Shape

Description automatically generated with medium confidence

Liepājas Valsts tehnikums

**Būvēšanas izvietošanas līnija**

Kvalifikācijas eksāmena prakstiskās daļas dokumentācija

Profesionālā kvalifikācija Programmēšanas tehniķis

Grupas nosaukums 4PT

Projekta izstrādātājs Marians Ziemelis

Eksāmena datums 2023. gada 21.jūnijs

Liepāja 2023

**Satura rādītājs**

[Ievads 4](#_Toc137420478)

[1. Uzdevuma formulējums 5](#_Toc137420479)

[2. Programmatūras prasību specifikācija 6](#_Toc137420480)

[2.1. Produkta perspektīva 6](#_Toc137420481)

[2.2. Sistēmas funkcionālās prasības 6](#_Toc137420482)

[2.2.2. Jānodrošina automatizēta programmatūras būvēšana 7](#_Toc137420483)

[2.2.3 Jānodrošina automatizēta programmatūras izvietošana. 8](#_Toc137420484)

[2.2.4 Jānodrošina manuāla programmatūras izvietošana. 8](#_Toc137420485)

[2.2.5 Veikt kodu analīzi, izmantojot SonarQube. 9](#_Toc137420486)

[2.2.6 Izveidot Docker attēlus 9](#_Toc137420487)

[2.2.7 Izvietot aplikāciju, izmantojot Terraform un AWS. 10](#_Toc137420488)

[2.2.8 Veikt paziņojumu nosūtīšanu komunikācijas lietotnē Slack 10](#_Toc137420489)

[2.3. Sistēmas nefunkcionālās prasības 11](#_Toc137420490)

[2.4. Gala lietotāja raksturiezīmes 11](#_Toc137420491)

[3. Izstrādes līdzekļu, rīku apraksts un izvēles pamatojums 12](#_Toc137420492)

[3.1. Iespējamo risinājuma līdzekļu un valodu apraksts 12](#_Toc137420493)

[3.2. Izvēlēto risinājuma līdzekļu un valodu apraksts 13](#_Toc137420494)

[4. Sistēmas modelēšana un strukturēšana 14](#_Toc137420495)

[4.1. Sistēmas struktūras modulis 14](#_Toc137420496)

[4.2. Klašu diagramma 15](#_Toc137420497)

[4.3. Funkcionālais un dinamiskais sistēmas modulis 15](#_Toc137420498)

[4.4. Aktivitāšu diagramma 17](#_Toc137420499)

[4.5. Lietujmgadījumu diagramma 17](#_Toc137420500)

[4.6. Sistēmas moduļu apraksts un algoritmu shēmas 18](#_Toc137420501)

[5. Lietotāju ceļvedis 19](#_Toc137420502)

[6. Testēšanas dokumentācija 22](#_Toc137420503)

[6.1. Izvēlētās testēšanas metodes, rīku apraksts un pamatojums 22](#_Toc137420504)

[6.2. Testpiemēru kopa 22](#_Toc137420505)

[6.3. Testēšanas žurnāls 23](#_Toc137420506)

[Secinājumi 24](#_Toc137420507)

[Lietoto saīsinājumu saraksts 25](#_Toc137420508)

[Literatūras un informācijas avotu saraksts 26](#_Toc137420509)

[Pielikums 27](#_Toc137420510)

# **Ievads**

Būvēšanas-izvietošanas līnija ir automatizācijas process, kas nodrošina programmatūras izstrādes un izvietošanas procesu efektivitāti un kvalitāti. Tas ļauj izstrādātājiem un administratoriem automātiski būvēt, testēt un izvietot programmatūru, samazinot cilvēka faktora iespējas un novēršot kļūdas.

Tā kā programmatūras izstrāde un izvietošana ir sarežģīti un laikietilpīgi procesi, šī programma sniedz vērtīgu risinājumu uzņēmumiem, kuriem nepieciešams strādāt efektīvi un ātri, vienlaikus nodrošinot augstu programmatūras kvalitāti. Tas palīdz izstrādātājiem ātri pielāgoties izmaiņām un ātrāk sasniegt uzņēmējdarbības mērķus.

Šajā dokumentā ir aprakstītas programmatūras prasību specifikācijas, izstrādes līdzekļu un rīku apraksts un izvēles pamatojums, kā arī sistēmas modelēšana un projektēšana, lietotāju ceļvedis, testēšanas dokumentācija, individuālais ieguldījums un secinājumi par būvēšanas-izvietošanas līnijas implementāciju.

# **1. Uzdevuma formulējums**

Uzdevuma formulējums būvēšanas – izvietošanas līnijas projekta darbam ir izveidot automatizētu būvēšanas un izvietošanas procesu, izmantojot dažādus rīkus. Galvenais mērķis ir uzlabot programmatūras izstrādes procesu, samazinot cilvēka darbības ietekmi un samazinot kļūdu skaitu, kas rodas manuālo darbību laikā.

