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

НАСТРОЙКА И АВТОМАТИЗАЦИЯ СРЕДЫ ДЛЯ РАЗВЕРТЫВАНИЯ WEB-ПРИЛОЖЕНИЯ С ПОМОЩЬЮ ИНСТРУМЕНТА НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ JENKINS

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Колесниковой Тамары Анатольевны

Научный руководитель
профессор, д. ф.-м. н. _____ Д. К. Андрейченко

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

Саратов 2018

ВВЕДЕНИЕ

Одним из наиболее сложных и напряженных моментов проектирования программного обеспечения является интеграция. При сборке воедино модулей, разработанных по отдельности, зачастую возникают проблемы, обнаружить которые крайне сложно. В проекте, выполняемом одним человеком с немногими внешними зависимостями, интеграция программного обеспечения — не слишком существенная проблема, но при увеличении сложности проекта возникает насущная потребность в интеграции и проверке слаженной работы компонентов программного обеспечения, причем заранее и часто. Дожидаться конца проекта для проведения интеграции и выявления всего спектра возможных ошибок — неразумно и зачастую приводит к удорожанию и задержке сдачи проекта. Непрерывная интеграция снижает подобные риски [1].

Разные этапы конвейера непрерывного развертывания требуют наличия на компьютере установленного программного обеспечения для выполнения приемочных тестов. Этот процесс необходимо автоматизировать, так как установка вручную слишком трудоемка. Кроме того, автоматизация существенно снижает риски, связанные с развертыванием новых версий в рабочем окружении, поскольку гарантирует единообразие установки необходимых окружений.

Непрерывная интеграция увеличивает возможности обратной связи. Она позволяет следить за состоянием проекта в течение дня. Автоматизация процесса непрерывной интеграции предоставляет преимущества в области повторяемых и склонных к ошибкам процессов. Целью дипломной работы, является автоматизация процесса непрерывного развертывания приложения. Были поставлены следующие задачи:

- разработать Docker контейнеры для запуска микро-сервисов приложения;
- установить интеграционный сервер;
- установить тестовый сервер для развертывания приложения;
- автоматизировать конфигурирование и настройку интеграционного сервера с использованием инструмента управления конфигурациями Ansible;
- сконфигурировать Jenkins;
- разработать Pipeline для непрерывного развертывание приложения.

1 Процесс непрерывной интеграции

Непрерывная интеграция (Continuous Integration) — это процесс интегрирования нового кода, написанного разработчиками, в основной код или ветку «мастер», осуществляемый в течение рабочего дня. Этот подход отличается от методики, в соответствии с которой разработчики работают с независимыми ветками неделями или месяцами, выполняя слияние кода в основную ветку только после полного завершения работы над проектом. Длительные периоды времени между слияниями приводят к тому, что в код вносится очень много изменений, что повышает вероятность появления ошибок. При работе с большими пакетами изменений гораздо труднее изолировать и идентифицировать фрагмент кода, который вызвал сбой. Если же используются небольшие наборы изменений, для которых часто выполняется слияние, поиск ошибок значительно упрощается [2].

В системах непрерывной интеграции после завершения слияния новых изменений обычно автоматически выполняется набор тестов. Эти тесты выполняются после фиксации изменений и завершения слияний. Это позволяет избежать накладных расходов, связанных с использованием ручного труда тестировщиков программного обеспечения. Результаты выполнения этих тестов часто визуализируются. Если результаты выделены зеленым цветом, значит, тест завершился успешно, а интегрированный программный релиз не содержит ошибок. Провальные или «красные» тесты означают, что релиз содержит ошибки и должен быть исправлен. Благодаря использованию этого рабочего потока идентификация и устранение проблем осуществляются намного быстрее [3].

1.1 Инфраструктура как код

Инфраструктура как код — это быстро развивающееся направление, в основе которого лежит использование скриптов для настройки инфраструктуры вычислений вместо настройки компьютеров вручную.

Модель «Инфраструктура как код», которую иногда называют «программируемой инфраструктурой», — это модель, по которой процесс настройки инфраструктуры аналогичен процессу программирования ПО. По сути, она положила начало устранению границ между написанием приложений и созданием сред для этих приложений. Приложения могут содержать скрипты,

которые создают свои собственные виртуальные машины и управляют ими. Это основа облачных вычислений и неотъемлемая часть DevOps.

