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 И ВНЕДРЕНИЕ РЕШЕНИЯ ПО УПРАВЛЕНИЮ КОНВЕЙЕРАМИ НЕПРЕРЫВНОЙ ИНТЕГРАЦИИ И ПОСТАВКИ С ПОМОЩЬЮ ДЕКЛАРАТИВНОГО ОПИСАНИЯ КОНФИГУРАЦИЙ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 090304 — Программная инженерия
факультета КНиИТ
Слепухина Владислава Дмитриевича

Научный руководитель
доцент, к.ф.-м.н. _____ В. Г. Самойлов
16.06.2017

Заведующий кафедрой
к.ф.-м.н. _____ С. В. Миронов
16.06.2017

Саратов 2017

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

— CI (англ. Continuous Integration) — непререрывная интеграция;
— XML (англ. eXtensible Markup Language) — расширяемый язык разметки;
— JSON (англ. JavaScript Object Notation) — текстовый формат описания объектов на основе языка JavaScript;
— DSL (англ. Domain-Specific programming Language) — предметно-ориентированный язык программирования.

ВВЕДЕНИЕ

Развитие гибких методологий разработки программного обеспечения и их повсеместное внедрение привело к появлению новой области информационных технологий, призванной привнести управление инфраструктурой и процессами, сопутствующими непосредственной разработке, а именно сборки, тестирования, развертывания, запуска, в сами команды по разработке продуктов. Эта область называется *DevOps* (Dev от англ. Development — разработка, Ops от англ. Operations — эксплуатация IT-инфраструктуры).

Данная сфера за почти десять лет своего существования популяризировала и распространила подходы непрерывной интеграции и непрерывной доставки таким образом, что настройка и поддержание данных процессов является неотъемлемой частью современной разработки программ в организациях и компаниях самого разного размера, как напрямую связанных с информационными технологиями, так и занимающихся другими сферами бизнеса.

Популяризация подходов породило проблему централизованного управления так называемыми конвейерами непрерывной интеграции и поставки в подразделениях, которые занимаются подобного рода автоматизацией для большого числа проектов компании. С проблемой централизованных DevOps-команд столкнулся один из клиентов компании Grid Dynamics.

В данной квалификационной работе описан подход, позволяющий существенно сократить как время внедрения конвейеров непрерывной интеграции и поставки на основе сервера непрерывной интеграции Jenkins CI для новых проектов компании, так и время внесения изменений для всех существующих проектов, на основе декларативного подхода “— конфигурационных файлов, формат которых был также разработан в рамках этой работы. Для достижения цели были поставлены следующие задачи:

— выделить ключевые задачи процессов непрерывной интеграции и непрерывной поставки;
— описать способ реализации этих задач в системе непрерывной интеграции Jenkins CI и выявить их ключевые параметры;
— спроектировать способы хранения и обработки параметров;
— реализовать процесс создания задач сервера непрерывной интеграции с использованием разработанного подхода к конфигурации.

В структуру бакалаврской работы входят: введение, три главы («Поста-

новка проблемы», «Реализация процесса обработки конфигураций задач» и «Пример применения обработчика конфигураций для типового конвейера») и заключение.

В первой главе описывается взаимосвязь сервера непрерывной интеграции и жизненного цикла изменения в проекте, выделяются ключевые задачи такого сервера и приводятся алгоритмы работы каждой из задач.

Во второй главе представляются результаты анализа выявленных ранее задач и их параметров. На этой основе предлагается формат хранения конфигураций и способ их обработке. Также приводятся алгоритмы внедрения разработанного подхода в процессы создания экземпляров задач на сервере непрерывной интеграции Jenkins CI и их выполнения.

В третьей главе приводится пример использования описанного подхода для типового конвейера проекта на языке Java, описывается код, необходимый как для создания задач сервера непрерывной интеграции, так и непосредственная реализация выявленных алгоритмов работы этих задач, входящих в конвейер непрерывной интеграции и поставки. В конце главы приводятся результаты внедрения данного подхода на проекте клиента компании Grid Dynamics.