Šī darba sasniedzamais rezultāts ir veiktās būves un to izvietošanas pārvaldība, izmantojot būvēšanas-izvietošanas līniju, kā arī nodrošināt to pareizu darbību. Mērķis tiks uzskatīts par sasniegtu, ja programmatūra tiek veiksmīgi uzbūvēta un izvietota uz mērķa vides, izmantojot automatizētu būvēšanas-izvietošanas līniju, izmantojot dažādus rīkus.

# **2. Programmatūras prasību specifikācija**

## **2.1. Produkta perspektīva**

Produkta perspektīva ir izveidot būvēšanas-izvietošanas līniju, lai automatizētu programmatūras izstrādes procesu un tās izvietošanu uz servera, lai tas būtu integrējams ar Jenkins sistēmu, lai tas nodrošinātu automātisku pārbaudi un testēšanu katrā attīstības posmā un tas jāizveido, lai nodrošinātu, ka programma atbilst uzstādītajiem standartiem un prasībām.

## **2.2. Sistēmas funkcionālās prasības**

**2.2.1 Jānodrošina Jenkins rīka integrācija un tās konfigurēšana.**

Mērķis: Jenkins rīka uzstādīšana ar vajadzīgajiem parametriem.

Ievaddati:

**1. tabula**

**Programmatūras dati**

|  |  |  |
| --- | --- | --- |
| Nosaukums | Obligāts | Piezīmes |
| Slack, MS Teams vai cita veida komunikācijas veida webhooks | Nē | Uzglabāsies šifrētā formā |
| AWS, Google Cloud, Digital Ocean, u.c. servisu autentifikācijas tokeni | Jā | Uzglabāsies šifrētā formā |
| SonarQube autentifikācijas tokeni(-s) | Nē | Uzglabāsies šifrētā formā |

Apstrāde:

Palaižot kādu no darbiem, Jenkins spēj šos parametrus padot un izmantot tos darba soļos. Bez Github repozitorijas adreses Jenkins nespēj iegūt darba soļus, un nekas nespēj notikt, kā arī bez dockera repozitorijas adreses tā nespēj veikt versionēšanu un bez mākoņu servisu autentifikācijas tokeniem pārbaudīt vai tas spēj palaist šo programmatūru uz citas vides.

Izvaddati:

1) Paziņojums, ka nav repozitorijas adrese, palaižot darbu.

2) Kļūda, ka nespēj augšuplejuplādēt docker bildi repozitorijā.

3) Kļūda, ka nespēj izveidot mākonī serveri un palaist programmatūru.

### 2.2.2. Jānodrošina automatizēta programmatūras būvēšana

Mērķis: Funkcija nepieciešama, lai spētu automātiski uzbūvēt programmatūru, testēt to un izveidot docker bildi.

Ievaddati:

1. Koda versijas kontroles sistēma, kas satur programmatūras kodu un resursus.

2. Konfigurācijas fails ar norādījumiem par programmatūras būvēšanas darbplūsmām, piemēram, kompilāciju, pārbaudi un izvietošanu.

3. Attiecīgi uzstādītas un konfigurētas programmatūras būvēšana un testēšana.

Apstrāde:

1. Sistēma spēj automātiski būvēt programmatūru no koda versijas kontroles sistēmas izmaiņām.

2. Sistēma spēj automātiski veikt koda pārbaudi, lai novērstu nevēlamas izmaiņas.

3. Sistēma nodrošina iespēju definēt un konfigurēt programmatūras būvēšanas darba plūsmas.

4. Sistēmai jābūt spējīgai integrēties ar dažādiem koda versijas kontrolētājiem un attīstības rīkiem.

5. Sistēmai jānodrošina iespēja veikt automatizētu programmatūras izvietošanu uz izstrādātāja izvēlēto platformu.

Izvaddati:

1) Automatizēts programmatūras būvēšanas process, kas samazina darbietilpīgumu un paaugstina produktivitāti.

2)Veiksmīgi uzbūvēta un izvietota programmatūra docker repozitorijā.

### 2.2.3 Jānodrošina automatizēta programmatūras izvietošana.

Mērķis: Nodrošināt automatizētu programmatūras izvietošanu uz attiecīgās platformas, lai samazinātu darbietilpīgumu un paaugstinātu produktivitāti.

Ievaddati:

1. Kompilētā programmatūra, kas ir gatava izvietošanai.

2. Konfigurācijas fails ar norādījumiem par izvietošanas procesa darbplūsmām.

3. Pieeja un atļaujas attiecīgajai platformai, uz kuras būs jāizvieto programmatūra.

Apstrāde:

1. Sistēmai jābūt spējīgai automātiski izvietot programmatūru uz norādītās platformas.

2. Sistēmai jābūt spējīgai integrēties ar konkrēto platformu un to specifisko izvietošanas prasībām.

3. Sistēmai jānodrošina iespēja definēt un konfigurēt izvietošanas darba plūsmas.

Izvaddati:

1) Automatizēts programmatūras izvietošanas process, kas samazina darbietilpīgumu un paaugstina produktivitāti.

2) Veiksmīgi izvietota programmatūra uz attiecīgās platformas un gatava darbam.

