LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

DEVOPS METODOLOĢIJAS UN RĪKU IZMANTOŠANA IT SISTĒMU IZSTRĀDĒ

BAKALAURA DARBS

Autors: **Ilmārs Vindačs**
Studenta apliecības Nr.: iv17041
Darba vadītājs: M.dat. Jānis Vempers

RĪGA 2021

ANOTĀCIJA

Bakalaura darba mērķis ir izpētīt izstrāddarbināšanas metodoloģijas sniegtās priekšrocības IT sistēmu izstrādē, kā arī apskatīt un salīdzināt izplatītākos ar izstrāddarbināšanas pieeju saistītos rīkus. Tēma ir aktuāla, jo izstrāddarbināšanas prakšu pielietošanas izplatība kļūst aizvien straujāka, un līdz ar šo izaugsmi kļūst pieejami ne tikai jauni risinājumi, bet arī turpina attīstīties jau esošie rīki.

Bakalaura darba ietvaros tiks apskatīta izstrāddarbināšanas metodoloģija un galvenās atšķirības starp izstrāddarbināšanu un spējās programmatūras izstrādes pieejām, kā arī apskatīti izplatītākie rīki, kas palīdz realizēt izstrāddarbināšanas pieeju.

Bakalaura darba noslēgumā, balstoties uz veikto izpēti, tiek rekomendēta iespējama optimāla izstrāddarbināšanas rīku kopa, kas vislabāk spētu nodrošināt nepārtrauktās izstrādes un piegādes procesus.

Atslēgvārdi: izstrāddarbināšana, spējā izstrāde, nepārtraukta izstrāde, nepārtraukta piegāde

ABSTRACT

USE OF DEVOPS METHODOLOGY AND TOOLS IN THE DEVELOPMENT OF IT SYSTEMS

The aim of the bachelor's thesis is to study the advantages of DevOps methodology in the development of IT systems, as well as to review and compare the most common tools related to the DevOps approach. The topic is relevant, as the spread of development practices is becoming more rapid, and with this growth, not only new solutions become available, but also existing tools continue to develop.

The bachelor's thesis will examine the DevOps methodology and the main differences between DevOps and Agile software development approaches, as well as review the most common tools that help to implement the DevOps approach.

At the end of the bachelor's thesis, based on the performed research a possible optimal chain of DevOps tools is recommended, which would be best able to ensure continuous development and delivery processes.

Keywords: DevOps, Agile, continuous development, continuous delivery

Saturs

Apzīmējumu saraksts

Jēdziens	Skaidrojums
Agile	Spējā programmatūras izstrāde. [1]
DevOps	Izstrāddarbināšana[1]
Dēmons	Fona programma, kas vajadzības gadījumā tiek izmantota datoros, kuri darbojas UNIX vai kādas citas operētājsistēmas vidē. Dēmons veic tādus pakalpojumus kā, piem., elektroniskā pasta ziņojumu maršrutēšanu, un tas parasti darbojas bez lietotāja iejaukšanās. [1]
IT	Informācijas tehnoloģija
Mākonis	Kāda iepriekš nenoteikta tīkla daļa, pa kuru dati tiek pārsūtīti, izmantojot, piem., kādu no protokolu saimes TCP/IP bezsavienojumu protokoliem. Tā kā mākoņi eksistē tāpēc, ka datu apmaiņa bezsavienojuma režīmā var notikt pa daudziem iespējamiem ceļiem, arī internets no datu nosūtītāja un datu saņēmēja viedokļa var tikt uzskatīts par mākoni.[1]

Ievads

Bakalaura darba mērķis ir izpētīt izstrāddarbināšanas metodoloģijas sniegtās priekšrocības IT sistēmu izstrādē, kā arī apskatīt un salīdzināt izplatītākos ar izstrāddarbināšanas pieeju saistītos rīkus ar nolūku sastādīt funkcionālu izstrāddarbināšanas rīku kopu, ko būtu iespējams izmantot jaunas tīmeklī bāzētas IT sistēmas veidošanā.

Lai sasniegtu uzstādīto mērķi bakalaura darba pirmajā daļā tiks apskatītas izstrāddarbināšanas pamatnostādnes, darbplūsma kā arī galvenās pieejas. Balstoties uz šo informāciju darba otrajā daļā tiks apskatīti autora izvēlēti katra izstrāddarbināšanas procesa posma rīki. Ir paredzēts apskatīt gan atvērta pirmkoda rīkus, gan arī maksas risinājumus, priekšroku dodot izplatītākajiem rīkiem, kā arī rīkiem, kas parādījušies nesen, bet jau ir izpelnījušies lietotāju atzinību. Apskatot konkrētos rīkus īpaša uzmanība tiks pievērsta galvenajām priekšrocībām, trūkumiem, kā arī cenai. Šie parametri būs noteicošie sastādot optimālu izstrāddarbināšanas principiem atbilstošu rīku kopu.

1. Izstrāddarbināšanas metodoloģija

1.1. Metodoloģijas vēsture

Izstrāddarbināšanas kustības pirmsākumi meklējami laika posmā starp 2007. un 2008. gadiem, kad IT operāciju un programmētāju kopienas pamazām sāka iebilst pret tradicionālo programmatūras izstrādes modeli, kas aicināja tos, kuri raksta kodu, organizatoriski un funkcionāli nodalīt no tiem, kas kodu uzstāda un uztur tā darbību.

Izstrāddarbināšanas koncepcijas nosaukums radās pateicoties DevOpsDay konferencei, kuru organizēja Patriks Deboī (*angl*. Patrick Debois) sadarbībā ar Endrjū Kleju (*angl.* Andrew Clay). Viņi bija noraizējušies par Agile trūkumiem un vēlējās nākt klajā ar kaut ko labāku. Šis pasākums, kas pulcēja gan izstrādātājus, gan IT operāciju pārstāvjus, drīzumā izpelnījās pietiekami lielu abu jomu ekspertu uzmanību un izraisīja dzīvas diskusijas vietnē Twitter, kur tēmturi drīz saīsināja līdz vienkārši DevOps.

Drīz vien parādījās IT nozares uzņēmumi, kas mēģināja ieviest izstrāddarbināšanu praksē, kā arī izveidot rīkus, kas palīdzētu realizēt jaunos mērķus. Visbeidzot 2011. gadā izstrāddarbināšanai uzmanību pievērsa ietekmīgā uzņēmuma Gartner analītiķis Kamerons Haits (Cameron Haight), kura pozitīvās prognozēs par izstrāddarbināšanu izraisīja pastiprinātu IT nozares interesi par jauno metodoloģiju. Rezultātā arī lielie IT uzņēmumi sāka ieviest savā darbībā izstrāddarbināšanas pieeju, padarot to par nākamo lielo lietu kopš spējas izstrādes.

Šo vēsturi ir nozīmīgi saprast, jo izstrāddarbināšanas metodoloģija ir radusies, izstrādātājiem un IT sistēmu administratoriem sanākot kopā, lai paustu savas idejas un bažas par nozari un to, kā vislabāk un visātrāk paveikt darbus, un tā ir idejas galvenā priekšrocība, jo tā ir radusies no cilvēkiem, kuriem tā ir paredzēta, lai palīdzētu.[2]

1.2. Izstrāddarbināšanas pamatnostādnes

Izstrāddarbināšanai nav vienas noteicošās definīcijas, taču populārākie skaidrojumi ir:

- Izstrāddarbināšana ir administratori, kas domā kā izstrādātāji un izstrādātāji, kas domā kā administratori.
- Izstrāddarbināšana ir pieeja, kur operāciju un izstrādes speciālisti kopīgi piedalās visā produkta dzīves ciklā, sākot no projektēšanas līdz izstrādes procesam un beidzot uzturēšanu.
- Izstrāddarbināšana ir kultūra, kustība vai prakse, kas uzsver gan programmatūras izstrādātāju, gan citu IT profesionāļu sadarbību un komunikāciju, vienlaikus automatizējot programmatūras piegādes procesu un infrastruktūras izmaiņas.

No pieejamajiem skaidrojumiem visprecīzākais šķiet pēdējais, jo lai gan izstrāddarbināšana ir praktiska metodoloģija, tā būtībā iekļauj sevī arī noteiktu domāšanas veidu un darba kultūru.

Galvenie principi, pēc kuriem vadās ir:

- **Automatizācija:** pēc iespējas automatizēt visu, piemēram, darbplūsmas, jaunā koda testēšanu, jauna koda piegādi un to, kā tiek nodrošināta infrastruktūra, lai palielinātu produktivitāti un samazinātu pārmērīgu darbu.
- **Iterēšana***(angl. iteration)***:** Izstrādi veikt ar sprintu palīdzību, rakstot mazus koda gabalus un veicot biežus laidienus.
- **Nepārtraukta uzlabošana:** nepārtraukta testēšana un mācīšanās no neveiksmēm un atgriezeniskās saites, lai optimizētu veiktspēju, izmaksas un laiku līdz piegādei.
- **Sadarbība:** apvienot komandas, veicināt saziņu un nojaukt barjeras starp izstrādi, IT operācijām un kvalitātes nodrošināšanu.[3]

Salīdzinājumā ar spējo izstrādi izstrāddarbināšana to drīzāk papildina nevis aizstāj, paplašinot to ar IT operācijām, kā arī ievērojami vairāk koncentrējoties uz automatizāciju un rīkiem, kas palīdz uzlabot koda piegādi. Automatizācijas galvenā priekšrocība ir, ka tā ļauj izstrādātājiem un sistēmu administratoriem apvienot savu darbu vienā procesā nodrošinot nepārtrauktu integrāciju un piegādi.

