

UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE



LUCRARE DE DIZERTATIE

Metode de upgrade in Kubernetes

Coordonator

Conf.dr.ing. Radu-Ioan Ciobanu

Absolvent

Eusebiu Rizescu

BUCUREȘTI

2021

CUPRINS

CUPRINS.....	1
Sinopsis	3
Abstract	3
1 Introducere	4
1.1 Context	4
1.2 Problema	4
1.3 Obiective	5
1.4 Soluția propusă.....	5
2 Tehnologii software folosite	6
2.1 Python si Flask	6
2.2 MySQL	6
2.3 Docker.....	6
2.4 Kubernetes si Minikube	7
2.5 Gitlab.....	7
2.6 Prometheus.....	8
2.7 Grafana.....	8
3 Metode de upgrade.....	9
3.1 Recreate.....	9
3.2 Rolling Update	9
3.3 Blue/Green	10
3.4 Canary	12
3.5 A/B Testing.....	13
4 Proiectarea aplicatiei.....	14
4.1 Descriere generala.....	14
4.2 Inscrierea si Autentificare	16
4.3 Pagina principala.....	18
4.4 Pagina utilizatorului	18
4.5 Actualizare si stergere cont.....	19
4.6 Baza de date	19
5 Detalii arhitectura testare	21
5.1 Prezentare generala	21
5.2 Clusterul de Kubernetes	22
5.3 Gitlab.....	22

5.4	Aplicatia.....	23
5.5	Scriptul de load test.....	25
5.6	Prometheus.....	25
5.7	Grafana.....	26
6	Concluzii	27
6.1	Rezultatele testelor.....	27
6.2	Concluzii	29
7	Bibliografie	29
8	Anexe	30
8.1	Baza de date	30
8.2	Script de rulare.....	30
8.3	Script load-test	32

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 Kubernetes.

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 Kubernetes infrastructure.

1 INTRODUCERE

1.1 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, ideea de CI/CD este intalnita in majoritatea companiilor care dezvoltă 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. Viteza cu care companiile pot introduce produse pe piață este esențială pentru susținerea avantajului concurențial, iar reducerea timpului ciclului de dezvoltare a produsului a devenit un obiectiv strategic pentru multe firme care se bazează pe tehnologie. Conceptul de CI/CD explorează modul în care acest lucru poate fi realizat și oferă câteva exemple practice și factori de succes.

1.2 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)

1.3 Obiective

Scopul lucrării este acela de a analiza mai multe metode de upgrade în Kubernetes. Pentru aceasta, s-a creat o aplicație web, un cluster de Kubernetes unde aplicația rulează, scripturi de testare, scripturi de deployment, monitorizare. Fiecare metodă a fost analizată, descrise avantajele și dezavantajele. Bineînțeles, nici o variantă de upgrade nu este perfectă, fiecare având plusurile și minusurile respective, iar alegerea asupra unei metode trebuie să țină cont de specificul aplicației, gradul de disponibilitate al aplicației care se dorește, cost, etc.

1.4 Soluția propusă

Soluția propusă constă într-o infrastructură prin care au putut fi analizate mai multe metode de upgrade în Kubernetes. Aplicația pe care s-au analizat metodele de upgrade este o aplicație web care reprezintă un magazin online. Aplicația este scrisă în Python cu ajutorul framework-ului Flask. Ca stocare persistentă, aplicația folosește o bază de date relațională, MySQL. Clusterul de Kubernetes unde rulează aplicația este Minikube, rulează pe mașina locală. Pentru a testa în mod identic toate metodele de upgrade, a fost creată o suită de scripturi Bash care fac upgrade-ul, pentru a nu exista diferențe între metodele comparate. Pentru a testa metodele de upgrade, a fost creat un script de load test, scris în Python, care lansează în permanentă cereri HTTP către aplicație. Pentru tragerea concluziilor, a fost creată o infrastructură de monitorizare formată din Prometheus și Grafana, care au monitorizat requesturile lansate de load tester, și au desenat grafice care au putut fi ulterior analizate.

Kubernetes este un sistem de gestionare a containerelor, open-source, folosit pentru automatizarea implementării, scalării și distribuirii aplicațiilor. Acesta a fost inițial proiectat de Google și este acum un proiect întreținut de Cloud Native Computing Foundation. Scopul său este de a oferi o "platformă de automatizare a implementării, scalării și operării containerelor de aplicații în grupuri de computere". Funcționează cu o gamă largă de unelte pentru containerizare, inclusiv cu Docker.

2 TEHNOLOGII SOFTWARE FOLOSITE

2.1 Python si Flask

Python este un limbaj de programare foarte cunoscut. Acest limbaj de programare este folosit in vaste domenii datorit versatilitatii sale. Cateva exemple de domenii in care este folosit acest limbaj ar fi: Data Mining si Data Science, Inteligenta Artificiala si Machine Learning, Web Development, Automation Scripting si Network development. Python este un limbaj de programare usor de inteles, de citit si de invatat. Python a fost proiectat cu scopul principal de a oferi numai functionalitatile strict necesare. Sintaxa limbajului este una usor de scris si mentinut. Datorita acestor avantaje Python este deseori ales de dezvoltatorii de aplicatii atat din randul studentilor cat si din randul celor din industrie.

Flask este un micro framework web care are la baza Python. Flask nu are nevoie de tool-uri sau de biblioteci speciale. Spre exemplu acest micro framework nu are layer de baza de date sau validare de inputuri. De asemenea nu are componente care ofera functionalitati commune mentinute de librarii 3rd party. Cu toate acestea Flask suporta extensii care pot adauga functionalitatile lipsa. Integrarea extensiilor se face foarte usor. Astfel de extensii exista pentru urmatoarele functionalitati: object-relational mappers, validare se formulare, handling de upload etc.

