UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

LUCRARE DE DIZERTATIE

Metode de upgrade in Kubernetes

**Coordonator Absolvent**

Conf.dr.ing. Radu-Ioan Ciobanu Eusebiu Rizescu

BUCUREŞTI

2021

# CUPRINS

[CUPRINS 1](#_Toc75173345)

[Sinopsis 3](#_Toc75173346)

[Abstract 3](#_Toc75173347)

[1 Introducere 4](#_Toc75173348)

[1.1 Context 4](#_Toc75173349)

[1.2 Problema 4](#_Toc75173350)

[1.3 Obiective 5](#_Toc75173351)

[1.4 Soluția propusă 5](#_Toc75173352)

[2 Tehnologii software folosite 7](#_Toc75173353)

[2.1 WWW/HTML/CSS/Bootstrap 7](#_Toc75173354)

[2.2 Python 7](#_Toc75173355)

[2.3 Flask 7](#_Toc75173356)

[2.4 MySQL 7](#_Toc75173357)

[2.5 Docker 7](#_Toc75173358)

[2.6 Kubernetes si Minikube 7](#_Toc75173359)

[2.7 Gitlab 7](#_Toc75173360)

[2.8 Prometheus 7](#_Toc75173361)

[2.9 Grafana 7](#_Toc75173362)

[3 Metode de upgrade 8](#_Toc75173363)

[3.1 Recreate 8](#_Toc75173364)

[3.2 Rolling Update 8](#_Toc75173365)

[3.3 Blue/Green 9](#_Toc75173366)

[3.4 Canary 11](#_Toc75173367)

[3.5 A/B Testing 11](#_Toc75173368)

[4 Proiectarea aplicatiei 12](#_Toc75173369)

[4.1 Descriere generala 12](#_Toc75173370)

[4.2 Inregistrare si Autentificare 15](#_Toc75173371)

[4.3 User Home Page 17](#_Toc75173372)

[4.4 Actualizare si stergere cont 18](#_Toc75173373)

[4.5 Adaugare, editare si stergere autovehicul 20](#_Toc75173374)

[4.6 Adaugare, editare, stergere si reimprospatare scadent 22](#_Toc75173375)

[5 Detalii implementare 26](#_Toc75173376)

[5.1 Prezentare generala 26](#_Toc75173377)

[5.2 Private Docker Registry 27](#_Toc75173378)

[5.3 Flask 28](#_Toc75173379)

[5.4 Emailer 30](#_Toc75173380)

[5.5 MySQL 31](#_Toc75173381)

[5.6 Securitate 33](#_Toc75173382)

[6 Concluzii si dezvoltari ulterioare 36](#_Toc75173383)

[6.1 Concluzii 36](#_Toc75173384)

[6.2 Dezvoltari ulterioare 36](#_Toc75173385)

[7 Bibliografie 38](#_Toc75173386)

[8 Anexe 39](#_Toc75173387)

[8.1 Baza de date 39](#_Toc75173388)

[8.2 Script build si rulare 40](#_Toc75173389)

# 

# Sinopsis

Un aspect foarte important atunci cand o aplicatie este folosita de clienti este upgrade-ul acesteia la o versiune noua, cu un impact cat mai mic resimtit de utilizatori. Clientii se asteapta ca aplicatia sa fie disponibila tot timpul, iar dezvoltatorii de cod doresc sa faca upgrade-uri cat mai des pentru a testa si imbunatati cat mai repede noul cod. Exista mai multe strategii de upgrade, care vor fi discutate in acest document. Pentru validarea si testarea acestora, a fost simulat un upgrade de aplicatie in mai multe moduri, pe infrastructura de Kuberenetes.

# Abstract

An important aspect when an application is used by customers is that the upgrade of it to a new version, should have the least impact felt by users. Customers expect the application to be available at all times and developers want to upgrade as often as possible to test and improve the new code as soon as possible. There are several upgrade strategies that will be discussed in this document. To validate and test them, an application upgrade was simulated in several ways, on the Kuberenetes infrastructure.

# Introducere

## Context

Actualizarea software este inevitabila, fiecare aplicatie care este folosita si dezvoltata trebuie actualizata, fie ca vorbim de adaugarea de functii noi, fie ca vorbim de rezolvarea unor defecte deja existente. În prezent, idea de CI/CD este intalnita in majoritatea companiilor care dezvolda software. Atat CI cat și CD compun coloana vertebrală a mediului DevOps. Acest mod de lucru umple spatiul dintre echipele de dezvoltare și echipele de operations, automatizând impachetarea aplicatiilor, testarea și deployment-ul acestora.

## Problema

Desi actualizarea software-ului poate parea un proces simplu și direct, aplicațiile de azi sunt adanc integrate în funcții, procese și sisteme. Inainte de fiecare punere in productie a unei aplicatii, trebuie analizat si dezvoltat un plan de upgrade. Nu toate aplicatiile sunt la fel, si astfel nu toate metodele de upgrade la fel.

Trebuie tinut cont de mai multe aspecte:

- Care este maximul de downtime pe care poate fi acceptat

- Care este maximul de resurse hardware care se pot adauga in timpul upgrade-ului

- Daca aplicatia poate rula pentru o perioada de timp cu 2 versiuni pornite in parallel (spre exemplu daca exista diferente de API care pot afecta utilizatorii finali)

- Daca aplicatia este stateful/stateless (daca este stateful, nu se poate pune versiunea noua instant, ci trebuie facut un upgrade treptat)

## Obiective

Scopul lucrarii este acela de a analiza mai multe metode de upgrade in Kubernetes. Pentru aceasta, s-a creat o aplicatie web, un cluster de kubernetes unde aplicatia ruleaza, scripturi de testare, sripturi de deployment, monotiorizare. Fiecare metoda a fost analizata, descrise avantajele si dezavantajele. Bineinteles, nici o varianta de uprade nu este perfecta, fiecare avand plusurile si minusurile respective, iar alegerea asupra unei metode trebuie sa tina cont de specificul aplicatiei, gradul de disponibilitate al aplicatiei care se doreste, cost, etc.