Līdzīgi kā spējajā izstrāde izstrāddarbināšanā par prioritāti uzskata nelielus iteratīvus procesus, kas ļauj veikt regulāru testēšanu un potenciālu kļūdu izķeršanu agri izstrādes laikā, ne tikai samazinot izmaksas un uzlabojot gala produkta kvalitāti, bet arī samazinot izstrādei veltīto laiku.

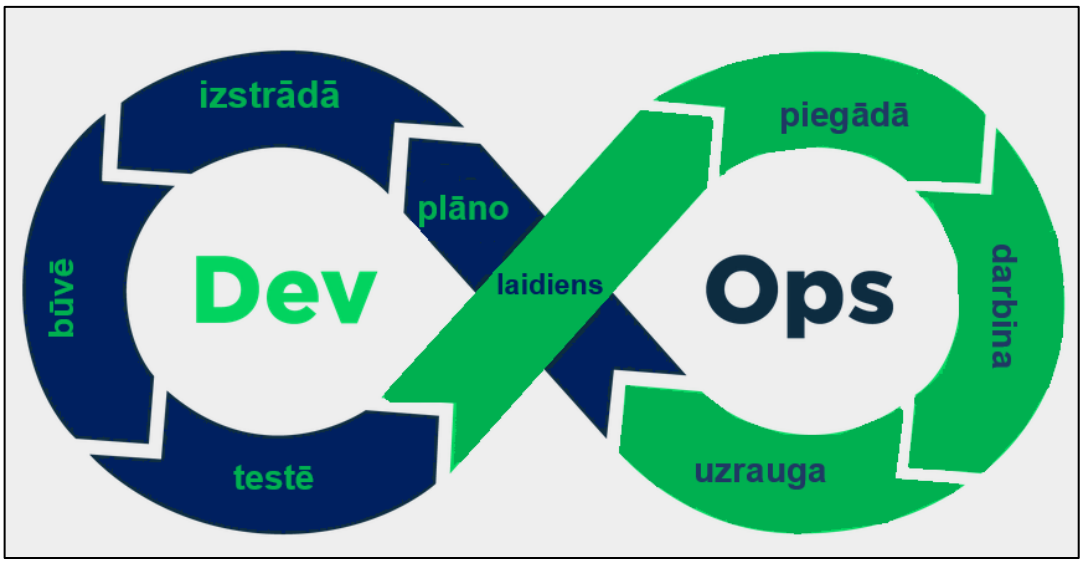
Kopsavilkumā galvenie ieguvumi no izstrāddarbināšanas pieejas izmantošanas ir:

- Īsāks izstrādes laiks
- Augstāka klientu apmierinātība
- Uzlabota sadarbība starp projektā iesaistītajiem darbiniekiem
- Agrīna kļūdu atrašana un izlabošana
- Optimizētas izmaksas projekta piegādes laikā

1.3. Izstrāddarbināšanas darbplūsma

Projektiem, kuru realizācijā tiek izmantota izstrāddarbināšanas pieeja, ir raksturīgs attēlā 1.1 redzamajā darbplūsma, kas sastāv no septiņiem pamatelementiem[5], kur pa vidu izstrādes un operāciju posmiem ir jaunas versijas laidiens:

- **Plānošana:** Šis solis palīdz noteikt biznesa mērķus un produkta prasības. Izplatītākie rīki, kas tiek izmantoti, lai palīdzētu ar šiem uzdevumiem ir Jira, Redmine u.c.
- **Izstrāde:** Šis solis ietver sevī programmatūras koda izstrādi, kā arī izstrādātā pirmkoda versionēšanu, izmantojot tādus rīkus kā GitHub, Bitbucket u.c.
- **Būvēšana:** Šajā posmā tiek veidoti programmatūras būvējumi un versijas, izmantojot automatizētus rīkus, lai palīdzētu apkopot un iepakot kodu turpmākajai izlaišanai produkcijā. Šajā solī arī tiek izmantota konteinerizācija, kas "iesaiņo" ne tikai pirmkodu, bet arī produkta izlaišanai nepieciešamo infrastruktūru.
- **Testēšana:** Šis posms ietver nepārtraukto testēšanu produkta dzīvesciklā, izmantojot automātiskos testus, lai uzlabotu koda kvalitāti. Atkarībā no projekta, šis posms iekļauj tādus rīkus kā Selenium, Junit, TestNG u.c.
- **Piegāde:** Šajā posmā tiek pārvaldīta, plānota un automatizēta produktu izlaišana produkcijā. Rīku piemēri iekļauj Drone, Jenkins u.c.
- **Darbināšana:** Šajā posmā produkcijas vidē tiek pārvaldīta uzstādītā programmatūra. Rīku piemēri iekļauj Kubernetes un Docker Swarm u.c.
- **Uzraudzība:** Šajā posmā identificē un apkopo informāciju par problēmām no konkrēta produkcijas programmatūras laidiena. Rīku paraugi ir Zabbix un Nagios XI u.c.



1.1. attēls Izstrāddarbināšanas darbplūsmas modelis[4]

1.4. Izstrāddarbināšanas pieejas

Lai veiksmīgi realizētu izstrāddarbināšanas darbplūsmu atbilstoši izstrāddarbināšanas principiem laika gaitā radušās vairākas pieejas, katrai no kurām ir radīti pielāgoti rīki, kas tiks sīkāk apskatīti un salīdzināti bakalaura darba 2. nodaļā. Šie rīki un pieejas ne vienmēr atbilst vienam konkrētam posmam izstrāddarbināšanas darbplūsmā, bet bieži vien uzreiz vairākiem.

Izplatītākās izstrāddarbināšanas pieejas:

- **Nepārtraukta izstrāde (***angl. continuous development*): apraksta procesu, kas ietver sevī nepārtraukto testēšanu, nepārtraukto integrēšanu, nepārtraukto piegādi un nepārtraukto izvēršanu.
- **Nepārtraukta testēšana** (*angl. continuous testing*): Šī pieeja ietver sevī automatizētus, regulārus testus. Tie var būt gan funkcionālie testi, gan integrācijas un vienībtesti, bet visbiežāk ir dažādu testu sajaukums. Automatizētā testēšana palīdz daudz ātrāk piegādāt koda izmaiņas, jo ietaupa laiku uz manuālās testēšanas rēķina, taču mīnuss ir tas, ka testi vispirms ir jāizstrādā, kā arī regulāri jāuztur, līdz ar to visvairāk ieguvumu no šīs pieejas var izjust projekta izstrādes beigās, kā arī uzturēšanas fāzē.
- **Nepārtraukta integrēšana** *(angl. continuous integration)*: Šī pieeja apvieno konfigurācijas pārvaldības (CM) rīkus ar citiem testēšanas un izstrādes rīkiem, lai izsekotu, cik liela daļa no izstrādātā koda ir gatava produkcijai. Tas ietver ātru atgriezenisko saiti starp testēšanu un izstrādi, lai ātri identificētu un atrisinātu koda problēmas.
- **Nepārtraukta piegāde** (*angl. continuous delivery*): Šī pieeja automatizē koda izmaiņu piegādi pēc testēšanas testa vidē. Šajā scenārijā darbinieks pieņem lēmumu vai uzstādīt jaunās izmaiņas produkcijā.
- **Nepārtraukta izvēršana** (*angl. continuous deployment*): Līdzīgi nepārtrauktajai piegādei, taču šī pieeja pilnībā automatizē jauna vai mainīta koda piegādi un uzstādīšanu produkcijas vidē. Uzņēmums, kas izmanto šādu pieeju var izlaist funkciju izmaiņas vairākas reizes dienā. Tomēr šāda pieeja ir ieteicama tikai gadījumos, kad produktam ir veikti aptveroši automatizētās testēšana, kā arī pārraudzīšans pasākumi. Nodrošināt nepārtrauktu izvēršanu, palīdzot saglabāt koda konsekvenci dažādās vidēs, var izmantojot konteineru tehnoloģijas, piemēram, Docker un Kubernetes.

- **Nepārtraukta pārraudzīšana** (*angl. continuous monitoring*): pieeja paredz nepārtrauktu gan darbībā esošā koda, gan saistītās IT infrastruktūras pārraudzību. Pieeja paredz arī atgriezenisko saiti, kas ziņo par kļūdām vai problēmām, nododot nepieciešamos izmaiņu pieprasījumus plānošanas fāzei.