### 2.2.4 Jānodrošina manuāla programmatūras izvietošana.

Mērķis: Nodrošināt manuālu programmatūras izvietošanu uz attiecīgās platformas, lai pārbaudītu kļūdas un manuāli palaistu programmas izvietošanu ar specifisko programmatūras versiju.

Ievaddati:

1. Kompilētā programmatūra, kas ir gatava izvietošanai.

2. Konfigurācijas fails ar norādījumiem par izvietošanas procesa darbplūsmām.

3. Pieeja un atļaujas attiecīgajai platformai, uz kuras būs jāizvieto programmatūra.

4. Programmatūras versija, kura ir uzbūvēta un pieejama.

Apstrāde:

1. Sistēmai jābūt spējīgai automātiski izvietot programmatūru uz norādīto platformu.

2. Sistēmai jābūt spējīgai integrēties ar konkrēto platformu un to specifisko izvietošanas prasībām.

3. Sistēmai jānodrošina iespēja definēt un konfigurēt izvietošanas darba plūsmas.

4. Sistēmai jānodrošina ziņojums par līnijas kļūdām un izdevušos izvietošanu ar programmas interneta adresi uz “Slack”.

Izvaddati:

1) Manuāls programmatūras izvietošanas process, kas samazina darbietilpīgumu un paaugstina produktivitāti.

2) Veiksmīgi izvietota programmatūra uz attiecīgo platformu un gatava darbam.

### 2.2.5 Veikt kodu analīzi, izmantojot SonarQube.

Mērķis: Uzlabot kodu kvalitāti, identificēt potenciālos riskus un veikt statiskās analīzes pārbaudes.

Ievaddati:

1. SonarQube konfigurācijas faili un piekļuves atslēga.

2. Next.js aplikācijas kods.

Apstrāde:

1. Konfigurēt SonarQube kodu analīzes vidi.

2. Izpildīt SonarQube analīzes skanējumu, izmantojot iepriekš konfigurēto vidi.

Izvaddati:

1) SonarQube analīzes rezultātu pārskats, kas parāda koda kvalitāti un potenciālos riskus.

### 2.2.6 Izveidot Docker attēlus

Mērķis: Izveidot Docker attēlus, kas satur Frontend aplikāciju.

Ievaddati:

1. Dockerfile, kas definē vajadzīgās Docker attēla būvniecības instrukcijas.

2. Next.js aplikācijas izpildāmais fails.

3. Jenkins būvēšanas numurs.

Apstrāde:

1. Izveidot Docker attēlu, izmantojot Dockerfile un definēto būvēšanas numuru.

Izvaddati:

1) Izveidots Docker attēls ar atbilstošu tagu.

### 2.2.7 Izvietot aplikāciju, izmantojot Terraform un AWS.

Mērķis: Automatizēt aplikācijas izvietošanu AWS infrastruktūras resursos, izmantojot Terraform.

Ievaddati:

1. Terraform konfigurācijas faili.

2. AWS EC2 atslēga.

3. Iepriekš izveidota Docker bilde.

Apstrāde:

1. Inicializēt Terraform projektu.

2. Izpildīt Terraform, lai izvietotu aplikāciju, pamatojoties uz definēto EC2 atslēgu un Docker bildi.

Izvaddati:

1) Izvietotās aplikācijas IP adrese.

### 2.2.8 Veikt paziņojumu nosūtīšanu komunikācijas lietotnē Slack

Mērķis: Ziņot par darbībām un izvades rezultātiem Slack kanālā, lai nodrošinātu informācijas pārraudzību un ziņošanu komandai.

Ievaddati:

1. Slack konfigurācijas informācija.

2. Paziņojumi par veiksmīgu vai neveiksmīgu būvēšanas un izvietošanas darbību.

Apstrāde:

1. Nosūtīt Slack paziņojumus, norādot veiksmīgu izvietošanas līnijas pabeigšanu vai par to, ka ir notikusi kļūda, paziņojumā ietilpst izvietotās programmatūras adrese un kļūdas.

Izvaddati:

1) Paziņojumi par kļūdām un līnijas beigām.

## **2.3. Sistēmas nefunkcionālās prasības**

1. Visi pieejas tokeni un paroles ir saglabātas šifrētas.

2. Konsolē visas adreses tiek šifrētas un saglabātas šifrētas, pat ja kāds tiek klāt serverim.

3. Programmatūra ir uzrakstīta angļu valodā.

## **2.4. Gala lietotāja raksturiezīmes**

Sistēma ir paredzēta galvenokārt IT speciālistiem. Lietotājs var konfigurēt būvēšanas un implementācijas procesus pēc saviem uzskatiem. Lietotājs spēj konfigurēt būvēšanas un izlaišanas posmus. Lietotājs spēj iegūt detalizētu informāciju par kļūdām un pārbaudēm.

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

## **3.1. Iespējamo risinājuma līdzekļu un valodu apraksts**

Jenkins - atvērtā koda būvēšanas izpildītāja platforma, kas ļauj veikt automatizētu kodu būvēšanu, testēšanu un izvietošanu uz vairākiem serveriem un mākoņu platformām.

