



Универзитет „Св. Кирил и Методиј“ во Скопје
Факултет за информатички науки и компјутерско инженерство

ДИПЛОМСКА РАБОТА

Платформа како сервис за менацирање на апликации во облак

Ментор
Доц. д-р Бојан Илијоски

Кандидат
Вангел Трајковски
181261

Јануари 2026

Апстракт

Корисниците се соочуваат со големи предизвици при хостирање на апликации, бидејќи покрај технологиите потребни за развој на самата апликација, неопходно е да поседуваат и познавање за околината во која таа се хостира, без разлика дали станува збор за инфраструктура во облак или за посветен сервер, како и познавање на платформата на која се извршува. Со цел да се надминат овие предизвици, изградено е решение за хостирање на апликации во облак кое овозможува лесно управување, распоредување и увид во состојбата на апликациите преку едноставен и интуитивен кориснички интерфејс.

За реализација на ова решение беа искористени современи технологии како Kubernetes, Spring Framework и други придружни алатки, кои овозможуваат интеграција на повеќе сервиси за управување и распоредување на корисничките апликации. Главната цел на архитектурата на решението е обезбедување на поголема едноставност за крајните корисници, при што истовремено се задржува флексибилноста и можноста за понатамошно проширување на системот. Овие аспекти се оставени во доменот на администраторите на платформата.

Покрај едноставноста, решението овозможува и увид во состојбата на апликациите во реално време, поддршка за градење на контејнерски слики, како и лесно распоредување на апликации во рамките на користената платформа. Додека за администраторите се дозволува додавање на можности за корисниците преку користење на шаблони за градење на сликите, како и управување со ресурсите на платформата.

Клучни зборови: *Хостирање, Kubernetes, градење на слики, Docker, конинуирана интеграција, распоредување на апликации*

Содржина

Апстракт	ii
1 Вовед	1
2 Користени технологии и библиотеки	2
2.1 Kotlin	2
2.2 Gradle	2
2.3 Spring framework	3
2.4 Svelte+kit	3
2.5 Kubernetes	4
2.6 PostgreSQL	5
2.7 Keycloak	6
2.8 Kaniko	6
2.9 Docker registry v2	7
2.10 WebDAV	7
2.11 WebSocket протоколот	8
3 Архитектура на системот	8
3.1 Сервиси	9
3.2 Преглед на системот	9
4 Кориснички интерфејс	12
4.1 Апликации и слики за контејнери	13
4.2 Прикачување на извршна датотека	13
4.3 Градење на извршната датотека	14
4.4 Изданија на апликација	16
4.5 Преглед на групи на хостирани апликации	17
4.6 Преглед на распоредувања на апликации	18
4.7 Креирање на распоредувања за хостирање на апликации	19
4.8 Основни шаблони за градење на апликации	20
4.9 Администрација на извршни датотеки	25
4.10 Администрација на регистарот за слики	26
4.11 Преглед за документација	28
4.12 Позадина на апликацијата	28
5 Предизвици при развој на апликацијата	30
5.1 Овозможување на преглед и следење низ процесот за хостирање	30
5.2 Градење на слики за контејнери	31
5.3 Простор за чување на извршни датотеки и слики	31
5.4 Развој на апликацијата и локално тестирање	32
5.5 Автентикација и авторизација	33
6 Подобрувања и осврт кон иднината	33
7 Заклучок	34
8 Библиографија	35

1 Вовед

Хостирање на апликации во облак е современ пристап кој овозможува флексибилност, скалабилност и лесно управување со апликациите, каде што сите опции за конфигурација од вмрежување до апликациски конфигурации се овозможени преку кориснички интерфејс, кој најчесто знае да биде доста комплициран за корисниците кои немаат претходно искуство со ваков тип на платформи. Целта на ова решение е да се овозможи платформа која ќе овозможи лесно хостирање на апликации во облак, за оваа цел е развиена веб апликација која овозможува корисниците да креираат, управуваат и бришат апликации на лесен начин преку кориснички интерфејс, без потреба од познавање на посложени конфигурации и концепти.

Целта на ова решение е да понуди едноставен, но мокен систем за хостирање на апликации во облак, кој ќе ја апстрагира сложеноста на инфраструктурата и ќе обезбеди интуитивен начин за градење, распоредување и управување со апликации. Главниот фокус е ставен на автоматизација, стандардизација и безбедност, со минимален број на чекори потребни од страна на корисникот. Со користење на современи cloud-native технологии, системот овозможува брзо креирање на контейнерски слики, нивно хостирање и следење, без директна интеракција со самата платформа.

Секогаш кога се бара баланс помеѓу мокност и едноставност, постои ризик од ограничување на функционалностите. Во овој случај, решението е дизајнирано да ги задоволи потребите на корисниците кои бараат брз и лесен начин за хостирање на апликации, додека истовремено обезбедува доволно можности за конфигурација и прилагодување за понапредните корисници, но изостава дел од контролата која ја овозможува платформата, и некои функционалности за контрола се оставени да се менаџираат од страна на администраторот.

Решението нуди можности за лесно хостирање и директен пристап на нивните апликации, овозможувајќи им на корисниците да се фокусираат на развојот на нивните апликации без да се грижат за инфраструктурните детали. Апликацијата е изградена со користење на платформата Kubernetes, која овозможува автоматско управување со апликациите, скалирање и обезбедување на висока достапност, избрана беше оваа платформа поради екстензibilноста што ја нуди и начинот на кој што се менаџираат ресурсите со дадени шаблони значи дека би било полесно да се автоматизираат одредени процеси.

Градењето на апликации знае да биде комплициран процес, кој често вклучува алатки за градење, потоа потребно е овие така наречени артефакти да се хостираат на одреден сервис, и на крајот потребно е да се овозможи пристап до истите од страна на корисниците, кои подоцна од нив би се барало да ги постават на системот каде е хостирана апликацијата.

Еден предизвик со кој се соочуваат корисници е градење на своите апликации, за оваа цел најчесто се користи CI/CD (Continuous integration/ Continuous deployment) цевковод, што значи дека секој пат кога корисникот ќе направи промени во својот код, тие промени автоматски се градат и распоредуваат на платформата. Бидејќи Kubernetes платформата работи со контейнери, потребно е да се овозможи лесно градење на тие контейнери, за оваа цел беше интегриран сервис наречен kaniko кој овозможува градење на Docker слики директно во рамки на кластерот, без потреба од дополнителни привилегии. Дополнително беше потребно да се чуваат овие слики,

за оваа цел беше интегриран регистар за слики кој е овозможен од страна на Docker и се нарекува Docker registry v2, кој овозможува лесно чување и повлекување на слики.

2 Користени технологии и библиотеки

2.1 Kotlin



Kotlin претставува модерен програмски јазик кој се извршува на Java виртуелната машина (JVM), и се смета за алтернатива на Java, кој во споредба со Kotlin се движи побавно и се грижи доста за компатибилноста со претходните верзии. Kotlin е дизајниран од ново во помодерно време и го нема зад себе истиот багаж како Java. Но во споредба со Java е доста покомплициран, и се фокусира на недостатоците на Java заради кои превзема доста карактеристики од други јазици кои доста го модернизираат јазикот, направен е поекспресивен и поконцизен.

Една голема предност пред Java е тоа што Kotlin нуди поголема безбедност при ракување со null вредности, што е чест извор на грешки во Java апликациите. Kotlin има вграден систем за типови кој го прави речиси невозможно да се појави NullPointerException. Kotlin исто така поседува првокласна поддршка за функционално програмирање, овозможувајќи на дефинирање на методи надвор од класи, екстензибилни функции кои се надоврзуваат на постоечки типови, и користење на ламбда изрази за пофлексибилно работење со колекции. Исто така и колекциите во Kotlin се многу поекспресивни и нудат повеќе функционалности во споредба со Java. Во Kotlin нема потреба од користење на стримови за работа со колекции, бидејќи колекциите имаат вградени методи за филтрирање, мапирање и агрегација на податоци.