## Soluția propusă

Solutia propusa consta intr-o infrastructura prin care au putut fi analizate mai multe metode de upgrade in Kubernetes. Aplicatia pe care s-au analiat metodele de upgrade este o aplicatie web care reprezinta un magazin online. Aplicatia este scrisa in Python cu ajutorul frameworkului Flask. Ca stocare persistenta, aplicatia foloseste o baza de date relationala, MySQL. Clusterul de Kubernetes unde ruleaza aplicatia este Minikube, ruleaza pe masina locala. Pentru a testa in mod identic toate metodele de upgrade, a fost creata o suita de scripturi Bash care fac upgrade-ul, pentru a nu exista diferente intre metodele comparate. Pentru a testa metodele de upgrade, a fost creat un script de load test, scris in Python, care lanseaza in permanenta cereri HTTP care aplicatie. Pentru tragerea concluziilor, a fost creata o infrastructura de monitorizare formata din Prometheus si Grafana, care au monitorizat requesturile lansate de load tester, si au desenat grafice care au putut fi ulterior analizate.

Kubernetes este un sistem de gestionare a containerelor, open-source, folosit pentru automatizarea implementarii, scalarii și distribuirii aplicatiilor. Acesta a fost inițial proiectat de Google si este acum un proiect intretinut de Cloud Native Computing Foundation . Scopul sau este de a oferi o "platformă de automatizare a implementarii, scalarii și operarii containerelor de aplicatii în grupuri de computere". Funcționează cu o gamă largă de unelte pentru containerizare, inclusiv cu Docker.

# Tehnologii software folosite

## WWW/HTML/CSS/Bootstrap

## Python

## Flask

## MySQL

## Docker

## Kubernetes si Minikube

Kubernetes este un sistem de gestionare a containerelor, open-source, folosit pentru automatizarea implementarii, scalarii și distribuirii aplicatiilor. Acesta a fost inițial proiectat de Google si este acum un proiect intretinut de Cloud Native Computing Foundation . Scopul sau este de a oferi o "platformă de automatizare a implementarii, scalarii și operarii containerelor de aplicatii în grupuri de computere". Funcționează cu o gamă largă de unelte pentru containerizare, inclusiv cu Docker.

## Gitlab

## Prometheus

## Grafana

# Metode de upgrade

## Recreate

Concept: Se sterg resursele corspunzatoare versiunii actuale si se creaza altele cu noua versiune

Resurse necesare in plus: Nu

Avantaje:

- Starea aplicatiei este reluata de la 0

- Nu este nevoie de resurse suplimentare

- Simplu de pus in practica

Dezavantaje:

- Downtime care depinde de oprirea/pornirea aplicatiei

Unde se poate folosi:

- In stadiul de dezvoltare a codului

- In aplicatii unde nu este foarte important downtime-ul

- In aplicatiile unde se face rar upgrade

Chart, bubble chart

Description automatically generated

Fig. 1

## Rolling Update

Concept: Aplicatie este inlocuita pas cu pas. Spre exemplu daca aplicatia are 3 poduri cu versiunea N. Se porneste un pod nou cu versiunea N+1. Se asteapta pana podul cu versiunea N+1 este functional. Se sterge un pod cu versiunea N. Apoi se repeta acest procedeu pana cand toate podurile au versiunea N+1.

Resurse necesare in plus: Da. Daca se doreste upgrade-ul secvential, atunci este nevoie de 1 pod in plus. Se poate bineinteles face si upgrade-ul mai multor poduri in paralel.

Avantaje:

- Versiunea noua este pusa live pas cu pas

- Folositoare in cazul aplicatiilor stateful care stiu sa isi rebalansese incarcarea

Dezavantaje:

- Necesita resurse suplimentare

- Upgrade/Rollback dureaza mai mult timp

- Neadecvata aplicatiilor care prezinta diferente de API-uri intre versiuni

Unde se poate folosi:

- Front-end-ul aplicatiilor

- Aplicatii unde faptul ca 2 versiuni pot coexista

- Cand nu este permisa existenta downtime-ului

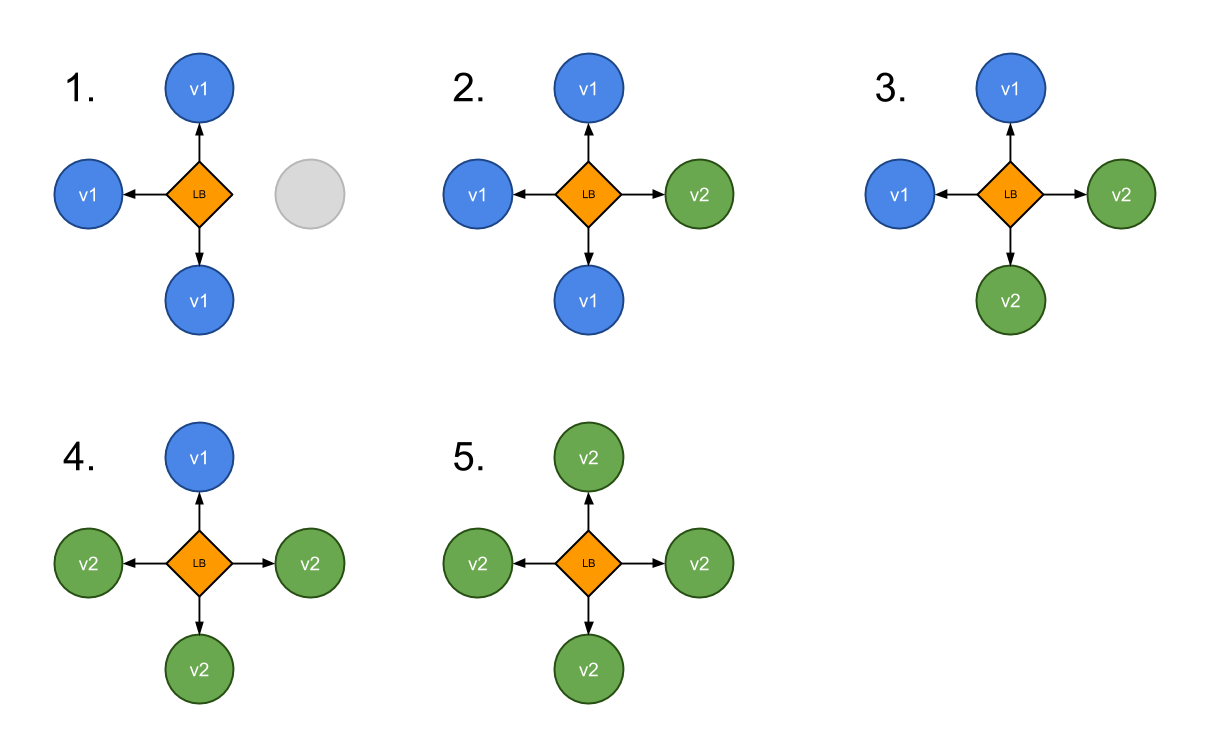


Fig. 2

## Blue/Green

Concept: Inainte de aplicatia propriu zisa, exsita un router (load-balancer) care directioneaza traficul catre versiunea Blue sau Green. Versiunea N+1 (Green) este pornita fara ca versiunea N (blue) a fi stearsa. Dupa testarea faptului ca noua versiune indeplineste cerintele, router-ul incepe sa trimita trafic catre versiunea Green.

Background pattern

Description automatically generated

Fig. 3

Resurse necesare in plus: Da. Deoarece schimbarea traficului se face abrupt, versiunea Green trebuie sa fie complet scalata pentru a servi traficul curent. Astfel, avem nevoie de dublul resurselor.

Avantaje:

- Upgrade/Rollback instant

- Evita problemele de diferente intre versiuni, deoarece schimbarea versiunii aplicatiei in intreg clusterul se face instant

- Usor de implementat

Dezavantaje:

- Dublul resurselor necesar in plus

- Este nevoie de testare riguroasa pana ca o versiune sa ajunga in procesul de upgrade

- Pot aparea dificultati in cazul aplicatiilor stateful

- Tranzactiile curente pot fi afectate

Unde se poate folosi:

- Cand avem diferente de API-uri intre versiunile aplicatiei

- Resursele aplicatiei sunt mici si/sau bugetul este mare

## Canary

Concept: Rutarea unui subset de utilizatori catre utilizarea noii versiuni (cu alte cuvinte se face testare direct in productie). Versiunea N+1 a aplicatiei este pornita, intr-o singura instanta, iar o parte mica din trafic este redirectata catre aceasta. Daca se observa ca pe acel subset de useri nu apar probleme, se mareste procentul, iar apoi versiunea canary se transforma in versiune de baza.

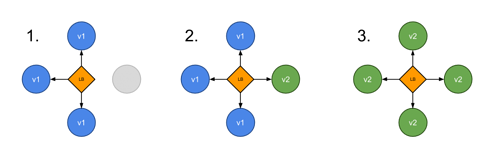


Fig. 4

Resurse necesare in plus: Da. Este necesar cel putin un pod in plus pentru noua versiune. Apoi se sterg poduri din versiunea veche si se adauga poduri cu versiunea noua.

Avantaje:

- Noua versiune ajunge la un subset de useri

- Folositoare pentru monitorizarea performantei si a ratei de eroare

- Rollback rapid

- Mai ieftina decat blue/green

Dezavantaje:

- Upgrade-ul dureaza mult timp

Unde se poate folosi:

- Aplicatii unde faptul ca 2 versiuni pot coexista

- Cand nu este permisa existenta downtime-ului

## A/B Testing

Concept: Aceasta tehnica este mai degraba o tehnica de business. Spre exemplu adaugarea unei functionalitati noi pentru un set de useri, sau modificare unei functionalitati pentru un set de useri si analiza ulterioara referitoare la care varianta este mai buna sau profitabila. A/B Testing este o extensie a Canary Deployment, in sensul ca subsetul de useri se poate calcula si pe trafic in sine, dar si pe anumite headere din request (User-Agent, headere speciale) precum si pe baza de cookies.

Graphical user interface

Description automatically generated

Fig. 5

Resurse suplimentare: Da. Este necesar cel putin un pod in plus pentru noua versiune. Apoi se sterg poduri din versiunea veche si se adauga poduri cu versiunea noua.

Avantaje:

- Control total asupra distributiei de traffic

- Rollback rapid

- Noua versiune ajunge la un subset de useri

Dezavantaje:

- Este nevoie de un loadbalancer inteligent

- Upgrade incet

Unde se poate folosi:

- Cand se doreste analiza unei modificari a unei functionalitati

- Front-end-ul aplicatiilor

# Proiectarea aplicatiei

## Descriere generala

Aplicatia nu are un scop clar, este mai degraba un sablon pentru un magazin online. Pe pagina principala avem produsele aflate in vanzare, iar cumparatorii isi pot adauga in cosul de cumparaturi produsele dorite. Pentru a lansa o comanda, cumparatorul trebuie sa isi creeze un cont, sa se autentifice, sa isi adauge produsele dorite in cosul de cumparaturi, si sa lanseze comanda. Aplicatia este scrisa in Python cu ajutorul framework-ului Flask. Am folosit si Bootstrap, care este un framework de CCS pentru a nu reinventa roata. Aplicatia foloseste ca stocare persistenta o baza de date relationala, si anume MySQL. Pentru a interactiona cu baza de date am folosit SQLAlchemy. SQLAlchemy este un toolkit si ORM care functioneaza ca un wrapper peste majoritatea bazelor de date relationale. Aceasta functioneaza prin maparea tabelelor din baza de date in clase din Python. Totodata, SQLAlchemy are si mecanisme ce previn SQL injection (acesta traduce comenzile in limbajul bazei de date si nu le executa direct).

