



Универзитет „Св. Кирил и Методиј“ - Скопје
Факултет за информатички науки и компјутерско инженерство



Continuous Integration / Continuous Deployment на веб апликација со помош на Jenkins.



ANSIBLE



docker



Jenkins



Изработил: Небојша Тодоровиќ 209015

Содржина

| | |
|--|----|
| Содржина..... | 2 |
| Вовед..... | 3 |
| Веб апликација..... | 4 |
| CI/CD..... | 6 |
| Jenkins..... | 6 |
| Ansible..... | 9 |
| NGINX..... | 10 |
| Docker..... | 10 |
| Споредба на контејнери во однос на виртуелни машини..... | 11 |
| Понатамошна разработка на проектот..... | 12 |
| Користена литература..... | 12 |

Вовед

Со развој на технологијата, промена на динамиката на животот, промена на потребите на бизнисите, паралелно се развивал и софтверот. Во денешницата скоро сите користиме софтвер, доколку некој не го користи директно за себе, со сигурност за него има податоци некаде во дигитален формат. Сите го користиме софтверот во комбинација со технологијата за различни цели, затоа и има различни типови на софтвер.

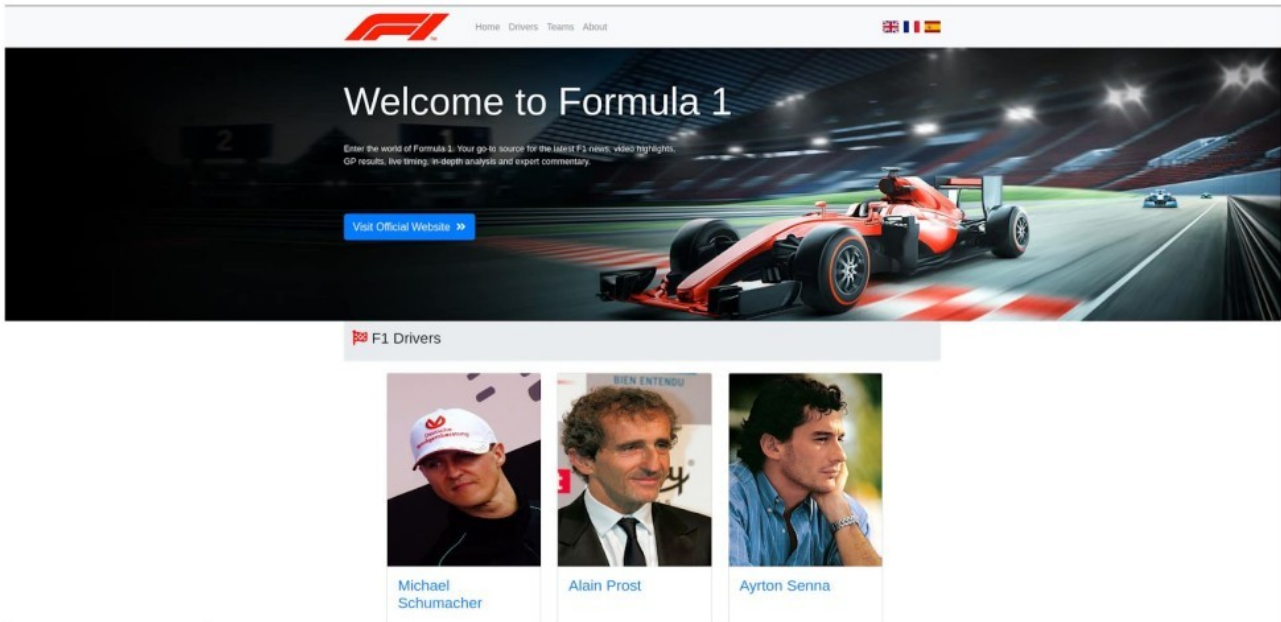
Луѓето се зависни од софтверот, ги извршуваат своите дневни задачи со помош на компјутер (во било каков облик), дури и возат припомогнати од страна на некаков софтвер. Но голем дел од софтверот, поради економска исплатливост и достапност, се наоѓа на вебот, односно е достапен онлајн. Многу бизниси зависат од тоа како се достапни и каде се достапни и затоа бараат брз квалитетен, модерен софтвер. Но да се биде онлајн повеќе не претставува само да се има статичка веб страна, туку постојат огромни софтверски системи. Динамиката на животот бара брзо прилагодување и брзо прифаќање на новите промени, така е и во развој на софтверот.