Docker - atvērtā koda konteineru platforma, kas ļauj izstrādāt, testēt un izvietot aplikācijas vienotā konteinerā. Tas palīdz samazināt izmaksas un nodrošina lielāku elastību.

Ansible - atvērtā koda konfigurācijas pārvaldības platforma, kas ļauj automatizēt infrastruktūras konfigurāciju un izvietošanu.

Terraform - atvērtā koda infrastruktūras koda pārvaldības rīks, kas ļauj veikt infrastruktūras automatizāciju un izvietošanu, izmantojot vienkāršu un deklaratīvu valodu.

Kubernetes - atvērtā koda konteineru orķestrēšanas rīks, kas ļauj automatizēt un pārvaldīt lielu skaitu konteineru un to izvietošanu uz vairākiem serveriem vai mākoņu platformām.

JavaScript - dinamiska, objektorientēta programmēšanas valoda, kas visbiežāk tiek izmantota tiešsaistes lietotņu un tīmekļa vietņu izstrādei.

Jenkins Declarative Pipeline Syntax - valoda, kas ļauj definēt Jenkins būvēšanas un izvietošanas darba plūsmas deklaratīvi, izmantojot YAML sintaksi. Tas nodrošina konfigurācijas pārvaldības priekšrocības, ieskaitot atkārtotu izmantošanu un pārvaldību versijās, vienlaikus nodrošinot skaidru un saprotamu plūsmu.

AWS - mākoņu tehnoloģiju uzņēmums, kas nodrošina tādus pakalpojumus kā infrastruktūras, kā koda, datu glabāšanas un analīzes risinājumi, datoru jaudas pirkšanu un daudz ko citu.

Google Cloud - mākoņu tehnoloģiju uzņēmums, kas nodrošina tādus pakalpojumus kā datu glabāšanas un analīzes risinājumi, datoru jaudas pirkšanu un daudz ko citu.

SonarQube - atvērtā koda kvalitātes pārbaudes platforma, kas palīdz novērst kodu kļūmes un uzlabot tā izpildīšanas veiktspēju. SonarQube nodrošina automatizētu koda pārbaudi, kas palīdz noteikt potenciālās problēmas un ļauj tās novērst pirms programmatūra tiek izlaista.

## **3.2. Izvēlēto risinājuma līdzekļu un valodu apraksts**

Jenkins Declarative Pipeline Syntax - valoda, kas ļauj definēt Jenkins būvēšanas un izvietošanas darba plūsmas deklaratīvi, izmantojot YAML sintaksi. Tas nodrošina konfigurācijas pārvaldības priekšrocības, ieskaitot atkārtotu izmantošanu un pārvaldību versijās, vienlaikus nodrošinot skaidru un saprotamu plūsmu.

Docker - atvērtā koda konteineru platforma, kas ļauj izstrādāt, testēt un izvietot aplikācijas vienotā konteinerā.

AWS - mākoņu tehnoloģiju uzņēmums, kas nodrošina tādus pakalpojumus kā infrastruktūras, kā koda, datu glabāšanas un analīzes risinājumi, datoru jaudas pirkšanu un daudz ko citu.

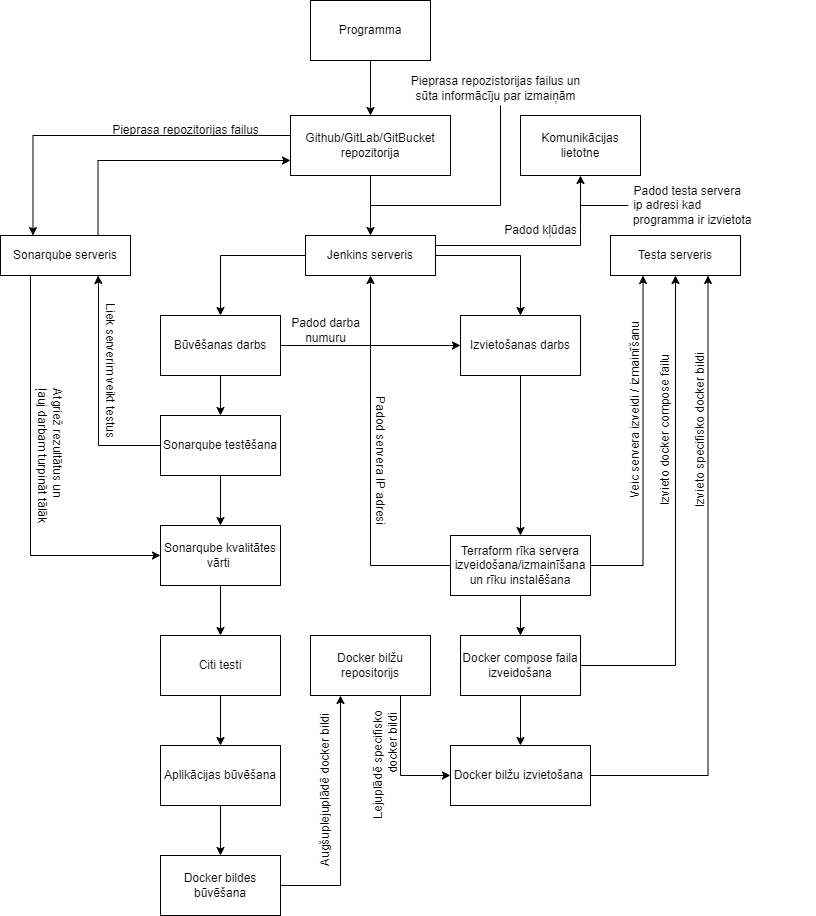
Jenkins - atvērtā koda būvēšanas izpildītāja platforma, kas ļauj veikt automatizētu kodu būvēšanu, testēšanu un izvietošanu uz vairākiem serveriem un mākoņu platformām.

