Valsts reģionālas attīstības aģentūra

VRAA kubernetes infrastruktūras komponenšu

---

versija

|  |
| --- |
|  |

Rīgā 2021

Dokumenta identifikācija

|  |  |
| --- | --- |
| Dokumenta ID: | ----V- |
| Dokumenta nosaukums: | VRAA kubernetes infrastruktūras komponenšu.  . |
| Dokumenta kods: | --- |
| Versija: | Versija , Laidiens (saīsināti V ) |

Saskaņojumi

|  |  |
| --- | --- |
| Organizācija | Atbildīgā persona |
|  | A.Bondarčuks |
| SIA "ABC software" | S.Degtjars, dokumenta autors |
| SIA "ABC software" | J.Korņijenko, projekta vadītājs no Izpildītāja puses |

Izmaiņu vēsture

|  |  |  |
| --- | --- | --- |
| Versija | Datums | Apraksts |
| 1.00 | 28.04.2021. | Izveidota dokumenta sākotnējā versija. |
| 1.01 | 25.05.2021 | Veikti labojumi atbilstoši pasūtītāja komentāriem, pievienota nodaļa “Pirmkoda piegāde uz VRAA Git repozitoriju”. |
| 1.02 | 28.05.2021 | Veikti labojumi atbilstoši pasūtītāja komentāriem |
| 1.03 | 21.07.2021 | Veikti labojumi balstoties uz klientu jautājumiem |
| 1.04 | 30.11.2021 | Pievienota sadaļa 6.9 par Docker reģionāliem uzstādījumiem |
| 1.05 | 29.08.2022 | Veikti papildinājumi sadaļā 3.6.3, LAC pieteikuma #12731 ietvaros. |

**Satura rādītājs**

[Attēlu saraksts 6](#_Toc112676580)

[1. Ievads 7](#_Toc112676581)

[1.1. Dokumenta nolūks 7](#_Toc112676582)

[1.2. Termini un pieņemtie apzīmējumi 7](#_Toc112676583)

[1.3. Saistība ar citiem dokumentiem 8](#_Toc112676584)

[1.4. Dokumenta pārskats 8](#_Toc112676585)

[2. Priekšnosacījumi 9](#_Toc112676586)

[3. Nodevumu piegādēs un uzstādīšanas automatizācijas process 10](#_Toc112676587)

[3.1. Nodevumu piegādes un uzstādīšanas procesa vīzija 10](#_Toc112676588)

[3.2. Nodevumu automatizācijas process pie Pasūtītāja (Jenkins) 11](#_Toc112676589)

[3.3. Produktu un komponenšu sadalījums, to versijas pārvaldības principi 12](#_Toc112676590)

[3.3.1. Kubernetes zonas (namesapces) 12](#_Toc112676591)

[3.3.2. Produktu un komponenšu sadalījums 12](#_Toc112676592)

[3.3.3. Komponenšu versijas pārvaldības principi 14](#_Toc112676593)

[3.4. Kubernetes Pod objekta apraksts 15](#_Toc112676594)

[3.5. Nexus nodevumu artefaktu krātuve 15](#_Toc112676595)

[3.5.1. Docker artefaktu krātuve 16](#_Toc112676596)

[3.5.2. Helm artefaktu krātuve 19](#_Toc112676597)

[3.6. Nodevuma automatizācijas procesa inicializācija no Izpildītajā puses. 20](#_Toc112676598)

[3.6.1. Helm pakotnes nodošana Pasūtītājā krātuvē 21](#_Toc112676599)

[3.6.2. Docker konteinera nodošana Pasūtītājā krātuvē 21](#_Toc112676600)

[3.6.3. Jenkins uzstādīšanas pipeline izsaukšana Pasūtītājā pusē 21](#_Toc112676601)

[3.6.4. Pirmkoda piegāde uz VRAA Git repozitoriju 22](#_Toc112676602)

[4. Helm pakotnes izstrādes principi 24](#_Toc112676603)

[4.1. Helm chart struktūra un saturs 24](#_Toc112676604)

[4.1.1. Helm Chart saknes mape 24](#_Toc112676605)

[4.1.2. Helm Chart templates mape 26](#_Toc112676606)

[5. Vides atkarīga konfigurācija 33](#_Toc112676607)

[5.1. Globāla vides konfigurācija 33](#_Toc112676608)

[5.2. Komponentes vides konfigurācija 35](#_Toc112676609)

[5.3. Helm chart un konfigurācijas pārbaude 36](#_Toc112676610)

[6. Piegādājamās programmatūras realizācijas labas prakses un rekomendācijas 37](#_Toc112676611)

[6.1. Konteineru resursu konfigurācija un pārvaldība 37](#_Toc112676612)

[6.2. Labas prakses darbam ar ārējām datu krātuvēm 37](#_Toc112676613)

[6.3. CronJob realizācijas rekomendācijas 37](#_Toc112676614)

[6.4. API izsaukumu apstrādes specifika 38](#_Toc112676615)

[6.5. Kubernetes izmitināmā konteinera statusa pārbaudes (*probes*) 38](#_Toc112676616)

[6.6. Kubernetes izmitināmā konteinera veiksmīga apstādināšana (*graceful shutdown*) 39](#_Toc112676617)

[6.7. Piegādājamo konteineru izmitināšana pa vairākiem datu centriem 39](#_Toc112676618)

[6.8. Sistēmas žurnāla pierakstu veidošana konteinerī 39](#_Toc112676619)

[6.9. Docker konteinera reģionālie uzstādījumi 39](#_Toc112676620)

[7. Nodevuma gatavības *check-list* automātiskajai piegādei un izmitināšanai VRAA Kubernetes platformā 41](#_Toc112676621)

# Attēlu saraksts

[1.attēls. Nodevuma piegādes un uzstādīšanas procesa vīzija 10](#_Toc112676622)

[2.attēls. Nodevuma automatizācijas process pie Pasūtītāja (Jenkins) 11](#_Toc112676623)

[3.attēls. Kubernetes POD un Istio savstarpēja darbība 15](#_Toc112676624)

[4.attēls. Nexus Docker repozitorija struktūra 17](#_Toc112676625)

[5.attēls. Nexus Docker repozitorija struktūra – e-pakalpojuma piemērs 18](#_Toc112676626)

[6.attēls. Nexus Helm repozitorija struktūra 19](#_Toc112676627)

[7.attēls. Nexus Helm repozitorija struktūra – E-pakalpojuma piemērs 20](#_Toc112676628)

[8.attēls. Pirmkoda repozitorija tag un commit apraksti. 23](#_Toc112676629)

[9.attēls. Helm chart struktūra 24](#_Toc112676630)

# Ievads

## Dokumenta nolūks

Dokuments, kurā sniegta informācija par tehniskām prasībām pret komponentēm, kuras paredzēts nodot un izvietot VRAA Kubernetes klasterī, izmantojot automatizēto piegādes un izvietošanas procesu.

## Termini un pieņemtie apzīmējumi

Dokumentā izmantotie termini ir apkopoti 1.tabulā.

1.tabula

Termini

| Termins | Skaidrojums |
| --- | --- |
| Aģents | Lietojumprogrammas sastāvdaļa, kura novēro notikumus un pārsuta novērojumus/informāciju resursdatoram tālākai apstrādei. |
| Autentifikācija | Process, kurā identificē lietotāju datoru ierīci vai procesu, veicot sniegtās identificējošās informācijas validāciju. |
| Autorizācija | Process, kurā datorsistēma lietotājam nosaka noteiktas pilnvaras un resursus sistēmā. Autorizācija tiek veikta tikai autentificētiem lietotājiem. |
| Docker | *Docker* ir servisa platformu kopiena, kura pielieto operētājsistēmas līmeņa virtualizāciju, lai pasniegtu lietojumprogrammas pakotnēs – konteineros. Konteineri ir savstarpēji izolēti un savā starpā komunicē, pielietojot skaidri definētus kanālus. |
| ElasticSearch | *ElasticSearch* ir meklētājprogramma, kura nodrošina JSON dokumentu *full-text* meklēšanu. |
| e-pakalpojumu platforma | Digitālā platforma, kuras mērķis ir paaugstināt valsts institūciju, iedzīvotāju un komercuzņēmumu sadarbības efektivitāti. Veidojot jaunus e-pakalpojumus. Šajā dokumentā e-pakalpojumu platformas komponentes tiek izmantotas kā piemērs piegādes procesa organizēšanai. |
| Galvene | Informācijas struktūra, kura sniedz identificējošu informāciju par tiem datiem, ar kuriem galvene ir saistīta. |
| Kubernetes | *Kubernetes* ir konteineru orķestrācijas sistēma, kura automatizē lietojumprogrammu izmitināšanu mērogošanu un pārvaldīšanu. |
| LogStash | *LogStash* ir datu apstrādes konveijers, kas nodrošina datu apkopošanu no vairākiem avotiem, šo datu transformāciju un to nosūtīšanu uz norādīto galapunktu. |
| Metadati | Informācija par datiem un to struktūru. |
| Particionēšana | Datu sadalīšana atšķirīgās un neatkarīgās grupās. |
| Pod | *Pod* ir Kubernetes specifikācija, kas apraksta viena vai vairāku konteineru grupējumu, kuriem ir vienoti glabāšanas/tīkla resursi un specifikācija, kā darbināt minētos konteinerus. |
| RabbitMQ | *RabbitMQ* ir ziņojumu pārvaldīšanas starpniekprogrammatūra, kura pielieto rindas ziņojumu uzglabāšanai, kamēr ziņojumi gaida tālāku apstrādi. |
| Rinda | Datu struktūra objektu saraksta glabāšanai, kuri gaida datu apstrādes sistēmas pakalpojumus. |

## Saistība ar citiem dokumentiem

1. VALSTS INFORMĀCIJAS SISTĒMU SAVIETOTĀJA (VISS) UN VIENOTĀ VALSTS UN PAŠVALDĪBU PAKALPOJUMU PORTĀLA WWW.LATVIJA.LV PILNVEIDOŠANA UN UZTURĒŠANA. VISS sistēmas žurnāls. Koplietojuma bibliotēku apraksts (VRAA-13\_7\_17\_41-VISS\_2016-VISS\_ZUR-KBA).

## Dokumenta pārskats

Dokumentu veido septiņi nodalījumi:

* Pirmajā nodalījumā – **Ievads** – aprakstīta informācija par dokumenta kopējo struktūru, izstrādājamā programmatūras produkta darbības sfēru, nolūku, izmantotajiem terminiem un apzīmējumiem, kā arī saistība ar citiem dokumentiem.
* Otrajā nodalījumā – **Priekšnosacījumi** – aprakstīti resursi, ar kuriem būtu jāiepazīstas auditorijai, kurai tiek paredzēts šis dokuments.
* Trešajā nodalījumā – **Nodevumu piegādes un uzstādīšanas automatizācijas process** – aprakstīta informācija par piegādājamo artefaktu nosaukumu konvenciju, to versionēšanu, kā arī to nodošanu procesu.
* Ceturtajā nodalījumā – **Helm pakotnes izstrādes principi** – aprakstīta informācija par *Helm Chart* uzbūves principiem un pielietojamo tehnoloģiju.
* Piektajā nodalījumā – **Vides atkarīga konfigurācija** – aprakstīta informācija par vides konfigurāciju un to pārvaldību.
* Sestajā nodalījumā – **Piegādājamās programmatūras realizācijas labas prakses un rekomendācijas** – dotas labas prakses rekomendācijas par dažādu komponenšu uzbūves un konfigurācijas principiem.
* Septītajā nodalījumā – **Nodevuma gatavības check-list automātiskajai piegādei un izmitināšanai VRAA K8s platformā** – apkopotas darbības (no šī dokumenta), kas ir jāveic, lai nodrošinātu pilnvērtīgo produkta nodošanu.