Системы управления конфигурацией — программы и программные комплексы, позволяющие централизованно управлять конфигурацией множества разнообразных разрозненных операционных систем и прикладного программного обеспечения, работающего в них.

Современные системы управления конфигурацией стремятся к тому, чтобы в полной мере реализовать принцип Infrastructure-as-a-Code, в соответствии с которым вся существующая IT-инфраструктура, машины, их конфигурация, связи между ними могут быть описаны одним или несколькими формальными файлами, а задача системы управления конфигурацией — воплотить описанную конфигурацию в жизнь [4].

При этом, состояние всей инфраструктуры остается обозримым и контролируемым. Ручное выполнение операций на узлах минимизировано или сведено к нулю.

Ansible — система управления конфигурациями, написанная на Python, с использованием декларативного языка разметки для описания конфигураций. Основным преимуществом Ansible перед аналогичными системами является то, что он работает без установки агента на управляемые хосты. Ansible использует ssh для подключения и выполняет изменения с помощью Python модулей [5].

В состав Ansible входит большое количество модулей (библиотека модулей). В текущей библиотеке находится порядка 200 модулей. Модули отвечают за действия, которые выполняет Ansible. При этом каждый модуль, отвечает за свою конкретную и небольшую задачу. Модули можно выполнять отдельно, в ad-hoc командах или собирать в определенный сценарий (play), а затем в playbook [6].

Для моделирования инфраструктуры проекта использовались виртуальные машины. Для создания которых была использована программа Virtual Box. Для удобства управления виртуальными машинами и установки необходимого программного обеспечения использовался Vagrant в комбинации с системой управления конфигурациями Ansible.

Первой задачей стала моделирование инфраструктуры, для развертывания приложения на тестовые сервер. Для этой цели были установлены Virtual

Box и Vagrant.

Virtual Box предоставляет графический интерфейс для управления виртуальными машинами и просмотра, задания характеристик виртуальных машин.

В то время как Vagrant предоставляет удобный CLI для управления ВМ. Базовой операционной системой, используемой на ВМ было решено выбрать Centos:7. Данная ОС обладает следующими преимуществами:

— стабильная версия Linux;
— свободно распространяемая ОС с открытым исходном кодом;
— предоставляет богатый интерфейс для работы в командной строке.

Для использовании ОС Centos необходимо было сначала добавит бокс с ее образом в Vagrant. Затем, был составлен конфигурационной файл Vagrantfile. В котором были указаны общие для всех ВМ характеристики:

После этого, по отдельности были описаны все сервера, входящие в модель инфраструктуры.

Для автоматизации установки необходимого программного обеспечения на сервера была выбрана система управления конфигурациями Ansible, которая предоставляет следующие преимущества:

— не требует установки на управляемом хосте;
— предоставляет широкий набор модулей для управления;
— использует для описания команд интуитивно понятный язык YAML.

Были составлена отдельные Ansible playbook для обеспечения:

— тестового сервера;
— производственного сервера.

На CI сервер необходимо было установить:

— JDK;
— инструмент для сборки Java приложений Maven;
— систему управления конфигурациями Ansible;
— систему управления версиями Git;
— Docker;
— интеграционный сервер Jenkins.

Так как для развертывания приложения было решено использовать контейнеры, на тестовый сервер необходимо было установить только Docker.

После этого мы получаем доступ к Jenkins на порту 8080 и к приложе-

нию, установленному на север, на порту 8081.

1.2 Интеграционный сервер

Сервер CI выполняет интеграционное построение всякий раз, когда в хранилище с контролем версий передаются изменения. Как правило, вы настраиваете сервер CI, чтобы проверять изменения в хранилище с контролем версий каждые несколько минут. Сервер CI получает файлы исходного кода и запускает сценарии построения. Серверы CI допускают планирование, обеспечивая построение с обычной частотой, например каждый час (однако, это уже не непрерывная интеграция). Кроме того, серверы CI обычно предоставляют удобную панель, где отображаются результаты построения [7].

Для реализации непрерывной интеграции было создано большое количество серверов сборки. В мире открытого программного обеспечения большой популярностью пользуется сервер непрерывной интеграции Jenkins.

