Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА СРЕДЫ ДЛЯ ИСПОЛНЕНИЯ И ВАЛИДАЦИИ ИСХОДНОГО КОДА

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Арутюняна Арсена Шаваршевича

Научный руководитель
зав. кафедрой, к. ф.-м. н. _____ 10.06.17 С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ 10.06.17 С. В. Миронов

Саратов 2017

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные системы дистанционного обучения представляют из себя порталы, на которых размещены задания, распределнные по соответствующим им курсам. Курсы, направленные на подготовку инженеров в области программного обеспечения, зачастую содержат задания, требующие написания программного кода на различных языках программирования. Большинство таких порталов поддерживают загрузку исходного кода в качестве выполненного задания, но только единицы поддерживают его исполнение. Это обусловлено рядом причин:

— широкое разнообразие языков программирования, на которых студент может написать программный код, требует наличия соответствующих пакетов на серверах портала. Положение осложняется еще тем, что код, написанный студентом, может быть несовместим с версией компилятора или интерпретатора, расположенного на сервере;

— с ростом нагрузки на портал, требуется увеличение количества серверов, на которых исполняются программы, что ведет к проблеме масштабируемости такого портала;

— требуется возможность запуска программного кода, состоящего из множества модулей.

Современные технологии контейнерной виртуализации позволяют решить вышеизложенные проблемы. Целью данной работы является разработка прототипа программного комплекса, предоставляющего интерфейсы загрузки, исполнения и валидации кода, а также добавления новых заданий в систему. В основе комплекса лежит виртуализация рабочих пространств, позволяющих быстро исполнять исходный код в безопасном и изолированном режиме.

1 Обзор существующих систем

Проблема проверки решений различных задач по программированию в автоматизированном режиме является достаточно актуальной, поскольку просмотр кода преподавателем, его запуск и валидация — трудозатратная процедура. Системы, позволяющие автоматизированно проверять работы студента, разрабатываются с целью решения такого рода проблемы.

В настоящий момент существует множество таких систем, как отечественных так и зарубежных. При выборе такой системы надо обращать внимание на ключевые возможности такого рода систем, такие как:

— количество языков, поддерживаемых системой;
— возможность добавления или удаления языка из системы;
— наличие банка задач и возможность пополнять его;
— безопасность системы;
— возможность тестирования на эталонных входных параметрах задачи;
— степень независимости от разработчиков системы.

Далее в работе описаны распространенные системы тестирования и запуска кода:

— Ejudge
— Contester
— ProgrammingContestControl
— Ideone
— Контестер

Все вышеизложенные продукты выполняют в той или иной степени возложенные на них задачи, но они в полной мере не соответствуют требованиям, изложенным в начале главы. Общим для всех систем недостатком является отсутствие возможности загрузки многомодульных программ и ограниченность и невозможность пользователем кастомизации команды запуска. Стремительное развитие технологий позволяет формировать новые архитектурные подходы к построению систем, в том числе и подобным выше.

2 Основные инструменты реализации программного комплекса

Основной функциональной возможностью комплекса является виртуализация рабочего пространства пользователя. Для конечного пользователя процесс виртуализации должен быть незаметен и занимать не больше нескольких секунд. Система должна поддерживать максимально возможное количество параллельно запущенных виртуальных пространств.

Существует ряд различных техник виртуализации и решений эти техники реализующие, но изложенные требования заметно сужают область поиска необходимого программного решения. Одной из такой техник является виртуализация уровня операционной системы.

Виртуализация на уровне операционной системы — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя, вместо одного. Эти экземпляры (часто называемые контейнерами или зонами) с точки зрения пользователя полностью идентичны реальному серверу [1, 2]. Для систем на базе UNIX, эта технология может рассматриваться как улучшенная реализация механизма chroot. Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

Проанализировав существующие реализации данного метода виртуализации, было решено использовать Docker ввиду его широкого применения в настоящий момент. Поскольку Docker написан на языке Go, язык обладает богатой библиотекой для взаимодействия с этой технологией. Поэтому Go был выбран основным языком реализации программного решения. Приложение включает в себя веб-клиент, который было решено написать на JavaScript фреймворке Vue.js.

Исходя из поставленной цели и проведенного анализа были выделены следующие задачи:

— изучить технологию контейнерой виртуализации Docker;
— изучить возможности языка Go;
— изучить веб-фреймворк Vue.js;
— реализовать программный комплекс.

Детальное описание используемых технолгий приведено в последующих

разделах.

2.1 Технология контейнерной виртуализации Docker

Docker — инструмент, облегчающий создание, упаковку, запуск и распространение приложений, используя контейнеры. Контейнеры позволяют разработчику запаковать приложение со всеми его зависимостями в единый пакет, который гарантированно запустится на любой другой Linux машине без дополнительных настроек.

