UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Open Pipelines

propusă de
Daniel Ștefan Anechitoaie

Sesiunea: *Iulie, 2017*

Coordonator științific
Lector Dr. Cristian Traian Vidrașcu

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ

Open Pipelines

Daniel Ștefan Anechitoaie

Sesiunea: *Iulie, 2017*

Coordonator științific
Lector Dr. Cristian Traian Vidrașcu

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „Open Pipelines” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, _____
(data)

Absolvent *Daniel Ștefan Anechitoaie*

_____
(semnătura)

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „Open Pipelines”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, _____
(data)

Absolvent *Daniel Ștefan Anechitoaie*

_____
(semnătura)

Cuprins

Introducere

Motivație

Open Pipelines este o aplicație web concepută pentru a fi instalată și a rula pe un server Linux care se adresează tuturor echipelor ce lucrează la proiecte de orice mărime și complexitate într-un mediu colaborativ.

Aplicația dorește să vină atât în ajutorul persoanelor responsabile cu revizuirea codului, a calității acestuia și a urmării standardelor dar și în ajutorul fiecărui membru al echipei care contribuie la scrierea acestui cod. Acest lucru se face posibil prin rularea testelor automate și prin oferirea feedback-ului, în timp real, în legătură cu problemele detectate.

Idea dezvoltării acestei aplicații s-a născut din dorința de a automatiza anumite task-uri și a detecta probleme care sunt des întâlnite în procesul de revizuire a codului, lucru care ar duce la salvarea unei cantități considerabile de timp pe durata unui proiect.

Prin rularea automată de teste și prin validarea imediată a codului, în vederea respectării regulilor stabilite pe fiecare proiect, se va ajunge per total la scrierea unui cod mai de calitate, mai rapid, și la integrarea mai ușoară a membrilor noi în echipele și proiectele deja existente ce au deja propriile reguli prestabilite.

Alte soluții similare acestei aplicații sunt:

- BitBucket Pipelines (https://bitbucket.org/product/features/pipelines)
- Travis CI (https://travis-ci.org/)

Dezavantajele acestor soluții ar fi că nu sunt open source, uneori au un cost ridicat și nu permit rularea lor decât în cloud, ne putând fi instalate pe un server privat. De asemenea, de obicei aceste aplicații nu sunt compatibile cu mai multe servicii de găzduire a codului ci sunt făcute special pentru a funcționa cu un serviciu specific (de exemplu, BitBucket Pipelines nu poate fi utilizat decât de către utilizatorii serviciului BitBucket).

Context

În prezent, sunt dezvoltate din ce în ce mai multe aplicații și tool-uri software pentru analiza codului, detectarea erorilor, impunerea unui stil anume de scriere a codului, etc.

Acest lucru este foarte bun, deoarece duce la o calitate crescută a scrierii codului, consistență, urmarea standardelor și implicit o calitate mai ridicată a proiectelor.

Problema este că fiecare membru al echipei trebuie să-și instaleze și configureze aceste tool-uri la el în calculator și trebuie să aibă grija în momentul în care face push la cod că le-a și folosit și că totul este ok. Dificultatea folosirii programelor de acest gen crește în cazurile în care o persoană lucrează pe mai multe proiecte și poate aceste proiecte au fiecare reguli și standarde diferite. De asemenea se va pierde timp și în situațiile în care vor veni noi colegi pe proiect, ei trebuind ajutați să-și instaleze și ei aceste tool-uri și familiarizați cu ele.

Ce se întâmplă în situațiile în care se modifică ceva? Se adaugă un test nou, sau se modifică unele reguli de validare a codului? Se va pierde mult timp pentru a sincroniza toți membrii echipei și a face pe toata lumea să ajungă din nou la același set comun de reguli și teste.

Avantajul unei astfel de aplicații ce rulează pe web și în cloud este evident. Toți membrii echipei sunt „pe aceeași lungime de undă" urmând același set de reguli. Când se modifică ceva, această modificare este propagată instant și toată lumea urmează din nou același set de reguli și teste fără a se pierde timp deloc.

Un alt avantaj major al acestei aplicații față de alte aplicații similare este licența sub care se distribuie. Fiind gratis și open source, o companie care preferă să aibă tot mediul de lucru în spatele unui firewall sau VPN poate instala aplicația pe serverele ei private fără a oferi acces persoanelor din exterior. În unele medii enterprise acest lucru este absolut necesar.

Funcționalități

Autentificarea

Versiunea curentă permite autentificarea folosind unul din cele două servicii suportate, și anume BitBucket sau GitHub.

Listarea proiectelor GIT

Odată autentificat, utilizatorul are acces la pagina în care îi sunt listate toate proiectele sale găzduite pe serviciul prin care s-a făcut autentificarea.

Configurarea proiectelor

Utilizatorul are posibilitatea de a configura și de a activa serviciul Open Pipelines pentru oricare din proiectele sale.

De asemenea el poate controla și dacă rezultatul unui build să fie privat (vizibil doar de către el) sau public (vizibil de către oricine are link-ul build-ului respectiv).

Configurarea fișierului open_pipelines.yml

Utilizatorul își poate configura acest fișier și defini în el imaginea Docker care să fie utilizată în funcție de limbajul de programare folosit pe proiectul respectiv și în funcție de software-ul adițional necesar rulării testelor.

Odată definită imaginea Docker, utilizatorul are de asemenea posibilitatea de a-și defini comenzile ce trebuie executate odată cu fiecare push de cod.

Aici, se pot rula programe pentru testarea unitară a codului, programe pentru analiza codului pentru potențiale erori, programe pentru urmarea unor reguli sau stil predefinit de scriere a codului, etc.

Vizualizarea statusului unui build

În momentul în care utilizatorul va face push la cod, build-ul va porni automat rulând toate testele definite de către utilizator în fișierul open_pipelines.yml.

Statusul acestui build poate fi urmărit în timp real fie pe pagina aplicației Open Pipelines fie pe pagina serviciului folosit de către utilizator fiind marcat cu una din cele trei valori posibile, și anume: în progres, succes, eșuat.

Vizualizarea rezultatului unui build

Rezultatul unui build poate fi văzut pe pagina aplicației Open Pipelines fie în timp real fie după terminarea lui. Aici utilizatorul poate vedea pas cu pas fiecare comandă executată, împreună cu output-ul generat de ea cât și erorile detectate în caz că build-ul a eșuat.

Scalabilitate

Una dintre cele mai importante „abilități” ale acestei aplicații este scalabilitatea. Aplicația a fost concepută în așa fel încât poate face față și procesa virtual un număr nelimitat de cereri și build-uri.

Folosind Celery ca și coadă de așteptare (task queue) / coadă de sarcini (job queue) și Redis ca message broker care face legătura între procesele de tip worker, singura limitare ar fi hardware-ul pus la dispoziție, aplicația având abilitatea de a distribui load-ul între toate procesele de tip worker.

Contribuții

La realizarea acestui proiect am contribuit în totalitate, începând cu idea care s-a născut din nevoia de a automatiza anumite lucruri în ceea ce privește procesul de revizuire si analiză a codului, continuând cu faza de analiză și pana la implementarea, instalarea și configurarea pe un server de test și testarea produsului finit.

Procesul de dezvoltare al acestui proiect a constat în analiza soluțiilor existente, cercetarea și înțelegerea tehnologiilor necesare și utilizate.

Am început prin crearea unor schițe despre cum ar trebui să arate și să funcționeze aplicația și am continuat prin definirea structurii bazei de date și mai apoi implementarea unui prototip al aplicației pe care am continuat să-l extind cu modulele necesare.

În cadrul acestui proiect am avut ocazia și motivația de a studia și a mă familiariza cu metodele necesare implementării unei arhitecturi modulare, scalabile și securizată, împreună cu învățarea și înțelegerea modului de utilizare a software-ului folosit.

Câteva dintre contribuțiile semnificative pe care le aduce acest proiect ar fi:

- Este open source – deci poate servi ca un bun exemplu de aplicație de acest gen
- Este modular – deci poate fi extins ușor pentru a se adăuga suport pentru servicii noi cât și funcționalități adiționale
- Este scalabil – poate face față unui număr foarte mare de utilizatori, singura limitare fiind hardware-ul pus la dispoziție (numărul de servere care pot fi folosite de procesele de tip worker)
- Timp redus pentru procesul de revizuire și validare a codului – prin automatizarea acestui proces se poate salva o cantitate considerabilă de timp ușurând cu mult munca persoanelor responsabile de acest lucru
- Încurajează și ușurează lucrul în echipă – având posibilitatea de a stabili și impune unele standarde de scriere a codului pe care această aplicație le va valida automat

1. Tehnologii folosite

În acest capitol sunt prezentate tehnologiile, framework-urile, bibliotecile și serviciile folosite în dezvoltarea acestui proiect discutând câte puțin despre fiecare și avantajele oferite care au contribuit la decizia de a le folosi.

1.1. Linux

Linux[1], care uneori mai este întâlnit și sub numele de GNU/Linux, este o familie de sisteme de operare de tip Unix care folosesc nucleul Linux (în engleză kernel). Linux poate fi instalat pe o varietate largă de hardware, începând cu telefoane mobile, tablete, console video, continuând cu calculatoare personale până la super computere.

Linux este cunoscut în principal pentru utilizarea sa ca server ajungând să domine acest domeniu.

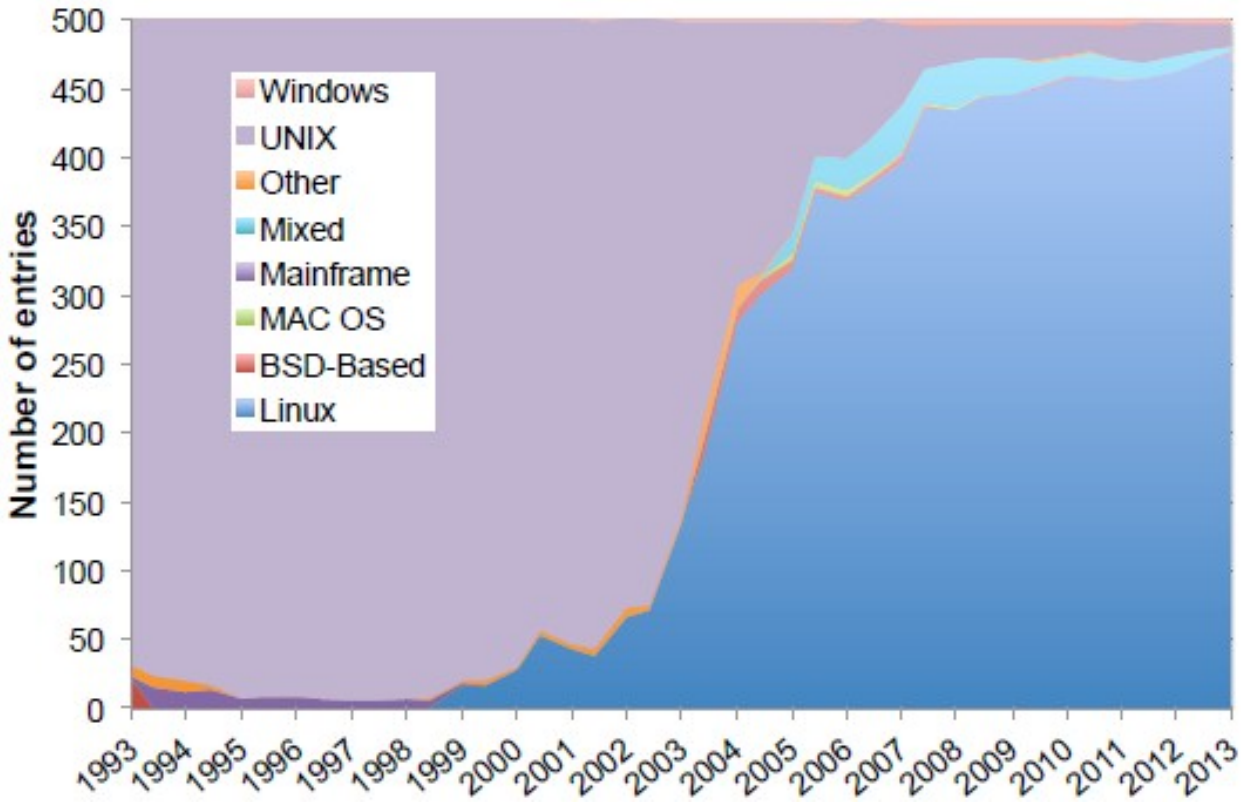

Figura 1: Ascensiunea Linux-ului în domeniul super computerelor
(imagine preluată de pe
https://www.theregister.co.uk/2013/08/02/linux_drives_supercomputing_and_maybe_clouds/)

1.2. Python

Python[2] este un limbaj de programare foarte popular, oferind posibilitatea programării structurate dar și orientate pe obiect și incluzând și elemente din paradigma funcțională. Este un limbaj de scripting, ceea ce înseamnă că este interpretat și nu compilat, economisind mult timp în procesul de dezvoltare

Python a fost creat în 1989 de programatorul olandez Guido van Rossum. Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfecționarea limbajul Python și implementarea de bază a acestuia, CPython, scrisă în C.