Kotlin е целосно компатибилен со Java, што значи дека може да се користи заедно со постоечки Java библиотеки, и ова му овозможува лесен премин од Java кон Kotlin. Нема големи ограничувања и брзината е иста како Java. Kotlin е доволно зрел јазик, и до овој момент е доста компатибилен со останати Java технологии, и има голема заедница која го користи и развива.

2.2 Gradle



Gradle^[1] е современ систем за автоматизација на градење кој се користи за управување со процесот на градење, тестирање и распоредување на софтверски проекти. Gradle е дизајниран да биде флексибилен и моќен, овозможувајќи на развивачите да дефинираат сложени градежни процеси преку така наречен DSL (Domain-Specific Language), за оваа намена Gradle дозволува користење на Groovy или Kotlin како јазик за дефинирање на градежните скрипти.

Со овој пристап Gradle нуди целосна функционалност за прилагодување на процесот за градење според специфичните потреби на проектот. Gradle поддржува инкрементално градење, што значи дека само изменетите делови од проектот се повторно

изградени, има добра интеграција со spring boot dev tools која е библиотека за подобрен развој на spring апликации, овозможувајќи побрзо време на развој и тестирање. Gradle има пристап до сите Java библиотеки преку кои доаѓаат преку Maven, но за разлика од Maven, Gradle овозможува поголема флексибилност и прилагодување на процесот на градење.

2.3 Spring framework



Spring рамката претставува еден од најпопуларните и најшироко користени рамки за развој на апликации во Java екосистемот. Тој обезбедува сеопфатен сет на алатки и библиотеки кои го олеснуваат развојот на модерни, скалабилни и одржливи апликации. Една од главните карактеристики на Spring е неговата модуларна архитектура, која овозможува на развиваите да ги изберат само оние компоненти кои им се потребни за нивната апликација. При тоа Spring нуди широк спектар на интеграции со различни технологии и библиотеки, што го прави лесно да се вклучат во апликацијата.

Една голема предност на Spring е неговата поддршка за инверзија на контрола (IoC) и зависноста на инјектирање (DI), што овозможува подобра модуларност и тестабилност на кодот. И сите интеграции со библиотеки се овозможени од страна на развиваите на Spring, доколку е потребно да се користи некоја технологија или библиотека, шансата да има веќе интеграција во Spring е многу голема.

Spring е алатка која го модернизира развивањето на апликации во Java екосистемот, во помодерно време Spring нуди доста добра интеграција и со Kotlin^[2], дозволувајќи пристап до Java екосистемот со помодерен јазик како што е Kotlin. Поради сите овие причини, Spring останува еден од најпопуларните избори за развој на апликации во Java и Kotlin.

2.4 Svelte+kit



Svelte^[3] е современ фронтенд фрејмворк кој се разликува од традиционалните фрејмворци како React или Angular по тоа што Svelte компајлира апликацискиот код во оптимизиран JavaScript код за време на изградбата, наместо да користи виртуелен DOM за ажурирање на корисничкиот интерфејс во реално време. Овој пристап резултира со побрзи перформанси и помалку трошоци за извршување.

SvelteKit^[4] е рамка изградена врз основа на Svelte која обезбедува целосен сет на алатки за развој на веб апликации. SvelteKit нуди функционалности како што се серверско рендерирање (SSR), статичко генерирање на страници (SSG), и лесна интеграција со различни бекенд технологии. SvelteKit има поддршка за повеќе страници, додека Svelte е повеќе наменет за изработка на еднострани апликации (SPA), SvelteKit испраќа на корисниците компајлиран HTML и CSS, а JavaScript за одредени делови кои не може да се познати при компајлирање.

За хостирање SvelteKit нуди исто така хостирање на еднострана апликација, но нуди и хостирање на серверски рендерирана апликација (SSR) што овозможува по-

добра пребарувачка оптимизација (SEO) и побрзо време на вчитување на страниците. SvelteKit користи рутирање базирано на датотеки и папки, што значи дека структурата на датотеките во проектот директно одговара на URL патеките на апликацијата. Ова го прави лесно за развишите да креираат и управуваат со различни страници и руте во апликацијата, покрај `.svelte` датотеки, има поддршка за `.server.ts` датотеки кои дозволуваат на дефинирање на апликациски интерфејс и служат како замена за контролери во Spring.

За хостирање на апликацијата потребно е да се има посебно распоредување само за корисничкиот интерфејс, ова најчесто значи дека има уште еден посветен Node.js сервер кој ќе се грижи за хостирање на корисничкиот интерфејс. При тоа треба доста да се внимава на тоа кој дел се извршува на страна на клиентот, а кој дел би се извршил на страна на серверот, како еден недостаток ова знае да биде доста комплицирано за нови корисници, и не е секогаш јасно кој дел од кодот е извршен на страна на серверот.

Голема предност е самата синтакса на Svelte, која е многу поедноставна и поекспресивна во споредба со други кориснички рамки. Svelte користи декларативен пристап за дефинирање на корисничкиот интерфејс, кодот се сместува во една датотека која содржи HTML, CSS и JavaScript, што го прави полесно за читање и одржување на кодот и тоа претставува една компонента, Svelte исто така овозможува полесно чување на состојбата на апликацијата, и овие споделени делови може да се стават во посебни датотеки кои се викаат складови (store), со овој пристап голема предност е тоа што за разлика од React нема потреба од заеднички елемент во рамки на дрвото на компоненти за да овозможи пристап до иста состојба.

2.5 Kubernetes



Kubernetes^[5] претставува платформа со отворен код за оркестрација и управување со контејнеризирани апликации. Тој обезбедува автоматизација на процесите на распоредување, скалирање и управување со апликации во контејнерски средини. Kubernetes овозможува на развишите и администраторите да ги менарираат своите апликации на ефикасен начин, без разлика дали се работи за мали или големи инфраструктури.

Во Kubernetes една апликација се состои од повеќе компоненти, како што се капсули (pods), сервиси (services) и распоредувања (deployments). Капсулите се најмалата единица на распоредување во Kubernetes и поддржуваат извршување на еден или повеќе контејнери, додека контејнерите претставуваат изолиран систем во кој се извршува изградена слика од апликацијата. За пристап до капсулите се користат сервиси кои овозможуваат стабилен пристап до нив, додека распоредувањата управуваат со капсулите, овозможувајќи автоматско скалирање и ажурирање на апликациите.

Една клучна функционалност зошто Kubernetes беше избран е изолацијата на апликациите преку Namespaces, што овозможува на различни апликации да се изолираат една од друга, со што се избегнуваат конфликти помеѓу клиенти. Исто така Kubernetes нуди автоматско скалирање на апликациите врз основа на оптоварувањето, што овозможува оптимално користење на ресурсите и подобри перформанси, и

сето ова може да се контролира со ресурси дифинирани со користење yaml јазикот. Покрај тоа Kubernetes обезбедува апликациски интерфејс за интеракција со платформата, со што се овозможува контрола преку главната апликација.

За интеракција со Kubernetes кластерот беше користена библиотеката **java-kubernetes-client**^[6], која овозможува лесна интеграција со Kotlin и Java апликации. Со оваа библиотека можи да се креираат, бришат и менацираат сите ресурси во рамки на кластерот, овозможувајќи на главната апликација да комуницира со Kubernetes кластерот и да ги менацира апликациите.

Дополнителни функционалности кои ги нуди Kubernetes се исто така хостирање на помошните сервиси за хостирање на апликациите, и вмрежување на апликациите преку вградени алатки како што се Ingress контролерите, кои овозможуваат управување со влезниот сообраќај кон апликациите.

Со користење на Kubernetes како платформа за хостирање на апликации, дозволува апликацијата да е директно врзана со платформата наместо со специфичен сервер или инфраструктура, што овозможува поголема флексибилност и скалабилност при хостирање на апликации во облак.