In prezent, aplicatia ruleaza in Docker (un container cu aplicatia, si un container cu baza de date). Imaginea pentru aplicatie este construita plecand de la Apline, si continand toate dependintele necesare (Python, pip, pachetele de python necesare, etc). Imaginea de MySQL este una oficiala. Pornirea containerelor este facuta cu ajutorul docker-compose. In final, aplicatia va fi deployata pe un cluster de Kubernetes, pentru a testa metodele de upgrade prezentate in semestrul anterior.

Diagrama site-ului este urmatoarea:

Diagram

Description automatically generated

Fig. 6

Paginile se impart in doua categorii:

1. Care nu necesita ca utilizatorul sa fie autentificat:
   * Home
     + Pagina principala. Aici se pot vedea produsele puse in vanzare de catre magazin. Pentru a putea adauga in cos produse, utilizatorul trebuie sa aiba cont si sa fie logat.
   * SignUp
     + Pagina de inscriere. De aici un utilizator nou poate completa un formular pentru a isi crea un cont nou. Acesta va primi pe mail un link de activare.
   * SignIn
     + Pagina de autentificare. De aici un utilizator se poate autentifica pentru a putea adauga produse in cosul de cumparaturi, goli cosul de cumparaturi, sau lansa o comanda.
   * Uitat parola
     + Pagina in care se poate reseta parola contului. Se completeaza un formular care contine adresa de email, iar daca exista un cont asociat acelei adrese de email, se va trimite un link de resetare a parolei.
2. Care necesita ca utilizatorul sa fie autentificat:
   * User Home
     + Pagina pe care un utilizator o vede dupa ce se logheaza. Aici se poate vedea continutul cosului de cumparaturi, goli cosul de cumparaturi, lansa o comanda.
   * Editare cont
     + Pagina de editare cont. De aici utilizatorul isi poate actualiza informatiile sale: numele, adresa de email, adresa fizica, parola.
   * Stergere cont
     + Pagina de stergere cont. De aici un utilizator isi poate sterge contul. Aceasta operatiune este ireversibila, deoarece contul va fi sters permanent din baza de date
   * Golire cos de cumparaturi
     + Cosul de cumparaturi va fi golit. Nu va fi trimisa nici o comanda..
   * Plasare comanda
     + De aici utilizatorul poate plasa o comanda cu produsele aflate in cosul de cumparturi.

## Inscrierea si Autentificare

Pentru ca un utilizator sa poata adauga produse in cosul de cumparaturi si implicit sa lanseze o comanda, aceasta trebuie sa fie autentificat. Pentru a se autentifica, mai intai are nevoie de un cont. Pentru a isi crea un cont (a se inscrie), utilizatorul trebuie sa acceseze pagina “Sign Up”, pagina dispobila de la pagina initial, prin apasarea butonului din dreapta sus numit “Sign Up”. Pentru a isi face un cont nou, utilizatorul trebuie sa completeze campurile de mai jos (Email, Nume, Prenume, Adresa, Parola, Confirmare Parola). Utilizatorul va primi bineinteles mesaje de eroare daca mailul nu este valid (spre exemplu nu contine “@”), sau daca mailul este deja existent in sistem, sau daca cele doua parole introduse in campurile “Parola” si “Confirmare Parola” nu corespund.

**A picture containing graphical user interface

Description automatically generated**

Fig. 7

Dupa introducerea campurilor si apasarea butonului “Inregistrare!”, utilizatorul va primi un mail cu un link de activare. Acel link de activare contine in el campul “hash” corespunator utilizatorului. Acest tip de verificare se numeste Email Validation.

Hash-ul este format din email si timestamp-ul de cand a fost creat contul. Fiecarui cont de utilizator ii este asociat un camp “hash” si un camp “activated”. Pana ce utilizatorul nu va accesa acel link-ul de activare primit, acesta va avea campul “activated” setat pe fals.

Cat timp campul “activated” asociat contului este setat pe fals, utilizatorul nu se va putea autentifica. Acest pas de activare pe email, este pentru a verifica ca persoana care doreste sa isi faca un cont nou, detine acest email.

Pagina autentificare – pagina de unde un utilizator se poate autentifica:

Shape

Description automatically generated with low confidence

Fig. 8

In baza de date parola utilizatorului nu este stocata in clar (din punct de vedere al securitatii). In schimb, este tinut un hash al acesteia (dupa cum se poate observa in figura 8). Acest hash este creat cvu ajutorul functiei “generate\_password\_hash” din catrul pachetului “wekzeug.security”, pachet care este folosit des in ecosistemul Python datorita gradului mare de securitate pe care il ofera.

Graphical user interface, text

Description automatically generated

Fig. 9

## Pagina principala

Aici se pot vedea produsele puse in vanzare de catre magazin. Pentru a putea adauga in cos produse, utilizatorul trebuie sa aiba cont si sa fie logat. In figura de mai jos, deoarece utilizatorul este autentificat (in dreapta sus apare butonul “logout” ceea ce inseamna ca utilizatorul este autentificat), utilizatorul poate sa adauge in cos produse. Daca nu ar fi fost autentificat, butonul de adaugat in cosul de conparaturi ar fi fost greyed out, si nu ar fi putut sa adauge in cos produsul dorit.

