



JIT SOLUTIONS

Selenium Grid, Docker, Zalenium i Jenkins,
czyli jak od zera zbudować infrastrukturę
testową dla aplikacji webowych



Tomasz Klepacki

16/10/2018

Agenda

- Testy przeglądarkowe – problemy i wyzwania
- Jenkins jako narzędzie wspierające CI/CD
- Docker - wprowadzenie
- Jenkinsa – instalacja z obrazu Docker’a
- Jenkins Pipeline – konfiguracja job’a oraz uruchomienie testów
- Jenkins – architektura master <-> slave
- Paralelizacji testów dla różnych przeglądarek z użyciem TestNG oraz Maven Surefire Plugin
- Postawienie infrastruktury testowej w oparciu Selenium Grid i Dockera oraz uruchomienie testów z Jenkins’a
- Postawienie infrastruktury testowej w oparciu Selenium Grid i Zalenium oraz uruchomienie testów z Jenkins’a



Sprawy organizacyjne

- Repozytorium:

<https://github.com/tklepacki/aa-days>

- Zadania i rozwiązania:

<https://drive.google.com/open?id=10lekMR2J81T1CjGVrqNRXB4MmyNeUUjp>

- Notatki:

<https://docs.google.com/document/d/1RqUe1DkDPB2K4awu-X7C8339D3F7fzcSs3kid5S84Uc/edit?usp=sharing>

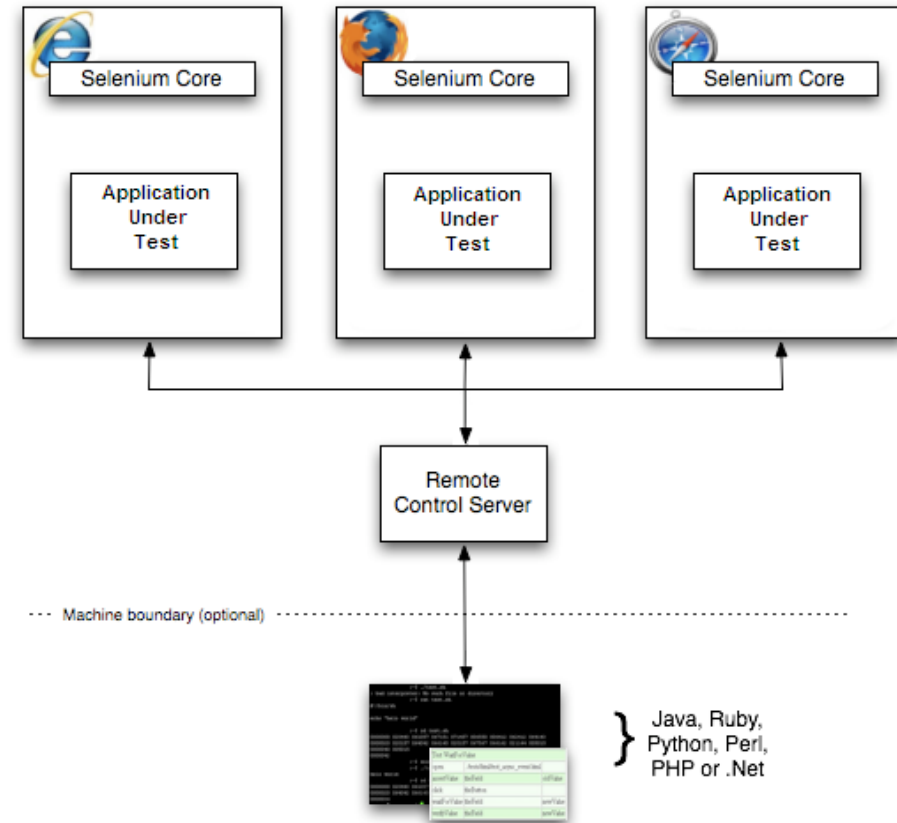


Selenium WebDriver

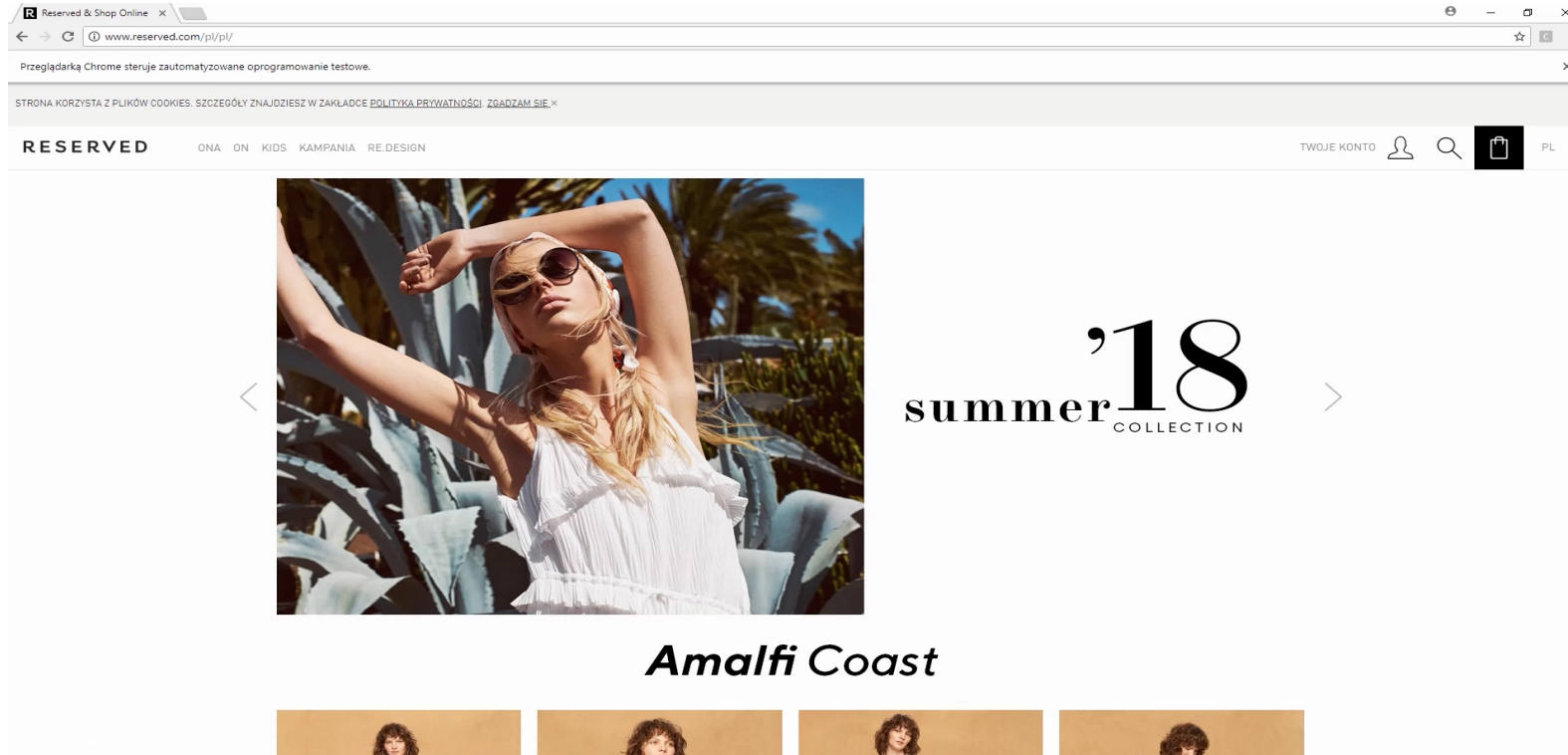
- Przyjazne API
- Współpracuje z wieloma językami programowania
- Obsługuje wiele przeglądarek
- Działanie -> RESTful web service -> JSON do komunikacji z przeglądarką



