

# Jenkins devqa





# Arek Gutkowski

[github.com/ArkadioG](https://github.com/ArkadioG)

# AGENDA

- Continuous Integration / Continuous Delivery
- DevOps
- Jenkins
  - instalacja
  - konfiguracja
  - założenie projektu (job)
  - projekty zależne
- Dobre Praktyki

# 1. CI – CD

# Continuous Integration

**CI** – to praktyka mergowania wszystkich gałęzi developerskich do głównej gałęzi kodu co najmniej raz dziennie.

Praktyka XP (eXtreme Programming) postuluje integrowanie kodu nawet kilkanaście razy dziennie

# Continuous Delivery

**CD** - Podejście do wytwarzania oprogramowania, w którym zespół developerski produkuje kod w krótkich cyklach, zapewniając **gotowość** kodu do wdrożenia w każdym czasie.

Osiaga się to poprzez automatyzację budowania, testowania i release'owania kodu.

## 2. DevOps

# DevOps

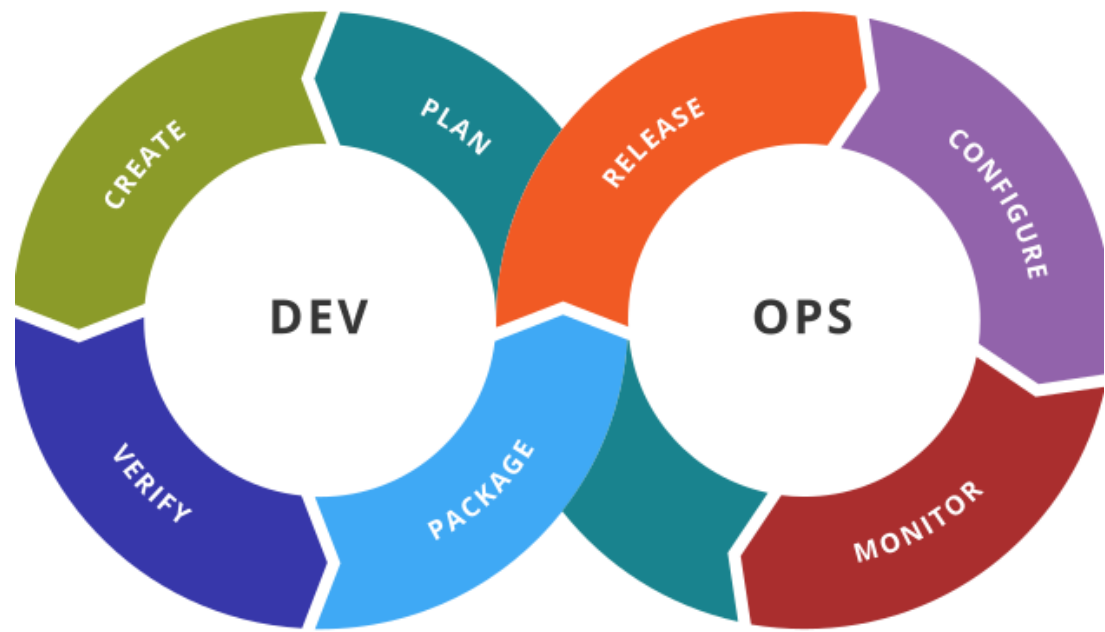
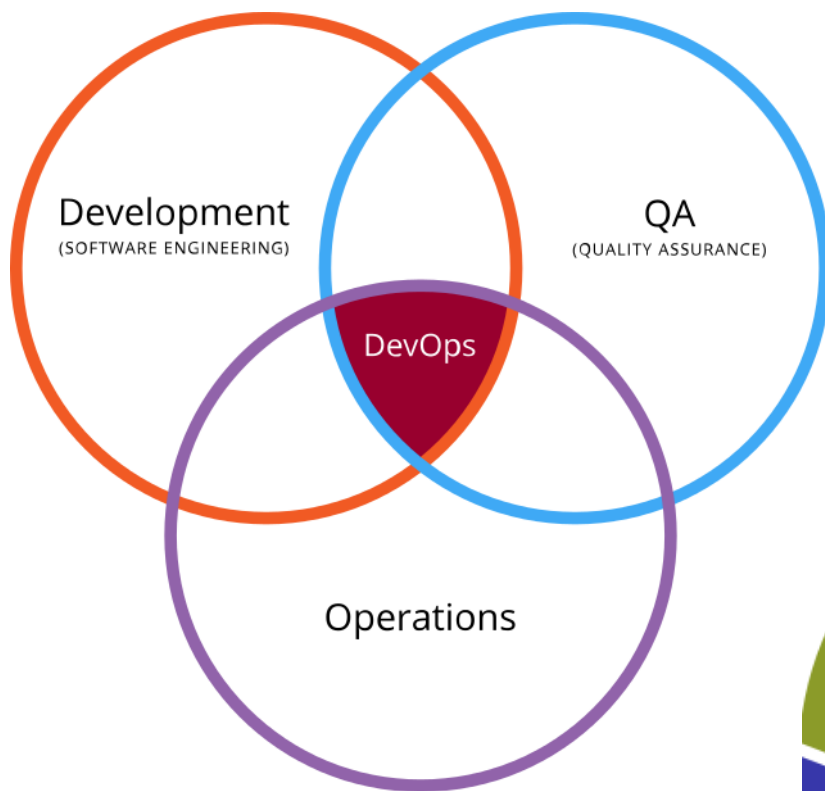
## Development and Operations