2. Rīki izstrāddarbināšanas realizācijai

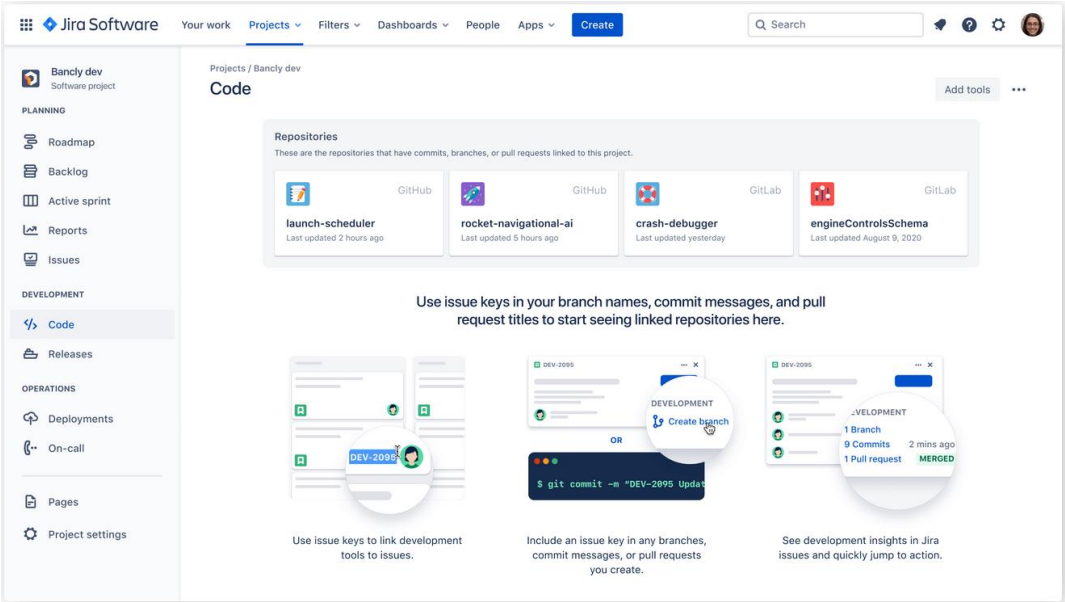
Lai varētu savā IT sistēmā ieviest izstrāddarbināšanas pieejas, pastāv ļoti liels iespējamo rīku un to kombināciju skaits, tāpēc šī darba ietvaros pārsvarā tiks apskatīti tikai izplatītākie risinājumi, kas sevi ir pierādījuši. Apskatot izstrāddarbināšanas pieejas realizācijai paredzētos rīkus, tiks apskatīts vispārējs katra produkta apraksts, tā galvenās iespējas kā arī rīka priekšrocības, trūkumi un cena.

2.1. Plānošanas posma rīki

Plānošanas posma mērķis ir noteikt biznesa mērķus un produkta prasības, kā arī nodrošināt projekta pārvaldību un skaidru darba uzdevumu infrastruktūru, kas palīdzētu visiem iesaistītajiem darbiniekiem skaidri orientēties darāmajos darbos. Šī darba ietvaros tiks apskatīti divi potenciāli risinājumi – Jira(maksas risinājums) un Redmine (atvērta pirmkoda risinājums).

2.1.1. Jira

Jira programmatūra ir daļa no produktu grupas, kas paredzēta visu veidu komandām, lai palīdzētu vadīt darbu. Sākotnēji Jira tika veidota kā kļūdu un problēmu izsekotājs. Bet šodien Jira ir pārtapis par spēcīgu darba pārvaldības rīku visdažādākajiem lietošanas gadījumiem, sākot no prasībām un testa lietu pārvaldības līdz veiklai programmatūras izstrādei.[6] Ar gandrīz 40% no projektu pārvaldības rīku tirgus daļu Jira ir visbiežāk izmantotais šīs kategorijas risinājums.[7]



2.1.1. attēls Jira saskarne

Priekšrocības:

- Lieliski piemērots spējās izstrādes pieejai. Jira nodrošina vienotu skatu visiem lietotāju stāstiem un ļauj ģenerēt nepieciešamos pārskatus dažādiem sprintiem. Tāpat lietotāji var organizēt reģistrētos pieteikumos sprintos un laidienos (attēls 2.1.1).
- Plašas integrācijas iespējas ar koda kontroles risinājumiem.
- Risinājums atbalsta līdz pat 10 000 lietotāju.
- Pieejami vairāk nekā 1000 spraudņi, kas uzreiz gatavi izmantošanai

Trūkumi:

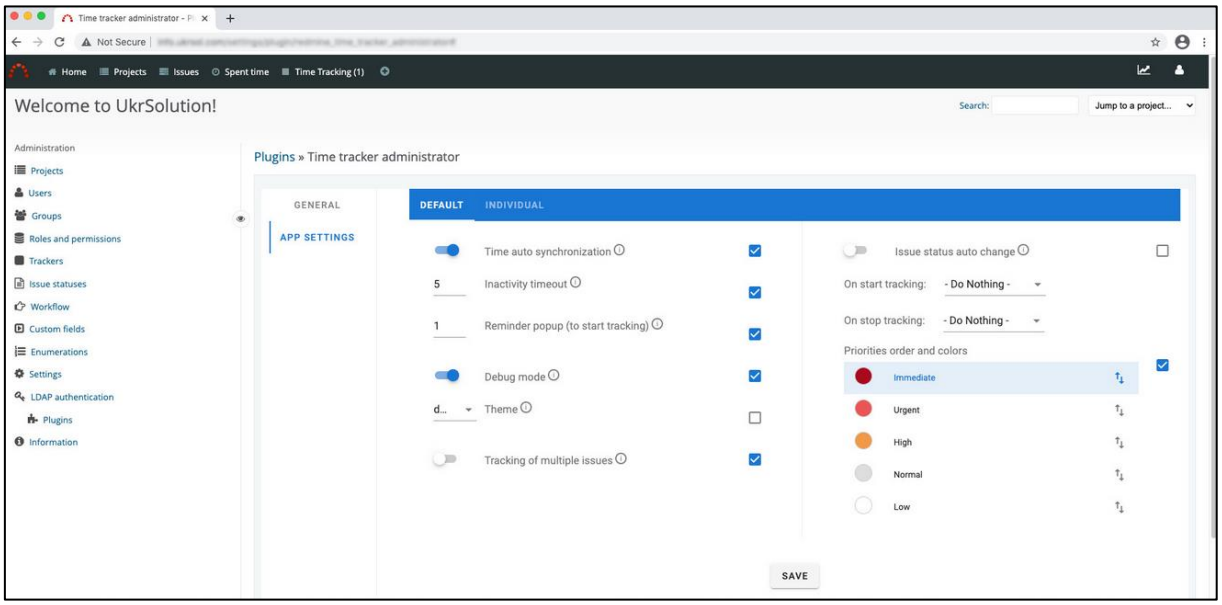
- Ierobežota faila lieluma augšupielāde līdz 10 MB vai mazāk.
- Sarežģīta integrācija un migrācija no citām projektu pārvaldības sistēmām, kas ietver manuālu katra artefakta migrēšanu uz Jira, kas ir ļoti laikietilpīgi.

Cena:

Maksas programmatūra, izmaksas sastāda 14 USD mēnesī par vienu lietotāju.

2.1.2. Redmine

Redmine ir bezmaksas atvērta pirmkoda, starpplatformu tīmekļa projektu pārvaldības un problēmu izsekošanas rīks, kas ļauj lietotājiem pārvaldīt vairākus projektus un saistītos apakšprojektus. Pēc noklusējuma ir pieejami katra projekta wiki un forumi, laika uzskaite un lomās balstīta piekļuves kontrole, kā arī Ganta diagrammu atbalsts, lai palīdzētu vizuāli attēlot projektus un to termiņus (attēls 2.1.2). Redmine spēj integrēties ar dažādām versiju kontroles sistēmām un ietver repozitorija pārlūku un izmaiņu skatītāju. Priekš Redmine ir pieejams plašs klāsts ar bezmaksas un maksas spraudņiem, kas papildina noklusējuma funkcionalitāti, piemēram, EasyRedmine.



2.1.2. attēls Redmine saskarne

Priekšrocības:

- Liels bezmaksas un maksas spraudņu klāsts, kas palīdzēs pielāgot Redmine savām vajadzībām,
- Nodrošina lietotāju ar visām svarīgākajām projekta pārvaldības instrumentiem
- Pateicoties atvērtā pirmkoda risinājumam, Redmine ir iespējams pielāgot un patstāvīgi integrēt ar citām sistēmām

Trūkumi:

- Salīdzinoši ilga uzstādīšana un konfigurācija
- Redmine saskarne neatbalsta Drag&Drop
- Sarežģīta mērogojamība un darbība lielu slodžu gadījumā

Cena:

Brīvi pieejams atvērtā pirmkoda risinājums, tomēr jāņem vērā, ka, lai sakonfigurētu Redmine atbilstoši savām vajadzībām nāksies patērēt daudz laika, vai arī pāriet uz kādu no maksas Redmine spraudņiem.

2.2. Izstrādes posma rīki

Izstrāddarbināšanas kontekstā izstrādes posmā svarīgākie rīki ir versiju kontroles rīki, jo tieši balstoties uz tur izvietoto kodu, tiks izpildīti nepārtrauktās integrēšanas un testēšanas procesi. Līdz ar to izvēloties starp versiju kontroles rīkiem ir ļoti svarīgi, lai tie būtu saderīgi ar maksimāli lielu skaitu izstrāddarbināšanas rīku.

