

Университет ИТМО

**Облачные системы и услуги**

**Лабораторная работа №4**

Выполнили: Данилов Владислав Андреевич K34201**,**

Ледванов Вадим Дмитриевич K34201,

Смирнов Кирилл Андреевич K34201

**Преподаватель:** Мигулаева Татьяна Алексеевна

Санкт-Петербург

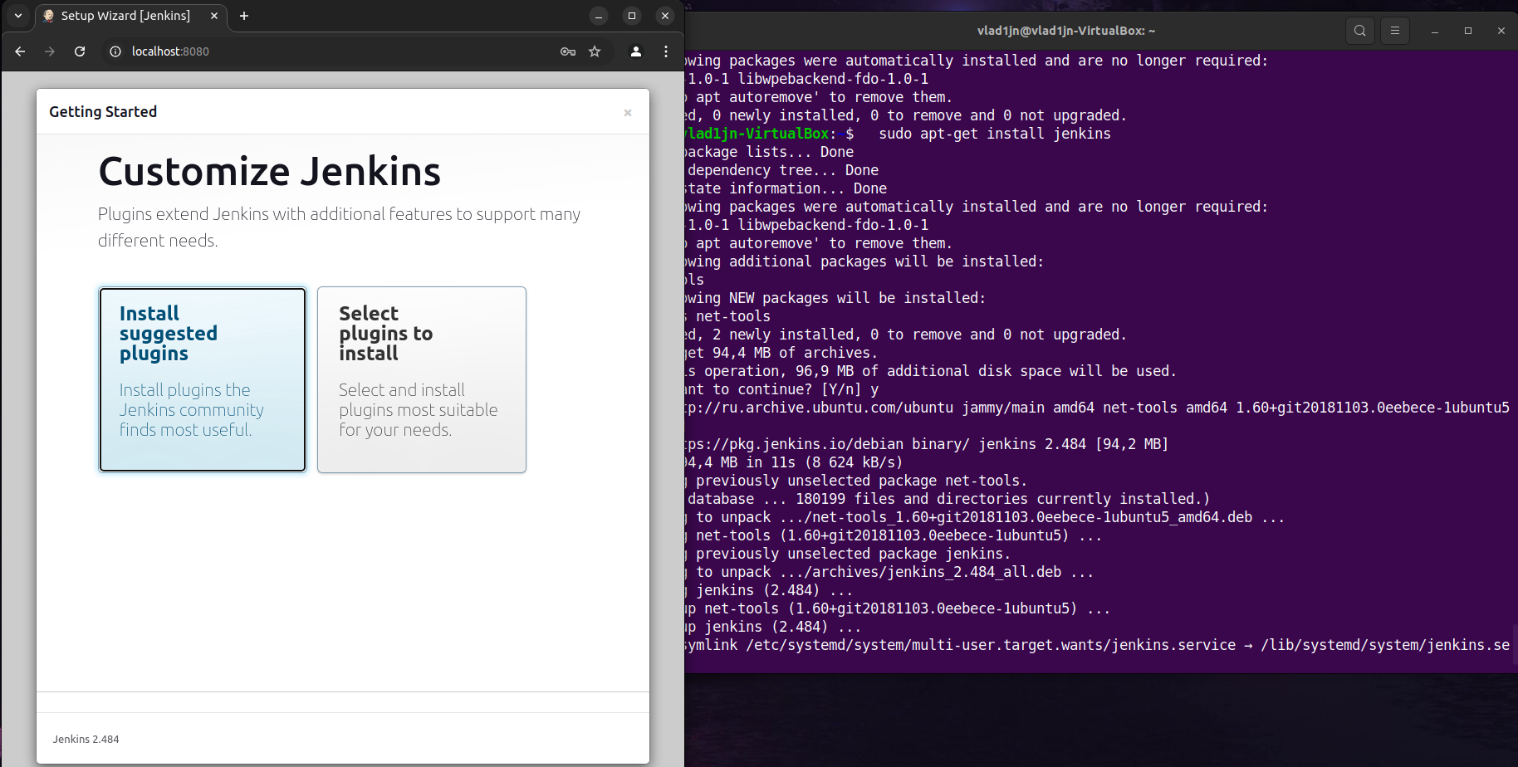
2024

**Цель работы:**

Изучить принципы написания CI/CD файлов и провести сравнительный анализ некорректных и оптимальных практик его разработки. Проанализировать две неэффективные практики, выделить их недостатки и на основе этого сформулировать выводы, способствующие повышению эффективности работы с непрерывной интеграцией.