# Priekšnosacījumi

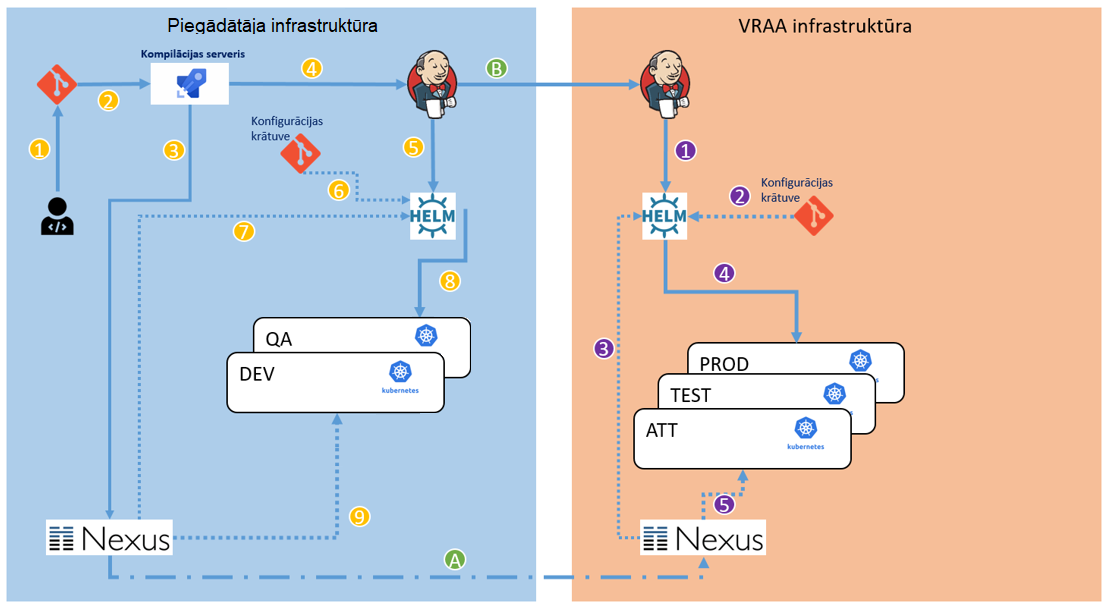
Lai pilnā apjomā izprastu dokumentā sniegto informāciju, lasītājam ir nepieciešamas dažādas priekšzināšanas:

* *Git* versiju kontroles sistēma – nepieciešama izpratne par *Git* darbības principiem, *Git* zarošanu, darbībām ar lokāliem un attālinātām krātuvēm (<https://git-scm.com>);
* *Jenkins* automatizācijas serveris – nepieciešama izpratne par darbības un izmantošanas labām praksēm (<https://www.jenkins.io>);
* *DevOps* – nepieciešama izpratne par *DevOps* metodoloģiju un kultūru, tajā pielietotiem rīkiem un izmantojamiem procesiem (<https://docs.microsoft.com/en-us/azure/devops/learn/what-is-devops>, <https://www.altexsoft.com/blog/engineering/devops-principles-practices-and-devops-engineer-role>, <https://www.youtube.com/results?search_query=devops>);
* *Docker* konteinerizācijas platforma – nepieciešama izpratne par konteinerizācijas darbības principiem (<https://docs.docker.com>);
* *Kubernetes* orķestrācijas platforma – nepieciešama izpratne par *Kubernetes* darbības principiem, to izmantojamiem objektiem (šobrīd VRAA izmanto Kubernetes v1.18.5, bet plāno veikt atjauninājumu uz jaunākām versijām; <https://v1-18.docs.kubernetes.io/docs/concepts>);
* *Helm* – nepieciešama izpratne par produktu izvietošanas rīku *Kubernetes* platformai, kas ir bāzēta uz veidņu sistēmas (šobrīd VRAA izmanto Helm v3; <https://v3.helm.sh/docs/>);
* *Istio* – nepieciešama izpratne par *Service Mesh* arhitektūras realizāciju, kas kalpo kā apakšslānis starp-servisu komunikācijai (<https://istio.io/v1.6/docs>).

# Nodevumu piegādēs un uzstādīšanas automatizācijas process

Šajā sadaļā tiek dots priekšskats par izmantoto automatizācijas procesu, kā arī prasības, kas jāievēro izstrādātājiem, gatavojot produkta nodošanu uz VRAA Kubernetes platformu.

## Nodevumu piegādes un uzstādīšanas procesa vīzija



1.attēls. Nodevuma piegādes un uzstādīšanas procesa vīzija

1.attēlā ir attēlo kopējā vizualizācija piegādes procesam. Process sastāv no vairākiem posmiem.

Produktu virzība izstrādātāja vidē (dzeltenā krāsā):

1. Izstrādātājs ielādē pirmkodu *Git* repozitorijā;
2. Automātiski tiek palaists kompilācijas un testēšanas process;
3. Ja kompilācija un testi notiek veiksmīgi, tiek izveidoti *Docker* *Image* un *Helm* *Chart,* un ielīkti *Nexus* repozitorijā
4. Kompilācijas serveris izsauc *Jenkins* servera atbilstošo ieejas punktu, kas iniciē uzstādīšanas procesu;
5. *Jenkins* iniciē *Python* skriptu izpildi, kas palaiž *Helm* instalāciju;
6. *Helm* instalācija pievelk attiecīgās konfigurācijas datnes no Git repozitorija, kas atbilst videi, kur notiek izmitināšana;
7. *Helm* instalācija pievelk attiecīgo *Helm* *Chart* no *Nexus* repozitorija;
8. *Helm* nokompilē *Kubernetes* manifestus un ielādē tos *Kubernetes* klasterī;
9. *Kubernetes* klasteris pēc ielādētiem manifestiem izveido resursus, pievelk attiecīgo *Docker* *Image* no *Nexus* repozitorija un palaiž konteineru no tā;

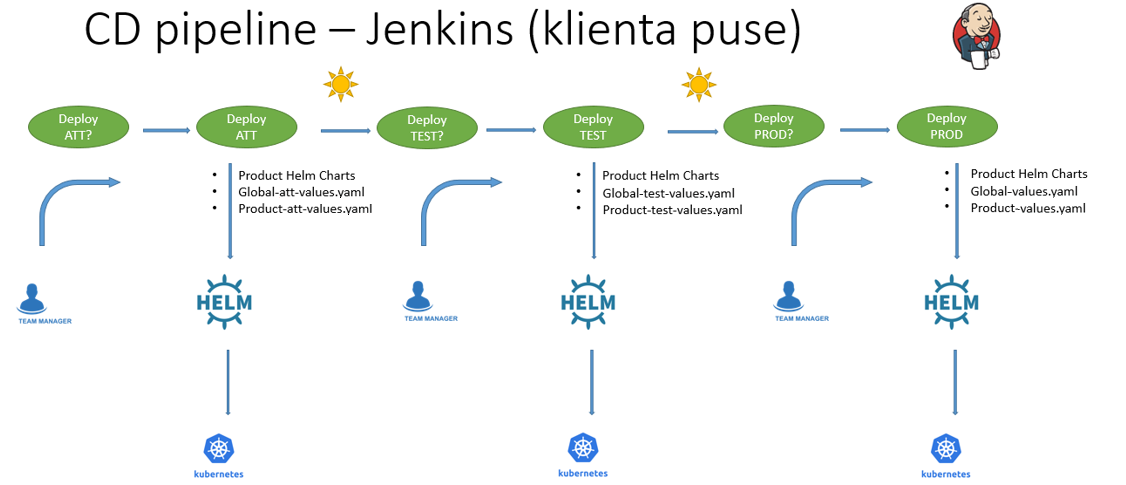
Pēc tam, kad produkts izgājis cauri visām izstrādātāja vidēm, un produkta menedžeris pieņem lēmumu veikt nodošanu klientam, tiek iniciēta *Jenkins* darbība, kas (zaļā krasā):

1. Ar skriptu palīdzību izlādē *Docker* *Image* un *Helm* *Chart* no izstrādātāja *Nexus* repozitorija, un ielādē tos VRAA *Nexus* repozitorijā;
2. Pēc veiksmīgas ielādēšanas izsauc klienta *Jenkins* serveri, atbilstošo ieejas punktam, kas iniciē uzstādīšanas procesu;
3. Piegādā administratora instrukcijas, konfigurācijas datnes piemērus;
4. Piegādā izejas kodus uz pasūtītāja GIT repozitoriju.

Pēc nodošanas klientam, klients no savas puses iniciē uzstādīšanas procesu savā *Jenkins* serverī (zilā krasā):

1. *Jenkins* iniciē *Python* skriptu izpildi, kas palaiž *Helm* instalāciju;
2. *Helm* instalācija pievelk attiecīgās konfigurācijas datnes no *Git* repozitorija, kas atbilst videi, kur notiek izmitināšana;
3. *Helm* instalācija pievelk attiecīgo *Helm* *Chart* no Nexus repozitorija;
4. *Helm* nokompilē *Kubernetes* manifestus un ielādē tos *Kubernetes* klasterī;
5. *Kubernetes* klasteris pēc ielādētiem manifestiem izveido resursus, pievelk attiecīgo *Docker* *Image* no *Nexus* repozitorija un palaiž konteineru no tā.

## Nodevumu automatizācijas process pie Pasūtītāja (Jenkins)



2.attēls. Nodevuma automatizācijas process pie Pasūtītāja (Jenkins)

2.attēlā ir attēlota piegāžu uzstādīšanas diagramma klienta pusē.

Kad klienta *Jenkins* saņem informāciju par jauna produkta piegādi, tas automātiski izveido piegādes *pipeline* ar attiecīgā produkta versiju, un automātiski to palaiž.

*Pipeline,* uzreiz pēc palaišanas apstājas un sagaida komandu no administratora, lai uzsāktu uzstādīšanu TEST vidē. Kad komanda ir saņemta, *Jenkins* izsauc CD skriptu, kas savukārt veic sekojošas darbības:

1. Veic nepieciešamās pārbaudes par vajadzīgo artefaktu pieejamību *Docker*, *Helm* un *Git* krātuvēs;
2. Ja pārbaudes procesā netika novēroti trūkumi, tas lejupielādē komponentes versijai atbilstošo *Helm* *Chart* no artefaktu repozitorija;
3. Lejupielādē komponentes vides konfigurācijas datni un globālo vides konfigurācijas datni no *Git* repozitorija;
4. Veic *Helm* komandas izpildi, lai nokompilētu komponentes *Kubernetes* resursus un to nodošanu *Kubernetes* klasterī;
5. *Kubernetes* klasteris pēc saņemtajiem resursiem lejupielāde atbilstošo *Docker* *image* no artefaktu repozitorija, un balstoties uz resursa datnes norādītajiem parametriem izveido un palaiž produkta konteineri;
6. CD skripts veic regulāro pārbaudi par pacelta konteinera statusu noradītajā laika intervālā:
   1. Ja konteiners nav pacelts pēc noradītā laika intervāla, CD skripts veic atriti (rollback) uz iepriekšējo versiju.
   2. Ja uzstādīšana ir veiksmīga, process apturas un sagaida komandu veikt izmitināšanu nākamajā vidē.

## Produktu un komponenšu sadalījums, to versijas pārvaldības principi

### Kubernetes zonas (namesapces)

VRAA Kubernetes platformā zonas tiek veidotas pēc šādiem principiem:

* xxx-public – publisko tīmekļa lietotņu un to fasādes izmitināšanai. Šajā zonā izmitina arī publiskos koplietojamos resursus – CDN/Assets;
* xxx-system – publiski pieejamo API izmitināšanai;
* xxx-internal – iekšējo API izmitināšanai, pieejami tikai no citām zonām, tipiski system zonas.