2.6 PostgreSQL



PostgreSQL^[7] претставува објектно-релативски систем за управување со база на податоци (ORDBMS) со отворен код, кој е познат по својата стабилност, скалабилност и големиот број на функционалности што ги нуди. PostgreSQL е дизајниран да поддржува сложени податочни структури и операции, што го прави погоден за широк спектар на апликации, од мали веб апликации до големи корпоративни системи, доколку функционалноста не е достапна директно во системот, постојат додатоци кои го нудат тоа.

PostgreSQL поддржува различни типови на податоци, вклучувајќи стандардни типови како што се цели броеви, текст и датуми, како и напредни типови како што се JSON, XML и геопросторни податоци. Ова овозможува на развивачите да ги моделираат своите податоци на начин кој најдобро одговара на потребите на апликацијата.

Апликацијата немаше потреба од комплицирани податочни типови, но беше избран PostgreSQL поради големата заедницата позади него и стабилноста што ја нуди, како и лесната интеграција со Spring преку Spring Data JPA, што овозможува лесно управување со базата на податоци и извршување на операции како што се креирање, читање, ажурирање и бришење на податоци.

За миграции се користи алатката Liquibase, која овозможува управување со верзии на базата на податоци и автоматско извршување на миграции при стартирање на апликацијата. Ова овозможува лесно одржување на структурата на базата на податоци помеѓу верзии на апликацијата и избегнување на конфликти при промени во

шемата на базата на податоци.

2.7 Keycloak



Keycloak^[8] претставува платформа со отворен код за управување со идентитети и пристап (Identity and Access Management - IAM), кој обезбедува централизирана автентикација и авторизација, овој сервис има поддршка за повеќе современи и стандардни протоколи за автентикација како што се OAuth 2.0, OpenID Connect и SAML 2.0, овозможувајќи полесна интеграција со различни технологии и рамки, како што се Spring Security, што овозможува лесно управување со корисничките сесии и пристапот до заштитени ресурси.

Keycloak нуди богати функционалности за управување со корисници, групи и улоги, овозможувајќи на администраторите да дефинираат различни нивоа на пристап и дозволи за корисниците. Дополнително.

Keycloak исто така нуди поддршка за повеќе надворешни провајдери на автентикација, како што се Google, facebook, потоа LDAP и Azure AD (Active Directory), овозможувајќи на корисниците да се најавуваат користејќи ги своите постоечки сметки од овие платформи. Keycloak поседува доста голема мрежа на поддржани интеграции, кога се развива решение кое бара автентикација и авторизација, но истовремено не е јасно кој систем го користат клиенти, Keycloak претставува одлично решение за оваа намена, бидејќи не е врзано со одреден еко систем и може да се интегрира со различни платформи.

Keycloak својата состојба ја чува во релативска база на податоци, за оваа намена беше користен PostgreSQL. Додека во базата на податоци на апликацијата се чува само референца кон корисничкиот идентификатор добиен од Keycloak, што овозможува поврзување на корисникот со неговиот именски простор и апликациите што ги поседува. Овој пристап ја зголемува безбедноста и го поедноставува управувањето со корисничките податоци, поради тоа што keycloak се грижи за безбедноста на корисничките податоци, ова не е грижа за време на развој на апликацијата.

2.8 Kaniko



Kaniko^[9] претставува алатка со отворен код наменета за градење на слики за контейнери директно во контејнеризирана околина, без потреба од инсталација Docker daemon или привилегиран пристап до системот. Ова го прави Kaniko особено погоден за употреба во Kubernetes околини, каде што безбедносните ограничувања и изолацијата на процесите се од големо значење.

Kaniko поддржува градење на слики со користење на Dockerfile, што овозможува на развишите да ги искористат своите постоечки знаења и алатки за градење на слики. Kaniko ги имплементира истиот формат и инструкции како Docker, но сепак има одредени разлики, шаблоните за сликите што ги одржува администратор се чуваат во базата и се практикат на оваа алатка.

Kaniko е интегриран користејќи kubernetes задачи (jobs), што овозможува на апликацијата да инициира процес на градење на слики во рамки на платформата. Ова ја прави алатката независна од апликацијата, доколку друга алатка има поддршка за Dockerfile инструкции може лесно да се замени со друга алтернативна алатка за градење на слики.

2.9 Docker registry v2

Docker registry v2^[10] претставува сервис со отворен код за складирање и дистрибуција на Docker слики. Овој регистар овозможува на корисниците да ги зачуваат своите контејнерски слики на едно место, сервисите што ја користат оваа алатка можат лесно да ги повлечат сликите кога им се потребни во поддржаниот формат.

Бидејќи Docker registry v2 ги поддржува сите стандарди за контејнери, овозможува лесна интеграција со различни алатки и платформи за контејнери, како што се kaniko, алатката за градење на слики, која исто така е потребно да ги зачува изградените слики, Kubernetes главната платформа, која треба да ги повлечи сликите за да ги изврши апликациите, за повлекување може да се користи containerd, cri-o, и сите тие се поддржани бидејќи го имплементираат OCI (Open Container Initiative) спецификациите.

За комуникација со овој сервис беше искористена Spring ЌебFlux^[11]библиотеката, која овозможува лесна интеграција со Spring апликации и овозможува асинхроно ражување со HTTP барања.

2.10 WebDAV



WebDAV (Web Distributed Authoring and Versioning)^[12] претставува протокол кој овозможува на корисниците да управуваат со датотеки на далечински сервер преку HTTP. WebDAV додава дополнителни методи на HTTP протоколот, овозможувајќи операции како што се креирање, бришење, преместување и менување на датотеки и папки на серверот. WebDAV претставува спецификација која кажува како изгледаат повиците и податоците за датотеките, но имплементацијата може да варира, и постојат различни сервери што го имплементираат овој протокол.

За интеракција со WebDAV серверот беше користена библиотеката *Sardine*, која овозможува лесна интеграција со Kotlin и Java апликации. Со оваа библиотека може да се креираат, бришат и менаџираат датотеки и папки на WebDAV серверот, овозможувајќи на главната апликација да ги зачува извршните датотеки кои се користат при градење на слики за контејнери.

WebDAV беше избран затоа што беше потребно да се чуваат изградените извршни датотеки на корисниците, за потоа да бидат достапни при градењето на слики за контејнери. Со користење на WebDAV, извршните датотеки се зачувуваат на едно место, притоа со самото тоа што WebDAV е стандардизиран протокол, овозможува лесна интеграција со различни сервери и платформи, веќе постојат библиотеки за оваа намена, и ова ја прави апликацијата независна од самиот сервер што складира датотеки.

2.11 WebSocket протоколот

За увид во логови и следење на апликациите на корисниците беше потребно да се овозможи комуникација во реално време, за оваа намена беше користен WebSocket протоколот, кој овозможува двонасочна комуникација помеѓу клиентот и серверот преку една постојана врска. За оваа намена беше користена библиотеката **spring-boot-starter-websocket**^[13], која овозможува лесна интеграција со Spring апликации.

Доколку се иницира конекција преку WebSocket протоколот, серверот може да испраќа пораки до клиентот во реално време, без потреба клиентот постојано да праќа барања за ажурирање. Еден голем предизвик е тоа што во секое време има можност да се затвори конекцијата, и доколку се случи такво нешто потребно е да се чуваат одредени логови во базата на податоци од моменталната состојба.

Логовите за градење на сликата се испраќаат од капсулата на задачата што ја гради сликата, и овие логови може да се читаат користејќи го апликацискиот интерфејс на kubernetes, овие логови се испраќаат преку WebSocket протоколот до корисникот во реално време, но доколку се иницира конекцијата за прв пат, потребно е да се прати цела состојба до тој момент, а потоа може да се праќаат само ажурирани логови, доколку градењето е завршено логовите се складираат во базата на податоци и при тоа не се користи WebSocket протоколот за оваа намена, туку само се земаат директно.