Девелоперите секојдневно се соочуваат со предизвици како брзо, ефикасно и квалитетно да ги достават бараните промени и подобрувања на нивниот софтвер, со што е можно помалку грешки во истиот. Клиентите сакаат и да имаат визуелен приказ на што конкретно се работи при развој на софтверот, што придонесува до подобра комуникација со нив. При ова гледање на клиентите се тестира самиот софтвер, што значи рано откривање на грешките, но сепак секоја апликација треба да има и строго напишани тестови, кои треба во било кој случај на извршување да имаат успешен резултат. Доста често има потреба од доставување на софтверот до клиентот додека сеуште се работи на негово развивање, а тоа претставува проблем за девелоперите.

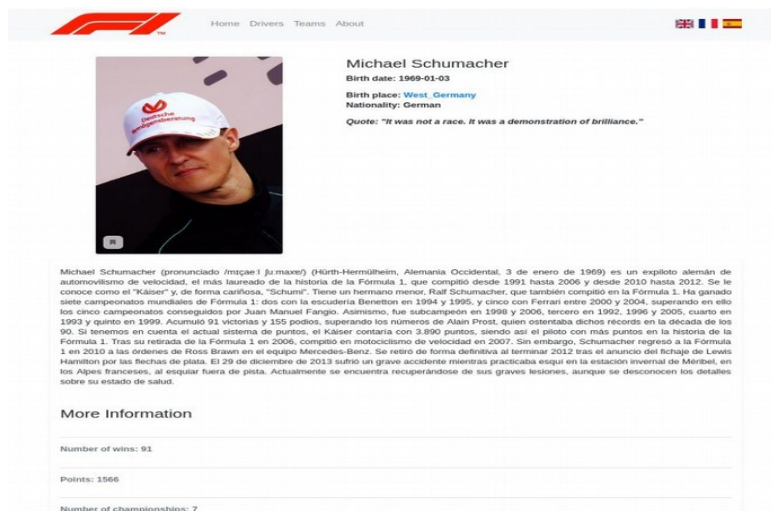
Целта на овој проект е да се олесни извршување на тестовите и достава до клиентот и корисникот на една веб апликација, да се исконфигурира сервер со конфигурација која е лесно пренослива. За извршување на тестовите и достава на софтверот додека е во развој („development“ околина) искористен е Jenkins, за конфигурација на серверот и самата достава на софтвер искористен е Ansible, а за опслужување на веб апликацијата, како и можноста за лесна преносливост искористен е Docker, а за полесно справување со Docker контејнерите искористен е Docker-compose.

Веб апликација

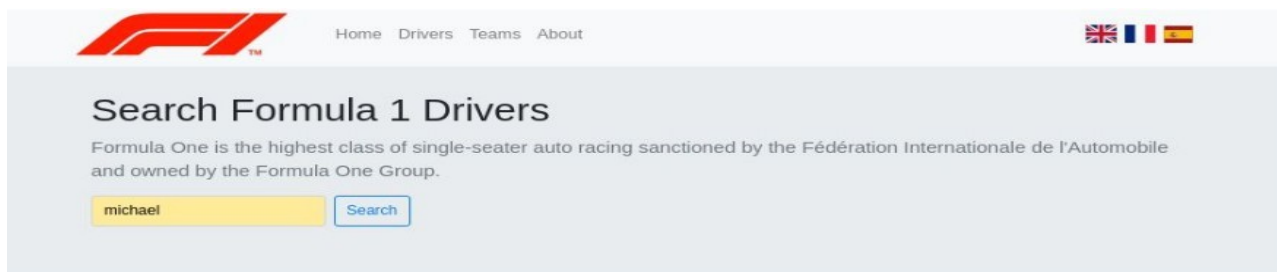
Веб апликацијата која е искористена за приказ на докеризација, достава и тестирање на истата претставува веб сајт на кој може да се пребаруваат и разгледаат разни информации за возачи или тимови на Формула 1, сите информации се достапни на француски, англиски и шпански јазик. На слика 1 е прикажан Номе page-от на оваа апликација. На слика 2 е прикажан изгледот на страницата која прикажува детали за еден возач, додека на слика 3 е прикажана страницата која корисникот ја гледа при пребарување на возачи.