Šobrīd izveidotās zonas:

* epak-public – e-pakalpojumu grafiskās saskarnes un to apkalpojošais slānis;
* epak-system - Lvp sistēmā izmantojamie publiskie API;
* epak-internal - e-pakalpojumu platformas iekšējie API, publiski nav pieejami – REST servisi, kas nodrošina pieeju pie VISS infrastruktūras servisu nodrošinātās funkcionalitātes;
* viss-public - Viss sistēmas grafiskās saskarnes un to apkalpojošais slānis;
* viss-system - Viss sistēmā izmantojamie publiskie API;
* viss-internal- Viss sistēmā izmantojamie iekšējie API, publiski nav pieejami;

### Produktu un komponenšu sadalījums

Visa programmatūra, kas tiek piegādāta uz VRAA Kubernetes platformu, tiek klasificēta pēc produkta un to komponentēm. Produktu un komponenšu nosaukumi tiek pakļauti stingrai hierarhijai, kas jāievēro, izstrādājot attiecīgo kodu.

* Hierarhijas pirmais līmenis nosaka piederību konkrētai platformai/sistēmai – piemēram – “Lvp” vai “Viss”;
* Hierarhijas otrais vai trešais līmenis reprezentē konkrētu produktu zem konkrētās platformas/sistēmas, kas aprobežo konkrētu biznesu vai eko-sistēmu; (piemēram: Lvp.Portal, Lvp.EservicePlatform.Backend, Viss.ApiManagement);
* Hierarhijās trešajā vai ceturtajā līmenī atrodas konkrētajam produktam piederošie servisi jeb komponentes (Lvp.Portal.IdentityServer, Viss.ApiManagement.TransactionApi, Lvp.EservicePlatform.Backend.ContextApi).

Katra komponente tiek identificēta pēc tās pilna nosaukuma (piemēri: Lvp.EservicePlatform.Backend.EdkApi), un tiek versionēta pēc zemāk dotiem norādījumiem.

Komponenšu trešā līmeņa nosaukumos izmantojamas iepriekš definētas vērtības, kas nosaka komponentes būtību.

2.tabula

Komponentes pieņemtie nosaukumi un to raksturojums

| Komponentes 3. līmeņa nosaukums | Komponentes izmantošanas veids |
| --- | --- |
| **webapp** | Konkrēta biznesa produkta tīmekļa lietotne, piemēram, Viss.ResourceCatalogue.WebApp vai e-pakalpojuma scenārijā Lvp.EservicePlatform.Eservices.ep213.webapp. |
| **Api/bff** | Konkrētas tīmekļa lietotnes fasāde (BFF). Svarīgi atzīmēt, ka fasāde pilda *agregatora* lomu, tāpēc tajā ir iekapsulētas dažāda biznesa funkcionalitātes apstrādes loģika. Tāpēc nosaukums “Api” nesatur nekādu piederību biznesa loģikas (BL) semantikai.  Publisks koplietojams VRAA infrastruktūras API, piemēram Viss.Profile.Api vai e-pakalpojuma scenārijā Lvp.EservicePlatform.Eservices.ep213.bff. |
| **<BL semantika>Api** | Biznesa *RESTful API*, kas tiek izsaukts vienas zonas ietvaros vai starp zonām. |
| **<BL semantika>Worker** | Process, kas darbojas autonomi bez API eksponēšanas, bet tā darbības sfēra ir ierobežota ar VRAA infrastruktūras resursiem. |
| **External<BL semantika>Worker** | Procesa, kas darbojas autonomi bez API eksponēšanas, bet tā darbības sfēra paredz integrāciju ar ārējām sistēmām, kas atrodas ārpus VRAA infrastruktūras, piemēram integrācijas ar PMLP, Uzņēmumu reģistru, Rīgas domi u.c. |

Šajā tabulā termins <BL Semantika> tiek uztverts kā servisa pamatdarbības pazīme. Visbiežāk tiek izmantoti šādi apzīmējumi:

* *LogicApi* – mikroserviss, kas ir atbildīgs par tiešo darbību ar *persistent* veida krātuvi (MongoDB, PostgreSQL un tml.) un atbilstošo izmaiņu notikumu izplatīšanu izmantojot brokeri, ja ir vajadzīgs.
* *IndexWorker* – mikroserviss-indeksators, kas nolasa no rindas (RabbitMQ) ziņojumus par izmaiņām *persistent* veida krātuvē un atbilstoši veic izmaiņas *ElasticSearch* meklēšanas indeksā;
* *SearchApi* – mikroserviss-meklētājs, kas pēc REST pieprasījuma veic resursu meklēšanu *ElasticSearch* indeksā. Pieļauts arī apvienot *SearchApi* un *IndexWorker* darbības viena servisa *IndexApi* gadījumā, ja apstrādāto dokumentu apjoms ir salīdzinoši neliels un/vai biznesa funkcionalitātes ziņā ir izdevīgi, ka mikroserviss pilda indeksatora un meklētāja lomas vienlaicīgi.
* *Lvp.EservicePlatform.Eservices.ep213.webapp* – e-pakalpojuma ar numuru 213 tīmekļa lietotne.

Ja ir paredzēts veidot atsevišķus servisus, kas strādā ar dažādām krātuvēm un veic dažādas biznesa loģikas darbības, ir pieņemams veidot divus API servisus, bet ir jāņem vērā, ka šie servisi nedrīkst strādāt ar vienu un to pašu loģisko krātuvi – tiem jāoperē vai nu ar atsevišķām *MongoDB* kolekcijām, vai nu SQL tabulām, vai nu jābūt dalāmiem pēc darbības principiem – *Read/Write*.

Servisiem, kas darbojās ar ārējām sistēmām, <BL Semantika> visbiežāk apzīmē ārējo sistēmu, ar kuru tiek veikta integrācija, piemērām – *PmlpSearchApi*, *HugoTranslationWorker.*

Ir gadījumi, kad internal zonas servisu paredzēts izsaukt no ārējās sistēmas, kas integrējas ar VRAA platformas resursiem, šajā gadījumā <BL Semantika> izpratne nemainās[[1]](#footnote-2).

### Komponenšu versijas pārvaldības principi

Katra komponente tiek pakļauta savai versiju vēsturei.

Ir ļoti būtiski ievērot versionēšanas principus, lai sakārtotu piegādājamās programmatūras inkrementus, atbilstoši notācijai veiktu funkcionalitātes izmaiņu uzskaiti un nodefinētu vienotus Deployment procesus, neatkarīgi no realizējamā biznesa.

Tāpēc ir jāņem vērā, ka VRAA automatizētais Deployment process nepieļauj izvietot Kubernetes klasterī versiju, kas ir zemāka par jau eksistējošo. Turklāt, ja tiek pārrakstīta esoša versija ar citu Docker vai Helm datni, var rasties situācija, ka Kubernetes klasterī atrodas dažādās komponentes ar vienādu versiju, vai Deployment process nevar atjaunot vai atiestatīt (rollback) komponenti.

Tiek izmantota t. s. otrā paaudzes semantiskā versionēšana (<https://semver.org>), kas paredz, ka versionējamajai vienībai (produktam, komponentei) ir trīs daļas: MAJOR.MINOR.PATCH, un ka:

* MAJOR daļa tiek paaugstināta, kad tiek veiktas pietiekami apjomīgas izmaiņas vai nesaderīgas API izmaiņas;
* MINOR daļa tiek paaugstināta, kad tiek pievienota jauna funkcionalitāte, netraucējot atpakaļsaderību (*backward compatibility*);
* PATCH daļa tiek paaugstināta, kad tiek veikti atpakaļsaderīgie kļūdu labojumi.

Versiju piešķiršana un izmaiņas notiek balstoties uz šādiem principiem:

1. MAJOR, MINOR un PATCH versiju paaugstināšana notiek secīgi, ar soli 1.
2. Jaunās komponentes izstrāde tiek iniciētā ar versiju v0.1.0.
3. Aizejot uz produkciju, jaunai komponentei ir jau jābūt v1.0.0.
4. MAJOR un MINOR daļas tiek mainītas tikai gadījumos, kad ir noformēts atbilstošs izmaiņu pieprasījums vai jauns projekts, kurš iniciē funkcionālās izmaiņas.
5. Gadījumā, ja notiek aktīvs *BugFix* komponentēm vai, piemēram, funkcionalitātes pievienošana, kura jau pēc līguma saistībām it kā jau jābūt esošajā versijā, bet kādu iemeslu dēļ, tā nebija piegādāta, tik un tā ir jāuzsver kā PATCH daļas izmaiņa.
6. Lai nepazaudētu iespēju iniciēt vairākas reizes *deployment* procedūru izstrādes vidē, ir jāizmanto tā saucamo komponentes klasificējamo gatavības stāvokli – *alpha*, *beta* un *rc* (release-candidate). Protams, tie stāvokļi varētu būt arī citi pēc *semver*, bet vienkāršības pēc rekomendējam izmantot pasaulē visbiežāk lietotus postfiksus. Un, tad lai atšķirtu vienu versiju no iepriekšējās, ir jāizmanto numerācija pēc produkta stāvokļa apzīmējuma, piemēram:

* 0.1.0-alpha.0  0.1.0-alpha.1  0.1.0-alpha.2  0.1.0  0.1.1-alpha.0  0.1.1-alpha.1  0.1.1-beta.0  0.1.1
* 0.3.0-alpha.0  0.3.0-beta.0  0.3.0-beta.1  0.3.0-rc.0  0.3.0-rc.1  0.3.0-rc.2  0.3.0

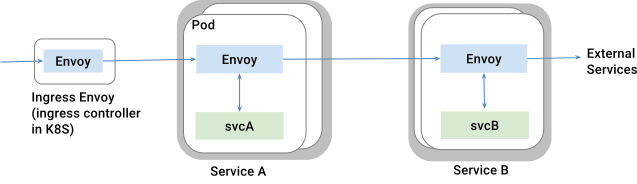
1. Komponentes gatavības stāvokļa interpretācija varētu mainīties atkarībā no projekta un realizējamās komponentes, bet, ja pieturētos klasikai: *alpha* reprezentē zaļo versiju bez īpašiem testiem, *beta* – jau stabilāko versiju ar testiem, *rc* – gandrīz pabeigta versija, kura varētu būt kā kandidāts nodošanai.
2. Minēto gatavības stāvokļu piešķiršanas kārtība pilnībā ir saderīga *ar .NET Core, Node.JS* un *Java* pasauli un atbilst pasaules atzītai semantiskai versionēšanai.
3. VRAA K8s platformas komponenšu versijas šablons: <MAJOR versija>.<MINOR versija>.<PATCH versija>-<alpha | beta | rc>.<kārtas laidena numurs>.
4. Produkcijas vidē pieļaujams izvietot tikai versijas bez *postfixa* – t.i. X.Y.Z; izņēmumā gadījumos tiek atļauts aizvietot rc.NNN stāvokļa versiju, ja nav iespējams rast citas iespējas.

## Kubernetes Pod objekta apraksts

*Kubernetes* infrastruktūrā POD objekts raksturo minimālo pārvaldāmo vērtību izmitināšanas procesu. No piegādes un izvietošanas skatupunkta tā ir viena komponente. POD satur vienu vai vairākus *Docker* konteinerus, kas ir apvienoti ar vienu *Docker* vārdu telpu (*namespace*). Iekš *Kubernetes* klastera POD’am tiek automātiski piešķirta iekšējā IP adrese, bet konteineri, kas pieder vienam POD’am var savā starpā komunicēt, izmantojot IP adrese – 127.0.0.1.

Gadījumā, kad *deployment* prasības paredz izvietot vairākas servisa instances, *Kubernetes* klasteris izveido vairākas neatkarīgas POD instances, katra no tiem dzīvo savu dzīvi un savā starpā nav saistītas. Bet, lai maršrutētu uz POD ienākošo trafiku starp vairākām instancēm, *Kubernetes* infrastruktūrā tiek veidots ***Service*** virtuālais objekts, kas veic slodzes dalīšanas funkciju.