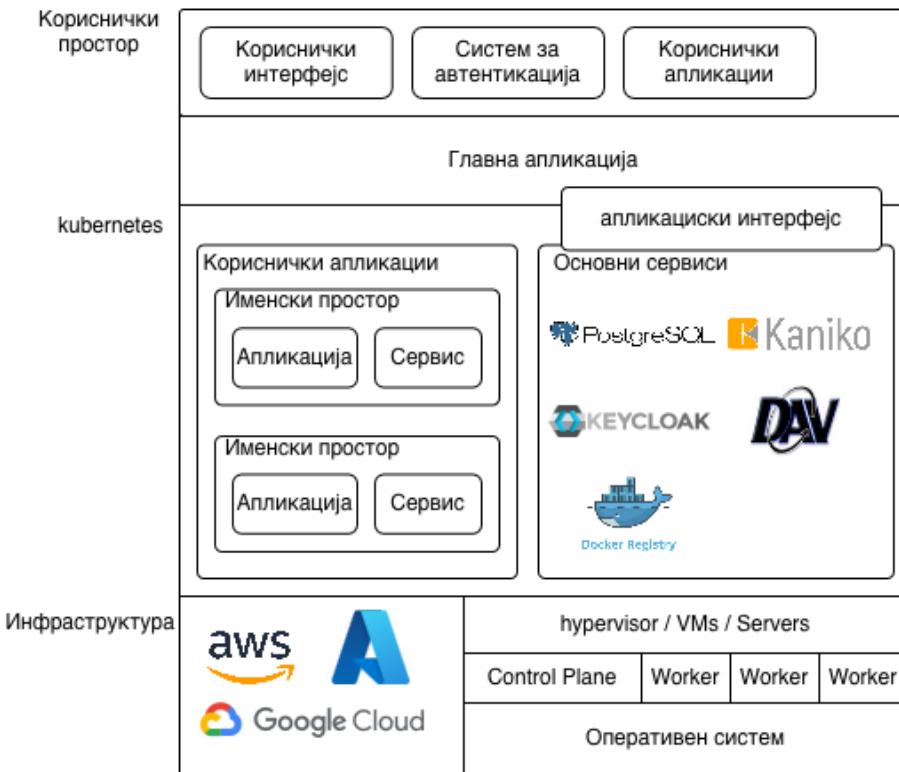
Статусот на апликациите се следи исто така преку апликацискиот интерфејс на kubernetes, и овие статуси се испраќаат преку WebSocket протоколот до корисникот во реално време, овие статуси се следат од страна на платформата, а платформата испраќа повици до апликацијата доколку се случи некој настан на страна на самата платформа, ова се случува само доколку барем еден клиентслуша за статусот на апликацијата.

3 Архитектура на системот

Техничкото решение се состои од претходно наведените технологии, апликацијата е организирана од повеќе различни сервиси кои се користат за да се достигни крајната цел на апликацијата, а тоа е да им овозможи на корисници да ги хостираат брзо и едноставно своите апликации.

Апликацијата е главно базирана врз основата на Kubernetes платформата, и врз основа на тоа се изградени структурата на базата на податоци, тука е вклучена комуникацијата со самите сервиси што се користат во решението.

Како што може да се види на слика 1 решението е поделено на неколку слоеви, првиот слој е слојот до кој корисникот има пристап и има директна интеракција, односно ова би било слојот кој се занимава со мрежата, еден слој подолу се наоѓа главната апликација која исто така може да е дел од Kubernetes слојот, апликацијата комуницира со Kubernetes слојот, каде што се наоѓаат помажните сервиси кои го овозможуваат градењето, хостирањето и прегледот на корисничките апликации, самите кориснички апликации немаат никаква поврзаност со главната апликација, и се менацираат од страна на Kubernetes платформата, потоа најдолниот слој претставува каде Kubernetes платформата е хостирана, ова може да биде локален сервер,



Слика 1: Слоеви на инфраструктурата

cloud платформа, виртуелни машини.

3.1 Сервиси

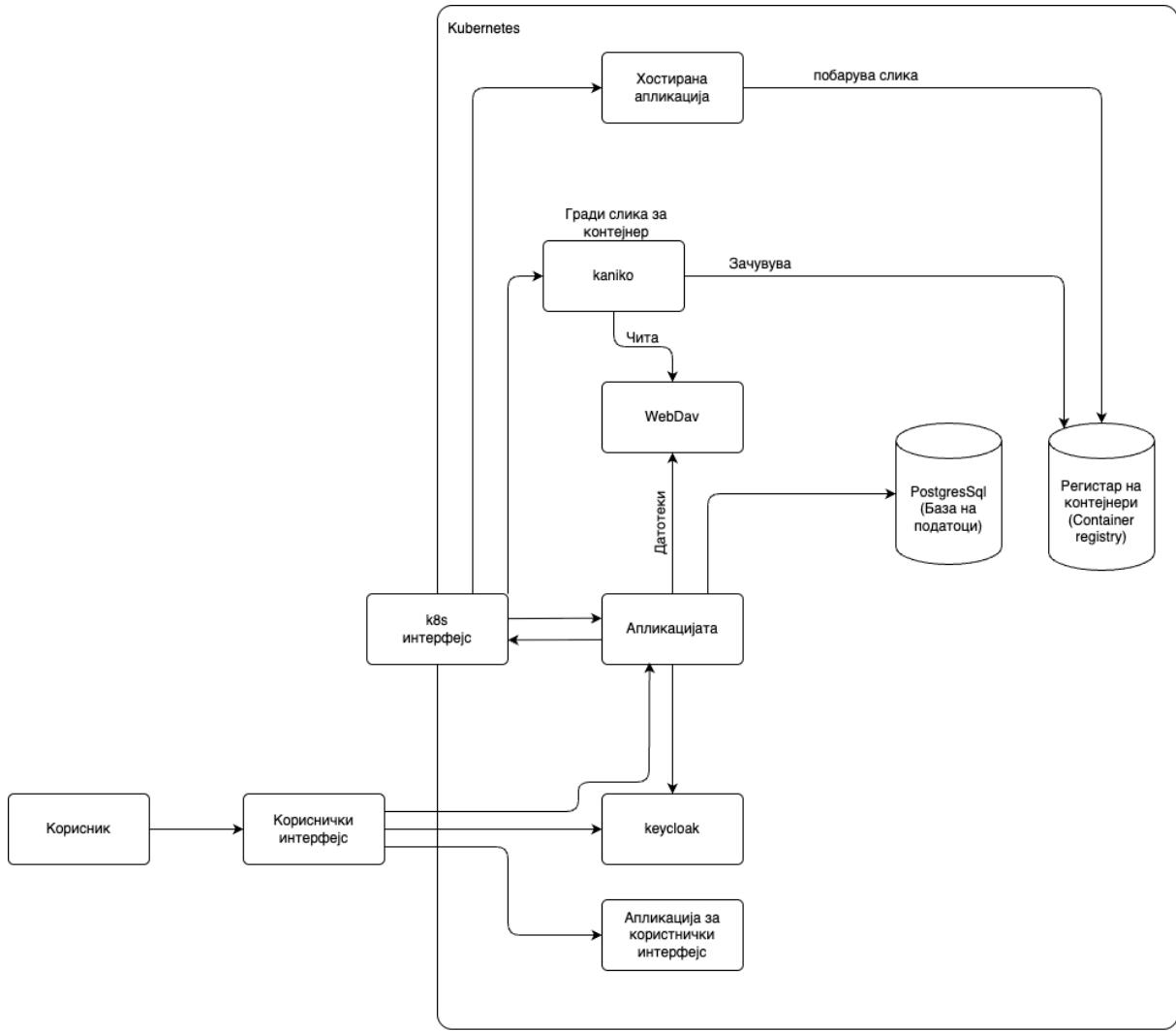
Вклучено во сервисите кои се користат се:

- Kubernetes
- Container registry (Docker)
- Keycloak
- WebDAV
- Nginx Ingress controller
- PostgreSQL
- kaniko

Овие технологии овозможуваат на решението да чува податоци, да менаџира со самата платформа и да ја користи за хостирање на други апликации, самата платформа дозволува комуникација помеѓу овие сервиси, исто така ни овозможуваат поголема безбедност при што тукамо имаме keycloak кој нуди автентикација.