Graphical user interface, application

Description automatically generated

Fig. 10

## Pagina utilizatorului

Aceasta este pagina catre care utilizatorul va fi rutat dupa autentificare. De aici utilizatorul isi poate edita si sterge contul, isi poate goli cosul de cumparaturi, si poate lansa o comanda. Tot din aceasta pagina, prin apasarea butonului “Editare cont” utilizatorul va fi rutat catre pagina de editare a contului.

Graphical user interface

Description automatically generated

Fig. 11

## Actualizare si stergere cont

De aici utilizatorul isi poate actualiza numele, adresa de email, parola si adresa fizica, sau isi poate sterge contul. Pentru a isi actualiza datele personale, utilizatorul trebuie sa isi completeze campurile pe care doreste sa le actualizeze si sa apese butonul “Actualizeaza”. De mentionat ca stergerea contului este o operatiune ireversibila, deoarece toate datele asociate cu acest cont vor fi sterse din baza de date. Prin apasarea butonului “Inapoi” utilizatorul se poate intoarge la pagina “User Home”.

A picture containing diagram

Description automatically generated

Fig. 12

## Baza de date

Baza de date contine doua tabele:

- tabela useri care contine userii

- tabela produs care contine produsele

Legatura dintre cele doua este facuta prin coloana “cosDeCumparaturi” a tabelei useri. Aceasta este un string care contine produsele (si cantitatea acestora) adaugate de utilizator in cos. Majoritatea campurilor sunt destul de usor de inteles, cu exceptia campurilor “activated” si “hash” care vor fi explicate in capitolul urmator. Acestea au legatura cu securitatea aplicatiei, si anume sunt folosite la crearea unui cont nou, si la accesarea functionalitatii de “Uitat Parola”.

Graphical user interface, application

Description automatically generated

Fig. 13

Integrarea aplicatiei cu baza de date se face prin intermediul modulului Flask-SQLAlchemy. Astfel baza de date este abstractizata, iar interactionarea cu aceasta se face prin intermediul obiectelor. Astfel se profita la maximum de puterea programarii orientata pe obiecte in lucrul cu baza de date. Totodata, datorita folosirii unui ORM (Object Relational Mapping), daca se doreste schimbarea bazei de date cu alt motor (Oracle, Postgres, etc), nu este nevoie de modificari in cod deoarece Flask-SQLAlchemy suporta cele mai des intalnite baze de date.

Text

Description automatically generated

Fig. 14

# Detalii arhitectura testare

## Prezentare generala

Arhitectura este formata din mai multe componente, care vor fi detaliate in subcapitolele ce urmeaza (se poate observa in figura legaturile dintre aceastea). Pe scurt, cand apare o versiune noua de aplicatie (in prezent, pentru teste, aplicatia are 2 versiuni, pentru a putea face testele), utilizatorul porneste din Gitlab pipeline-ul de upgrade al aplicatiei. Gitlab este integrat cu clusterul de Kubernetes local, si acest pipeline ii trimite clusterului instructiuni de a face upgrade aplicatiei (bineinteles, cu tipul de upgrade dorit). In tot acest timp, scriptul de loat test trimite in permanenta requesturi aplicatiei pentru a monitoriza proportia de requesturi intre cele 2 versiuni, si pentru a masura procentul de erori.

Diagram

Description automatically generated

Fig. 15

## Clusterul de Kubernetes

Clusterul de Kubernetes ruleaza cu ajutorul Minikube, pe masina locala, adica intr-un singur nod. In productie, acest cluster este necesar sa ruleze pe mai multe clustere pentru a asigura toleranta la defecte hardware sau software la nivelul serverelor. In testele facute, aici ruleaza aplicatia cu componentele sale. Cu acest cluster interactioneaza Gitlab care il instructioneaza prin API ce resurse sa porneasca/modifice/opreasca. Totodata, scriptul de load test, trimite permanent requesturi la aplicatia ce ruleaza pe acest cluster, pentru a analiza codurile de raspuns.

## Gitlab

In Gitlab sunt tinute atat manifesturile de Kubernetes care descriu resursele si infrastructura aplicatiei, cat si pipeline-ul de creare, upgrade, stergere. El interactioneza cu clusterul de Kubernetes prin ajutorul Kubernetes API. Pentru conectare, de pe routerul local s-a facut forwardare de porturi catre masina locala unde ruleaza Kubernetes si realizata conexiunea cu accest cluster (pe baza de certificat si token), pentru ca mai apoi runneri din Gitlab.com sa poata executa comenzi pe acest cluster.

Datorita conexiunii dintre Gitlab si clusterul de Kubernetes, putem vedea in interfata de Gitlab statusul actual al clusterului (cate poduri avem pornite, starea lor). In plus, in cazul Canary, putem seta din interfata procentul de trafic pe care il dorim catre noua versiune de aplictie:

Graphical user interface, application

Description automatically generated

Fig. 16

Astfel, direct din interfata de Gitlab, putem vedea cate poduri de aplicatie avem (3 cuburi versi) si cate poduri de versiune Canary avem (1 cub verde cu bulina portocalie).

Pipeline-ul din Gitlab este urmatorul

* Pasul 0: Apeleaza pasii: pasul 4 🡪 asteapta 60s 🡪 pasul 1 🡪 asteapta 300s 🡪 pasul 2 🡪 asteapta 300s 🡪 pasul 3 🡪 asteapta 300s 🡪 pasul 4
* Pasul 1: Porneste resursele necesare pentru a livra Versiunea 1
* Pasul 2: Porneste resursele necesare pentru a livra si Versiunea 2
* Pasul 3: Sterge Versiunea 2 pentru a livra doar Versiunea 1
* Pasul 4: Sterge toate resursele din namespace

Graphical user interface

Description automatically generated with medium confidence

Fig. 17

## Aplicatia

Dupa cum a fost prezentat si in capitolul anterior, aplicatia este o aplicatie web, dezvoltata in Flask (framework Python). Aceasta are doua versiuni, impachetate in 2 imagini Docker, si stocate pe DockerHub.