**Ход работы:**

**Задание 1**.

В качестве CI/CD системы был выбран Jenkins. Устанавливаем и логинимся в нем под администратором:

Для выполнения задания лабораторной напишем пайплайн, который выполняет вычисление квадратного корня из числа, которое вводится пользователем в качестве параметра. Он вычисляет корень, проверяет результат и выводит информацию об успешности или ошибке.

Пайплан проектировался так, чтобы:

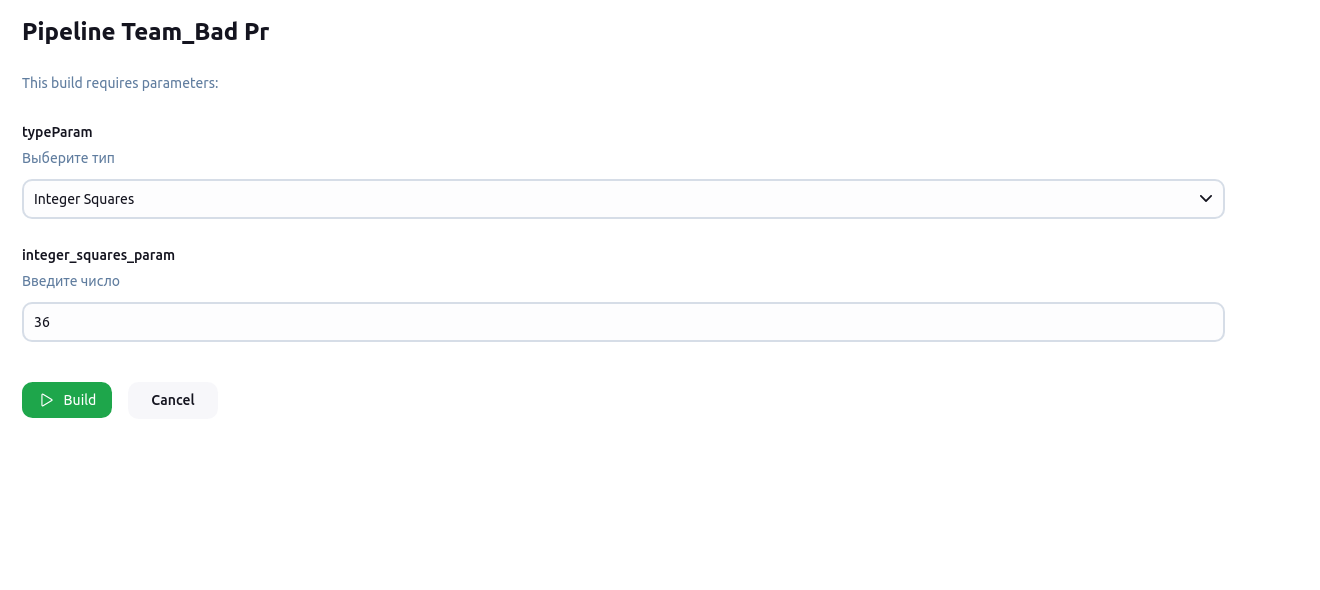
1. Пользователь указывал тип значения (typeParam — может быть либо "целым квадратом" или "нецелым квадратом") и вводит число (integer\_squares\_param), для которого нужно рассчитать корень:

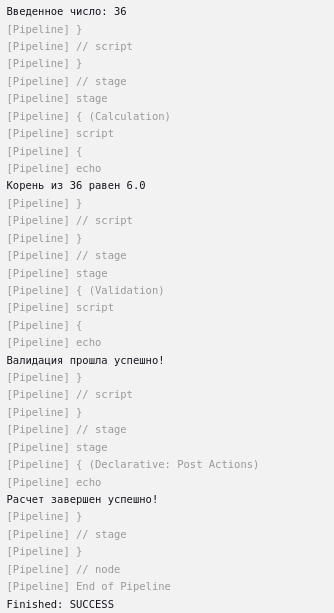


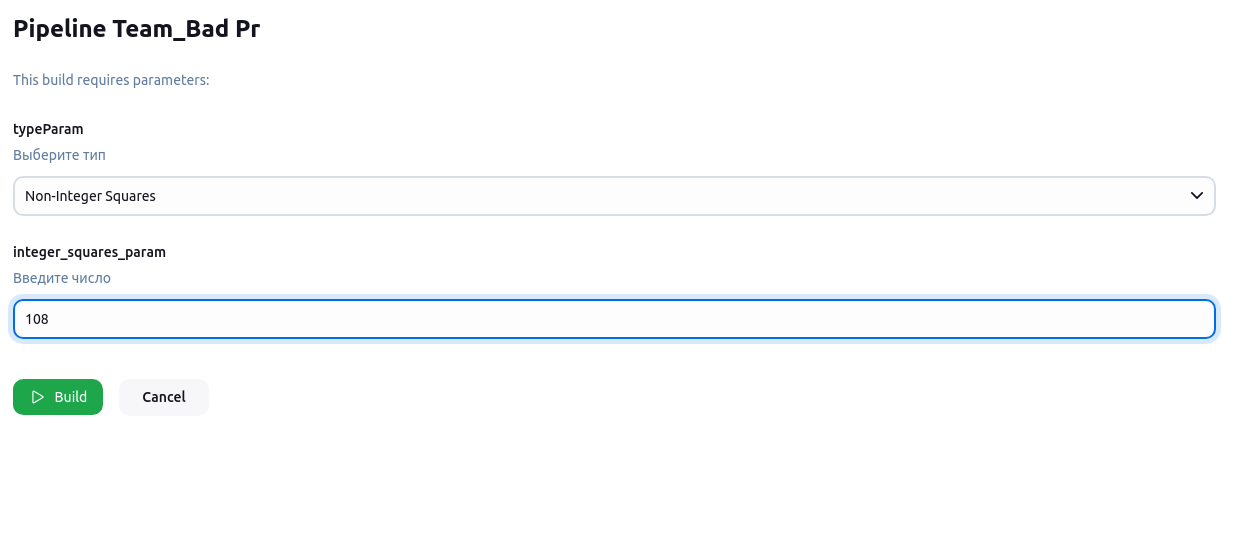
1. Производился расчет. Если все этапы прошли успешно, выводится сообщение об успешном расчете. В случае ошибки, например при некорректном вводе или неверной валидации, выводится сообщение о неудаче.

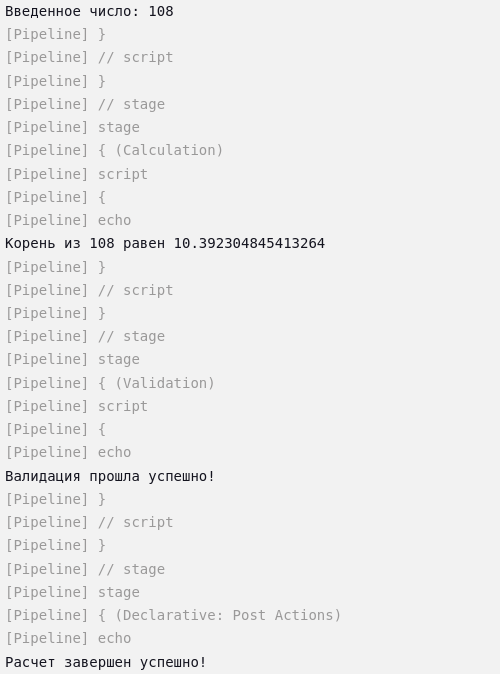
Полный код “bad practices” Пайплайна, написанный на Groovy, представлен ниже:

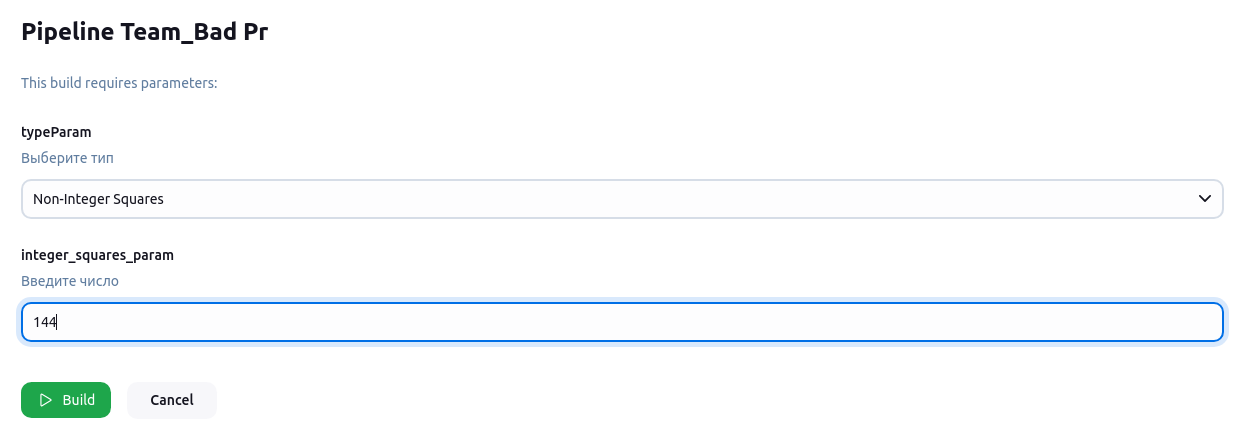
Он запускается и выполняет валидацию следующим образом:

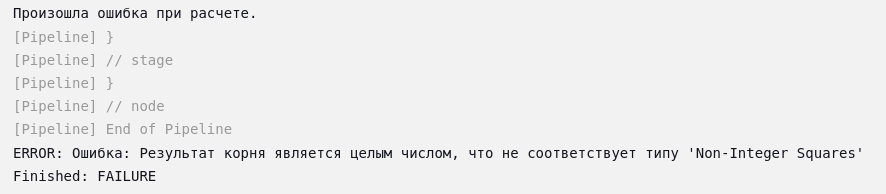
1)



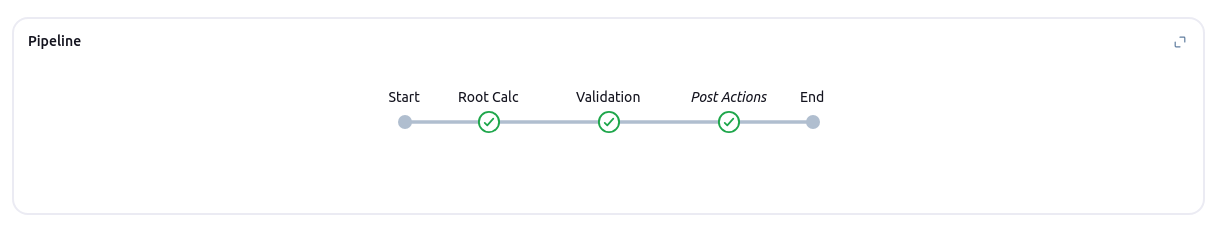
2)



3)



Схематичный вид пайплайна:



**Задание 2**.

Создание данного плайплана было выполнено аналогично.

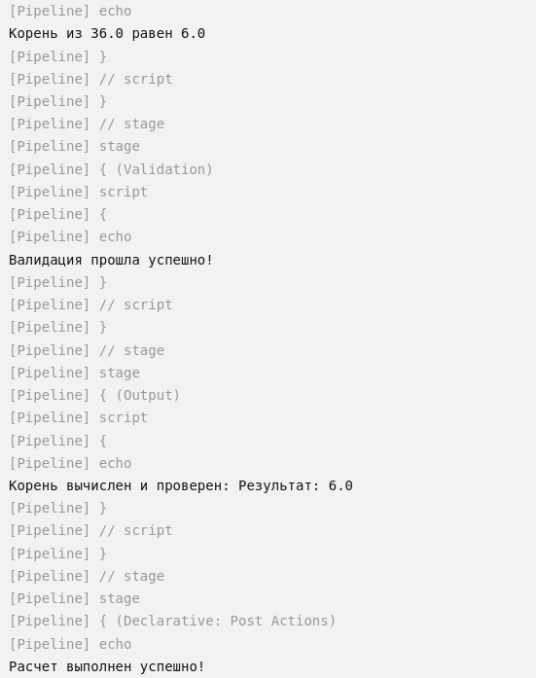
Groovy-код, представленный выше был отлажен и переписан.