Windows, Linux, or Mac (as appropriate)...



Selenium WebDriver - Rejestracja klienta



JIT SOLUTIONS

Testy przeglądarkowe - wyzwania

- Zbyt długi czas trwania testów UI
- Deweloperzy – chcą szybki feedback czy ich kod działa prawidłowo
- Biznes – chciałby testów na różnych przeglądarkach

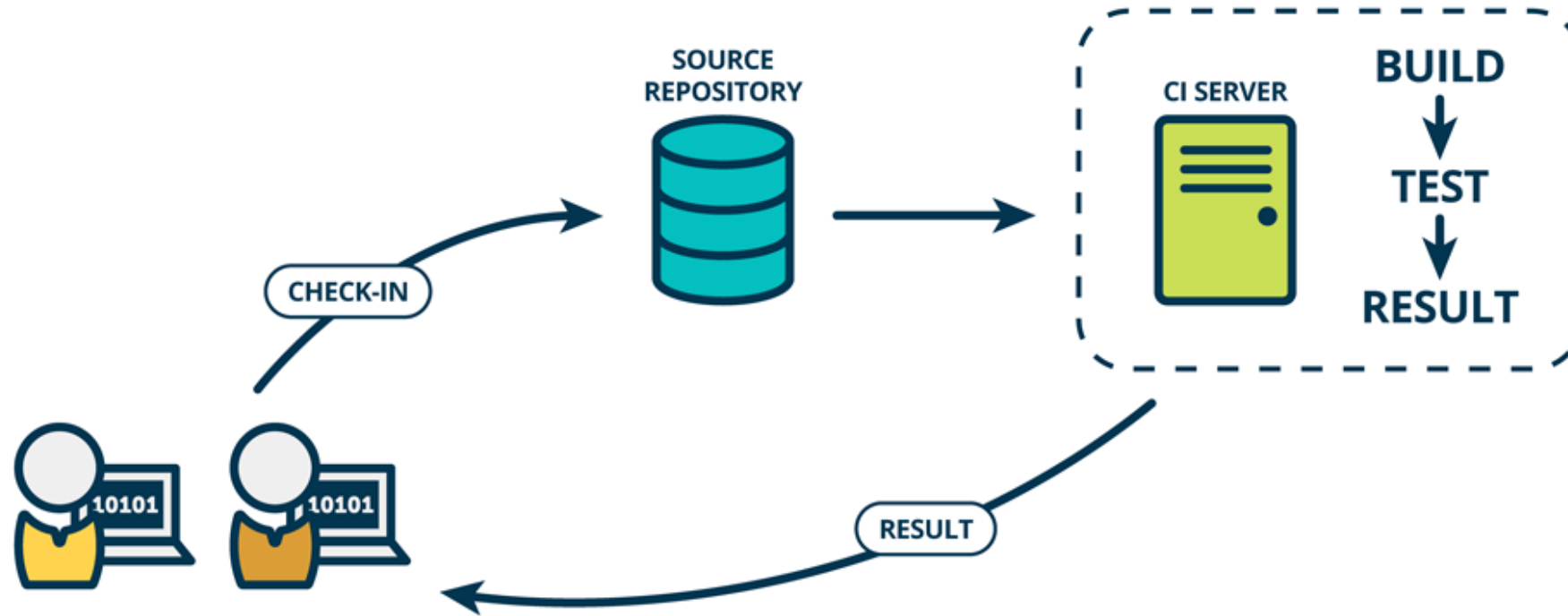


Zadanie I - Uruchomienie testów - weryfikacja (5 min)

1. Uruchom zestaw testowy (***src/test/resources/testsuite.xml***) z poziomu IDE.
2. Wejdź do folderu z projektem w konsoli i uruchom zestaw testowy z poziomu Maven'a:
 - ***./mvnw clean install*** (Linux)
 - ***mvnw clean install*** (Windows)
3. W obu przypadkach powinny wykonać się cztery test logowania użytkownika – dwa dla sklepu E-commerce Reserved w przeglądarce Firefox oraz dwa dla sklepu Mohito w przeglądarce Chrome. Testy powinny uruchomić się sekwencyjnie (jeden po drugim).



Continuous Integration



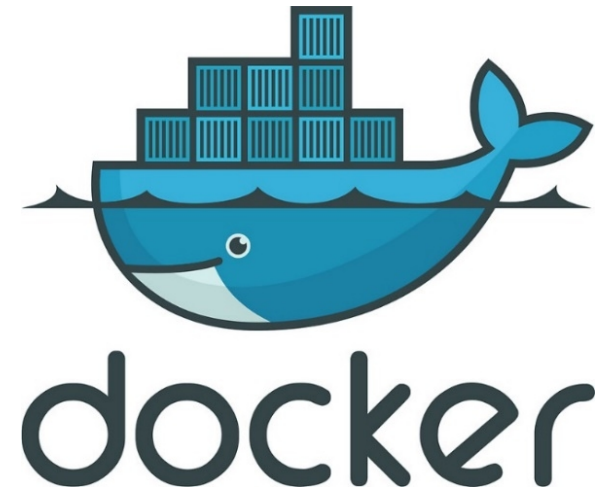
Jenkins

- Automatyzuje budowanie, deployment oraz testowanie aplikacji
- Wieloplatformowość
- Skalowalny, rozszerzalny
- Open-Source + Community












Docker

Docker – narzędzie, które pozwala umieścić program oraz jego zależności w lekkim, przenośnym, wirtualnym kontenerze, który można uruchomić na serwerze z systemem Linux oraz od niedawna również na systemie Windows



