A blue and white logo

Description automatically generated with medium confidence

“Liepājas Valsts tehnikums”

**Lietojumprogrammu veidošanas, testēšanas un izvietošanas rīku kopums.**

Kvalifikācijas eksāmena praktiskās daļas tehniskā dokumentācija

Izglītības programma **33484011 Programmēšana**

Profesionālā kvalifikācija **Programmēšanas tehniķis**

Projekta izstrādātājs Mārtiņš Bardulis

Eksāmena datums 2023. gada 21. jūnijs

Liepāja 2023

**Saturs**

[**Ievads** 4](#_Toc137448067)

[**1. Uzdevuma formulējums** 5](#_Toc137448068)

[**2. Programmatūras prasību specifikācija** 7](#_Toc137448069)

[**2.1. Produkta perspektīva** 7](#_Toc137448070)

[**2.2. Sistēmas funkcionālās prasības** 7](#_Toc137448071)

[**2.2.1. Izveides darba plūsma** 7](#_Toc137448072)

[**2.2.2. Manuālās izvietošanas darba plūsma** 7](#_Toc137448073)

[**2.2.2. Automātiskās izvietošanas darba plūsma** 8](#_Toc137448074)

[**2.3. Sistēmas nefunkcionālās prasības** 9](#_Toc137448075)

[**2.3.1. Drošība** 9](#_Toc137448076)

[**2.3.2. Uzturamība** 9](#_Toc137448077)

[**2.3.3. Lokalizācija** 9](#_Toc137448078)

[**2.3.4. Lietojamība** 9](#_Toc137448079)

[**2.4. Gala lietotāja raksturiezīmes** 9](#_Toc137448080)

[**3. Izstrādes līdzekļu, rīku apraksts un izvēles pamatojums** 10](#_Toc137448081)

[**3.1. Izvēlēto risinājuma līdzekļu un valodu apraksts** 10](#_Toc137448082)

[**3.1.1. GitHub** 10](#_Toc137448083)

[**3.1.2. Jenkins** 10](#_Toc137448084)

[**3.1.3. Visual Studio Code (VS Code)** 10](#_Toc137448085)

[**3.1.4. SonarQube** 11](#_Toc137448086)

[**3.1.5. Groovy** 11](#_Toc137448087)

[**3.1.6. AWS (Amazon Web Services)** 11](#_Toc137448088)

[**3.2. Iespējamo (alternatīvo) risinājuma līdzekļu un valodu apraksts** 12](#_Toc137448089)

[**3.2.1 GitLab** 12](#_Toc137448090)

[**3.2.2 Travis CI** 12](#_Toc137448091)

[**3.2.3 IntelliJ IDEA** 13](#_Toc137448092)

[**3.2.4 Google Cloud Platform** 13](#_Toc137448093)

[**3.2.5 ESLINT** 14](#_Toc137448094)

[**3.2.6 Python** 14](#_Toc137448095)

[**4. Sistēmas modelēšana un projektēšana** 16](#_Toc137448096)

[**4.1. Sistēmas struktūras modelis** 16](#_Toc137448097)

[**4.2. Implementācijas diagramma** 17](#_Toc137448098)

[**5. Lietotāju ceļvedis** 18](#_Toc137448099)

[**5.1. Izveides darbplūsmas ceļvedis** 18](#_Toc137448100)

[**5.2. Izvietošanas darbplūsmas ceļvedis** 19](#_Toc137448101)

[**5.3. Izvietotās programmas piekļūšanas ceļvedis** 20](#_Toc137448102)

[**6. Testēšanas dokumentācija** 21](#_Toc137448103)

[**6.1. Izvēlētās testēšanas metodes, rīku apraksts un pamatojums** 21](#_Toc137448104)

[**6.2. Testēšanas pārskats** 22](#_Toc137448105)

[**Secinājumi** 23](#_Toc137448106)

[**Literatūras un informācijas avotu saraksts** 24](#_Toc137448107)

**Ievads**

Mūsdienu ātrajā digitālajā vidē uzņēmumi tiecas nodrošināt ātru un efektīvu lietojumprogrammu piegādi. Šī rīku kolekcija automatizē lietojumprogrammu izvietošanu, testēšanu un veidošanu darbības plūsmā. Šis produkts optimizē procesus, palielina produktivitāti, nodrošina uzticamību un paātrina augstas kvalitātes lietojumprogrammu piegādi.

Manuālie lietojumprogrammu ievietošanas procesi var būt laikietilpīgi, pakļauti kļūdām un kavē produktivitāti. Šie rīki izslēdz manuālo iejaukšanos, automatizējot svarīgās darbības plūsmas posmus. Vai nu ievietojat jaunāko lietojumprogrammas versiju, veicot visaptverošus testus vai izveidojot optimizētas pakotnes,  produkts ar to visu tiks galā bez problēmām.