1 Основное содержание работы

1.1 Постановка проблемы

В первую очередь, для введения в контекст работы, даны определения процессам непрерывной интеграции и поставки. Ввиду того, что данные процессы зачастую идут вместе особое внимаение уделено границей между ними, наглядно представленной на рисунке:

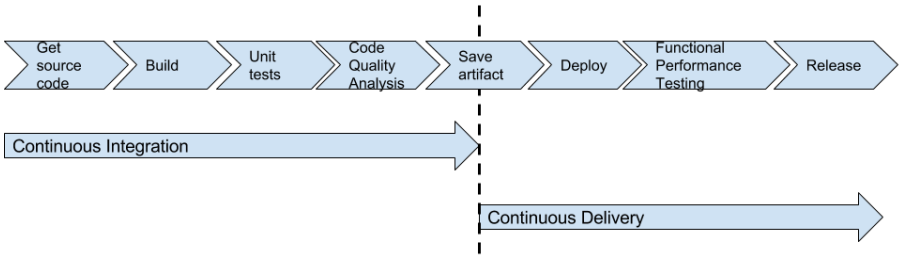

Рисунок 1 – Непрерывная интеграция и поставка

Однако основной темой данной главы является жизненный цикл изменения в каком-либо проекте, процесс его принятия в основную кодовую базу проекта и связанная с этим автоматизация. Хранение исходного кода базируется на подходе «Git Flow» [1]. Данный подход подразумевает наличие основной ветки, в которой всегда должен находиться стабильный код и некоторое число веток разрабатываемого функционала. Попадание готового функционала в основную ветку проходит через процесс открытия запросов на внесение изменений. В рамках обработки этого запроса производится два основных действия: обзор кода другими участниками проекта для проверки соответствия написанного кода требованиям, предложения лучших подходов решения поставленной задачи, а также автоматическая проверка кода и запуск тестов.

Для конкретизации всех описываемых процессов было введено понятие *типового проекта*, основывающееся на анализе подходов, применяемым к разработке продуктов клиента компании Grid Dynamics. Типовым называется такой проект, который не содержит значительных отличий в процессе сборке, тестирования и развертывания от всех остальных проектов компании. Таким образом, основным языком разработки конечных продуктов (но не решения, предлагаемого в данной квалификационной работе) стал наиболее популярный в мире [2, 3] язык Java, а системой сборки — Apache Maven. Все остальные детали являются малозначительными в рамках этой работы.

Автоматическая проверка включает в себя обычно компиляцию исходного кода, запуск модульного тестирования и статического анализа [4, с.60, 87]. Данный шаг является первым в конвейере непрерывной интеграции. Его выполняет первая рассмотренная задача сервера непрерывной интеграции под названием *Pull Request Build*.

После попадания изменения в основную ветку запускается конвейер непрерывной интеграции и поставки, описанный в [4, 5]. Ключевыми шагами конвейера являются:

— повторная сборка, возможно, с более глубокой (а соответственно долгой) оценкой кода, повторным запуском тестов и сборкой конечного артефакта и его сохранением во временный репозиторий;
— развертывания на тестовом окружении;
— запуск артефакта на тестовом окружении;
— запуск функциональных, регрессионных, нагрузочных тестов;
— сохранение в репозиторий кандидатов для релизов в случае успешного прохождения всех проверок

Для каждого из вышеописанных действий было дан пошаговый алгоритм задач на сервере Jenkins CI: *Project Build*, *Deploy*, *Maintenance*, *Post Deployment Validations*, *Promote* соответственно.

Также в данной главе обуславливается выбора сервера непрерывной интеграции Jenkins CI, который является самым популярным [6] среди аналогичных решений: по состоянию на 2016 год 30% организаций, разрабатывающих более 50 продуктов используют именно это решение. Jenkins CI предоставляет возможность создания различных задач, то есть некоторых последовательностей действий, выполняемых сервером и имеющими входные параметры, как при помощи веб-интерфейса, что может быть актуально при разработке достаточно малого числа проектов, так и при помощи программного интерфейса.

Далее, приведены технологии, используемые для автоматизации создания задач с использованием предоставляемого программного интерфейса. Наибольшее внимание уделено двум расширениям: Jenkins JobDSL, который позволяет описывать задачи при помощи расширения языка Groovy [7] и Jenkins Pipeline Plugin, который дает возможность задавать шаги задач также на языке Groovy с некоторыми дополнениями [8] для запуска класси-

ческих шагов. Последний из них привносит парадигму Pipeline as a Code [9], являющегося прямым аналогом подхода Infrastructure as a Code для организации конвейеров непрерывной интеграции и поставки.