VRAA K8s platformā tiek pielietots *Istio Service-Mesh* spraudnis, kas pie POD izvietošanas automātiski pievieno papildu konteineri *Istio-Proxy* (realizācija ir bāzēta uz *Envoy* programmatūras) katram POD objektam. Šis konteiners caurspīdīgi kalpo kā starpslānis, nodrošinot drošu komunikāciju starp servisiem un ārējām sistēmām.



3.attēls. Kubernetes POD un Istio savstarpēja darbība

*Kubernetes* nosaka politiku uzsaukumiem starp servisiem iekš Kubernetes klastera un uz ārējām sistēmām pēc principa – **Deny All, Allow Some**.

Pie pacelšanas *Istio-Proxy* konteiners nolasa no *Istio* pārvaldības slāņa datus par galapunktiem, uz kuriem dotajam servisam ir atļauts veikt izsaukumus, un, tādā veidā nodrošinot, papildu drošību starp servisu komunikāciju. Lai izveidotu šo atļauto galapunktu sarakstu, katram servisam tiek veidots *Sidecar* virtuālais objekts un tiek iekļauts *Helm* instalācijas pakotnē.

Papildu lasīšanai ir pieejami šādi tīmekļa resursi, kas domāti iesācēju apmācībai par Kubernetes darbības principiem:

* The Illustrated Children’s Guide to Kubernetes: <https://www.cncf.io/the-childrens-illustrated-guide-to-kubernetes>
* Phippy Goes to the Zoo (A Kubernetes Story): <https://www.cncf.io/phippy-goes-to-the-zoo-book>

## Nexus nodevumu artefaktu krātuve

Katram izstrādātājam jāielādē komponenšu *Docker* un *Helm* artefakti centralizētajā VRAA *Nexus* repozitorijā. Lai veiktu automātisko izvietošanu, tiem jāpakļaujas noteiktai vārdu notācijai.

Viena nodevuma ietvaros nododamo artefaktu vārdiem un versijām jāsakrīt (ievērojot konkrētā artefakta vārda semantiku). Nexus repozitoriji veidoti hierarhijas veidā, lai atbilstu esošajai komponenšu nosaukumu piešķiršanas kārtībai.

Šī vārdu notācija izriet no konkrētā repozitorija un automatizēta procesa darbības principiem.

### Docker artefaktu krātuve

*Docker image* nosaukumam ir jābūt apakšējā reģistrā un jāsatur sistēmas, produkta un komponentes nosaukumus, kas ir atdalīti ar “/” simbolu un komponentes versiju.

Piemērs: lvp/eserviceplatform/backend/navigationapi:1.0.3

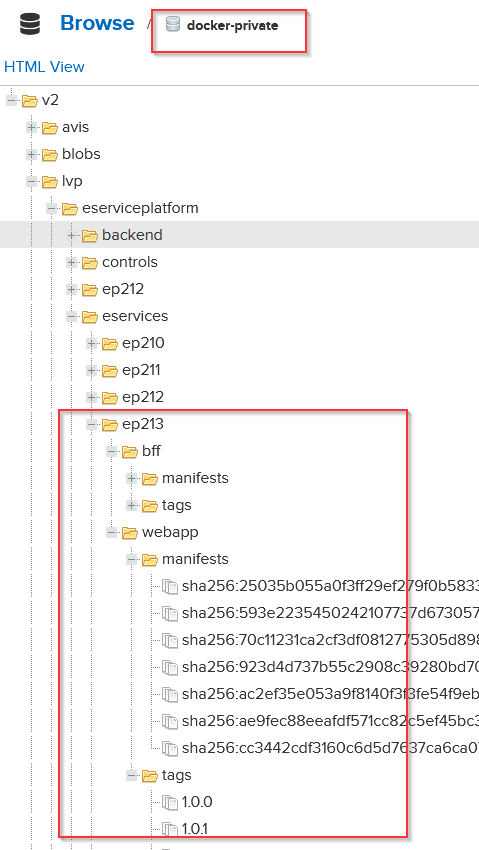
Šādā veidā *Nexus* repozitorijā tiek veidota pareiza hierarhija.



4.attēls. Nexus Docker repozitorija struktūra

E-pakalpojuma piemērs kas satur divus docekrus: lvp/eserviceplatform/eservices/ep213/webapp:1.5.0 un lvp/eserviceplatform/eservices/ep213/bff:1.5.0

Šādā veidā *Nexus* repozitorijā tiek veidota pareiza hierarhija.



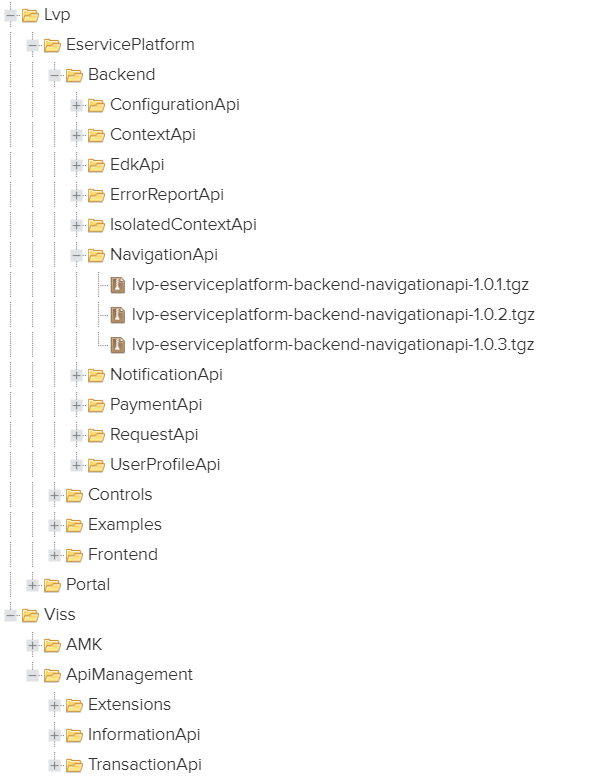
5.attēls. Nexus Docker repozitorija struktūra – e-pakalpojuma piemērs

### Helm artefaktu krātuve

*Helm* pakotnes (skat. 4. nodaļu) nosaukums tiek veidots automātiski, *Helm* pakotnes izveidošanas brīdī, un satur sistēmas, produkta, komponentes un to versiju apakšējā reģistrā, kas ir atdalīti “-” simbolu.

Piemērs: lvp-eserviceplatform-backend-navigationapi-1.0.3.tgz

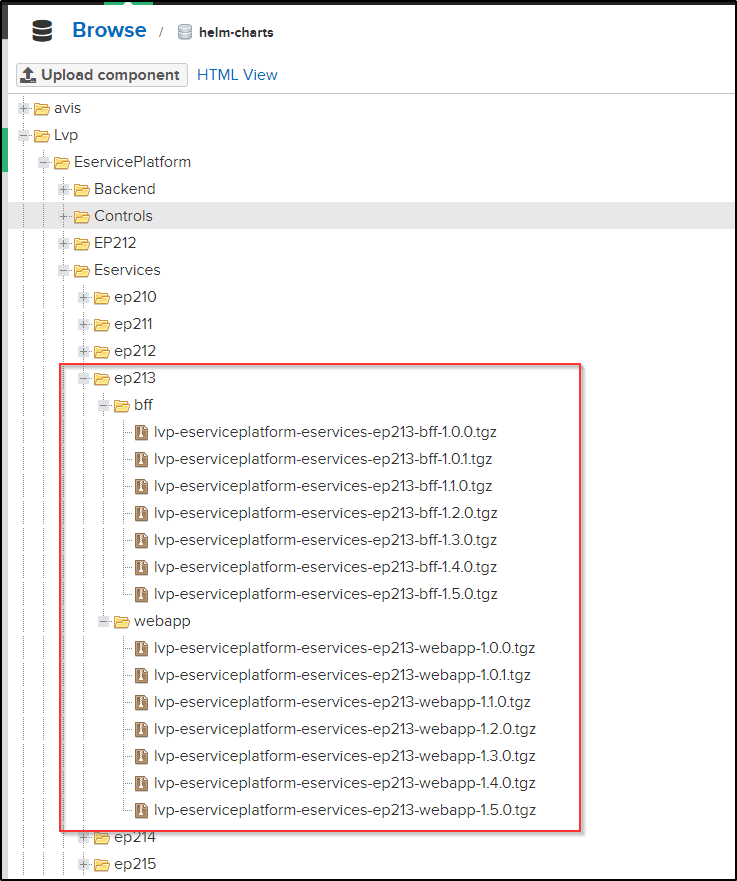
Savukārt *Helm* pakotne tiek ievietota *Nexus* repozitorija atbilstošā hierarhijā, kurai mapes nosaukumi tiek piešķirti, izmantojot *PascalCase* notāciju, piemēram: Lvp/EservicePlatform/Backend/NavigationApi



6.attēls. Nexus Helm repozitorija struktūra

E-pakalpojuma piemērs kas satur divus helm skriptus: lvp-eserviceplatform-eservices-ep213-bff-1.5.0.tgz un lvp-eserviceplatform-eservices-ep213-webapp-1.5.0.tgz

Savukārt *Helm* pakotne tiek ievietota *Nexus* repozitorija atbilstošā hierarhijā, kurai mapes nosaukumi tiek piešķirti, izmantojot *PascalCase* notāciju, piemēram: Lvp/EservicePlatform/Eservices/ep213/bff



7.attēls. Nexus Helm repozitorija struktūra – E-pakalpojuma piemērs

## Nodevuma automatizācijas procesa inicializācija no Izpildītajā puses.

Zemāk ir doti piemēri, kādā veidā ir jāorganizē komponenšu nodevumu piegādes uz VRAA *Nexus* repozitoriju.

### Helm pakotnes nodošana Pasūtītājā krātuvē

Lai nodotu *Helm* pakotni uz klienta repozitoriju, ir jāveic izsaukums caur HTTP POST.

Piemērs ar *Curl*:

curl --location --request POST 'https://nexus.vraa.gov.lv/service/rest/v1/components?repository=helm-charts' \

--header 'Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=' \

--form 'raw.directory=Lvp/EservicePlatform/Backend/NavigationApi' \

--form 'raw.asset1.filename=lvp-eserviceplatform-backend-navigationapi-1.0.4.tgz' \

--form 'raw.asset1=@/lvp-eserviceplatform-backend-navigationapi-1.0.4.tgz'

Pieprasījumā jāpadod sekojoši parametri:

* *Authorization* galvene – lietotāju akreditācijas dati *Basic* formātā (lietotājs:parole kodēts base 64 formātā), lietotājam jābūt tiesībām attiecīgajā hierarhijā izvietot helm pakotnes;
* *raw.directory* – mapes pilns nosaukums;
* *raw.asset1.filename* – *Helm* pakotnes pilns nosaukums;
* *raw.asset1* – *Helm* pakotnes datnes saturs.