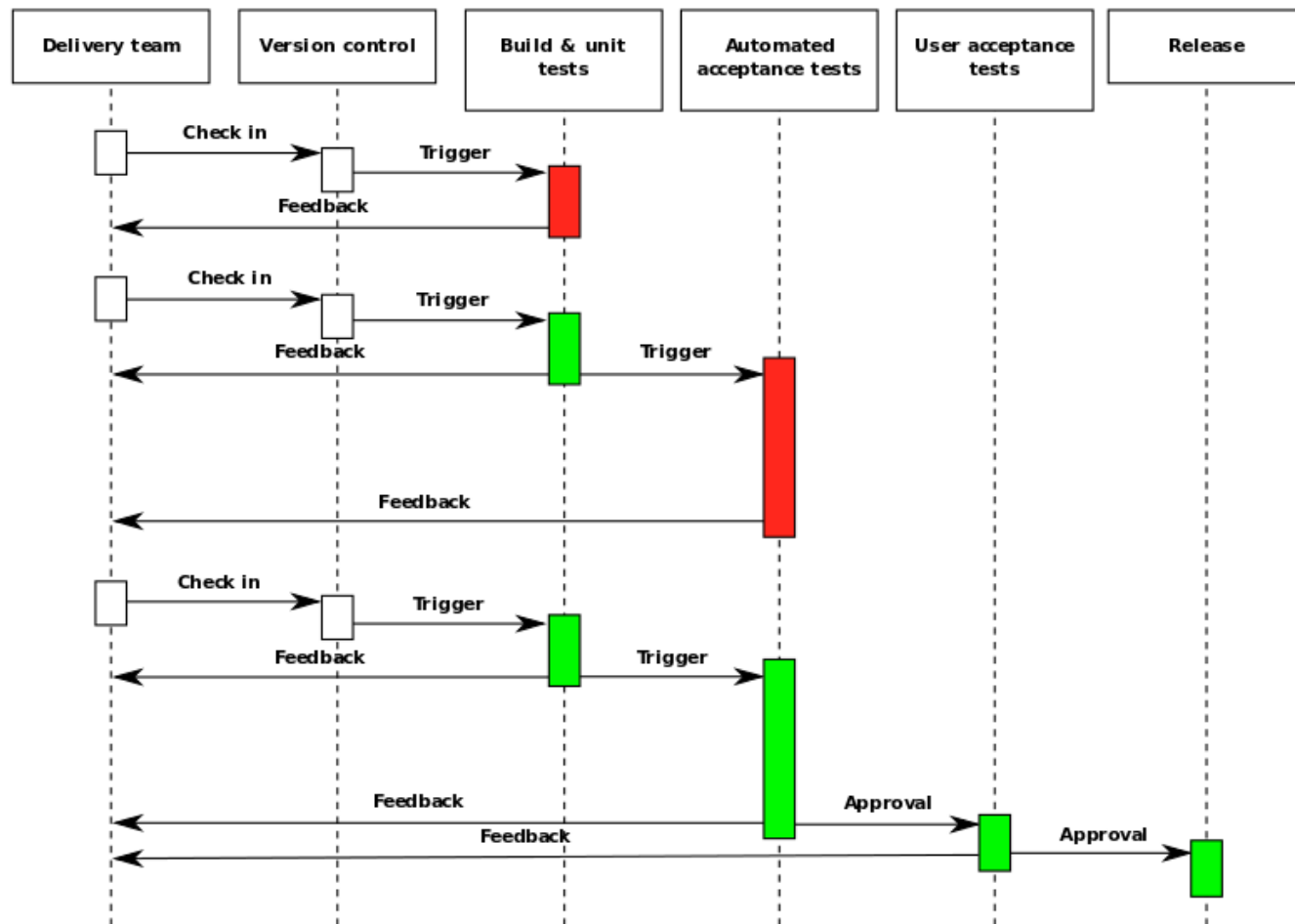
Proces wytwarzania kodu, kładący nacisk na komunikację i współpracę między managerami, zespołami developerskimi oraz zespołami operacyjnymi (tech)

Osiąga się to przez automatyzację i monitorowanie procesów integracji, testowania, wydawania (deployment) oprogramowania oraz zmianami infrastruktury.