Cateva exemple de aplicatii care folosesc Python + Flask sunt Pinterest si LinkedIn.

2.2 MySQL

MySQL este un tip de baze de date relationale inceput open-source si cumparat ulterior de Oracle. Este un sistem open source de baze de date relationale . Numele acestui sistem este o combinatie intre “My” si SQL. “My” vine de la numele fiicei cofondatorului Michael Widenius si “SQL” vine de la Structured Query Language. Este un software gratis care da posibilitatea de a interactiona cu o baza de date SQL. MySQL este folosit in combinatie cu alte programe si limbaje pentru a dezvolta aplicatii care au nevoie de un sistem de baza de date relationale. MySQL poate rula pe platforme de cloud computing cum ar fi Microsoft Azure, Amazon EC2 si Oracle Cloud Infrastructure. Exista cateva modele de deployment special pentru MySQL si acestea includ folosirea unei masini virtual sau folosirea MySQL ca serviciu.

2.3 Docker

Docker este o platforma pentru dezvoltare, livrare si rulare a aplicatiilor. Docker poate sa “impacheteze” si sa ruleze aplicatiile intr-un container. Un container reprezinta un mediu controlat, specific nevoilor fiecarei aplicatii. Containerele sunt configurate sa contina tot ce este nevoie pentru

rularea unei aplicatii si sunt “lightweight”. Un avantaj este ca pe un host pot sa ruleze mai multe containere.

Fiecare container reprezinta un mediu stabil si sigur in care aplicatia va rula. Acestea sunt foarte potrivite pentru flow-uri CI/CD (continuous delivery, continuous integration) pentru ca are la baza concepte de izolare. Un container are propriul sistem de fisere, spatiu pentru procese si stiva de retea. Un container porneste instant si este fiabil.

2.4 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.

Minikube este un instrument care face posibila instalarea unui cluster de Kubernetes pe sistemul local. Este special creat pentru dezvoltare de cod si testare de infrastructuri, instalarea acesuia fiind destul de usoara.

2.5 Gitlab

Gitlab este o platforma open source care ajuta la manage-uirea proiectelor si a codului. Gitlab a inceput ca un proiect open source care avea ca scop principal imbunatatirea colaborarii dintre echipe pentru dezvoltarea de software. Gitlab reprezinta locul unde mai multi oameni pot contribui la dezvoltarea de proiecte.

Gitlab este o aplicatie care ajuta la imbunatatirea ciclului de dezvoltare de cod. Gitlab ofera servicii de continuous integration pentru aplicatii. Aceste servicii ajuta la compilarea si testarea aplicatiilor. De fiecare data cand un developer comite cod procesul de compilare si testare va fi pornit. De asemenea Gitlab mai ofera si servicii de Continuous Deployment care presupun schimbarea si deployment-ul zilnic a codului pe diferite medii de lucru.

2.6 Prometheus

Prometheus este un instrument de monitorizare a alertelor si a evenimentelor unei aplicatii. Aceasta aplicatie permite implementarea de reguli si alerte pentru o mai buna monitorizare a sistemelor. Prometheus este un sistem gratis care stocheaza metrice “real-time ” intr-o baza de date orientate pe timp. Aceasta baza de date este construita pe modelule de HTTP pull cu query-uri flexible si alerte in timp real.

Prometheus este un proiect scris in Go si este licentiat sub Apache 2 License, iar codul sursa al proiectului se afla pe GitHub. Prometheus este format din mai multe tool-uri si anume multiple exportere care ruleaza pe host-uri monitorizate, manager de alerte, centralizator de metrice, Grafana pentru generarea de pagini de monitorizare si PromQL ca limbaj de query de date pentru alerte si monitorizare.

Prometheus nu a fost gandit ca o solutie de dashboarding. Din acest motiv este imperechiat cu Grafana pentru a genera pagini de monitorizare. Acest lucru poate fi vazut si ca un dezavantaj datorita complexitatii adaugate setup-ului.

2.7 Grafana

Grafana este o solutie open source pentru analizare si monitorizare a datelor unei sau mai multor aplicatii. Grafana ofera pagini de monitorizare customizabile si se poate conecta cu orice sursa de date posibile ca de exemplu Graphite, Prometheus, Influx DB, ElasticSearch, MySQL etc.

Acest instrument ajuta la studierea, analizarea si monitorizarea datelor pe parcursul unei perioade de timp. Ajuta sa urmarirea comportamentului aplicatiei, a utilizatorului, a sistemului si a erorilor prin oferirea de date relative.

Grafana poate fi deployata on-prem lucru care o face un mare avantaj pentru companiile care nu isi doresc ca datele sa ajunga in cloud din punct de vedere al securitatii.