Darbības plūsmas automatizācija nodrošina vairākas priekšrocības. Tā samazina ievietošanas laiku, ļaujot ātri reaģēt uz tirgus pieprasījumu un iegūt konkurences priekšrocības. Izslēdzot cilvēku kļūdas, tiek nodrošināta vienmērīga un uzticama lietojumprogrammu ievietošana. Šie daudzpusīgie rīki integrējas ar dažādām izstrādes vides, padarot tos piemērotus gan maziem, gan lieliem komandu darbiem.

Secinājumā šī rīku kolekcija revolucionē lietojumprogrammu ievietošanas procesu, automatizējot svarīgos darbības plūsmas posmus. Palielinot produktivitāti, uzticamību un ātrumu, šis produkts ļauj uzņēmumiem veiksmīgi darboties digitālajā vidē, koncentrējoties uz inovāciju un izaugsmi.

**1. Uzdevuma formulējums**

**Programmatūras produkta nepieciešamība:**

Šis produkts aizpilda nozīmīgu vajadzību nozarē, automatizējot lietojumprogrammu izvietošanas procesu. Izstrādāts manā DevOps praksē, šis rīku kopums risina manuālās izvietošanas laikietilpīgumu un kļūdainības problēmas. Šis risinājums samazina laiku, kas nepieciešams, lai lietojumprogramma nonāktu tirgū, izslēdz kļūdas un palielina produktivitāti. Tas bez problēmām integrējas ar dažādām izstrādes vidēm, piemērots gan mazu, gan lielu komandu darbiem.

**Sasniedzamais rezultāts:**

Šī produkta galīgais mērķis ir optimizēt lietojumprogrammu izvietošanu, nodrošinot efektīvu un uzticamu augstas kvalitātes lietojumprogrammu piegādi. Automatizējot būtiskos uzdevumus, šis produkts palīdz paātrināt lietojumprogrammu izvietošanu, izslēgt kļūdas, palielināt produktivitāti un uzlabot lietojumprogrammu kvalitāti.

**Kā tiks sasniegts mērķis:**

Galīgais mērķis - optimizēt lietojumprogrammu ievietošanu un sasniegt efektīvu un uzticamu augstas kvalitātes lietojumprogrammu piegādi - tiks sasniegts, veicot šādus soļus:

1. Automatizācija: Automatizējot būtiskos uzdevumus darbības plūsmā, piemēram, izvietošanu, testēšanu un izveidi, tiek izslēgta manuāla iejaukšanās, samazinot kļūdu risku un nodrošinot vienmērīgu un uzticamu lietojumprogrammu ievietošanu..
2. Optimizācija: Produkts koncentrējas uz lietojumprogrammu izstrādes darba plūsmas optimizēšanu, automatizējot laikietilpīgus uzdevumus, palielinot komandas produktivitāti un taupot vērtīgo laiku un resursus.
3. Testēšana un kvalitātes nodrošināšana: Ieviesti visaptveroši testēšanas procesi, kas ļauj identificēt un novērst problēmas, nodrošinot augstas kvalitātes lietojumprogrammu piegādi, kas atbilst noteiktajiem standartiem un lietotāju ekspektācijām.
4. Elastība un pielāgojamība: Produkts ir izstrādāts, lai atbalstītu dažādas izstrādes vides un pielāgotos dažādu lielumu komandām, nodrošinot elastību un pielāgojamību.

**Kā tiks konstatēts, ka mērķis sasniegts:**

Mērķis tiks sasniegts, kad tiks novēroti šādi radītāji:

1. Optimizēta izvietošanas procesa organizācija: Lietojumprogrammu izvietošanas process kļūst ievērojami ātrāks un efektīvāks. Manuāla iejaukšanās ir izslēgta, un automatizētās darba plūsmas nodrošina vienmērīgu un uzticamu izvietošanu.
2. Samazināts laiks, lai lietojumprogramma nonāktu pie lietotājiem: Nepieciešamais laiks, lai piegādātu lietojumprogrammas tirgū, ir ievērojami samazināts. Lietojumprogrammas var tikt ātri izvietotas, ļaujot ātri reaģēt uz lietotāju prasībām.
3. Kļūdu samazināšana: Cilvēku radītas kļūdas izvietošanas procesā tiek samazinātas vai izslēgtas. Automatizēti testēšanas un kvalitātes nodrošināšanas procesi nodrošina lietojumprogrammu rūpīgu pārbaudi, rezultātā iegūstot augstas kvalitātes un uzticamu gala produktu.

Novērojot šos rādītājus var novērtēt, vai mērķis ir sasniegts, un veikt nepieciešamos uzlabojumus, lai sasniegtu izvirzīto mērķi.

