МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

СОЗДАНИЕ И ПОДДЕРЖКА НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ ДЛЯ РАЗРАБОТКИ И ТЕСТИРОВАНИЯ MIRANTIS CONTAINER CLOUD

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Лалова Сергея Андреевича

Научный руководитель
доцент, к. ф.-м. н. _____ А. С. Иванова

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

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Скорость сборки продукта - важное конкурентное преимущество в разработке программного обеспечения. Путь к ускорению релизов лежит через автоматизацию и внедрение CI/CD. С увеличением и усложнением проекта, для слаженной работы команд, необходима правильно настроенная инфраструктура. Для построения идеальной инфраструктуры не достаточно придерживаться лучших практик и использовать передовые технологии. Выбранное решение должно основываться на специфике конкретного проекта.

Цель настоящей работы является разработка и модификация непрерывной интеграции для тестирования и разработки продукта Mirantis Container Cloud компании Мирантис. Для поставленной цели должны быть выполнены следующие задачи:

- создание адаптивных функциональных тестов для разных провайдеров;
- разработка триггеров для запуска новых тестов;
- создание запланированных еженедельных тестовых запусков;
- анализ и перенос необходимых тестов в еженедельные запуски;
- обновление запланированных ежедневных тестовых запусков.

1 Теоретическая часть

В теоретической части работы будут:

— разобран термин непрерывной интеграции;
— проведен анализ данного подхода для разных проектов;
— описан процесс тестирования в рамках непрерывной интеграции;
— рассмотрен процесс разработки в рамках непрерывной интеграции;
— проведен анализ основных инструментов для реализации непрерывной интеграции;
— рассмотрены инструменты Docker и Kubernetes;
— описание проекта Mirantis Container Cloud.

1.1 Непрерывная интеграция

Непрерывная интеграция и непрерывная поставка представляют собой набор принципов и практик, которые позволяют разработчикам чаще и надежнее разворачивать и тестировать изменения программного обеспечения. С точки зрения DevOps инженера, CI/CD — это одна из DevOps-практик.

Непрерывная интеграция — это методология разработки, при которых в код вносятся изменения с частыми коммитами.Так как большинство современных приложений разрабатываются с использованием платформ и инструментов, то появляется необходимость в механизме интеграции и тестировании.

1.1.1 Разработка

С точки зрения разработки непрерывная интеграция во многом упрощает, а как следствие и ускоряет процесс разработки.

Выделим основные плюсы данного подхода для разработчика:

— разработчик и клиент стремятся к увеличению скорости посредством внесения изменений и согласования правок;
— реализация среды. У разработчиков настроено единое окружение, что ускоряет взаимодействие при совместной работе;
— методология уменьшает время запуска обновлений до нескольких дней. Благодаря этому, разработчики могут более быстро тестировать изменения и внедрять решения быстрее конкурентов на рынке.

1.1.2 Тестирование

Одним из главных плюсов CI/CD является минимизация ручного тестирования. Для данного подхода продукт покрывается автоматизированными тестами, которые в свою очередь запускают CI, запуск которых и является одной из его основных функций.

Для максимальной автоматизации процесса, необходимо разработать матрицу проверок для успешного прохождения сценария.

Проверка на работоспособность приложения достаточно проста в настройке, но несет огромную функцию - сокращение времени. Так же необходимо внести проверку Code style в CI/CD, для сокращения времени на проверку данного функционала другими разработчиками.

Следующим шагом в настройке CI является настройка юнит тестов с анализом покрытия, а также интеграционные и системные тесты. Также при необходимости могут быть добавлены нефункциональные тесты.

Помимо автоматического запуска тестов и выполнения определенных метрик качества кода, CI/CD позволяет QA специалистам выстроить релизный процесс.

Основная цель непрерывной поставки (CD) — максимально ускорить поставку новых релизов клиентам. Это невозможно без автоматического тестирования. Стоит отметить, что для автоматизации поставки кода необходимо постараться полностью избавиться от ручного тестирования.

Процесс CD — лишь часть сложного конвейера развертывания. CD следует за этапом непрерывной интеграции (CI).

1.2 Сравнение инструментов для реализации CI/CD

В данном разделе были рассмотрены инструменты, которые необходимы для создания собственного конвейера CI/CD. Важно отметить, что будут рассмотрены только базовые наборы инструментов DevOps для внедрения, эксплуатации, ввода в действие и оптимизации конвейеров CI/CD. Практическое применение для создания непрерывной интеграции будут рассмотрены в практической части работы. [1]

1.2.1 Jenkins

Jenkins - это сервер автоматизации с открытым исходным кодом, в котором происходит центральная сборка и непрерывный процесс интеграции. [2]

1.2.2 Bitbucket Pipelines

Bitbucket Pipelines — это инструмент CI, напрямую интегрированный в Bitbucket, облачную систему контроля версий Atlassian. Bitbucket Pipelines дополнительно предлагает непрерывную поставку (CD). Это означает, что проекты, созданные с помощью Bitbucket Pipelines, можно развернуть и в рабочей инфраструктуре.