Cu timpul Grafana a inceput sa castige popularitate in industrie si in prezent este folosita de PayPal Ebay Intel si multe alte companii

3 METODE DE UPGRADE

3.1 Recreate

Concept: Se sterg resursele corespunzatoare 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

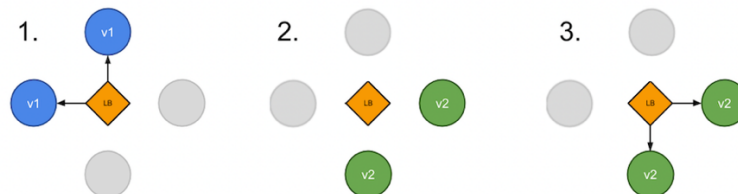


Fig. 1

3.2 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
- Neadekvata 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

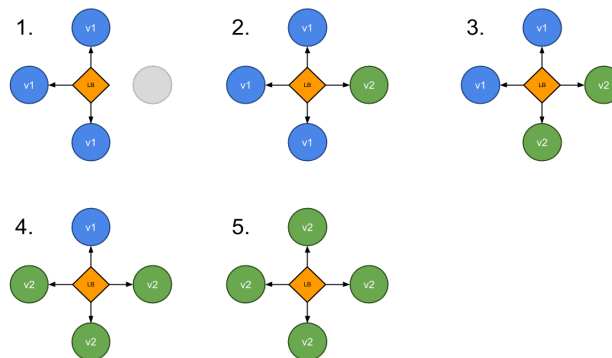


Fig. 2

3.3 Blue/Green

Concept: Inainte de aplicatia propriu zisa, exista 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.

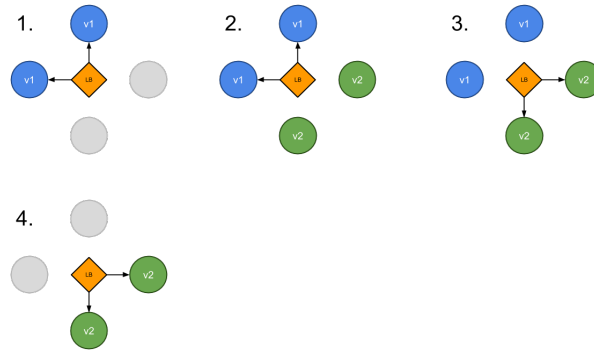


Fig. 3

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

3.4 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 maresc procentul, iar apoi versiunea canary se transforma in versiune de baza.

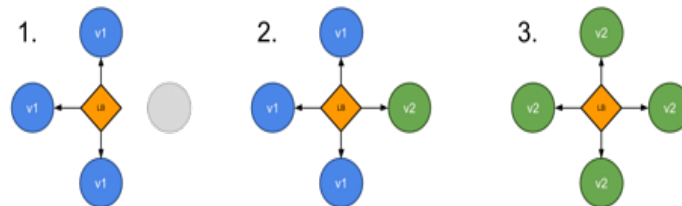


Fig. 4

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

3.5 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.

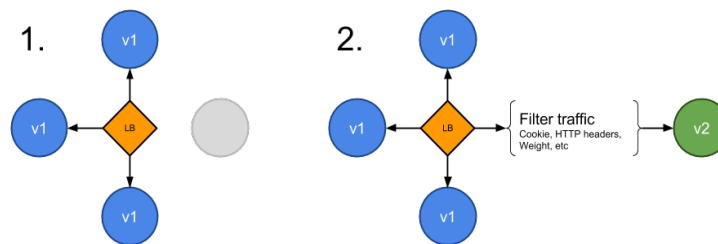


Fig. 5

Resurse suplimentare: Da. Este necesar cel puțin un pod în plus pentru noua versiune. Apoi se șterg poduri din versiunea veche și se adaugă poduri cu versiunea nouă.

Avantaje:

- Control total asupra distribuției de trafic
- Rollback rapid
- Noua versiune ajunge la un subset de utilizatori

Dezavantaje:

- Este nevoie de un loadbalancer inteligent
- Upgrade încet

Unde se poate folosi:

- Când se dorește analiza unei modificări a unei funcționalități
- Front-end-ul aplicațiilor

4 PROIECTAREA APLICATIEI

4.1 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 tabelor 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).

Diagrama site-ului este urmatoarea:

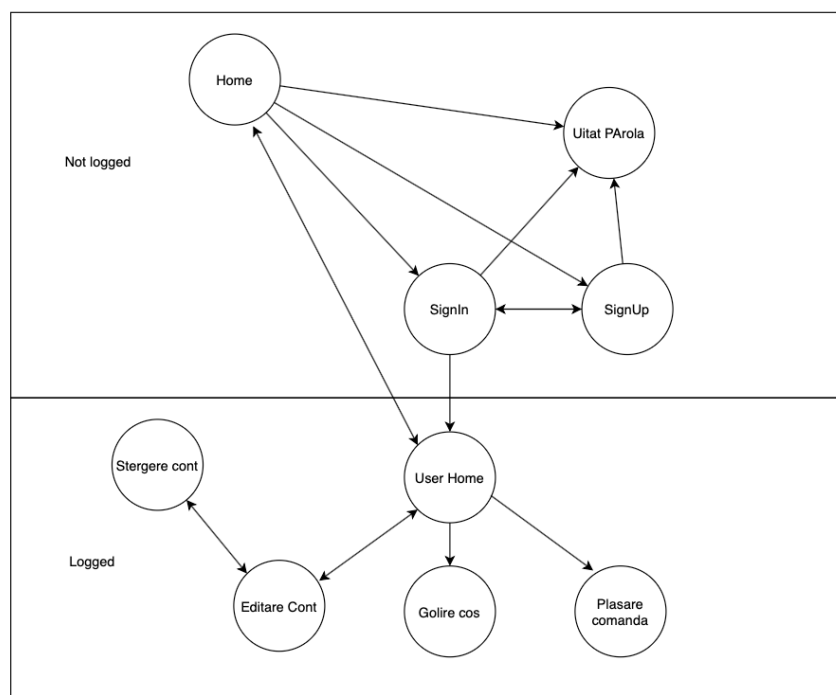


Fig. 6

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.

Paginile se impart in doua categorii:

a. 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.

b. 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.

