

CI/CD

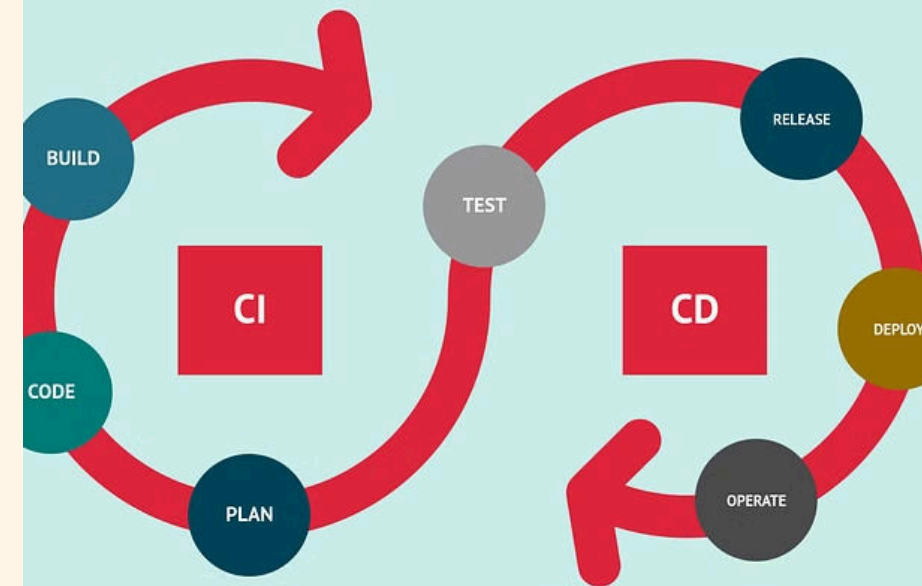
Непрерывная интеграция (CI) и непрерывная доставка (CD) - это набор практик, которые помогают разработчикам плавно и надежно выпускать программное обеспечение. Они улучшают скорость и качество процесса разработки, а также снижают риски, связанные с развертыванием новых версий.



by **Илья Исаев**

egration (CI) /Continuous Deployment

Continuous Deployment (CI/CD) introduces automation into the stages of development and deployment. CI/CD introduces continuous automation and integration, from testing to delivery and then deployment.



CODE ANALYSIS



GITHUB

RE

Что такое непрерывная интеграция (CI)?

Непрерывная интеграция (CI) - это практика разработки программного обеспечения, при которой разработчики регулярно объединяют свои изменения в единое хранилище кода. Это позволяет им автоматически тестировать и выявлять проблемы на ранней стадии, улучшая качество и скорость выпуска продукта.

CI требует от команд разработчиков объединять изменения несколько раз в день, а также использовать автоматизированные сборки и тестирование для проверки каждого набора изменений на ошибки и несовместимость. Это помогает быстро находить и исправлять проблемы, не допуская их накопления.

Зачем нужен CI

- Быстрота
- Лень
- Время

QA

- Linters (flake8, PMD)
- Code Complexity (mccabe, radon)
- Unit Testing (jUnit, pytest)
- Code Coverage
- End-To-End (Robot Framework), Integration Testing
- Load Testing (Yandex Tank, Apache JMeter), в целом Performance Testing
- Testing Result (TestRail)