Docker контейнеры схожи с виртуальными машинами. Главное отличие, что вместо создания полностью виртуальной операционной системы, Docker позволяет использовать процессам контейнера то же ядро Linux, что и система на которой этот контейнер запущен. Такой подход дает значительное увеличение производительности и уменьшает объемы потребляемых ресурсов.

Важно, Docker является проектом с открытым исходным кодом, любой человек или компания может внести свой вклад в базу кода и расширить функциональность в соответствии с собственными потребностями [3, 4].

2.2 Язык Go

Язык Go, часто именуемый golang для упрощения поиска сведений о нем в Интернете, — это статически типизированный и компилируемый язык программирования с открытым исходным кодом, разработку которого ведет компания Google. Ключевую роль в разработке языка сыграли Роберт Грисемер (Robert Griesemer), Роб Пайк (Rob Pike) и Кен Томпсон (Ken Thompson), сделалав попытку создать язык, нацеленный на масштабирование современных программных систем. Язык имеет открытый исходный код, благодаря чему в процессе разработки участвуют разработчики со всего мира.

Go — не обычный статически типизированный и компилируемый язык. Его статическая типизация имеет черты, делающие ее похожей на динамическую, а скомпилированные двоичные файлы включают среду выполнения со встроенным сборщиком мусора. На структуру языка наложили отпечаток проекты, для которых он должен был использоваться в Google: большие проекты, поддерживающие масштабирование и разрабатываемые многочисленными группами разработчиков.

Простота Go означает, что он поддерживает не все возможности, имеющиеся в других языках. Например, в языке Go отсутствуют тернарный оператор (обычно это ?:) и обобщенные типы. Не содержит он и некоторых других функциональных возможностей, присутствующих в современных языках, что служит поводом для его критики, но это не должно служить поводом для отказа от использования языка Go [5]. В мире программирования одна и та же задача часто может быть решена множеством способов. Даже если в Go отсутствуют какие-то возможности, имеющиеся в других языках, вместо них он предоставляет иные пути решения проблем, ничуть не хуже.

Несмотря на простоту ядра языка Go, встроенная система управления пакетами позволяет добавлять в него дополнительные возможности. Многие из недостающих элементов можно подключить как сторонние пакеты и включить в приложение.

Минимальный размер и сложность дают свои преимущества. Язык можно быстро освоить, и он хорошо запоминается. Это важное преимущество, когда требуется быстро исследовать чужой код [6]. В настоящее время область применения языка Go гораздо шире задуманной. Множество инструментов, работающих под высокой нагрузкой, написаны на языке Go, который сейчас используется как высокопроизводительный язык программирования общего назначения.

2.3 Веб-фреймворк Vue.js

Vue (произносится /vju:/, примерно как view) — это прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от фреймворков-монолитов, Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для создания сложных одностраничных приложений (SPA, Single-Page Applications), если использовать его совместно с современными инструментами и дополнительными библиотеками.

Другой важной концепцией Vue являются компоненты [7] Эта абстракция позволяет собирать большие приложения из меньших кусочков. Компоненты представляют собой пригодные к повторному использованию объекты. Не вдаваясь в подробности, можно сказать, что компоненты — это пользова-

тельские элементы, к которым компилятор Vue привязывает определённое поведение [8, 9]

3 Реализация программного комплекса

Данный раздел представляет практическую часть работы. Здесь описаны архитектура разработанной системы и взаимодействие ее компонентов.

3.1 Принцип работы системы

Основной задачей системы является запуск виртуальных рабочих пространств, в которых может исполняться исходный код студента. Как уже упоминалось ранее, основным инструментом для этого в данной работе служит Docker. Соответственно, этот инструмент должен быть предустановлен прежде осуществелния запуска программного комплекса.

3.2 Компоненты системы

Система состоит из двух основных компонентов:

— Серверная часть — обработка поступающих запросов на запуск контейнеров;

— Клиентская часть — веб-приложение, предоставляющее интерфейс для загрузки кода и его запуска.

Разберем каждый компонент по отдельности.

3.2.1 Структура проекта

Ниже приведен список модулей проекта и их краткое описание:

1. Модуль `Godeps` служит хранилищем мета-информации о зависимостях, которые были использованы при разработке системы;
2. Модуль `common` представляет из себя Go-пакет, в котором создаются общие для всего проекта переменные;
3. Модуль `models` является Go-пакетом, который содержит описание основных моделей, используемых в проекте;
4. Модуль `public` содержит исходный код веб-клиента, написанного на популярном JavaScript фреймворке Vue.js;
5. Модуль `util` — это Go-пакет, содержащий утилитные методы;
6. Модуль `vendor` содержит все зависимые Go-пакеты, необходимые для разработки системы.