Docker Hub - Repozytoria obrazów

 <input type="text" value="Search"/>				Dashboard	Explore	Organizations	Create	 tklepacki
Explore Official Repositories								
	nginx official	9.6K STARS	10M+ PULLS	> DETAILS				
	alpine official	4.3K STARS	10M+ PULLS	> DETAILS				
	busybox official	1.4K STARS	10M+ PULLS	> DETAILS				
	httpd official	2.0K STARS	10M+ PULLS	> DETAILS				
	redis official	5.8K STARS	10M+ PULLS	> DETAILS				
	mongo official	5.0K STARS	10M+ PULLS	> DETAILS				
	ubuntu official	8.4K STARS	10M+ PULLS	> DETAILS				



Zadanie II - Docker - Podstawowe komendy (5 - 10 min)

1. ***docker info*** – wyświetla szczegółowe informacji o Dockerze zainstalowanym na maszynie użytkownika.
2. ***docker pull*** – pobiera obraz repozytorium np. ***docker pull hello-world***.
 - Wprowadź komendę: ***docker pull hello-world***
3. ***docker image ls*** – wyświetla wszystkie pobrane obrazy.
4. ***docker run*** – tworzy i uruchamia kontener np. ***docker run hello-world***.
 - Wprowadź komendę: ***docker run hello-world***
5. ***docker ps -a*** – wyświetla wszystkie utworzone kontenery.
6. ***docker rm <id kontenera>*** – usuwa kontener o podanym id.



Zadanie III - Instalacja Jenkinsa z obrazu Dockera (15 min)

1. Uruchom kontener z obrazu Jenkins'a:
 - **`docker run -v jenkins_home:/var/jenkins_home -p 8080:8080 -p 50000:50000 --name jenkins jenkins/jenkins:its`** (Linux)
 - **`docker run -v jenkins_home:/var/jenkins_home -p 8080:8080 -p 50000:50000 --name jenkins jenkins/jenkins:its`** (Windows)
2. Wejdź do Jenkins'a poprzez: **`localhost:8080`** lub **`<ip_maszyny_dockera>:8080`** z poziomu przeglądarki. IP maszyny dockera, możesz uzyskać wykonując komendę **`docker-machine ip`** (docker toolbox)
3. Wejdź do terminala kontenera: **`docker exec -t -i jenkins /bin/bash`**
4. Z terminala kontenera odczytaj hasło administracyjne: **`cat /var/jenkins_home/secrets/initialAdminPassword`**
5. Odblokuj Jenkinsa wpisując hasło administracyjne.
6. Wybierz instalację wersji z najbardziej popularnymi pluginami.
7. Utwórz użytkownika administracyjnego wypełniając formularz.
8. Na etapie wpisywania „**Instance Configuration**” pozostaw w polu Jenkins URL domyślną wartość czyli:
`http://localhost:8080` lub **`<ip_maszyny_dockera>:8080`**



Zadanie IV – Jenkins Pipeline (15 min)

1. Kliknij w „**New Item**”.
2. Utwórz projekt typu „**Pipeline**” z dowolną nazwą np. „**Web_Test_Automation**”
3. W oknie konfiguracji przejdź do sekcji „**Pipeline**”, kliknij w „**try simple Pipeline**” i wybierz: „**GitHub + Maven**”
4. Przerób skrypt w taki sposób, aby składał się trzech etapów:
 - **Pobranie repozytorium** – jako pobierane repozytorium ustaw projekt – <https://github.com/tklepacki/aa-days.git>. Dodaj parametr do komendy git, tak aby kod był pobierany ze wskazanego brancha. Do pomocy skorzystaj z modułu pipeline-syntax.
 - **Uruchomienie testów** – w zależności od systemu operacyjnego jakiego posiadasz wstaw odpowiednią komendę mvn, która uruchomi testy. Usuń kod definiujący zmienną mvnHome.
 - **Publikacja wyników testów** - zainstaluj plugin **testng-plugin** pozwalający na publikację raportów TestNG w Jenkinsie (Manage Jenkins → Manage Plugins → Zakładka Available → znajdź TestNG Results plugin → zainstaluj bez restartowania Jenkinsa). Znajdź dokumentację pluginu i sprawdź w jaki sposób możesz zdefiniować publikowanie danych testowych w skrypcie Pipeline.
5. Uruchom Build – wejdź do logów konsoli i sprawdź co się wydarzyło.



Jenkins CI - Terminologia



Master



Udostępnia interfejsy typu WEB oraz API



Przechowuje oraz kolejkuje zadania (builds)



Analizuje raporty, generuje powiadomienia



Slave



Wykonuje zadania (builds)



Generuje raporty



JIT SOLUTIONS

Jenkins CI - Architektura Master <-> Slave



Zadanie V - Utworzenie maszyny typu SLAVE (10 min)

1. Stwórz folder, który będzie zawierał agenta oraz przestrzeń roboczą dla zadań: „ ***/home/tomek/jenkins_node***” (Linux) lub ***C:\Users\<USER>\jenkins_node*** (Win)
2. Kliknij w „***Manage Jenkins***”, a następnie „***Manage Nodes***”.
3. Kliknij w „***New Node***”, a następnie zaznacz „***Permanent Agent***”. Wpisz nazwę node’a: „***local_machine***”.
4. W „Remote root directory” wpisz: ***/home/<USER>/jenkins_node*** (Linux) lub ***C:\Users\<USER>\jenkins_node*** (Win)
5. Wpisz w „***Labels***” : „***local_machine***”
6. Jako „***Launch method***” wybierz „***Launch agent via Java Web Start***”.
7. Zatwierdź konfigurację.
8. Wejdź na node „***local_machine***”.
9. Pobierz ***agent.jar*** i umieść go w folderze ***/home/<USER>/jenkins_node*** lub ***C:\Users\<USER>\jenkins_node***
10. Skopiuj komendę uruchamiającą agenta Jenkinsa i wykonaj ją w terminalu.
11. Sprawdź czy połączenie między slavem a masterem zostało ustanowione.
12. Zmodyfikuj skrypt Pipeline’a wskazując node’a, na którym build będzie wykonywany - ***node ('local_machine')***
13. Uruchom build.