3.2 Преглед на системот

На слика 2. јасно може да се види комуникацијата помеѓу овие сервиси, прво се започнува со корисникот, кој ја пристапува страницата за хостирање и го добива корисничкиот интерфејс.



Слика 2: Преглед на сервисите

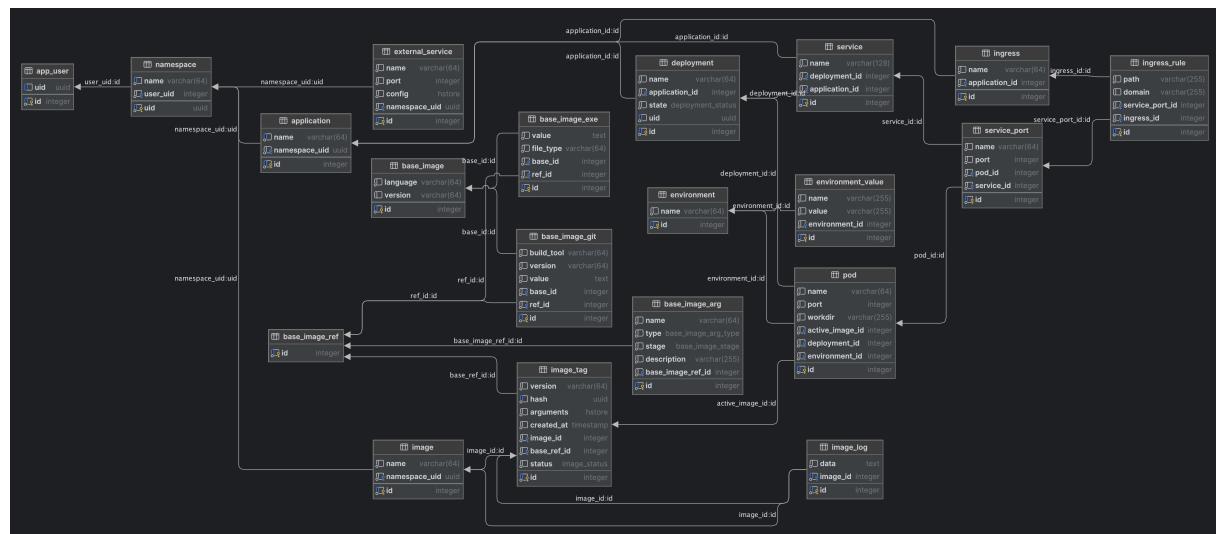
Потоа корисничкиот интерфејс го пренасочува на страница каде се бара негова автентикација преку користење на keycloak сервисот, откако корисникот ќе се најави, добива пристап до апликацијата, и може да ги види само своите податоци, во позадина се користи keycloak со OpenID и OAuth 2.0 протоколот, при што апликацијата го чита токенот и знае кој клиент е најавен и им овозможува контрола на корисниците врз своите апликации.

Откако корисникот ќе се најави, има пристап до останатите сервиси, при што има можност да ја пристапи главната апликација и да хостира свои апликации, при што корисникот има избор од 2 начини, еден од нив е хостирање од со обезбедување на извршна датотека, при овој користење на овој тек на апликацијата се користи WebDAV сервисот кој е хостиран во околната за да ги зачува овие датотеки, а доколку се оди со вториот начин, а тоа е со користење на git репо, главната апликацијата прави директно повик до околната во која е хостирана и се започнува со закажување на нова задача која ја гради сликата за контејнерот со користење на kaniko, оваа алатка добива шаблон како да ја изгради сликата, во првиот случај со извршната датотека ја презема датотеката од WebDAV сервисот, ја гради сликата и ја зачувува

во регистарот за слики кој исто така е хостиран во околната, во вториот случај без извршна датотека според дадениот шаблон ја изградува извршната датотека и ја гради сликата и повторно ја складира во регистарот за слики.

Во периодот на градење на сликите апликацијата има можност да ги следи логовите додека се гради апликацијата и да ги прикажи на корисникот користејќи WebSocket протоколот, кога ќе се заврши со градење на корисникот му се прикажуваат сите изградени слики на корисничкиот интерфејс.

Во следниот чекор корисникот има можност да креира распоред на апликацијата која подоцна се хостира на самата платформа за ова корисникот ја одбира сликата и нејзината верзија која сака да се хостира и апликацијата моменталната состојба ја зачува во база на податоци, во овој случај за таа цел се користи PostgreSQL, откако се зачува состојбата апликацијата продолжува со креирање на ресурсите на самата Kubernetes платформа, при тоа се овозможува следење на логови и статусот на апликацијата.



Слика 3: Релациски дијаграм на ентитетите

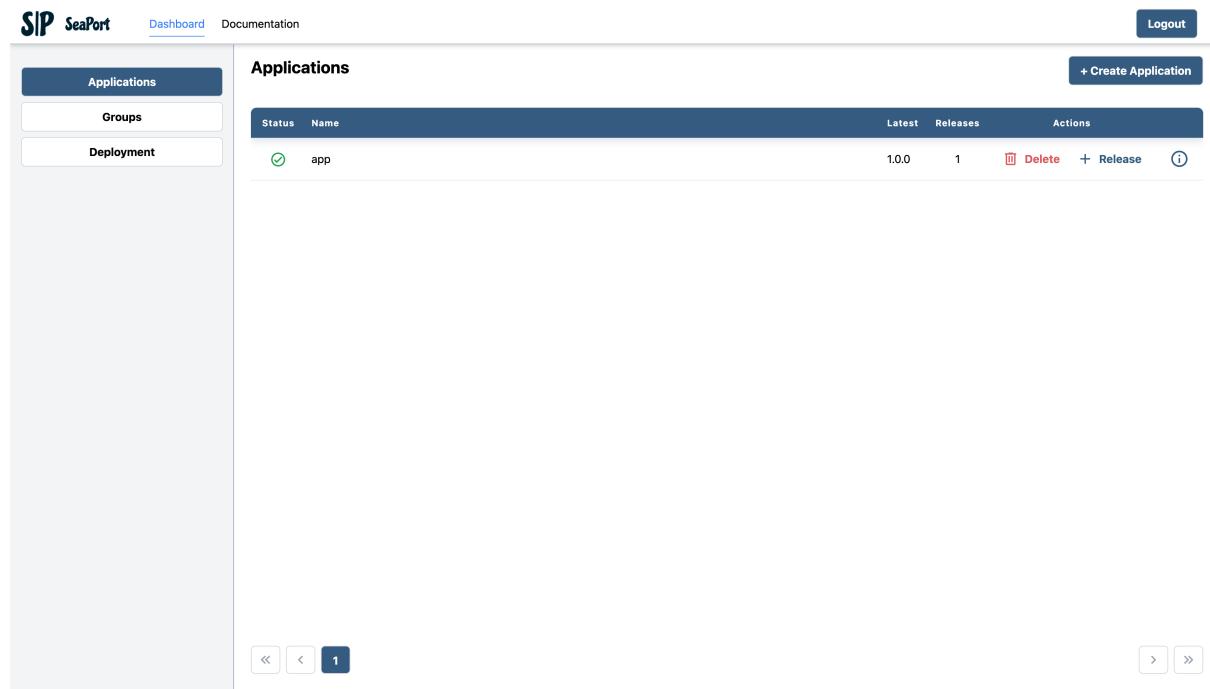
Како што може да се види на слика 3. Апликациите на еден корисник спаѓаат под еден така наречен именски простор (namespace), секој корисник со најава добива еден именски простор, и сите апликации спаѓаат под негова контрола, корисник теоретски може да има повеќе именски простори, но решението креира и работи со само еден, при што не дозволува креирање на повеќе, и секој корисник кој се најавува на системот автоматски добива еден именски простор за сите апликации, доколку корисник сака да ги поврзи своите апликации добива линк од платформата за интерна комуникација, за тоа се користат сервиси од Kubernetes алатката.