**2. Programmatūras prasību specifikācija**

Šajā dokumenta sadaļā tiek aprakstīta produkta perspektīva, funkcionālās un nefunkcionālās prasības un produkta gala lietotāja raksturiezīmes.

**2.1. Produkta perspektīva**

Šis produkts automatizē aplikācijas izvietošanas, testēšanas un izveidošanas procesus ražošanas darba plūsmā. Tas optimizē darbplūsmas, integrējoties esošajā vidē. Šis risinājums ir pielāgojams un viegli mērogojams piedāvājot pielāgošanas iespējas dažādiem projektu izmēriem un prasībām.

Kopumā produkta perspektīva akcentē risinājuma vērtību izstrādes procesā, piedāvājot spēcīgu un efektīvu rīku komplektu, lai optimizētu ražošanas darba plūsmas un dotu komandām iespēju nodrošināt augstas kvalitātes programmatūru.

**2.2. Sistēmas funkcionālās prasības**

**2.2.1. Izveides darba plūsma**

**Mērķis:**

Būvēšanas darba plūsmas mērķis ir automatizēt un optimizēt programmatūras izstrādes procesu, nodrošinot konsekventu aplikāciju būvēšanu un testēšanu.

**Ievaddati:**

Aplikācijas pirmkods, Jenkinsfile, Dockerfile un pieejas marķieri.

**Apstrāde:**

Pēc izveides darba plūsmas palaišanas, tiek veikta pirmkoda analīze, testēšana, aplikācijas izveide un tiek izsūtīts paziņojums ar darba plūsmas rezultātiem.

**Izvaddati:**

1. Testēšanas rezultāti.
2. Paziņojums ar darba plūsmas rezultātiem.
3. Docker image ar aplikācijas būvējumu.

**2.2.2. Manuālās izvietošanas darba plūsma**

**Mērķis:**

Manuālā izvietošanas darba plūsma ir paredzēta aplikācijas kontrolētai un izstrādātāju pārraudzītai izvietošanai. Tā nodrošina elastību, pielāgojamību un ļauj uzaudzīt sensitīvas vai kritiski svarīgas izvietošanas situācijas, kurās nepieciešama cilvēka izvērtēšanas spējas. Manuālās izvietošanas darba plūsmas ir piemērotas ne-automatizējamu uzdevumu veikšanai un kļūdu apstrādei izvietošanas procesā.

**Ievaddati:**

Pieejas marķieri, pieejas dati, Docker image aplikācijas būvējums, Terraform main.tf fails un aplikācijas būvējuma versija.

**Apstrāde:**

Pēc izvietošanas darba plūsmas palaišanas, tiek veikta servera izveide, izvēlētās aplikācijas būvējuma versijas izvietošana uz šī servera un tiek izsūtīts paziņojums ar darba plūsmas rezultātiem

**Izvaddati:**

1. Servera vietne ar pieeju izvietotajai lietojumprogrammai.
2. Paziņojums ar darba plūsmas rezultātiem.

**2.2.2. Automātiskās izvietošanas darba plūsma**

**Mērķis:**

Automtiskās izvietošanas darba plūsmas mērķis ir automatizēt un optimizēt programmatūras lietojumprogrammu izvietošanas procesu dažādos vidēs. Tas nodrošina konsekventas un uzticamas izvietošanas procedūras, apstrādā vidēm specifiskas konfigurācijas, veic izvietošanas pārbaudes, ļauj atgriezties pie iepriekšējās versijas kļūdu gadījumā un nodrošina pārskatamību. Izvietošanas darba plūsma ietaupa laiku, samazina kļūdas un nodrošina efektīvu programmatūras izvietošanu.

**Ievaddati:**

Pieejas marķieri, pieejas dati, Docker image aplikācijas būvējums, Terraform main.tf fails.

**Apstrāde:**

Pēc izvietošanas darba plūsmas palaišanas, tiek veikta servera izveide, jaunākā aplikācijas būvējuma izvietošana uz šī servera un tiek izsūtīts paziņojums ar darba plūsmas rezultātiem

**Izvaddati:**

1. Servera vietne ar pieeju izvietotajai lietojumprogrammai.
2. Paziņojums ar darba plūsmas rezultātiem.

**2.3. Sistēmas nefunkcionālās prasības**

**2.3.1. Drošība**

Sistēmai jānodrošina paroļu un pieeju marķieru glabāšana šifrētā veidā.

**2.3.2. Uzturamība**

Sistēmai jānodrošina aplikācijas izveidi, testēšanu un izvietošanu jebkurā brīdi, lai izstrādāji varētu izstrādāt, labot vai pārbaudīt savu programmu jebkurā sev nepieciešamā brīdī.