Popularitatea în creștere, dar și puterea limbajului de programare Python au dus la adoptarea sa ca limbaj principal de dezvoltare de către programatori specializați, de către companii ca Google, Yahoo și Facebook pentru programarea aplicațiilor web și chiar și la predarea limbajului în unele medii universitare. Din aceleași motive, multe sisteme bazate pe Unix, inclusiv Linux, BSD și Mac OS X includ din start interpretatorul CPython.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare.

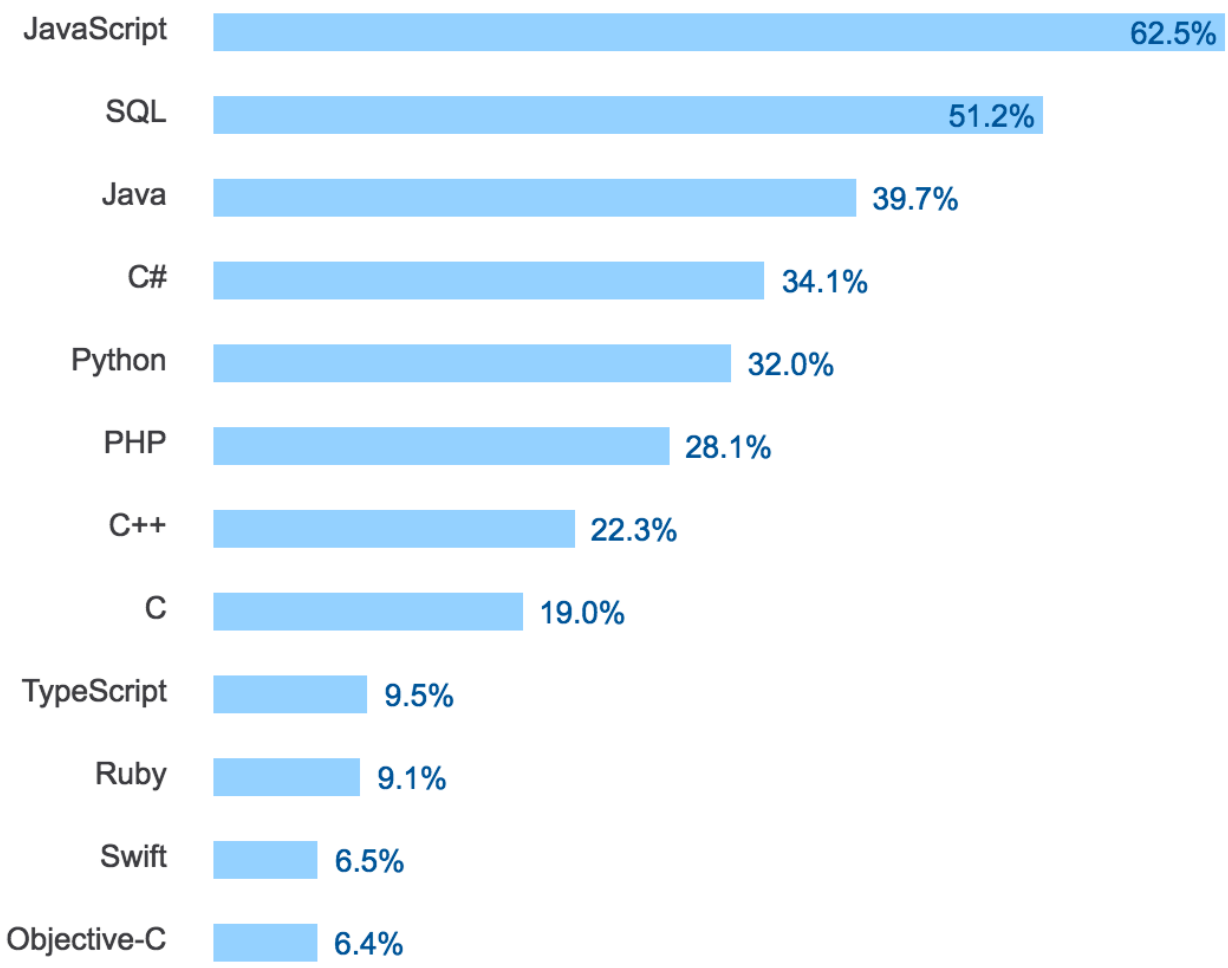

Figura 2: Topul celor mai populare limbaje de programare conform unei analize făcute de StackOverflow.com[3] (imagine preluată de pe https://insights.stackoverflow.com/survey/2017)

Conform unui sondaj făcut în Ianuarie 2017 de către site-ul StackOverflow.com PHP a fost depășit de către Python pentru prima data în ultimii cinci ani de zile, lucru care denotă popularitatea în creștere al acestui limbaj de programare.

1.3. Django

Django[4] este un framework gratuit și open source folosit în dezvoltarea aplicațiilor web pentru limbajul de programare Python care urmează modelul arhitectural MVC (Model-View-Controller).

Scopul principal al acestui framework este acela de a facilita crearea de aplicații web complexe care interacționează cu bazele de date punând accent pe modularitate și reutilizarea codului și ghidându-se după principiul „nu te repeta” (en. DRY – Don’t Repeat Yourself).

Conform paginii de prezentare de pe site-ul oficial al acestui framework unele din avantajele folosirii lui ar fi:

- Ridicol de rapid – a fost conceput să ajute ducerea unui proiect din faza de concept în faza de proiect complet cât mai rapid.
- „Cu bateriile incluse” – vine la pachet cu zeci de module adiționale care te ajută să rezolvi multe din problemele comune aplicațiilor web. Module pentru autentificare, administrare de conținut, securitate, etc.
- Securizat – securitatea este tratată cu cea mai mare seriozitate având parte de audituri periodice pentru a preveni probleme ca SQL injection, cross-site scripting, cross-site request forgery și clickjacking.
- Scalabil – unele dintre cele mai active și populare site-uri de pe internet folosesc Django și reușesc să facă față cu succes numărului ridicat de utilizatori.
- Versatil – poate fi folosit pentru tot felul de lucruri, de la aplicații de administrare a conținutului până la rețele sociale.

Câteva site-uri web bine cunoscute care utilizează Django sunt:

- Pinterest - https://www.pinterest.com/
- Instagram - https://www.instagram.com/
- The Washington Times - http://www.washingtontimes.com/
- Disqus - https://disqus.com/
- BitBucket - https://bitbucket.org/
- Mozilla - https://www.mozilla.org/en-US/
- National Geographic - http://www.nationalgeographic.com/

1.4. Celery

Celery[5] este o coadă de așteptare (task queue) / coadă de sarcini (job queue) asincronă, bazată pe transmiterea mesajelor distribuite. Scopul principal este procesarea în timp real a datelor. Unitățile de execuție, numite sarcini (task-uri), sunt executate simultan pe un singur sau mai multe servere de lucrători (workers).

Task-urile pot fi executate asincron (în fundal) sau în mod sincron (așteptând până când este gata).

Celery comunică prin mesaje, de obicei folosind un broker pentru a media între clienți și lucrători. Pentru a iniția un task clientul adaugă un mesaj la coadă, brokerul livrează acel mesaj unui lucrător.

Procesele lucrătorilor dedicați monitorizează în permanență cozile de sarcini pentru a efectua noi lucrări primite de la broker.

Celery este folosit în sistemele de producție pentru a procesa milioane de task-uri pe zi.

Celery este:

- Ușor de învățat și simplu de utilizat – Nu necesită fișiere de configurare. Are o comunitate activă și prietenoasă.
- Rapid – Un singur proces Celery poate procesa milioane de task-uri pe minut
- Flexibil – Aproape fiecare parte din Celery poate fi extinsă sau folosită independent

1.5. uWSGI

uWSGI[6] este un server de aplicații de mare performanță și care are un consum redus de resurse. Deși numărul de funcționalități oferite este destul de mare și în același timp mereu în creștere, este folosit cel mai des ca și server pentru aplicațiile Python făcând legătura între serverul web (NGINX, Apache, etc.) și aplicația scrisă în Python.

Figura 3: Fluxul datelor de la client la aplicația Python prin serverul web (NGINX) și serverul de aplicații (uWSGI)

1.6. NGINX

NGINX[7] este un server web care poate fi folosit ca reverse proxy, load balancer și HTTP cache.

NGINX folosește o abordare asincronă orientată către evenimente pentru a gestiona cererile, având o arhitectură modulară, la fel bazată pe evenimente, care poate oferi performanțe mai previzibile chiar și la sarcini mari.

Câteva dintre avantajele folosirii NGINX ca și server web sau reverse proxy:

- Abilitatea de a gestiona mai mult de 10.000 de conexiuni simultane cu un consum redus de memorie
- Manipularea fișierelor statice, a fișierelor index și a auto-indexării
- Reverse proxy cu caching
- Toleranță la defecte
- Suport pentru FastCGI, SCGI, uWSGI cu caching
- Suport pentru protocolul HTTP/2
- Compresie și decompresie GZip
- Limitarea conexiunilor concurente
- Suport pentru TLS/SSL

Conform analizei făcute de Netcraft în Noiembrie 2016, NGINX a fost găsit ca fiind al doilea în topul celor mai utilizate servere web folosit de site-urile „active”.

NGINX a fost scris cu scopul explicit de a depăși performanța serverului web Apache. În servirea fișierelor statice, NGINX utilizează considerabil mult mai puțină memorie decât Apache și poate gestiona aproximativ de patru ori mai multe cereri pe secundă. Această creștere a performanței are la bază costul unei flexibilități scăzute, cum ar fi abilitatea de a suprascrie setările de acces la sistem per fișier (Apache realizează acest lucru cu un fișier .htaccess, în timp ce NGINX nu are nici o astfel de funcționalitate încorporată).

Inițial, adăugarea de module terțe la NGINX necesita recompilarea aplicației de la sursă cu modulele compilate în mod static. Acest lucru a fost parțial depășit în versiunea 1.9.11, cu adăugarea încărcării dinamice a modulelor. Cu toate acestea, modulele trebuie să fie compilate în același timp cu NGINX, și nu toate modulele sunt compatibile cu acest sistem.

1.7. Redis

Redis[8] este un server de structuri de date stocate în memoria RAM. Poate fi folosit ca și bază de date non-relațională, cache sau message broker.

Câteva dintre particularitățile care îl fac să se diferențieze de alte aplicații de acest gen:

- Accesul la datele stocate este foarte rapid
- Toată baza de date este păstrată în memorie cu posibilitatea de a folosi discul pentru persistență fie prin salvarea periodică a datelor fie prin tinerea unui jurnal al tuturor comenzilor primite.
- Dispune de un set larg de tipuri de date: string-uri, hash-uri, liste, seturi, etc.
- Datele pot fi replicate asincron la un număr nelimitat de servere de tip „slave”.

În contextul acestui proiect Redis este folosit pe post de message broker pentru Celery.

1.8. MySQL / MariaDB

MySQL[9] este cel mai popular sistem de gestiune a bazelor de date relaționale, produs de compania suedeza MySQL AB și distribuit sub Licența Publică Generală GNU. Popularitatea sa ca SGBD pentru aplicațiile web este strâns legată de cea a PHP-ului care este adesea combinat cu MySQL și denumit Duo-ul Dinamic. În multe cărți de specialitate este precizat faptul că MySQL este mult mai ușor de învățat și folosit decât multe dintre restul aplicațiilor de gestiune a bazelor de date disponibile.

Pentru a administra bazele de date MySQL, se poate folosi modul linie de comandă sau, prin descărcare de pe internet, o interfață grafică cum ar fi: MySQL Administrator și MySQL Query Browser sau chiar aplicația gratuită, scrisă în PHP, phpMyAdmin.