Покрај апликациите и конфигурациите на корисниците тука исто така спаѓаат и основните слики, кои се користат како шаблони за креирање на кориснички апликации, при тоа овој дизајн му овозможува на администратори на апликацијата поголема флексибилност, со што се дава можност во иднина да се имплементираат останати технологии покрај Java, без прилагодување на самото решение, покрај нови технологии ова му овозможува на администратор да додади и останати готови алтернативни сервиси, како што се бази на податоци, брокери за пораки, и останати

помошни сервиси, при тоа со развивање на овие дополнителни основни слики решението е доволно флексибилно да им овозможува на корисници потполна мок да развијат и хостираат било каква апликација.

Именскиот простор е врзан со сервисот за автентикација, каде што за секој корисник се креира еден именски простор, поради тоа што сервисот за автентикација е екстерен и има своја база на податоци, решението знае само дали корисникот е автентициран и добива контекст за неговиот идентитет, користејќи го ова се креира ентитет во базата на податоци во кој се чува именски простор поврзан со секој корисник, ова го поврзува корисникот со сите негови апликации.

4 Кориснички интерфејс



The screenshot shows the SeaPort application management interface. At the top, there is a navigation bar with the SeaPort logo, 'Dashboard', 'Documentation', and 'Logout' buttons. Below the navigation bar is a sidebar with three tabs: 'Applications' (selected), 'Groups', and 'Deployment'. The main area is titled 'Applications' and contains a table with one row. The table columns are 'Status', 'Name', 'Latest', 'Releases', and 'Actions'. The single row shows 'Status' as green (checked), 'Name' as 'app', 'Latest' as '1.0.0', 'Releases' as '1', and 'Actions' with buttons for 'Delete' (red), 'Release' (blue), and 'Info' (info icon). At the bottom of the main area are navigation buttons for page navigation.

Слика 4: Преглед на апликации

Корисничкиот интерфејс за корисниците се состои од една контролна табла каде секој клиент ги гледа своите апликации, сите апликации во овој простор припаѓаат на еден клиент, и секогаш има можност за самите да комуницираат помеѓу себе како сервиси.

Потоа корисничкиот интерфејс содржи и дел за документација, каде се опишуваат ограничувањата на системот при што клиентот кој ја користи алатката може да се прилагоди врз системот.

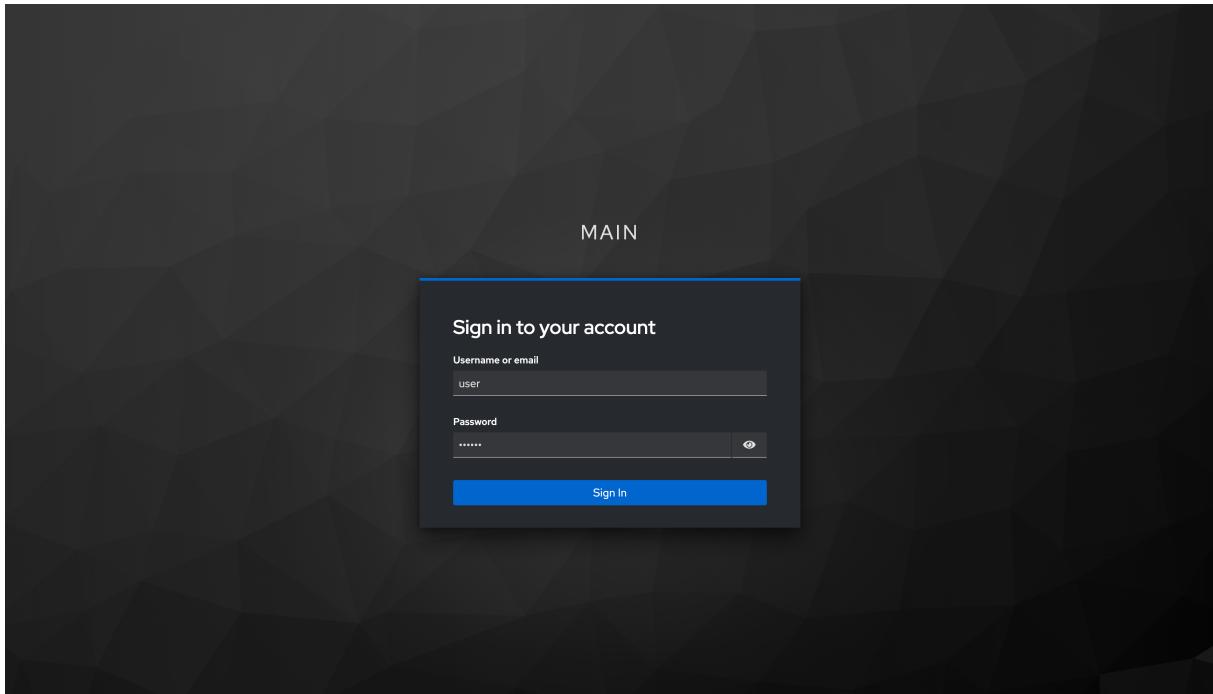
За крај исто така има и панел за администрација кој е достапен само до корисник со соодветна улога, тука администратор има пристап до можности за градење на контејнери кои се доста флексибилни и би му овозможиле на администратор да додади многу повеќе функционалности, неколку примери:

- додавање на поддршка за креирање на бази на податоци

- поддршка за други алатки како RabbitMQ, kafka, keycloak итн.
- Додавање на поддршка за автентицирани Git репозиториуми
- Додавање на поддршка за останати јазици или алатки за градење

Потоа сите овие апликации може да комуницираат со сите останати апликации, во рамки на Kubernetes, при што би се користеле сервиси за интерна комуникација.

4.1 Апликации и слики за контејнери



Слика 5: Автентикација со keycloak

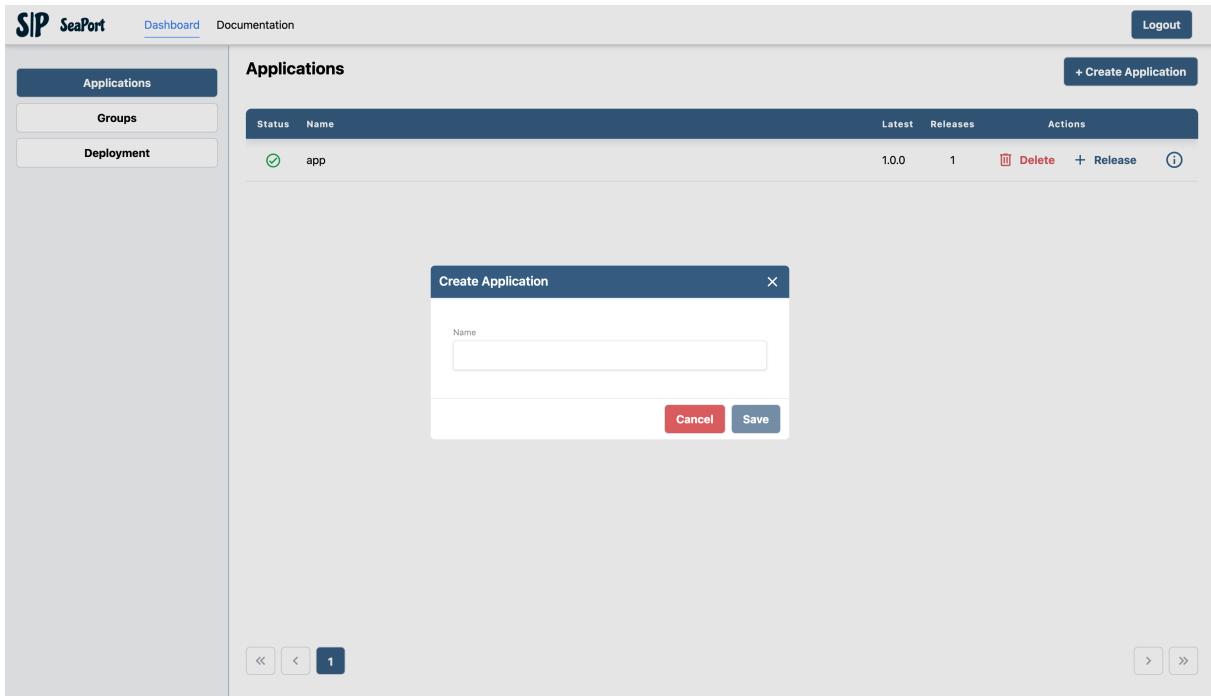
На корисничкиот интерфејс апликации се нарекуваат сликите за контејнерите, при што корисник прво создава слика, при што од него се бара само името на новата слика, потоа добива опција за да направи објава на нова верзија од истата, и има две опции:

- Прикачување на извршна датотека
- Градење на извршната датотека

Корисникот има можност да креира слика со специфицирање на сите полиња кои се под целосна контрола на администратор, администратор може да додади или одземи одредени полиња, целата состојба за ова се чува во базата на податоци, со избор на јазик и вид на слика корисникот има можност да добие сосема различни полиња и опции.