3.2.2 Серверная часть

Как уже было сказано ранее, серверная часть системы написана на языке Go [10, 11]. Отправной точкой в проекте является файл `main.go`. В этом

файле инициализируются глобальные переменные. Например Docker-клиент из пакета `common`:

```
common.CLI, err = client.NewEnvClient()
if err != nil {
    panic(err)
}
```

Далее, создается HTTP-сервер, который будет обслуживать запросы на загрузку, исполнение, валидацию кода.

```
var httpAddr = flag.String("http", "0.0.0.0:8080", "HTTP service address")
flag.Parse()

e := echo.New()
```

Чтобы HTTP-сервер смог обслуживать запросы, необходимо к так называемым HTTP endpoints привязать управляющие функции:

```
e.Static("/", "public")
e.POST("/upload", handlers.UploadHandler)
e.POST("/tasks/launch", handlers.LaunchWorkHandler)
e.POST("/tasks", handlers.CreateTaskHandler)
e.GET("/tasks", handlers.GetTasksHandler)
e.GET("/images", handlers.GetImagesHandler)
e.GET("/images/search", handlers.SearchImageHandler)
e.GET("/images/:name/tags", handlers.ListImageTagsHandler)
e.GET("/images/:nameprefix/:namesuffix/tags", handlers.ListImageTagsHandler)
e.GET("/images/pull", handlers.PullImageHandler)

```

Теперь, для работы с сервером, необходимо посылать HTTP-запросы на определенные эндпойнты.

3.2.3 Клиентская часть

Клиентская часть представляет из себя веб-приложение, написанное на популярном JavaScript фреймворке Vue.js. Веб-интерфейс позволяет:

— навигироваться по существующим задачам;

— создавать новые задачи;

— загружать работу;

— валидировать работу на разных входных значениях.

4 Работа с системой

Исходный код программного комплекса находится в публичном репозитории на GitHub. Проект распространяется под GNU General Public License.

Следующие инструменты понадобятся для функционирования приложения и работы с ним:

1. Git клиент.
2. Инструмент Docker.
3. Пакетный менеджер npm.
4. Компилятор языка Go.
5. Веб-браузер.
6. Утилита cURL.

В последующих главах будут рассмотрены развертывание системы и основные правила взаимодействия с ее компонентами.

4.1 Работа с веб-клиентом

Рассмотрим основные интерфейсы веб-клиента.

На рисунке 1 изображен интерфейс создания задания.

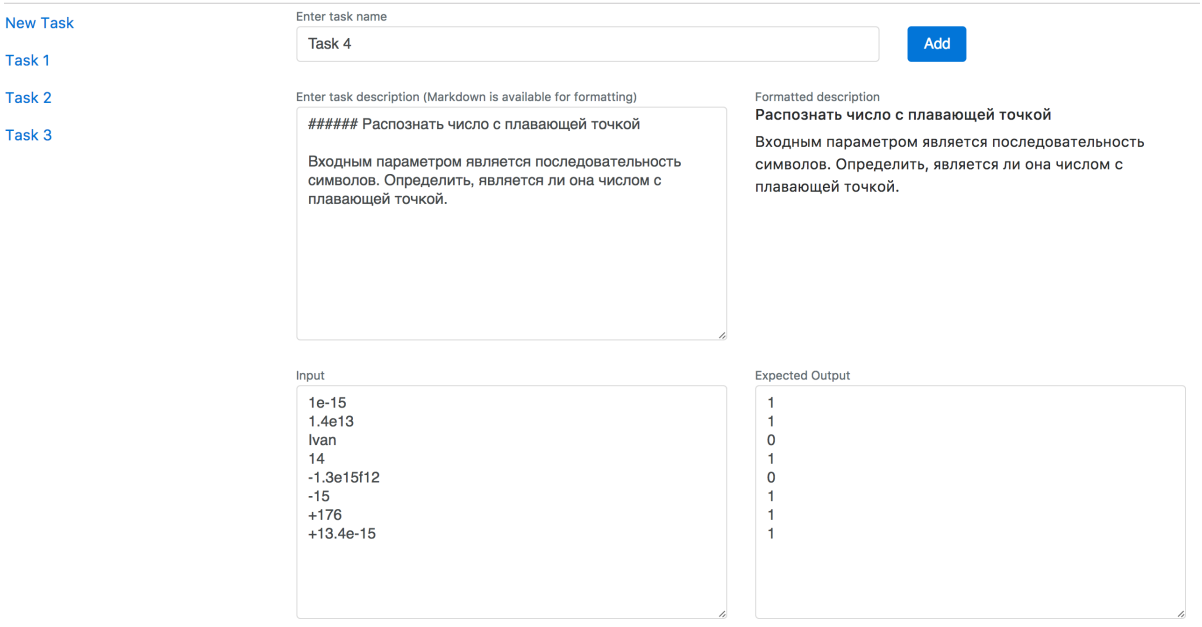

Рисунок 1 – Интерфейс создания задания

Для того чтобы исполнить загруженный исходный код, необходимо заполнить все поля на странице задания и нажать на кнопку `Launch`. В результате чего загруженный код выполнится на тестовых входных параметрах задания. Входные параметры можно переопределить и повторно нажать