MySQL poate fi rulat pe multe dintre platformele software existente: FreeBSD, GNU/Linux, Mac OS X, NetBSD, Solaris, SunOS, Windows 9x/NT/2000/XP/Vista.

Recent, din cauza unor controverse pornite de la faptul că MySQL a fost achiziționată de Sun și apoi de Oracle a luat naștere MariaDB[10] care este un derivat al MySQL dezvoltat de comunitatea open source.

MariaDB intenționează să mențină o compatibilitate ridicată cu MySQL, asigurând o capacitate de înlocuire „drop-in” cu echivalența binară a bibliotecii și potrivire exactă cu API-urile și comenzile MySQL. Dezvoltatorul său principal este Michael "Monty" Widenius, care este chiar unul dintre fondatorii MySQL.

Câteva dintre motivele pentru care cineva care încă folosește MySQL ar trebui sa migreze la MariaDB sunt:

- Dezvoltarea este mai deschisă
- Versiuni noi mai rapide și mai transparente în ceea ce privește securitatea
- Mai multe funcționalități de ultimă oră
- Mai multe motoare de stocare
- Performanță mai bună
- MariaDB continuă să crească în popularitate față de MySQL

1.9. Docker

Docker[11] este un proiect open-source care automatizează implementarea aplicațiilor în containerele software.

Docker oferă un strat suplimentar de abstractizare și automatizare a virtualizării la nivel de sistem de operare pe Windows și Linux. Docker utilizează caracteristicile de izolare a resurselor kernel-ului Linux pentru a permite "containerelor" independente să ruleze într-o singură instanță Linux evitând costurile generale de pornire și întreținere a mașinilor virtuale.

Într-un fel, Docker este ca o mașină virtuală dar, spre deosebire de o mașină virtuală, Docker permite aplicațiilor să utilizeze același kernel Linux ca sistemul pe care rulează și nu cere decât ca aplicațiile să fie livrate cu lucrurile care nu exista deja pe computerul gazdă. Acest lucru oferă un impuls semnificativ de performanță și reduce dimensiunea aplicației.

Dezvoltatorii folosesc programul Docker pentru a elimina problemele de genul "la mine în calculator merge" atunci când lucrează în colaborare cu colegii. Operatorii folosesc Docker pentru a rula și gestiona aplicațiile una lângă alta în containere izolate pentru a obține o densitate mai bună de calcul. Întreprinderile folosesc Docker pentru a pune la punct procedee agile de livrare a programelor pentru a livra noi funcționalități mai rapid, mai sigur și cu încredere atât pentru aplicațiile Linux cât și pentru Windows Server.

1.10. OAuth2

OAuth2[12] este un protocol de autentificare pentru delegarea accesului, folosit în mod obișnuit ca o modalitate prin care utilizatorii de internet pot acorda site-urilor web sau aplicațiilor accesul la informațiile lor de pe alte site-uri web, fără a le oferi parolele. Acest mecanism este utilizat de companii precum BitBucket, GitHub, Google, Facebook, Microsoft și Twitter pentru a permite utilizatorilor să facă schimb de informații despre conturile lor cu aplicații sau site-uri terțe.

În general, OAuth2 oferă clienților un "acces delegat securizat" la resursele serverului în numele unui proprietar de resurse. Acest protocol definește un proces pentru proprietarii de resurse prin care să autorizeze accesul unor terțe părți la resursele lor pe server fără să le împărtășească credențialele.

Conceput special pentru a lucra cu Hypertext Transfer Protocol (HTTP), OAuth2 permite în mod esențial emiterea de token-uri de acces unor clienți terță parte de către un server de autorizare, cu acordul proprietarului resursei. Partea terță utilizează apoi token-ul de acces pentru a accesa resursele protejate găzduite de serverul de resurse.

Roluri definite de acest protocol:

Aplicația terță (clientul)

Clientul este aplicația care încearcă să obțină acces la contul utilizatorului și care trebuie să obțină permisiunea utilizatorului înainte ca acesta să poată face acest lucru.

API-ul (serverul de resurse)

Serverul de resurse este serverul API folosit pentru a accesa informațiile utilizatorului.

Serverul de autorizare

Acesta este serverul care prezintă interfața în care utilizatorul aprobă sau respinge solicitarea. În implementările mai mici, acesta poate fi același server ca serverul API, dar implementările pe scară mai largă vor construi adesea acest lucru ca o componentă separată.

Utilizatorul (proprietarul resursei)

Proprietarul resursei este persoana care dă acces la o parte din contul său.

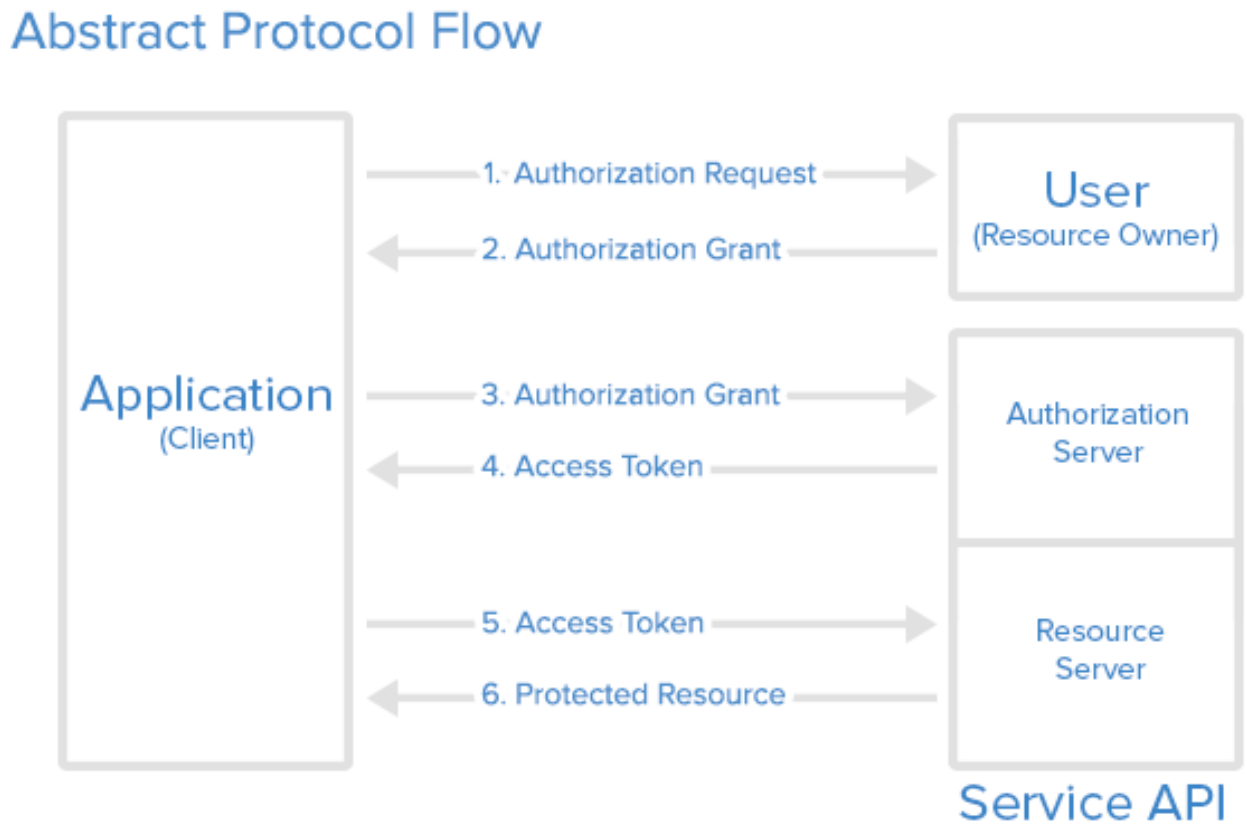

Figura 4: Diagramă funcționare OAuth2[13]
(imagine preluată de pe https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2)

1. Aplicația solicită autorizarea accesării resurselor serviciului respectiv de la utilizator
2. Dacă utilizatorul a autorizat solicitarea, aplicația primește un grant de autorizare
3. Aplicația solicită un token de acces de la serverul de autorizare (API) prezentând autentificarea identității proprii și grantul de autorizare primit la pasul 1
4. Dacă identitatea aplicației este autentificată și grantul de autorizare este valabil, serverul de autorizare (API) emite un token de acces la aplicație. Autorizația este completă.
5. Aplicația solicită resursa de la serverul de resurse (API) și prezintă tokenul de acces pentru autentificare

6. Dacă tokenul de acces este valabil, serverul de resurse (API) servește resursa aplicației

1.11. GIT

GIT[14] este unul dintre cele mai populare sisteme de versionare a codului. Este gratuit, open source, distribuit și conceput pentru a lucra cu orice tip de fișiere, de la proiecte foarte mici până la proiecte uriașe, într-un mod cat mai ușor și rapid.

GIT este ușor de învățat, consumă puține resurse oferind o foarte bună performanță, permițând gestionarea versiunilor multiple are fișierelor precum și asigurarea lucrului colaborativ asupra acestor fișiere. Fiecare modificare efectuată de către un utilizator, va fi înregistrată conținând atât diferențele noii versiuni, precum și autorul acestei noi versiuni.

Sistemele de acest gen, au fost concepute pentru a permite membrilor echipei să lucreze în paralel și să efectueze modificări pe același proiect, urmând ca aceste modificări să poate fi reunite într-o versiune următoare a proiectului.

Terminologie

- repository – componenta server ce conține informații privind ierarhia de fișiere și reviziile
- checkout – preluarea în mediul local a unei anumite revizii publicate pe server
- working copy – versiunea locala și activa a proiectului
- commit – cerere de publicare pe server a unor modificări
- pull – acțiunea de sincronizare și actualizare (update) a informațiilor locale cu cele de pe server
- push – acțiunea de sincronizare și actualizare a informațiilor de pe server cu cele locale
- conflict – apare atunci când mai mulți utilizatori vor să publice modificări aplicate acelorași fișiere din proiect fără ca sistemul de aplicare a versiunilor diferite să poată îmbina automat aceste modificări
- revert – revenirea la o versiune anterioară pe un anume fir de dezvoltare (branch)
- branch – ramuri secundare de dezvoltare a proiectului

Companii și proiecte majore care folosesc GIT:

- Google
- Facebook
- Microsoft
- Android
- Linux

- PostgreSQL

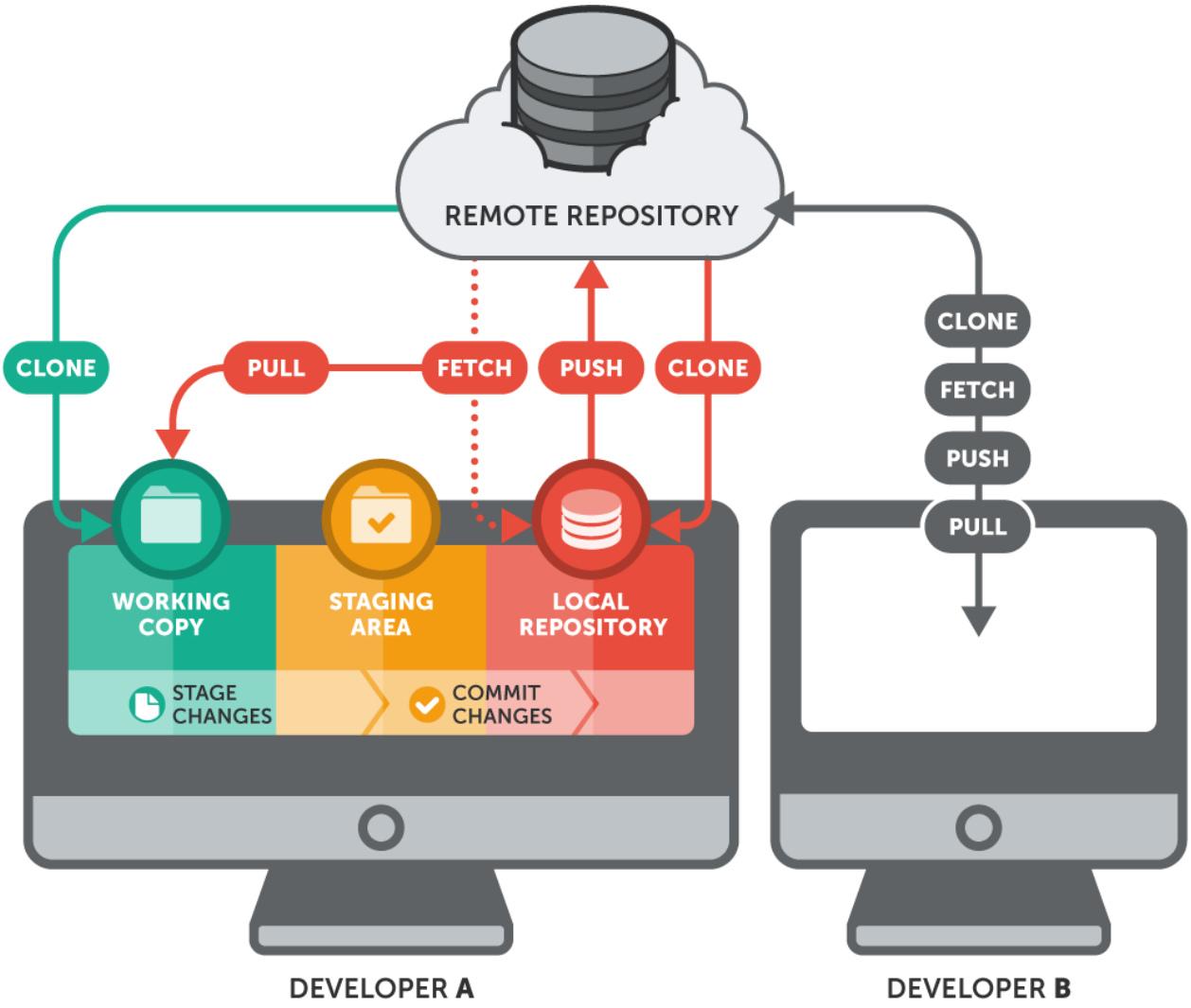

Figura 5: Diagramă funcționare GIT[15]
(imagine preluată de pe https://www.git-tower.com/learn/git/ebook)

1.12. BitBucket

BitBucket[16] este un serviciu bazat pe web, care este deținut de Atlassian și care este folosit pentru găzduirea codului sursă și a proiectelor de dezvoltare care utilizează sisteme de control al versiunii cum ar fi Mercurial (de la lansare) sau GIT (din Octombrie 2011).

Oferă atât planuri comerciale, cât și conturi gratuite. Acesta oferă conturi gratuite cu un număr nelimitat de proiecte private (care pot avea până la cinci utilizatori în cazul conturilor gratuite).

BitBucket se integrează cu alte aplicații Atlassian precum Jira, HipChat, Confluence și Bamboo.

Câteva motive pentru care BitBucket este un serviciu de preferat în favoarea altor site-uri concurente:

- Code review – interfața web oferita de BitBucket oferă posibilitatea de a vedea și analiza modificările făcute de către ceilalți membrii ai echipei putând lăsa comentarii și aproba/respinge modificările făcute de ei.
- Proiecte private nelimitate – disponibile chiar și pe conturile gratuite
- Interfață prietenoasă și ușor de folosit
- Posibilitatea de a importa codul automat din alte sisteme
- Discuții in-line – se poate discuta și lasă comentarii pe fiecare linie de cod din fișierele proiectului

1.13. GitHub

La fel ca și BitBucket, GitHub[17] este un serviciu de găzduire pentru proiecte de dezvoltare a software-ului care utilizează sistemul de control al versiunilor GIT.

GitHub oferă planuri tarifare pentru proiecte private, și conturi gratuite pentru proiecte open source.

În contrast cu BitBucket, site-ul GitHub oferă funcționalități asemănătoare unei rețele sociale unde utilizatorii săi au posibilitatea de a urmări activitatea altor proiecte, utilizatori de interes, etc.

Câteva dintre avantajele folosirii GitHub:

- Încurajarea contribuirii la proiecte open source – acest lucru se datorează în principal felului cum a fost gândit și structurat acest serviciu și anume asemănător unei rețele sociale.
- Posibilitatea de a găsi ușor proiecte de interes
- Documentație excelentă – secțiunea de ghiduri și articole este plină și actualizată periodic cu informații relevante pentru cineva care lucrează cu GIT

2. Analiza soluției

Dezvoltarea aplicației Open Pipelines a pornit din nevoia de a avea o modalitate de a automatiza procesul de verificare a codului scris de membrii unei echipe din cadrul unui proiect.

Pentru a înțelege mai bine circumstanțele și nevoia care a dus inițial la idea și mai apoi și la dezvoltarea acestui proiect, voi explica mai jos cam care sunt procedurile de lucru folosite zi de zi și unde și cum ar contribui acesta aplicație nu doar la reducerea timpului necesar dezvoltării unui proiect ci și la o mai bună calitate a codului.

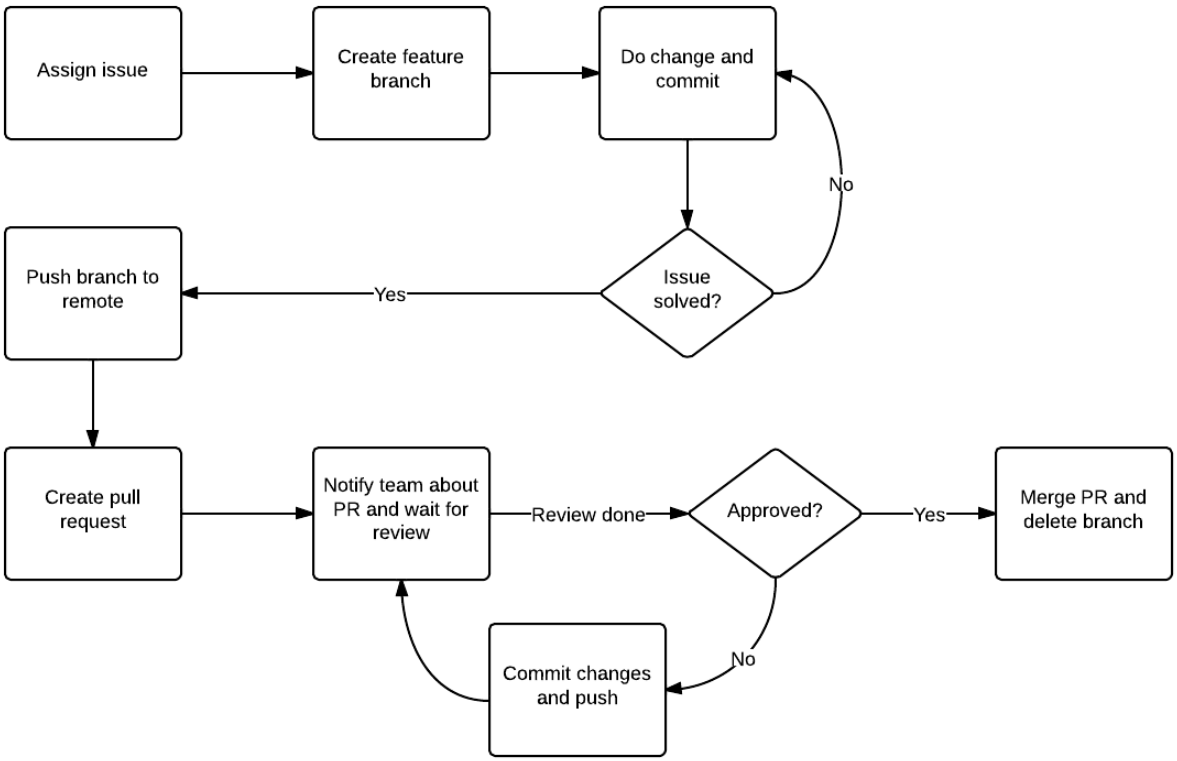

Figura 6: Diagramă mod de lucru cu GIT și pull requests[18]
(imagine preluată de pe http://slidedeck.io/gish/pull-request-presentation)

În momentul în care unui membru al echipei i se asignează un task, acesta începe prin a clona local ultima versiune a codului de pe repo-ul remote de GIT.

Va continua prin a crea un branch nou local, în care va implementa funcționalitatea cerută urmând ca pentru fiecare modificare efectuată să facă un commit local.

Odată ce consideră că a terminat de implementat cerințele acestui task, va face un push al codului din feature branch-ul lui local în cel al repo-ului remote unde interfața web a serviciului respectiv (BitBucket sau GitHub) îi va oferi posibilitatea de a crea un pull request.

Pull request-ul nu este nimic altceva decât reprezentarea grafică (care este specifică serviciului ales și modul în care este afișată poate sa difere de la serviciu la serviciu neexistând un standard decis în acesta privință) a intenției persoanei respective de a integra modificările făcute în branch-ul principal.

Odată creat acest pull request rămâne deschis până cei responsabili cu revizuirea codului respectiv, și aprobarea sau declinarea lui, iau o decizie.

Pe parcursul procesului de revizuire a codului se pot vizualiza toate fișierele modificate împreună cu modificările făcute de către cel care a lucrat la acest task și de asemenea se pot lasă comentarii la nivel de linie de cod dacă se observă ceva greșit, programatorul având ocazia să corecteze aceste greșeli sau probleme semnalate.

După ce se termină procesul de revizuire a codului și toți cei implicați au aprobat aceste modificări de cod, pull request-ul este integrat (merged) în branch-ul principal.

În acest moment echipa de testare poate începe să testeze task-ul și la rândul lor îi pot da o aprobare finală sau îl pot întoarce înapoi, caz în care autorul acestui task va trebui să facă din nou un alt pull request în care să adreseze comentariile primite de la echipa de testare.

Locul unde aplicația Open Pipelines își are locul este la pasul de revizuire și validare a codului împreună cu interacțiunea dintre persoanele desemnate cu revizuirea codului și discuțiile efectuate pe pull request.

În momentul în care pull request-ul este creat aplicația Open Pipelines va interveni și va face automat anumite validări (care vor fi definite în prealabil) putând să marcheze la rândul ei acest pull request ca fiind valid sau nu și de asemenea să semnaleze și problemele detectate.

Automatizarea acestui proces duce nu doar la salvarea timpului consumat ci și la detectarea unor erori care e posibil să fie trecute cu vederea de către o persoana în anumite circumstanțe.

Echipa ce se ocupă cu revizuirea codului va interveni abia după ce aplicația validează pull request-ul și îl aproba, urmând ca ei doar să verifice lucrurile care nu se pot automatiza.