**2.3.3. Lokalizācija**

Sistēmai ir jābūt veidotai un aprakstītai angļu valodā. Sistēmai ir jābūt viegli saprotamai.

**2.3.4. Lietojamība**

Sistēmai jābūt ar skaidru un intuitīvu lietotāja interfeisu, kas ļauj lietotājiem viegli pārvietoties starp dažādām funkcijām un ekrāniem. Izkārtojumam jābūt loģiski organizētam ar viegli izmantojamām izvēlnēm, pogām un saitēm.

**2.4. Gala lietotāja raksturiezīmes**

Produkta gala lietotājiem, kuriem galvenais mērķis ir automatizēt izvietošanas, testēšanas un būvēšanas procesus ražošanas darbplūsmā, var būt šādas raksturīgās iezīmes:

1. Tehniskā prasme: No produkta gala lietotājiem ir sagaidāms noteikts prasmes līmenis attiecībā uz DevOps procedūrām, izvietošanas darba plūsmām un iesaistītajiem rīkiem. Viņiem jābūt pazīstamiem ar programmatūras izvietošanas un automatizācijas jēdzieniem.
2. Pieredze programmatūru izstrādē un uzturēšanā: Galalietotāji var būt ar dažādām pieredzēm, ieskaitot programmatūras izstrādi, sistēmas administrēšanu vai programmatūru uzturēšanu. Viņiem jābūt izpratnei par programmatūras izstrādes dzīves cikla praksēm un infrastruktūras pārvaldību.
3. Darba plūsmas integrācija: Produkta gala lietotājiem būs jāiekļauj automatizētie izveidošanas, testēšanas un izvietošanas procesi esošajās ražošanas darba plūsmās. Viņiem jāsaprot, kā šos procesus integrēt savās darba plūsmās un pielāgot tos savām specifiskajām prasībām.

**3. Izstrādes līdzekļu, rīku apraksts un izvēles pamatojums**

Šajā nodaļā tiek aprakstīti izmantotie līdzekļi, rīki un programmēšanas valodas, kas tika izmantotas produkta izstrādē. Šie rīki tika izvēlēti, pamatojoties uz to individuālajām stiprajām pusēm, savstarpējo saderību un spēju atbalstīt konkrētās produkta prasības, veicinot izstrādes darbplūsmas efektivitāti, automatizāciju un kodu kvalitāti.

**3.1. Izvēlēto risinājuma līdzekļu un valodu apraksts**

**3.1.1. GitHub**

**Apraksts**:

GitHub ir plaši izmantota versiju kontroles sistēma, kas ļauj veidot programmatūru sadarbības veidā. Tas nodrošina platformu, kurā var glabāt un pārvaldīt kodu, veicinot efektīvu sadarbību starp izstrādātājiem. GitHub piedāvā daudzas un dažādas funkcijas, piemēram, zaru pārvaldību, "pull request" izvērtēšanu, koda pārskatu un problēmu izsekošanu, kas veicina efektīvu koda izstrādi un sadarbību.

**Izvēles pamatojums:**

GitHub tika izvēlēts pateicoties tam, ka tam ir spēcīgas versiju kontroles iespējas, lietotājam draudzīga saskarne un tika plaši izmantots prakses vietā.

**3.1.2. Jenkins**

**Apraksts**:

Jenkins ir atvērtā koda automatizācijas serveris, kurš tiek izmantots lietojumprogrammu veidošanai, testēšanai un izvietošanai. Tas piedāvā elastīgu un paplašināmu platformu nepārtrauktai integrācijai un nepārtrauktai izsūtīšanai (CI/CD). Jenkins ļauj automatizēt dažādas programmatūras izstrādes posmus, ieskaitot koda kompilāciju, testēšanu un izvietošanu, tādējādi optimizējot izstrādes un publicēšanas procesu.

**Izvēles pamatojums:**

Jenkins tika izvēlēts pateicoties tam, ka tam ir lieliska pielāgojamība, plaša spraudņu ekosistēma un iespēja integrēties ar citiem rīkiem DevOps darbplūsmā, kā arī ikdienā tas tika plaši izmantots prakses vietā.

**3.1.3. Visual Studio Code (VS Code)**

**Apraksts**:

Visual Studio Code (VS Code) ir koda redaktors, kas nodrošina izcilu izstrādes vidi. Tas piedāvā plašu funkciju klāstu, piemēram, inteliģento koda aizpildīšanu, atkļūdošanu, versiju kontroles integrāciju un plašu paplašinājumu bibliotēku. VS Code atbalsta vairākas programmēšanas valodus un platformas, padarot to par daudzpusīgu rīku izstrādātājiem.