SonarQube - atvērtā koda kvalitātes pārbaudes platforma, kas palīdz novērst kodu kļūmes un uzlabot tā izpildīšanas veiktspēju. SonarQube nodrošina automatizētu koda pārbaudi, kas palīdz noteikt potenciālās problēmas un ļauj tās novērst pirms programmatūra tiek izlaista.

Terraform - atvērtā koda infrastruktūras koda pārvaldības rīks, kas ļauj veikt infrastruktūras automatizāciju un izvietošanu, izmantojot vienkāršu un deklaratīvu valodu.

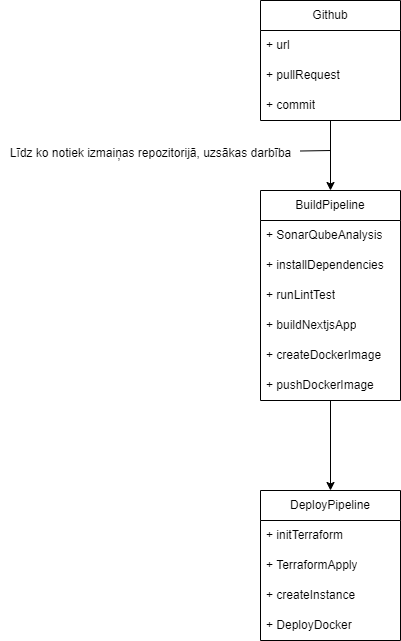
# **4. Sistēmas modelēšana un strukturēšana**

## **4.1. Sistēmas struktūras modulis**



**1. attēls. Sistēmas struktūras modelis**

**4.2. Klašu diagramma**



**2. attēls. Klašu diagramma**

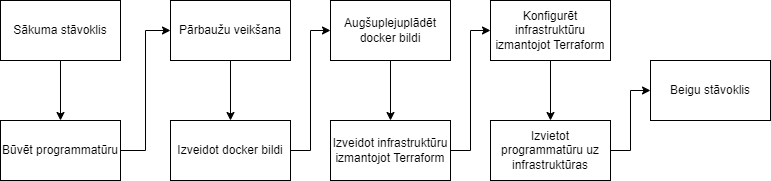
## **4.3. Funkcionālais un dinamiskais sistēmas modulis**

A picture containing text, screenshot, line, diagram

Description automatically generated

**3. attēls. Datu plūsmas diagramma**

## **4.4. Aktivitāšu diagramma**



**4. attēls. Aktivitāšu diagramma**

## **4.5. Lietujmgadījumu diagramma**

Diagram

Description automatically generated

**5. attēls. Lietojumgadījumu diagramma**

## **4.6. Sistēmas moduļu apraksts un algoritmu shēmas**

A picture containing text, screenshot, font, line

Description automatically generated

**6. attēls. Algoritmu shēma**

# **5. Lietotāju ceļvedis**

Būvēšanas izvietošanas līnijas galvenās komponentes ir Jenkins, SonarQube, AWS. Šī līnija ir domāta NextJs web aplikācijām.

Šī būvēšanas izvietošanas līnija ir izveidota tam, lai atvieglotu izstrādātāju darbus, samazinātu darbietilpību un palielināt darba efektivitāti, tamdēļ šī līnija ir pilnīgi automatizēta, tā spēj automātiski nolasīt izmaiņas Github repozitorijā un uzsākt šo darbu līniju.

Jenkins ir galvenais rīks, ar kuru var sekot līdzi izpildes līnijai un redzēt darbības, kļūdas un manuāli palaist šīs līnijas, kļūdu vai kāda cita iemesla gadījumā. Izvietošanas līnijas galā vai kļūdu gadījumā, Jenkins nosūta ziņu uz Slack ar specifisku informāciju. Izvietošanas līnijas galā, tas nosūta adresi ar izvietoto programmatūru un paziņojumu, ka līnija ir pabeigta. Kļūdas gadījumā tas nosūta līnijas numuru ar adresi uz šo līnijas kļūdu.

SonarQube ir rīks, kurš spēj automātiski testēt kodu un uzrādīt lietotājam kļūdas.

AWS ir serviss, kurā tiek izvietota šīs līnijas laikā uzbūvētā programmatūra.

Būvēšanas līnija:

Jenkins būvēšanas līnija sastāv no vairākiem posmiem, kas tiek izpildīti secīgi.