2.2.1. GitHub

GitHub ir tīmeklī bāzēts Git versiju kontroles rīks. Papildus Git versiju kontroles un pirmkodu pārvaldības funkcionalitātei, GitHub piedāvā arī sava funkcijas. Tas nodrošina piekļuves kontroli, sadarbības funkcijas, piemēram, kļūdu izsekošanu, uzdevumu pārvaldību, nepārtrauktu integrāciju un wiki katram projektam. Projektiem vietnē GitHub.com var piekļūt un tos pārvaldīt, izmantojot jebkuru standarta Git komandrindas saskarni. GitHub.com arī ļauj lietotājiem pārlūkot vietnes publiskos repozitorijus. Priekš GitHub ir pieejami vairāki Git spraudņi, kā arī darbvirsmas klienti.

Priekšrocības:

- Integrāciju ar GitHub atbalsta visi nepārtrauktās integrācijas rīki.
- Kods tiek uzglabāts mākonī.
- Gists funkcija, kas ļauj atsevišķas datnes pārkonvertēt funkcionālā Git repozitorijā, kas var noderēt, daloties ar konfigurācijas datnēm.

Trūkumi:

- Bezmaksas risinājums neder lielākiem uzņēmumiem un projektiem.

Cena:

- Bezmaksas, ja kods tiek glabāts kā atvērts pirmkods, vai arī ja netiek pārsniegti izmantošanas ierobežojumi (2000 automatizācijas minūtes mēnesī).
- Team maksas plāns: 4 USD par lietotāju mēnesī.

2.2.2. Bitbucket

Bitbucket ir versiju kontroles repozitoriju mitināšanas pakalpojums, kas tika izveidots 2008. gadā un pieder uzņēmumam Atlassian. Šis Git krātuves pārvaldības risinājums ir rakstīts izmantojot Python un ir pārsvarā mērķēts uz uzņēmumiem, kas nevēlas publicēt savu kodu ārpus uzņēmuma, kā arī vēlas papildus atvieglotu integrāciju ar atsevišķiem rīkiem, piemēram, Jira.

Priekšrocības:

- Vienkārša integrācija ar tādiem rīkiem kā Jira, Jenkins un Bamboo.
- Iespēja importēt repozitorijus no Git, GoogleCode un SVN.
- Autentifikācijas atbalsts priekš GitHub, Google, Facebook un Twitter.
- Ir pieejama Android lietotne – Bitbeaker.
- Kods netiek uzglabāts publiski.
- Kods tiek uzglabāts mākonī.

Trūkumi:

- Maksas pakalpojums.

Cena:

- Standarta plāns 3 USD par lietotāju mēnesī
- Premium plāns: 6 USD par lietotāju mēnesī

2.3. Būvēšanas un piegādes posma rīki

Programmatūras būvēšanas posma rīkos var izdalīt divas galvenās kategorijas: nepārtrauktas integrēšanas rīki (Drone, Jenkins, CircleCi), kas veic programmatūras projektu būvējumus, un konteinerizācijas rīki (Docker, Podman), kas paredzēti, lai būvējumu iepakotu konteinerī, kas satur visu nepieciešamo infrastruktūru koda izpildei. Tomēr šajā darbā apskatītie nepārtrauktas integrēšanas rīki ir spējīgi veiksmīgi izpildīt arī nepārtrauktas piegādes funkcijas, kas dod tiem priekšroku salīdzinājumā ar rīkiem, kas spēj realizēt tikai vienu no funkcijām

2.3.1. Drone

Drone ir nepārtrauktas integrācijas un piegādes rīks, kas lietotājiem dod iespēju automatizēt būvējumu izveidi, testēšanu un piegādi. Drone darbība balstās Docker konteinerizācijas tehnoloģijas izmantošanā, izpildot katru soli atsevišķā konteinerī. Attiecīgi, lai definētu un izpildītu darba plūsmu elementus, pietiek ar vienkāršu YAML konfigurācijas failu. Tomēr neskatoties uz to, ka Drone ir vieglāk integrēt ar Docker un DockerSwarm risinājumiem, ir iespējams to izmantot arī priekš Kubernetes.

Priekšrocības:

- Integrēts ar GitHub, BitBucket un Google Code.
- Viegla uzstādīšana, vajadzīgs tikai Docker.
- Docker integrācija vienkāršo piegādes konfigurēšanu.

Trūkumi:

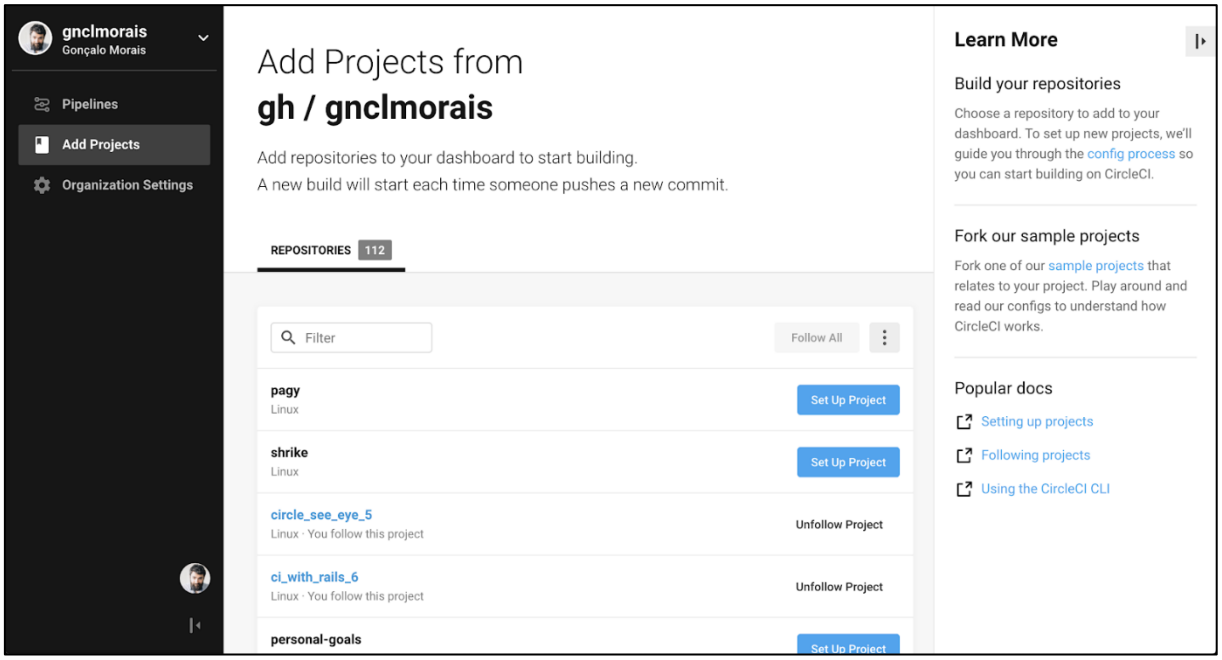
- Neļauj konfigurēt vairākus projektus, kas izmanto vienu un to pašu GitHub repozitoriju.
- Salīdzinoši neliels spraudņu piedāvājums salīdzinājumā ar citiem rīkiem, piemēram Jenkins.

Cena:

Bezmaksas atvērta pirmkoda risinājums.

2.3.2. CircleCI

CircleCI tika izveidots ar mērķi panākt ātrumu un mērogojamību. Rīkā līdzīgi kā Drone būvējumu konfigurācijas tiek saglabātas programmatūras repozitorijā kā YAML konfigurācijas datne, tādējādi atvieglojot izmaiņu izsekošanu (attēls 2.3.1). Orb sistēma ļauj ar minimālu piepūli pagarināt darbplūsmas, izmantojot citu rīku funkcionalitāti. Šo funkciju esamība ļauj viegli integrēt CircleCI gandrīz jebkurā projektā.



2.3.1. attēls CircleCI grafiskā saskarne [8]

Priekšrocības:

- Vienkārša integrācija, izmantojot YAML konfigurāciju
- Ērta sasaiste ar Git repozitorijiem
- Viegli pievienot jaunus projektus
- Ērta grafiskā saskarne

Trūkumi:

- Rīks dažreiz mēdz būt salīdzinoši lēns
- Vāja dokumentācija, saskaroties ar problēmām
- Pārāk dārgs rīks, ņemot vērā konkurentu piedāvājumu

Cena:

Sākot no 30 USD mēnesī, turpmāks cenas pieaugums atkarīgs no izmantošanas

2.3.3. Jenkins

Jenkins pirmo reizi tika izlaists 2011. gadā un ir vecākais no apskatītajiem rīkiem. Sākotnēji Jenkins tika izstrādāts kā nepārtraukta integrēšanas risinājums, taču tagad tas spēj tikt gala arī ar nepārtrauktas piegādes uzdevumiem. Ja Jenkins uzstāda bez nekādiem spraudņiem, tad tā funkcionalitāte ir diezgan ierobežota. Tieši lielais spraudņu klāsts, kas ļauj pielāgot Jenkins gandrīz jebkuram uzdevumam ir šī rīka stiprākā puse. Diemžēl tā vienlaikus ir arī Jenkins vājā puse, jo rīka pareiza konfigurācija un uzturēšana ir darbietilpīgs process.