на кнопку выполнения задания. Результат исполнения кода изображен на рисунке 2

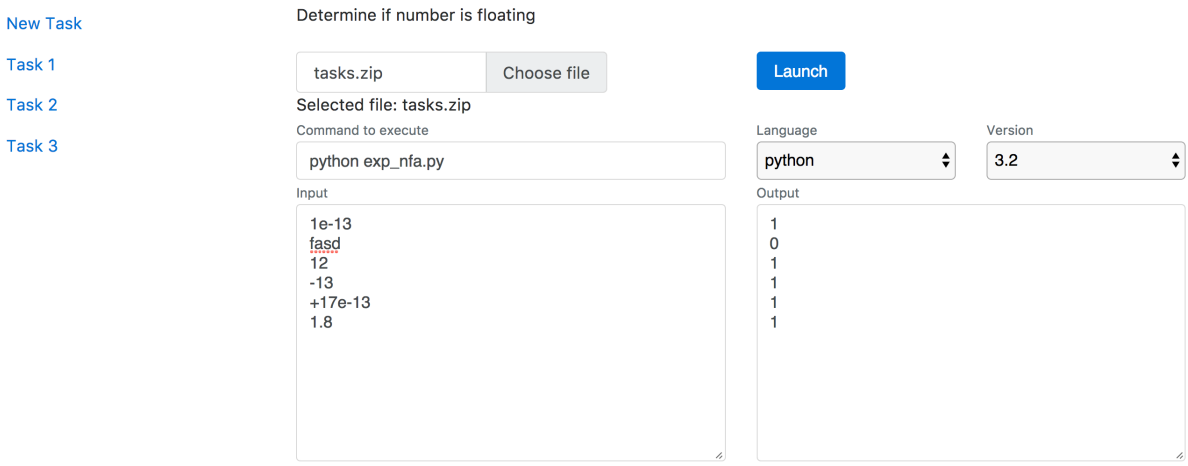

Рисунок 2 – Интерфейс исполнения кода

ЗАКЛЮЧЕНИЕ

В результате данной работы был написан программный комплекс для исполнения и валидации программного кода. Комплекс выполняет возложенные на него задачи:

— загрузка исходного кода на удаленный сервер;
— исполнение загруженного кода на сервере;
— валидация кода на заданных входных значениях;
— возможность добавления в систему новых заданий.

Помимо разработанного программного комплекса были:

— изучены возможности языка Go и применены при разработке серверной части программного комплекса;
— получены навыки веб-разработки на фреймворке Vue.js;
— получены навыки разработки сложных систем.

Разработанная система может быть развернута как на публичной инфраструктуре и использоваться в качестве общедоступного API для написания собственных клиентов для автоматического тестирования кода, так и в приватной инфраструктуре портала, организации, университета.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Josyula, V.* Cloud Computing: Automating the Virtualized Data Center / V. Josyula, M. Orr, G. Page. Cisco Press Networking Technology. — Cisco Press, 2012.

2 *Reifschneider, O.* Evaluation of Linux Container Virtualization Technology for Web Application Deployment / O. Reifschneider. — 2014.

3 *Turnbull, J.* The Docker Book: Containerization is the new virtualization / J. Turnbull. — James Turnbull, 2014.

4 Guide | Echo - High perfomance, minimalist Go web-framework [Электронный ресурс]. — URL: `https://echo.labstack.com/guide` (Дата обращения 02.02.2017). Загл. с экр. Яз. англ.

5 *Butcher, M.* Go in Practice / M. Butcher, M. Farina. — Manning Publications Company, 2016.

6 *Ryer, M.* Go Programming Blueprints / M. Ryer. Community experience distilled. — Packt Publishing, 2016.

7 *Banks, A.* Learning React: Functional Web Development with React and Redux / A. Banks, E. Porcello. — O'Reilly Media, 2017.

8 *Kyriakidis, A.* The Majesty of Vue.js / A. Kyriakidis, K. Maniatis. — Packt Publishing, 2016.

9 Что такое Vuex? [Электронный ресурс]. — URL: `https://vuex.vuejs.org/ru/intro.html` (Дата обращения 02.05.2017). Загл. с экр. Яз. рус.

10 *Richardson, L.* RESTful Web Services / L. Richardson, S. Ruby. — O'Reilly Media, 2008.

11 *Gourley, D.* HTTP: The Definitive Guide: The Definitive Guide / D. Gourley, B. Totty, M. Sayer, A. Aggarwal, S. Reddy. Definitive Guides. — O'Reilly Media, 2002.

10.06.2017