TestNG jako narzędzie wspierające paralelizację egzekucji testów

- Alternatywa dla biblioteki JUnit
- Dodatkowe funkcjonalności: grupowanie testów, tworzenie zależności pomiędzy testami, parametryzacja danych testowych
- **Zawiera wbudowaną funkcjonalność jednoczesnego wykonywania testów**



JIT SOLUTIONS

Test
NG

TestNG - Zestawy testowe

- Definiowanie zakresu zestawów testowych w plikach XML – ***testng.xml***
- Poziomy paralelizacji: metody testowe, klasy testowe, testy w zestawach testowych

```
<?xml version="1.0" encoding="UTF-8"?>
<suite name="Test Method Suite" parallel="methods" thread-count="2">
  <test name="Test Method Test">
    <classes>
      <class name="ParallelMethodTest" />
    </classes>
  </test>
</suite>
```



Maven Surefire Plugin

- Wykorzystywany w trakcie etapu wykonywania testów
- Wspiera JUnit oraz TestNG
- Generuje raporty testowe
- **Wspiera paralelizację wykonywania testów**

maven



JIT SOLUTIONS

Paralelizacja zestawów testowych - konfiguracja POM.XML

Paralelizacja per:

- Methods
- Classes
- Suites
- SuitesAndClasses
- SuitesAndMethods
- ClassesAndMethods
- All

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.22.0</version>
  <configuration>
    <suiteXmlFiles>
      <file>src/test/resources/testsuitechrome.xml</file>
      <file>src/test/resources/testsuitefirefox.xml</file>
    </suiteXmlFiles>
    <properties>
      <property>
        <name>suitethreadpoolsize</name>
        <value>2</value>
      </property>
    </properties>
  </configuration>
</plugin>
```



TestNG + Maven Surefire Plugin - Parametryzacja zestawów testowych

■ testsuitechrome.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="All" parallel="classes" thread-count="4">
    <test name="loginChromeTests">
        <parameter name="browser" value="chrome"></parameter>
        <packages>
            <package name=".*" />
        </packages>
    </test>
</suite>
```

■ testsuitefirefox.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="All" parallel="classes" thread-count="4">
    <test name="loginFirefoxTests">
        <parameter name="browser" value="firefox"></parameter>
        <packages>
            <package name=".*" />
        </packages>
    </test>
</suite>
```

■ @BeforeMethod

```
@Parameters("browser")
@BeforeMethod
public void setUp(String browser) {
    driver = new WebDriverCreator().createDriver(browser);
    commonTestSteps = new CommonTestSteps(driver);
    manager = new PageObjectManager(driver);
}
```

■ pom.xml

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.22.0</version>
    <configuration>
        <suiteXmlFiles>
            <file>src/test/resources/testsuitechrome.xml</file>
            <file>src/test/resources/testsuitefirefox.xml</file>
        </suiteXmlFiles>
        <properties>
            <property>
                <name>suitethreadpoolsize</name>
                <value>2</value>
            </property>
        </properties>
    </configuration>
</plugin>
```



Zadanie VI - Uruchomienie wszystkich testów równoległe na dwóch różnych przeglądarkach (10 min)

1. Stwórz dwa zestawy testowe: „**testuitechrome.xml**” oraz „**testsuitefirefox.xml**”.
2. Każdy z nich będzie uruchamiał wszystkie testy, ale na dwóch różnych rodzajach przeglądarek – **Firefox** i **Chrome**.
3. Każdy zestaw będzie posiadał nazwę parametru „**browser**” z wartością „**chrome**” dla jednego zestawu i „**firefox**” dla drugiego.
4. Każdy zestaw będzie uruchamiał pakiet całego projektu: **<package name=".*" />**
5. Oprócz tego każdy zestaw będzie posiadał paralelizację na poziomie klas z maksymalną ilością wątków 4.
6. W **pom.xml** zdefiniuj paralelizację na poziomie suite’ów z ilością wątków 2. Zdefiniuj uruchamianie zestawów testowych, podając do nich ścieżki: „**src/test/resources/testuitechrome.xml**” oraz analogicznie „**src/test/resources/testuitefirefox.xml**”.
7. Zmodyfikuj metodę **setUp** w każdej klasie tak, aby przyjmowała parametr typu string – **browser** – dodaj anotację **@Parameters(„browser”)** nad metodą **setUp(String browser)**.
8. Uruchom testy komendą: **./mvnw clean install** (Linux) **lub mvnw clean install** (Windows) lokalnie.



Opcja I: SaaS - Infrastruktura w chmurze

■ Plusy:

- Gotowa infrastruktura do wykonywania testów UI
- Wsparcie obsługi wszystkich popularnych przeglądarek
- Łatwość debugowania testów
- Wsparcie paralelizacji
- Brak konieczności utrzymywania infrastruktury testowej

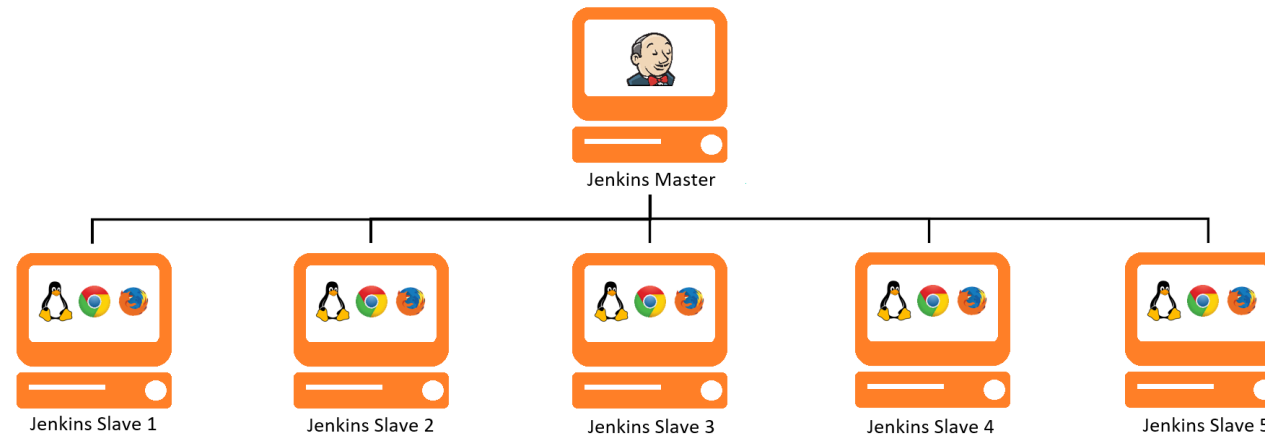
■ Minusy:

- Długi czas działania testów dla aplikacji widocznych w sieci wewnętrznej
- Długi czas tworzenia sesji
- Wysoki koszt subskrypcji (rośnie w miarę zwiększania paralelizacji)