Līnija sākas ar vides un resursu konfigurāciju, piemēram, Github repozitorijas norādīšanu un tās klonēšanu, kā arī definējot Docker konteineru repozitoriju un SonarQube konfigurācijas failu atrašanās vietu.

Tālāk seko posmi, kas veic kodu analīzi ar SonarQube, instalē pakotnes un veido Next.js aplikācijas izpildāmo failu.

Pēc tam ir posms, kas gaida SonarQube kvalitātes vārtus, lai novērtētu kodu un noteiktu, vai turpināt vai pārtraukt būvēšanas procesu.

Tālāk seko posmi, kas izveido Docker bildi, nosūta to uz ECR (Elastic Container Registry) un saglabā attiecīgos tagus.

Visbeidzot, ir posms, kas izsauc izvietošanas līniju ar specifisku parametru kas nosaka, kuru versiju izvietot.

Izvietošanas līnija:

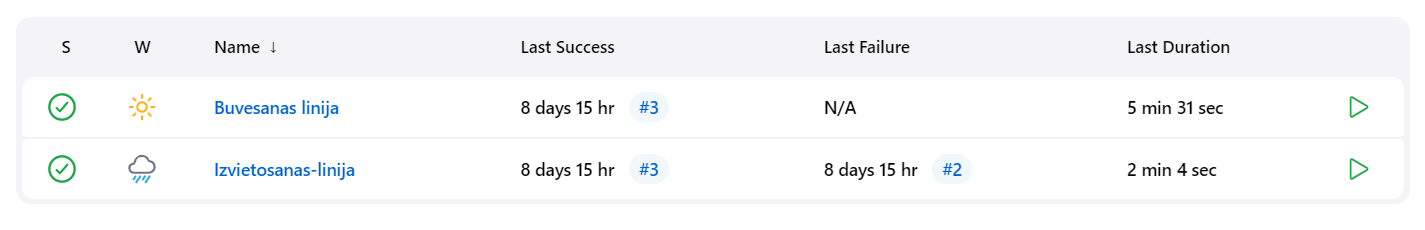
Izvietošanas līnija izmanto Docker un Terraform, lai izvietotu aplikāciju uz infrastruktūras resursiem.

Pirms izvietošanas sākuma, līnija definē vides mainīgos, piemēram, EC2 atslēgu, ECR repozitorijas informāciju un iegūst docker bildes versiju, kuru izvietot.

Pirmajā posmā līnija raksta Docker Compose failu, kas definē konteinerus un to konfigurāciju.

Nākamais posms sāk Terraform inicializācijas procesu.

Pēc tam seko posms, kas izsauc terraform apply komandu, lai veiktu infrastruktūras izveidi un docker bildes izvietošanu.

Atverot Jenkins rīku, lietotājs nonāk Jenkins sākuma lapā, kur var redzēt visus esošos darbus.  
****

**7. attēls. Jenkins sākuma lapa**

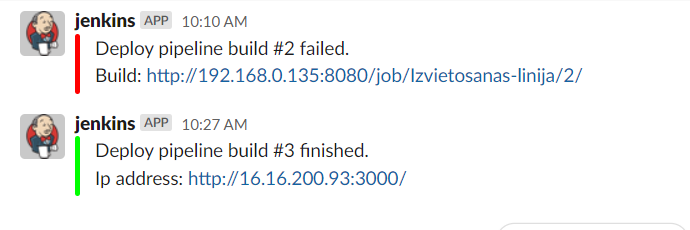
Nonākot sākuma lapā, lietotājam, uzpiežot uz darbu, atveras jauna lapa, kurā tiek detalizētāk parādīti darbi, darba soļi, to izpildes laiki un kļūdas, kā arī SonarQube kvalitāšu vārtu rezultāti.

A screenshot of a computer

Description automatically generated with medium confidence

**8. attēls. Darba sākuma lapa**

Tālāk lietotājam ir iespēja manuāli palaist darbu, apskatīt detalizētāku informāciju par darba procesu, sekot līdzi darba procesiem un redzēt kļūdas.

Lietotājam nāk paziņojumi uz Slack par darba līnijas kļūdām un izdevušos līnijas izpildi ar izvietotās programmatūras ip adresi.  


**9. attēls. Slack paziņojumi**

# **6. Testēšanas dokumentācija**

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

Testēšanai izvēlējos melnās kastes testēšanas metodi, ar kuru pārbauda programmas vai sistēmas funkcionalitāti, pamatojoties tikai uz tās ieejas un izvades rezultātiem. Testēju būvēšanas izvietošanas līnijas startēšanu un ziņojumu sūtīšanu. Testēšanai izmantoju MS Excel, lai izveidotu testpiemēru kopas un testēšanas žurnālu, un interneta pārlūkprogrammu, kā arī Github. Testēšanas mērķis ir pārbaudīt sistēmas funkcionalitāti.