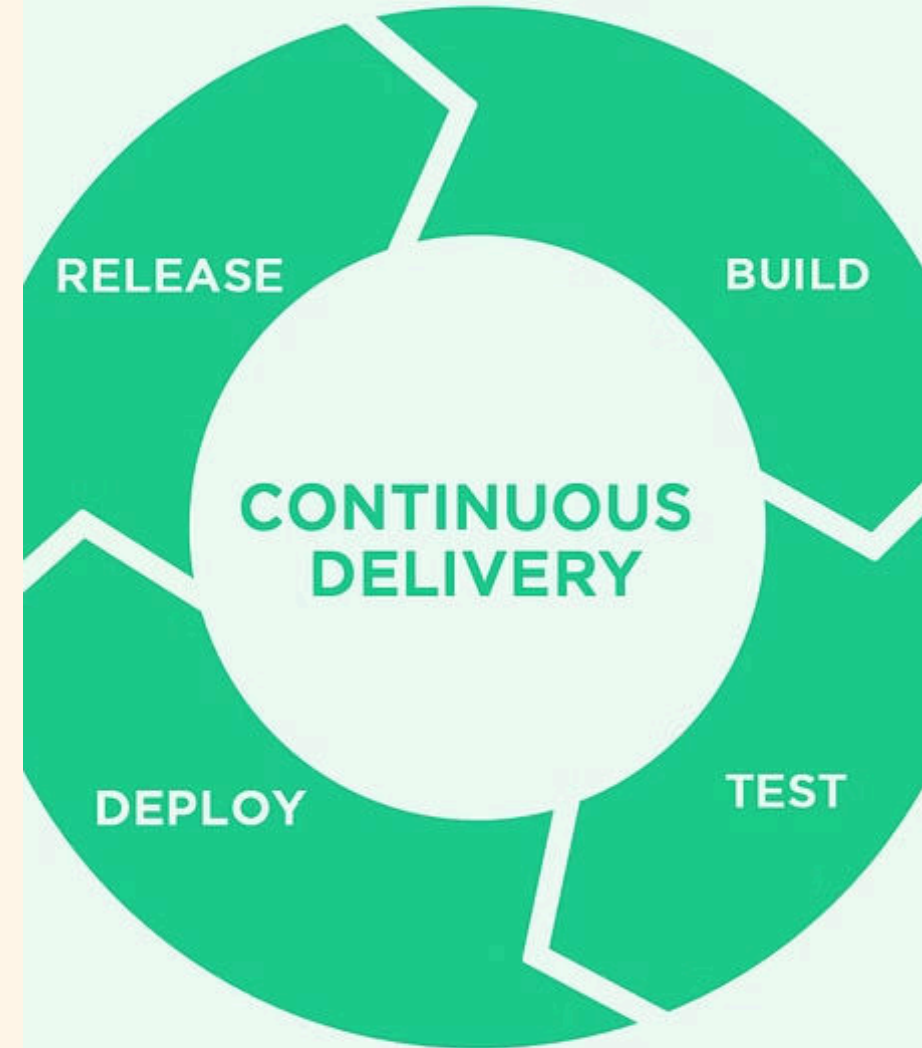
SAST (Static Application Security Testing)