4.2 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.

 The image shows a registration form titled "Inscriere" (Registration) overlaid on a background with colorful, abstract circular patterns. The form contains the following elements:

- A title "Inscriere" at the top.
- Input fields with labels and icons: "Email*" (with a person icon), "Nume" (with a document icon), "Prenume" (with a document icon), "Adresa" (with a location pin icon), "Parola*" (with a key icon), and "Confirmare Parola*" (with a key icon).
- A yellow button labeled "Inregistrare!" at the bottom right of the form.
- Below the button, two links: "Ai uitat parola?" (Forgot your password?) and "Ai deja cont? Conecteaza-te" (Already have an account? Log in).

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:



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 cu ajutorul functiei “generate_password_hash” din cadrul pachetului “werkzeug.security”, pachet care este folosit des in ecosistemul Python datorita gradului mare de securitate pe care il ofera.

```
mysql>
mysql> select * from useri where email="rizescueusebiu@gmail.com" \G
***** 1. row *****
  IDUser: 2
  numeUser: Rizescu
  prenumeUser: Eusebiu
  email: rizescueusebiu@gmail.com
  parola: pbkdf2:sha256:50000$seGFleoM$c78987a943a81ff40a61c892e5c0e87f8c3972e2ebb00e4531b713f68b1fe378
  adresa: Str. Opulentei, nr. 1
  shopping: SH03=3;AMTR01=1;KN04=12
  activated: 1
  hash: 02bf5a0d67f55b90cb28cdaaffec5814ae9ab068
1 row in set (0.00 sec)

mysql>
```

Fig. 9

4.3 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 comparaturi ar fi fost greyed out, si nu ar fi putut sa adauge in cos produsul dorit.

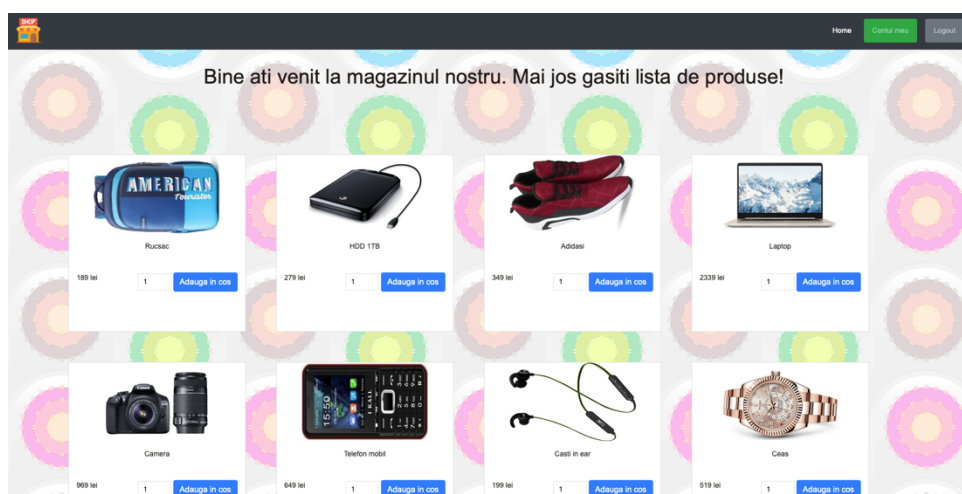


Fig. 10

4.4 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.

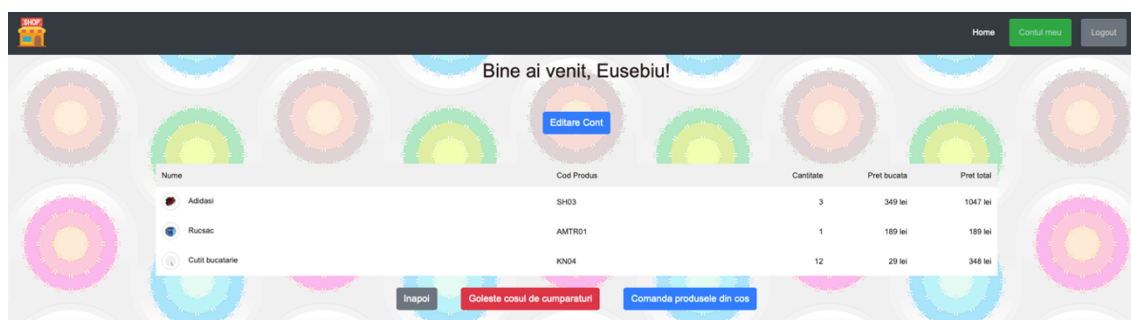


Fig. 11

4.5 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”.

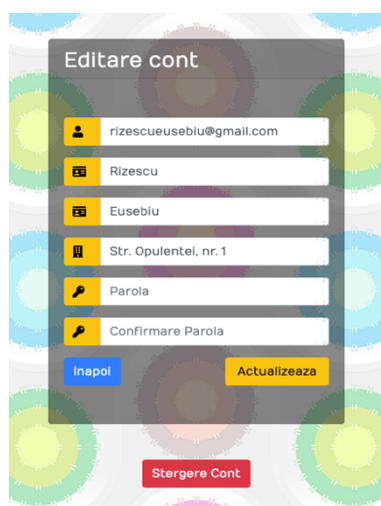


Fig. 12

4.6 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 tabelii 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”.

useri		produs	
IDUser	int(4)	IDProdus	int(4)
numeUser	varchar(20)	nume	varchar(50)
prenumeUser	varchar(20)	cod	varchar(20)
email	varchar(30)	imagine	varchar(100)
adresa	varchar(50)	pret	int(10)
parola	varchar(128)		
activated	bool		
hash	varchar(128)		
cosDeCumparaturi	varchar(300)		