Слика 1: Номе page



Слика 2: Детали за возач



Слика 3: Пребарување на возачи

Самата апликација нема своја база на податоци туку има серверска страна развиена во Spring Boot, во програмскиот јазик Java и изградена со помош на Gradle, таа всушност претставува API кое испраќа барања до dbpedia зависно од повиците кои ги добива. Барањата до dbpedia се испраќаат со помош на Sparql queries, кои ни овозможуваат и пребарување од различни ресурси, менување на јазикот на кој е прикажана содржината, филтрирање и сортирање. Серверската страна е достапна и за нова клиентска страна.

Клиентската страна е изработена со помош на Angular. Нејзината улога е приказ на информации како и праќање на повици до серверската страна зависно од акциите на корисникот.

Потребно е двете апликации да бидат спремни за достава, спакувани во форма да можат да бидат опслужени. Ова е извршено со помош на Docker контејнери, со цел полесно да може да се променат некои библиотеки доколку има потреба а од друга страна генерираните датотеки да не зависат од локалната околина каде што се извршуваат овие наредби. Наредбите за подготовка на клиентската страна се:

```
"sudo docker run --rm -v ${PWD}/Angular:/usr/src/app todorovikj/install_angular_modules"
```

Која се користи за инсталација на библиотеките потребни за опслужување на апликацијата, а втората наредба е самото пакување и подготовка на апликацијата:

```
„sudo docker run --rm -v ${PWD}/Angular:/usr/src/app todorovikj/build_angular_img"
```

Наредбата за пакување и подготовка на серверскиот дел од апликацијата е извршено со наредбата:

```
„sudo docker run --rm -u gradle -v ${PWD}/Backend_gradle:/home/gradle/project -w /home/gradle/project gradle gradle clean build“
```

Сите Docker images, користени во овој проект се јавно достапни на Docker hub!

CI/CD

Continuous integration претставува интеграција на работата од повеќе девелопери дури и неколку пати на ден. Односно спојување на така наречени гранки, на кои работеле различни девелопери или тимови, во една и воедно извршување на тестовите како и праќање на известување до девелоперите за резултатите од тестовите.

Continuous delivery означува дека гранката која е наменета за продукција е секогаш спремна за достава на продукција, односно дека сите тестирања се претходно извршени успешно.

Continuous deployments означува дека доставата се врши автоматски при одредени услови, истата не мора да е на продукција, туку може и да е на девелопмент околина како што тоа е случај во овој проект.

Jenkins

Jenkins е сервер базиран, „open-source“ (софтвер со отворен код, односно кодот е достапен за корисниците и можат да го сменат или реискористат во согласност со условите на користење) софтвер, кој е бесплатен. Тој е сервер за автоматизација, односно помага да се автоматизираат делови од развој на софтверот како подготовка за достава, тестирање и самата достава на софтверот. Jenkins поддржува алатки за контрола на верзијата и може да извршува проекти базирани на Maven, Gradle, како и произволни „shell“ скрипти. Објавен е под лиценца на MIT. Сите овие карактеристики го прават Jenkins погоден за искористување при „Continuous Integration / Continuous Delivery“.