4.2 Прикачување на извршна датотека

Во овој случај од страна на корисникот се бара извршна датотека, при што е потребно самиот да ја изгради и да ја прикачи, исто така има дополнителни опции за тоа која верзија на одбраницот јазик да се користи.



Слика 6: Дијалогот за создавање на нова слика

Во овој случај решението е доста флексибилно, им дозволува на клиентите да користат која било алатка за да изградат извршна датотека, и го прави креирањето на слики доста едноставно.

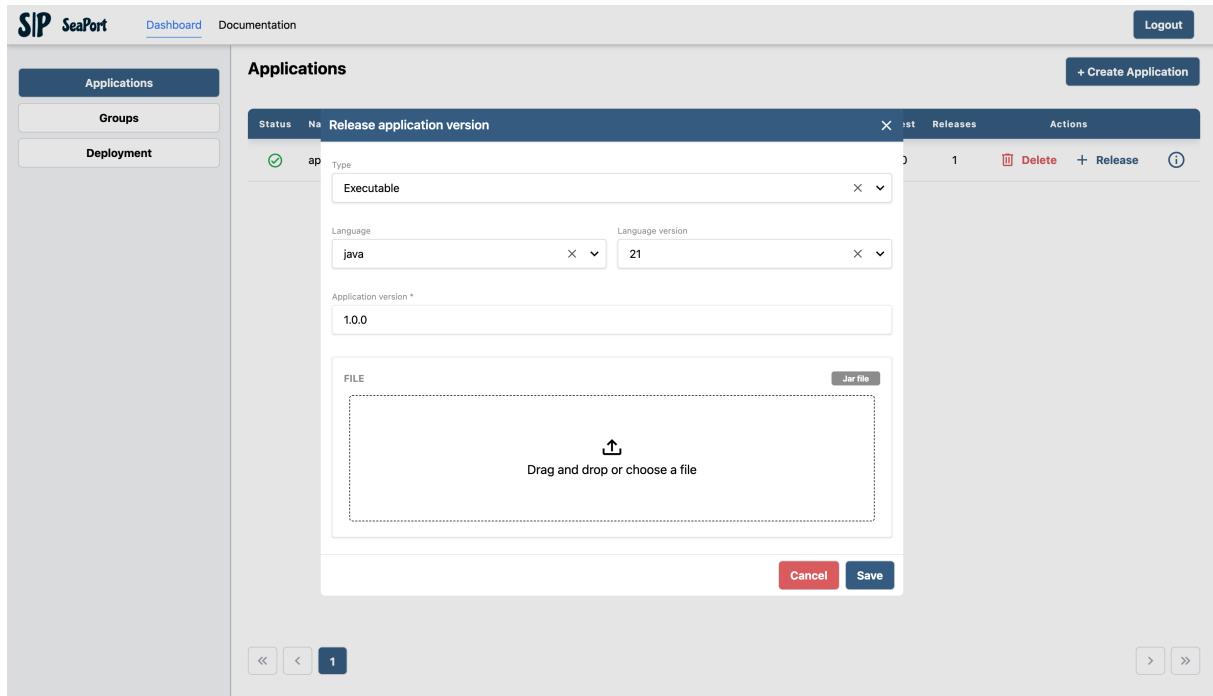
Во случајот на Java како што може да се види на слика 7, од корисникот се бара „*.jar*“ извршна датотека, овие полниа зависат од страна на администраторот, при што може во случај да се користи друг јазик да се бара друг вид на датотека, како што во .NET може да се побара „*.NET assembly*“ или само „*.zip*“ датотека, која подоцна би се искористила во самиот процес на градење.

4.3 Градење на извршната датотека

Решението им овозможува на корисници градење на извршната датотеката, при што за оваа цел се користи „**Git**“, но системот не е лимитиран на само оваа алатка, доколку администратор сака да додади повеќе алатки тоа апликацијата го овозможува,

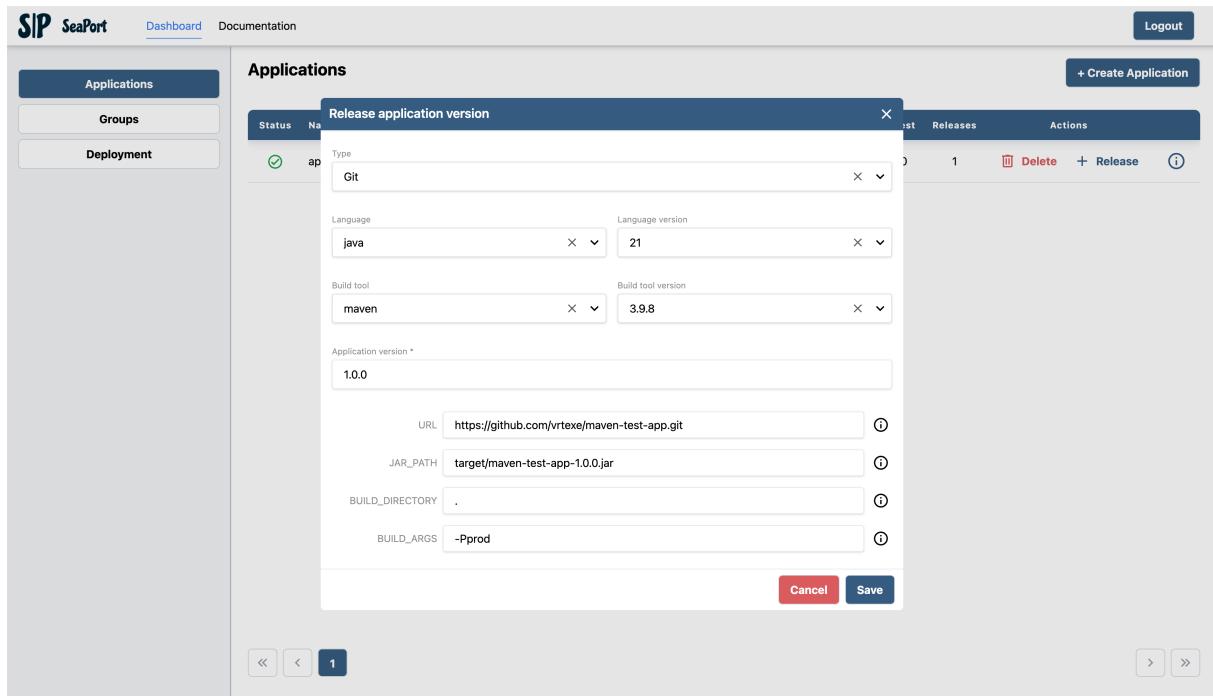
Во овој случај како што може да се виде на слика 8. од корисникот се бараат повеќе информации, како што се која алатка за градење ќе се користи, притоа исто така корисник има контрола врз некои претходно дефинирани вариабли кои се користат за градење на извршната датотека, како што може да се види на сликата, од корисникот се бара URL од Git репозиториумот, една лимитација на ова решение, односно на дефиницијата за градење на слика за контејнер е дека има поддршка само за јавни репозиториуми, но ова многу лесно може да се надмине, со додавање на дополнителни вариабли за автентификација, како на пример за токен.