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.

```

1  import os
2  from flask import Flask
3  from flask_sqlalchemy import SQLAlchemy
4  from flask_login import LoginManager
5
6  app = Flask(__name__)
7
8  app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+mysqlconnector://root:mypassword@dizertatie-service-db.canary
9  .svc.cluster.local/webapplication'
10 app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
11 app.config['SQLALCHEMY_POOL_RECYCLE'] = 3600
12
13 db = SQLAlchemy(app)
14

```

Fig. 14

5 DETALII ARHITECTURA TESTARE

5.1 Prezentare generala

Arhitectura este formata din mai multe componente, care vor fi detaliate in subcapitolele ce urmeaza (se poate observa in figura legaturile dintre acestea). 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 load test trimite in permanenta requesturi aplicatiei pentru a monitoriza proportia de requesturi intre cele 2 versiuni, si pentru a masura procentul de erori.

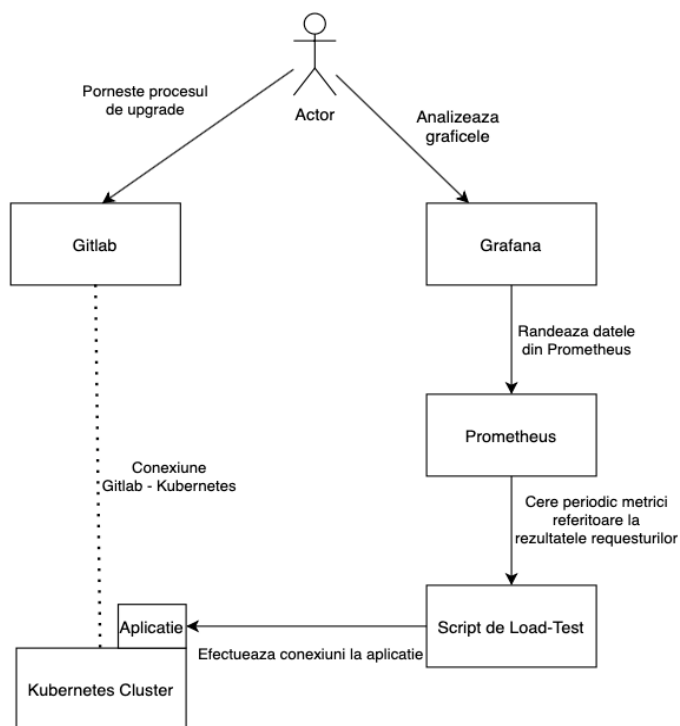


Fig. 15

5.2 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.

5.3 Gitlab

In Gitlab sunt tinute atat manifesturile de Kubernetes care descriu resursele si infrastructura aplicatiei, cat si pipeline-ul de creare, upgrade, stergere. El interactioneaza 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 acest 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 aplicatie:

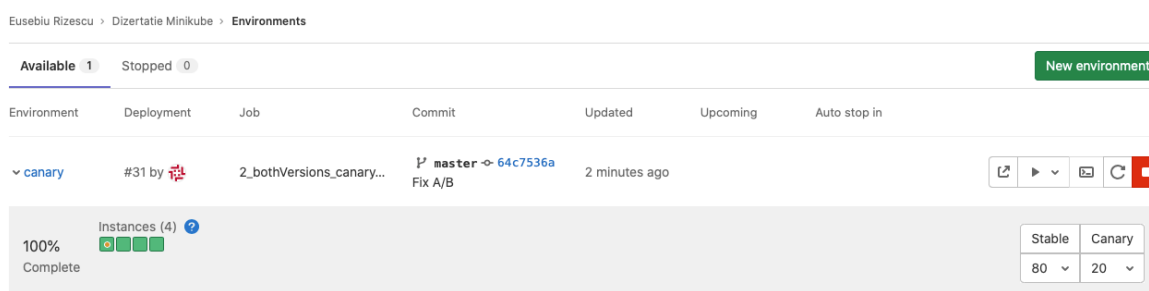


Fig. 16

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

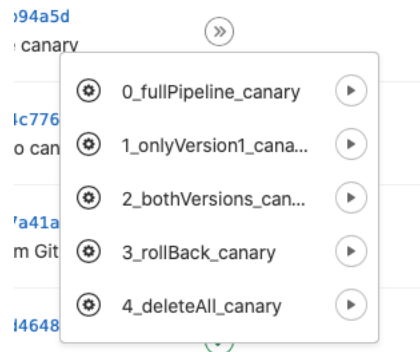


Fig. 17

5.4 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.

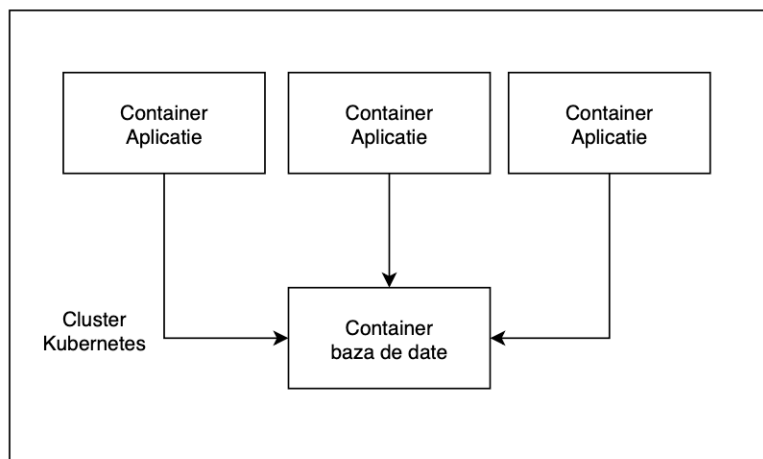


Fig. 18

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.

