сценариев использования.

Проблемой многих коммерческих решений становится тот факт, что пользователь существующих решений либо не может добавить собственную задачу для тестирования вовсе, либо имеет большое ограничение на добавление собственного функционала. Более того, лишь малая часть существующих решений может предложить возможности расширения областей тестирования. С другой стороны, поддержка, обслуживания, и оплата ресурсов для хостинга решений с открытым исходным кодом зачастую требуют знания инструментария DevOps инженера и неоправданно больших вложений.

В этой работе мы рассмотрим возможные облачные сервисы, которые позволят решить указанные проблемы, проанализируем их архитектуру, преимущества и недостатки, а также представим альтернативу данным решениям.

1.2 Цель и требования проекта

Целью данной работы является создание альтернативной платформы тестирования тренировочных задач, имеющих следующие функциональные требования:

- Поддерживаются C++ и Python: два основных языка программирования, используемых целевой аудиторией — школьниками и студентами младших курсов.
- Пользователю предоставляется достоверная информация о результатах проверки и использовании вычислительных ресурсов: памяти и времени исполнения («runtime»).
- Решение реализовано в облачной среде, с соблюдением подхода CaaS (Cloud as a Service).

Наряду с озвученными выше требованиями, решение также имеет ряд нефункциональных требований, присущих системам тестирования:

- Скорость выполнения запроса на проверку должна соответствовать запрошенному виду тестирования: несколько секунд для предопределенного набора тестов.
- Отчёт о проведении проверки полон, а её результаты корректны.
- Система удобна в использовании, её интерфейс понятен, а результат проверки автоматически отправляется пользователю в удобном формате по завершении работы.

- Система является open-source решением с возможностью внедрения, настройки и улучшения, а также необходимой документацией

1.3 Ожидаемые результаты работы

Ожидается, что в результате работы будет представлена open-source система (с открытым исходным кодом) проверки тренировочных задач по программированию, которая будет представлять больший функционал по сравнению с предшествующими решениями. Важно также отметить, что данное решение также должно быть доступно к развёртыванию для любого пользователя, знающего принципы работы с Яндекс.Облаком и не уступать своим конкурентам по скорости выполнения операций.

Исходный код: <https://github.com/polarnights/ContestED>

1.4 Структура работы

Для удобства ознакомления со статьей, читателю предлагается краткая структура работы.

- В параграфе «1.Введение» размещена постановка задачи.
- В параграфе «2.Существующие решения» проводится анализ решений, подмечаются общие преимущества и недостатки подходов.
- В параграфе «3.Выбор подхода к решению» даётся обоснование выбора методологии бессерверных вычислений, рассказывается о возможности их применения в работе.
- В параграфе «4.Архитектура решения» подробно объяснена взаимосвязь компонент в архитектуре решений, обоснован выбор языка программирования, библиотек.
- В параграфе «5. Веб интерфейс и HTTP API» рассказывается о функционале платформы для пользователя: web-интерфейса и telegram бота.
- В параграфе «6. Администрирование платформы» показан процесс мониторинга инфраструктуры и потребляемых ресурсов.
- В параграфе «7. Эксперименты и производительность» исследуются результаты тестирований, проведенных над платформой.
- В параграфе «8. Актуальность и значимость» предлагается сравнение платформы «ContestED» с её альтернативами и возможные сценарии использования платформы.
- В параграфе «9. Полученные результаты» резюмируется проведенная над проектом работа.
- В параграфе «10. Дальнейшее развитие платформы» рассуждается о возможностях улучшения текущего функционала.

2 Существующие решения и подходы

Перед анализом уже существующих решений, важно отметить, что большинство из них представляют собой коммерческий продукт. Это создаёт определенные трудности, связанные с отсутствием исходного кода, соответствующей теоретической составляющей в виде документации, литературы или формализованных SLA (Service Level Agreement — договор-соглашение об уровне предоставляемых услуг). Более того, в рамках данной работы мы будем ограничивать

функционал системы строго в рамках проведения курсов по программированию, алгоритмам и структурам данных и прочих, где ответом на задачу является программный код. Особенности функционала, не относящегося напрямую к процессу тестирования, но повышающего пользовательский опыт эксплуатации (называемый «UX» — user experience), отмечены в разделе 9.

Как можно заметить, совокупность параметров сильно ограничивает возможности исследования инфраструктурных особенностей платформ. Именно поэтому плюсы и минусы платформ будут описаны с опорой именно на пользовательский опыт и имеющуюся документацию. Логическим завершением этой части работы будет являться таблица со сравнительной характеристикой платформ.

2.1 Яндекс Контест

Яндекс Контест — одна из наиболее популярных в России систем автоматической проверки различных задач, появившаяся более 12 лет назад. Однако, обратившись к статье одного из разработчиков [4] и блогам пользователей на альтернативной платформе проведения соревнований по программированию Codeforces [7], понимаем, что активная работа над изменением функционала системы не ведется уже около трёх лет. Имеется 22 доступных языка программирования в 124 различных версиях [4].

2.1.1 Архитектурные особенности платформы

Разработчики активно выстраивали инфраструктуру платформы ещё в 2011-2012 годах, и с тех пор она не переживала глобальных изменений. Процесс проверки решений проходит в три фазы: фаза компиляции, затем тестирование и агрегация результатов теста. Используемое разработчиками окружение следующее: существует кластер виртуальных машин, на которых запущен набор легковесных образов контейнеров (например, Docker-контейнеров [9]), на каждом из которых работает приложение, отвечающее за все этапы проверки. Существует также оркестратор, который отвечает за распределение задач по отдельным единицам приложения.

Необходимость в хранении большого числа библиотек на каждой отдельной VM была решена при помощи OverlayFS [11]. Гибридная файловая система позволяет «накладывать» часть файлов операционной системы, используемых строго на чтение, на другие файловые системы, отличной от текущей. Такое решение позволило разработчикам сэкономить около 10-20 ГБ операционной системы на один лишь контейнер, а следовательно, дать больший масштаб и для горизонтального, и для вертикального масштабирования.

2.1.2 Недостатки платформы

Несмотря на все преимущества Яндекс.Контеста, платформа имеет ряд недостатков, которые отмечают в том числе и сами авторы в другой статье [8].

Во-первых, в силу необходимости проведения соревнований по программированию, в том числе Yandex Cup [12], силы разработчиков уходят на уравнивание времени решения задач на разных виртуальных машинах. Например, для достижения этого разработчики отключают технологию увеличения частоты CPU под нагрузкой Intel Turbo Boost [10] и стараются избавиться от конкуренции за ядра. Однако такой компромисс негативно влияет на скорость выполнения задач для остальной, что важнее — большей аудитории платформы, которая не участвует в реальных соревнованиях. Кроме того, ограничения на функционал процессора приводит к необходимости выделения дополнительных мощностей и обязательного использования однородного железа на весь набор виртуальных машин, а значит улучшение решающих машин должно проходить одновременно. Как итог, такой подход недопустим в решениях с открытым исходным кодом, поскольку обслуживание платформы стоит огромных средств, система требовательна к

аппаратному обеспечению, а также подразумевает задержку в скорости обновления оборудования.

Во-вторых, решение совсем не использует возможности Яндекс.Облака, что приводит к необходимости регулярного сбора статистики и обслуживания физических серверов. На момент написания статьи, задача переноса системы в облачную среду, о которой упоминали сами авторы решения [8] в 2021 году, не решена. Использование современного подхода к размещению высоконагруженных систем также упростило бы работу над повышением надёжности и отказоустойчивости системы.