DevOps to nie tylko proces ale "kultura" organizacji.







## 3. Jenkins

# Przygotowanie

git clone <https://github.com/ArkadioG/jenkins-testy>

fork: <https://github.com/LableOrg/java-maven-junit-helloworld>

# Jenkins docker

(przygotowane na poprzednich zajęciach)

# Jenkins – Instalacja docker

hardway:

tworzymy plik o nazwie `dockerfile` zawartość pliku:

```
FROM jenkins/jenkins:lts
USER root
ENV JENKINS_OPTS --httpPort=-1 --httpsPort=9090
RUN apt-get update && apt-get install -y maven
```

w terminalu (z folderu z `dockerfile`) polecenie:

```
sudo docker build -t jenkins-maven .
```

**easy way** – kopiujemy plik `dockerfile` z repo i wpisujemy w terminalu powyższą komendę (*sudo docker build...*)

Po instalacji można sprawdzić czy image jest zainstalowany:

```
sudo docker images
```

# Jenkins – przed uruchomieniem docker

Przygotowanie folderów (tylko dla celów zajęć)

```
mkdir -p /home/username/jenkins  
chmod 777 /home/username/jenkins  
mkdir -p /home/username/wildfly  
chmod 777 /home/username/wildfly
```

# Jenkins –uruchomienie docker

Wpisujemy w terminalu polecenie (to jest jedna linijka):

```
sudo docker run --name jenkins-maven -p 9090:9090 -p 50000:50000  
-v ~/jenkins:/var/jenkins_home  
-v ~/wildfly:/opt/wildfly jenkins-maven
```

Po instalacji:

<https://localhost:9090>

Powyższe polecenie - docker uruchamia obraz Jenkins-maven, konfiguruje przekierowanie portów z kontenera oraz mapuje foldery z kontenerów na foldery fizycznej maszyny

znak tyldy ~ oznacza folder domowy czyli `/home/username`



# Jenkins standardowa instalacja

# Wymagania - Java

- Linux:

polecenia w terminalu

```
java -version          -sprawdzamy wersję  
sudo apt-get install default-jre    - instalacja
```

- Windows:

Pobieramy instalator: <https://www.java.com/pl/download/>

**Uważamy aby odznaczyć śmieci, które proponuje nam zainstalować Oracle!**



# Jenkins - Instalacja

<https://jenkins.io/doc/book/getting-started/installing/>

hard way – wpisujemy ręcznie polecenia:

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -  
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >  
/etc/apt/sources.list.d/jenkins.list'  
sudo apt-get update  
sudo apt-get install jenkins
```

**easy way** - uruchamiamy skrypt install\_jenkins.sh pobrany z repo:

```
sudo sh install_jenkins.sh
```

Po instalacji sprawdzamy czy usługa Jenkinsa działa