Detalizētākai informācijai skatīt Nexus components-api dokumentāciju - <https://help.sonatype.com/repomanager3/rest-and-integration-api/components-api>

### Docker konteinera nodošana Pasūtītājā krātuvē

Lai nodotu *Docker* *image* uz VRAA *Docker* repozitoriju, nepieciešams izveidot sesiju un augšupielādēt (*push*) *docker image* uz klienta repozitoriju.

docker login -u <username> -p <password> nexusrep.vraa.gov.lv

docker push nexusrep.vraa.gov.lv /lvp/eserviceplatform/backend/navigationapi:1.0.4

### Jenkins uzstādīšanas pipeline izsaukšana Pasūtītājā pusē

Pēc tam, kad *Doker* un *Helm* artefakti veiksmīgi nodoti uz *Nexus* repozitoriju, nepieciešams veikt izsaukumu uz *Jenkins* *pipeline*, kas veiks komponentes izmitināšanai vajadzīgos sagatavošanas darbus, un nodos tālākai izpildei VRAA administratoriem. Lai izsauktu *Jenkins* programmatūru, ir jāveic HTTP POST izsaukumu uz norādīto adresi, adrese satur platformas/sistēmas identifikatoru, piemēram Lvp vai Viss atkarībā no komponentes kura tiek piegādāta, izsaukumā padodot atbilstošus parametrus:

Piemērs komponentei Lvp.EservicePlatform.Backend.NavigationApi, kuras nosaukums satur 4 līmeņus ar C*u*rl:

curl --location --request POST 'https://jenkins.vraa.gov.lv/job/Lvp/job/\_System/job/InitComponentDeployment/buildWithParameters' \

--header 'Authorization: Basic dXNlcm5hbWU6YXBpdG9rZW4=' \

--form 'product=Lvp.EservicePlatform.Backend ' \

--form 'component= lvp-eserviceplatform-backend-navigationapi \

--form 'componentShort=NavigationApi' \

--form 'version=1.0.4' \

--form 'timeout=100' \

--form 'namespace=epak-internal' \

--form 'deploymentType=api'

Piemērs komponentei Lvp.Portal.IdentityServer, kuras nosaukums satur 3 līmeņus ar C*u*rl:

curl --location --request POST 'https://jenkins.vraa.gov.lv/job/Lvp/job/\_System/job/InitComponentDeployment/buildWithParameters' \

--header 'Authorization: Basic dXNlcm5hbWU6YXBpdG9rZW4=' \

--form 'product=Lvp.Portal ' \

--form 'component= lvp-portal-identityserver \

--form 'componentShort=IdentityServer \

--form 'version=1.0.1' \

--form 'timeout=100' \

--form 'namespace=epak-public' \

--form 'deploymentType=api'

Pieprasījumā jāpadod sekojoši parametri:

* *Authorization* galvene – *Basic formatā header* ar *loginu* un API *tokenu* (lietotājs:parole vai lietotājs:tokens kodēts base 64 formātā. detalizētāku informāciju skatīt <https://www.jenkins.io/doc/book/using/remote-access-api/>);
* *product* – *PacalCase* produkta nosaukums (piemēram: ja komponentes nosaukums sastāv no 3 līmeņiem satur 1. un 2. līmeni, bet ja komponentes nosaukums satur 4 līmeņus, tad satur 1., 2. un 3. līmeni., utt.);
* *componentShort* – *PascalCase* komponentes nosaukums (piemēram: ja komponentes nosaukums sastāv no 3 līmeņiem, tad tikai 3. līmenis; vai tikai 4.līmenis, ja komponentes nosaukums sastāv no 4 līmeņiem, utt.);
* *component* – *lowercase* pilns komponentes nosaukums, atdalot nosaukumus ar domu zīmi
* *version* – komponentes versija;
* *namespace* – *Kubernetes* *namespace*, *ur* paredzēts izvietot doto komponente, jāsakrīt ar vērtību, kas ir norādīta *Helm* pakotnē;
* *timeout* – laika intervāls sekundēs, kurā paredzēts, ka *deployment* tiek pacelts darbojošā stāvoklī (atbild uz *readiness* signālu);
* *deploymentType* – nodevuma tips, var būt viens no:
* *api* – standarta serviss, kura *Helm* pakotnē satur *Deployment* manifestu;
* *job* – serviss, kas nesatur *Deployment* manifestu, bet satur tikai un vienīgi *CronJob* manifestu.

### Pirmkoda piegāde uz VRAA Git repozitoriju

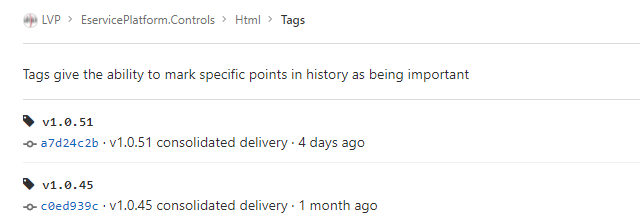
Katrai komponentei VRAA izveido pirmkoda glabāšanas GIT repozitoriju un nodrošina izstrādātājam pieeju pie tā.

Repozitorija mapju hierarhija tiek veidota no komponentes pilnā nosaukuma, sadalot to 3 līmeņos:

1. Sistēmas nosaukums ar lieliem burtiem, piemēram, “LVP” vai “VISS”;
2. Moduli identificējošā informācija izmantojot *PascalCase* notāciju, piemēram, “EservicePlatform.Eservices”. Satur visas nosaukuma daļas kas ir starp sistēmas un komponentes nosaukumiem;
3. Komponentes nosaukums izmantojot *PascalCase* notāciju, piemēram “Html”;

Izstrādātājs piegādājot jaunu versiju kopā ar to šajā repozitorijā ievieto arī konsolidētu pirmkoda piegādi un uzliek tai atbilstošu tag.

Veicot *commit* aprakstā iekļauj komponentes versiju, apraksta piemērs: “v1.0.45 consolidated delivery”. Un šim commit izveido arī atbilstošu tag kura nosaukumā iekļauj komponentes versiju, piemēram, “v1.0.45”.



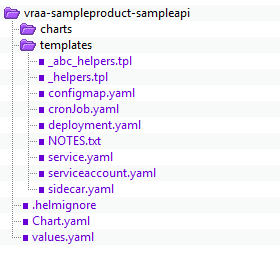
8.attēls. Pirmkoda repozitorija tag un commit apraksti.

# Helm pakotnes izstrādes principi

*Helm* pakotne (*chart*) tiek izstrādāta neatkarīgi katrai nododamai komponentei un tiek veidota pēc Helm dokumentācijas. VRAA tiek izmantota *Helm* versija v3, bet helm pakotne var tiks veidota izmantojot arī v2 sintaksi. Pielikumā pie šī dokumenta tiek sniegts arhīvs ar *Helm* *chart* piemēru, kuras tiek aprakstītas nākamajās sadaļās.

## Helm chart struktūra un saturs

*Helm* pakotne ir YAML formāta datņu kopums, kas tiek veidots atbilstoši *Helm* dokumentācijas noteiktajai hierarhiai. Sākotnējās mapes, kur tiek veidots *Helm chart* repozitorijs, nosaukumam jāsakrīt ar komponentes pilno nosaukuma *lowercase* formātu, kur nosaukuma daļas ir atdalītas ar domu zīmi.



9.attēls. Helm chart struktūra

(!!!) – piemēru rindas, kas ir apzīmēti ar šādu simbolu, vienmēr ir jānomaina atbilstoši izstrādājamai komponentei

### Helm Chart saknes mape

Saknes mape satur divas svarīgas datnes *– Chart.yaml* un *values.yaml*, kas definē pakotnes saturu un noklusētās vērtības.

* *Chart.yaml* datne satur informāciju par *Helm* pakotni:

apiVersion: v1

appVersion: 0.1.0 # Komponentes versija (!!!)

description: A Helm chart for Kubernetes # Komponentes apraksts

name: vraa-sampleproduct-sampleapi # Komponentes nosaukums, sakrīt ar mapes nosaukumu (!!!)

version: 0.1.0 # Komponentes versija (!!!)

* *values.yaml* datne satur konfigurācijas vērtības dotajai komponentei, kas nemainās no vides uz vidi. Šajā datnē vērtības ir jānorada “Production ready” stāvoklim, un pēc nepieciešamības tos maina ar konkrētas vides konfigurācijas parametriem.

# Default values

# This is a YAML-formatted file.

# Declare variables to be passed into your templates.

replicaCount: 1 # servisa instances skaits

image:

  repository: nexus.abc:5001 # repozitorija aderse - tiek nomainita automatiski deployment brīdī

  name: "vraa/sampleproduct/sampleapi" # Docker image nosaukums (!!!)

  tag: 0.1.0 # Docker Image tag - versija (!!!)

  pullPolicy: IfNotPresent

namespace: "application-namespace" # Kubernetes namespace (!!!)

timestamp: 0

nameOverride: ""

fullnameOverride: ""

service:

  type: ClusterIP # Kubernetes servisa tips

  port: 80 # Kubernetes servisa port

resources: # Kubernetes resusrsu definicija priekš šis komponentes (!!!)

  limits: # Maksimāli pieļaujami resusrsi

    cpu: 100m

    memory: 128Mi

  requests: # Resursu rezervācija

    cpu: 100m

    memory: 128Mi

nodeSelector: {}

tolerations: []

affinity: {}

mongodb: # Parametri priekš MongoDB pieslēguma

  #username: "mongouser"

  #password: "mongopass"

  database: "database"

  connectionParams: # papildus parametri prikš MongoDb connection string

    readPreference: "primaryPreferred"

postgres: # PostgreSQL pieslēguma parametri

  #username: "pguser"

  #password: "pgpassword"

  database: "dbname"

rabbitmq: # RabbitMQ pieslēguma parametri

  #username: "rabbitmqusername"

  #password: "rabbitmqpassword"

  vhost: "vHostName"

elasticsearch: {} # ElasticSearch pieslēguma parametri

  #username: "elkusername"

  #password: "elkpassword"

sidecar: # Istio Sidecar resursa aizpildīšanas parametri (!!!)

  shared: # speciāli servisi kas tiek aprakstīti values-globas.yaml vides datnē

  - mongodb

  - redis

  - elasticsearch

  - rabbitmq

  - postgres

  custom: # no vides neatkarīgi resursi

  - "./\*" # visi resursi tekoša namespace

  - "epak-internal/lvp-eserviceplatform-backend-edkapi.epak-internal.svc.cluster.local" # konkrets serviss epak-internal namespace

  - "viss-system/viss-apimanagement-transactionapi.viss-system.svc.cluster.local" # konkrets serviss viss-system namespace

config: # komponentes papildus konfigurācija – piemēra noradītām vērtībām ir jābūt atbilstoši biznesa funkcionalitātes nosauktam un aizpildītam (!!!)

aspnetcore:

environment: Production

basepath: /lvp-eserviceplatform-backend-configurationapi

logLevel: Warning

searchEngineClient: {}

### Helm Chart templates mape

*Templates* mape satur *Kubernetes* resursu definīcijas, kas tiek nodotas uz *Kubernetes* klasteri pie nodevuma izmitināšanas. Datnes, kas atrodas šajā mapē, pēc būtības ir veidnes, kas tiek aizpildītas ar attiecīgiem parametriem komponentes izmitināšanas brīdī ar automātiskā izmitināšanas procesa palīdzību.