2.2 Ejudge

Ejudge [13] — это одно из немногочисленных решений с открытым исходным кодом [21], предоставляющих систему для проведения мероприятий с автоматической проверкой задач по программированию. Система появилась ещё в 2007 году, и с тех пор репозиторий получил 177 «звёзд» на GitHub. Исходный код системы на протяжении многих лет используется в ведущих высших учебных заведениях России, среди которых есть ВШЭ [17], МФТИ [19], КФУ [20] и прочие. На момент написания статьи систему продолжают поддерживать и обновлять (выход последней версии датируется декабрём 2023 года), в том числе посредством pull-request'ов в репозитории. Поддерживается тестирование 79 версий различных языков, однако более 10 из них, по заверениям авторов, устарели.

2.2.1 Обзор платформы

Платформа написана преимущественно на языке C для использования на ОС Linux. Анализируя исходный код платформы [21], можно выделить три основные компоненты системы:

- I. Компоненты, отвечающие за доступ к базе данных пользователей и компиляции решений. Реализованы в виде `daemon`, которые запускаются в фоновом режиме (то есть в качестве процесса, не имеющего управляющего терминала, без прямого взаимодействия с пользователем). Такие компоненты запускаются в единственном экземпляре, что ограничивает возможности по масштабированию системы и накладывают ряд требований к хост-устройству. Другой особенностью этих компонент является использование `GNU C Library` и специфичных для Linux механизмов передачи информации (UNIX-сокет), что делает невозможным использование Ejudge даже на других unix-подобных системах. Примеры таких компонент в исходном коде: `ej-compile`, `ej-jobs`, `ej-super-server`.
- II. Компоненты CGI (Common Gateway Interface), отвечающие за обработку запросов пользователя. В данном решении CGI позволяет передавать HTTP запросы исполняющим компонентам из первой группы, а также вспомогательным программам. Важно отметить, что CGI, появившийся ещё в 1993 году, хоть и по-прежнему используется в части программ, в данный момент можно назвать устаревшим: его повсеместно заменяет стандарт WSGI (Web Server Gateway Interface), используемый в таких фреймворках, как Flask [16], будучи более легковесный и простой в работе, Django [15], обладающий большим набором инструментов, и прочих. Примеры таких компонент в исходном коде: `ej-compile`, `ej-jobs`, `ej-super-server`.
- III. Вспомогательные компоненты администрирования ejudge, выполняющиеся из командной строки. Помимо стандартного обслуживания сервера, на котором находится система, администратору необходимо корректно настроить множество конфигурационных файлов. Основными файлами для первичной настройки являются базовые конфигурации системы и отдельного контеста, описываемые в качестве файлов с расширением

`.cfg`. При этом интерпретация настроек в файлах данного формата не имеет единого регламента, потому необходимо ознакомление с документацией. Также к ключевым можно отнести файлы для настройки условий и тестов задач, написанные в формате `XML`. Система позволяет контролировать полномочия пользователей, настраивать ограничения доступа по IP-адресам. Однако часть навыков DevOps и SRE инженеров: мониторинг системы, логгирование в удобном формате (отличном от построчной записи всех посылок в виде plain text в файл), по-прежнему остаётся обязанностью администратора системы, что повышает и так высокий «порог входа». Долгое время безопасный запуск решений был возможен лишь при помощи специального патча к ядру Linux, ограничивающего права пользователя, с которого проводится тестирование. Безусловно, воплотить такой способ на выделенной виртуальной машине без обращения в техническую поддержку, невозможно, что значительно ограничивало сценарии использования.

2.3 Сравнительная характеристика решений

Проанализировав вышеупомянутые решения, можно выделить ряд наблюдений об основных тенденциях.

- Большинство платформ поддерживает тестирование задач на большом количестве языков программирования. Однако со своевременной поддержкой новых компиляторов для них часто возникают сложности.
- Большинство платформ проводит тестирование в контейнеризованных приложениях.
- Часть платформ не предлагают автоматизированный отчет о решении и обратную связь или же требуют рецензирование кода («code-review»).
- Большинство платформ предоставляют фиксированные возможности кастомизации (либо не предоставляют вовсе) и не предполагают изменения.

Основные характеристики платформ были выведены в единую таблицу. Преимущество всех сервисов заключается в автоматизированной системе оценивания и предоставлении задач из реального опыта. Что касается недостатков, мы можем увидеть, что множество платформ требует оплату за использование их услуг («LeetCode», «HackerRank» и другие). Кроме того, не у всех имеются определенные удобства: в «Coursera» не реализовано добавление новых задач, в «Manytask» — возможность взаимодействия с аудиторией платформы. Среди всех сервисов мы можем выделить два наиболее функциональных — «Яндекс Контест» и «Codeforces». Учитывая, что обе платформы во многом схожи, далее мы будем опираться на опыт построения лишь одной из них — «Яндекс Контест».

Платформа	Добавление новых задач	Без оплаты	Автоматическое оценивание	Поддержка сообщества	«Real-world» окружение
LeetCode	Да	Нет	Да	Да	Да
Yandex.Contest	Да	Да	Да	Да	Да
Coursera	Нет	Нет	Да	Да	Да
Manytask	Да	Да	Да	Нет	Да
HackerRank	Да	Нет	Да	Да	Да
CodeSignal	Да	Нет	Да	Да	Да
Codeforces	Да	Да	Да	Да	Да

Таблица 2.1: Сравнение платформ для тестирования заданий по программированию

3 Выбор подхода к решению

Исследуем различные подходы к построению платформ тестирования тренировочных задач.

3.1 Модели решения

Для того, чтобы определиться с подходом к разработке платформы, необходимо чётко разделить ответственность разработчика и облачного провайдера. Изучив ряд недостатков, имеющих у традиционного подхода «on-premises» к решению задачи тестирования тренировочных задач (см. 2.3), можно заключить: ответственность за аппаратное обеспечение (hardware), на котором будет размещен сервис, и виртуализацию (созданию виртуальных машин) будет лежать на облачном провайдере.

Далее перед нами стоит выбор между другими моделями использования облачных вычислений. Кратко ознакомимся с отличиями сфер ответственности пользователя и облачного провайдера, составив сравнительную характеристику. Более подробно о различии платформ указано в статье Google Cloud [31].

Услуга	CaaS	PaaS	FaaS
Управление инфраструктурой	Пользователь	Облачный провайдер	Облачный провайдер
Управление контейнером	Пользователь	—	—
Разработка приложения	Пользователь	Облачный провайдер	Пользователь
Масштабирование	Вариативно*	Облачный провайдер	Облачный провайдер
Контроль над инфраструктурой	Высокий	Средний	Низкий
Сложность использования	Высокая	Средняя	Низкая

Вариативно* — может быть как пользователь (для собственной оркестрации контейнеров), так и облачный провайдер (для облачного обеспечения систем Kubernetes или бессерверных контейнеров)

Таблица 3.1: Сравнение подходов CaaS, PaaS, and FaaS

Напомним, что важным требованием к системе «ContestED» является низкий порог входа: минимальные затраты на установку системы, и минимальные усилия на её обслуживание. Таким образом, нашей задачей является грамотно использовать возможности этих платформ, совмещая их в решении.