В терминологии Jenkins сборка называется заданием (job), которое выполняется в рабочем пространстве (workspace). Это каталог в файловой системе, где сохраняются результаты сборки. В стандартной установке Jenkins позволяет создавать произвольные задания сборки, которые объединяют самые разные шаги. Дополнительные типы заданий доступны в виде плагинов. Поддерживается большое количество настроек, определяющих поведение среды и заданий. Они могут изменяться с помощью страниц веб-интерфейса Jenkins [8, 9].

Для установки Jenkins была написана отдельная Ansible роль.

Для настройки авторизации был написан Goovy скрипт, который копируется в директорию init.d. Все скрипты, находящиеся в этой папке, выполняются при загрузке Jenkins.

Также в роли `jenkins-conf` автоматизируется установка необходимых плагинов.

— GitHub плагин используется для подключения к системе контроля версий с исходным кодом приложения.
— Ansible плагин необходим для использования Ansible playbook, которая устанавливает приложение на тестовый сервер.
— Jenskins Pipeline плагин для возможности описания процесса непрерывной интеграции в виде pileline на языке Groovy.

— JobDSL плагин для возможности описания заданий с помощью языка DSL.

1.3 Разработка Docker контейнеров

Одна из самых серьезных болевых точек, которые традиционно возникают при взаимодействии команд разработчиков и поддержки, способ максимально быстрого выполнения изменений, требуемых для эффективной разработки, не рискуя при этом стабильностью производственной среды и инфраструктуры. Относительно новая технология, которая позволит в какой-то степени избавиться от этой болевой точки, заключается в использовании программных контейнеров.

Контейнеры представляют собой средства инкапсуляции приложения вместе с его зависимостями. На первый взгляд контейнеры могут показаться всего лишь упрощенной формой виртуальных машин – как и виртуальная машина, контейнер содержит изолированный экземпляр операционной системы, который можно использовать для запуска приложений [10], [11].

Но контейнеры обладают некоторыми преимуществами, обеспечивающими такие варианты использования, которые трудно или невозможно реализовать в обычных виртуальных машинах.

1. Контейнеры совместно используют ресурсы основной ОС, что делает их на порядок более эффективными. Контейнеры можно запускать и останавливать за доли секунды. Для приложений, запускаемых в контейнерах, накладные расходы минимальны или вообще отсутствуют, по сравнению с приложениями, запускаемыми непосредственно под управлением основной ОС;
2. Переносимость контейнеров обеспечивает потенциальную возможность устранения целого класса программных ошибок, вызываемых незначительными изменениями рабочей среды
3. Упрощенная сущность контейнера означает, что разработчики могут одновременно запускать десятки контейнеров, что дает возможность имитации работы промышленной распределенной системы. Инженеры по эксплуатации могут запустить на одном хосте намного больше контейнеров, чем при использовании отдельных виртуальных машин;
4. Кроме того контейнеры предоставляют преимущества конечным пользователям и разработчикам без необходимости развертывания приложения

в облаке. В свою очередь, разработчики подобных приложений могут избежать проблем, связанных с различиями в конфигурациях пользовательских сред и с доступностью зависимостей для этих приложений.

Приложение, для которого настраивался процесс непрерывной интеграции, было написано с использованием технологии Spring Java и базы данных PostgreSQL. Для удобства установки приложения на сервер было решено разбить его на два отдельных микро сервиса: базу данных и приложение. Для этих целей были разработаны Docker контейнеры.

В качестве базового использовался контейнер с операционной системой Centos. После чего был разработан контейнер с web-сервером Apache Tomcat, для которого с помощью команды FROM был указан, родительский контейнер с операционной системой Centos.

Перед сборкой образа контейнера для приложения, необходимо сначала собрать готовые образы всех родительских контейнеров и загрузить их в регистр образов.

Для контейнера базы данных был разработан отдельный Dockerfile, в котором помимо установки дистрибутива PostgreSQL, также создается и настраивается база данных. Отметим что настройка базы данных происходит не во время сборки образа, а во время запуска контейнера.

Таким образом мы смоделировали контейнеры, которые можно использовать для развертывания приложения на сервер, на котором не должно быть установлено никакого программного обеспечения кроме Docker.