Jenkins во овој проект е опслужуван во Docker контејнер. Овој контејнер има потреба од две мапирања на датотеки за да ги зачува неговите информации, тие треба да се креирани однапред на машината каде што се извршува самиот контејнер. За успешно извршување на задачите на Jenkins му требаат негови додатоци кои се достапни преку неговото мени, или може да се инсталираат преку Docker image-от, исто така потребна му е и Docker инсталација за да може да креира контејнери за своите задачи. Во овој проект искористен е Jenkins Blue додаток за подобра визуелизација на извршување на задачите како и испишување на логовите.

За целта на овој проект искористен е „CI/CD pipeline“ со помош на Jenkins, кој претставува „multi branch pipeline“, кој всушност е поврзан со репозиториумот на проектот на Github и за секоја гранка при секое прикачување на код извршува некакви наредби. За да може да се поврзе Jenkins со Github потребно е да имаме достапен ssh клуч за пристап со соодветни привилегии. Најчесто првин се извршуваат тестовите, така и во овој проект, при што

резултатот од тестовите е достапен за девелоперот на Github, а доколку сака може истиот да го види преку Jenkins. Додека Jenkins ги извршува задачите на Github е прикажан симбол за чекање.

Jenkins ги добива задачите кои треба да ги изврши преку Jenkinsfile кој е достапен во проектот. Во оваа датотека дефинирани се фази и чекори во тие фази. Првата фаза е извршување на дефинираните тестови на апликацијата и соопштување на резултатот до Github, на овој начин девелоперот има увид дали ново напишаниот код има направено несакани промени на нешто што веќе работело. До овој чекор успешно е постигнат continuous integration.