**Izvēles pamatojums:**

Visual Studio Code (VS Code) tika izvēlēts pateicoties tam, ka tam ir lietotājam draudzīga saskarne, plašas pielāgošanas iespējas un efektīva koda rediģēšanas funkcionalitāte. Šis rīks tika izmantots arī citos prakses un skolas projektos.

**3.1.4. SonarQube**

**Apraksts**:

SonarQube ir populārs koda kvalitātes analīzes rīks, kas palīdz izstrādātājiem agrīnā izstrādes posmā atklāt un novērst koda problēmas. Tas piedāvā plašu koda analīzes noteikumu kopu, kas sniedz atziņas par koda kļūdām, drošības trūkumiem un dublējumiem. SonarQube palīdz uzturēt koda kvalitāti, uzlabot uzturējamību un ieviest kodēšanas standartus.

**Izvēles pamatojums:**

SonarQube tika izvēlēts pateicoties tā spējai veikt koda analīzi, sniegt noderīgu atgriezenisko saiti un nodrošināt koda kvalitāti visā izstrādes ciklā. Šis rīks tika izmantots arī citos prakses projektos.

**3.1.5. Groovy**

**Apraksts**:

Groovy ir dinamiskā programmēšanas valoda kura bez problēmām integrējas ar Java. Groovy plaši tiek izmantots skriptēšanai un testēšanas automatizācijai.

**Izvēles pamatojums:**

Groovy tika izvēlēts pateicoties tam, ka tas ir viegli lasāms un saprotams, spēj rakstīt saīsinātus skriptus un piedāvā lieliskas iespējas automatizācijas uzdevumiem DevOps darbplūsmā.

**3.1.6. AWS (Amazon Web Services)**

**Apraksts**:

AWS (Amazon Web Services) ir populāra mākoņa skaitļošanas platforma, kas piedāvā plašu pakalpojumu klāstu lietojumprogrammu un infrastruktūras veidošanai, izvietošanai un pārvaldībai. Tas nodrošina elastīgus un mērogojamus resursus, tai skaitā virtuālos serverus, datu glabāšanu, datu bāzes un tīklošanu, kas ļauj izveidot un palaist lietojumprogrammas mākoņa vidē.

**Izvēles pamatojums:**

AWS (Amazon Web Services) tika izvēlēts pateicoties tās plašajam pakalpojumu spektram, drošībai, elastībai un uzticamībai, kas ļauj veiksmīgi īstenot lietojumprogrammu izstrādes un darbplūsmas vajadzības mākoņa vidē.

**3.2. Iespējamo (alternatīvo) risinājuma līdzekļu un valodu apraksts**

Šie alternatīvie rīki piedāvā līdzīgas funkcionalitātes iepriekš minētajiem rīkiem, sniedzot dažādas iespējas, kas atbilst konkrētajām vajadzībām un priekšrocībām.

**3.2.1 GitLab**

**Apraksts**:

GitLab ir tīmekļa bāzēta DevOps platforma, kas nodrošina versiju kontroli, nepārtrauktu integrāciju un nepārtrauktu izsūtīšanu (CI/CD). Līdzīgi kā GitHub, tas piedāvā kodu pārvaldīšanu, problēmu izsekošanu un koda pārskatu funkcijas. GitLab arī nodrošina iebūvētas CI/CD darbplūsmas, padarot to par visaptverošu alternatīvu GitHub.

A picture containing text, screenshot, font, number

Description automatically generated

**1.attēls GitHub un GitLab salīdzinājums**

**3.2.2 Travis CI**

**Apraksts**:

Travis CI ir nepārtrauktas integrācijas platforma, kas ļauj viegli veidot, testēt un izvietot programmatūras projektus. Tas labi integrējas ar GitHub un piedāvā vienkāršu konfigurāciju un uzstādīšanu. Travis CI atbalsta dažādas programmēšanas valodas un nodrošina automātiskas izvietošanas iespējas uz dažādiem mākoņu risinājumiem.

A picture containing text, screenshot, font, number

Description automatically generated

**2.attēls Jenkins un Travis CI salīdzinājums**

**3.2.3 IntelliJ IDEA**

**Apraksts**:

IntelliJ IDEA ir integrēta izstrādes vide (IDE), kas atbalsta vairākas programmēšanas valodas un ietvarus. Tā piedāvā uzlabotas koda rediģēšanas funkcijas, inteliģentu koda aizpildīšanu, atkļūdošanas rīkus un iebūvētu versiju kontroles integrāciju. IntelliJ IDEA ir pazīstama ar savu spēcīgo spraudņu ekosistēmu un izcilajām produktivitātes funkcijām.