1.2.3 GitLab

Данный инструмент входит в набор инструментов, используемых для управления различными аспектами жизненного цикла разработки программного обеспечения. Основным продуктом является веб-менеджер репозитория Git с такими функциями, как отслеживание проблем и аналитика.

Платформа позволяет маркетологам запускать сборки, запускать тесты и разворачивать код при каждой фиксации или отправке. Позволяет создавать задания на виртуальной машине, в контейнере Docker или на другом сервере.

1.2.4 TeamCity

TeamCity — это сервер непрерывной интеграции от JetBrain, который позволяет создавать и разворачивать множество проектов, включая интеграцию с Visual Studios и IDE.

1.3 Docker

При описании продукта для тестирования и разработки непрерывной интеграции которого проводилась практическая работа необходимо дать определения таким инструмента как Docker и Kubernetes.

Docker — это открытая платформа для разработки, доставки и запуска приложений. Docker позволяет отделить приложения от инфраструктуры, для достижения быстрой доставки программного обеспечения. С помощью Docker можно управлять инфраструктурой. Воспользовавшись методологиями Docker для быстрой доставки, тестирования и развертывания кода, можно значительно сократить задержку между написанием кода и его запуском в рабочей среде. [3]

Docker оптимизирует жизненный цикл разработки, позволяя разработчикам работать в стандартизированных средах, используя локальные контейнеры, которые предоставляют приложения и службы. Контейнеры отлично подходят для рабочих процессов непрерывной интеграции и непрерывной доставки

(CI/CD).

1.4 Kubernetes

Kubernetes, также известная как K8s, представляет собой платформу с открытым исходным кодом, используемую для управления контейнерами Linux в частных, общедоступных и гибридных облачных средах. Компании также могут использовать Kubernetes для управления микросервисными архитектурами. Контейнеры и Kubernetes можно развернуть у большинства облачных провайдеров.

Разработчики приложений, системные администраторы и инженеры DevOps используют Kubernetes для автоматического развертывания, масштабирования, обслуживания, планирования и эксплуатации нескольких контейнеров приложений в кластерах узлов. Контейнеры работают поверх общей операционной системы на хост-компьютерах, но изолированы друг от друга, если только пользователь не решит их подключить. [4]

1.5 Описание продукта

В данной главе описан продукт в рамках которого производилась практическая работа.

Mirantis Container Cloud (ранее Docker Enterprise Container Cloud) предлагает предприятиям беспрецедентную скорость для более быстрой доставки кода в публичные облака и локальную инфраструктуру.

Container Cloud интегрируется с Lens, самой популярной в мире интегрированной средой разработки Kubernetes, чтобы расширить возможности нового поколения разработчиков Kubernetes. Container Cloud обеспечивает данные функции за счет автоматизированного управления жизненным циклом полного стека и непрерывных обновлений, а также путем предоставления инструментов для оперативного анализа и управления, поддерживающих разработку облачного программного обеспечения. [5]

Также стоит заметить, что Container Cloud позволяет разворачивать собственные мультиоблако.

2 Практическая часть

В практической части работы будет продемонстрировано написание функциональных тестов и дальнейшее их внедрение в CI продукта. Также будет решена задача переноса части функциональных тестов в еженедельные запуски и как следствие модификация ежедневных запусков.

2.1 Создание адаптивных функциональных тестов

Для разработки функциональных тестов и деплоя продукта на разные провайдеры используется собственный фреймворк si-tests. Данный фреймворк был разработан для удобного тестирования собственно разработанных Kubernetes ресурсов. На примере функционального теста будут показаны основные возможности решения задач по покрытию новых изменений продукта.

В si-tests используется среда для тестирования pytest, о которой было рассказано в теоретической части работы.

Для решения задачи тестирования прокси на дочернем кластере Kubernetes необходимо провести аналитическую работу, оценить требования и выбрать максимально подходящую структуру для данного сценария.

Исходя из технического задания и аналитики архитектуры можно сделать вывод, что данный функционал должен работать независимо от провайдера. Соответственно, одним из лучших решений будет объединить проверки в один сценарий. [6]

Специфика функционального тестирования продукта заключается в приведении кластера в исходное состояние после необходимых проверок, за исключением сценариев обновления. [7]

2.2 Разработка триггеров

Триггеры используются для запуска тестов в еженедельных и ежедневных тестовых запусках, а также для тестирования после изменения функционала. Триггеры задаются в jenkins job и передаются непосредственно в сценарии определенного деплоя или теста. [8]

Для запуска разработанного теста необходимо создать jenkins job, которую могут использовать разные команды для своих индивидуальных сценариев.

Для демонстрации архитектуры и реализации решения запуска тестов по триггеру был рассмотрен провайдер Openstack. Стоит отметить, что реализация триггеров не зависит от выбранного провайдера.

Данное решение будет реализовано в проекте продукта Mirantis Container Cloud и использует актуальный код мастера. Необходимо определить триггеры, реализация которых была продемонстрирована выше.