1. **\_helpers.tpl** un **\_abc\_helpers.tpl** – datnes, kas satur palīgklases, kas tiek izmantotas citas definīcijās;
2. **serviceaccount.yaml** – satur *Kubernetes Service Account* definīciju, tā ir nepieciešama, lai darbotos *Kubernetes* drošības slānis;
3. **sidecar.yaml** – datne satur *Istio Sidecar* resursa definīciju;

apiVersion: networking.istio.io/v1alpha3

kind: Sidecar

metadata:

  name: {{ include "fullname" . }}

  namespace: {{ required "A valid namespace entry required!" .Values.namespace | quote }}

spec:

  workloadSelector:

    labels: # Atsauce uz POD template labels elementiem, pēc kuriem notiek Sidecar objekta piesaiste pie POD objekta

      app: {{ include "name" . }}

      release: {{ .Release.Name }}

  egress:

  - hosts:

    - "istio-system/\*"

    {{- if .Values.sidecar }}

    {{- if .Values.sidecar.custom }}

    {{- range .Values.sidecar.custom }}

    {{- printf "- \"%s\"" . | nindent 4 }}

    {{- end }}

    {{- end }}

    {{- if .Values.sidecar.envspecific }}

    {{- range .Values.sidecar.envspecific }}

    {{- printf "- \"%s\"" . | nindent 4 }}

    {{- end }}

    {{- end }}

    {{- if .Values.sidecar.shared }}

    {{- include "sharedSidecar" . }}

    {{- end }}

    {{- end }}

1. **configmap.yaml** – neobligāta datne, kas satur vērtības, kuras iespējams piemontēt konteinera failu sistēmai kā datni pie POD pacelšanas. Kā piemēru, var minēt .*net* projektiem izmantoto *appsettings.json* datni, kuru paredzēts veidot izmantojot konkrētas vides parametrus izmitināšanās brīdī;
2. **service.yaml** – datne, kas apraksta *Kubernetes* *Service* resursu; šai datnei ir jābūt obligātai *Helm* pakotnē, jo tā nodrošina *Service* *Discovery* funkcionalitāti *Kubernetes* orķestrācijas platformā:

apiVersion: v1

kind: Service

metadata:

  name: {{ include "fullname" . }}

  namespace: {{ required "A valid namespace entry required!" .Values.namespace | quote }}

  labels:

    app: {{ include "name" . }}

    chart: {{ include "chart" . }}

    release: {{ .Release.Name }}

    heritage: {{ .Release.Service }}

spec:

  type: {{ .Values.service.type }}

  # Ja tiek publicēts serviss kuram nav eksponēta API, piemēram servisam Wroker tipa, tad jāatkomentē nakamo rindu un jāaizkometnē targetPort

vertību "ports" masīvā

  #clusterIP: None

  ports:

    - port: {{ .Values.service.port }}

      targetPort: http  # eksponēta servisa porta nosaukums atbilstoša POD definicijā - deployment.yaml

      protocol: TCP

      name: http # service porta nosaukums - jābūt ar http prefiksu

  selector:

    app: {{ include "name" . }}

    release: {{ .Release.Name }}

1. **deployment.yaml** – piemērs *Kubernetes* *Deployment* resursa specifikācija, kas apraksta izmitināmo komponenti:

apiVersion: apps/v1

kind: Deployment

metadata:

  name: {{ include "fullname" . }}

  namespace: {{ required "A valid namespace entry required!" .Values.namespace | quote }}

  labels:

    app: {{ include "name" . }}

    chart: {{ include "chart" . }}

    release: {{ .Release.Name }}

    heritage: {{ .Release.Service }}

    version: {{ .Values.image.tag }}

spec: # Deployment specifikācija

  replicas: {{ .Values.replicaCount }}

  selector:

    matchLabels:

      app: {{ include "name" . }}

      release: {{ .Release.Name }}

  template: # POD specifikācija - metadati

    metadata:

      labels:

        app: {{ include "name" . }}

        release: {{ .Release.Name }}

        version: {{ .Values.image.tag }}

      annotations:

        timestamp: {{ required "A valid timestamp entry required!" .Values.timestamp | quote}}

    spec: # POD specifikācija - konteineri un datnes

      serviceAccountName: {{ include "fullname" . }}

      imagePullSecrets:

      - name: "pullsecret"

      dnsConfig:

        options:

          - name: ndots

            value: "1"

      containers: # konteineru apraksts

        - name: {{ include "name" . }}

          image: "{{ .Values.image.repository }}/{{ .Values.image.name }}:{{ .Values.image.tag }}"

          imagePullPolicy: {{ .Values.image.pullPolicy }}

          ports: # konteinera eksponējamais ports

          - name: http # porta nosaukums - jabūt ar http prefiksu, tiek izmantots Service definicijā

            containerPort: 80 # tcp ports kas ir eksponēts no konteinera

            protocol: TCP

          env: # Linux Environmet mainigie, kas tiek uzstādīti pie POD konteinera palaišanas

          # Piemērs MongoDB konfigurācija

          - name: MONGODB\_USER

            value: {{ required "A valid mongodb username entry required!" .Values.mongodb.username | quote }}

          - name: MONGODB\_PASSWORD

            value: {{ required "A valid mongodb password entry required!" .Values.mongodb.password | quote }}

          - name: MONGODB\_CONNSTR

            value: {{ include "mongodbConnStr" . | quote }} # Tiek izmantota paligklāse lai izveidotu connection string no vides parametriem

          # Piemērs PostgreSQl pieslēgums

          - name: PG\_USER

            value: {{ required "A valid postgresql username entry required!" .Values.postgres.username | quote }}

          - name: PG\_PASSWORD

            value: {{ required "A valid postgresql password entry required!" .Values.postgres.password | quote }}

          - name: PG\_HOST

            value: {{ required "A valid postgresql host entry required!" .Values.global.postgres.host | quote }}

          - name: PG\_PORT

            value: {{ required "A valid postgresql port entry required!" .Values.global.postgres.port | quote }}

          # Piemērs - Elasticsearch pieslēgums

          - name: ELASTICSEARCH\_USER

            value: {{ required "A valid elasticsearch username entry required!" .Values.elasticsearch.username | quote }}

          - name: ELASTICSEARCH\_PASSWORD

            value: {{ required "A valid elasticsearch password entry required!" .Values.elasticsearch.password | quote }}

          - name: ELASTICSEARCH\_HOSTS

            value: {{ include "elkHosts" . | quote }} # Tiek izmantota paligklāse lai izveidotu hostu sarakstu no vides parametriem

          - name: ELASTICSEARCH\_TLS

            value: {{ .Values.global.elasticsearch.tls | quote }}

          # Piemērs RabbitMQ pieslēgums

          - name: RABBITMQ\_USER

            value: {{ required "A valid rabbitmq username entry required!" .Values.rabbitmq.username | quote }}

          - name: RABBITMQ\_PASSWORD

            value: {{ required "A valid rabbitmq password entry required!" .Values.rabbitmq.password | quote }}

          - name: RABBITMQ\_HOSTS

            value: {{ include "RmqHosts" . | quote }} # Tiek izmantota paligklāse lai izveidotu hostu sarakstu no vides parametriem

          - name: RABBITMQ\_TLS

            value: {{ .Values.global.rabbitmq.tls | quote }}

          # Piemērs - Redis Sentinel pieslēgums

          - name: REDIS\_SENTINELS

            value: {{ include "RedisHosts" . | quote }} # Tiek izmantota paligklāse lai izveidotu hostu sarakstu no vides parametriem

          - name: REDIS\_MASTER

            value: {{ required "A valid Redis master name entry required!" .Values.global.redis.name | quote }}

          - name: REDIS\_PASSWORD

            value: {{ required "A valid Redis master password entry required!" .Values.global.redis.password | quote }}

          # Ārejo resursu montējums pie konteinera, volume definicija ir zemāk "volumes" sekcijā

          volumeMounts:

          - name: config-volume # datne no ConfigMap

            mountPath: /app/appsettings.json

            subPath: appsettings.json

          - name: cache-volume # īslaiciga mape

            mountPath: /app/tempfolder

          livenessProbe: # Kubernetes Livenes probe definīcija

            httpGet:

              path: /

              port: http

          readinessProbe: # Kubernetss Readiness probe definīcjia

            httpGet:

              path: /

              port: http

          resources:

            {{- toYaml .Values.resources | nindent 12 }}

      volumes: # volume definīcijas

      - name: cache-volume # īslacīga mape,kas tiek veidota automātiski un tiek notirīta pēc POD izbeigšānas

        emptyDir:

          sizeLimit: 10Mi # maskimālais pieļaujamais mapes izmērs

      - name: config-volume # datne kas tiek veidota no nodevumā iekļauta ConfigMap resursa

        configMap:

          name: {{ include "fullname" . }}

      {{- with .Values.nodeSelector }}

      nodeSelector:

        {{- toYaml . | nindent 8 }}

      {{- end }}

      {{- with .Values.affinity }}

      affinity:

        {{- toYaml . | nindent 8 }}

      {{- end }}

      {{- with .Values.tolerations }}

      tolerations:

        {{- toYaml . | nindent 8 }}

      {{- end }}

1. **cronJob.yaml** – regulāro darbību definīcija, kura ir pievienojama pēc nepieciešamības.

*CronJob* ir *Kubernetes* resurss, kas tiek pielietots, kad ir nepieciešams veikt kāda konteinera palaišanu ar noteiktu regularitāti, pie nosacījuma, kad konteiners izpildot iepriekš definētas darbības pārtrauc savu darbību. Pilna *CronJob* resursa API definīcija ir pieejama pēc adreses https://v1-18.docs.kubernetes.io/docs/reference/generated/kubernetes-api/v1.18/#cronjob-v1beta1-batch

Gadījumā, ja tiek izmatots šāda tipa resurss, nepieciešams nodrošināt papildus *Sidecar* objekta definīciju, kas tiek piekabināta pie *CronJob* definētās vērtības zem  
spec.jobTemplate.spec.template.metadata.labels.app elementa.

*CronJob* palaišanas intervāls tiek uzdots atbilstoši *Cron* definīcijai – <https://en.wikipedia.org/wiki/Cron>

Piemēri:

# Palaist CronJob katru stundu 30 minutē – 00:30 , 01:30 un tml.

config:

schedule: "30 \* \* \* \*"

# Palaist CronJob katras 10 minutēs

config:

schedule: "\*/10 \* \* \* \*"

# Palaist CronJob katru sestdienu 23:45

config:

schedule: " 45 23 \* \* 6"

Definējot palaišanas intervālu, jāņem vērā to, ka konteinerī laiks tiek uzskaitīts pēc GMT (UTC) laikā zonas!

apiVersion: batch/v1beta1

kind: CronJob

metadata:

  name: vraa-sampleproduct-sampleapi-cronjob # CronJob nosaukums, jabut ar postfiksu cronjob

  namespace: {{ required "A valid namespace entry required!" .Values.namespace | quote }}

spec: # CronJob specifikācija

  concurrencyPolicy: "Forbid"

  schedule: {{ required "A valid Schedule required!" .Values.config.schedule | quote }} # Palaišanas intervāls no konfigurācijas datnes

  startingDeadlineSeconds: 3600

  jobTemplate:

    spec: # Job specifikācija, kas tiek pielietota katru reizi, kad Job uzstartējas

      backoffLimit: 0 # Specificē, cik reizes Job var restartēties pirm atzīt to par Failed

      template: # POD specifikācija, kas tiek palaista zem JOB definīcijas

        metadata: # POD metadata, kas tiek palaista zem JOB definīcijas

          labels: # Obligāti japielieto "app" label, uz kuru atsaucās Sidecar definīcija

            app: "vraa-sampleproduct-sampleapi-cronjob"

        spec: # POD definīcija, kas tiek palaista zem JOB definīcijas

          dnsConfig:

            options:

            - name: ndots

              value: "1"

          imagePullSecrets:

          - name: "pullsecret"

          restartPolicy: OnFailure

          serviceAccountName: {{ include "fullname" . }}

          volumes:

          - name: appsettings-volume

            configMap:

              name: {{ template "fullname" . }}

          containers:

          - name: {{ include "name" . }}

            image: "{{ .Values.image.repository }}/{{ .Values.image.name }}:{{ .Values.image.tag }}"

            imagePullPolicy: {{ default "IfNotPresent" .Values.image.pullPolicy }}

            args:

            - "-i"

            envFrom:

            - secretRef:

                name: {{ include "fullname" . }}

            volumeMounts:

            - name: appsettings-volume

              mountPath: /app/appsettings.json

              subPath: appsettings.json

            resources:

              {{- toYaml .Values.resources | nindent 14 }}

          {{- with .Values.nodeSelector }}

          nodeSelector:

            {{- toYaml . | nindent 12 }}

          {{- end }}

          {{- with .Values.affinity }}

          affinity:

            {{- toYaml . | nindent 12 }}

          {{- end }}

          {{- with .Values.tolerations }}

          tolerations:

            {{- toYaml . | nindent 12 }}

          {{- end }}

# Vides atkarīga konfigurācija

Katrā vidē tiek izmantotas neatkarīgas datnes, kas nosaka globālās un konkrētās komponentes konfigurācijas vērtības. Izstrādātāja pienākums kopā ar nodevumu sniegt komponentes konfigurācijas datnes paraugu, ar visām nepieciešamajām vērtībām, lai VRAA administratori to ielādētu attiecīgajā *Git* struktūrā.