В последнем разделе главы показывается, что несмотря на достаточно удобные и эффективные решения по описанию задач с использованием этих двух расширений, при настройке процессов непрерывной интеграции и поставки на множестве *типовых проектов* так или иначе возникают проблемы с распространением изменений во все или несколько проектов с минимальными усилиями ввиду обширной кодовой базы. Кроме этого, введение нового проекта занимает далекое от минимального время, что также является значительной проблемой.

1.2 Реализация процесса обработки конфигураций задач

Глава является центральной в рамках данной квалификационной работы и описывает разработанный подход к хранению и обработке конфигураций задач сервера Jenkins CI.

В первую очередь, был представлен способ описания задач с использованием расширения JobDSL, который заключается в создании класса задачи с описанием шагов, а также разрабатываемого подхода к хранению конфигураций. Были рассмотрены два ключевых типа задач:

— произвольные (англ. *freestyle*), которые имеют доступ ко всем классическим расширениям, задаваемые последовательностью шагов;

— задачи на основе Pipeline Plugin. Они имеют единственный шаг — вызов этого самого расширения, внутри которого при помощи языка, основанного на Groovy описаны различные действия.

Далее был описан алгоритм создания экземпляров задач на самом сервере при помощи вспомогательной задачи, называющейся *Seed*:

1. Скачать и распаковать артефакт с исходным кодом и конфигурациями задач.
2. Для каждого проекта проанализировать конфигурации запустить процесс генерации задач.
3. В случае ошибки при генерации, откатить изменения до предыдущего состояния. Обработка ошибок такого рода встроена в генератор JobDSL и выполняется автоматически.

Во втором разделе главы проведен анализ параметров выявленных в

предыдущей главе задач сервера непрерывной интеграции. Было выявлено, что все параметры можно разбить на некоторые группы: настройки сервера репозитория хранения кода, параметры сборки и тестирования, информация об облачном сервисе создания окружений и доступа к нему и так далее.

Эти группы могут быть определены на разных уровнях и переопределять свое содержимое (или добавлять новое) на более нижних. Было выделено три уровня:

1. Существует глобальная конфигурация, устанавливающая ключевые параметры для всех задач. Это конфигурация первого уровня.
2. Существуют общие конфигурации по типам задач. Это конфигурация второго уровня.
3. Последним, третьим, уровнем является конфигурация проекта, которая только необходимые параметры для сборки конкретного проекта и старается как можно больше переиспользовать конфигурации более высоких уровней.

Таким образом, базисом идеи стало наследование конфигураций, основанное на парадигме наследования из объектно-ориентированных языков программирования [10–12]

Для хранения всех конфигураций был выбран формат, предоставляемый классом обработчика GroovyConfigSlurper [13] ввиду его близости к основному языку разработки.

Конфигурационные файлы уровней 1 и 2 имеют формат набора пар ключ-значение (ассоциативных массивов). Эти конфигурации были названы *конфигурациями первого типа.* Каждое значение на верхнем уровне также является набором пар ключ-значение. Таким образом, в более глубоких конфигурациях значениями могут выступать строка, число, булевское значение или список.

Конфигурационный файл уровня 3 отличается по своему формату вследствие необходимости хранения описания самих задач для сервера непрерывной интеграции в рамках одного проекта. Это *конфигурации второго типа.* Каждый файл этого формата содержит секции, то есть пары ключ-значение, где ключом выступает название задачи, а значением другие ассоциативные массивы, описывающие параметры задачи. Например:

```
1 MyBuildJob {
2     github {
```

```
3         org = "myOrg"
4         repo = "myRepo"
5     }
6     maven {
7         goals = "compile test sonar:sonar deploy"
8         extraParams = "-Dx=y"
9     }
10 }
11 MyPromotionJob {
12     promotion {
13         groupId = "com.company.project"
14     }
15 }
```

Файл конфигурации задач имеет следующие секции:

1. Секция глобального наследования. Имеет фиксированное имя `imports`. Эта секция является списком, включающим имена файлов, которые нужно последовательно применить к конфигурациям задач в порядке следования в первую очередь.
2. Секция файл-локального наследования. Имеет фиксированное имя `common`. Данная секция будет применена ко всем задачам в рамках данного файла конфигураций.
3. Секция опционального наследования. Имя начинается с с двух нижних подчеркиваний `__`. Данные параметры будут только применены к тем задачам, которые сами выберут наследование конкретных секций.
4. Обязательные секции задач. Могут иметь произвольные имена, допускаемые сервером Jenkins CI [14].Каждая секция определяет класс JobDSL для загрузки и параметры самой задачи. Каждая секция имеет собственную подсекцию наследования, в которую можно включить внешний конфигурационный файл или секцию опционального наследования.

Таким образом, ключевым отличием файлов конфигураций первого типа от второго типа является отсутствие возможности обращаться к другим файлам и секциям. Наглядное представление правил наследования и уровней вложенности:

Далее, был разработан процесс обработки и обеспечения механизма наследования в момент загрузки конфигурационных файлов второго типа при создании экземпляров задач на сервере, заключающийся в следующих шагах:

1. Определить секции опционального наследования и исключить из явной обработки, сохранив предварительно в список.

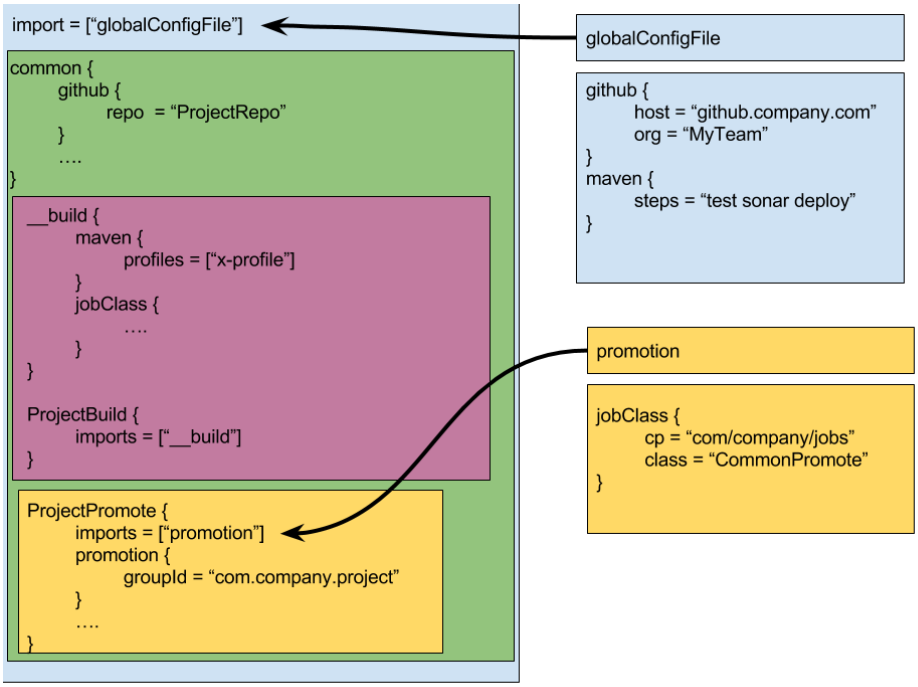

Рисунок 2 – Конфигурационный файл второго типа с основными видами наследования секций

2. Создать пустую начальную конфигурацию, общую для всех описанных задач.
3. Выяснить, существует ли секция глобального наследования и обработать ее, добавив результат к начальной конфигурации.
4. Выяснить, существует ли секция файл-локального наследования и добавить ее содержимое к начальной конфигурации.
5. Для каждой секции, представленной в файле и являющейся секцией описания задач, создать экземпляры конфигураций с данным состоянием, полученным из предыдущей обработки секций.
6. В каждый результат добавить некоторые поля, необходимые для успешной генерации задач.
7. Вернуть список обработанных конфигураций первого типа в место вызова.