Priekšrocības:

- Pateicoties skriptiem, integrācijām un spraudņiem var paveikt gandrīz jebkādu uzdevumu, kas saistīts ar nepārtraukto integrēšanu un piegādi.
- Tiek aktīvi uzturēts
- Liela zināšanu bāze, kas nozīmē, ka, ja sanāk saskarties ar problēmu, kāds cits arī jau ir saskaries ar kaut ko līdzīgu
- Ir iespējama integrācija ar visiem nozīmīgākajiem izstrāddarbināšanas rīkiem

Trūkumi:

- Sarežģīta un laikietilpīga konfigurācija.
- Daudziem spraudņiem ir atkarības no citiem esošiem spraudņiem apgrūtinot uzturēšanu, jo ne visi spraudņi tiek regulāri atjaunoti.

Cena:

Bezmaksas, tomēr izvēloties Jenkins, jāņem vērā izmaksas, kas radīsies dēļ patērētā laika, lai rīku sagatavotu un uzturētu atbilstoši vajadzībām

2.3.4. Docker

Docker ir konteinerizācijas rīks, kas paredzēts, lai atvieglotu lietojumprogrammu izveidi, izvietošanu un palaišanu, izmantojot konteinerus. Konteineri ļauj izstrādātājam iesaiņot lietojumprogrammu ar visām nepieciešamajām daļām, piemēram, bibliotēkām un citām atkarībām, un izvietot to kā vienu pakotni. Pateicoties konteineram, izstrādātājs var būt drošs, ka viņa izstrādātā programma darbosies jebkurā citā Linux datorā neatkarīgi no visiem ierīces pielāgotajiem iestatījumiem, kas varētu atšķirties no koda rakstīšanai un testēšanai izmantotās ierīces. [9]

Priekšrocības:

- Izveidotais konteineris darbosies vienlīdz labi neatkarīgi no ierīces, kur tas tiks palaists
- Katram konteinerim ir savs resurss, un tas ir izolēts no citiem konteineriem
- Konteineru izmantošana atvieglo un pāatrina koda piegādi
- Samazinātas resursu prasības, salīdzinot ar pieejām, kas neizmanto konteinerus

Trūkumi:

- Pastāvīga datu uzglabāšana ir sarežģīta. Konteineru arhitektūra paredz, ka visi konteinera iekšienē esošie dati pazūd uz visiem laikiem, kad konteiners tiek izslēgts, ja vien jūs tie vispirms netiek saglabāti citur. Viens no veidiem kā tomēr var pastāvīgi saglabāt datus Docker ir, piemēram, Docker Data Volumes,

Cena:

Bezmaksas, atvērta pirmkoda risinājums.

2.3.5. Podman

Podman ir atvērtā pirmkoda bezdēmonu konteinerizācijas rīks, lai izveidotu, pārvaldītu un palaistu atvērto konteineru iniciatīvas (*angl. Open Container Initiative jeb OCI*) standarta konteinerus un konteineru attēlus.

Podman nodrošina ar Docker saderīgu komandrindas priekšgalu, kas var vienkārši aizstāt Docker cli. Podman piedāvā arī ligzdā aktivizētu REST API pakalpojumu, kas ļauj attālinātām lietojumprogrammām palaist konteinerus pēc pieprasījuma. Šis REST API atbalsta arī Docker API, ļaujot Docker-py un Docker-Compose lietotājiem mijiedarboties ar Podman kā ar pakalpojumu.

Podman darbojas tikai uz Linux platformām, tomēr Podman attālais REST API klients pastāv Mac un Windows platformās un var sazināties ar Podman pakalpojumu, kas darbojas Linux mašīnā vai virtuālajā mašīnā caur SSH.[10]

Priekšrocības:

- Visas nodaļā 2.3.4 minētās Docker raksturojošās priekšrocības.
- Atšķirībā no Docker nav nepieciešams dēmons.
- Ļauj kontrolēt konteinera slāņus.
- Tas ļauj palaist konteinerus kā ne-root lietotājam, kā rezultāta nekad nav jāpiešķir lietotājam root atļauja resursdatorā, mazinot drošības riskus.

Trūkumi:

- Nedarbojas ar rīkiem, kas saziņai ar konteinerizācijas dzini izmanto Docker API.

Cena:

Bezmaksas, atvērta pirmkoda risinājums.

2.4. Testēšanas posma rīki

Šī darba ietvaros tiks apskatīti ar tīmekļa projektiem saistīti automatizētās testēšanas rīki, jo šāda tipa projekti šī brīža IT vidē līdz ar mobilajām lietotnēm ir vieni no izplatītākajiem un tāpēc, ka bakalaura darba tvērumā tiek apskatīti rīki, kas orientēti uz IT sistēmu, kuras saskarne ir tīmekļa mājaslapa.

2.4.1. Selenium

Selenium ir rīku komplekts tīmekļa pārlūkprogrammu automatizēšanai. Viena no galvenajām īpašībām ir, ka Selenium nodrošina W3C WebDriver specifikācijas infrastruktūru - platformu un valodas neitrālu kodēšanas saskarni, kas ir saderīga ar visām izplatītākajām tīmekļa pārlūkprogrammām. 2021. gadā Selenium joprojām ir vadošais rīks automatizēto testu veikšanai, kā rezultātā ir pieejama ļoti plaša informācijas bāze, kas ļauj salīdzinoši ātri apgūt šo rīku.

Priekšrocības:

- Var izmantot dažādās pārlūkprogrammās, piemēram, Chrome, IE, Mozilla un Opera.
- Var veikt testus ar jebkuru operētājsistēmu, piemēram, Android, Windows, Mac, iOS un Linux.
- Lai izveidotu pārskatus un pārvaldītu testa gadījumus, Selenium var integrēt ar citām sistēmām, piemēram, JUnit, TestNG un NUnit.

Trūkumi:

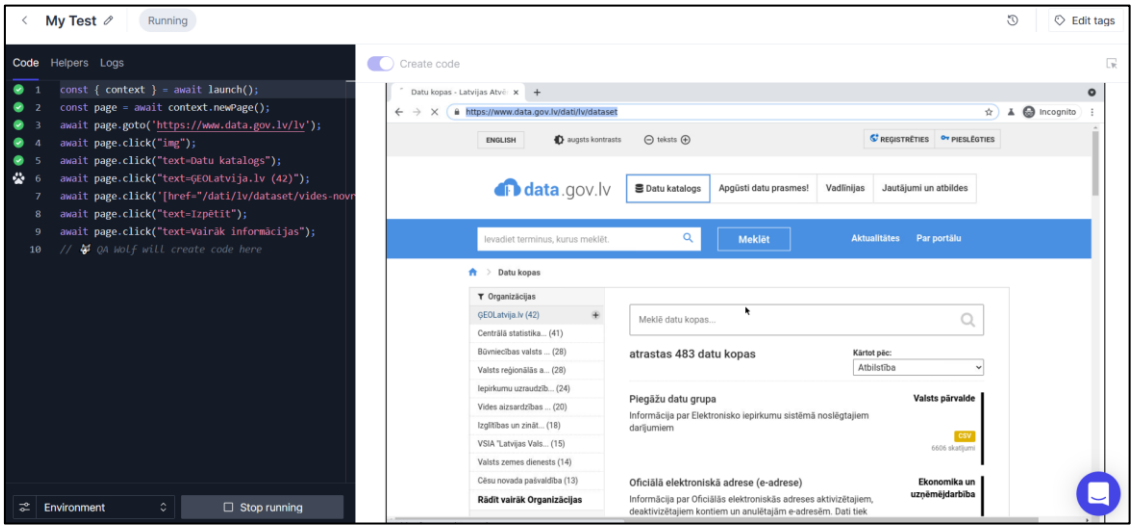
- Neviena nenodrošina regulāru tehnisko atbalstu, jo tas ir atvērtā koda rīks. Līdz ar to visa informācija un palīdzība jāmeklē patstāvīgi.
- Selenium testi mēdz būt nestabili. Ja tiek izlaistas jaunas Selenium bibliotēku versijas, tas var traucēt vai padarīt iepriekš rakstītos testus nederīgus.

Cena:

Nav. Selenium ir atvērta pirmkoda testēšanas rīks

2.4.2. QA Wolf

QA Wolf ir vienkārša un ātra alternatīva Selenium un ir optimizēta, lai testus varētu paralēli veikt vairāki lietotāji. Šis rīks ir jaunums automatizētās testēšanas pasaulē, un kopš tā publiskās palaišanas 2021. gada janvārī ir guvis ļoti pozitīvas atsauksmes. Tā galvenā priekšrocība pār citiem testēšanas rīkiem ir tas, cik viegli un ātri ir iespējams izveidot testus, kas pārbauda tīmekļa mājaslapas darbību. Kā papildus bonuss tas ir atvērta pirmkoda rīks.