3. Proiectare

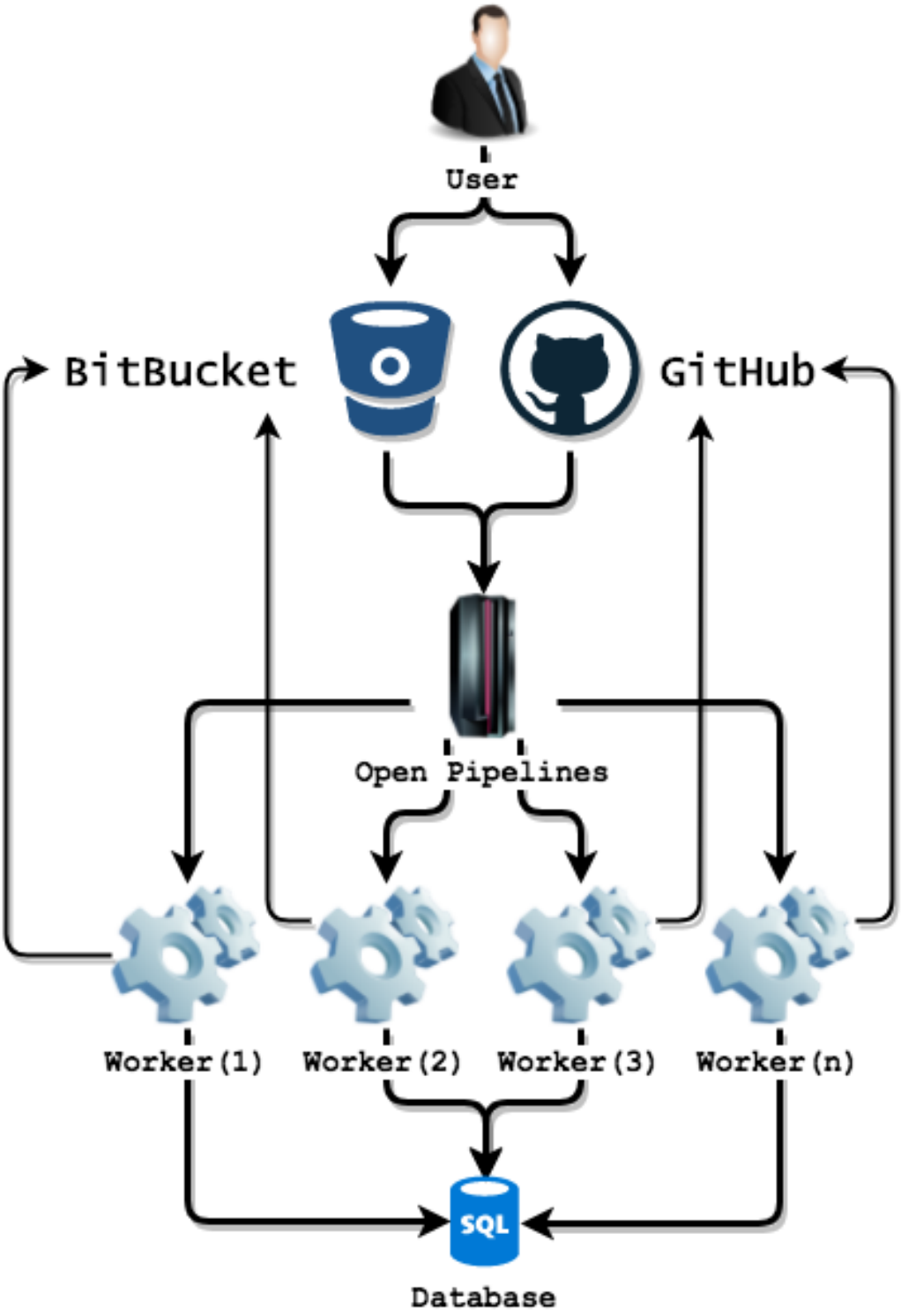

Figura 7: Arhitectura aplicației

Proiectul Open Pipelines este o aplicație ce rulează în totalitate pe server neavând nevoie de o componentă specială pentru partea de client.

Din punct de vedere al clientului interacțiunea cu această aplicație (pentru a o configura sau a accesa pagina cu rezultatele unui build) se face prin intermediul oricărui browser web cum ar fi Chrome, Firefox, Safari, Internet Explorer, etc.

Aplicația este structurată folosind modelul arhitectural MVT (Model-View-Template)

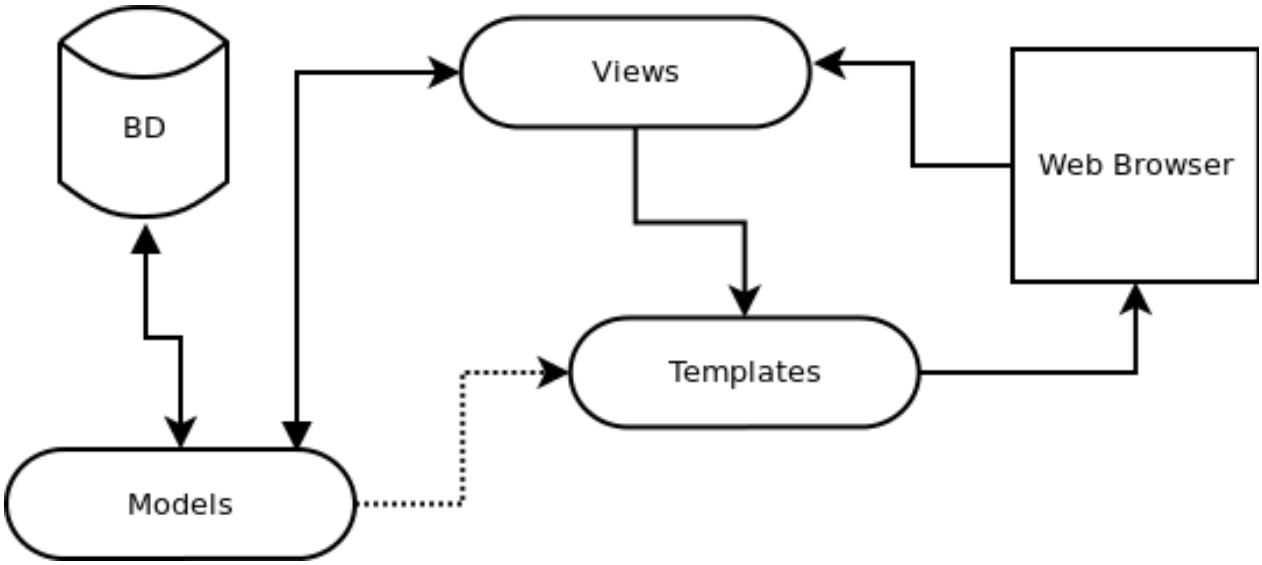

Figura 8: Modelul arhitectural al aplicației[19]
(imagine preluata de pe http://scipion.cnb.csic.es/old-docs/bin/view/TWiki/IntroductionDjango)

Pentru a returna un răspuns aplicația urmează pașii următori:

1. Când un utilizator inițiază o cerere din browser către o pagina servita de către aplicație aceasta cerere este preluata de către un view (echivalentul controller-ului în modelul MVC)
2. View-ul interacționează cu modelul pentru a returna datele necesare
3. Modelul este responsabil cu returnarea datelor necesare de la baza de date
4. Când view-ul a primit înapoi de la model toate datele necesare acesta le trimite mai departe template-ului
5. Template-ul formează datele după structura definită și returnează răspunsul înapoi la browser

Modelele în cadrul aplicației Open Pipelines

Pentru versiunea curentă a aplicației a fost nevoie să definim următoarele modele:

User

```
class User(models.Model):
    uuid             = models.UUIDField(default=uuid4, primary_key=True)
    username         = models.CharField(max_length=255, db_index=True)
    display_name     = models.CharField(max_length=255)
    service_id       = models.CharField(max_length=255)
    service_username = models.CharField(max_length=255)
    service_atoken   = models.CharField(max_length=255)
    service_rtoken   = models.CharField(max_length=255, null=True, blank=True)
    service_etoken   = models.DateTimeField(null=True, blank=True)
    last_login       = models.DateTimeField(auto_now_add=True)
```

*Figura 9: Modelul **User***

Acest model are definite următoarele câmpuri (care mai apoi sunt reprezentate ca și coloane în tabela users din baza de date):

- username – folosit pentru a ține numele utilizatorului ce se autentifică în aplicație. Se creează automat odată cu autentificarea lui folosind unul din cele două servicii (BitBucket sau GitHub)
- display_name – numele complet al utilizatorului obținut prin interogarea API-ului serviciului cu care s-a făcut autentificarea. Acest câmp se actualizează la fiecare autentificare.
- service_id - ID-ul serviciului cu care s-a creat acest cont (BitBucket sau GitHub)
- service_username – numele de utilizator asociat serviciului cu care s-a autentificat utilizatorul
- service_atoken – token-ul de acces generat de serviciul respectiv folosit de către aplicație pentru a accesa API-ul
- service_rtoken – token-ul folosit pentru a genera un token nou de acces când acesta expiră
- service_etoken – timpul după care token-ul de acces este considerat expirat și se cere generarea unui nou token
- last_login – data ultimei autentificări în aplicația Open Pipelines a utilizatorului curent

Repo

```
28
29  class Repo(models.Model):
30      uuid    = models.UUIDField(default=uuid4, primary_key=True)
31      user    = models.ForeignKey(User)
32      path    = models.CharField(max_length=255, db_index=True)
33      enabled = models.BooleanField()
34      public  = models.BooleanField()
35
```

*Figura 10: Modelul **Repo***

Acest model este folosit pentru a salva datele specifice unui proiect GIT configurat de către utilizator și are definite următoarele câmpuri în baza de date:

- user – legătura către modelul User și reprezintă utilizatorul care deține acest proiect (GIT repo)
- path – porțiune de URL care reprezintă calea către repo-ul utilizatorului. Folosită pentru cererile către API-ul serviciilor respective.
- enabled – daca aplicația Open Pipelines să fie activă sau nu pentru proiectul curent
- public – dacă rezultatele build-urilor efectuate pentru proiectul curent să fie publice sau nu

Build

```
37  class Build(models.Model):
38      uuid                = models.UUIDField(default=uuid4, primary_key=True)
39      repo                = models.ForeignKey(Repo)
40      docker_image        = models.CharField(max_length=255, null=True, blank=True)
41      ref                 = models.CharField(max_length=255)
42      commit              = models.CharField(max_length=255)
43      message             = models.CharField(max_length=255)
44      author_username     = models.CharField(max_length=255)
45      author_display_name = models.CharField(max_length=255)
46      status              = models.CharField(max_length=255)
47      output              = models.TextField()
48      datetime_start      = models.DateTimeField(null=True, blank=True)
49      datetime_end        = models.DateTimeField(null=True, blank=True)
```

*Figura 11: Modelul **Build***

Modelul ce reprezintă un build asociat unui proiect și care conține următoarele câmpuri în baza de date:

- repo – legătura către modelul Repo și reprezintă proiectul căreia îi aparține acest build
- docker_image – imaginea Docker folosita în build-ul curent
- ref – referința GIT folosită în build-ul curent (branch-ul sau tag-ul folosit)
- commit – commit-ul build-ului curent
- message – mesajul adăugat de către user pe commit-ul care a pornit acest build
- author_username – numele utilizatorului care a făcut acest commit. Este folosit pentru a genera un link către pagina lui de profil de pe serviciul respectiv.
- author_display_name – numele care va fi afișat pe pagină în referință la userul curent (în mod normal acesta va fi numele lui complet)
- status – statusul build-ului. Poate avea una din cele trei valori suportate: în progres, terminat cu succes sau eșuat
- output – output-ul generat de către build. Acest câmp va conține pas cu pas mesajele afișate de către imaginea Docker pe parcursul build-ului împreună cu rezultatul final care poate fi mesajul de eroare ce a cauzat eșuarea build-ului sau mesajul de succes.
- datetime_start – timpul când a pornit acest build
- datetime_end – timpul la care s-a terminat acest build

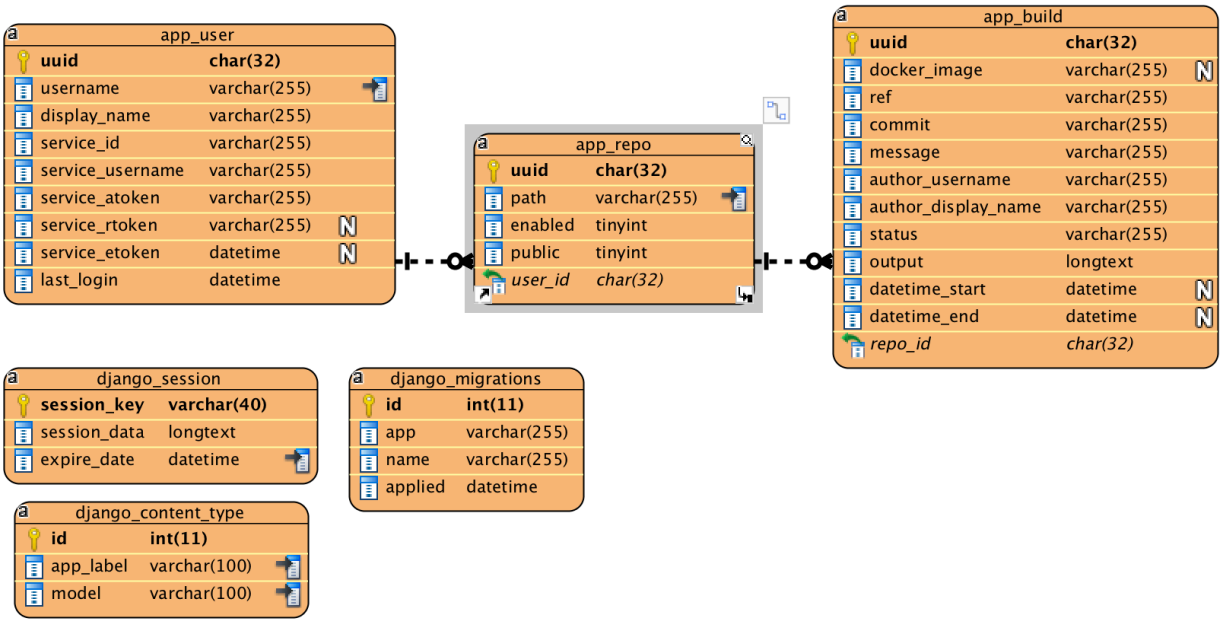

Figura 12: Structura bazei de date generată pe baza modelelor

View-urile în cadrul aplicației Open Pipelines

Aplicația expune următoarele view-uri:

Index

```
28      # Index
29      url(r'^$',
30          view = IndexView.as_view(),
31          name = "index"
32      ),
33
```

*Figura 13: View-ul **Index***

Este asociat următorului URL „/” și folosit pentru a afișa prima pagină a aplicației dacă utilizatorul este autentificat sau pentru redirecționarea lui către pagina de autentificare în caz contrar.