A picture containing text, screenshot, font, number

Description automatically generated

**3.attēls Visual Studio Code un IntelliJ IDEA salīdzinājums**

**3.2.4 Google Cloud Platform**

**Apraksts**:

Google Cloud Platform ir Google piedāvāts mākoņu skaitļošanas pakalpojumu klāsts. Tas nodrošina dažādas rīkus un pakalpojumus lietojumprogrammu veidošanai, izvietošanai un pārvaldībai mākoņos. GCP piedāvā dažādas funkcijas, piemēram, virtuālās skaitļošanas mašīnas, datu glabāšanu, datu bāzes un mašīnmācīšanās pakalpojumus.

A table with black and white text

Description automatically generated with low confidence

**4.attēls Amazon Web Services un Google Cloud Platform salīdzinājums**

**3.2.5 ESLINT**

**Apraksts**:

ESLint ir populārs atvērtā koda JavaScript rīks, kas palīdz atklāt un labot programmatūras koda kļūdas un uzturēt koda kvalitāti. Tas piedāvā pielāgojamu noteikumu kopu un atbalsta dažādus koda stilus un ietvarus. ESLint integrējas ar lielāko daļu koda redaktoru un būvēšanas sistēmu, padarot to par daudzpusīgu alternatīvu SonarQube koda analīzei un kvalitātes nodrošināšanai.

A picture containing text, screenshot, font, number

Description automatically generated

**5.attēls SonarQube un ESLint salīdzinājums**

**3.2.6 Python**

**Apraksts**:

Python ir daudzpusīga un plaši izmantota programmēšanas valoda, kas pazīstama ar savu vienkāršību un vieglo lasāmību. Tā piedāvā lielu bibliotēku un ietvaru ekosistēmu, kas ļauj ātri veidot un automatizēt programmatūru. Python var izmantot dažādās jomās, tostarp skriptēšanai, tīmekļa izstrādei un datu analīzei.

A table with black and white text

Description automatically generated with low confidence

**6.attēls Groovy un Python salīdzinājums**

**4. Sistēmas modelēšana un projektēšana**

Šajā dokumenta sadaļā tiek aprakstīti dažādi diagrammu veidi, kas tiek izmantoti šajā projektā, lai vizualizētu projekta struktūru, darbības plūsmu un attiecības starp projekta daļām un komponentiem. Šīs diagrammas sniedz ieskatu projekta iekšējās darbības un palīdz saprast, kā dažādi komponenti mijiedarbojas un veido projekta kopējo sistēmu.

**4.1. Sistēmas struktūras modelis**

Sistēmas struktūras diagramma tiek izmantota, lai vizuāli parādītu sistēmas komponentu struktūru un to savstarpējo saistību. Šī diagramma sniedz ieskatu par to, kā sistēmas elementi ir saistīti un kā tie veido sistēmas kopējo struktūru.

A picture containing text, line, glasses, screenshot

Description automatically generated

**7.attēls. Sistēmas struktūras modeļa diagramma**

**4.2. Implementācijas diagramma**

Implementācijas diagramma, kas attēlo divu darbplūsmu procesus: “deploy.pipeline” un "build.pipeline". "deploy.pipeline" tiek aktivizēts manuāli vai automātiski un sastāv no dažādiem posmiem, piemēram, "Terraform init", "Terraform destroy" un "Create EC2 Instance and Deploy docker". Tas ietver pēcapstrādes darbības darbplūsmas veiksmes un kļūmes gadījumā "build.pipeline" tiek aktivizēts pēc koda izmaiņām vai manuālas palaišanas un ietver posmus, kas paredzēti SonarQube analīzei, pakotņu uzstādīšanai, Next.js lietotnes būvēšanai, kvalitātes vārtu pārbaudei, Docker attēla izveidei un būvējuma augšupielādei. Līdzīgi kā deploy darbplūsma, arī šajā gadījumā ir iekļautas pēcapstrādes darbības veiksmes un kļūmes gadījumā.

**A picture containing diagram, line, text

Description automatically generated**

**8.attēls. Implementācijas diagramma**

**5. Lietotāju ceļvedis**

Šajā sadaļā tiek aprakstīts lietotāja ceļvedis par to, kā izmantot izveides darbplūsmu Jenkins vidē, izvietošanas darbplūsmu Jenkins vidē un kā piekļūt izvietotajai programmatūrai.

**5.1. Izveides darbplūsmas ceļvedis**

1. Piekļūt Jenkins:

- Atveriet tīmekļa pārlūku un dodieties uz Jenkins vietni, kuru esat saņēmuši.

- Pierakstieties, izmantojot savus akreditācijas datus.

2. Pārvietoties uz būvēšanas darbplūsmu:

- Jeknins informācijas panelī atrodiet būvēšanas darbplūsmas projektu, ko vēlaties izmantot.

- Noklikšķiniet uz darbplūsmas nosaukuma vai pārvietojieties uz projekta lapu.