- Bandit
- SonarQube
- JFrog Xray
- Snyk
- Checkmarx
- CodeScoring

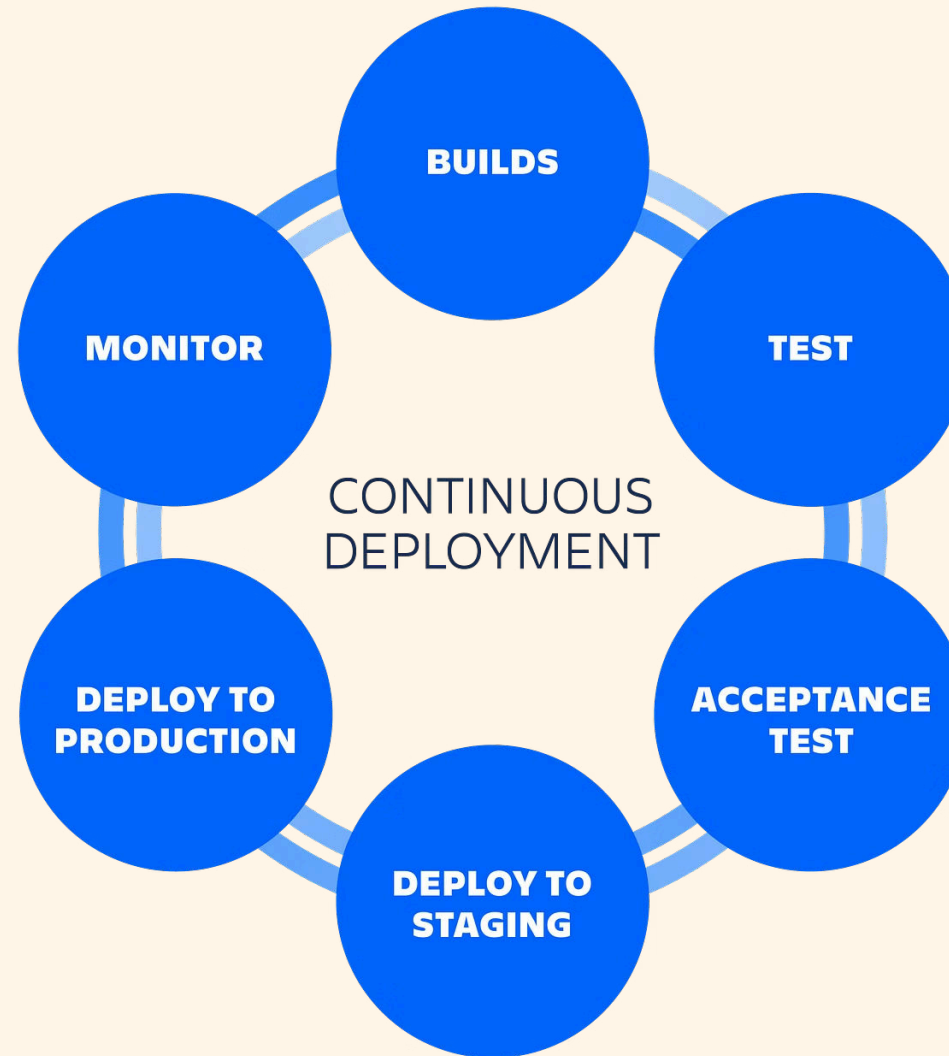
Artifacts

- Nexus
- Artifactory

Continuous Delivery



Continuous Deployment



Системы CI/CD

- TeamCity
- GitLab CI/CD
- Jenkins
- CircleCI
- GoCD
- Travis CI
- GitHub Actions

Концепции

- master node → slave node
- Jenkins Run On Master (выключить запуск пайплайнов на master)

Концепции

- TeamCity (Build → Step)
- GitLab (Pipeline → Job)
- Jenkins (Job → Stage)

Secrets

- не хранятся в Git, а в CI
- Privileged Access Management (PAM) (hashicorp vault)

TeamCity

- Java (проприетарная лицензия)
- Free tier on-premise (100 конфигураций и 3 агента)
- Downstream job (запуск себя после сборки проекта)
- Шаблоны из своего билда
- Метараннеры

TeamCity

- Build
- Triggers
- Метараннер
- Build chains (цепочки сборок)
- Kotlin generator

Blog Helm

Commit Stage <All branches>

Overview History Change Log Issue Log Statistics Compatible Agents 1 Pending Changes Build Chains Settings

Build chains: <All build chains>

Showing 20 most recent build chains.

▼	... Commit Stage	master	#1.3.18	✓ Tests passed: 1 ▼	
<input checked="" type="checkbox"/>	Show details	<input type="checkbox"/>	Group composite builds	<input type="checkbox"/>	Group by projects
...	... Commit Stage ▼	... Deploy To Test ▼	... Deploy To Acceptance ▼	... Deploy To Production ▼	
	master	master	master	master	
	#1.3.18	#1.3.18	#1.3.18	#1.3.18	
	✓ Tests passed: 1 ▼	✓ Success ▼	✓ Success ▼	✓ Success ▼	
	Changes (1) ▼	Changes (2) ▼	Changes (32) ▼	Changes (32) ▼	
	Artifacts ▼				

□ Compile

General Settings

Version Control Settings

Build Step: Command Line

Triggers

Failure Conditions

Build Features

Dependencies

Parameters

Agent Requirements

Suggestions

« Hide unconfigured

Settings are stored in VCS (view history)

Edit in UI



Help

Feedback

Build configuration settings are stored in Kotlin DSL, consider changing the settings in the Kotlin scripts instead of user interface

Copy to clipboard

Version 2018.2 ▾ ☒ Portable

```
package _Self.buildTypes
```

```
import jetbrains.buildServer.configs.kotlin.v2018_2.*
```

```
object Compile : BuildType({  
    name = "Compile"
```

```
    artifactRules = "application.jar"
```

```
    steps {
```

```
        script {
```

```
            scriptContent = "./generate.sh"
```

```
        }
```

```
        maven {
```

```
            goals = "clean package"
```

```
            mavenVersion = defaultProvidedVersion()
```

```
        }
```

```
    }
```

```
})
```

New build step



TeamCity Professional 2018.2.2 (build 61245)

[License agreement](#)

Copyright © 2006–2019 JetBrains s.r.o.