1.4 Разработка Pipeline для непрерывного развертывания приложения

Jenkins Pipeline представляет собой набор плагинов, которые поддерживают реализацию процесса непрерывной доставки в виде программного кода. Pipeline непрерывной доставки описывает процесс перехода исходного кода из СУВ к установленному на производственную среду приложению.

Плагин предоставляет широкий набор инструментов для моделирования простых и сложных Pipeline с использованием синтаксиса языка DSL. Определение Jenkins Pipeline, как правило, записывается в текстовый файл (называемый Jenkinsfile), который добавляется в систему управления версиями. Таким образом, CD Pipeline становится частью приложения, версионируется и проверяется как любой другой исходный код [12].

Создание Jenkinsfile и добавление его в СУВ, предоставляет существенные преимущества:

- автоматическое создание процесса построения для всех веток СУВ;
- итерацию версий и проверку кода Pipeline (вместе с остальным исходным кодом);
- единый источник Pipeline, который доступен для просмотра и изменения всем участникам проекта.

Jenkinsfile может быть написан с использованием двух типов синтаксиса — декларативный и скриптовый. Декларативные и скриптовые Pipeline конструируются фундаментально разными элементами. Декларативный стиль, являются более новым и:

- предоставляет большой набор синтаксических конструкций для описания Pipeline;
- создан для облегчения разработки и понимания кода Pipeline.

Интеграционный Pipeline состоит из следующих основных стадий.

1. Выгрузка исходного кода из системы контроля версий.
2. Сборка исходного кода с помощью Maven в jar файл.
3. Копирование jar файла и сборка образа Docker контейнера.
4. Добавление Docker образа в регистр.
5. Установка контейнеров с приложением и базой данных на тестовый сервер.
6. Проверка результатов установки.
7. Выполнение автоматизированных тестов.

Для того чтобы сделать процесс интеграции непрерывным, воспользуемся Multibranch плагином, который позволяет настроить запуск Pipeline на каждое изменение исходного кода в системе контроля версий. Для создания Pipeline задания, воспользуемся JobDSL описанием, где зададим параметры репозитория СУВ. А также установим промежуток времени для сканирование репозитория.

Multibranch плагин автоматически создает отдельное задание для каждой ветки репозитория. Таким образом если какие-то изменения приведут к ошибкам в работе приложения, с помощью Pipeline непрерывной интеграции это проявится вскоре после добавление этих изменений в репозиторий.

Полученный образ используется при развертывании приложения. Для

развертывания приложения на сервер была написана отдельная Ansible роль – `deploy.yml`.

Теперь, когда настроен Pipeline для непрерывного развертывания приложения, на каждое изменения вносимое в систему контроля версий, будет выполнятся описанный ранее процесс.

Pipeline плагин предоставляет визуализацию процесса выполнения задания:

— индикатором успеха, является синий кружок 1;
— желтый круг, свидетельствует о нестабильности сборки 1;
— красный идентифицирует «сломанную» сборку 3.

Перейдя по названию конкретной ветки, можно увидеть Pipeline таблицу с историей всех сборок (см. рисунок 2). В данной таблице наглядно показано, какие именно изменения привели к нерабочему состоянию приложения. Для более подробного изучения неисправностей можно воспользоваться опубликованным отчетом об успешности прохождения автоматических тестов (см. рисунок 5). Таким образом, благодаря процессу непрерывной интеграции, неисправности работы приложения обнаруживается моментально после добавления изменений в систему контроля версий, что сильно упрощает процесс их исправления.