Следната фаза е фазата која што всушност ја доставува апликацијата на серверот, односно го прави напредокот во развојот и промените видливи и достапни. Оваа фаза се извршува само доколку гранката на која е прикачен кодот е develop, односно девелоперот сака да ја достави таа верзија на корисникот за тој да ја истестира нормално само ако претходната фаза завршила успешно (апликацијата успешно ги поминала тестовите). Во оваа фаза првин се подготвува за достава серверската апликација, а потоа клиентскиот дел од проектот, ова е извршено со помош на Docker, со помош на наредбите објаснети на почетокот. По подготвување на апликацијата за достава се извршува Ansible скрипта која ги доставува потребните датотеки и ги рестартира контејнерите за тест околината на серверот. Последните чекори ги бришат локално генерираните фајлови при подготовката за достава на серверската и клиентската апликација. Овие последни чекори се потребни затоа што во иднина кога некој ќе прикачи код повторно, нема пермисии да ги избрише овие локално креирани фајлови од страна на Jenkins. По оваа фаза постигнат е и continuous deployment. Приказот од извршување на овие задачи е прикажан на Слика 5, 6 и 7. Освен фазите и задачите во Jenkinsfile дефинирано е и време на извршување, што значи ако фазата не заврши за одредено време, таа ќе биде обележана како неуспешна и ќе биде прекината, со цел да се избегнат непрекинати циклуси од задачи. На слика 4 е прикажана Jenkinsfile датотеката:

```
stage('Deploy'){
    options {
        timeout(time: 10, unit: "MINUTES")
    }

    when {
        branch 'develop'
    }

    steps{
        // ADD my existing ssh key
        // sh 'eval "$(ssh-agent -s)"'
        // sh 'ssh-add ~/.ssh/id_rsa'

        // get backend ready for deploy
        sh "sudo docker run --rm -u gradle -v ${env.WORKSPACE}/Backend_gradle:/home/gradle/project -w /home/gradle/project gradle gradle clean build"

        // get angular ready for deploy
        sh "sudo docker run --rm -v ${env.WORKSPACE}/Angular:/usr/src/app todorovikj/install angular_modules"
        sh "sudo docker run --rm -v ${env.WORKSPACE}/Angular:/usr/src/app todorovikj/build angular_img"

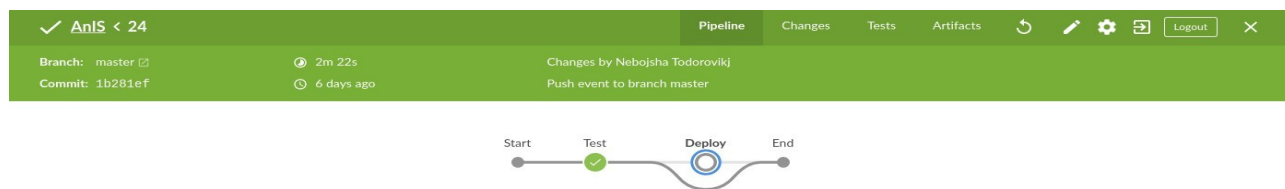
        // RUN ANSIBLE dev script
        dir(path: './playbooks') {
            sh "ansible-playbook dev-playbook.yml"
        }

        // clean backend files after deploy
        dir(path: './Backend_gradle') {
            sh "sudo rm -r build"
        }

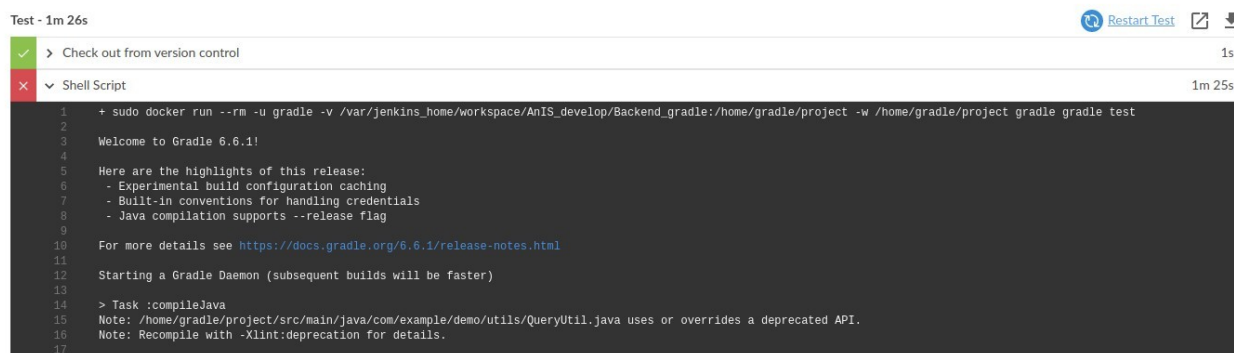
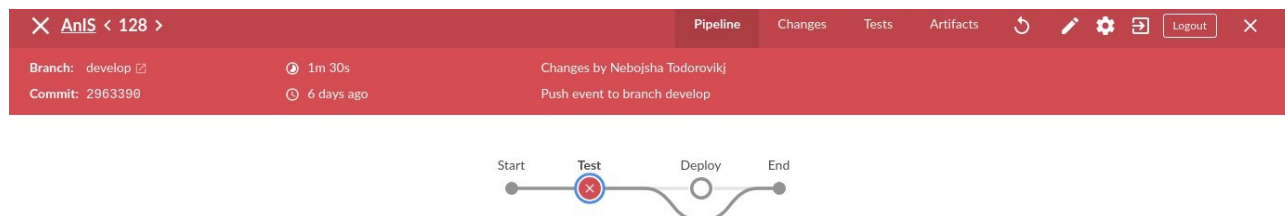
        // clean angular files after deploy
        dir(path: './Angular') {
            sh "sudo rm -r node_modules"
            sh "sudo rm -r dist"
        }
    }
}
```