5.5 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 metrice: „requests200V1”, „requests500V1”, „requests200V2”, „requests500V2”.

Problema care apare este aceea de stocare a acestor metrice in functie de timp. Astfel, acest script de load-test expune si metrice (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 metrice, iar, cu ajutorul framework-ului Flask, scriptul le expune pe calea „/metrics”. Avand in vedere ca aceste metrice sunt crescatoare (countere), la fiecare request al Prometheus-ului catre scriptul de load test, metricele vor fi resetate. Cu aceasta abordare, in Prometheus vom avea metrice de forma „sum(requests) per 5 minute”.

5.6 Prometheus

Pentru a stoca istoria statusurilor requesturilor initiale de scriptul de load-test, se va folosi o instanta de Prometheus. Aceasta este configurata ca la fiecare 20 secunde sa colecteze metrice 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.

```

1 global:
2   scrape_interval: 20s
3   scrape_configs:
4     - job_name: 'dizertatie'
5       scrape_interval: 20s
6       metrics_path: /
7       static_configs:
8         - targets: ['192.168.100.100:7001']

```

Fig. 19

5.7 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).

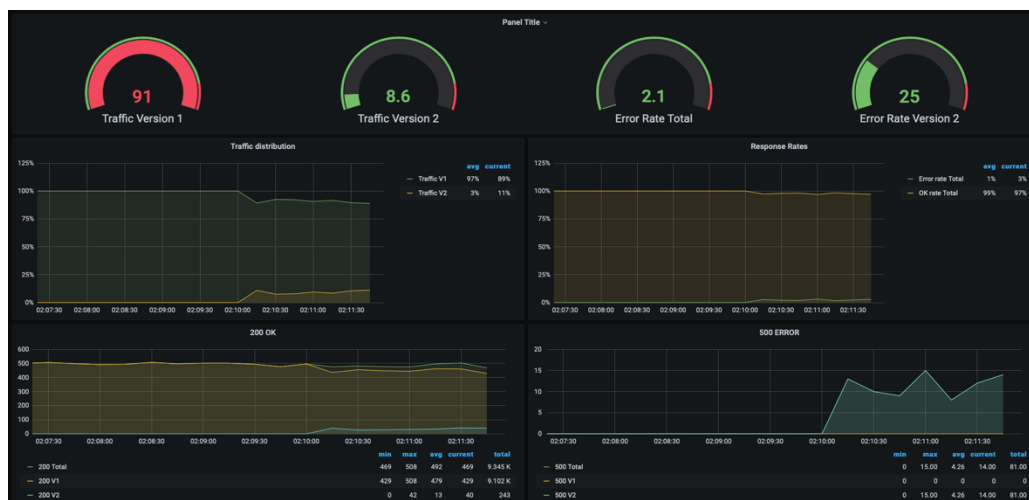


Fig. 20

6 CONCLUZII

6.1 Rezultatele testelor

Pentru fiecare varianta de deploy testele au constatat in:

- Namespace-ul de Kuberenetes nu contine nici o resursa
- Se porneste versiunea 1
- Nu se intervine 300 secunde
- Se lanseaza versiunea 2
- Nu se intervine 300 secunde
- Se face rollback la versiunea 1