Diagram

Description automatically generated

Fig. 18

Ambele versiuni, la un request GET pe path-ul „/”, raspund cu numarul versiunii. Facem acest lucru pentru a putea identifica in grafice proportia dintre versiuni.

Pentru a avea teste cat mai apropiate de productie, versiunea 1 a aplicatiei raspunde mereu cu status code „200” OK, pe cand versiunea 2, in anumite cazuri va raspunde simuland un „500” Internal Server Error. Pentru o mai buna monitorizare, acest „500” simulat, este tot un raspuns de „200” care contine JSON-ul:

{"appVersion": "2", "status": "500"}

Bineinteles, versiunea 1 returneaza appVersion = 1 si versiunea 2 returneaza appVersion = 2. Pentru a avea control mai bun asupra analizei codurilor de raspuns, ambele versiuni de aplicatie returneaza mereu codul HTTP 200OK, insa un raspuns care contine status = 500 este considerat in analiza ca fiind eroare.

Aceasta aplicatie va rula pe clusterul de Kubernetes prezentat in subcapitolul anterior.

Considerente pentru teste:

1. Pentru a simula un caz din realitate, ambele versiuni de aplicatie returneaza status = 500 pentru primele 60 de secunde de la pornirea aplicatiei, si astfel un pod nou, nu va fi pus in poolul de poduri sanatoase in primele 60 secunde. Ar fi neprofesionist a se pune in poolul de poduri active o aplicatie despre care se stie cu certitudine ca pentru o perioada de timp raspunde incorect.

2. Versiunea 1 returneaza mereu status = 200, adica este stabila pentru orice user agent.

3. Versiunea 2 returneaza pentru un singur user-agent, status = 500, cu o probabilitate configurabila din cod, iar pentru toti ceilalti user-agent returneaza status = 200. Note despre modul in care au fost configurate testele:

- Versiunea 2 de aplicatie returneaza status = 500 pentru user-agent-ul „Sometimes-Fail”, cu probabilitatea de 50%

- In load-test, se vor executa requesturi in numar egal pentru 2 useri agenti: „Always-Good” si „Sometimes-Fail”

- Astfel, cu considerentele de mai sus, daca 100% din trafic va fi pe versiunea 2, vom avea 75% din trafic considerat corect si 25% din trafic considerat cu eroare.

4. Versiunea 1 va rula pe 3 pod-uri. Versiunea 2 va rula pe numar variabil de poduri in functie de metoda de upgrade.

## Scriptul de load test

Acest script este dezvoltat in Python 3 si trimite permanent (cu pauza de 0.01 secunde) requesturi catre aplicatie. Acest script, trimite requesturi in numar egal folosind 2 useri agenti: „Always-Good” si „Sometimes-Fail”, pentru a simula trafic din realitate. Bineinteles, s-ar fi putut folosi 100 de user-agenti, dar pentru a se observa mai usor pe grafice diferentele intre modalitatile de deploy, am ales numere mici (2 useri agenti) si care sa se imparta frumos (rata de fail 50% pe versiunea 2) .

Cum a fost precizat mai sus, ambele versiuni de aplicatie raspund cu codul HTTP 200OK, insa pentru a determina daca raspunsul este interpretat ca fiind corect sau eroare, vom analiza JSON-ul primit ca raspuns. Scriptul va analiza fiecare raspuns si va forma 4 metrici: „requests200V1”, „requests500V1”, „requests200V2”, „requests500V2”.

Problema care apare este aceea de stocare a acestor metrici in functie de timp. Astfle, acest script de load-test expune si metrici (similar cu node-exporter), care vor fi ulterior citite periodic (fiecare 5 secunde), de catre Prometheus. Requesturile permanente ruleaza pe un thread separat si formeaza aceste metrici, iar, cu ajutorul framework-ului Flask, scriptul le expune pe calea „/metrics”. Avand in vedere ca aceste metrici sunt crescatoare (countere), la fiecare request al Prometheus-ului catre scriptul de load test, metricile vor fi resetate. Cu aceasta abordare, in Prometheus vom avea metrici de forma „sum(requests) per 5 minute”.

## Prometheus

Pentru a stoca istoria statusurilor requesturilor initiate de scriptul de load-test, se va folosi o instanta de Prometheus. Aceasta este configurata ca la fiecare 20 secunde sa colecteze metrici de la scriptul de load-test, pe care le stocheaza in baza lui de date interna. Acesta este deploy-at tot local, intr-un container Docker. Fisierul de configurare al Prometheus-ului poate fi vazut in figura de mai jos. Pe scurt, acesta este configurat sa se duca la portul pe care scriptul de load-test il expune, pentru a cere date referitor la requesturile lansate de acesta.

Text

Description automatically generated

Fig. 19

## Grafana

Deoarece Prometheus nu exceleaza la capitolul de vizualizare a metricilor, s-a folosit Grafana, care are ca sursa de date Prometheus-ul configurat anterior. In Grafana s-a creat un dashboard care contine mai multe grafice care ne pot ajuta la compararea metodelor de deploy. Acestea sunt:

1. Distributia de trafic (cat % versiunea 1 si cat % versiunea 2)
2. Rata de eroare versiunea 2
3. Rata de eroare total
4. Numarul de requesturi: V1 200, V1 500, V2 200, V2 500. (V1 500 nu este cazul deoarece avem mereu 200 pe versiunea 1).

A screenshot of a computer

Description automatically generated with medium confidence

Fig. 20

# Concluzii si dezvoltari ulterioare

## Concluzii

Aplicatia Carplanner este un instrument care vine in ajutorul posesorilor de masini dar si firmelor care detin numeroase vehicule, si le ofera un suport pentru gestionarea mai eficienta a documentelor si a reviziilor vehiculelor.