2.4.1. attēls QA Wolf automatizētā testa piemērs

Priekšrocības:

- Ļoti viegli un ātri veidojami testi, pateicoties tam, ka testa kods tiek automātiski ģenerēts, balstoties uz lietotāja darbībām testējamajā mājaslapā (attēls 2.4.1).
- Testus var veidot un laist QA Wolf mājaslapā, nav nepieciešama uzstāde un konfigurācija
- Ātrs testu izpildes laiks

Trūkumi:

- Automātiski ģenerētais kods der tikai gadījumos, kad vajag pārbaudīt vai veicamās darbības ir iespējams, ja ir nepieciešams arī pārbaudīt rezultātu, tad kods ir patstāvīgi jāpapildina.

Cena:

QA Wolf makoņpakalpojums izmaksā 119 USD mēnesī par 1000 testu izpildi.

2.5. Darbināšanas posma rīki

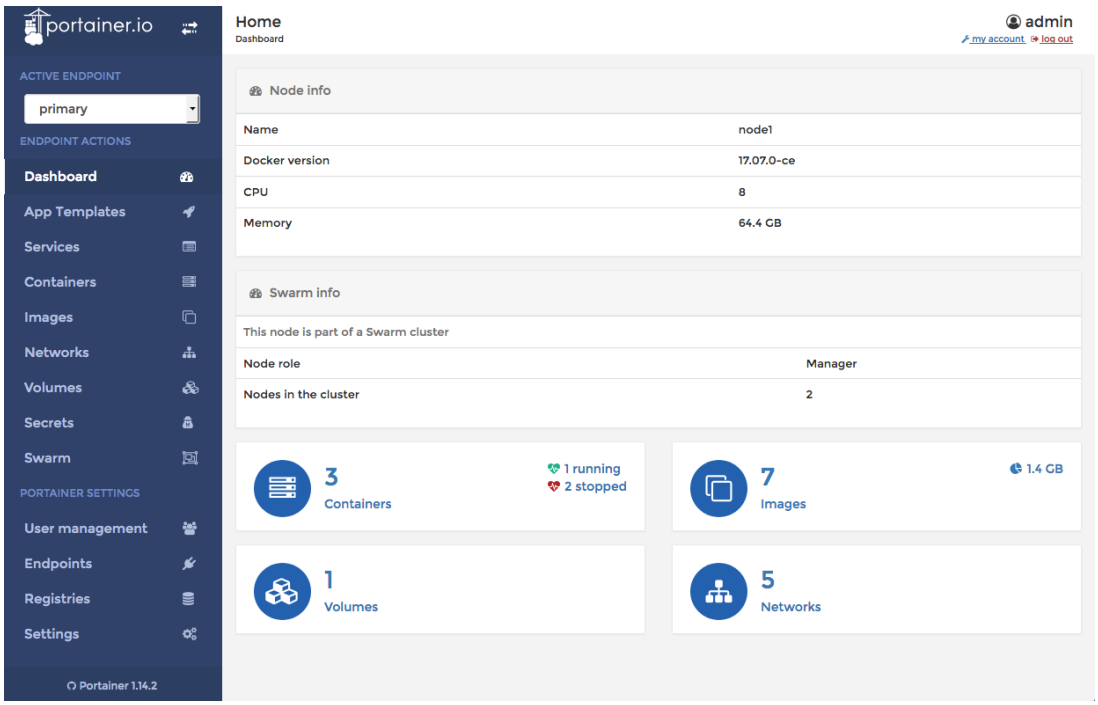
Lielākā daļa mūsdienu IT risinājumu tiek darbināti izmantojot konteinerus, lai tos pārvaldītu, organizētu un darbinātu eksistē divi galvenie rīki: Docker Swarm un Kubernetes, kas arī tiks apskatīti sīkāk.

2.5.1. Docker Swarm

Docker Swarm ir paša Docker konteineru pārvaldības rīks. Tas izmanto standarta Docker API un tīklošanu, padarot to viegli integrējamu vidē, kurā jau tiek strādāts ar Docker konteineriem. Docker Swarm ir izstrādāts, lai realizētu četrus galvenos principus:

- Sasniegt "vienkārši darbojas" lietotāja pieredzi.
- Izturīga arhitektūra, kas neapstāsies, ja kāda tās daļa pārstās strādāt kā plānots.
- Panākt augstu drošības līmeni automātiski ģenerējot sertifikātus.
- Atgriezeniska savietojamība ar esošajām komponentēm.

Diemžēl pēc noklusējuma Docker Swarm nav grafiskās saskarnes un viss ir jādara ar komandrindas palīdzību, bet šo mīnusu iespējams apiet izmantojot atvērta pirmkoda risinājumu Portainer, kas nodrošina Docker Swarm ar viegli pārvaldāmu grafisko saskarni(attēls 2.5.1).



2.5.1. attēls Portainer grafiskā saskarne

Priekšrocības:

- Jau sākotnēji veidots, lai izmantotu Docker dzini
- Ir savs Swarm API
- Viegla integrācija ar citiem Docker rīkiem kā Docker Compose un Docker CLI
- Rīki, pakalpojumi un programmatūra, kas darbojas ar Docker konteineriem, labi darbosies arī ar Swarm
- To ir viegli uzstādīt un sagatavot priekš Docker vidēm[11]

Trūkumi:

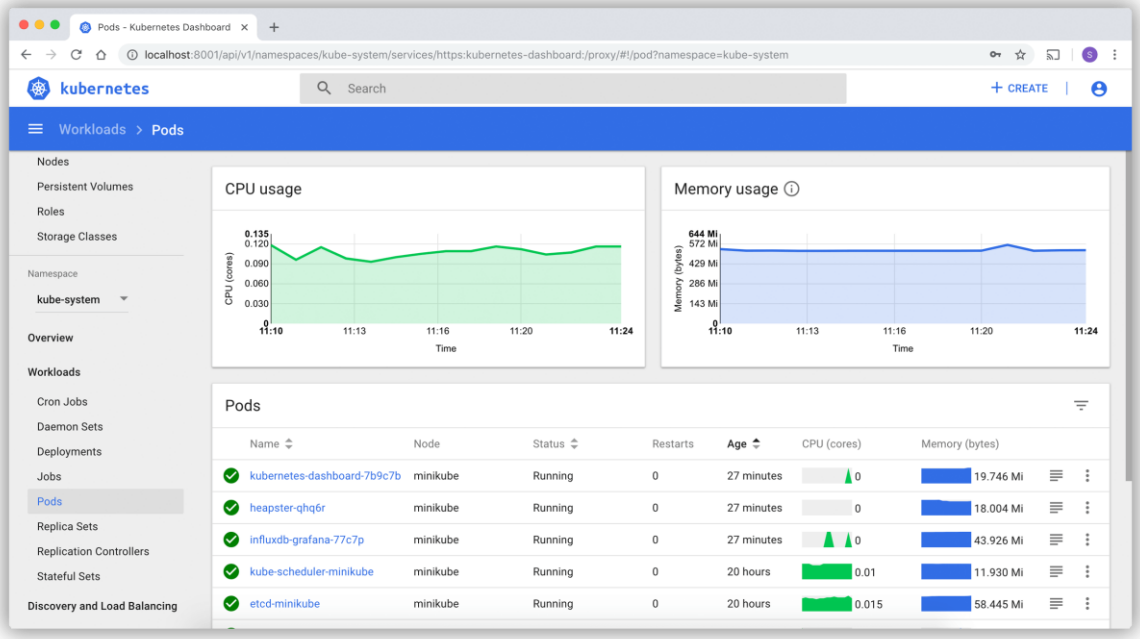
- Ierobežoti pielāgojumi un paplašinājumi
- Mazāk funkcionalitātes nekā Kubernetes

Cena:

- Bezmaksas atvērta pirmkoda risinājums.

2.5.2. Kubernetes

Kubernetes ir kļuvusi par standarta rīku konteineru pārvaldībai. To atbalsta visi galvenie mākoņpakalpojumu sniedzēji, padarot to par loģisku izvēli organizācijām, kuras vēlas vairāk lietojumprogrammu pārvietot uz mākoni. Kubernetes nodrošina kopēju satvaru dalīto sistēmu darbināšanai, lai izstrādes komandām ir pastāvīga, nemainīga infrastruktūra sākot ar katra projekta izstrādi līdz piegādei produkcijā. Kubernetes var pārvaldīt mērogojamību, pieejamību, izvietošanas modeļus un daudz ko citu. Pēc noklusējuma Kubernetes arī nav grafiskās saskarnes taču to ir viegli uzstādīt ar tikai vienu komandrindas komandu (attēls 2.5.1).



2.5.1. attēls Kubernetes grafiskā saskarne Dashboard UI

Priekšrocības:

- Ļoti aktīva atvērtā koda kopiena, kas izstrādā koda bāzi
- Visvairāk izmantotais konteineru pārvaldības risinājums, ko izmanto arī lielākie IT uzņēmumi, piemēram, Google un IBM
- Darbojas lielākajā daļā operētājsistēmu
- Pieejams publiskajā mākonī vai lokāli - visu lielo mākoņpakalpojumu sniedzēju pārvaldīti vai nepārvaldīti piedāvājumi (IBM Cloud, AWS, Microsoft Azure, Google Cloud Platform utt.)