Build MacOS

```
version = "2018.2"

project {
    buildType(BuildForMacOSX)
}

object BuildForMacOSX : BuildType({
    name = "Build for Mac OS X"

    vcs {
        root(DslContext.settingsRoot)
    }

    steps {
        maven {
            goals = "clean package"
            mavenVersion = defaultProvidedVersion()
            jdkHome = "%env.JDK_18%"
        }
    }

    requirements {
        equals("teamcity.agent.jvm.os.name", "Mac OS X")
    }
})
```

TeamCity

Пример настройки TeamCity может включать следующие шаги:

1. Настройка проекта и его параметров.
2. Настройка системы управления версиями (например, Git) и настройка интеграции с TeamCity.
3. Настройка сборочного агента для выполнения сборки проекта.
4. Создание и настройка сборочного пайплайна, включая шаги сборки, тестирования и развертывания.
5. Настройка уведомлений и отчетов о результатах сборки и тестирования.
6. Запуск и отслеживание процесса сборки и доставки программного обеспечения.

GitLab

- Go
- Integrated in GitLab EE/CE
- Free tier SaaS
- As a Code
- Параллельность исполнения
- Job per node
- Шаблоны

GitLab Continuous Deployment

```
image: java:8
```

```
stages:
```

- build
- deploy

```
build:
```

```
  stage: build
```

```
  script: ./mvnw package
```

```
  artifacts:
```

```
    paths:
```

- target/demo-0.0.1-SNAPSHOT.jar

```
production:
```

```
  stage: deploy
```

```
  script:
```

- curl --location "https://cli.run.pivotal.io/stable?release=linux64-binary&source=github" |

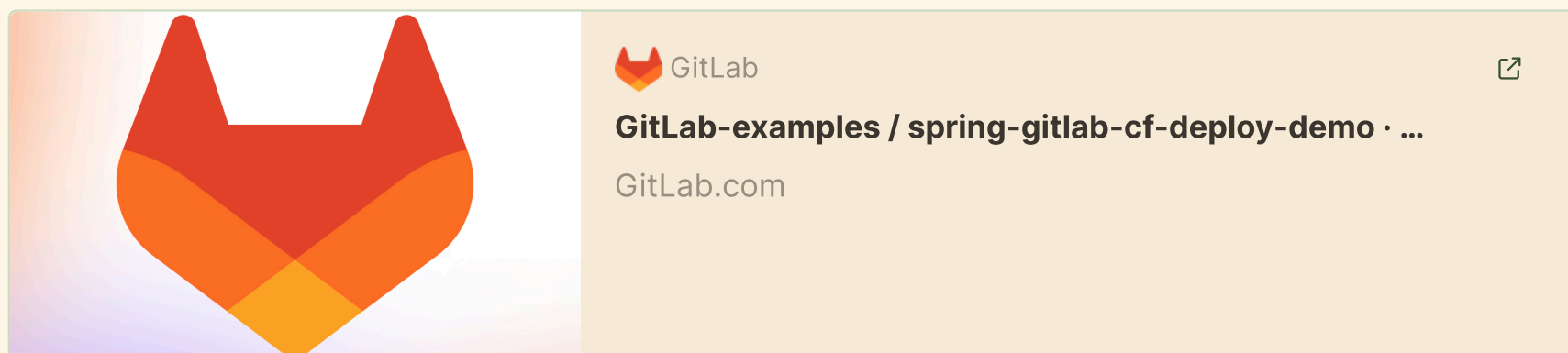
```
tar zx
```

- ./cf login -u \$CF_USERNAME -p \$CF_PASSWORD -a api.run.pivotal.io

- ./cf push

```
only:
```

- master





Jenkins


- Java (MIT)
- Open Source Automation Server
- Groovy DSL
- Параллельность выполнения

Jenkins

- Freestyle Job (параметры в интерфейсе)
- Pipeline Job (GitOps)
 - Declarative
 - Scripted
- Jenkins Shared-libraries



 Pipeline Syntax



Pipeline Syntax

Jenkins – an open source automation server which enables developers around the world to reliably build, test, and depl...