Repos

```
34      # Repos
35      url(r'^repos\.json$',
36          view = ReposJsonView.as_view(),
37          name = "repos_json"
38      ),
39
```

*Figura 14: View-ul **Repos***

Este asociat URL-ului „/repos.json” și este folosit pentru a returna lista de proiecte, în format JSON, pe care le are utilizatorul curent.

Repo by Path

```
40      # Repo by Path
41      url(r'^repos\/(.+)\.json$',
42          view = RepoByPathJsonView.as_view(),
43          name = "repo_by_path_json"
44      ),
45
```

*Figura 15: View-ul **Repo by path***

Este asociat URL-ului „/repos/<username>/<repo_name>.json” și este folosit pentru a returna informații despre un proiect anume. Răspunsul este returnat în format JSON.

Build by UUID

```
46      # Build by UUID
47      url(r'^builds\/([0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{4}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12})$',
48          view = BuildByUUIDView.as_view(),
49          name = "build_by_uuid"
50      ),
51
```

*Figura 16: View-ul **Build by UUID***

Este asociat URL-urlui „/builds/<UUID>” și este folosit pentru a afișa în format HTML pagina cu rezultatele unui build. Pe aceasta pagina utilizatorul are posibilitatea de a vedea în timp real (prin intermediul cererilor către server de tip AJAX) progresul build-ului curent împreună cu mesajele afișate de către acesta. Tot pe această pagină utilizatorul are posibilitatea și de a reporni acest build în caz că dorește acest lucru.

Build by UUID (JSON)

```
52      # Build by UUID JSON
53      url(r'^builds\/([0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{4}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12})\.json$',
54          view = BuildByUUIDJsonView.as_view(),
55          name = "build_by_uuid_json"
56      ),
57
```

*Figura 17: View-ul **Build by UUID JSON***

Este asociat URL-ului „/builds/<UUID>.json” și este folosit pentru a returna în format JSON răspunsul cererilor AJAX generate de către o persoana ce se află pe pagina descrisa mai sus.

Acest view returnează informații despre build-ul cerut, informații cum ar fi statusul, autorul, progresul build-ului, mesajele generate de către build, etc.

Webhook

```
58      # Webhook
59      url(r'^webhook\/([0-9A-Za-z]{8}-[0-9A-Za-z]{4}-[0-9A-Za-z]{4}-[0-9A-Za-z]{4}-[0-9A-Za-z]{12})$',
60          view = WebhookByUUIDView.as_view(),
61          name = "webhook_by_uuid"
62      ),
63
```

*Figura 18: View-ul **Webhook***

Este asociat URL-ului „/webhook/<UUID>” și este responsabil cu primirea notificărilor de la serviciile configurate (BitBucket sau GitHub) generate de către utilizatorii care fac push la cod.

Acest view primește datele în formatul specific serviciului respectiv și mai apoi „decodifica” aceste informații folosind unul din modulele implementate.

După validarea datelor se creează un task care se trimite la Celery care este responsabil la rândul lui mai departe să asigneze task-ul la unul din procesele de tip worker disponibile.

Login

```
64      # Login
65      url(r'^login$',
66          view = LoginView.as_view(),
67          name = "login"
68      ),
69
```

*Figura 19: View-ul **Login***

Este asociat URL-ului „/login” și reprezintă pagina de autentificare unde utilizatorul poate alege cu ce serviciu dorește să se autentifice în aplicația Open Pipelines.

Login With Service

```
70      # Login - With Service
71      url(r'^login\/(.+)$',
72          view = LoginWithServiceView.as_view(),
73          name = "login_with_service"
74      ),
75
```

*Figura 20: View-ul **Login With Service***

Este asociat URL-ului „/login/<ServiceID>” și se ocupă cu pașii necesari autentificării folosind protocolul OAuth2 pentru serviciul selectat la pasul anterior.

În momentul de față aplicația are suport pentru două servicii BitBucket și GitHub. La accesarea acestui URL aplicația determină serviciul curent din paramentul trimis în URL și mai apoi inițializează și folosește modulul specific serviciului ales.

Logout

```
76      # Logout
77      url(r'^logout$',
78          view = LogoutView.as_view(),
79          name = "logout"
80      ),
```

*Figura 21: View-ul **Logout***

Este asociat URL-ului „/logout” și este ultimul view expus de către aplicație.

În momentul accesării acestui URL sesiunea curentă este ștearsă și utilizatorul este deconectat de la aplicație.

Toata setările și proiectele lui nu sunt afectate și vor fi disponibile din nou după o nouă autentificare. Doar sesiunea curenta este ștearsă.

Decoratori

Ce este un decorator?

Un decorator este numele folosit pentru un model de design software. Decoratorii modifică dinamic funcționalitatea unei funcții, a unei metode sau a unei clase, fără a fi nevoiți să utilizeze direct subclasele sau să schimbe codul sursă al funcției care este decorată.

Ce este un decorator în Python?

„Decoratorii" despre care vorbim în legătură cu limbajul de programare Python nu sunt exact același lucru ca modelul de design software descris mai sus. Un decorator Python este o schimbare specifică a sintaxei Python care ne permite să modificăm mai convenabil funcțiile și metodele (și eventual clasele într-o versiune viitoare). Aceasta ajută la scrierea aplicațiilor într-un mod mai ușor de citit, dar și alte utilizări.

```
@classmethod
def foo (arg1, arg2):
    ....
```

Figura 22: Exemplu decorator în Python
(imagine preluata de pe https://wiki.python.org/moin/PythonDecorators#What_is_a_Decorator)

Pentru a securiza aplicația și accesul la anumite view-uri aplicația Open Pipelines a definit următorii decoratori:

login_required

Acest decorator este responsabil cu validarea request-urilor primite de către view-urile pe care este folosit și verifică daca utilizatorul este autentificat și are acces la resursa cerută sau nu.

În caz în care utilizatorul nu are acces, acest decorator generează un răspuns de tip redirect către pagina de autentificare.

login_required_json

Ca și decoratorul anterior, acesta este responsabil pentru verificarea dacă request-ul primit este de la un utilizator valid, autentificat și cu acces la resursa curentă sau nu.

Diferența între acest decorator și cel anterior este că acesta a fost conceput special pentru răspunsuri de tip JSON unde ar fi apărut probleme daca am fi returnat același răspuns de redirecționare către pagina de autentificare. În schimb, acest decorator returnează un răspuns în format JSON cu un mesaj aferent erorii întâlnite.

Template-urile în cadrul aplicației Open Pipelines

Pentru că majoritatea funcționalității oferită de această aplicație se concentrează pe partea de server și pe procesele ce rulează în background, nu s-au folosit un număr mare de template-uri.

Cele câteva template-uri folosite ar fi pentru:

- pagina de autentificare
- pagina de listare a proiectelor
- pagina de configurare a unui proiect
- pagina de afișare a rezultatului unui build

Fluxul de lucru al aplicației Open Pipelines

În continuare vom încerca să urmărim fluxul de lucru al aplicației prin evenimentele generate atât de către acțiunile unui utilizator cât și de către cele generate de serviciile integrate.

Un utilizator al aplicației face push la cod către serverul serviciului respectiv

1. În acest moment serviciul folosit (BitBucket sau GitHub) va emite un eveniment de tip push, cu datele aferente, în format JSON, către URL-ul specificat în pagina de configurare.
2. Aplicația citește datele primite, și identifică de la ce serviciu a venit acest request.
3. Dacă serviciul este recunoscut se inițializează modulul responsabil cu acest serviciu și datele sunt interpretate după formatul specific serviciului curent.
4. Se creează o nouă instanță a modelului Build care se populează cu datele extrase din cererea primită de la serviciul respectiv
5. Statusul build-ului curent este setat ca fiind „pending” (în așteptare)
6. Se trimite ID-ul acestui build nou creat către Celery
7. Celery asignează build-ul unui proces de tip worker în funcție de disponibilitatea acestora
8. Pe baza ID-ului primit worker-ul încarcă datele necesare din baza de date
9. Statusul build-ului este setat pe „în progres”
10. Se face un request către API-ul serviciului respectiv pentru a returna conținutul fișierului open_pipleines.yml pentru versiunea curentă de cod care a pornit acest build
11. Se interpretează conținutul fișierului și se generează lista de comenzi ce trebuie executată
12. Se pornește o nouă instanța Docker cu imaginea și dependințele definite în fișierul open_pipelines.yml
13. Se execută comenzile în interiorul acestei imagini Docker
14. Se interpretează linie cu linie mesajele generate de către instanța Docker și se salvează în baza de date
15. În funcție de rezultatul execuției listei de comenzi se actualizează statusul build-ului în baza de date ca fiind „succes” sau „eșuat”
16. Folosind API-ul se actualizează statusul build-ului și pe pagina serviciului respectiv

Un utilizator al aplicației accesează pagina cu rezultatele unui build

1. Aplicația verifică dacă utilizatorul care accesează build-ul curent are acces la această pagină sau nu. (Dacă build-ul este public, sau dacă este privat și utilizatorul curent este deținătorul proiectului)
2. Se afișează informațiile aferente build-ului curent. Informații cum ar fi data la care a început, durata, statusul, autorul acestui push, branch-ul de pe care s-a făcut build-ul, commit-ul, mesajele generate de către build, dacă încă este în progres sau dacă s-a terminat cu eroare sau succes
3. Dacă build-ul este în progres atunci se pornesc o serie de cereri AJAX la un interval predefinit care vor actualiza statusul build-ului împreună cu mesajele generate de aceasta

4. Ghid de instalare

Prezentul ghid acoperă pașii necesari instalării și configurării aplicației Open Pipelines pe sistemul de operare Linux (CentOS).

4.1. Python

```
[root@centos ~]# yum install python36
```

4.2. Django

```
[root@centos ~]# pip install django
```

4.3. Celery

```
[root@centos ~]# pip install celery
```

4.4. uWSGI

```
[root@centos ~]# pip install uwsgi
```

4.5. NGINX

```
[root@centos ~]# yum install nginx
```

4.6. Redis

```
[root@centos ~]# yum install redis
```

4.7. MariaDB

```
[root@centos ~]# yum install mariadb mariadb-server
[root@centos ~]# mysql_secure_installation
```

4.8. Docker

```
[root@centos ~]# yum install docker
```

4.9. Open Pipelines

Primul pas este copierea codului sursă al aplicației într-un director pe server. De exemplu în /opt/open_pipelines.

Odată ce avem codul sursă pe server, se continuă prin editarea fișierului „settings.py” unde trebuie configurate următoarele secțiuni:

- **DATABASES** – Aici se vor seta datele necesare conectării la baza de date MariaDB și anume adresa server, port, utilizator, parola și numele bazei de date care urmează a fi folosită.
- **CELERY_BROKER_URL** – Adresa și portul serverului Redis.
- **OPEN_PIPELINES_BASE_URL** – Adresa de la care va fi accesibilă aplicația pe web. De exemplu daca aplicația se va putea accesa folosind https://open-pipelines.anechitoaie.ro/ aici trebuie trecut https://open-pipelines.anechitoaie.ro/
- **OPEN_PIPELINES_BITBUCKET_OAUTH_KEY** – Această cheie pentru BitBucket OAuth API se obține accesând secțiunea „OAuth consumers" de pe pagina de setări a contului dumneavoastră de pe site-ul BitBucket.
- **OPEN_PIPELINES_BITBUCKET_OAUTH_SECRET** – Această valoare se obține din același loc descris în pasul anterior.
- **OPEN_PIPELINES_GITHUB_OAUTH_KEY** - Această cheie pentru GitHub OAuth API se obține accesând secțiunea „OAuth applications" de pe pagina de setări a contului dumneavoastră de pe site-ul GitHub.
- **OPEN_PIPELINES_GITHUB_OAUTH_SECRET** – Această valoare se obține din același loc descris în pasul anterior.