Помимо обычных чайлдов, продукт MCC позволяет использовать мультирегиональные сценарии. Пример мультирегионального кластера - административный кластер имеет провайдер aws, а дополнительный кластер имеет провайдер azure. Так что, при деплое чайлд кластеров при данном сценарии, будут развернуты два чайлда на каждом из представленных провайдеров.

Триггеры для включения тестов определяются в функции commonParameters, это связанно с настройкой CI. При запуске определенных тестов, выбранные тесты запускаются на всех выбранных провайдерах. Именно поэтому булевские параметры определяются до непосредственно определения переменных для провайдеров. [9]

Результатом работы является запуск сценария на выбранном провайдере. [10]

2.3 Создание запланированных еженедельных тестовых запусков

С увеличением функционала продукта, ежедневное тестирование становится более сложной задачей, это связанно в первую очередь с увеличением функциональных возможностей, а как следствие и функциональных тестов. С ростом функциональных тестов увеличивается время прохождения сценария для каждого провайдера.

Данную проблему можно решить распределением тестов на ежедневные и еженедельные запуски. В рамках данной задачи необходимо провести аналитическую работу и выяснить какие из тестов необходимы для ежедневной проверки.

Для уменьшения времени прохождения ежедневных прогонов было сокращенно количество тестов до минимально необходимого рабочего функционала, максимально приближенного к используемому конечным пользователем.

Еженедельная матрица включает в себя необходимые функциональные тесты, но в связи с тем, что с каждым релизом функциональных возможностей становится больше, данная матрица должна являться более гибкой. Помимо этого, можно оптимизировать еженедельную матрицу, сократив прогоны одинаковых тестов на разных провайдер, исключив возможность отличий в архитектуре провайдеров.

Одним из основным функционалом, который затрагивает всех пользователей является обновление главного и дочерних кластеров. Прохождение функционала реализованного в более ранние релизы и невыпущенного, было вынесено в еженедельные прогоны.

Результатами перехода на обновленную матрицу ежедневных прогонов, являются:

- более частые успешные прохождение сценариев, а как следствие более наглядное понимание готовности продукта;
- уменьшение времени прохождения сценариев в несколько раз;
- уменьшение нагрузки облачных провайдеров, в связи с уменьшением времени прохождения.

ЗАКЛЮЧЕНИЕ

Целью работы являлась создание и поддержка непрерывной интеграции для разработки и тестирования.

В теоретической части работы был разобран термин непрерывной интеграции, проведен анализ данного подхода для разных проектов. Также описан процесс тестирования в рамках непрерывной интеграции, рассмотрен процесс разработки в рамках непрерывной интеграции и проведен анализ основных инструментов для реализации непрерывной интеграции. Также рассмотрены инструменты Docker, Kubernetes и описан проект Mirantis Container Cloud.

В практической части работы были выполнены следующие задачи:

- созданы адаптивные функциональные тесты для разных провайдеров;
- разработаны триггеры для запуска новых тестов;
- созданы запланированные еженедельные тестовые запуски;
- проанализированы и перенесены необходимые тесты в еженедельные запуски;
- обновлены запланированные ежедневные тестовые запуски.

В результате проделанной работы было достигнуто:

- уменьшение загруженности среды тестирования посредством сокращения времени ежедневных запусков;
- стабилизирование ежедневных запусков;
- более глубокое тестирование функционала в еженедельных запусках;
- уменьшение платы за ресурсы облачных провайдеров.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Инструменты непрерывной интеграции [Электронный ресурс]. — URL: https://www.atlassian.com/ru/continuous-delivery/continuous-integration/tools (Дата обращения 22.03.2022). Загл. с экр. Яз. рус.

2 *Laster, B.* Jenkins 2: Up and Running / B. Laster. — New York: O'Reilly Media, 2018.

3 *Эдриен, .* Использование Docker / . Эдриен. — М.: O'Reilly Media, 2018.

4 *Марко, .* Kubernetes в действии / . Марко. — М.: ДМК Пресс, 2018.

5 Mirantis Container Cloud [Электронный ресурс]. — URL: https://www.mirantis.com/software/mirantis-container-cloud/ (Дата обращения 21.03.2022). Загл. с экр. Яз. анг.

6 The Kubernetes API [Электронный ресурс]. — URL: https://www.pygame.org/news/2019/3/1-9-5-released-into-the-wilds (Дата обращения 27.03.2022). Загл. с экр. Яз. анг.

7 *Даг, .* Стандартная библиотека Python 3. Справочник с примерами / . Даг. — М.: Вильямс, 2018.

8 *Башар, .-.* Groovy и Grails. Практические советы / .-. Башар. — М.: ДМК Пресс, 2018.

9 *Скотт, .* Linux. Карманный справочник / . Скотт. — М.: Вильямс, 2018.

10 Get Docker [Электронный ресурс]. — URL: https://docs.docker.com/get-docker/ (Дата обращения 01.04.2022). Загл. с экр. Яз. анг.