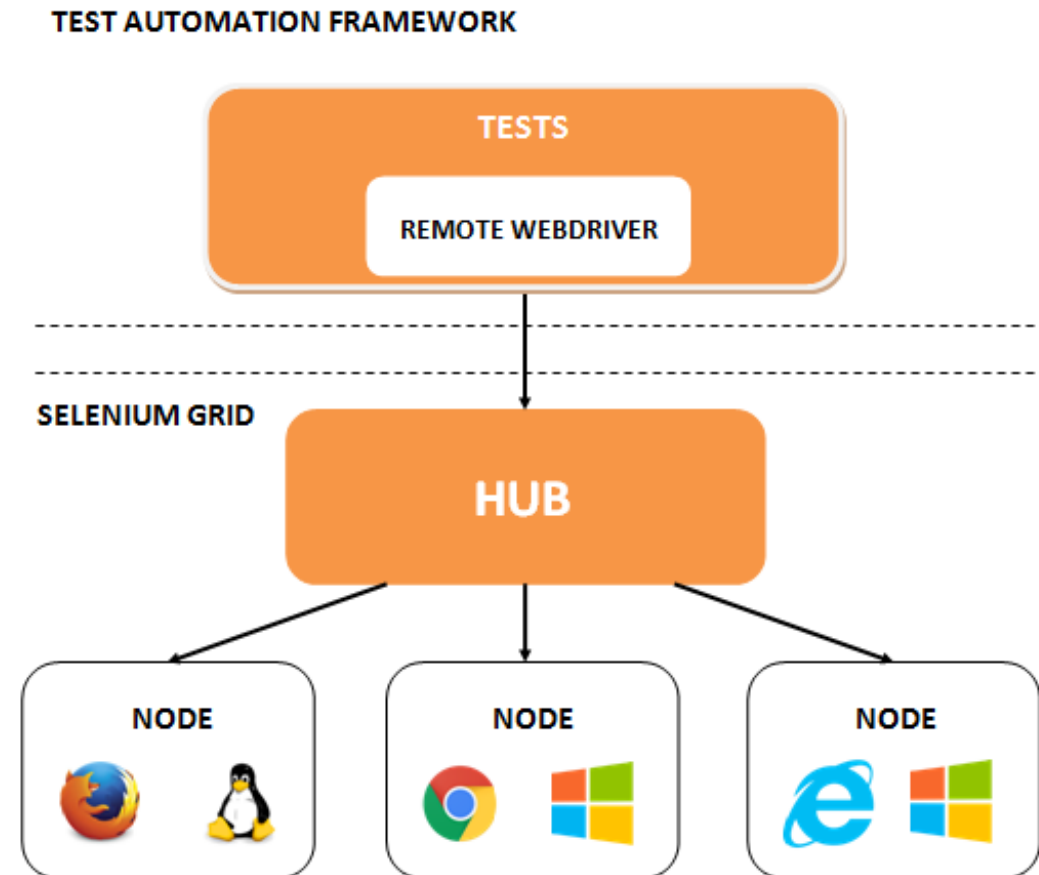
Opcja II: Samodzielne postawienie infrastruktury za pomocą VM's

- Plusy:
 - Większa kontrola nad infrastrukturą testową
- Minusy:
 - Konieczność ciągłego utrzymywania infrastruktury testowej
 - Wysoki koszt
 - Konieczność zapewnienia paralelizacji



Opcja III: Selenium Grid

- Dedykowane narzędzie do uruchamiania testów Selenium na wielu przeglądarkach, systemach oraz maszynach
- Sam w sobie nie posiada funkcjonalności paralelizacji
- Oferuje infrastrukturę składającą się z Hub'a oraz Node'ów



Selenium Grid - konfiguracja

- Komenda uruchamiająca HUB:

```
java -jar selenium-server-standalone-3.14.0.jar -port 4444 -role hub
```

- Komendy uruchamiające Node'y dla przeglądarek:

- ✓ Firefox

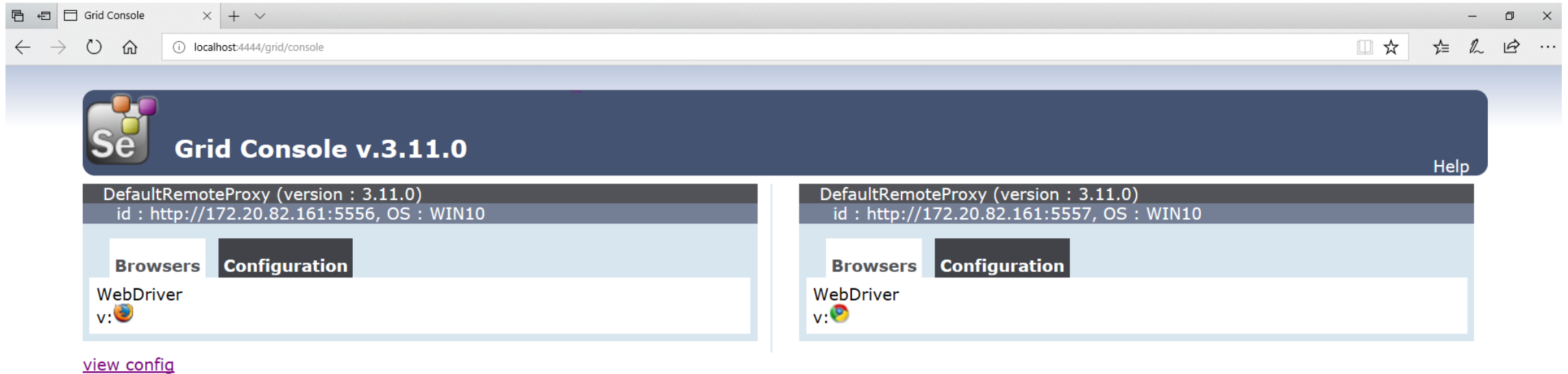
```
java -Dwebdriver.gecko.driver=geckodriver.exe -jar selenium-server-standalone-3.14.0.jar  
-role node -hub http://localhost:4444/grid/register -browser browserName=firefox -port 5556
```

- ✓ Google Chrome

```
java -Dwebdriver.chrome.driver=chromedriver.exe -jar selenium-server-standalone-3.14.0.jar  
-role node -hub http://localhost:4444/grid/register -browser browserName=chrome -port 5557
```





Selenium Grid - konsola



The screenshot displays the Selenium Grid Console v.3.11.0 interface in a web browser window. The browser's address bar shows the URL `localhost:4444/grid/console`. The interface features a dark blue header with the Selenium logo (Se) and the text "Grid Console v.3.11.0". A "Help" link is located in the top right corner of the header. Below the header, the interface is divided into two main sections, each representing a "DefaultRemoteProxy (version : 3.11.0)".

Each section contains the following information:

- id**: `http://172.20.82.161:5556, OS : WIN10` (for the left node) and `http://172.20.82.161:5557, OS : WIN10` (for the right node).
- Browsers** and **Configuration** tabs.
- WebDriver** v:  (for the left node) and  (for the right node).

A [view config](#) link is visible below the left node's details.