Următorul pas este configurarea serverului de aplicații uWSGI, lucru care se face prin editarea fișierului de configurare „uwsgi.ini".

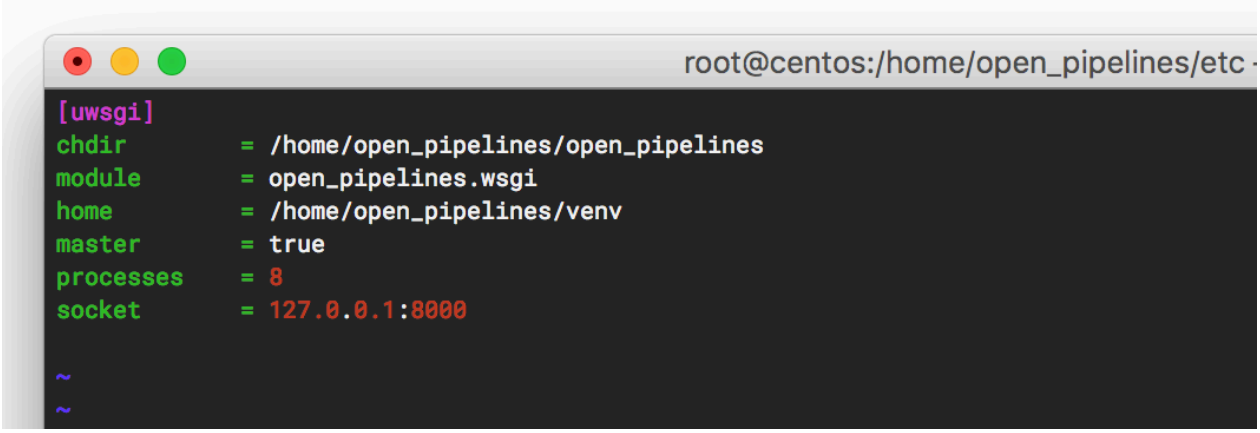

Figura 23: Exemplu de fișier de configurare uwsgi.ini folosit pentru aplicația demonstrativa găzduită la adresa https://open-pipelines.anechitoaie.ro/

În continuare vom configura serverul web (NGINX) pentru a se putea conecta la serverul de aplicații uWSGI și a răspunde la request-urile primite de la utilizatori.

Pentru aceasta vom crea un fișier nou de configurare în /etc/nginx/conf.d/open-pipelines.conf

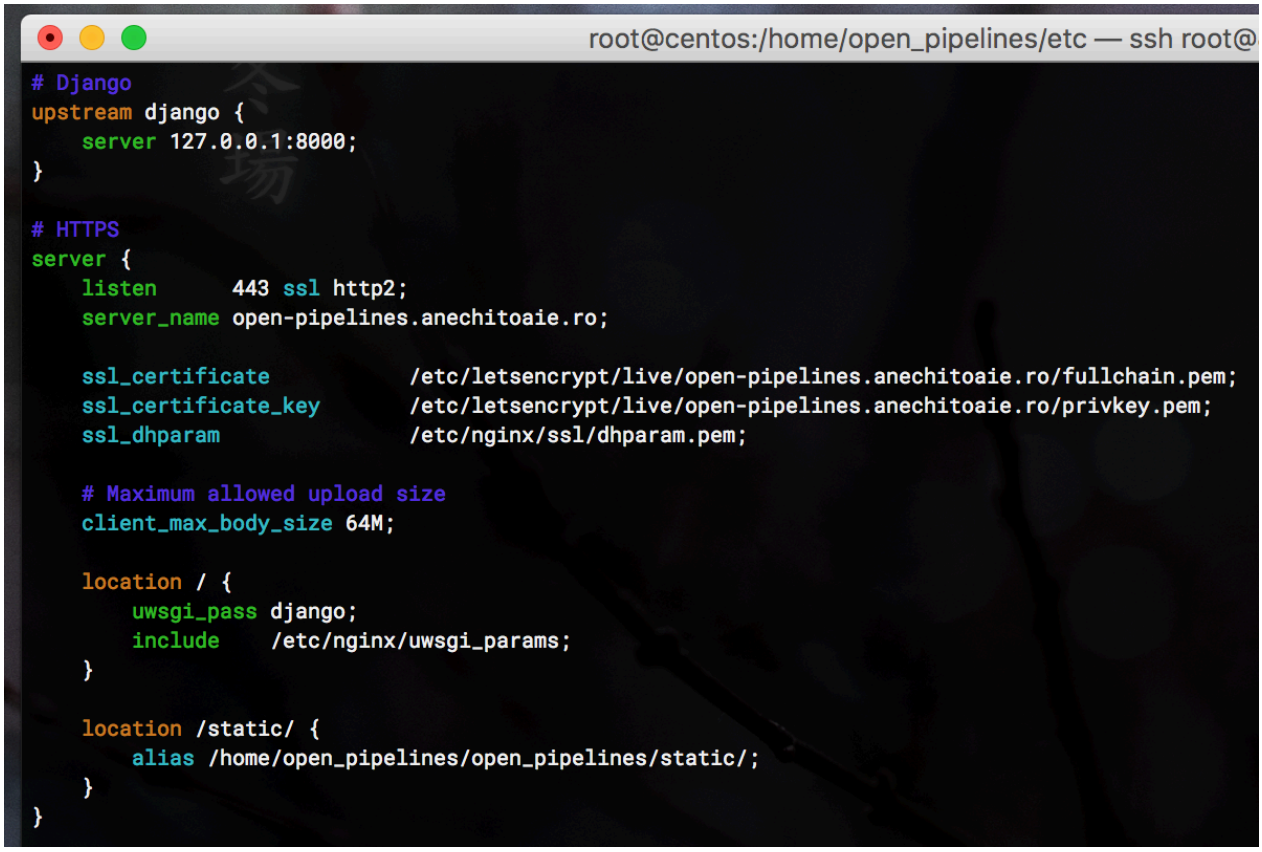

Figura 24: Exemplu de fișier de configurare open-pipelines.conf folosit pentru aplicația demonstrativa găzduită la adresa https://open-pipelines.anechitoaie.ro/

În acest moment, putem porni serviciile necesare rulării aplicației Open Pipelines folosind următoarele comenzi:

```
[root@centos ~]# systemctl start mariadb.service
[root@centos ~]# systemctl start docker.service
[root@centos ~]# systemctl start uwsgi.service
[root@centos ~]# systemctl start celery.service
[root@centos ~]# systemctl start nginx.service
```

1. Instrucțiuni de utilizare

1.1. Autentificare

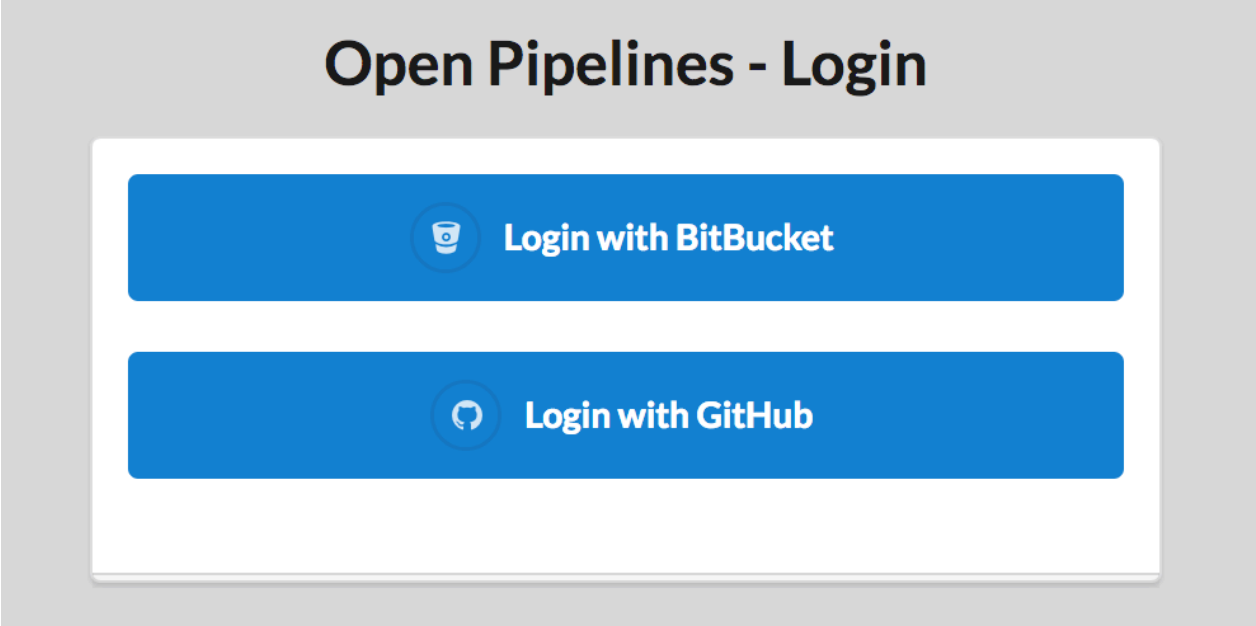

Figura 25: Pagina de autentificare

Pagina de autentificare este prima pagină pe care utilizatorul o va întâmpina în momentul accesării aplicației. Ea conține numele aplicației alături de cele doua butoane ce pornesc procesul de autentificare prin intermediul serviciului ales.

Autentificarea se face folosind protocolul OAuth2 pe baza unui cont de la unul din cele două servicii suportate în versiunea curentă a aplicației, și anume BitBucket sau GitHub.

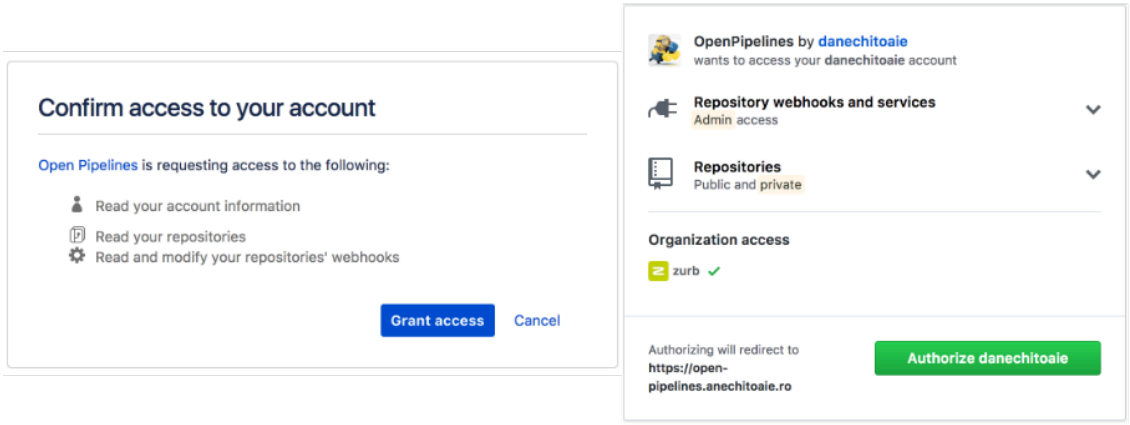

Figura 26: Pagina de autorizare a aplicației Open Pipelines pentru a accesa datele utilizatorului

După selectarea unui serviciu, utilizatorul trebuie să confirme accesul aplicației Open Pipelines la contul serviciului respectiv și la datele aferente.

1.2. Listare proiecte GIT

Figura 27: Pagina de listare a proiectelor GIT

După autentificare, utilizatorul va primi acces la pagina unde sunt listate proiectele GIT asociate contului cu care tocmai s-a autentificat. Lista de proiecte este generată în tip real folosind API-ul serviciului respectiv.

Proiectele sunt grupate pe pagini, utilizatorul având posibilitatea de a naviga intre paginile următoare și anterioare pentru a localiza proiectul dorit.

1.3. Configurare proiect (Open Pipelines)

Figura 28: Pagina de configurare a unui proiect GIT

Pentru a accesa această secțiune, utilizatorul trebuie să dea click pe butonul albastru cu iconița în forma de cheie.

Aici, aplicația va genera un URL unic pentru acest proiect care va trebui configurat mai apoi în pagina de administrare pe site-ul serviciului respectiv.