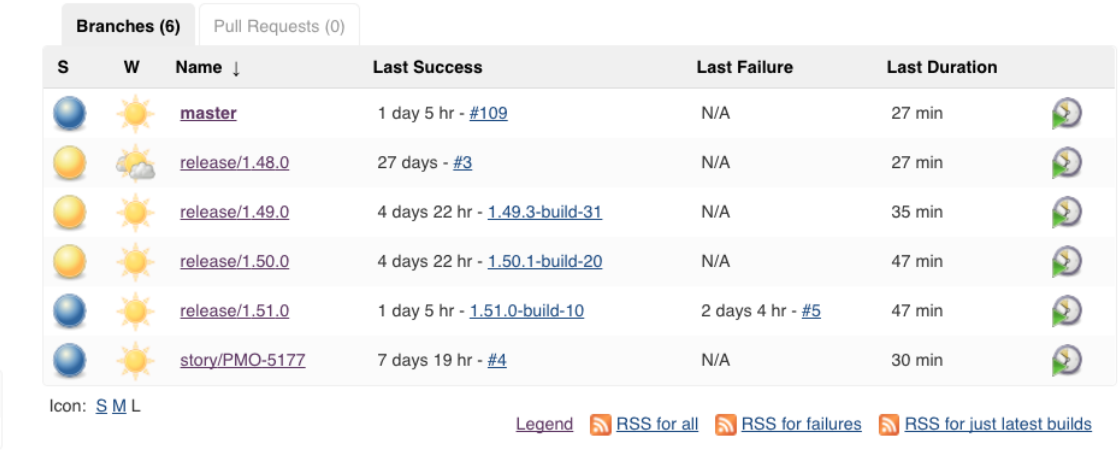

Рисунок 1 – Графический интерфейс Pipeline для непрерывного развертывания

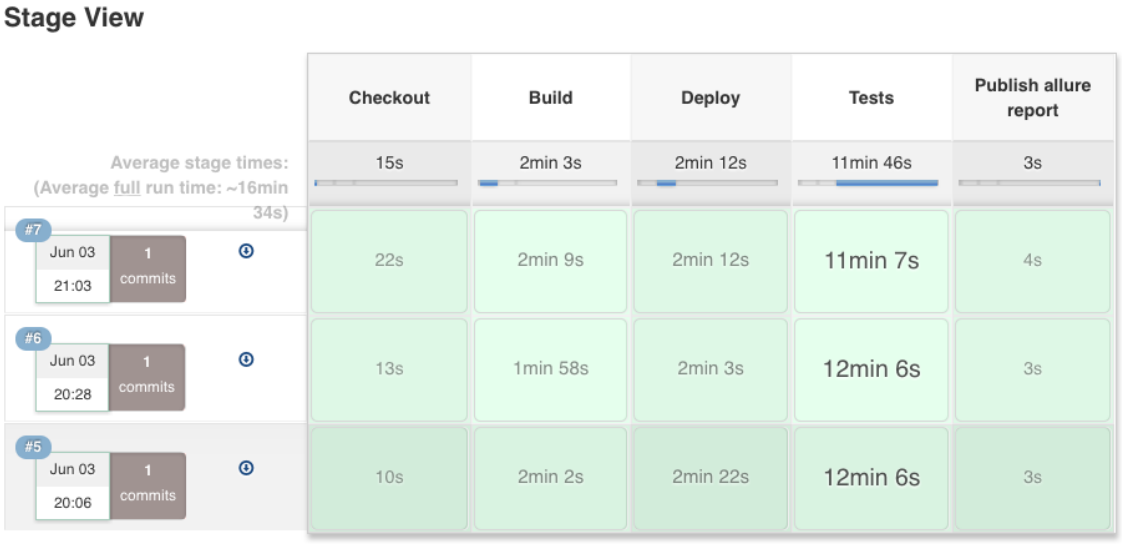

Рисунок 2 – Таблица сборок

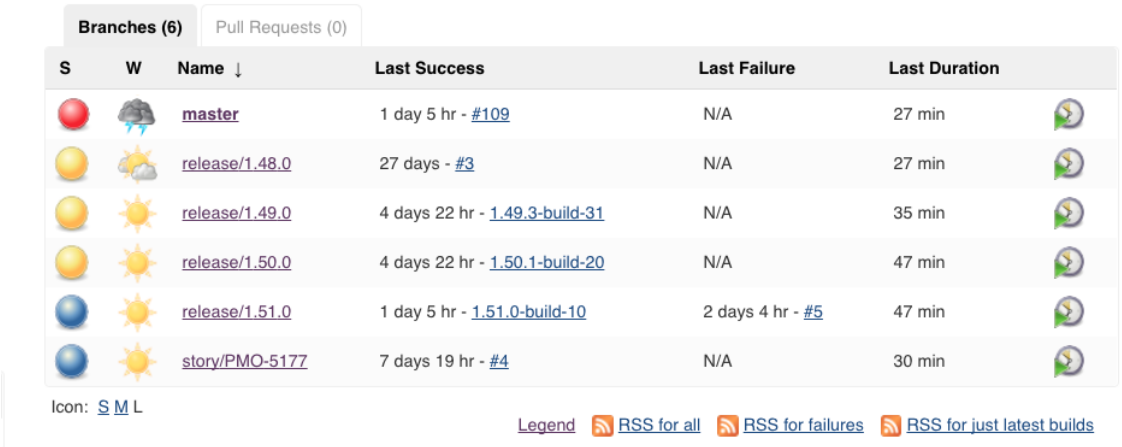