JIT SOLUTIONS

Selenium Grid - wyzwania

- Potrzeba instalacji dependencji na VM's
- Instalacja i aktualizacja serwera Selenium
- Utrzymywanie infrastruktury testowej
- Postawienie hub'a oraz node'ów











Docker - obrazy SeleniumHQ



selenium

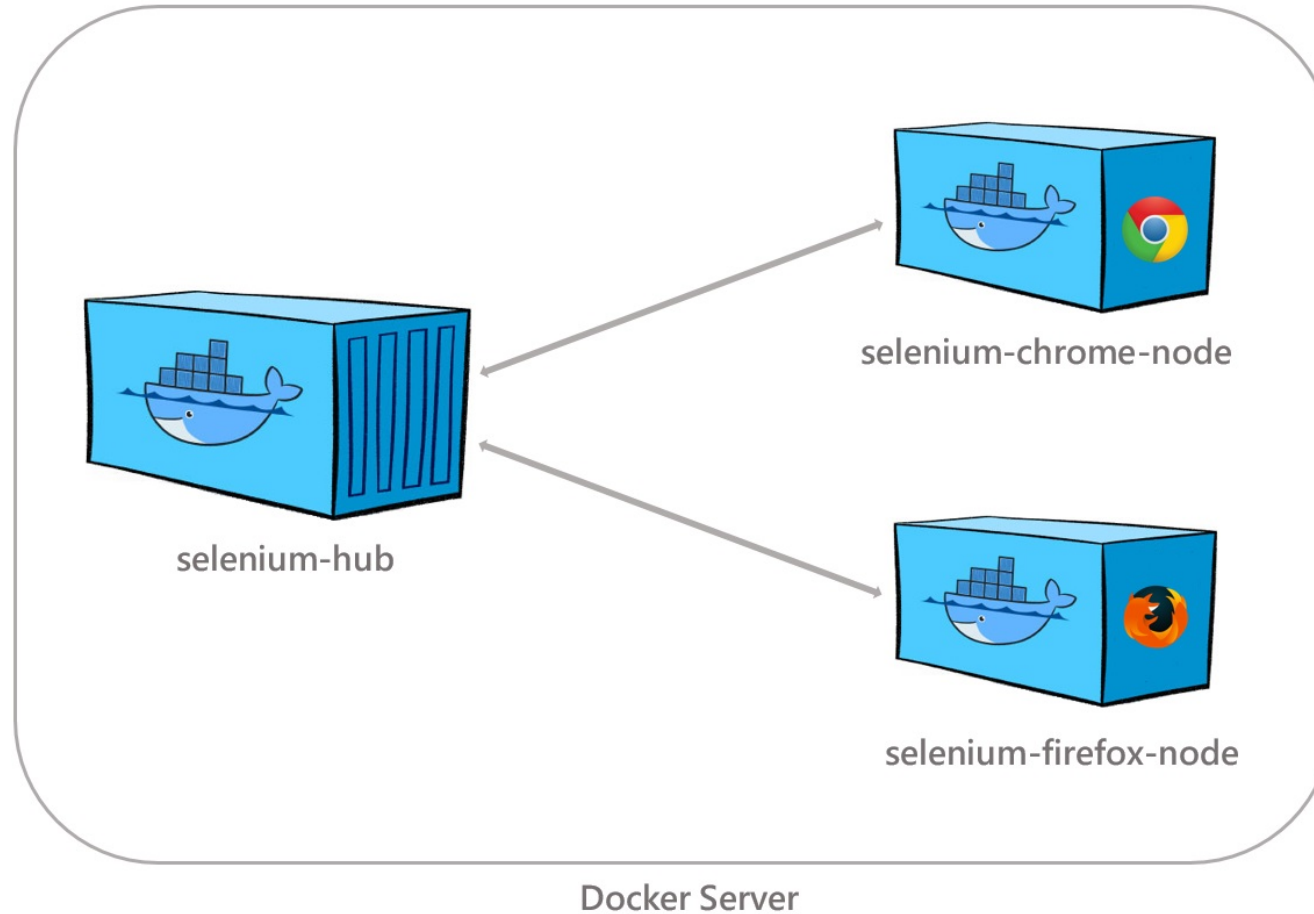
<http://docs.seleniumhq.org/>
Joined November 2014

 selenium/node-chrome public automated build	151 STARS	10M+ PULLS	> DETAILS
 selenium/standalone-chrome public automated build	122 STARS	10M+ PULLS	> DETAILS
 selenium/hub public automated build	259 STARS	5M+ PULLS	> DETAILS
 selenium/node-firefox public automated build	100 STARS	5M+ PULLS	> DETAILS
 selenium/standalone-chrome-debug public automated build	36 STARS	1M+ PULLS	> DETAILS
 selenium/standalone-firefox public automated build	78 STARS	1M+ PULLS	> DETAILS
 selenium/node-firefox-debug public automated build	22 STARS	1M+ PULLS	> DETAILS
 selenium/node-chrome-debug public automated build	36 STARS	1M+ PULLS	> DETAILS



JIT SOLUTIONS

Selenium Grid i Docker - architektura

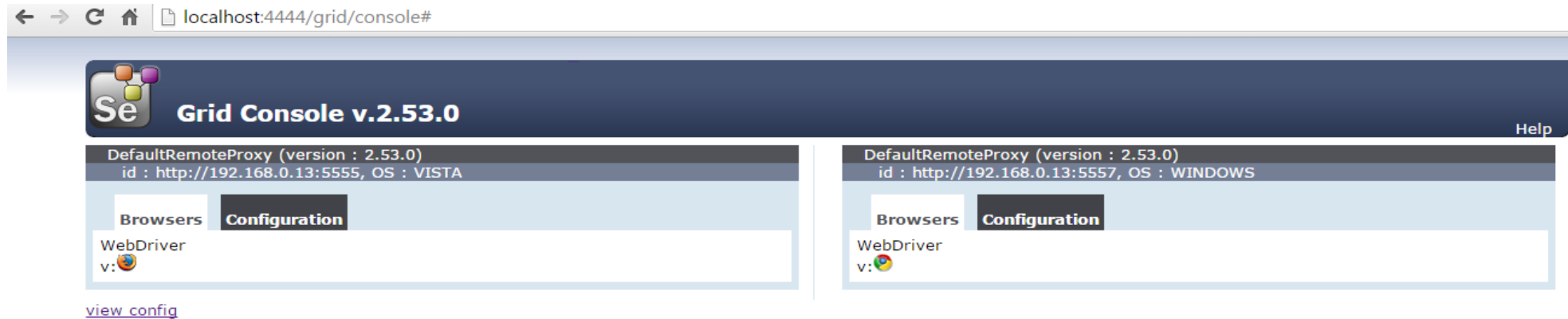


Zadanie VII - Postawienie infrastruktury Selenium Grid w kontenerach (10 - 15 min)