De asemenea, utilizatorul va avea acces și la un exemplu de fișier de configurare care îl va putea folosi ca și punct de plecare în configurarea pașilor care dorește sa fie urmați și executați de către aplicația Open Pipelines, în momentul în care se va face push la cod pe proiectul respectiv.

Tot aici, se mai poate configura și accesul la rezultatele build-urilor, dacă să fie publice (accesibile de către orice persoana care știe URL-ul respectiv) sau private (accesibile doar de către utilizatorul care deține acest proiect).

Toate modificările sunt salvate instant, nefiind necesar a se da click explicit pe un buton special făcut pentru a salva setările.

1.4. Configurare proiect (BitBucket/GitHub)

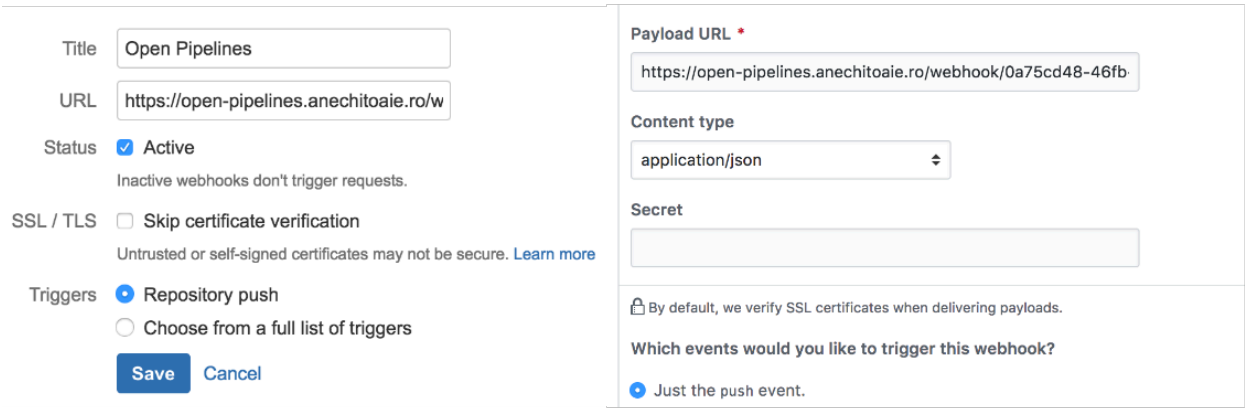

Figura 29: Pagina de configurare a webhook-ului pentru un proiect GIT

Următorul pas, este configurarea webhook-ului în pagina de setări a proiectului respectiv în funcție de serviciul ales.

Aici utilizatorul, va trebui să adauge URL-ul generat în pasul anterior și de asemenea să selecteze „repository push” ca și trigger pentru webhook.

1.5. Adăugarea fișierului open_pipelines.yml

```
1  # This is a sample build configuration for Javascript (Node.js).
2  # Only use spaces to indent your .yml configuration.
3  # -----
4  # You can specify a custom docker image from Docker Hub as your build environment.
5  image: node:latest
6
7  pipeline:
8      - node --version
9      - npm install
10     - npm test
```

Figura 30: Exemplu de fișier open_pipelines.yml

Ultimul pas înainte de a putea folosi aplicația, este adăugarea unui fișier „open_pipelines.yml” care va fi folosit pentru a defini pașii și comenzile ce vor fi executate de către aplicație la fiecare push al utilizatorului către proiectul respectiv.

„open_pipelines.yml” este un fișier text, în format YAML, mai sus având un exemplu cu formatul necesar pentru un proiect ce folosește NodeJS.

1.6. Afișarea rezultatului unui build

Din acest moment, aplicația este gata de utilizare și va asculta, recunoaște și interpreta request-urile primite de la webhook-urile serviciului respectiv, generate de evenimentele de tip GIT push ale utilizatorilor care au acces la proiect în momentul în care vor face commit cu modificările lor.

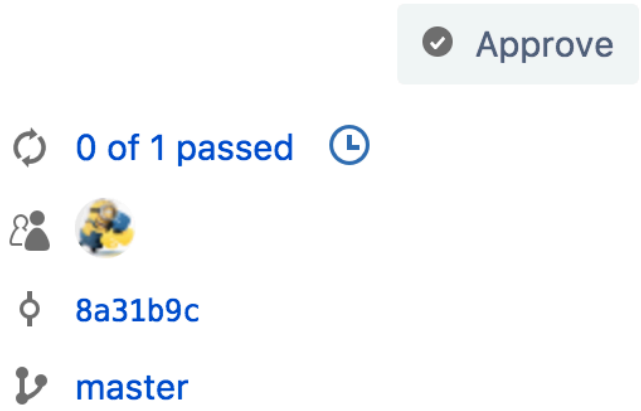

Figura 31: Build-ul este în progres

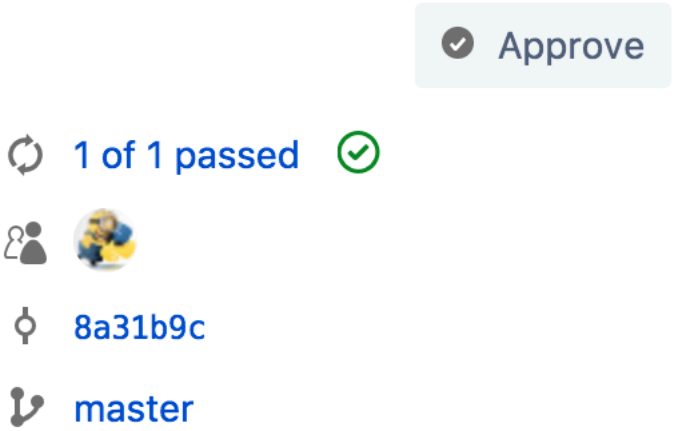

Figura 32: Build-ul s-a terminat cu succes

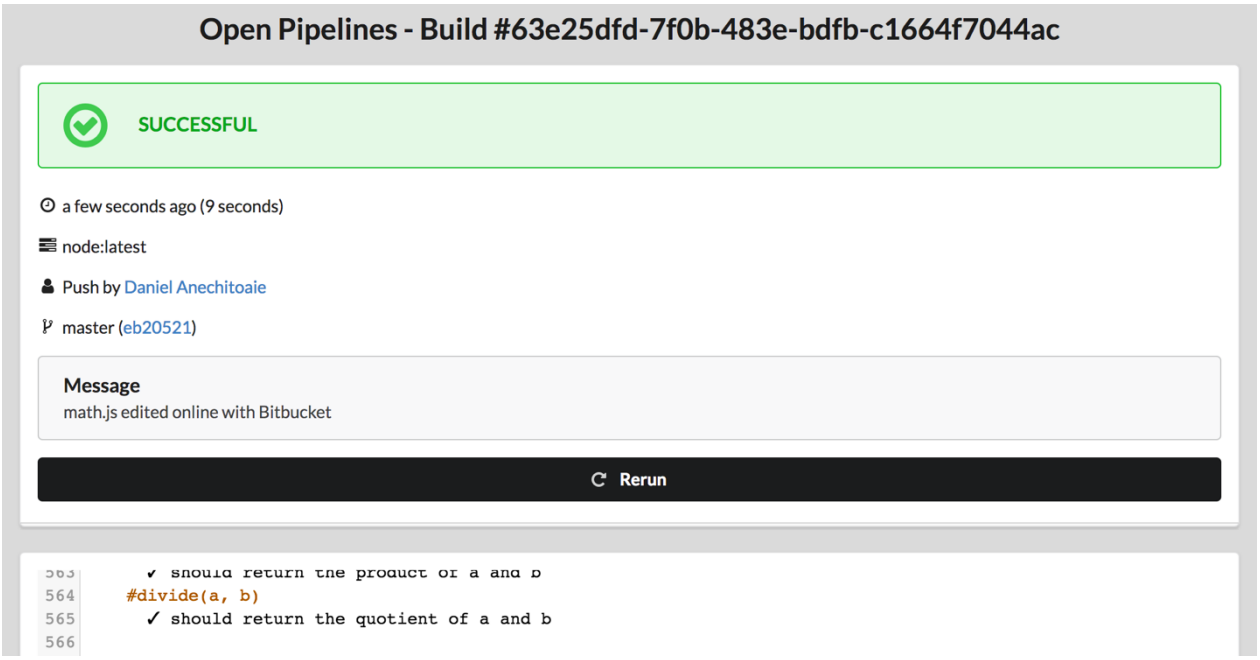

Figura 33: Pagina cu rezultatul build-ului și mesajele generate de aceasta

Progresul build-ului poate fi vizualizat în timp real fie pe pagina de commits a serviciului respectiv (BitBucket sau GitHub) fie pe pagina aplicației Open Pipelines unde utilizatorul poate vedea și mesajele generate de procesul de build, ce teste au fost făcute cu succes, ce teste au generat o eroare, etc.

Aici utilizatorul, are posibilitate și de a rula build-ul din nou în caz că dorește să ruleze din nou testele pe același cod.

2. Concluzii

Aplicația Open Pipelines se adresează tuturor echipelor ce lucrează într-un mod colaborativ în dezvoltarea proiectelor software. De la echipe mici, pana la echipe și proiecte mari, această aplicație poate asista și poate fi de ajutor în reducerea timpului necesar procesului de revizuire a codului, de impunere a unor standarde de scriere a codului și identificării erorilor.

Un lucru foarte important îl prezintă potențialul acestei aplicații. În momentul de față, versiunea curentă oferă suport pentru două dintre cele mai populare servicii de găzduire a codului pentru proiectele ce folosesc GIT și anume BitBucket și GitHub, dar, arhitectura modulară permite ușor extinderea aplicației și adăugarea de suport pentru servicii noi.

Câteva idei pentru a extinde aplicația și a oferi noi funcționalități ar putea fi:

- Mai multă granularitate în ceea ce privește accesul utilizatorilor la rezultatele unui build. În momentul de față, se poate configura doar dacă rezultatul să fie public (accesibil de către oricine are URL-ul build-ului respectiv) sau privat (accesibil doar de către deținătorul proiectului)
- Configurarea automată a webhook-urilor prin intermediul API-ului oferit de serviciile implementate. Momentan, acest pas se face manual de către utilizatorul aplicației.
- Limitarea numărului de workers disponibili unui proiect anume la un moment dat. În momentul de față un proiect poate porni un număr nelimitat de build-uri. Acest lucru nu va cauza probleme în sensul ca toate aceste procese sunt puse într-un queue și executate în funcție de numărul de workers, dar poate face ca restul proiectelor sa aștepte ceva mai mult până se eliberează un worker.
- Notificări. Ar fi destul de util implementarea de notificări, pentru ca utilizatorii să fie anunțați automat când s-au detectat probleme în codul lor.
- Integrarea cu programe de management al proiectelor cum ar fi Jira, Bugzilla, pentru a seta automat statusul unui task în funcție de rezultatele build-ului.

De asemenea aplicația poate fi și un exemplu de arhitectură modulară și scalabilă care poate fi folosit ca model în dezvoltarea altor aplicații de acest gen.

3. Bibliografie

Linux
[1] https://www.linux.org/

Python
[2] https://www.python.org/

Topul celor mai populare limbaje de programare conform unei analize făcute de StackOverflow.com
[3] https://insights.stackoverflow.com/survey/2017

Django
[4] https://www.djangoproject.com/

Celery
[5] http://www.celeryproject.org/

uWSGI
[6] https://uwsgi-docs.readthedocs.io/en/latest/

NGINX
[7] https://www.nginx.com/resources/wiki/

Redis
[8] https://redis.io/

MySQL
[9] https://www.mysql.com/

MariaDB
[10] https://mariadb.org/

Docker
[11] https://www.docker.com/

OAuth2
[12] https://oauth.net/2/

Diagramă funcționare OAuth2
[13] https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2

GIT
[14] https://git-scm.com/

Diagramă funcționare GIT
[15] https://www.git-tower.com/learn/git/ebook

BitBucket
[16] https://bitbucket.org/

GitHub
[17] https://github.com/

Diagramă mod de lucru cu GIT și pull requests
[18] http://slidedeck.io/gish/pull-request-presentation

Diagrama modeleului arhitectural al aplicației
[19] http://scipion.cnb.csic.es/old-docs/bin/view/TWiki/IntroductionDjango