Также важным фактором сложности управления платформой является вопрос автоматизация масштабирования частей систем. В случае, когда платформа требует совмещенного использования разных сервисов, масштабирование всей системы целиком приведёт к неоправданно большим тратам. Для «ContestED» грамотным решением будет использование гранулярного

масштабирования (granular scaling) - подхода, при котором масштабирование происходит индивидуально по «гранулам» - каждому отдельному облачному сервису, функции, или контейнеру.

3.2 Концепция 'Serverless'

Концепция бессерверных вычислений (serverless computing) — стратегия использования облачных вычислений, подразумевающее передачу ответственности за автоматизированное динамическое управление выделенными ресурсами. Такой подход позволяет разработчикам сосредоточиться на написании кода, в то время как за предоставление и обслуживание инфраструктуры, её взаимодействие с другими сервисами, масштабирование отвечает облачный провайдер.

Этот подход сочетает в себе модель FaaS и event-driven архитектуру (то бишь, архитектуру, при которой определённые события и условия приводят к вызову функции) [5]. Появление бессерверных вычислений стало большим преимуществом как для облачных провайдеров, так и для конечных пользователей. Использование serverless технологий предоставляет возможность компоновать ресурсы, минимизируя число серверов, которые нужно обслуживать. Далее рассмотрим основные преимущества концепции для разработчиков.

- Снижение требований к стеку (от англ. stack — набор инструментов, применяющийся при работе в проектах), ускорение процесса разработки, достижимому благодаря упрощению управления инфраструктурой и гибкости системы сервисов крупных облачных провайдеров
- Меньшая плата за потребляемые ресурсы при грамотном использовании подхода, предоставляемая тарификацией по системе «оплата за использование» (pay-as-you-go). Это снижает риск избыточного или недостаточного выделения ресурсов, которое может привести к лишним расходам и ошибкам в работе системы.
- С учётом выбора подходящей архитектуры, гранулярное масштабирование работает «автоматически» и не требует оркестрации или настройки (как, например, горизонтального масштабирования через отдельный ресурс Horizontal Pod Autoscaler в Kubernetes[6])

Однако при выборе serverless подхода необходимо понимать ряд его недостатков. В модели бессерверных вычислений функции выполняются по запросу, а контейнер, в котором выполняется функция, создается и удаляется автоматически. Это означает, что при каждом вызове функции может происходить «холодный старт» — дополнительное время на выделение ресурсов и создание окружения, в котором будет производиться вычисление. Это, безусловно, приводит к увеличению задержки и снижению производительности.

Чтобы смягчить последствия холодного старта, некоторые бессерверные платформы используют такие методы, как «предварительный прогрев» (pre-warming), при котором контейнеры продолжают работать в фоновом режиме и готовы обрабатывать запросы, как только они понадобятся. Однако такой подход может привести к увеличению затрат и увеличению использования ресурсов, поскольку контейнеры продолжают работать, даже если они не используются.

В целом, «холодный запуск» является неотъемлемой характеристикой бессерверных вычислений, и разработчикам необходимо учитывать это при разработке и развертывании своих приложений. Основными стратегиями по борьбе с ним являются оптимизация кода приложения, уменьшение размера контейнера и использование менее специфичных языковых сред выполнения.

Как итог, бессерверные вычисления особенно подходят для сценариев, требующих высокой доступности и обработки данных в реальном времени. Примерами таких сценариев являются веб-приложения, обработка данных (в том числе в прямом времени) и событий. В веб-приложениях бессерверные вычисления позволяют разработчикам создавать масштабируемые

и высокодоступные приложения, способные справляться с внезапными скачками трафика. При обработке данных бессерверные вычисления позволяют быстро обрабатывать большие массивы данных, что дает возможность в реальном времени делать выводы и аналитику. Наконец, при обработке событий бессерверные вычисления позволяют обрабатывать и индексировать большие объемы данных, поддерживая аналитику и возможность применения машинного обучения в режиме реального времени.

3.3 Выбор провайдера

Как было сказано ранее, в качестве основного провайдера аппаратного обеспечения было выбрано Яндекс.Облако. Для обоснования выбора необходимо задаться рядом вопросов, которые интересны тем, кто планирует внедрять ContestED в обиход.

3.3.1 Удобство использования сервиса

У провайдера есть библиотека встроенных сервисов, упрощающих настройку облачной инфраструктуры и сети, раздел для работы с бессерверными вычислениями и мониторингом системы. Плата за использование сервисов не взимается, оплачиваются лишь используемые ресурсы. Для ознакомления пользователям предоставляется документация и примеры проектов [28] на русском языке; основная часть документации также переведена на английский. Провайдер также предоставляет free-tier (бесплатного уровня, то есть доступные новому пользователю) тарифный план, позволяющий опробовать сервисы и ознакомиться с сервисами и утилитами для работы с платформой.

3.3.2 Тарификация

К сравнению предлагаются три основных облачных провайдера, на которых, согласно исследованиям SRG [33] приходится более 60% процентов соответствующего рынка. Amazon Web Service [23], Microsoft Azure [37], Google Cloud Platform [38], а также крупнейший в России облачный провайдер Яндекс Облако [39].

В основной части таблицы рассматриваются три фундаментальных serverless сервиса: функции, контейнеры и базы данных. Для них приведено подробное описание цены за минимальный и максимальный набор ресурсов соответственно. Развёрнутость сравнения обусловлена важностью ценообразования этих компонентов: именно на них приходится большая часть расходов в проекте. Для остальных сервисов, использованных при разработке платформы, приведена краткая справка политики тарификации Яндекс Облака, и средней стоимости использования сервиса (за условную единицу) среди остальных платформ.

По итогам анализа можно утверждать: ценообразование различных облачных провайдеров взаимосвязано между собой. Более того, использование большинства сервисов Яндекс Облака будет обходиться не дороже, чем для аналогичных сервисов (с аналогичными характеристиками) других провайдеров.

Сервис	Провайдер	Политика ценообразования (за исп.)
Serverless containers	AWS Fargate	\$0.00000672 (0.25 vCPU и 0.5 GB) \$0.01072 (4 vCPU и 8 GB)
	Google Cloud Run	\$0.0000025 (1 vCPU и 256 MB) \$0.000015 (4 vCPU и 4 GB)
	Azure Container Instances	\$0.0000022 (0.5 vCPU и 1 GB) \$0.000044 (2 vCPU и 4 GB)
	Yandex CGI	\$0.0000022 (0.5 vCPU и 1 GB) \$0.000044 (2 vCPU и 4 GB)
Serverless functions	AWS Lambda	\$0.000000002 (менее чем 128 MB) \$0.000016 (3008 MB)
	Google Cloud Functions	\$0.0000025 (менее чем 128 MB) \$0.000015 (2 vCPU и 896 MB)
	Azure Functions	\$0.0000002 (менее чем 128 MB) \$0.000016 (3.5 vCPU и 1408 MB)
	Yandex CGI	\$0.0000004 (менее чем 128 MB) \$0.000016 (4 vCPU и 1792 MB)
Serverless databases	Amazon S3	\$0.023 за 1 GB в месяц (standard storage) \$0.0125 за 1 GB в месяц (cold storage)
	Yandex Cloud Object Storage	\$0.015 за 1 GB в месяц (standard storage) \$0.0075 за 1 GB в месяц (cold storage)
	Google Cloud Storage	\$0.026 за 1 GB в месяц (standard storage) \$0.015 за 1 GB в месяц (nearline storage)
	Azure Blob Storage	\$0.0184 за 1 GB в месяц (hot storage) \$0.0122 за 1 GB в месяц (cool storage)
Serverless logging and monitoring services	AWS CloudWatch	\$0.005 за 1,000 запросов \$0.002 за 1,000 запросов (data processing)
	Google Cloud Logging	\$0.006 за 1,000 запросов \$0.002 за 1,000 запросов (data processing)
Serverless API Gateways	AWS API Gateway	\$3.50 за миллион запросов \$0.90 за миллион запросов (regional)
	Google Cloud Endpoints	\$3.00 за миллион запросов \$0.80 за миллион запросов (regional)
Serverless message queues	AWS Simple Queue Service	\$0.0000004 за запрос \$0.000000004 за запрос (first 1 million)
	Google Cloud Pub/Sub	\$0.0000004 за запрос \$0.000000004 за запрос (first 1 million)
Other features		
Google Cloud Functions Environment Variables		

Таблица 3.2: сравнение ценообразования serverless-стека облачных провайдеров AWS, Google Cloud, Microsoft Azure & Яндекс Облако.