- Plašs Kubernetes atbalsts no mākoņa rīku pārdevēju ekosistēmas, piemēram, Portworx Sysdig un LogDNA.

Trūkumi:

- Atvērtā koda kopienas atjauninājumi tiek veikti bieži, un ir nepieciešama rūpīga to uzstādīšana, lai izvairītos no darba traucējumiem
- Nav īsti piemērots priekš individuāliem izstrādātājiem, kas izstrādā vienkāršas lietotnes, kuras reti tiek piegādātas.
- Bieži nepieciešami papildu rīki (piemēram, kubectl CLI), pakalpojumi, nepārtrauktas integrācijas / nepārtrauktas izvietošanas darbplūsmas un citas izstrāddarbināšanas prakses, lai pilnībā pārvaldītu piekļuvi, identitāti, pārvaldību un drošību. [11]

Cena:

- Bezmaksas atvērta pirmkoda risinājums.

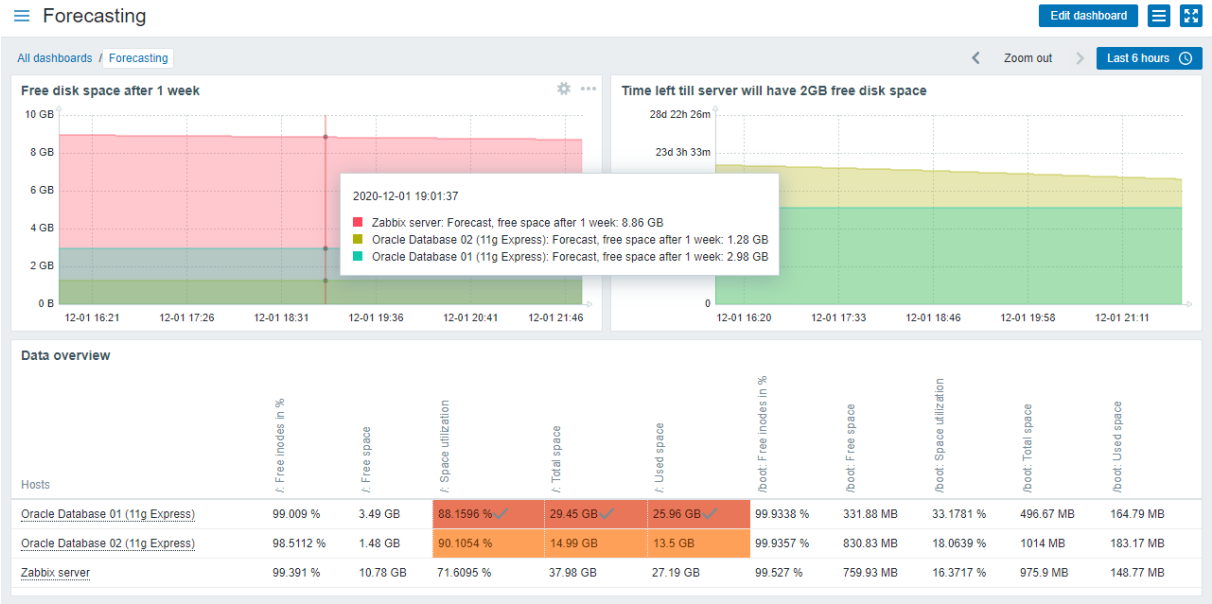
2.6. Pārraudzīšanas posma rīki

Pēdējais izstrāddarbināšanas darba plūsmas posms ir pārraudzīšana. Laicīga kļūdu atklāšana ļauj būt drošam, ka, ja ar uzstādīto IT sistēmu kaut kas notiek, tas tiks maksimāli ātri novērsts. Šajā nodaļā sīkāk tiks apskatīti divi no izplatītākajiem pārraudzības rīkiem Zabbix un Nagios XI.

2.6.1. Zabbix

Zabbix ir atvērtā koda uzraudzības programmatūras rīks dažādām IT komponentēm, tostarp tīkliem, serveriem, virtuālajām mašīnām un mākoņpakalpojumiem. Zabbix nodrošina uzraudzības metriku, kas ļauj identificēt kļūdas programmatūras darbībā, kā arī noteikt tīkla izmantošanu, procesora slodzi un diska vietas patēriņu (attēls 2.6.1).un veikt atbilstošas prognozes, brīdinot sistēmas pārvaldnieku (attēls 2.6.2). Rīkā ir iespējams iestatīt funkcionalitāti, ka kļūdu vai brīdinājumu gadījumā atbildīgais darbinieks saņem e-pastu.

2.6.1. attēls Servera statusu pārskats Zabbix rīkā[12]



2.6.2. attēls Prognožu pārskats Zabbix rīkā[12]

Priekšrocības:

- Liels skaits ar pārraudzības veidnēm priekš serveriem, kas ir uzreiz gatavas lietošanai.
- Prognozēšanas funkcionalitāte.
- Plašas pielāgošanas iespējas.

Trūkumi:

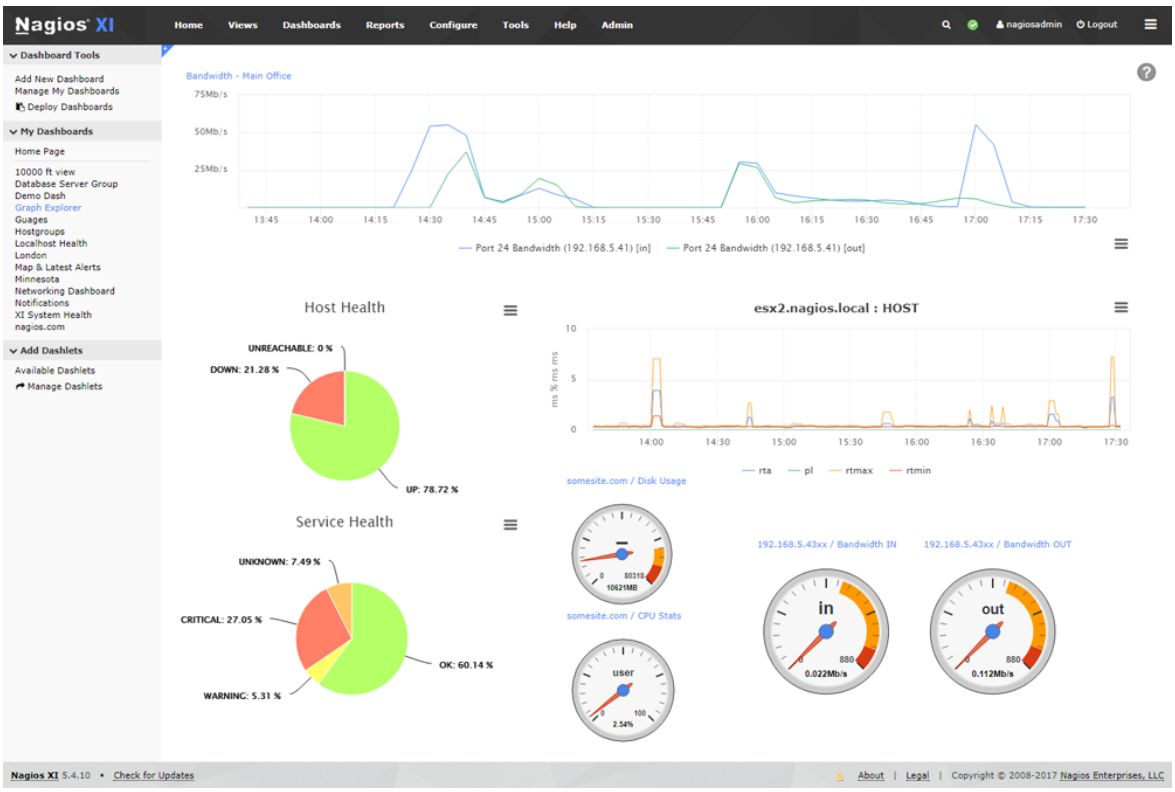
- Salīdzinoši sarežģīta pirmreizējā uzstādīšana un konfigurēšana.
- Sākotnēji grūti orientēties grafiskajā saskarnē.

Cena:

- Bezmaksas. Atvērta pirmkoda risinājums.

2.6.2. Nagios XI

Nagios XI ir serveru un tīkla uzraudzības programmatūra (2.6.3. attēls), kas palīdz uzņēmumam uzraudzīt lietojumprogrammas, pakalpojumus un tīklu vienā centrālā risinājumā. Pateicoties simtiem trešo pušu papildinājumu Nagios XI spēj nodrošināt praktiski visu iekšējo un ārējo lietojumprogrammu, pakalpojumu un sistēmu uzraudzību. Līdzīgi kā Zabbix arī Nagios XI ir pieejama progozēšanas funkcionalitāte.



2.6.3. attēls Sistēmas pārskats Nagios XI rīkā

Priekšrocības:

- Viegli izmantojams, kā arī ir pieejams klientu atbalsts.
- Ļoti plaša funkcionalitāte, padarot rīku izmantojamu jebkurā sistēmā.
- Plašas pielāgošanas iespējas.
- Iespēja ģenerēt regulāras atskaites
- Auditēšanas funkcionalitāte (pieejama tikai ar Enterprise licenci)

Trūkumi:

- Dārgs rīks, ja vēlas izmantot tā pilno funkcionalitāti.