Ca orice solutei nou aparuta, trebuie incercata in piata pentru a ne da seamna daca are succes sau nu. In momentul de fata, aplicatia poate fi pusa pe un server in cloud (spre exemplu Amazon AWS), achizitionat un domeniu si un certificat si pusa in piata fara probleme.

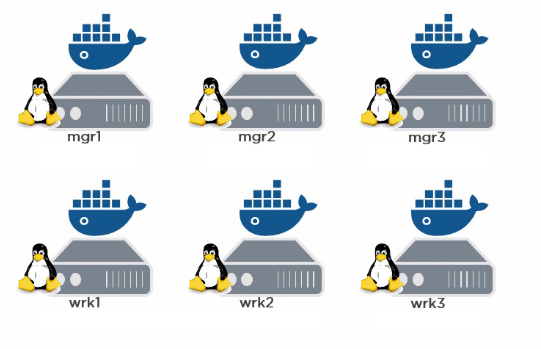
In realizarea acestei solutii am folosit mai multe tehnologii, unele deja cunoscute de mine MySQL, Python, Docker iar aletele in totalitate noi, cum ar fi Flask, WTForms, HTML+CSS.

## Dezvoltari ulterioare

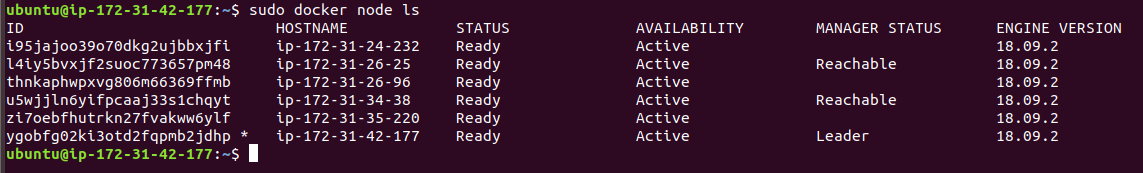
* Upgrade baza de date

Pentru a creste sansele de a avea succes, aplicatia necesita multe imbunatatiri viitoare:

* Deploy in cloud, pe mai multe servere, pentru a creste fiabilitatea in caz de esuare. Un exemplu ar fi urmatorul: 3 manageri si 3 workeri. Pe manageri nu ruleaza containere (astfel pot avea resurse mai putine), acestia doar coordonand alocarea cintainerelor pe nodurile worker:



*Fig. 33 – Eventuala configuratie cloud*



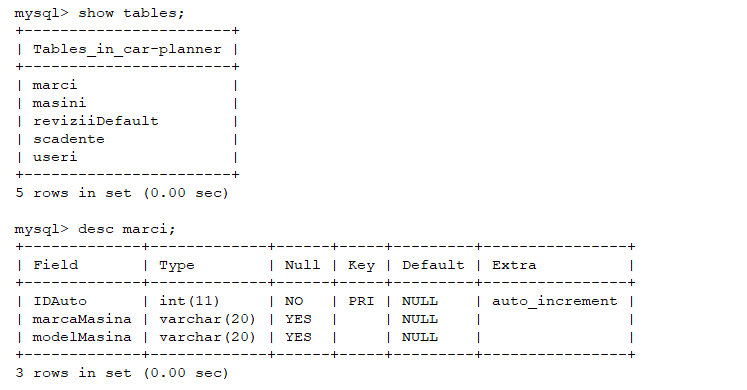
*Fig. 34 – Eventuala configuratie cloud*

* Componenta MySQL sa poata fi replicata, pentru a creste eficienta in caz ca numarul de utilizatori va creste. Se poate utiliza modul default de replicare al MySQL, master-slave sau se poate utiliza spre exemplu GlusterFS pentur a rula in modul master-master (ambele noduri pot scrie simultan fata de modul default master-slave unde doar nodul master poate scrie, iar nodurile slave pot doar citi)
* In cazul in care se opteaza pe un deployment pe mai multe servere, se poate configura si un load-balancer, pentru o mai buna manipulare a cererilor clientilor
* Pentru a putea sugera corect lista de revizii default pe care un utilizator este intrebat daca doreste sa le adauge unui nou vehicul adaugat, acestea trebuiesc sa fie corespunzatoare cu ce recomanda producatorul. Astfel trebuie populata corect si cat mai complet tabela „reviziiDefault” pentru fiecare model de masina in parte
* O alta imbunatatire ar fi permiterea utilizatorului in a customiza vehiculele si scadentele pentru care doreste sa primeasca notificari, dar si cu cate zile sa fie notificat inainte de expirarea fiecarui scadent
* Se poate, de asemnea, opta si pentru notificarea prin SMS a utilizatorului. Trebuie modificata baza de date astfel incat sa se stocheze si numarul de telefon al utilizatorului, si de asemnea achizitionat o solutie care sa trimita efectiv notificarile SMS
* Putem adauga un newsletter cu articole referitoare la noile legislatii si amenzi, noutati in lumea automobilelor, detalii tehnice si orice altceva ce are legatura cu domeniul auto.