1. Wykonaj komendę uruchamiającą Selenium HUB:
 - ***docker run --name selenium-hub -p 4444:4444 selenium/hub***
2. Wykonaj komendy uruchamiające kolejno chrome-node oraz firefox-node:
 - ***docker run --name chrome-node -e NODE_MAX_SESSION=4 -e NODE_MAX_INSTANCES=4 --link selenium-hub:hub -v /dev/shm:/dev/shm -P selenium/node-chrome-debug***
 - ***docker run --name firefox-node -e NODE_MAX_SESSION=4 -e NODE_MAX_INSTANCES=4 --link selenium-hub:hub -v /dev/shm:/dev/shm -P selenium/node-firefox-debug***
3. Wejdź na ***localhost:4444*** lub ***<ip_maszyny_dockera>:4444*** z poziomu przeglądarki (IP maszyny dockera, możesz uzyskać wykonując komendę ***docker-machine ip***) i zweryfikuj czy do hub'a dołączone zostały node'y dla Chrome oraz FF.
4. Uruchom ***VNC Viewer***, a następnie utwórz połączenia z node'ami wpisując ***localhost:<port>*** lub - numer portu kontenera znajdz po wywołaniu komendy ***docker ps***. Wymagane hasło przy połączeniu to „***secret***”.



Selenium Grid - RemoteWebDriver



```
public RemoteWebDriver createDriver(String browser) throws MalformedURLException {
    switch (browser) {
        case "firefox":
            FirefoxOptions firefoxOptions = new FirefoxOptions();
            RemoteWebDriver firefoxDriver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"),
                firefoxOptions);
            setDefaultSettings(firefoxDriver);
            return firefoxDriver;
        case "chrome":
            ChromeOptions chromeOptions = new ChromeOptions();
            RemoteWebDriver chromeDriver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), chromeOptions);
            setDefaultSettings(chromeDriver);
            return chromeDriver;
        default:
            throw new IllegalArgumentException("The Browser Type is Undefined");
    }
}
```



Zadanie VIII - RemoteWebDriver oraz uruchomienie testów w infrastrukturze Selenium GRID z Jenkins'a (15 min)

1. Utwórz klasę inicjalizującą RemoteWebDriver'a.
2. Zmodyfikuj klasy testowe, aby korzystały z RemoteWebDriver'a przy uruchomieniu.
3. Utwórz własny branch ze zmianami, a następnie wypchnij go do zdalnego repozytorium na Github.
4. Dodaj do Pipeline job'a parametr odnoszący się do brancha zaciąganego z repozytorium
5. Uruchom build z parametrem wskazującym na utworzony branch.



Selenium Grid i Docker - docker-compose

```
version: "3"
services:
  selenium-hub:
    image: selenium/hub
    container_name: selenium-hub
    ports:
      - "4444:4444"
```

```
chrome-node:
  image: selenium/node-chrome
  container_name: chrome-node
  depends_on:
    - selenium-hub
  volumes:
    - /dev/shm:/dev/shm
  environment:
    - HUB_HOST=selenium-hub
    - HUB_PORT=4444
    - NODE_MAX_INSTANCES=4
    - NODE_MAX_SESSION=4
```

```
firefox-node:
  image: selenium/node-firefox
  container_name: firefox-node
  depends_on:
    - selenium-hub
  volumes:
    - /dev/shm:/dev/shm
  environment:
    - HUB_HOST=selenium-hub
    - HUB_PORT=4444
    - NODE_MAX_INSTANCES=4
    - NODE_MAX_SESSION=4
```

Docker-compose <file_name> up



Zadanie IX - Postawienie infrastruktury Selenium GRID za pomocą docker-compose z Jenkinsa (15 min)

1. Upewnij się, że kontenery ***selenium-hub***, ***firefox-node*** oraz ***chrome-node*** zostały zatrzymane i usunięte.
2. Zmodyfikuj skrypt Pipeline'a w taki sposób, aby przed testem automatycznie uruchamiał infrastrukturę Selenium GRID z docker-compose, a następnie po zakończeniu testów, zatrzymywał kontenery i usuwał je.
3. Dodaj do Pipeline Joba parametr typu (Choice Parameter) o nazwie ***TEST_INFRASTRUCTURE*** odnoszący się do nazwy katalogu „***seleniumgrid***”, w którym znajduje się plik ***docker-compose.yml***
4. Uruchom testy podając w parametrze nazwę własnego brancha oraz wybierając katalog „***selenium-grid***”