Рисунок 3 – Ветка master находится в неисправном состоянии

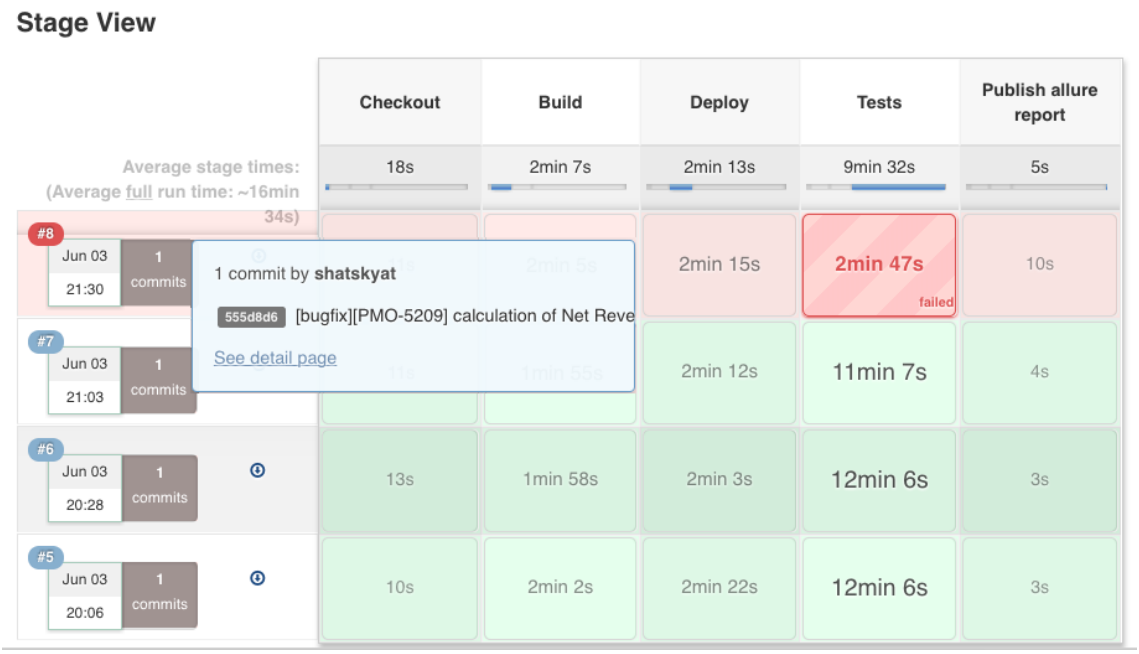

Рисунок 4 – Обнаружение того что, последние изменения привели к неисправной работе приложения