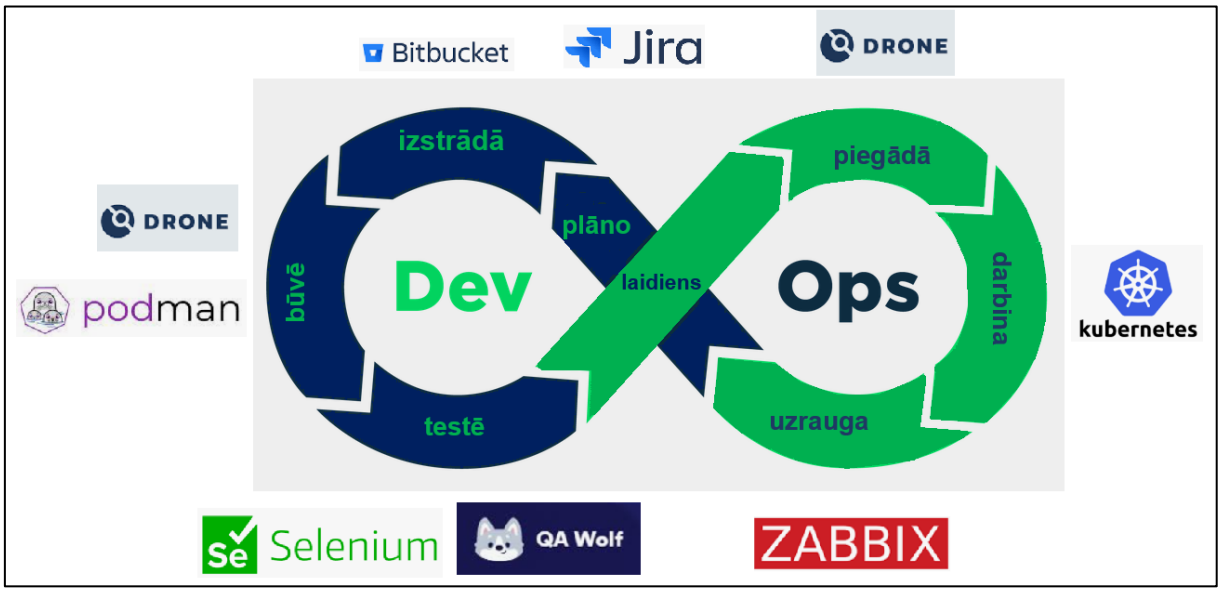
Cena:

- Standarta licences cena 1995 USD vienam lietotājam.
- Enterprise licences cena 3495 USD vienam lietotājam.

3. Rezultāti

Darba ievadā tika noteikts mērķis sastādīt pēc iespējas labāko izstrāddarbināšanas rīku kopu (3. attēls), lai realizētu tīmeklī bāzētu IT sistēmu. Pēc darbā veiktās rīku analīzes tika secināts, ka plānošanas posmā vislabāk būtu izmantot Jira rīku, jo tas sniedz ļoti plašas integrācijas iespējas bez nepieciešamības veltīt laiku ilgstošai spraudņu meklēšanai un konfigurācijai kā tas varētu būt Redmine gadījumā.

Par versijas kontroles rīku izvēlējos Bitbucket dēļ atvieglotās integrācijas ar Jira, kā arī tādēļ, ka par to pašu naudu no Bitbucket iespējams iegūt vairāk vērtīgas funkcionalitātes nekā no GitHub. Par nepārtrauktas integrācijas un piegādes rīku tika izvēlēts Drone tā vieglās lietojamības, konfigurācijas un integrācijas ar BitBucket dēļ. Savukārt par konteinerizācijas rīku un konteineru pārvaldības rīku tika izvēlēts Podman un Kubernetes kombinācija, kas pēdējā laikā ir kļuvusi par izplatītāko industrijas standartu. Noslēgumā rūpējoties par kvalitāti testēšanai tiktu izmantots gan Selenium, gan QA Wolf rīks. Ņemot vērā, ka Selenium testu rakstīšana un uzturēšana aizņem daudz laika, QA Wolf rīks spētu nodrošināt projektu ar ātri sastādāmiem un izpildāmiem testiem, lai varētu izvairīties no situācijas, ka tiek piegādāts netestēts kods. Un noslēgumā par pārraudzības rīku izraudzīts Zabbix, pateicoties ļoti plašajam funkciju klāstam, kas ir pieejama par brīvu, pat neskatoties uz to, ka Zabbix piemīt daži trūkumi.



3. attēls Izvēlētā izstrāddarbināšanas rīku kopa

Secinājumi

Bakalaura darba noslēgumā var secināt, ka izstrāddarbināšanas jeb DevOps pieeju ievērošana, veicot izstrādi, testēšanu, kā arī IT operācijas, var būtiski uzlabot gala rezultātā izveidotās IT sistēmas kvalitāti, kā arī nodrošināt veiksmīgu izveidotā risinājuma uzturēšanu.

Katrs no izstrāddarbināšanas posmiem dod savu pienesumu, gala rezultāta kvalitātei. Plānošanas rīki palīdz iesaistītajiem darbiniekiem sadalīt uzdevumus, kā arī nodrošina projektu vadītājiem iespēju kontrolēt projekta norisi. Pirmkoda uzglabāšanas un versionēšanas rīki palīdz sekot līdzi koda izmaiņām, kā arī nodrošina integrāciju ar integrācijas un testēšanas rīkiem, kas nodrošina to, ka līdz piegādes fāzei nonāk tikai pārbaudīts kods. Sistēmas darbināšanas un pārraudzības posmi savukārt nodrošina to, ka arī pēc koda piegādes, izstrādātais risinājums turpina darboties ar minimāli iespējamiem traucējumiem.

Analizējot katram posmam specifiskos rīkus, nācās saskarties ar ļoti plašu šo rīku piedāvājumu. Ikkatrā no posmiem nācās izvēlēties, kurus rīkus bakalaura darba ietvarā ir vērts apskatīt un kurus nē, jo neskatoties uz plašo piedāvājumu, daudzi rīki tā arī nekad neiegūs ievērojamu tirgus daļu un paliks par nišas risinājumiem, kas vislabāk derēs tikai kādā konkrētā scenārijā. Tomēr bija arī patīkami pārsteigumi, no apskatītajiem rīkiem ļoti pozitīvu iespaidu atstāja tikai šogad iznākušais automatizētās testēšanas rīks QA Wolf, kas ideāli noderēs komandām, kas varbūt ne vienmēr pagūst uzrakstīt testus, izmantojot Selenium vai citu rīku.

Kopumā bakalaura darba rakstīšanas rezultātā iegūtās zināšanas ievērojami palīdzēja labāk izprast izstrāddarbināšanas pieeju, kā arī veiksmīgi izpildīt darba sākumā izvirzīto mērķi patstāvīgi sastādīt izstrāddarbināšanas rīku ķēdi, ko varētu izmantot, lai izstrādātu tīmeklī bāzētu IT sistēmu.

Izmantotā literatūra

[1] Latvijas Nacionālais terminoloģijas portāls [tiešsaiste]. – [atsauce 20.05.2021.] Pieejams: https://termini.gov.lv/

[2] Atlassian raksts "History of DevOps" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://www.atlassian.com/devops/what-is-devops/history-of-devops

[3] Lucidchart raksts "Understanding the DevOps process flow" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://www.lucidchart.com/blog/devops-process-flow

[4] Devopedia raksts "Devops" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://devopedia.org/devops

[5] Netapp raksts "What is DevOps" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://www.netapp.com/devops-solutions/what-is-devops

[6] Atlassian pamācība "What is Jira used for" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for

[7] Projektu pārvaldības rīku tirgus daļas salīdzinājums[tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://www.datanyze.com/market-share/project-management--217/jira-market-share

[8] CircleCI bloga raksts "Capture screenshots of flaky end to end tests" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://circleci.com/blog/capture-screenshots-of-flaky-end-to-end-tests

[9] Opensource raksts "What is Docker?" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://opensource.com/resources/what-docker

[10] Podman raksts "What is Podman? Simply put: alias docker=podman" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://podman.io/whatis.html

[11] IBM bloga raksts "Docker Swarm vs. Kubernetes: A comparison" [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://www.ibm.com/cloud/blog/docker-swarm-vs-kubernetes-a-comparison

[12] Zabbix saskarnes ekrānattēli [tiešsaiste]. – [atsauce 28.05.2021.] Pieejams: https://www.zabbix.com/screenshots

Bakalaura darbs „DevOps metodoloģijas un rīku izmantošana IT sistēmu izstrādē” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti.

Autors: Ilmārs Vindačs 31.05.2021.

Rekomendēju darbu aizstāvēšanai
Vadītājs: M.dat. Jānis Vempers 31.05.2021.

Recenzents: docents Dr.sc.comp. Leo Trukšāns

Darbs iesniegts Datorikas fakultātē 31.05.2021.
Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē
__.06.2021. prot. Nr. ___.
Komisijas sekretārs: asociētais profesors Dr.sc.comp. Sergejs Kozlovičs