```
sudo service jenkins status
```

**Folder instalacji:**

```
/var/lib/jenkins/
```

# Jenkins – przed uruchomieniem

Z Jenkinsa korzysta się poprzez przeglądarkę – domyślna instalacja użyje portu **8080**. (localhost:8080 ; 127.0.0.1:8080)

Jeśli na tym porcie już mamy jakąś usługę to należy zmienić konfigurację Jenkinsa aby korzystał z innego portu. Zmieniamy pliki:

**Linux:**

**/etc/default/jenkins**

zmieniamy wartość klucza **HTTP\_PORT=8080**

**Win:**

**c:\...ścieżka...\jenkins.xml**

zmieniamy wartość **-httpPort=8080**

# Jenkins konfiguracja

Konfiguracja krok po kroku opisana w poprzedniej prezentacji!

# Jenkins – uruchomienie

W przeglądarce:

localhost:8080   127.0.0.1:8080   nazwa\_komputera:8080

## Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

# Jenkins – uruchomienie

Postępujemy zgodnie z poleceniami:

- kopiujemy ze wskazanego pliku hasło do pierwszego logowania
- instalujemy domyślne pluginy
- tworzymy własnego użytkownika – wchodzimy do J

# Jenkins - konfiguracja

konfigurujemy najczęściej w dwóch miejscach

- zarządzaj Jenkinsem -> skonfiguruj system
- zarządzaj Jenkinsem -> Globalne Narzędzia

Przy zdecydowanej większości opcji jest znak zapytania, wystarczy go kliknąć aby uzyskać odpowiedź.



# Jenkins - konfiguracja zajęcia

Instalujemy pluginy: (Zarządzaj Jenkinsem - Zarządzaj wtyczkami)

git plugin

github plugin

maven integration

Po instalacji restartujemy Jenkinsa.

# Jenkins - projekty

# Jenkins - Projekt

- Add Project – Freestyle Project, podajemy nazwę – bez spacji!
- konfigurujemy projekt - workspace
- repozytorium – Git – podajemy dane github
- wyzwalacz – Pobierz z repozytorium kodu
- budowanie
- kroki po budowaniu – publishing, powiadomienia

# Jenkins podstawowy projekt

- Tworzymy nowy projekt –
- podajemy nazwę projektu (bez spacji)
- wybieramy typ – Freestyle Project
- klikamy ok
- konfiguracja: scm, trigger, build, akcje post build

UWAGA - ścieżki są relatywne do workspace'a

# Jenkins podstawowy projekt zadania

1. Utwórz projekt hello\_world - nazwa wyświetlana 'Hello World', bez SCM, trigger ręczny, akcja build - w shellu wypisać Hello World ([echo](#))
2. projekt moje\_pliki, SCM - github (jakieś własne repo), trigger - pobranie co 3 minuty, build - w shellu wyświetlenie zawartości pliku ([cat](#) / [less](#))

# Jenkins podstawowy projekt zadania

3 / 4. Przygotuj job dla projektu maven - raz używając typu ogólny projekt, raz typ Maven - trigger pobranie co 5 minut, scm git ([fork tego repo](#), lub własny projekt maven),

# Jenkins projekty zależne

Możemy utworzyć projekty, które będą budować się w zależności od poprzedniego projektu.

Np. projekt z testami uruchomi się po projekcie budowania aplikacji.


Dzięki takiemu układowi możemy mieć tylko jeden projekt puszczający testy, który będzie obsługiwać wiele projektów głównych.



# Jenkins projekty zależne

Korzystamy z opcji w projekcie:

**Akcje po zadaniu**

 **Uruchom inne zadania**

Projects to build

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

**Dodaj akcje po zadaniu** ▼

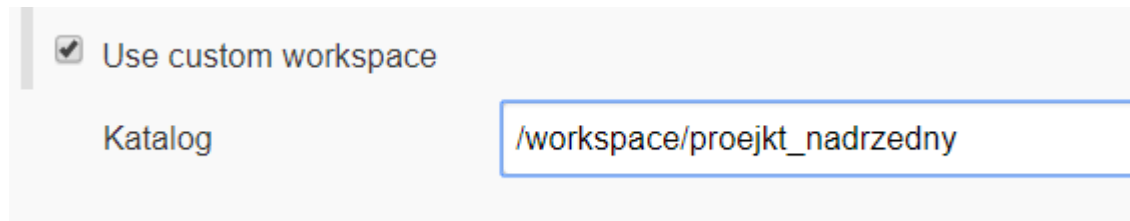


# Jenkins projekty zależne

Musimy zastanowić się czy w projektach zależnych będziemy korzystać z workspace'a głównego projektu, czy będą one posiadać własne workspace'y.

Konfiguracja projektu - General / Advanced - Use custom workspace

Ścieżka jest relatywna do folderu \$JENKINS\_HOME



☒ Use custom workspace

Katalog

# Jenkins projekty zależne

## zadania

5. Utwórz (używając repo z projektem maven) zależne projekty odpowiadające mavenowym goal'om, uruchamiające się jeden po drugim, korzystające ze wspólnego workspace'a

# Jenkins - możliwości

- pobieranie kodu z wielu rodzajów SCM
- zadania mogą uruchamiać się nawzajem, warunkowanie zachowania
- wykonywanie skryptów w różnych językach
- wykonywanie testów, historia testów, code coverage
- generowanie dokumentacji
- prezentacja danych
- powiadomienia (Slack, mail, skype etc.)

## 3. Dobre Praktyki

# Dobre praktyki

- ❑ Jedno repozytorium dla wszystkich, miej strategię branchowania
- ❑ automatyzacja budowania
- ❑ uruchamianie testów automatycznie z buildami
- ❑ wszyscy komitują jak najczęściej (codziennie)
- ❑ każdy commit do głównego brancha powinien być zbudowany
- ❑ optymalizacja procesu CI – ma być szybko!
- ❑ testuj w klonie środowiska dev
- ❑ wizualizuj stan buildów
- ❑ zautomatyzuj deployment



# Thanks!!

Arkadiusz Gutkowski

[github.com/ArkadioG](https://github.com/ArkadioG)