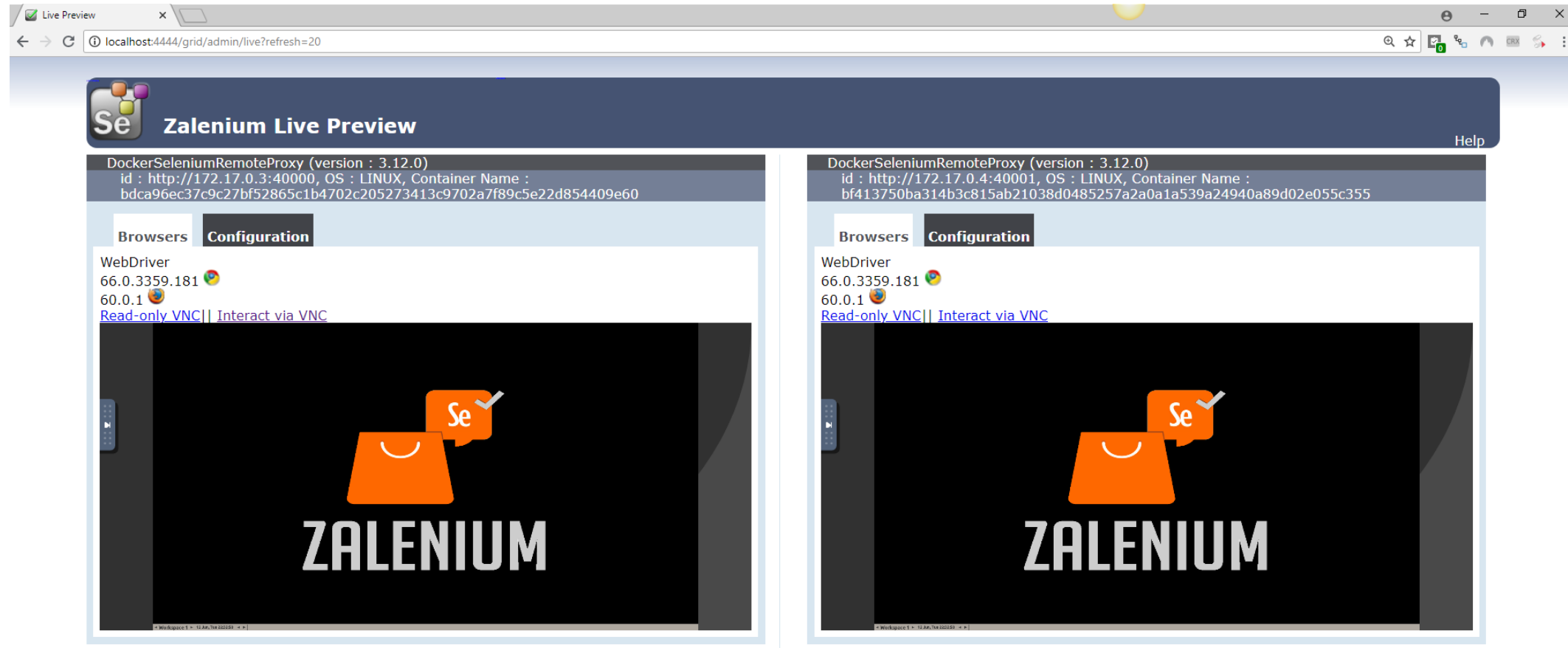


Zalenium

- Automatycznie skalowalna infrastruktura testowa oparta na Selenium Grid
- Zapewnia video z przebiegu każdego testu
- Podgląd ekranu przeglądarki w wersji LIVE
- Dashboard



Zalenium - Podgląd ekranów przeglądarek



Zalenium - Dashboard

Zalenium - Dashboard

Cleanup

Search

Tests (6)

8d08dfd5-bc2f-4b1b-b391-3d8edad1a6c5

13-Jun 20:55:11

firefox 60.0.1, LINUX

Zalenium

Completed

66915ef6-4916-4bbd-bc1a-268697c88aee

13-Jun 20:55:00

firefox 60.0.1, LINUX

Zalenium

Completed

c3fa7933-2cf3-49c7-a6d0-da5c4585f330

13-Jun 20:54:54

chrome 66.0.3359.181, LINUX

Zalenium

Completed

6e40675a-d950-4f7c-bdc8-060b1ec584a4

13-Jun 20:54:51

chrome 66.0.3359.181, LINUX

Zalenium

Completed

3dc968ec-ea01-4bbd-b7bb-dfa3b13853ec

13-Jun 20:54:50

chrome 66.0.3359.181, LINUX

Zalenium

Completed

7bbb020f-f882-486c-adfa-9d5f197e5456

13-Jun 20:54:44

firefox 60.0.1, LINUX

Zalenium

Completed

8d08dfd5-bc2f-4b1b-b391-3d8edad1a6c5

Completed

Date

13-Jun 20:55:11

Browser/Platform

firefox 60.0.1, LINUX

Screen Dimension

1920x1080

Time Zone

Europe/Berlin

Proxy

Zalenium

Build

Video

Logs

Mohito Shop Online - Mozilla Firefox

www.mohito.com.pl

MOHITO

Shop online

Get inspired

Szukaj

Zaloguj

Koszyk

Dominika Grosicka x MOHITO

Akcja charytatywna #mistrzowskiepolo

ZOBACZ WIECEJ

Nowości

0:20 / 1:10

Zalenium - konfiguracja

- Pobranie obrazu docker-selenium

```
docker pull elgalu/selenium
```

- Pobranie obrazu Zalenium

```
docker pull dosel/zalenium
```

- Uruchomienie Zalenium

```
docker run --rm -ti --name zalenium -p 4444:4444 -v /var/run/docker.sock:/var/run/docker.sock  
-v /tmp/videos:/home/seluser/videos --privileged dosel/zalenium start
```



Zadanie X - Postawienie infrastruktury testowej opartej na Zalenium i uruchomienie testów w Jenkins'ie (15 min)

1. Zatrzymaj, a następnie usuń kontenery selenium-hub, firefox-node oraz chrome-node jeśli takowe istnieją.

LINUX

2. Usuń ze skryptu Pipeline'a etap zatrzymujący i usuwający kontenery po ukończeniu testów.
3. W parametrze **TEST_INFRASTRUCTURE** dodaj do listy wyboru katalog o nazwie „**zalenium**”
4. Uruchom testy z własnego brancha oraz z parametrem wybierającym katalog „**zalenium**”, w którym znajduje się odpowiedni plik „**docker-compose.yml**”

WINDOWS

5. Uruchom ręcznie infrastrukturę testową ZALENIUM:
 - **`docker run --rm -ti --name zalenium -p 4444:4444 -v /var/run/docker.sock:/var/run/docker.sock -v /tmp/videos:/home/seluser/videos --privileged dosel/zalenium start`**
6. Usuń ze skryptu Pipeline'a etapy uruchamiające oraz zatrzymujące/usuwające kontenery po ukończeniu testów.
7. Uruchom testy z własnego brancha w Jenkinsie.

OSX

8. Uruchom ręcznie infrastrukturę testową ZALENIUM:
 - **`docker run --rm -ti --name zalenium -p 4444:4444 -e DOCKER=17.06.2-ce -v /var/run/docker.sock:/var/run/docker.sock -v /tmp/videos:/home/seluser/videos --privileged dosel/zalenium start`**
9. Usuń ze skryptu Pipeline'a etapy uruchamiające oraz zatrzymujące/usuwające kontenery po ukończeniu testów.
10. Uruchom testy z własnego brancha w Jenkinsie.

WSZYSTKIE SYSTEMY

11. Śledź wykonywanie testów w przeglądarce Chrome - wejdź na „**localhost:4444/grid/admin/live**” lub „**<ip_maszyny_dockera>:4444/grid/admin/live**” z poziomu przeglądarki. IP maszyny dockera, możesz uzyskać wykonując komendę **`docker-machine ip`** (docker toolbox)
12. Po wykonaniu testów wejdź na „**localhost:4444/dashboard/**” lub „**<ip_maszyny_dockera>:4444/dashboard/**” z poziomu przeglądarki. IP maszyny dockera, możesz uzyskać wykonując komendę **`docker-machine ip`** (docker toolbox)- sprawdź wyniki testów, odtwórz video z testu.



Selenium Grid, Docker, Zalenium i Jenkins - Podsumowanie

Podczas warsztatów dowiedzieliśmy się jak:

- Lokalnie stworzyć instancję Jenkinsa w kontenerze;
- Skonfigurować pipeline'y w Jenkinsie;
- Równolegle uruchamiać testy na kilku przeglądarkach;
- Postawić infrastrukturę testową opartą o Selenium Grid i Dockera używając oficjalnych Docker'owych obrazów dostarczanych przez SeleniumHQ;
- Postawić infrastrukturę testową opartą o Zalenium;
- Uruchomić testy Selenium w sposób równoległy z Jenkins Pipeline w infrastrukturze testowej Zalenium.



Dziękuję za uczestnictwo! :)



JIT SOLUTIONS