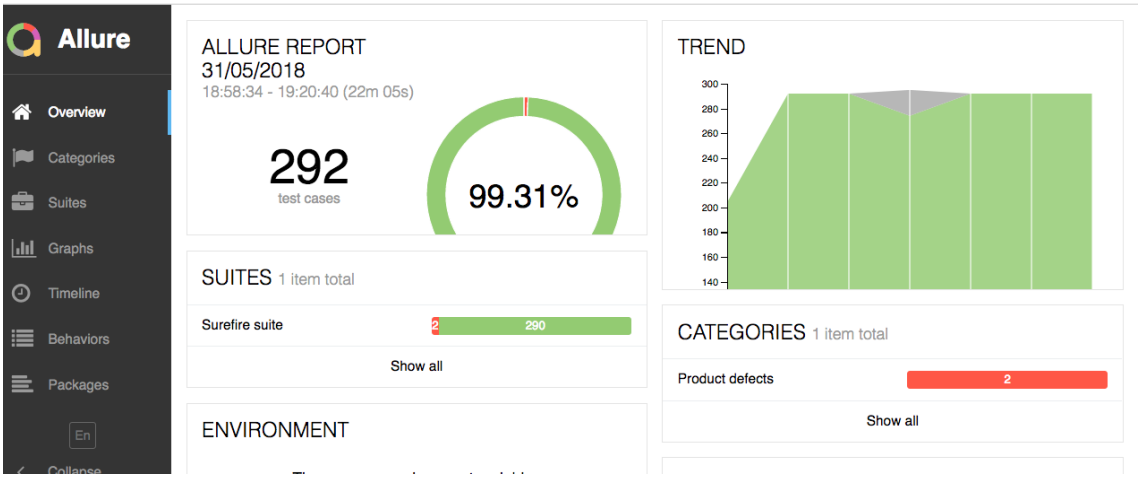

Рисунок 5 – Отчет идентифицирующий, что тесты прошли успешно

ЗАКЛЮЧЕНИЕ

В ходе данной дипломной работы, была смоделирована инфраструктура и настроен процесс для непрерывного развертывания приложения.

Данный подход к разработке программного обеспечения, как было показано в ходе работы, обладает большими преимуществами. При его использовании уменьшается время затрачиваемое на обратную связь, ошибки обнаруживается быстрее, а их причины локализуется к определенному набору изменений исходного кода. В результате чего процесс устранение неисправностей становится намного легче.

В разработанной в ходе работы модели, в каждый момент времени у нас есть контейнер с рабочим приложением, который можно установить в течении нескольких минут. В случае если очередная сборка оказалась неудачной, мы всегда можем скачать контейнер с предыдущей версии приложения из регистра образов. Поскольку контейнер инкапсулирует программное обеспечение и зависимости, необходимые для работы приложения, его можно запустить на любом компьютере, на котором установлен Docker. Таким образом можно сделать вывод от том что, использование контейнеров, упрощает и автоматизирует процесс развертывания программного обеспечения.

В ходе работы, также была смоделирована инфраструктуры, для развертывания приложения, с использование системы управления конфигурацией. При этом, если нам нужно будет развернуть подобную модель на другом компьютере или в случае если, существующая модель подвергнется повреждениям, мы всегда сможем настроит ее заново запуском одной команды. Используя систему управления конфигурациями, мы придерживаемся подхода «инфраструктуру как код», что облегчает и автоматизирует настройку рабочего окружения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Вольф, Э.* Continuous Delivery. Практика непрерывных апдейтов / Э. Вольф. — Санкт-Петербург: Питер, 2016.

2 *Дюваль, П. М.* Непрерывная интеграция. Улучшение качества программного обеспечения и снижение риска / П. М. Дюваль. — Москва: Вильемс, 2008.

3 *Humble, J.* Continuous Delivery / J. Humble. — Boston: Pearsob Education, 2011.

4 An Introduction to Configuration Management [Электронный ресурс]. — URL: https://www.digitalocean.com/community/tutorials/an-introduction-to-configuration-management (Дата обращения 10.05.2018). Загл. с экр. Яз. анг.

5 Ansible Documentation [Электронный ресурс]. — URL: http://docs.ansible.com/ansible/latest/index.html (Дата обращения 10.05.2018). Загл. с экр. Яз. анг.

6 *Hochstein, L.* Ansible: Up and Running / L. Hochstein. — CA: O'Reilly, 2012.

7 About Jenkins [Электронный ресурс]. — URL: https://www.cloudbees.com/jenkins/about (Дата обращения 10.05.2018). Загл. с экр. Яз. анг.

8 *Freguson, J.* Jenkins: The Definitive Guide / J. Freguson. — CA: O'Reilly Media, 2011.

9 *McAllister, J.* Mastering Jenkins / J. McAllister. — CA: Packt, 2015.

10 Docker Documentation [Электронный ресурс]. — URL: https://docs.docker.com/ (Дата обращения 10.05.2018). Загл. с экр. Яз. анг.

11 *Mouat, A.* Using Docker / A. Mouat. — CA: O'Reilly, 2016.

12 Jenkins Pipeline Plugin [Электронный ресурс]. — URL: https://jenkins.io/doc/book/pipeline/ (Дата обращения 10.05.2018). Загл. с экр. Яз. анг.