3. Sākt darbplūsmu:

- Lai sāktu darbplūsmu, noklikšķiniet uz “Build now” pogas.

- Darbplūsma sāks izpildīties, sekojot secīgiem posmiem.

4. Uzraudzīt darbplūsmas izpildi:

- Darbplūsmas izpildes laikā varat uzraudzīt katra posma progresu.

- Jenkins sniedz reāllaika atjauninājumus par katra posma statusu.

5. Skatīt būvēšanas darbplūsmas rezultātus:

- Kad darbplūsma ir pabeigta, varat skatīt kopējos rezultātus un statusu.

- Jenkins parasti sniedz kopsavilkumu lapu ar informāciju par būvēšanu, testu rezultātiem un radītajiem artefaktiem.

Tas viss! Sekojot šiem soļiem, jūs varat izmantot izveidošoanas darbplūsmu Jenkins vidē.

A screenshot of a computer

Description automatically generated

**9.attēls. Izveides darbplūsma**

**5.2. Izvietošanas darbplūsmas ceļvedis**

Lietotāja ceļvedis par to, kā izmantot izvietošanas darbplūsmu Jenkins vidē.

1. Piekļūt Jenkins:

- Atveriet tīmekļa pārlūku un dodieties uz Jemkins vietni, kuru esat saņēmuši.

- Pierakstieties, izmantojot savus akreditācijas datus.

2. Pārvietoties uz izvietošanas darbplūsmu:

- Jenkins informācijas panelī atrodiet izvietošanas darbplūsmas projektu, ko vēlaties izmantot.

- Noklikšķiniet uz darbplūsmas nosaukuma vai pārvietojieties uz projekta lapu.

3. Sākt izvietošanu:

- Lai sāktu izvietošanas procesu, noklikšķiniet uz “Build now" pogas.

- Darbplūsma sāks izpildīties, sekojot secīgiem posmiem.

4. Uzraudzīt izvietošanas darbplūsmas izpildi:

- Izvietošanas procesa laikā varat uzraudzīt katra posma progresu.

- Jenkins sniedz reāllaika atjauninājumus par katra posma statusu.

5. Skatīt izvietošanas rezultātus:

- Kad izvietošana ir pabeigta, varat skatīt kopējos rezultātus un statusu.

- Jenkins parasti sniedz kopsavilkumu lapu ar informāciju par izvietoto lietojumprogrammu, vides statusu un attiecīgajiem žurnāla ierakstiem.

Tas viss! Sekojot šiem soļiem, jūs varat izmantot izvietošanas darbplūsmu Jenkins vidē.

A screenshot of a computer

Description automatically generated with medium confidence

**10.attēls. Izvietošanas darbplūsma**

**5.3. Izvietotās programmas piekļūšanas ceļvedis**

Lietotāja ceļvedis par to, kā piekļūt izvietotajai programmai tīmekļa vidē.

1. Piekļūt Amazon Web Services:

- Atveriet tīmekļa pārlūku un dodieties uz Amazon Web Services vietni, kuru esat saņēmuši.

- Pierakstieties, izmantojot savus akreditācijas datus.

2. Pārvietoties uz EC2 instanču sadaļu:

- Meklēšanas logā ieraksties EC2.

- No rezultātiem izvēlaties pirmo un noklikšķiniet.

3. Noskaidrot vietnes adresi:

- Sameklēt EC2 instanci, kuras statuss ir aktīvs un nosaukums ir "App deploy".

- Attiecīgajai EC2 instancei atrast IPv4 adresi.

- Apmeklēt http://${atrastāIPv4adrese}:3000

4. Skatīt izvietošanas rezultātus:

- Kad vietne ir atvērta, varat aplūkot izvietoto programmatūru.

Tas viss! Sekojot šiem soļiem, jūs varat izmantot izvietošanas darbplūsmu Jenkins vidē.

A screenshot of a computer

Description automatically generated with medium confidence

**11.attēls. Amazon Web Services informācijas panelis**

A screenshot of a computer

Description automatically generated with medium confidence

**12.attēls. Izvietotā parauga programmatūra**

**6. Testēšanas dokumentācija**

Šajā dokumenta sadaļā ir apkopota pārskatāma un viegli uztverama informācija par šī projekta testēšanas procesu. Šī sadaļa sastāv no izvēlētas un alternatīvas testēšanas metodes apraksta, kā arī no testpiemēru kopas un testēšanas žurnāla.