Замечание. Из-за текущих ограничений на использование платёжных систем гражданами РФ, действующих на момент написания статьи, использование других провайдеров затруднительно. Однако стоит отметить и то, что в рамках данного проекта не ставилась цель привяз-

ки пользователя к определённому облачному провайдеру, называемой cloud lock-in. Проблема сложности смены облачного провайдера далеко не нова и обсуждалась в научных статьях [29]. Основной трудностью для разработчика является сложность переноса инфраструктуры проекта, особенно — без downtime, то бишь времени простоя, а также финансовые издержки на настройку новой системы и адаптации к ней. В действительности, аналогичный функционал сервисов, исполняемых в работе над ContestED, представлен у других облачных провайдеров, в том числе указанных ранее. Если ознакомиться с терминологией желаемого провайдера, предоставленного в таблице, и обладать навыками эксплуатации соответствующей платформы, перенос проекта не составит трудностей.

4 Архитектура решения

Рассмотрим схему архитектуры полученного решения.

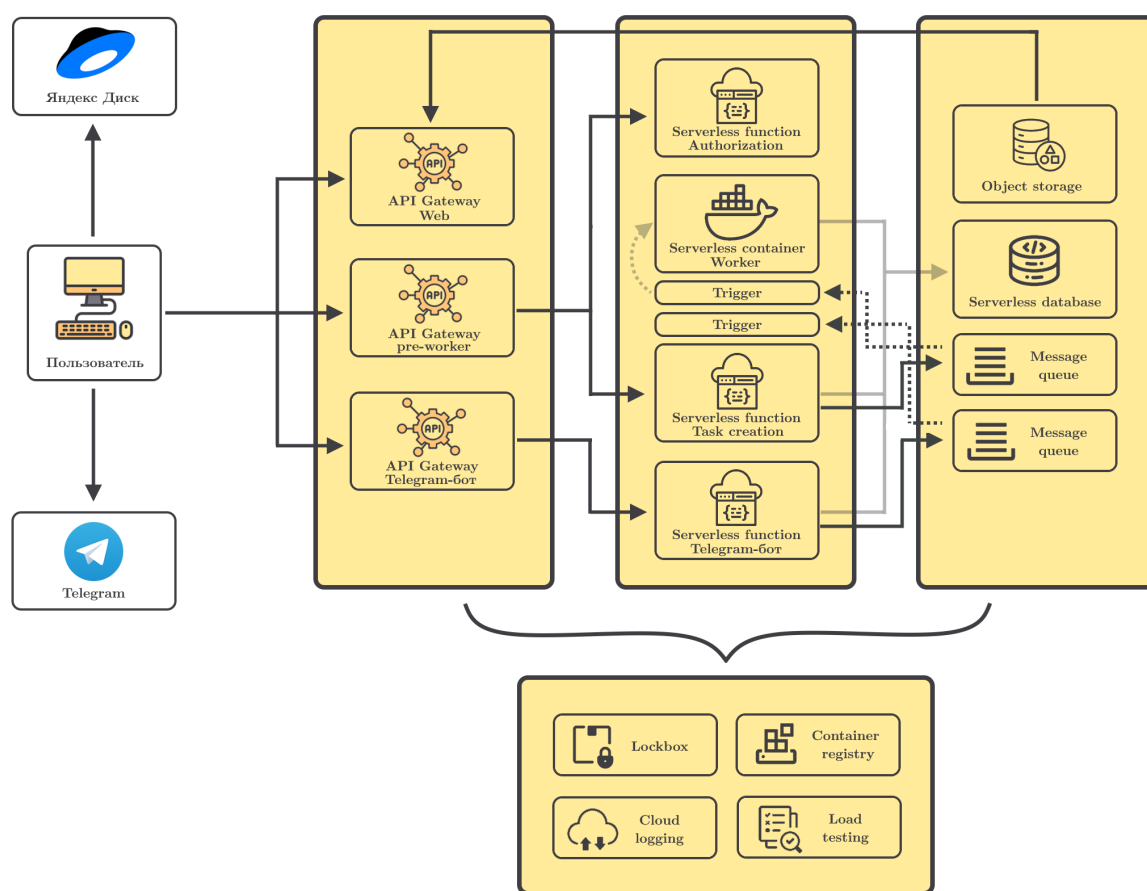


Рис. 4.1: Архитектура инфраструктуры облачной платформы «ContestED»

Нетрудно заметить, что элементы в схеме можно разделить на три основных слоя и два вспомогательных. Кратко разберём вспомогательные слои.

Во-первых, представлен «нулевой» этап обработки: он включает в себя стадии предварительной подготовки к использованию платформы, а именно загрузку решения на Яндекс.Диск и первичное взаимодействие пользователя с Telegram ботом.

Во-вторых, существует набор служебных инструментов:

- LockBox, сервис, поддерживающий надежное шифрование и безопасное хранение секретов, доступ к которому происходит по статическому ключу.

- Container Registry, используемый для использования Docker-образа в бессерверных контейнерах.
- Cloud Logging — инструмент логгирования и мониторинга облачных компонент.
- Load Testing — сервис, позволяющий проводить нагрузочное тестирование системы.

4.1 Алгоритм исполнения задачи

Выстроена следующая цепь последовательных действий для тестирования одного решения:

1. Пользователь получает доступ к системе: при первичном заходе это процесс регистрации, далее аутентификации и авторизации. Альтернативным вариантом является вход через социальную сеть.
2. Далее идёт этап взаимодействия web-интерфейсом системы: пользователь отправляет HTTP-запрос (его структура описана в 5.2) для тестирования решения.
3. Следом запрос обрабатывается облачными функциями: в базе данных создаётся задача с информацией, указанной пользователем в запросе.
4. После получения новой задачи, задача «автоматически» распределяется на serverless контейнер. В случае отсутствия готового окружения, запускается новый контейнер с легковесным образом, который находится в реестре облачного провайдера.
5. Затем проходит этап тестирования задачи, измеряются затраченные на этот процесс ресурсы.
6. Результаты тестирования вместе с полученными метриками сохраняются в хранилище, после чего запрос помечается завершённым.
7. По завершении запроса, опционально отправляется уведомление в мессенджере Telegram, дублирующее полученные результаты тестирования.

4.2 Слой API

Первым слоем в данном случае будет слой API, поддерживаемый облачным сервисом API Gateway.