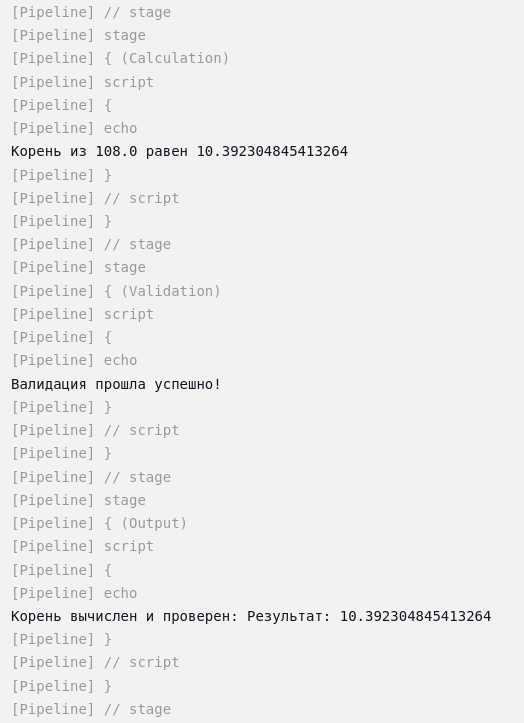
Полный код “good practices” Пайплайна, написанный на Groovy, представлен ниже:



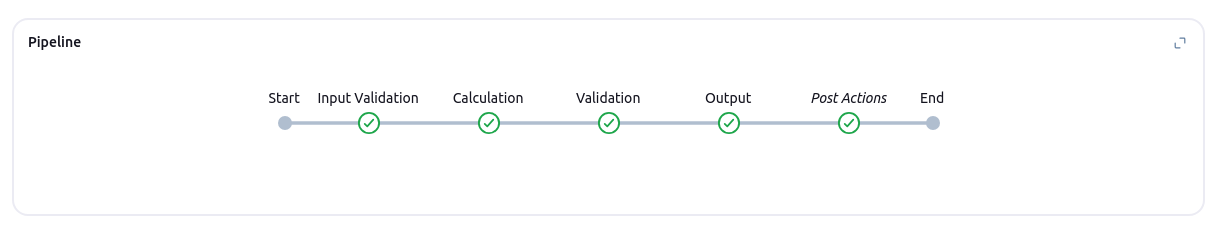
Интерфейс запуска этого пайплайна схожий с предыдущим.

Он запускается и выполняет валидацию следующим образом:

1)

2)

3)

Схематичный вид пайплайна:

**Задание 3**.

(Копия из Readme)

В "плохом" пайплайне:

1. В этом примере не проверяется, является ли число “integer\_squares\_param” целым числом. Если пользователь введёт не число, то может возникнуть внутренняя ошибка Groovy при преобразовании.
2. В этапе ‘Calculation’ производится как вычисление, так и обновление описания сборки:

‘currentBuild.description’ это

Это нарушает принцип разделения ответственности и может усложнять отладку и сопровождение кода.

1. ‘Validation’ считывает результат из currentBuild.description, разделяя логику между этапами. Происходит прямая проверка результата из описания сборки. Это снижает читаемость и может вызвать ошибки, если формат описания изменится.
2. Валидация результата вычисления происходит на Stage 'Calculation' с условным выводом сообщения об ошибке. Это делает код менее гибким: ошибки выводятся непосредственно в момент проверки, а не централизованно.
3. Отсутствие отдельного метода для вычисления корня  
   В плохом примере логика расчета корня выполняется непосредственно в этапе Calculation, что нарушает принцип повторного использования кода. Если потребуется изменить общую логику пайплайна, придётся переписывать код в нескольких местах.

Как она (“bad practices”) была исправлена:

1. В хорошем примере добавлен stage ‘Input Validation’, где проверяется, является ли ‘integer\_squares\_param’ числом. Это предотвращает потенциальные ошибки и повышает надёжность пайплайна.
2. Этапы пайплайна чётко разделены: этап ‘Calculation’ отвечает только за расчёт, а ‘Validation’ за проверку.

Это улучшает читаемость и поддерживаемость кода.

1. Результат вычисления сохраняется в переменной и проверяется в ‘Validation’, далее выводится в Лог. Это делает пайплайн более гибким, позволяя модифицировать описание сборки, не нарушая логику работы.
2. Валидация вынесена в отдельный этап с централизованной логикой
3. Используется метод calculateSquareRoot для расчета значения корня.

Это позволяет использовать код повторно и облегчает его изменение.

**Вывод:**

В работе были изучены принципы написания CI/CD файлов на примере Jenkins и Groovy, а также выполнен анализ “good practices” и “bad practices”. Это позволило сделать вывод, что чёткое разделение этапов, ранняя валидация данных и структурированный код делают CI/CD файлы более надёжными и удобными в поддержке. Для повышения эффективности работы с Jenkins важно использовать Groovy-скрипты, с ним можно создавать удобные и понятные конвейеры. При этом нужно разделять функциональность между стейджами и уделять больше внимания отладке.