Слика 4: Jenkinsfile

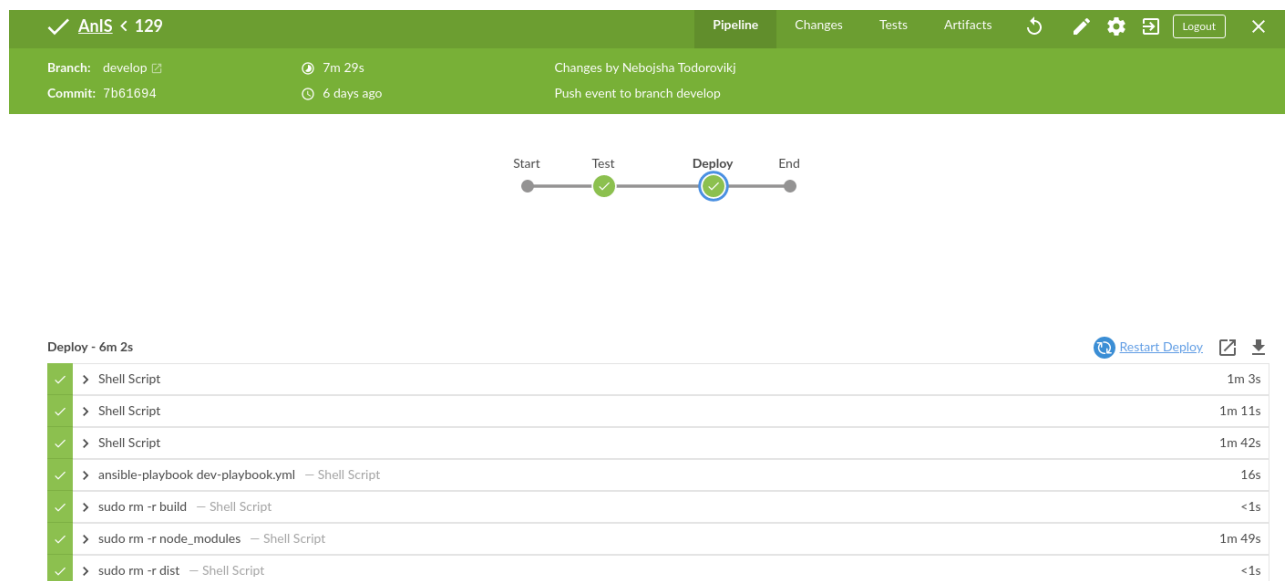
На следните неколку слики се прикажани резултати од извршување на задачите од страна на Jenkins.



Слика 5: Успешно извршени тестови, но гранката не е develop и нема достава



Слика 6: Неуспешно извршување на тестовите



Слика 7: Успешно извршување на тестови и достава

Ansible

Ansible е алатка за менаџирање на конфигурација. Со негова помош може да се менаџираат сервери, да се провери нивната конфигурација, да се проверат сервиси и процеси, кои фајлови ги содржат и со кои пермисии. Оваа алатка може да се користи и за достава на софтвер исто така, бидејќи на лесен начин може да се инсталираат софтверски пакети, копираат фајлови или извршат одредени наредби на серверот или било која друга машина која ја конфигурираме. Посебно добар за достава на софтвер е затоа што овозможува лесна оркестрација на настаните, односно корисникот може да дефинира во кој редослед и на која машина ќе се извршат наредбите.

Ansible користи YAML синтакса и се поврзува до машините кои што ги конфигурираме со помош на SSH. За негово користење потребно е локално да го инсталираме, да имаме дозвола за пристап до машината која ја конфигурираме, и секако при извршување на скриптата да тој да знае на која машина треба да се направат промените. Добра карактеристика на Ansible е тоа што на машината која што ја конфигурираме треба да има само SSH и Python, што најчесто доаѓа инсталирано со самиот оперативен систем, односно нема потреба од инсталација на дополнителен софтвер.