С развитием облачных приложений, разработчики стали нуждаться в стандартизированном подходе к созданию и управлению большим набором API. В этом контексте API Gateways стали выполнять роль промежуточной точкой доступа, находящейся посреди самих запросов и исполнительской, конечной точки (например, другого API) [30]. Они позволяют создать единый интерфейс взаимодействия, однако это взаимодействие может быть как внешним (External API GW, когда запросы выполняются клиентом), так и внутренним (Internal API GW, когда запросы посылает определенный микросервиса или другой внутренний API). Более того, микросервисной архитектуре приложений необходима маршрутизация запросов, предоставляющая единую схему проверки подлинности и авторизации, а также безопасное хэширование. Такой сервис с одноимённым названием, обладающий также функциями балансировки нагрузки и оптимизации потребления ресурсов для набора отдельных API, есть в Яндекс.Облаке [24].

В исходном коде ContestED присутствует несколько различных API Gateway, написанные в YAML формате со спецификацией OpenAPI 3.0. Разные API в этом случае отвечают за разные компоненты системы: web-приложение, обработку запроса, telegram бота. Их исходный код находится в репозитории размещен по пути Cloud/Gateways.

Каждый API Gateway имеет несколько конечных точек запроса, называемых «endpoint». Для каждого запроса формат принимаемых данных может быть указан в отдельной схеме, которая описывается в том же файле. В разделе `schemas` указываются схемы входных данных, передающихся как «query»-параметры (последовательно указанные в формате «ключ - значение» в самом HTTP запросе), а также схемы для возврата ошибок в выполнении запроса.

Каждый из них имеет интеграцию с другим облачным сервисом. Для web-приложения это Object Storage, для обработки запроса — Serverless Container, а для работы Telegram бота — Serverless функция.

Рассмотрим схему такого файла, взяв для примера этап предобработки запроса в формате `YML` из репозитория.

В нашем случае конечная точка `/status` взаимодействует с бессерверной функцией `preworker_function`, и запускается под сервисным аккаунтом. Требования к сервисному аккаунту указаны в 6.1.

```
openapi: "3.0.0"
info:
  version: 1.0.0
  title: Grading API
servers:
- url: https://<APIGW_ID>.apigw.yandexcloud.net
paths:
  /status:
    get:
      summary: Get Status
      operationId: gettaskstatus
      security:
        - httpBearerAuth: [ ]
      # Involving the auth cloud function;
      parameters:
        # ... ;
        # Specification of params to pass;
      responses:
        '200':
          description: Operation has gone successfully
          content:
            'application/json':
              schema:
                $ref: '#/components/schemas/Task'
          default:
            # ... ;
            # Returning an error by the scheme below ;
      x-yc-apigateway-integration:
        type: cloud_functions
        function_id: <PREWORKER_FUNCTION_ID>
        tag: "$latest"
        service_account_id: <SERVICE_ACCOUNT_ID>
```

Для единого формата передачи данных, созданы `components:schemas`. Поскольку вызов `preworker_function` может быть осуществлен лишь авторизованным пользователем, каждый метод также включает `header: security`, который взаимодействует с облачной функцией авторизации.

```
# ... ;
```



```

# Specifying other endpoints ;
components:
  schemas:
    Error:
      # ... ;
      # Specification for scheme type, required params, properties;
      # ... ;
      # Specification for other schemes ;
  security:
    httpBearerAuth:
      type: http
      scheme: bearer
      x-yc-apigateway-authorizer:
        type: function
        function_id: <AUTH_FUNCTION_ID>
        authorizer_result_ttl_in_seconds: 3600

```

4.3 Слой исполнения

Вторым слоем является слой, отвечающий за приложение. В рамках концепции бессерверных вычислений, мы будем использовать два вида сервисов. В первую очередь, это сервис по созданию и настройке облачных функций [31]. Облачные функции позволяют создать окружение, в котором будет запускаться программный код. Главными отличиями такого окружения является в том, что функция выполняется в зависимости от действий: прямого её вызову, или действию триггера при выполнении указанного события, меняющего состояние инфраструктуры. Для исполнения кода провайдер предоставляет готовое окружение, а именно некоторый набор серверов, на которых провайдер выделяет часть ресурсов на исполнение задач. Техническое обслуживание такого окружения, его автоматическое масштабирование соразмерное текущей нагрузке, контейнеризация окружения и разделение вычислительных мощностей, а также базовая диагностика и логгирование операций в этом подходе являются прямыми обязанностями провайдера.

Для того, чтобы использование облачных функции было оправдано, необходимо придерживаться ряда основных концепций.

1. Тарификация облачных функций складывается из трёх компонент: количество событий, которые воспроизводят нашу функцию, объема и длительности использования вычислительных ресурсов и использования входящего и исходящего трафика. Более подробно о ценовой политике облачных провайдеров можно ознакомиться в параграфе 3.3.2.
2. Грамотная настройка оркестрации провайдером позволяет минимизировать число активных виртуальных машин и грамотно утилизировать невостребованные ресурсы. Именно поэтому облачные функции должны быть легковесны, и не требовать большого объема вычислительных ресурсов. Основными сценариями использования облачных функций, согласно статье Google Cloud [31], являются кратковременные процессы передачи, обработки и обогащения данных в реальном времени, интеграция с third-party сервисами и API.
3. Помимо объема используемых ресурсов, стоит оценить и частоту запуска данных функций и использование трафика, как входящего, так и исходящего.

Для ContestED эти концепции учтены благодаря распределению нетребовательных задач на исполнение в serverless функциях, и лишь тестирование — в контейнере. Далее предоставим описание реализации компонент.

В первую очередь, на платформе реализована функция, отвечающая за авторизацию и аутентификацию пользователя. В нашем случае авторизация выполняется с помощью взаимодействия с внешней библиотекой Auth0 [42] и библиотеки jwt [44]. Библиотека предоставляет доступ к использованию JSON Web Token, который представляет собой открытый стандарт токенов доступа, основанный на формате JSON. Код лямбда-функции авторизации написан на языке Python. Отметим, что готового сервиса регистрации и авторизации для управления пользователями в web-приложениях, аналогичному AWS Cognito [43], в Яндекс Облаке нет, хотя соответствующая просьба пользователей о реализации уже более года присутствует на сайте.

Для функции предварительной обработки задачи («pre-worker») использовалась библиотека boto3 [46]. Взаимодействие с сервисами Яндекс Облака осуществлялось через сессию **boto-session** с указанным в хранилище Lockbox секретным ключом доступа. Для получения доступа к ресурсам, а именно Message Queue, DynamoDB и Object Storage, реализована соответствующая функция. При поступлении новой задачи, лямбда-функция генерирует уникальный идентификатор задачи **TASK_ID**, сохраняя информацию о ней в базе данных. Далее производится процесс проверки соответствия файла указанным требованиям к размеру и формату, а также корректности указанных данных. В завершении работы функции производится смена статуса выполнения задачи; если корректность входных данных подтверждена, в очередь сообщений передаётся информация для тестирования решения в контейнере.

Функций, которые срабатывают по определённому сценарию (триггеру), выполняются при изменении состояния в очереди сообщений. Для ContestED такие функции это «pre-worker» и функция взаимодействия с ботом для платформы Telegram. В первом случае триггер необходим для запуска serverless контейнера при поступлении новой задачи. Для другой же функции, триггер срабатывает по завершении выполнения задачи, и активирует метод отправки сообщения о готовности отчёта тестирования.

Docker-образ тестирующей системы, используемый в бессерверных контейнерах, также размещен в исходном коде репозитория.