Также был разработан алгоритм обработки каждой секции:

1. Создать новый экземпляр конфигурации с текущим содержанием.
2. Определить, есть ли в секции список наследований.
3. Если такой список найден, то необходимо запустить процесс обработки слева направо по всем конфигурационным файлам первого типа.
4. Если текущим элементом списка является секция опционального насле-

дования, то найти эту секцию в данном файле конфигурации и добавить ее содержимое к текущей конфигурации согласно правилу сложения ассоциативных массивов. Если указанная секция не найдена, то необходимо прекратить выполнение и вывести ошибку.

5. Если текущим элементом списка является другой файл, то необходимо открыть этот файл и сложить его содержимое аналогичным образом. Если файл не найден, следует вывести ошибку и прекратить работу.
6. Пройти по содержанию текущей секции и сложить его с существующей конфигурацией. На этом этапе обработка заканчивается.

Наглядное представление процесса обработки конфигурационных файлов:

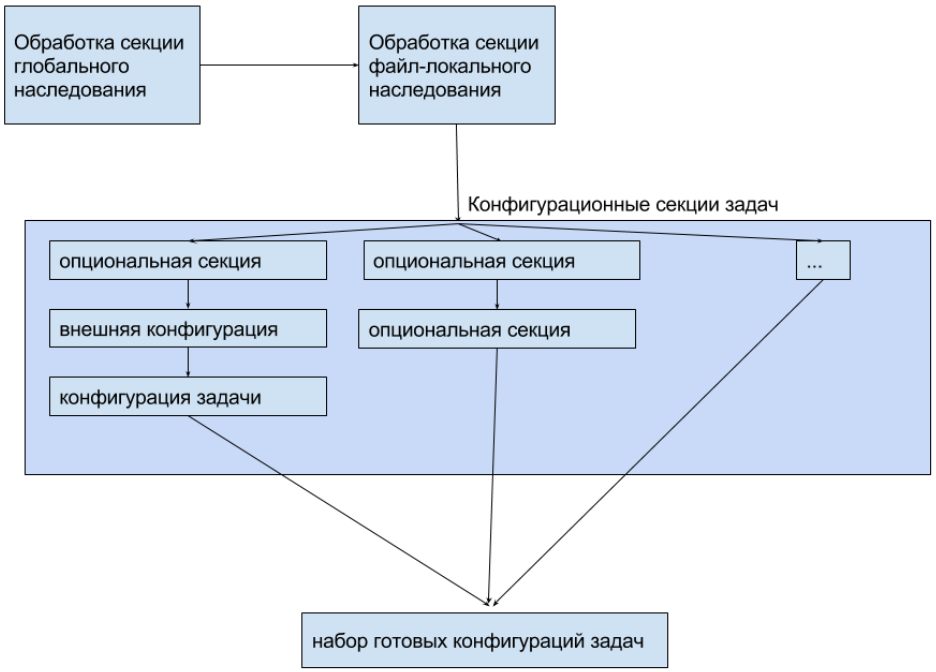

Рисунок 3 – Процесс обработки

Кроме этого в данной главе рассматривается вопрос об использовании конфигураций не только во время создания задач на сервере непрерывной интеграции, но и во время их выполнения. Особенную актуальность этот вопрос имеет в отношении задач типа Pipeline Plugin, которые получают очень мало информации о параметрах работы при создании самих экземпляров. Было предложено четыре различных способа, суть которых заключается в следующем:

1. При обработке скрипта для Pipeline Plugin заменять определенный токен строковым представлением конфигурации в формате JSON или XML.
2. Создать функцию повторной обработки конфигурационных файлов в рамках синтаксиса и шагов Pipeline Plugin.
3. Создать отдельную задачу, получающую на вход название задачи и версию кода автоматизации проектов и проводящую повторную обработку. Результат сохраняется в переменную окружения, которую затем можно считать из вызывающей задачи.
4. Сохранять закодированную конфигурацию на сетевое хранилище пар ключ-значение, например, etcd [15] или Consul [16] в момент создания задачи, а затем получать при помощи запроса к данным системам во время каждого запуска задачи.

С учетом всех проанализированных достоинств и недостатков, оцениваемых по следующим критериям: инкапсуляция данных и уровней ответственности, сложность интеграции, необходимость внешних ресурсов, время работы, был выбран и внедрен вариант 3.

1.3 Пример применения обработчика конфигураций для типового конвейера

В данной главе было дано описание задачам, входящим в конвейер непрерывной интеграции и поставки для *типового проекта*.