Во овој проект има две playbooks, кои ги извршува Ansible. Едната е за достава на продукција, која всушност и може да се искористи со помош на Jenkins и да се достигне continuous delivery. Во оваа скрипта се инсталираат сите потребни алатки на серверот, се креираат сите папки, се копираат сите потребни датотеки и се покренува или рестартираат контејнерите кои всушност одговараат на повиците кои пристигнуваат до продукциската околина. Погодно е тоа што за да се префрли апликацијата на друг сервер треба само да се изврши оваа скрипта на него и се што е потребно ќе функционира. Другиот playbook е всушност за достава на девелоп околината и е доста пократок затоа што само ги рестартира контејнерите коишто опслужуваат на develop околината. Потребно подобрување: Да се раздели скриптата за достава на продукција од скриптата за конфигурација на серверот. На Слика 8 е прикажан кодот кој ги инсталира потребните софтверски пакети, на слика 9 е прикажан кодот за креирање на папките кои му се потребни на Jenkins, и на слика 10 е прикажан кодот кој ги рестартира Docker контејнерите.

```
---
- name: Install required packages
  hosts: azure-server
  become: true
  tasks:
    - name: install docker
      apt: name=docker update_cache=true

    - name: install docker-compose
      get_url:
        url: "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-{{ansible_system}}-{{ansible_architecture}}"
        dest: /usr/local/bin/docker-compose
        mode: 0777

    - name: install npm
      apt: name=npm update_cache=true

    - name: install node.js
      apt: name=nodejs update_cache=true

    - name: install ansible
      apt: name=ansible update_cache=true

    - name: install angular cli
      command: npm install -g @angular/cli
```

Слика 8: Инсталација на потребни пакети

```

- name: Setup jenkins directories
  hosts: azure-server
  become: true
  tasks:
    - name: Create jenkins_home directory
      file:
        path: /var/jenkins_home
        state: directory
        mode: 1000

```

Слика 9: Креирање на папки за Jenkins

```

- name: Restart docker-compose containers
  hosts: azure-server
  become: true
  tasks:
    - name: Stop docker compose
      command: chdir=/home/AnIS docker-compose stop

    - name: Start docker compose
      command: chdir=/home/AnIS docker-compose up -d

```

Слика 10: Рестартирање на Docker контејнерите

NGINX

Според досегашните конфигурации и контејнери, доколку ги пуштиме истите на било која машина тие ќе функционираат. Но проблем на ваквото функционирање е пристапот до истите, поради тоа што тие се достапни на одредени порти. Ова значи дека доколку некој корисник сака да пристапи некаде ќе треба да го внесе URL-то за пристап до серверот а потоа :x каде што x е бројот на порта на која може да се пристапи до контејнерот. Ваков начин не е воопшто пријатен и пријателски за корисниците, дури ниту еден јавно достапен веб сајт не функционира на ваков начин. За таа цел искористен е истиот Docker image кој го ослужува NGINX, тој слуша на порта 80 (тука стандардно пристигнуваат барањата) и зависно од атрибутите во URL-то, ги пренасочува барањата до соодветниот контејнер. Поради овие пренасочувања и се искористени мрежи од Docker контејнери, тие се заслужни за комуникацијата помеѓу контејнерите. На овој контејнер му е мапирана конфигурација од локалната машина, за да може лесно да се промени истата и само со рестарт да работи со новата конфигурација.

Docker

Docker е збир од „platform as a service (PaaS)“ продукти кои користат виртуелизација на ниво на оперативниот систем за да испорачаат софтверски решенија во пакети наречени

контејнери. Тој е „open-source“ и помага при доставување на апликацијата во форма на контејнери.

Предности на Docker се тоа што тој е лесен за работа, не побарува многу ресурси и е брз. На брзината и додава тоа што тој се заснова на „copy-on-write“, односно се менува само она коешто сакаме да го промениме, а се друго останува исто, ова се случува во форма на слоеви. Оваа карактеристика придонесува и до тоа да се користење на Docker, девелоперот може лесно да ја премести својата апликација од едно место на друго (локално од кај него на сервер, или од сервер на сервер) и притоа да знае дека апликацијата ќе работи на истиот начин поради истата работна околина во која се извршува, така се намалува ризикот да се појави честиот проблем кога една апликација работи локално или во околина за тестирање, а не работи на продукција.