4.4 Базы данных

В решении было использованы два хранилища данных: AWS DynamoDB, доступный в Яндекс Облаке и Object Storage (аналогичное хранилищу S3 для Amazon). Отметим, что они предназначены для решения различных задач.

DynamoDB подходит для хранения структурированных или полуструктурированных данных. Для него есть верхние ограничения по размеру записи, что обусловлено очень высокой скоростью доступа, не более 10 мс в среднем.

Object Storage используется для хранения файлов. Файлы можно читать по протоколу HTTP с помощью его REST API. Он позволяет хранить гораздо больший массив данных, многократно превышающий аналогичные лимиты для DynamoDB), с приемлемой скоростью доступа.

Для большинства целей эффективно использовать оба хранилища вместе, разделяя статичные объекты, и те, доступ к которым нужен редко, в Object Storage, а DynamoDB использовать для хранения прочей информации.

В нашем случае DynamoDB служит табличной базой данных для хранения информации о тестировании задач, а в Object Storage размещаются zip-файлы с кодом участников, файлы для serverless web-приложения, и прочее.

4.5 Структура теста

Для корректности составления набора входных данных для тестирования необходимо формализовать требования кода, который принято считать корректным. Опорой для этого послужил опыт проведения именитых соревнований по программированию и тренировочных соревнований в рамках курсов «Основы и методологии программирования», «Алгоритмы и структуры данных» 1-2 курсов ФКН ВШЭ [47]. Среди известных олимпиадных состязаний по программированию были выделены ICPC и олимпиада «Innopolis Open», схема тестирования которых известна. Важно отметить, что такие соревнования вовсе не покрывают часть «технических» заданий, кардинально отличающихся по спецификации. Такие примеры можно встретить в тренировочных контестах на реализацию математических функций и методов. В итоге, был сформулирован следующий набор принципов тестирования:

- Первичный набор тестов, называемый открытыми тестами, принято предоставлять участникам для наглядной демонстрации формата входных данных, и возможности базового протестировать своё решение локально.

Наблюдение. Для первичного набор тестов существует приём, который можно использовать для снижения общей нагрузки тестирующей системы. Для этого достаточно предоставить участникам набор входных данных, не соответствующих «лобовым» решениям и отражающим требуемую вычислительную сложность алгоритма (см. пункт 4.5).

- Основной набор тестов, отвечающий за проверку всех логических ветвей алгоритма. Иногда такой набор генерируют случайно, например, с помощью соответствующих фреймворков [36].
- Набор тестов, проверяющих вырожденные случаи. К таким относятся те случаи, в которых алгоритм решения в корне отличается от основного набора тестов. К таким случаям относятся также нулевые примеры ввода, если они допустимы.

Наблюдение. В действительности, такие случаи в алгоритмических задачах встречаются довольно редко; обычно это задачи, сводящиеся к математическим, где могут присутствовать вырожденные значения параметров.

- Набор тестов, направленных на проверку эвристических алгоритмов. Такие тесты проверяют логику используемого подхода, и возможные сценарии, при которых не работают приближенные решения. Например, в части задач учащиеся могут ошибочно предположить, что задача может быть решена жадным алгоритмом или бинарным поиском по ответу.

Наблюдение. Некоторые задачи не имеют алгоритмического решения с полиномиальной сложностью, удовлетворяющего требованиям по времени и памяти. Такие задачи принято называть NP-полными, а правильное решение к ним — «перебором с предпочтением».

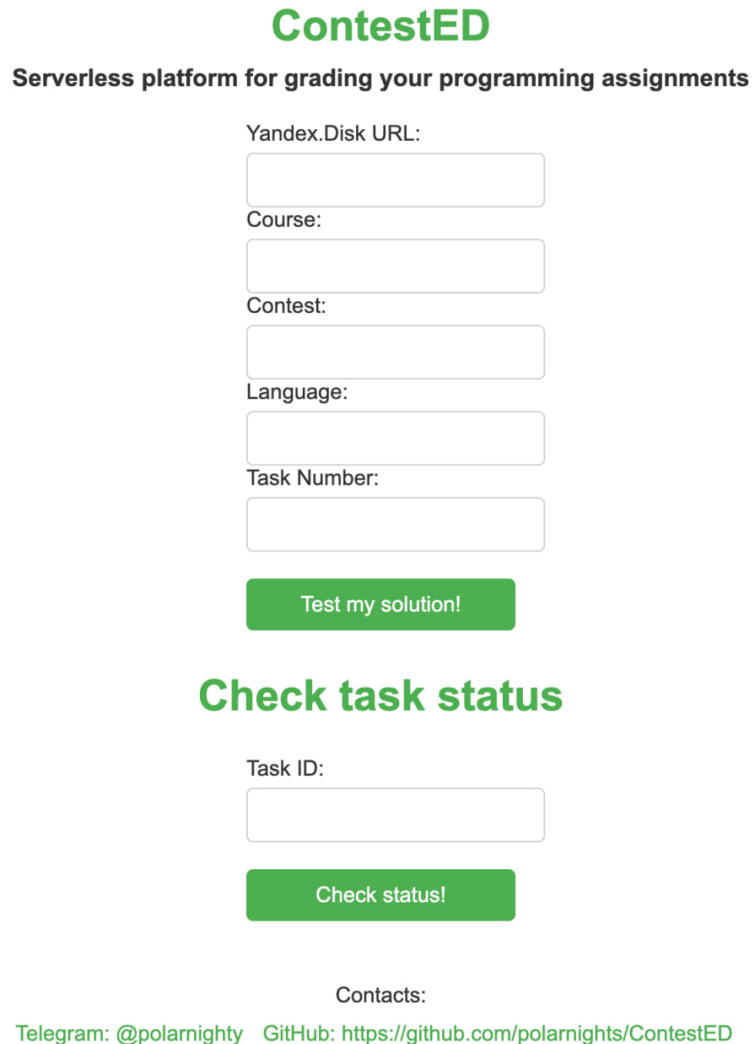
- Последним набором тестов служат тесты, проверяющие вычислительную сложность алгоритма. Такие тесты в основном состоят из граничных случаев, где входные данные близки к максимально (или минимально) допустимым значениям.

Наблюдение. В действительности, понять вычислительную сложность алгоритма не всегда представляется возможным. На практике часто важна константа, которая получается в верхней оценке сложности решения.

5 Веб интерфейс и API

5.1 Веб интерфейс

Web-интерфейс платформы выглядит следующим образом:



ContestED
Serverless platform for grading your programming assignments

Yandex.Disk URL:

Course:

Contest:

Language:

Task Number:

Test my solution!

Check task status

Task ID:

Check status!

Contacts:
Telegram: @polarnighty GitHub: <https://github.com/polarnights/ContestED>

Рис. 5.1: Web-интерфейс платформы «ContestED»

5.2 API

Функционал API выглядит следующим образом:

GET /status

Получение статуса обработки запроса.

В запросе передаётся параметр task_id.

Возможные статусы:

- NEW (Задача создана, происходит первичная обработка запроса)

- UPLOADING (Происходит процесс обработки файла)

- PROCESSING (Происходит процесс тестирования решения)

- DONE (Процесс тестирования завершен)

- REJECTED (Запрос отклонен с ошибкой)

```
POST /check_disk
# Обработка запроса с предзагруженным на Яндекс.Диск архивом
# В запросе передаются параметры src_url, course, contest, language, task_n

POST /check_upload
# Обработка запроса с загрузкой при помощи presigned URL
# В запросе передаются параметры course, contest, language, task_n
```

6 Администрирование платформы

6.1 Инструкция по установке

Инструкция для установки платформы размещена в `README.md` файлах репозитория.