Для каждой задачи были описаны следующие элементы:

— конфигурационный файл первого типа, если предполагалось, что данная задача может быть переиспользована в проекте автоматизации;
— секция из конфиуграционного файла для типового проекта, реализующая данную задачу;
— алгоритм работы с учетом выполнения на сервере непрерывной интеграции Jenkins CI;
— класс для создания экземпляра задачи на основе расширения JobDSL с использованием конечной конфигурации, полученной после обработки разработанным способом в момент создания экземпляра задачи;
— описание работы конкретных шагов, вызываемых в рамках выполнения задачи;
— вспомогательные скрипты для формирования параметров задач, напри-

мер, для получения веток из репозитория исходного кода, или же для получения версий артефактов из репозитория бинарных артефактов;
— для задач, которые могли быть реализованы при помощи Pipeline Plugin, приведен скрипт для этого расширения.

Задачи *Project Build*, *Post Deployment Validations*, *Pull Request Build* реализованы как *freestyle*-задачи и не требуют скриптов для Pipeline Plugin. Для каждой из этих задач реализован один общий шаг, заключающийся в вызове системы сборки Maven с заданными параметрами. Данный шаг особым образом показывает гибкость разработанного подхода, заключающуюся в возможности настроить один и тот же шаг конвейера или задачи самым различным образом при помощи соответствующей конфиуграционной секции. Все шаги этих задач целиком и полностью были заданы при помощи JobDSL, то есть хранятся в классах задач.

Задачи *Pipeline*,*Promote*, *Deploy*, *Maintenance* имеют вид задач Pipeline Plugin. Для них необходимо были описаны скрипты, которые должны храниться в том же репозитории, что и весь код автоматизации проекта. В качестве параметров они имеют названия других задач и координаты артефактов.

Входной точкой разработанного конвейера непрерывной интеграции и поставки является задача *AutoBuild*, рассмотренная в процессе анализа типовых конвейеров и внесенная в ход работы ввиду определенных ограничений сервера непрерывной интеграции. Эта задача вызывает ключевую задачу *Pipeline*, которая в свою очередь вызывает каждую из разработанных задач, выполняющих сборку артефакта, развертывание и запуск приложения на тестовом окружении и запуск функционального тестирования против данного окружения.

В качестве типового проекта был взять Spring Pet Clinic [17], представляющий собой приложение, построенное на классической трехуровневой архитектуре и использующее систему сборки Maven.

В работе были подробно описаны конфигурации обоих типов с указанием возможных подходов наследования их при увеличении числа поддерживаемых проектов.

В последнем разделе главы также были описаны результаты внедрения разработанного подхода, подтвержденные прилагаемой к квалификационной работе справкой о внедрении. После внедрения описанного подхода на про-

екте заказчика время по внесению изменений в существующие конвейеры непрерывной интеграции и поставки, а также время добавления новых проектов существенно сократилось, что отражено на графике выполнения задач, основывающемся на приватной системе управления проектами клиента компании:

Рисунок 4 – Показатели эффективности решения

ЗАКЛЮЧЕНИЕ

В ходе данной работы была рассмотрена проблема централизации команд, внедряющих процессы непрерывной интеграции и непрерывной поставки в проекты по разработке программного обеспечения для тех или иных сфер бизнеса.

Эта проблема заключается в отсутствии эффективных методов централизованного управления автоматизацией процессов сборки, развертывания, запуска и тестирования программного обеспечения даже при использовании последних подходов к автоматизации данных действий предлагаемых наиболее популярной платформой для построения описываемых в работе процессов — Jenkins CI.

Для решения существующей задачи и достижении цели по разработке эффективного подхода к управлению конфигурациями задач сервера непрерывной интеграции Jenkins CI был применен комплексный подход, заключавшийся в первую очередь в выявлении, описании и анализе задач, входящих в жизненный цикл разрабатываемого программного обеспечения и изменений в нем, типовых проектов клиента компании Grid Dynamics, столкнувшегося с необходимостью эффективного управления множеством продуктов единственной DevOps-командой.

Результаты анализа позволили выделить ключевые параметры каждой задачи. Структура этих параметров и варианты использования были исследованы с целью выявления наиболее удобного и эффективного варианта представления.