**6.1. Izvēlētās testēšanas metodes, rīku apraksts un pamatojums**

Šim projektam es veicu rūpīgu manuālo testēšanu no lietotāja saskarnes puses. Manuālais testēšanas veids no lietotāja saskarnes puses ir vislabāk piemērots šāda veida projektiem, jo tas ļauj visaptveroši novērtēt lietotāja saskarni, atklāt vizuālas neatbilstības, veikt izpētes testēšanu un ir samērā vienkārša testēšanas metode, salīdzinot ar citām testēšanas metodēm. Izmantojot manuālo testēšanu, es nodrošināju augstu kvalitāti un lietojamību lietotnē, uzlabojot lietotāju pieredzi..

Manuālā testēšana ir programmatūras testēšanas metode, ko veic testētāji bez automātizācijas rīkiem. Testētāji izpilda testa scenārijus, atklāj defektus un pārbauda programmatūras funkcionalitāti saskaņā ar iepriekš definētajiem prasību izmēriem. Tā nodrošina detalizētāku izpratni par lietotnes darbību, bet var būt laikietilpīga un pakļauta daudzām neuzmanības kļūdām.

**6.2. Testēšanas pārskats**

**A picture containing text, screenshot, parallel, number

Description automatically generated**

**13.attēls. Testēšanas pārskats Nr.1**

**A picture containing text, screenshot, line, parallel

Description automatically generated**

**14.attēls. Testēšanas pārskats Nr.2**

**Secinājumi**

**Automatizācija un efektivitāte:**

Projekta galvenais mērķis ir automatizēt izvietošanas, testēšanas un aplikācijas būvēšanas procesus ražošanas plūsmā. Izmantojot dažādus rīkus, es vēlējos optimizēt šos uzdevumus un paaugstināt kopējo efektivitāti.

**DevOps prakse:**

Ir svarīgi uzsvērt, ka šo projektu es izstrādāju, strādājot kā DevOps praktikants. Šī pieredze apliecina manas praktiskās prasmes un spēju sniegt ieguldījumu reālo projektu veidošanā DevOps jomā.

**Rīku izvēle:**

Projekts izmanto piecus konkrētus rīkus - GitHub, Jenkins, Visual Studio Code, AWS, SonarQube un Groovy. Katrs rīks kalpo noteiktai funkcijai ražošanas plūsmā, un es tos izvēlējos, pamatojoties uz to funkcionalitāti un saderību ar projektu.

**Vēlamais rezultāts:**

Projekta galvenais mērķis ir izveidot stabilu un automatizētu ražošanas plūsmu. Izmantojot atlasītos rīkus, es vēlējos panākt kļūdu un problēmu brīvu izvietošanu, efektīvu testēšanu un veiksmīgu aplikācijas būvēšanu.

**Sasniegtā novērtēšana:**

Projekta veiksmi var novērtēt, vērtējot automatizācijas procesu efektivitāti un efektivitāti. Galvenie rādītāji var būt izvietošanas laika samazināšanās, uzlabota testēšanas precizitāte, stabilāka būvēšanas procesa stabilitāte un palielināta produktivitāte attīstības komandai.

Kopumā projekts risina automatizācijas nepieciešamību ražošanas plūsmā, un, izmantojot noteiktus rīkus, es centos izveidot efektīvu un optimizētu darba plūsmu. DevOps praksē esmu ieguvis praktisku pieredzi, kas dod projektam papildu vērtību.

**Literatūras un informācijas avotu saraksts**

* <https://api.slack.com/> (API Slack)
* <https://aws.amazon.com/documentation> (AWS Documentation)
* <https://www.digitalocean.com/> (DigitalOcean)
* <https://medium.com/> (Medium)
* <https://docs.sonarqube.org/latest/> (SonarQube Documentation)
* <https://stackoverflow.com/> (StackOverflow)

**Pielikumi**

A screenshot of a computer

Description automatically generated with medium confidence

**1.pielikums AWS Elastic Container Registry ar būvējumiem.** **A screenshot of a computer

Description automatically generated with medium confidence**

**2.pielikums AWS Elastic Container Registry repozitorīji.**

<http://16.16.172.225:9000/> admin Koprade2023

**3.pielikums Pieejas dati SonarQube serverim**

[http://16.16.146.139:8080/](http://16.16.172.225:9000/) martinsbardulis Koprade2023

**4.pielikums Pieejas dati Jenkins serverim**

<https://eu-north-1.console.aws.amazon.com/ec2/home?region=eu-north-1> userfortestmartins.b@gmail.comKoprade2023

**5.pielikums Pieejas dati Amazon Web Service**

<https://docs.google.com/spreadsheets/d/1xQLLwCDJQCCnyOGJ7wvvSPSl4Fxd-HEKty-4Z9MiQAc/edit?usp=sharing>

**6.pielikums Testēšanas pārskats**

<https://github.com/Martins-B/Kursa_darbs_Bardulis_4PT>

**4.pielikums Pieejas dati Jenkins serverim**