## **6.2. Testpiemēru kopa**

A screenshot of a computer

Description automatically generated with low confidence

**10. attēls. Testēšanas indikātori**

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

Description automatically generated**

**11. attēls. Testēšanas prasības**

**A screenshot of a computer

Description automatically generated with low confidence  
12. attēls. Testpiemēri**

**6.3. Testēšanas žurnāls**  


**13. attēls. Testēšanas žurnāls**

# **Secinājumi**

1. Izstrādājot šo dokumentu, esmu ieguvis dziļāku izpratni par IT sistēmu izstrādes procesu. Manuprāt, grūtākais bija saprast, kā pareizi secīgi izpildīt darbības, jo daudzas funkcijas sistēmā mijas viena ar otru. Tāpēc izstrādes laikā paralēli pašai kodēšanai, veicu arī sava veida analīzi darba izpildes procesam, un bieži vien tas tiešām palīdzēja iztikt bez kādiem starpgadījumiem.
2. Es sapratu, kāpēc programmēšanas uzņēmumos ir šķietami vairāk darbinieku nekā citos gadījumos būtu nepieciešams. Tas tāpēc, ka darba apjoms ir gana liels, un, man vienam izstrādājot šāda veida produktu, pagāja ilgāks laiks nekā tas varētu būt, veicot darbu komandā. Katrā ziņā secinājums tāds, ka katrs komandas dalībnieks ir svarīgs, izstrādājot lielus projektus, laikietilpības un efektivitātes dēļ.
3. Īpašu gandarījumu man sagādāja strādāt ar rīkiem, kā Jenkins un Docker. Šo rīku integrēšana manā sistēmā ar manu kodu nebija nemaz tik vienkārši, kā sākumā šķita.
4. Ja būtu jāveic izstrāde no jauna, tad darītu visu vairāk vai mazāk tāpat, tikai kā jau augstāk minēju, organizētu savu darba vidi, laika plānojumu un darbību secību nedaudz savādāk.
5. Kopumā esmu pat ļoti apmierināts ar savu darbu. Pašā sākumā jau zināju, ka vēlos radīt kaut ko saistītu ar IT sistēmu izstrādi. Varu teikt, ka man izdevās izveidot produktu, un tas nozīmē to, ka idejai, topot par modeli un projektējumu, es spēju izstrādāt produktu, kuram ir pietiekoši liels potenciāls nākotnē tapt vēl funkcionālākam.

# **Lietoto saīsinājumu saraksts**

AWS: Amazon Web Services ir mākoņpakalpojumu sniedzējs, kas nodrošina dažādas infrastruktūras un pakalpojumus, kas nepieciešami aplikāciju izvietošanai un darbībai mākoņvidē.

Jenkins: Jenkins ir atvērtā pirmkoda automatizācijas serveris, ko izmanto, lai veiktu būvēšanas, testēšanas un izvietošanas procesus.

SonarQube: SonarQube ir rīks, kas izmantojams statiskā koda analīzei, lai novērtētu koda kvalitāti, identificētu defektus un veiktu kodolaižu analīzi.

Terraform: Terraform ir programmēšanas valoda un rīks, kas ļauj deklaratīvi aprakstīt infrastruktūru kā kodu un automatizēti pārvaldīt resursus dažādās infrastruktūras platformās, tostarp AWS.

Slack: Slack ir tīmekļa lietojumprogramma un mobilo ierīču lietojumprogramma, kas nodrošina darba grupām komunikācijas, sadarbības un failu kopīgošanas iespējas.

EC2: Amazon Elastic Compute Cloud (EC2) ir elastīgs tīmekļa serviss, kas nodrošina drošu, skalējamu un pielāgojamu jaudu, lai palaižtu un pārvaldītu virtuālās mašīnas AWS infrastruktūrā.

ECR: Amazon Elastic Container Registry (ECR) ir pakalpojums, kas nodrošina drošu uzglabāšanu, pārvaldību un izmantošanu Docker bildēm AWS platformā.

Github: Github ir vietne un pakalpojums, kas ļauj izstrādātājiem glabāt, pārvaldīt un koplietot programmatūras projektus, izmantojot Git versiju kontroles sistēmu.

Docker: Docker ir atvērtā pirmkoda platforma, kas ļauj automatizēti izveidot, izvietot un palaist aplikācijas konteineros.

# **Literatūras un informācijas avotu saraksts**

1. <https://aws.amazon.com/what-is-aws/?nc1=f_cc>
2. <https://aws.amazon.com/what-is-cloud-computing/?nc1=f_cc>
3. <https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html>
4. <https://www.techtarget.com/searchsoftwarequality/definition/Jenkins>
5. <https://www.sonarsource.com/products/sonarqube/>
6. <https://www.terraform.io/use-cases/infrastructure-as-code>
7. <https://nextjs.org/docs>
8. <https://www.ansible.com/>
9. <https://www.hpe.com/us/en/what-is/cloud-services.html>

# **Pielikums**

**1. pielikums**

Github tīmekļa adrese: <https://github.com/ziemelis/ziemelis-buvesanas-izvietosanas-linija>