Замечание. На момент написания работы, присутствуют сервисы, настройка которых невозможна инструментами командной строки. Для таких сервисов присутствует файл с демонстрацией настроек при установке. В действительности, многие из них имеют более простую и понятную установку через web-интерфейс в сравнении с аналогичной при использовании bash-скриптов.

6.2 Мониторинг системы

Стандартной практикой предотвращения и пост-фикса после проблем является логгирование и мониторинг серверов с помощью Grafana - утилиты, предоставляющей интерактивные доски с информацией о системе.

Она используется для мониторинга всех частей системы. Возможен просмотр как общего использования CPU и GPU, так и более детальное рассмотрение, например, при использовании навигационных дашбордов, переводящих нас на информацию о количестве успешных тестирований, количестве ошибок при исполнении задачи и т.д.

Эта утилита также полезна для мониторинга проблем с безопасностью. Полезно установить определённое пороговое значение потребления ресурсов одним кластером. В случае превышения этого значения мы можем предполагать, что в системе произошел сбой: например, возникли проблемы с масштабированием системы, или систему намеренно эксплуатируют: засылают много решений, или засылают заведомо вредоносные файлы, вроде fork-бомб.

7 Эксперименты и производительность платформы

Одним из важнейших критериев работоспособности платформы является возможность адаптивно подстраиваться под возрастающую нагрузку, сохраняя корректную работоспособность сервера.

Для этого проведём нагрузочное тестирование напрямую через API облачной функции pre-worker. Отметим, что в этом случае функция авторизации временно была исключена из YAML конфигурации для упрощения процесса тестирования.

Тестирование было проведено в сервисе Yandex Load Testing над легковесным запросом, который представлял собой получение статуса выполнения задачи. Профиль нагрузки: линейное изменение в течение 30 секунд, количеством запросов, или же «RPS» (requests per second), равномерно повышающимся от 1 до 100 на 1000 тестирующих потоков. Обработка происходила на

тестирующем агенте «Small» с характеристиками: 2 ГБ RAM, 2 vCPU (100%). Информация об агентах и процессах тестирования получена из соответствующей документации [35].

В рамках тестирования получены следующие результаты:

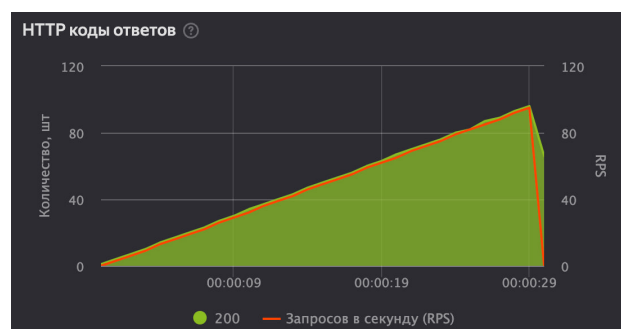


Рис. 7.1: HTTP коды ответов агента линейного профиля нагрузочного тестирования

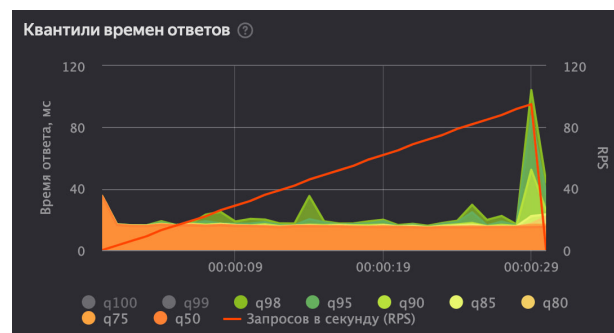


Рис. 7.2: Квантили времён ответа агента линейного профиля нагрузочного тестирования

Проанализировав графики, можно отметить, что с повышением нагрузки система справляется исправно.

Также был рассмотрен альтернативный вид тестирования с разными профилями нагрузки: unlimited (пиковая нагрузка в течение определенного срока), constant (постоянная нагрузка), step (пошаговое равномерное увеличение нагрузки). Реализовано тестирование на языке Python, посредством библиотеки Locust [40]. Выбор этого инструмента обоснован простой его использования и наличием удобного пользовательского интерфейса. Locust также является одним из самых популярных среди всех с открытым исходным кодом и имеет 23.8 тысяч «звезд» на GitHub [41]. Далее продемонстрирована часть конфигураций для тестирования. Все полученные результаты и исходный код тестирования размещены в репозитории проекта в директории **Testing/Results**.

```
startup:
  type: once
  times: 1000
rps:
  - type: line
    from: 1
    to: 100
    duration: 30s
```

```
startup:
  type: once
  times: 1000
rps:
  type: step
  steps:
    - from: 1
      to: 50
      duration: 10s
    - from: 50
      to: 100
      duration: 10s
    - from: 100
      to: 200
      duration: 10s
```

```
startup:
  type: once
  times: 1000
rps:
  type: burst
  base: 100
  spike: 500
  duration: 30s
  period: 10s
```

8 Актуальность и значимость

8.1 Сценарии использования

Рассмотрим ожидаемые сценарии использования платформы:

- Начинающие преподаватели дисциплин, обучающих алгоритмам и структурам данных, не имеющие в распоряжении платформы для тестирования, могут свободно ис-

пользовать материалы репозитория для настройки собственной платформы тестирования.

- Студенты начальных курсов могут использовать платформу, как образовательный проект, позволяющий ознакомиться с методикой бессерверных вычислений. При этом применить платформу они могут в рамках факультативных курсов, где не подразумевается тестирование задач.

9 Дальнейшее развитие платформы

Среди возможных сценариев развития платформы можно выделить три категории.

К первой категории относится работа над удобством эксплуатации платформы для конечных пользователей платформы: улучшение web-интерфейса, внедрение текстового редактора с подсветкой синтаксиса. Одним из ключевых нововведений, уникальных для подобных платформ, было бы внедрение нейронных языковых моделей, способных давать студентам подсказки касательно их кода.

Также потенциал к развитию имеет и сфера администрирования платформы: установка компонент и мониторинг. В частности, установку компонент системы возможно объединить в единый скрипт для командной строки, а для просмотра посылок преподавателем может быть добавлен отдельный интерфейс.

Для расширения сценариев использования и аудитории платформы в целом, возможно неограниченное количество решений: от простых в реализации, как например добавление альтернативных способов регистрации и авторизации, до узконаправленных, но соответствующих тематике платформы, среди которых внедрение fuzzing и monkey тестирований.

10 Полученные результаты

В рамках работы была разработана платформа проверки решений задач по программированию, позволяющая оценивать корректность работы решения и измерять метрики качества выполнения кода. Архитектура решения полностью основана на serverless-стэке Яндекс Облака: бессерверные вычисления предоставляют возможность гибкого плана тарификации и «автоматическое» масштабирование. Ключевым отличием полученной платформы является широкая возможность кастомизации платформы, гибкая система тарификации и простота в администрировании.

Для выбора подхода к решению были проанализированы подходы построения схожих систем: начиная с традиционного подхода «по требованию» и заканчивая FaaS подходом. По результатам сравнения была составлена характеристика основных тенденций существующих решений, включая решения с открытым исходным кодом и коммерческие продукты. В статье была продемонстрирована общая архитектура решения, объяснены ключевые детали реализации.