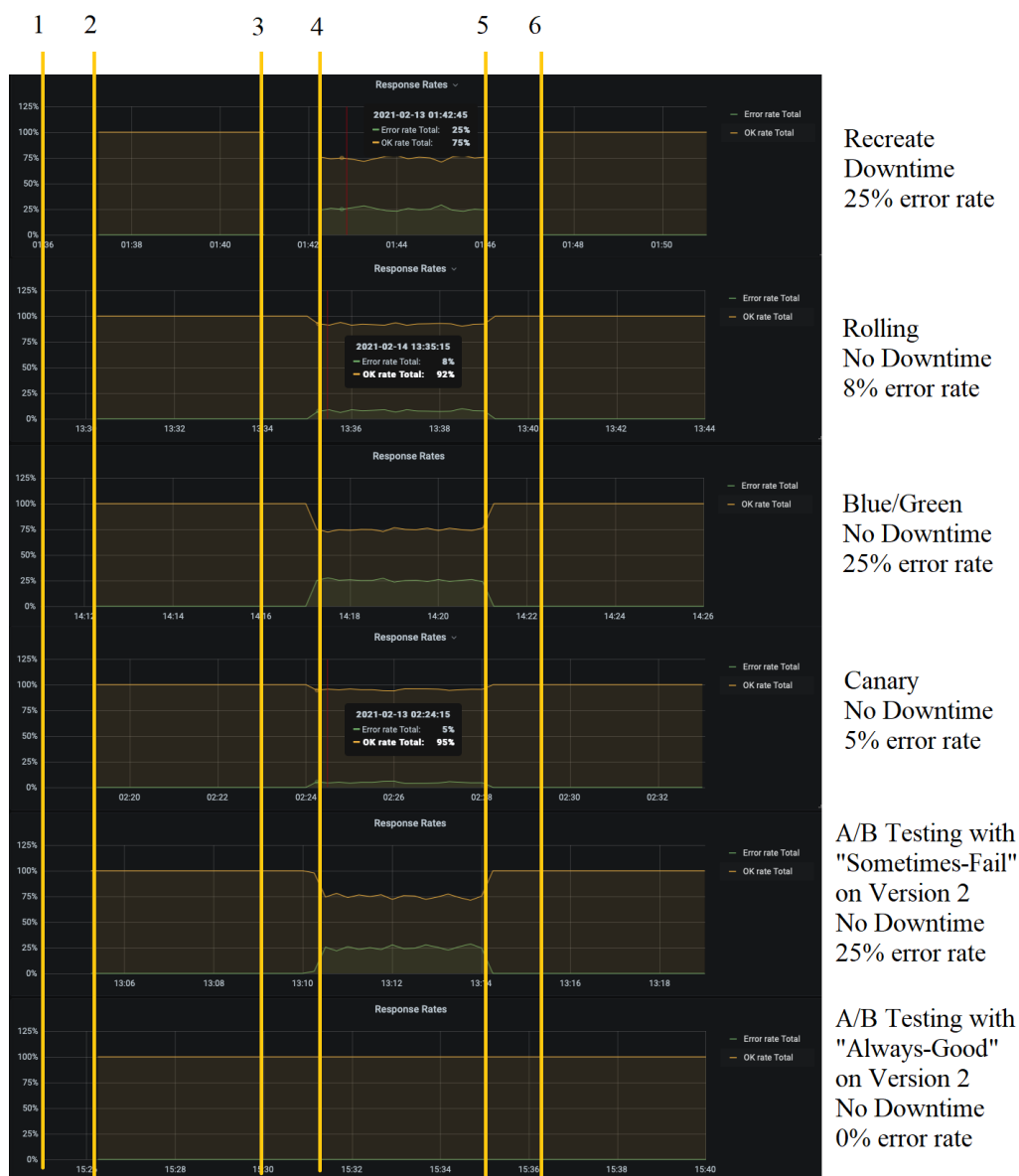
Metoda/Parametru	Downtime la upgrade	Rata de eroare dupa upgrade	Downtime la downgrade	Resurse in plus necesare
Recreate	Da	25%	Da	Nu
Rolling update	Nu	8%	Nu	+33%, podul cu versiunea noua
Blue/Green	Nu	25%	Nu	+100%
Canary (20%)	Nu	5%	Nu	+33%, podul pentru canary
A/B testing (Always-Good pe Versiunea 2)	Nu	25%	Nu	+100%
A/B testing (Sometimes-Fail pe Versiunea 2)	Nu	0%	Nu	+100%

Fig. 21

Reamintim cum este configurata aplicatia:

- Un pod nou de aplicatie, in primele 60 secunde nu serveste requesturi
- Versiunea 1 raspunde numai cu OK
- Versiunea 2 raspunde pentru user-agent-ul „Sometimes-Fail” cu 50% probabilitate ERROR
- Aplicatia ruleaza pe 3 poduri (acest lucru conteaza la calcularea procentului de ERROR)
- Scriptul de load-test lanseaza requesturi cu 2 useri agenti: „Always-Good” si „Sometimes-Fail” in proportii egale

In figura de mai jos, se poate observa cum se comporta fiecare metoda de upgrade:



Timpi:

- 1 (t0) - Pornire versiunea 1
- 2 (t0 + 60s) - Versiunea 1 este live
- 3 (t0 + 300s) - Upgrade la versiunea 2
- 4 (t0 + 360s) - Versiunea 2 este live
- 5 (t0 + 600s) - Rollback (in afara de Recreate, la toate celelalte trecera se face instant)
- 6 (t0 + 660s) - Versiunea 1 este live (la Recreate)

Fig. 22

6.2 Concluzii

Din pacate, nu exista o solutie de upgrade care sa se potriveasca fiecarei aplicatii. Este necesar ca inainte de a alege o varianta, aceasta sa fie inteleasa foarte bine, dar si sa se analizeze si celelalte variante existente. In plus, este important ca echipele de developeri si de operations sa lucreze impreuna pentru a alege varianta corecta pentru aplicatia in cauza. In acest document au fost prezentate si testate diferite tehnici individual, dar acestea pot fi combinate in functie de cerinte, pentru a obtine o solutie adecvata aplicatiei si infrastructurii existente.

7 BIBLIOGRAFIE

[1] Continuous Delivery - Reliable Software Releases Through Build, Test And Deployment Automation – Humble J., Farley D.

[2] <https://kubernetes.io>

[3] <https://blog.container-solutions.com/kubernetes-deployment-strategies>

[4] <https://medium.com/google-cloud/kubernetes-canary-deployments-for-mere-mortals13728ce032fe>

[5] <https://www.udemy.com/course/python-and-flask-bootcamp-create-websites-using-flask>

[6] https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

[7] <https://werkzeug.palletsprojects.com/en/2.0.x/utils/>

[8] <https://flask.palletsprojects.com/en/2.0.x/>

[9] <https://www.oracle.com/ro/cloud-native/container-registry/what-is-docker/>

[10] <https://about.gitlab.com/what-is-gitlab/>

8 ANEXE

8.1 Baza de date

Baza de date este formata din doua tabele, “useri” si “produse”. Ambele tabele folosesc ca cheie primara un ID, care este incrementat automat la adaugarea unei noi intrari in cadrul tabelui.

```
mysql>
mysql> show tables;
+-----+
| Tables_in_webapplication |
+-----+
| produse                    |
| useri                      |
+-----+
2 rows in set (0.00 sec)

mysql>
mysql> desc useri;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| IDUser | int(11) | NO | PRI | NULL | auto_increment |
| numeUser | varchar(30) | NO | | NULL | |
| prenumeUser | varchar(30) | NO | | NULL | |
| email | varchar(30) | NO | | NULL | |
| parola | varchar(128) | YES | | NULL | |
| adresa | varchar(30) | YES | | NULL | |
| shopping | varchar(300) | YES | | NULL | |
| activated | tinyint(1) | YES | | NULL | |
| hash | varchar(50) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
9 rows in set (0.00 sec)

mysql>
mysql> desc produse;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| IDProdus | int(11) | NO | PRI | NULL | auto_increment |
| nume | varchar(300) | NO | | NULL | |
| cod | varchar(300) | NO | | NULL | |
| imagine | varchar(300) | NO | | NULL | |
| pret | int(11) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)

mysql>
```