“deploy.jenkinsfile” kods



**1. attēls**

**2. pielikums**

**A picture containing screenshot

Description automatically generated**

**2. attēls**

**3. pielikums**

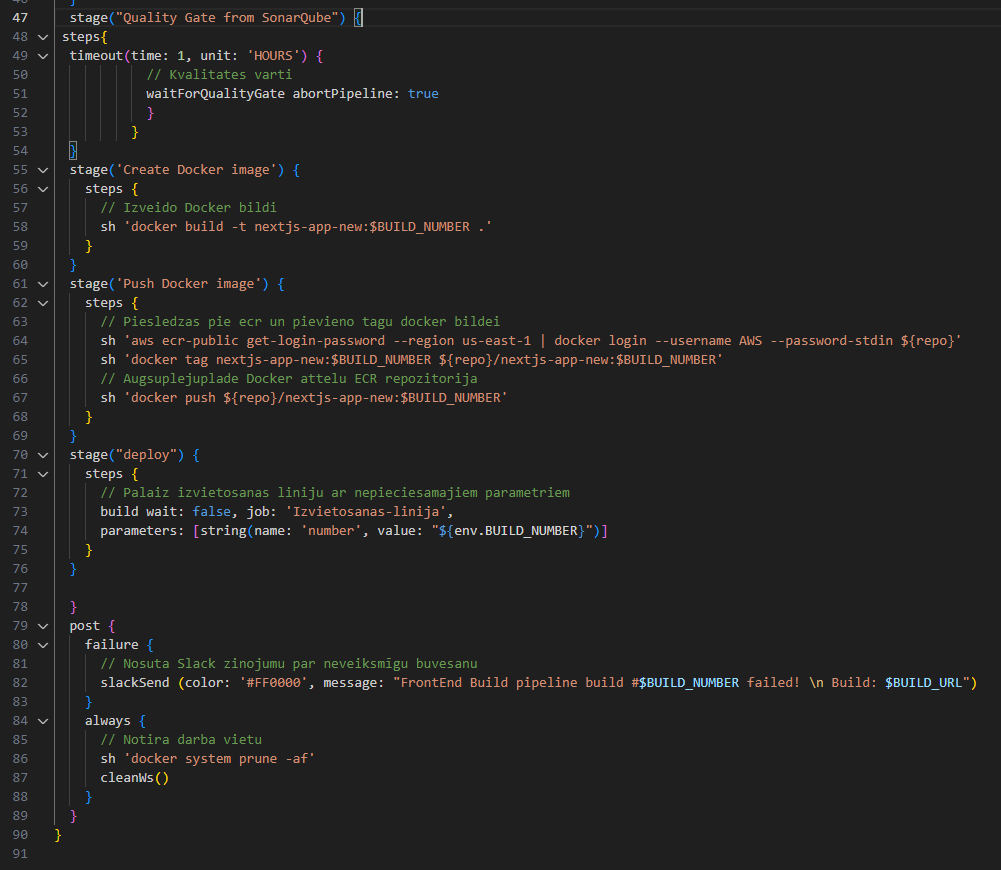
“build.jenkinsfile” kods

**A screen shot of a computer program

Description automatically generated with low confidence**

**3. attēls**

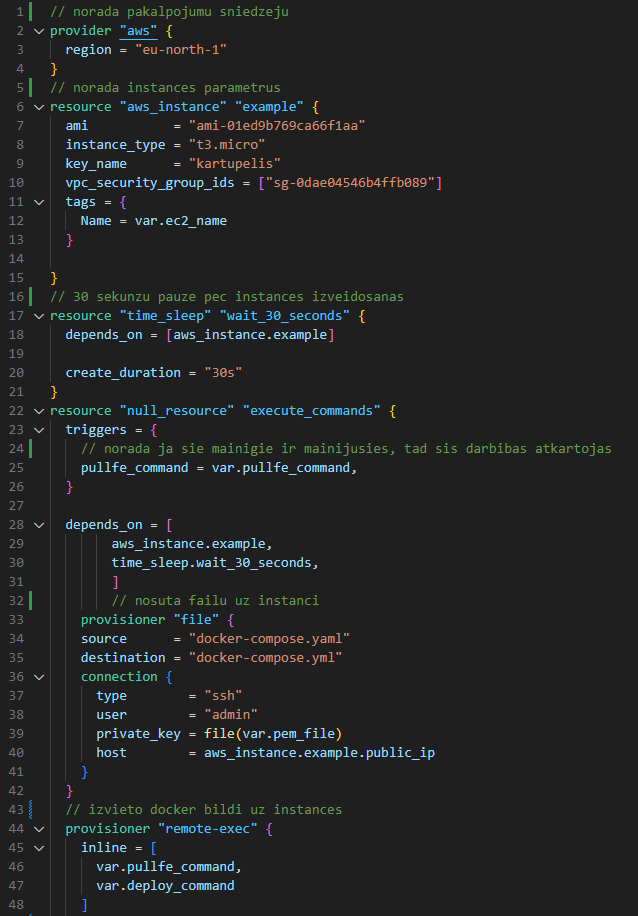
**4. pielikums**

****

**4. attēls**

**5. pielikums**

“main.tf” kods

****

**5. attēls**

**6. pielikums**

**A screen shot of a computer program

Description automatically generated with medium confidence**

**6. attēls**