Далее были разработаны способы хранения конфигураций и их обработки. Данные процессы были внедрены в ход работы задачи сервера Jenkins CI по созданию экземпляров других задач для унифицированной передачи требуемых параметров большому числу разрабатываемых продуктов. Кроме этого было проведено сравнение различных вариантов обеспечения доступа к конфигурациям и настройкам во время выполнения самих задач.

В качестве примера применения разработанного подхода в данной квалификационной работе было дано описание задач, скриптов и конфигураций для типового проекта разрабатываемого на языке Java с использованием системы сборки Maven.

Представленный подход был также внедрен на проекте клиента ком-

пании и позволил значительно сократить время по внесению изменений в существующие продукты, а также время внедрения процессов непрерывной интеграции и поставки для новых разрабатываемых проектов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 A successful Git branching model [Электронный ресурс]. — URL: http://nvie.com/posts/a-successful-git-branching-model/ (Дата обращения 20.05.2017). Загл. с экр. Яз. англ.

2 Java Software | Oracle [Электронный ресурс]. — URL: https://www.oracle.com/java/index.html (Дата обращения 10.05.2017). Загл. с экр. Яз. англ.

3 TIOBE Index [Электронный ресурс]. — URL: https://www.tiobe.com/tiobe-index/ (Дата обращения 10.05.2017). Загл. с экр. Яз. англ.

4 *Duvall, P. M.* Continuous Integration: Improving Software Quality and Reducing Risk / P. M. Duvall, A. Glover, S. Matyas. — Addison-Wesley Professional, 2007.

5 *Humble, J.* Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation / J. Humble, D. Farley. — Addison-Wesley Professional, 2011.

6 The State of Jenkins - 2016 Community Survey [Электронный ресурс]. — URL: https://jenkins.io/blog/2017/03/24/jenkins-community-survey/ (Дата обращения 22.05.2017). Загл. с экр. Яз. англ.

7 Job DSL Plugin [Электронный ресурс]. — URL: https://wiki.jenkins-ci.org/display/JENKINS/Job+DSL+Plugin (Дата обращения 11.05.2017). Загл. с экр. Яз. англ.

8 Pipeline Syntax [Электронный ресурс]. — URL: https://jenkins.io/doc/book/pipeline/syntax/ (Дата обращения 11.05.2017). Загл. с экр. Яз. англ.

9 Pipelines as code [Электронный ресурс]. — URL: https://www.thoughtworks.com/radar/techniques/pipelines-as-code (Дата обращения 11.05.2017). Загл. с экр. Яз. англ.

10 *Tempero, E.* What programmers do with inheritance in java // Lecture Notes in Computer Science. — Vol. 7920. — Springer, Berlin, Heidelberg: 2013. — Pp. 577–601.

11 *Verona, J.* Practical DevOps / J. Verona. — Packt Publishing, 2016.

12 *Morris, K.* Infrastructure as Code: Managing Servers in the Cloud / K. Morris. — O'Reilly Media, 2016.

13 GroovyConfigSlurper [Электронный ресурс]. — URL: `http://docs.groovy-lang.org/2.4.7/html/gapi/groovy/util/ConfigSlurper.html` (Дата обращения 22.05.2017). Загл. с экр. Яз. англ.

14 What characters are illegal to include in job names? [Электронный ресурс]. — URL: `https://groups.google.com/forum/#!msg/jenkinsci-users/me2WC-AqvgY/cRur-eKYIeQJ` (Дата обращения 22.05.2017). Загл. с экр. Яз. англ.

15 etcd 3.1.8 Documentation [Электронный ресурс]. — URL: `https://coreos.com/etcd/docs/latest/` (Дата обращения 01.06.2017). Загл. с экр. Яз. англ.

16 Consul by Hashicorp [Электронный ресурс]. — URL: `https://www.consul.io/` (Дата обращения 01.06.2017). Загл. с экр. Яз. англ.

17 A sample Spring-based application [Электронный ресурс]. — URL: `https://github.com/spring-projects/spring-petclinic` (Дата обращения 25.05.2017). Загл. с экр. Яз. англ.

ВФ / (Плухин)
16.06.2017