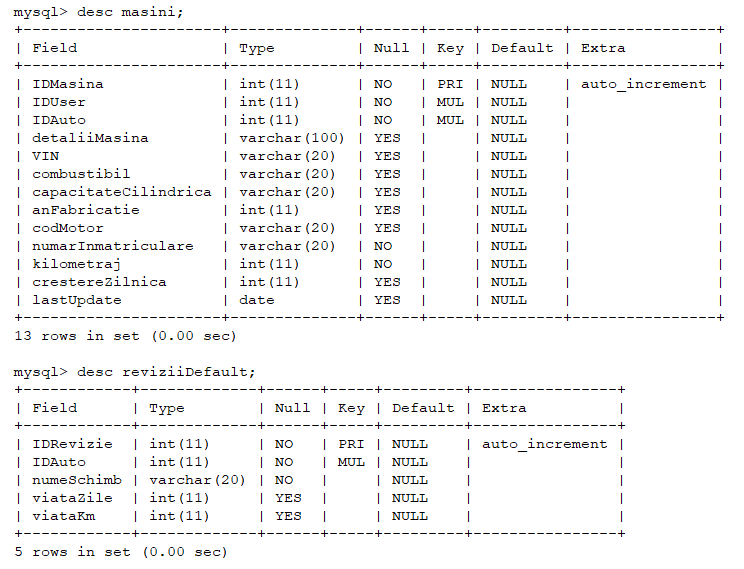
# Bibliografie

# Anexe

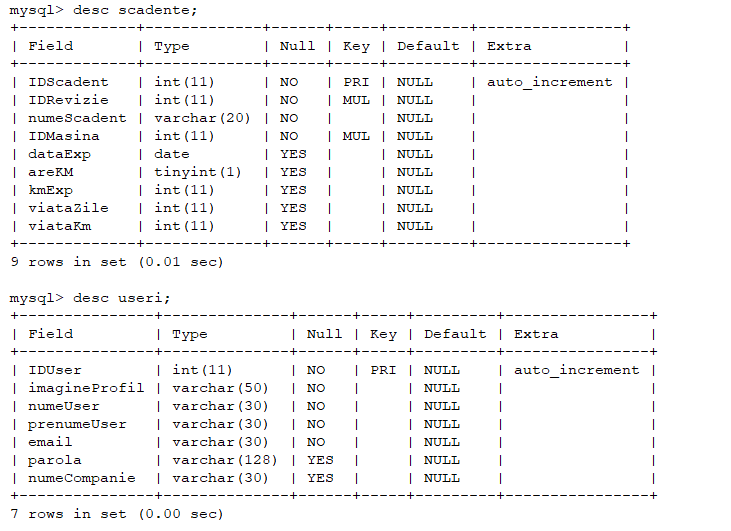
## Baza de date



*Fig. 35 – Tabele 1*



*Fig. 36 – Tabele 2*



*Fig. 37 – Tabele 3*

## Script build si rulare

#! /bin/bash

if [ "$1" == "help" ]; then

echo "./build.sh help <- help"

echo "./build.sh first <- first run on machine"

echo "./build.sh flask <- build flask-image and start docker-compose"

echo "./build.sh emailer <- build emailer-image and start docker-compose"

echo "./build.sh all <- build all images and start docker-compose"

echo "./build.sh start <- start docker compose"

echo "./build.sh stop <- stop docker compose"

echo "./build.sh populate <- populate database tables"

exit

fi

if [ "$1" != "populate" ]; then

echo Exit Docker Swarm

sudo docker swarm leave --force

sleep 3

if [ "$1" != "stop" ]; then

echo Init Docker Swarm

sudo docker swarm init

fi

sleep 3

fi

if [ "$1" == "first" ]; then

# delete all containers

sudo docker rm -f $(docker ps -a -q)

sleep 1

# delete all images

sudo docker image rm -f $(docker images -q -a)

# clean tree

sudo rm -rf /var/lib/car-planner

# create tree

sudo mkdir /var/lib/car-planner

sudo mkdir /var/lib/car-planner/registry

sudo mkdir /var/lib/car-planner/mysql

sudo mkdir /var/lib/car-planner/flask

sudo mkdir /var/lib/car-planner/emailer

# copy application files

sudo cp -R ./Flask/\* /var/lib/car-planner/flask

sudo cp -R ./Emailer/\* /var/lib/car-planner/emailer

sleep 1

fi

if [ "$1" != "stop" ] && [ "$1" != "first" ]; then

echo "Removing /var/lib/car-planner/flask/\*"

sudo rm -rf /var/lib/car-planner/flask/\*

echo "Removing /var/lib/car-planner/emailer/\*"

sudo rm -rf /var/lib/car-planner/emailer/\*

echo "Copying to /var/lib/car-planner/flask/\*"

sudo cp -R ./Flask/\* /var/lib/car-planner/flask/

echo "Copying to /var/lib/car-planner/emailer/\*"

sudo cp -R ./Emailer/\* /var/lib/car-planner/emailer/

fi

if [ "$1" == "flask" ] || [ "$1" == "first" ]; then

echo "Start building flask-image"

sudo docker build --no-cache ./Dockerfiles -f ./Dockerfiles/FlaskDockerfile -t flask-image

fi

if [ "$1" == "emailer" ] || [ "$1" == "first" ]; then

echo "Start building emailer-image"

sudo docker build --no-cache ./Dockerfiles -f ./Dockerfiles/EmailerDockerfile -t emailer-image

fi

if [ "$1" == "all" ]; then

echo "Start building flask-image"

sudo docker build --no-cache ./Dockerfiles -f ./Dockerfiles/FlaskDockerfile -t flask-image

echo "Start building emailer-image"

sudo docker build --no-cache ./Dockerfiles -f ./Dockerfiles/EmailerDockerfile -t emailer-image

fi

if [ "$1" != "stop" ]; then

sleep 1

sudo docker stack deploy -c docker-compose.yml car-planner

fi

if [ "$1" == "stop" ]; then

# delete all containers

echo "Delete all containers"

sudo docker rm -f $(docker ps -a -q)

fi

if [ "$1" == "populate" ]; then

docker exec $(docker ps | grep flask | cut -d " " -f1 | head -1) python3 populateTables.py

fi

TODOooooOOOOOOOO

* Imaiginile sa fie numerotate bine
* Sa am spatierea peste tot la fel
* Sa am font si size peste tot la fel
* Daca n-am pagini, pot sa mai scriu de „securitate”, activare pe mail, bla bla