Veicot nodevuma uzstādīšanu, dažādu līmeņu konfigurācijas tiek apvienotas šādas prioritātes secībā:

1. *Helm Chart* iekļauta *values.yaml* datne;
2. Komponentes vides konfigurācijas datne;
3. Globāla vides konfigurācijas datne.

Konfigurācijas vērtības tiek apvienotas, ievērojot šādus principus:

* Augstākā līmeņa datnes noradītas vērtības pārraksta vai papildina iepriekšējo līmeni;
* Objektu vērtības tiek apvienotas;
* Masīva vērtības tiek pārrakstītas.

Vērtību apvienošanas piemērs:

# A. līmeņa konfigurācija

config:

key1: "value1-1"

key2: "value1-2"

array:

- a1

- a2

# B. līmeņa konfigurācija

config:

key2: "value2-2"

key3: "value2-3"

array:

- a3

- a4

# rezultējoša konfigurācija

config:

key1: "value1-1"

key2: "value2-2"

key3: "value2-3"

array:

- a3

- a4

## Globāla vides konfigurācija

Vides globāla konfigurācija satur koplietojamajās vērtības, kas nemainās vairākām vienas sistēmas komponentēm, tādas, kā piemēram:

* *Docker* repozitorija adrese;
* Ārējās krātuves (*MongoDB*, *PostgreSQL*, *ElasticSearch* utt.) adreses un konfigurācija[[2]](#footnote-3);
* Dažādas vides atkarīgas adreses vai parametri, kas ir koplietojamas no vairākām komponentēm.

Vides globāla konfigurācija tiek pārvaldīta tikai un vienīgi no VRAA administratoru puses. Katrai platformai/sistēmai ir sava globālās konfigurācijas datne.

Zemāk ir dots piemērs, kā var izskatīties Lvp platformas vides globālā konfigurācija.

image:

repository: nexusrep.vraa.gov.lv

# For ZZDats Components

imagePullSecrets: pullsecret

global:

vpm:

baseAddress: "http://epakvisstv.vraa.gov.lv/STS/VISS.LVP.STS/Default.aspx"

metadataAddress: "http://epakvisstv.vraa.gov.lv/STS/VISS.LVP.STS/FederationMetadata/2007-06/FederationMetadata.xml"

pfasAuth:

baseAddress: "http://epakvisstv.vraa.gov.lv/STS/VISS.Pfas.STS"

wso2:

baseAddress: "http://apitestgw.vraa.gov.lv"

searchEngine:

baseAddress: "http://ventabalancer.vraa.gov.lv/VISS.SearchEngine/WS/stable/SearchEngine.svc"

realm: "https://epakvisstv.vraa.gov.lv/VRAA.VISS2.SE/SearchEngine.svc"

requestService:

baseAddress: "http://ausmatest8.vraa.gov.lv/Request.WebService/v1-9/WcfService"

anonymousEndpointAddress: "http://ausmatest8.vraa.gov.lv/Request.WebService/v1-9/WcfService/soap12WsAddresing"

securedEndpointAddress: "http://ausmatest8.vraa.gov.lv/Request.WebService/v1-9/WcfService/ws2007FederationNoSct"

realm: "URN:VISSTV:VISS.REQUEST.SERVICE"

edk:

baseAddress: "http://ventabalancer.vraa.gov.lv/VISS.EDK/WS2/stable"

realm: "URN:TEST:VISS.EDK.WS2"

notificationService:

baseAddress: "http://ausmatest8.vraa.gov.lv"

realm: "https://ivis.eps.gov.lv/NOT.WebService"

addressFinder:

baseAddress: "https://vissapi-test.vraa.gov.lv/AMK.AddressFinder.RestApi/v1.0"

eservicePlatform:

assetsBaseAddress: "https://eservices-test.vraa.gov.lv/EservicePlatform.Assets"

contextBaseAddress: "https://eservices-test.vraa.gov.lv/EservicePlatform.ContextApi"

navigationBaseAddress: "https://eservices-test.vraa.gov.lv/EservicePlatform.NavigationApi"

gatewayBaseAddress: "https://eservices-test.vraa.gov.lv"

idsBaseAddress: "https://epakvisstv.vraa.gov.lv/IdentityServer"

searchUrl: "https://lvptest.vraa.gov.lv/{language}/Meklesana"

executedEserviceUrl: "https://lvptest.vraa.gov.lv/{language}/KDV/E\_pakalpojumu\_saraksts"

idleTime: "7"

globalResourceUrl: "https://epakvisstv.vraa.gov.lv/Lvp.EservicePlatform.Resources/Global/global.yaml"

breadcrumbs:

- lv: "Sākums"

ru: "Начало"

en: "Home"

link: "https://lvptest.vraa.gov.lv/{language}"

- lv: "E-pakalpojumi"

ru: "Э-услуги"

en: "E-services"

link: "https://lvptest.vraa.gov.lv/{language}/Epakalpojumi"

- lv: "{eserviceName}"

en: "{eserviceName}"

ru: "{eserviceName}"

link: "https://lvptest.vraa.gov.lv/{language}/Epakalpojumi/{eserviceId}"

queryEncryptionKey: "SA54df1s6d1G#56sdf1we65wer65w1er"

mssql:

datasource: "diana2.viss.int"

host: "diana2.viss.int"

rabbitmq:

hosts:

- host: venta1.viss.int

port: 5672

tls: false

## Komponentes vides konfigurācija

Komponentes vides konfigurācijas datne satur konkrētas komponentes konfigurācijas vērtības, kas mainās no vides uz vidi. Piemēram, ārējas krātuves pieslēguma rekvizīti vai citas vērtības, kas ietekmē komponentes darbību.

Komponentes vides konfigurācijas datnes šablonam ir jābūt iesniegtam kā nodevuma sastāvdaļa ar komentāriem vai aprakstītam administratora rokasgrāmatā, pēc kuriem VRAA administratori aizpilda to ar savas vides attiecīgajām vērtībām.

Piemērs komponentes konfigurācijai:

replicaCount: 1 # Pēc noklusējuma testa vidē ir ceļama tikai 1 replika katram mikroservisam, savukārt produkcijā minimālo repliku skaits ir 2 (turklāt automātika centīsies izmitināt un sadalīt noslodzi starp dažādiem VRAA pieejamiem infrastruktūras datu centriem).

config:

logLevel: Information

cache:

distributed:

initialCatalog: "EServiceCache"

userId: "dbuser"

password: "dbparole"

lvpIds:

credentials:

clientId: "ContextAPI"

clientSecret: "6cff5a3f4e57431bd3fa772a34534503454f1d0c0a86d85f83377ba3"

baseAddress: "http://epakvisstv.vraa.gov.lv/IdentityServer"

pfasAuth:

signing:

certificateFileName: "certificates/Lvp.EservicePlatform.Backend.ContextApi.pfx"

certificatePassword: "certpsw!"

realm: "urn:oauth2:8f423103-eb3a-45b0-9a74-d88d0c79e6bb"

policyDecision:

cacheExpirationSeconds: "600"

sidecar:

envspecific:

- "\*/epakvisstv.vraa.gov.lv"

- "\*/ausmatest8.vraa.gov.lv"

- "\*/apitestgw.vraa.gov.lv"

- "\*/ventabalancer.vraa.gov.lv"

## Helm chart un konfigurācijas pārbaude

Izstrādātajam ir iespēja pārbaudīt gan *Helm Chart*, gan gala rezultāta pareizību, izmantojot *helm* komandas palīdzību:

helm template –n “komponetes-nosaukums” <Helm Chart mape> -f komponentes\_konfigurācijas.yaml –f globalas\_konfigurācijas.yaml

# Piegādājamās programmatūras realizācijas labas prakses un rekomendācijas

Izstrādājot komponentes kodu, izstrādātājiem jāpieturas pie dažādiem labas prakses piemēriem, kas izriet no *Kubernetes* orķestrācijas platformas, ārējo krātuvju darbības principiem un VRAA K8s klastera uzbūves topoloģijas.

## Konteineru resursu konfigurācija un pārvaldība

Veicot konteinera koda izstrāde ir jāņem vērā, ka darbojoties *Kubernetes* platformā konteineriem ir pieejami ierobežotā daudzuma resursi – tādi ka CPU un RAM. Veicot izstrādi ir jānodefinē resursu daudzums, kuru patērē kods normālas darbības stāvoklī (pie noslodzes), un šīs vērtības ir jānorada *Helm* konfigurācijas datnē *values.yaml resources* blokā (skat. 4.1.1. paragrāfu).

Resursu rezervācija nosaka minimālo resursu pieprasījumu, lai konteiners varētu palaisties un normāli strādāt pie slodzes.

Resursu limits nosaka maksimāli pieļaujamo resursu patēriņu ar šādiem nosacījumiem:

* Ja konteiners patērē virs CPU limita – koda izpilde tiek ierobežota šī limita ietvaros;
* Ja konteiners patērē virs RAM limita – konteiners automātiski tiek apturēts ar pazīmi (OOM killed), un tiek startēta jauna konteinera instance;

Ja no koda ir nepieciešams saglabāt kādu datni, to nedrīkst darīt konteinera failu sistēmā, bet jāpiemontē, izmantojot *Deployment* definīciju “emptyDir” veidā *Kubernetes volume*. Pretējā gadījumā pēc konteinera restarta šie dati būs zaudēti bez atjaunināšanas iespējām.

## Labas prakses darbam ar ārējām datu krātuvēm

Veicot darbības ar ārējām datu krātuvēm, tādām kā *MongoDB*, *PostgreSQL*, *ElasticSearch* un *RabbitMQ*, jāpieturas pie noteiktiem darbības principiem:

1. Tā kā ārējās krātuves infrastruktūras līmeņi tiek organizēti klasteros, kas kopīgoti izmantošanai no vairākiem VRAA datu centriem, formējot savienojumus, ir jālieto ENV mainīgie, kas definē piekļuves akreditācijas datus (*username, password*), *hosts* adreses, iekļaujot portus un specifiskus savienojumu parametrus (krātuves nosaukums, TLS atbalsts, persistences prasības utml.) atkarībā no krātuves veida.
2. Jāizmanto attiecīgās API bibliotēkas iespējas, lai nodrošinātu atkārtotu pieslēgšanu, gadījumā, ja pazūd savienojums ar krātuvi.
3. Strādājot ar *ElasticSearch* meklēšanas indeksiem, pie mikroservisa pacelšanas ir jāvalidē indeksa *mapping*, lai pārliecinātos, ka indekss atbilst esošai saskarnes versijai. Pretējā gadījumā mikroservisam ir jāiziet ar kļūdu, ierakstot sistēmas žurnālā informāciju par indeksa neatbilstību. Sākotnēja indeksa konfigurācijā jānorada replicēto indeksu instanču skaits – 0 (šī vērtība ir dinamiski maināma produkcijas vidē), un indeksa dalījumu (*shard*) skaits – 1, lielāko dalījumu skaitu iespējams pielietot gadījumā, ja paredzētais indeksa izmērs būs vairāk nekā 5-10 GiB.

## CronJob realizācijas rekomendācijas

Realizējot funkcionalitāti, kas tiek darbināta kā regulāra darbība, izmantojot *CronJob* definīciju, nepieciešams iestrādāt kodā šāda veida pārbaudes:

1. Pirms veikt izsaukumus uz ārējiem resursiem nepieciešams pārliecināties, ka *Istio-proxy* konteiners ir pacelts un gatavs darbībai. Lai to izdarītu, ir nepieciešams cikliski, ar nelielu aizturi, veikt HTTP GET izsaukumus uz adresi <http://127.0.0.1:15020/healthz/ready> kamēr netiks saņemts 200 atbildes kods.
2. Pie koda pabeigšanas vai jebkādas kļūdas, kas izraisa izeju no koda ir nepieciešams iestrādāt HTTP POST izsaukumu uz adrese <http://127.0.0.1:15020/quitquitquit>, lai nodrošinātu *istio-proxy* apstādināšanu. Šīs solis ir kritisks, jo citādāk *Job* POD mūžīgi paliek *Running* statusā.