Для внедрения платформы добавлена инструкция по её настройке, а также администрированию. Для удобства её использования также описан пользовательский интерфейс.

Исходный код размещён на [GitHub](#).

Список источников

- [1] Платформа проверки решений тренировочных задач по программированию **Яндекс.Контест**, 2024
URL: <https://contest.yandex.ru/edu>
- [2] Платформа проверки решений тренировочных задач по программированию **LeetCode**, 2024
URL: <https://leetcode.com/>
- [3] Проект по публикации образовательных онлайн-курсов **Coursera**, 2024
URL: <https://www.coursera.org/>
- [4] Как запустить 100+ компиляторов и выстоять. Опыт **Яндекс.Контеста**, 2021
URL: <https://habr.com/ru/companies/yandex/articles/>
- [5] Serverless is More: From PaaS to Present Cloud Computing, 2018
Авторы: E. Eyk, L. Toader, S. Talluri, A. Iosup, L. Versluis & Alexandru Uta
URL: <https://www.researchgate.net/publication/328088482>
- [6] Horizontal Pod Autoscaling in **Kubernetes**, 2024
URL: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>
- [7] Платформа проведения соревнований по программированию **Codeforces**, 2024
URL: <https://codeforces.com/>
- [8] Как заставить код выполняться за одинаковое время? Способы от **Яндекс.Контеста**, 2020
URL: <https://habr.com/ru/companies/yandex/>
- [9] Документация и обзор **Docker.Containers**, 2024
URL: <https://www.docker.com/resources/what-container/>
- [10] What Is **Intel** Turbo Boost Technology?, 2024
URL: <https://www.intel.com/>
- [11] Документация **Overlay Filesystem**, 2024
URL: <https://docs.kernel.org/filesystems/overlayfs.html>
- [12] Чемпионат по программированию **Yandex Cup**, 2023
URL: <https://yandex.ru/cup/>
- [13] Система автоматической проверки задач **Ejudge**, 2023
URL: https://ejudge.ru/wiki/index.php/Main_Page
- [14] Репозиторий с исходным кодом системы **Ejudge** на **Github**, 2024
URL: <https://github.com/blackav/ejudge>
- [15] Документация высокоуровневого **Python**-фреймворка для создания веб-приложений **Django**, 2024
URL: <https://docs.djangoproject.com/en/5.0/>
- [16] Документация **Python**-фреймворка для создания веб-приложений **Flask**, 2024
URL: <https://flask.palletsprojects.com/en/3.0.x/>
- [17] Использование **Ejudge** в **ВШЭ** для проведения курса АКОС, 2024
URL: <https://caos2023.myltsev.ru/>

- [18] Использование **Ejudge** в **МФТИ** для проведения курса АКОС, 2024
URL: <https://ejudge.atp-fivt.org/>
- [19] Использование **Ejudge** в **МФТИ** для проведения курса АКОС, 2024
URL: <https://ejudge.atp-fivt.org/>
- [20] Использование **Ejudge** в **КФУ** для проведения курсов по программированию, 2024
URL: <https://ejudge.kpfu.ru/>
- [21] Настройка по установке системы **Ejudge**, 2023
URL: https://ejudge.ru/wiki/index.php/Инсталляция_системы_ejudge
- [22] The Rise of Serverless Computing , 2019
Communications of the ACM, Vol. 62
Авторы: Paul Castro, Vatche Ishakian, Vinod Muthusamy & Aleksander Slominski
URL: <https://www.researchgate.net/publication/337429660>
- [23] Бессерверные вычисления в **AWS**, 2024
URL: <https://aws.amazon.com/ru/serverless/>
- [24] Документация API Gateway в **Яндекс.Облаке**, 2024
URL: <https://yandex.cloud/ru/docs/api-gateway/>
- [25] Документация Object Storage в **Яндекс.Облаке**, 2024
URL: <https://yandex.cloud/ru/docs/storage/>
- [26] Документация S3 в **AWS**, 2024
URL: <https://aws.amazon.com/ru/s3/>
- [27] Блог изменений в **Яндекс.Облаке**, 2024
URL: <https://yandex.cloud/ru/blog>
- [28] Репозиторий примеров работы с **Яндекс.Облаком**, 2024
URL: <https://github.com/yandex-cloud-examples>
- [29] Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective, 2016
Авторы: J. Opara-Martins, R.Sahandi & F. Tian (Bournemouth University)
URL: <https://dl.acm.org/doi/10.1186/s13677-016-0054-z>
- [30] What is an API Gateway? Статья от **Nginx**, 2024
URL: <https://www.nginx.com/learn/api-gateway/>
- [31] Cloud Functions overview by **Google**, 2024
URL: <https://cloud.google.com/functions/docs/concepts/overview>
- [32] Cloud Computing: Comparison and Analysis of Cloud Service Providers—AWS, Microsoft and Google, 2020
Авторы: Manish Saraswat & R.C. Tripathi (TMU)
URL: <https://ieeexplore.ieee.org/document/9337100>
- [33] Облачный провайдер Market Share Trend, 2023
Отчёт Synergy Research Group (**SRG**)
URL: <https://www.srgresearch.com/articles/>

- [34] Репозиторий C++ фреймворка тестирования задач **tcframe** на **Github**, 2024
URL: <https://github.com/ia-toki/tcframe>
- [35] Документация «агентов» для нагрузочного тестирования в **Yandex Cloud**, 2024
URL: <https://yandex.cloud/ru/docs/load-testing/concepts/agent>
- [36] Introducing tcframe: A Simple and Robust Test Cases Generation Framework, 2015
Автор: Ashar FUADI
Indonesia Computing Olympiad Alumni Association
URL: <https://www.srgresearch.com/articles/>
- [37] Документация serverless решений в **Microsoft Azure**, 2024
URL: <https://azure.microsoft.com/en-us/solutions/serverless>
- [38] Документация serverless решений в **Google Cloud**, 2024
URL: <https://cloud.google.com/serverless>
- [39] Документация serverless решений в **Yandex Cloud**, 2024
URL: <https://yandex.cloud/ru/solutions/serverless>
- [40] Документация для **Locust**: an open-source performance& load testing tool, 2024
URL: <https://docs.locust.io/en/stable/>
- [41] Репозиторий **Locust** на **GitHub**
URL: <https://github.com/locustio/locust>
- [42] Система интеграции процесса авторизации и аутентификации от **Auth0**
URL: <https://auth0.com/>
- [43] Cognito — встроенная в **AWS** система авторизации и аутентификации
URL: <https://aws.amazon.com/cognito/>
- [44] **PyJWT** — a Python library used to encode and decode JSON Web Tokens (JWT)
URL: <https://pyjwt.readthedocs.io/en/latest/>
- [45] **Telegram Bot API** — an HTTP-based interface created for building bots for Telegram
URL: <https://core.telegram.org/bots/api>
- [46] **Boto3** library — writing software that makes use of Amazon S3 and Amazon EC2
URL: <https://pypi.org/project/boto3/>
- [47] Конспекты по дисциплине АиСД на **Wiki ФКН**
URL: <http://wiki.cs.hse.ru/>
- [48] PaaS vs. IaaS vs. SaaS vs. CaaS: How are they different? Статья от **Google Cloud**
URL: <https://cloud.google.com/learn/paas-vs-iaas-vs-saas>