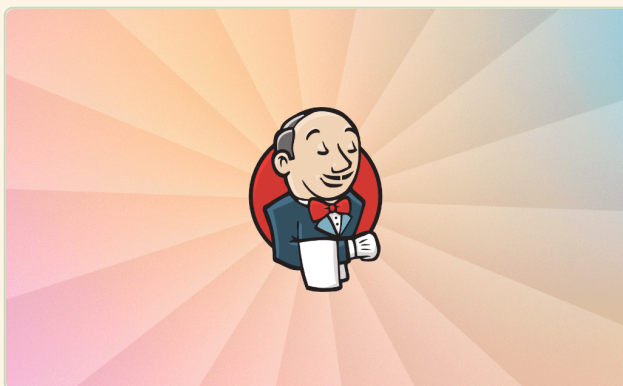
Jenkins Pipelines

```
pipeline {
  agent any
  stages {
    /* "Build" and "Test" stages omitted */

    stage('Deploy - Staging') {
      steps {
        sh './deploy staging'
        sh './run-smoke-tests'
      }
    }

    stage('Sanity check') {
      steps {
        input "Does the staging environment look ok?"
      }
    }

    stage('Deploy - Production') {
      steps {
        sh './deploy production'
      }
    }
  }
}
```



Deployment

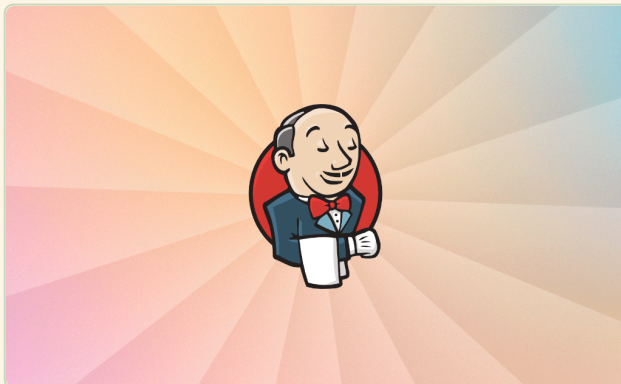


Deployment

Jenkins – an open source automation server which enables developers around the world to reliably build, test, and depl...

Job DSL

```
pipelineJob('job-dsl-plugin') {  
  definition {  
    cpsScm {  
      scm {  
        git {  
          remote {  
            url('https://github.com/jenkinsci/job-dsl-plugin.git')  
          }  
          branch('*/master')  
        }  
      }  
    }  
    lightweight()  
  }  
}
```



Job DSL



Job DSL

This plugin allows Jobs and Views to be defined via DSLs

CASC Configuration as Code

```
jenkins:
  systemMessage: "Jenkins configured automatically by Jenkins Configuration as Code
  plugin\n\n"
  numExecutors: 5
  scmCheckoutRetryCount: 2
  mode: NORMAL

globalNodeProperties:
  - envVars:
    env:
      - key: FOO
        value: BAR

authorizationStrategy:
  loggedInUsersCanDoAnything:
    allowAnonymousRead: false
```



configuration-as-code-plugin/demos/jenkins/jenki...

Jenkins Configuration as Code Plugin. Contribute to
jenkinsci/configuration-as-code-plugin development by...



Преимущества CI/CD

1. Ускорение времени вывода продукта на рынок (**time-to-market**) за счет автоматизации процессов сборки, тестирования и развертывания.
2. Повышение качества программного обеспечения благодаря раннему обнаружению и исправлению ошибок и уязвимостей.
3. Улучшение взаимодействия между разработчиками, тестировщиками и операционными командами через эффективную совместную работу.
4. Сокращение рисков, связанных с развертыванием новых версий, за счет постепенного и контролируемого внедрения изменений.
5. Возможность быстрого реагирования на изменения требований и рыночные тенденции.

GitHub

- Ruby on Rails, Erlang
- Удобно для проектов с открытым кодом
- Простая настройка

Пример

```
name: GitHub Actions
run-name: ${ github.actor } is testing out GitHub Actions
on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
jobs:
  init:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Starting Java
        uses: actions/setup-java@v4
        with:
          java-version: '17'
          distribution: 'adopt'
      - name: Build with Gradle
        run: ./gradlew build
```

Другое

- makefile
- docker caches
- CI/CD - не всегда непрерывен