Minētās adreses nepieciešams iestrādāt Helm values.yaml datnē, lai būtu iespējas tos mainīt caur vides parametriem, gadījumā, ja tas tiek mainītas Istio realizācijā, bez papildus koda pārkompilācijas.

## API izsaukumu apstrādes specifika

Sistēmā ir izmantojamas dažādas HTTP pieprasījumā sūtītas *Header* vērtības, kas tiek izmantotas sistēmas pārvaldībās slānī.

Nepieciešams iestrādāt kodā realizāciju, kas atbilst šādām prasībām:

1. Ja API veic izsaukumu uz citu servisu, balstoties uz saņemto pieprasījumu, nepieciešams padot izsaukumā sekojošas *Header* vērtības bez izmaiņām, nokopējot to no pieprasījuma:

* *x-tabId* (reprezentē lietotāja darba sesiju; kopīgota visiem izsaukumiem);
* *x-transactionId* (reprezentē konkrētu transakciju kuras ietvaros tiek veikti izsaukumi; vienā izsaukumu ķēdē šī vērtība ir nemaināma);
* *x-milestoneId* (pieturpunkta identifikators, identificē procesa pašreizējo stāvokli);
* x-*b3*-*traceid* (tehniskie *header* priekš *Istio* *Distributed* *Tracing*);
* *x-b3-spanid* (tehniskie *header* priekš *Istio* *Distributed* *Tracing*);
* *x-b3-parentspanid* (tehniskie *header* priekš *Istio* *Distributed* *Tracing*);
* x-b3-sampled (tehniskie *header* priekš *Istio* *Distributed* *Tracing*);
* x-b3-flags (tehniskie *header* priekš *Istio* *Distributed* *Tracing*);
* *x-ot-span-context* (tehniskie *header* priekš *Istio* Distr*i*buted *Tracing*);
* *b3* (tehniskie *header* priekš *Istio* Distr*i*buted *Tracing*);
* *x-request-id* (tehniskie *header* priekš *Istio* *Distributed* *Tracing*).

1. Ja uz servisu tiek padotas citas *Header* vērtības pēc biznesa, lietojumam tā jāapstrādā pēc biznesa vajadzībām un jāpieņem lēmums par nepieciešamību tās pārsūtīt tālāk;
2. *Header* nosaukumu saraksts, kas tiek pārsūtīts uz nākamo izsaukumu, nepieciešams iestrādāt kā konfigurējamo vērtību *Helm values.yaml* datnē.

## Kubernetes izmitināmā konteinera statusa pārbaudes (*probes*)

*Kubernetes* *deployment* definīcija (skat. 4.1.2.sadaļu) satur *Readiness* un *Liveness* pārbaudes:

* *Readiness* pārbaude ir obligāta un kalpo, lai noteiktu konteinera statusu, vai tas ir gatavs apkalpot ienākošos pieprasījumus. Realizējot šo pārbaudi nepieciešams, lai tas notiktu ātri un neietekmētu pamatkoda darbību. Uz šo izsaukumu jāatgriež 200 HTTP kods pie nosacījuma, ka visi koda darbībai nepieciešamie resursi ir pieejami. Ņemot vērā, ka šī pārbaude notiek ar regulāro izsaukumu no platformas pārvaldības slāņa, nav pieļaujams, ka šī pārbaude veic ilgstošas vai resursu ietilpīgas pārbaudes. Ja šī pārbaude nesaņem 200 HTTP kodu pēc noteikta (konfigurējama) mēģinājumā skaita, PODam tiek piešķirts statuss *NotReady*, un tas tiek izņemts no apkalpes procesa, kamēr šī pārbaude neatgriezīs normālu atbildi.
* *Liveness* pārbaude kalpo, lai noteiktu konteinera “dzīvības” statusu. Ja šī pārbaude neatgriež 200 HTTP atbildi pēc noteikta (konfigurējama) mēģinājuma skaita, konteiners tiek automātiski restartēts.

## Kubernetes izmitināmā konteinera veiksmīga apstādināšana (*graceful shutdown*)

Lietojuma kodā ir jāapstrādā SIGINT un SIGTERM signāli no OS slāņa, kas tiek padoti, ja *Kubernetes* platformai ir nepieciešams apstādināt piegādātās komponentes konteineri.

Apstrādājot šos signālus nepieciešams atslēgt ienākošo trafiku apstrādi, veiksmīgi pabeigt tekošos darbus, un jāiziet no koda ar 0 statusu (*success*).

## Piegādājamo konteineru izmitināšana pa vairākiem datu centriem

Ņemot vērā VRAA infrastruktūras specifiku, katrai izstrādājamajai komponentei jeb mikroservisam jābūt uzprojektētam tā, lai varētu pacelt paralēli vairākas konteineru instances (produkcijas vidē minimums 2), kas savā starpā nav saslēgtas. Izsaukumu maršrutēšana uz šīm instancēm notiek pēc *Round-Robin* principa, un nevar garantēt, ka secīgi izsaukumi no cita lietojuma nokļūst uz vienu un to pašu instance. Ņemot vērā šo uzvedību lietojuma projektējumā jāparedz, ka konteinera kods darbojās bez piesaistes pie lietotāja sesijas (t.i. session-less arhitektūrā). Ja nav iespējams nodrošināt šo prasību ievērošanu biznesa specifikas dēļ, sesijas uzgrābāšanai jāizmanto kopīgoto ārējo krātuvi (piemēram, *Redis Sentinel, MongoDB*), un jāparedz procesu, kas iztīra no krātuves vairs nevajadzīgus pierakstus.

## Sistēmas žurnāla pierakstu veidošana konteinerī

VRAA platformas arhitektūra paredz, ka katras biznesa komponentes izpildkods ir atbildīgs par sistēmas žurnāla pierakstu veidošanu. Bet šiem pierakstiem ir jābūt veidojamiem pēc noteikta JSON formāta specifikācijas, kas ir nodefinēta [1] dokumentā, un izvadāmiem uz STDOUT konteinera plūsmu. No šīs plūsmas tie automātiski tiek pārķerti ar *Docker* dzinēja palīdzību un nodoti tālākai apstrādei VRAA K8s platformai.

## Docker konteinera reģionālie uzstādījumi

Pēc noklusējuma Docker konteineri tiek izmantotas POSIX lokalizācijas uzstādījumi, un UTC laika zona.

Jā pēc biznesa ir nepieciešams lai kods darbotos ar citiem uzstādījumiem nepieciešams veikt labojumus Dockerfile datnes saturā, blokā kas atbild par izpildāmo daļu.

Piemēri ir doti priekš .net konteineriem kas ir bāzēti uz Alpine , priekš cita veida konteineriem jāseko to dokumentācijai.

Lai izmantotu Latvijas laika zonu datumiem un laikiem:

FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-alpine

# Install cultures (same approach as Alpine SDK image)

RUN apk update && apk add --no-cache icu-libs tzdata

RUN cp /usr/share/zoneinfo/Europe/Riga /etc/localtime

RUN echo "Europe/Riga" > /etc/timezone

ENV TZ Europe/Riga

WORKDIR /app

Lai izmantotu Latvijas reģionālus uzstādījumus:

FROM mcr.microsoft.com/dotnet/core/aspnet:3.1-alpine

# Install cultures (same approach as Alpine SDK image)

RUN apk update && apk add --no-cache icu-libs tzdata

ENV LANG lv\_LV.UTF-8

ENV LANGUAGE lv\_LV.UTF-8

ENV LC\_ALL lv\_LV.UTF-8

# Disable the invariant mode (set in base image)

ENV DOTNET\_SYSTEM\_GLOBALIZATION\_INVARIANT=false

WORKDIR /app

# Nodevuma gatavības *check-list* automātiskajai piegādei un izmitināšanai VRAA Kubernetes platformā

Šajā tabulā ir apkopoti esošajā dokumentā aprakstītas prasības, kas ir jāpārbauda pirms nodevuma piegādes un nosaka tā gatavību izmitināšanai klienta uzstūrētajā Kubernetes infrastruktūrā.

3.tabula

Nodevuma pārbaudes check-list

|  |  |
| --- | --- |
| Pārbaudes apraksts | Atbilstība |
| 1. Docker image atbilst VRAA komponenšu nosaukumu piešķiršanas un versionēšanas principiem |  |
| 1. Helm chart atbilst VRAA komponenšu nosaukumu piešķiršanas un versionēšanas principiem |  |
| 1. Helm chart values.yaml datne: |  |
| * satur Production-ready vērtības |  |
| * izņemtas vai aizkomentētas vērtības, kas tiek mainītas no vides uz vidi un tiem nav noklusētas vērtības (piem. lietotājvārdi un paroles, ārējo servisu adreses) |  |
| * satur Istio Sidecar konfigurāciju priekš vajadzīgiem ārējiem resursiem |  |
| * satur CPU/RAM konteineru resursu atbilstošas vērtības |  |
| 1. Helm Chart satur ServiceAccount resursa definīciju |  |
| 1. Helm Chart satur Service resursa definīciju |  |
| 1. Helm Chart satur Deployment resursa definīciju: |  |
| * satur ReadinessProbe konfigurāciju |  |
| * satur LivenessProbe konfigurāciju |  |
| * ENV parametriem tiek pielietota “Quote” funkcija |  |
| 1. Helm Chart satur Configmap resursa definīciju (pēc nepieciešamības) |  |
| 1. Helm Chart satur Sidecar resursa definīciju |  |
| 1. Helm Chart satur papildus Sidecar resursa definīcijās priekš CronJob resursiem (pēc nepieciešamības) |  |
| 1. Helm Chart Chart.yaml datne satur komponentei atbilstošas vērtības |  |
| 1. Helm Chart veidnēs priekš obligātiem parametriem, kuriem nav noklusētas vērtības, tiek izmantota “Required” funkcija |  |
| 1. Priekš CronJob veida konteineriem tiek iestrādāti pārbaudes izsaukumi uz Istio-Proxy |  |
| 1. Savienojumi ar ārējiem datu krātuvēm notiek atbilstoši rekomendācijām |  |
| 1. Žurnālpieraksti tiek rakstīti uz stdout atbilstoši shēmai |  |
| 1. Tiek nodrošināta noteikto pieprasījumā iekļautas header vērtības pārsūtīšanā bez izmaiņām pie cita API izsaukuma |  |
| 1. Docker Image ielādēts VRAA Nexus repozitorijā atbilstoši notācijai |  |
| 1. Helm Chart ieladēts VRAA Nexus repozitorijā atbilstoši notācijai |  |
| 1. VRAA administratoriem nodots komponentes vides konfigurācijas paraugs ar nepieciešamo vērtību aprakstu |  |
| 1. Palaista (ar izsaukumu) VRAA Jenkins automātiskas uzstādīšanas ķēde |  |

1. Gadījumiem, kad iekšējās zonas serviss tiek eksponēts, lai to varētu izsaukt no ārējām sistēmām, tiek pielietoti divi varianti:

   • Serviss tiek izmitināts xxx-system zonā – šis ir rets gadījums, kad serviss tiek veidots tieši publisks.

   • Servisam tiek piešķirts ar *Istio* maršrutēšanas palīdzību ārējais ieejas punkts ar prefiksu *Public* tiek maršrutēts uz iekšējo aizsargātu internal zonu. [↑](#footnote-ref-2)
2. Ir jāņem vērā, ka ārējās krātuves ir paceltas klasteros, tāpēc globālā konfigurācija satur vairākas krātuves *host* adreses, kuras tiek apstrādātas ar TPL palīgklasēm *Helm chart* līmenī un padotas uz komponentes vides konfigurāciju jau salinkotā veidā. [↑](#footnote-ref-3)