Fig. 23

8.2 Script de rulare

In timpul dezvoltarii aplicatiei, pentru deploy-ul aplicatiei am folosit docker-compose pentru o mai usoara gestiune a containerelor. Stackul actual este format din doua servicii, si anume baza de date (care ruleaza si ea in Docker) si aplicatia. Pentru ambele servicii am configurat implicit volume cu nume, deoarece durata spatiului de stocare in docker este efemera (la oprirea containerului, datele nesalvate intr-un volum sunt pierdute).

pentru a imi fi usurata munca de actualizare a noului cod, am folosit urmatorul script. Acest script opreste containerele actuale de docker, copiaza noul cod, si reporneste containrele.

```

#!/bin/bash

if [ "$1" == "help" ]; then
    echo "./build.sh help      <-  help"
    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
    docker exec $(docker ps | grep flask | cut -d " " -f1 | head -1) python3
    populateTables.py
    exit
fi

echo Exit Docker Swarm
sudo docker swarm leave --force
sleep 1
if [ "$1" != "stop" ]; then
    echo Init Docker Swarm
    sudo docker swarm init
fi

if [ "$1" != "stop" ]; then
    echo "Removing /Users/eusebiu.rizescu/dizertatie/flask/*"
    sudo rm -rf /Users/eusebiu.rizescu/dizertatie/flask/*
    echo "Removing /Users/eusebiu.rizescu/dizertatie/mysql-conf/*"
    sudo rm -rf /Users/eusebiu.rizescu/dizertatie/mysql-conf/*

    echo "Copying to /Users/eusebiu.rizescu/dizertatie/flask/*"
    sudo cp -R ./WebApplicationCode/*
/Users/eusebiu.rizescu/dizertatie/flask/
    echo "Copying to /Users/eusebiu.rizescu/dizertatie/mysql-conf/*"
    sudo cp -R ./MySQL/* /Users/eusebiu.rizescu/dizertatie/mysql-conf/

    sudo docker stack deploy -c docker-compose.yml webapplication
fi

if [ "$1" == "stop" ]; then
    # delete all containers
    echo "Delete all containers"
    sudo docker rm -f $(docker ps -a -q)
fi

```

Fig. 24

8.3 Script load-test

```
import time #for sleep
import json #for JSON manipulation
import requests #for requests launching
import threading #for threading purposes
from flask import Flask #used for exposing metrics

# Variables

destinationIP = "http://192.168.64.2"
userAgent1 = "Always-Good"
userAgent2 = "Sometimes-Fail"
hostnameToUse = "dizertatie.com"
portToExposeMetrics = 7001

# Flask app
app = Flask(__name__)

# Global counters
requests200V1 = 0
requests500V1 = 0
requests200V2 = 0
requests500V2 = 0

# Load test function
def loadTest():
    global requests200V1
    global requests500V1
    global requests200V2
    global requests500V2

    while True:
        try:
            # User Agent 1
            headers = {'User-Agent': userAgent1, 'Host': hostnameToUse}
            resp = requests.get(destinationIP, headers=headers).json()
            if resp["appVersion"] == "1" and resp["status"] == "200":
                requests200V1 += 1
            if resp["appVersion"] == "1" and resp["status"] == "500":
                requests500V1 += 1
            if resp["appVersion"] == "2" and resp["status"] == "200":
                requests200V2 += 1
            if resp["appVersion"] == "2" and resp["status"] == "500":
                requests500V2 += 1
```

```

# User Agent 2
headers = {'User-Agent': userAgent2, 'Host': hostnameToUse}
resp = requests.get(destinationIP, headers=headers).json()
if resp["appVersion"] == "1" and resp["status"] == "200":
    requests200V1 += 1
if resp["appVersion"] == "1" and resp["status"] == "500":
    requests500V1 += 1
if resp["appVersion"] == "2" and resp["status"] == "200":
    requests200V2 += 1
if resp["appVersion"] == "2" and resp["status"] == "500":
    requests500V2 += 1
except:
    pass
time.sleep(0.01)

# Page that is queried by Prometheus to get metrics
@app.route('/')
def index():
    global requests200V1
    global requests500V1
    global requests200V2
    global requests500V2

    # Compose return string
    stringToReturn = ""
    stringToReturn += "requests200V1 " + str(requests200V1) + "\n"
    stringToReturn += "requests500V1 " + str(requests500V1) + "\n"
    stringToReturn += "requests200V2 " + str(requests200V2) + "\n"
    stringToReturn += "requests500V2 " + str(requests500V2) + "\n"

    # Reset the counters
    requests200V1 = 0
    requests500V1 = 0
    requests200V2 = 0
    requests500V2 = 0

    return stringToReturn

# Main
if __name__ == "__main__":
    loadTestThread = threading.Thread(target=loadTest, name="loadTester", args=())
    loadTestThread.start()
    app.run(host='0.0.0.0', port=portToExposeMetrics)

```

Fig. 25