Docker работи по принцип на сервисно ориентирани и микросервисни архитектури. Препорака е секој контејнер да опслужува една единствена апликација или еден единствен процес. Затоа Docker е добар за дистрибуција на една апликација, бидејќи таа веќе е дистрибуирана низ повеќе контејнери иако можеби тие се на една иста машина, а воедно го олеснува и скалирањето, дебагирањето на апликациите.

Docker во овој проект е искористен за креирање контејнери, во кои има стартувано сервиси (production/development front-end & back-end, jenkins, nginx, контејнери кои ги креира и брише jenkins). Овозможува опслужување на повеќе сервиси на иста физичка машина, кои работат во своја околина. Контејнерите овозможуваат минимален down-time, бидејќи користат локална меморија од хостот и потребен е само рестарт за да биде достапната верзија. Има и техника за достава без down-time. Docker-compose е искористен за полесно менаџирање на контејнерите, бидејќи нивно рачно стартување бара многу информации (порти, мапирања на датотеки, мрежа...).

Споредба на контејнери во однос на виртуелни машини

Доста често се дискутира дека виртуелните машини се побезбедни од контејнерите поради поголемата изолација која ја обезбедуваат, како и тоа дека немаат ограничувања во однос на тоа што и на кој оперативен систем може да се опслужи. Јас немам искуство со виртуелните машини, но според прочитаното од разни извори овие информации се точни и се предност на виртуелните машини, но контејнерите овозможуваат други предности кои ги прават погодни за користење.

Минимум барања за виртуелна машина 2gb RAM (препорачливо е 4gb RAM, само за машината, но плус ќе ни треба за апликацијата). Минимум 8gb меморија (препорачливо е 20gb) за виртуелна машина. Како за споредба целиот Docker систем бара минимум 512 MB RAM за да функционира, и меморија зависно од контејнерите. Возможна е динамичка алокација и ограничувања. Овој проект има 6-7 контејнери кои се активни на машина со само 8GB RAM и 30GB меморија, со процесор со 2 јадра. За споредба со виртуелните машини веројатно би ни биле потребни минимум 14GB RAM и 140GB меморија.

Друга предност на контејнерите е тоа што полесно се врши верзионирање со контејнери, што го олеснува враќањето назад на претходна верзија при катастрофална грешка. Со виртуелни машини овој процес е покомплициран, поради нови верзии на библиотеките, кои не се

изолирани како што тоа е случај кај контејнерите, туку се инсталираат и ја менуваат состојбата на машината.

Уште една предност на контејнерите е тоа што полесно се покренува инстанца од контејнер отколку виртуелна машина и поради помалата комплексност истиот процес се извршува побрзо.

Понатамошна разработка на проектот

Постојат повеќе можни подобрувања на проектот како најважни би ги издвоил:

1. Consul за информации за контејнерите, бидејќи може да имаме повеќе инстанци од ист сервис, како и порта на која е достапен, затоа што повеќе сервиси не може да слушаат на иста порта и подобро е да биде автоматски доделена.
2. Динамичка конфигурација на NGINX – во овој проект конфигурацијата е статична, односно секоја промена треба да се направи рачно, постои опција Consul да прави динамичка конфигурација зависно од состојбата на контејнерите.
3. Blue-green deployment 0 downtime – Со користење на оваа техника би се овозможило критични сервиси да се секогаш достапни.
4. Опслужување во Kubernetes/Swarm кластер, полесно скалирање, односно додавање на нова машина во кластерот кој према девелоперот и корисникот се однесува како да е еден сервер.
5. Health checks & recovery

Користена литература

- Jenkins документација: <https://www.jenkins.io/doc/>
- Docker документација: <https://docs.docker.com>
- Docker-compose документација: <https://docs.docker.com/compose/>
- Docker Hub: <https://hub.docker.com/>
- Ansible – документација: <https://docs.ansible.com/>
- Spring boot – документација:
<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
- Angular – документација: <https://angular.io/docs>
- Baeldung: <https://www.baeldung.com/>
- The Docker Book Version: v18.09.2 (b71a7e7) James Turnbull
- Ansible: Up and Running Second Edition Lorin Hochstein and René Moser