Дополнително, од корисникот се бараат директориумот од каде да се започни со градење, ова се бара во случај да организацијата на проектот не е стандардна, или



Слика 7: Креирање на слика со прикачување на извршна датотека

во еден репозиториум се наоѓаат повеќе проекти, потоа се бара патот до ‘jar‘ фајлот кој на крај се користи за започнување на апликацијата и за крај им се дозволува на корисници да пратат дополнителни аргументи на алатката кои им се потребни за да го изградат својот проект или апликација.



Слика 8: Креирање на слика со градење на извршна датотека

4.4 Изданија на апликација

Сите изданија од апликацијата се прегледни на страната за информации за апликацијата, на слика 9. може да се види издание со верзија 1.0.0, при што ова издание е направено со прикачување на извршна датотека.

На оваа страница им се овозможува на корисници брзо издавање на апликација, со претходно наведените варијабли за некоја постара верзија, исто така и има преглед на претходно наведените варијабли.

The screenshot shows the SIP SeaPort application management interface. The top navigation bar includes 'Logout' and a 'New Release' button. On the left, a sidebar has 'Applications' selected, along with 'Groups' and 'Deployment'. The main content area is titled 'app' and shows a table of releases. The table columns are 'Status', 'Version', 'Released', 'Type', and 'Actions'. One row is highlighted in orange, representing a release with status 'OK', version '1.0.0', released on '16.07.2025 18:36', type 'Executable', and actions including '+ Release', 'Logs', and a help icon.

Слика 9: Изданија на апликација

Корисникот може во реално време да ги следи логовите од градењето на сликата, доколку дојди до проблем има увид во што точно се случило, доколку е проблемот од негова страна може да го поправи, како на пример грешно поставена вариабла, ова може да се види на слика.

Следењето на логови е овозможено со веб-сокети при што системот кога ќе добие веб-сокет повик се чува сесијата, и доколку има градење на датотека вкулечно за прикачува таму таа сесија, доколку нема ништо моментално се бараат логовите во базата на податоци, бидејќи тие откако ќе заврши градењето на сликата се губат тие логови и е потребно да се зачуваат, тие се чуваат во базата на податоци, а за логови се користи Kubernetes интерфејсот, при што се слуша на секој запис на лог и се препраќа подоцна во системот, каде се чува во меморија, а подоцна се праќа и на корисникот, кој може да го гледа во реално време.

Корисникот исто така може да го следи самиот статус на градење на сликата, додека сликата се гради наместо зелената ознака се прикажува на корисник знак за вчитување, доколку дојди до грешка на корисникот им се прикажува знак за грешка.

```

48 INFO [0023] Taking snapshot of files...
49 INFO [0023] RUN jlink --add-modules ALL-MODULE-PATH --strip-debug --no-man-pages --no-header-files --output jre
50 INFO [0023] Initializing snapshotter ...
51 INFO [0023] Taking snapshot of full filesystem...
52 INFO [0024] Cmd: /bin/sh
53 INFO [0024] Args: [-c jlink --add-modules ALL-MODULE-PATH --strip-debug --no-man-pages --no-header-files --output jre]
54 INFO [0024] Running: [/bin/sh -c jlink --add-modules ALL-MODULE-PATH --strip-debug --no-man-pages --no-header-files --out
55 WARNING: Using incubator modules: jdk.incubator.vector
56 INFO [0032] Taking snapshot of full filesystem...
57 INFO [0034] Saving file build/jre for later use
58 INFO [0035] Deleting filesystem...
59 INFO [0035] Retrieving image manifest alpine:3.20.2
60 INFO [0035] Returning cached image manifest
61 INFO [0035] Executing 0 build triggers
62 INFO [0035] Building stage 'alpine:3.20.2' [idx: '2', base-idx: '-1']
63 INFO [0035] Unpacking rootfs as cmd COPY --from=jrebuilder /build/jre jre requires it.
64 INFO [0036] COPY --from=jrebuilder /build/jre jre
65 INFO [0036] Taking snapshot of files...
66 INFO [0038] COPY --from=downloader /download/app.jar .
67 INFO [0038] Taking snapshot of files...
68 INFO [0038] ENV JAVA_ARGS ""
69 INFO [0038] ENV JAR_ARGS ""
70 INFO [0038] ENV JAVA_HOME /jre
71 INFO [0038] ENV PATH $JAVA_HOME/bin:$PATH
72 INFO [0038] CMD java $JAVA_ARGS --jar $JAR_ARGS app.jar
73 INFO [0038] Pushing image to 10.96.101.236/app:1.0.0
74 INFO [0040] Pushed 10.96.101.236/app@sha256:83b4d11b7ca2f28336e47d3c62bdf8e0d59260071ee2fd973f327e5ff32fccda
75

```

Слика 10: Логови за изградба на слика за контејнер

4.5 Преглед на групи на хостирани апликации

Групите се користат за подобро организирање на апликациите, корисник мора пред да ја хостира некоја апликација да ја стави во група, групата се креира доста едноставно, се бара од корисникот само името на групата, доколку корисникот сака да избриши цела група на хостирани апликации тоа би ги избришало сите ресурси поврзани со таа група, прегледот е прикажан на слика 11.

Секој корисник има можност да поседува повеќе групи, овие групи главно се користат да се распределат логички апликациите едни од други, но не постои лимитација во интерната комуникацијата помеѓу повеќе апликации, доколку апликациите се наоѓаат во еден именски простор корисникот може да го користи директниот линк до сервисот за таа апликација, но доколку корисникот сака да оствари комуникација помеѓу апликации во различни сервиси, или да комуницира со друга апликацијата од во рамки на самата платформа, тоа можи да го направи со користење на целосното име (DNS) кое платформата го доделува на самите сервиси, ова е илустрирано со код 1.

```

1 #!/usr/bin/env bash
2
3 NAMESPACE="4C212F98-199F-47EE-BD60-7678B990652A"
4 SERVICE_NAME="app-svc"
5 SERVICE_URL="http://$SERVICE_NAME.$NAMESPACE.svc.cluster.local"
6
7 curl $SERVICE_URL

```

Код 1: Повик на сервис во различен именски простор

Слика 11: Преглед на групи

4.6 Преглед на распоредувања на апликации

Едно распоредување (deployment) на апликација значи дека таа апликација е хостирана, прегледот на хостираните апликации е претставен на слика 12, на овој преглед на корисниците им се овозможува да управуваат со својата апликација.

Опциите кои им се нудат на корисниците се следните:

1. Запирање и започнување на апликацијата.
2. Преглед на логовите во апликацијата во реално време.
3. Целосно briшење на хостираната апликација
4. Приспособување на конфигурацијата за хостирање на самата апликација (промена на верзија, патека од линкот, конфигурациски опции, итн...)
5. Преглед на интерниот линк до апликацијата во самиот кластер

Најчеста промена што би била очекувана од страна на корисници е правење на надградби на апликацијата, тоа би значело промена на верзијата и конфигурацијата, при тоа треба да се има во предвид дека хостираната апликација при промена на конфигурација се запира и се креира нова апликација.

За безбедност и тестирање корисникот може во било кое време многу лесно да хостира две верзии од истата апликација, но доколку апликацијата побарува повеќе сервиси потребно би било сите сервиси да се хостирали по втор пат, или да се користат истите сервиси од двете апликации, во случај да апликациите имаат многу зависности ова ја зголемува комплексноста за хостирање, откако корисникот заврши со тестирање има можност да го промени линкот од старата верзија, при што би се тргнала од употреба и новата верзија на апликацијата да го добие стариот линк со

што би се заменила оваа апликација со новата верзија, и старата верзија може да се избриши.

Status	Name	Group	Application	Version	Service	External	Actions
○	app-deployment	g2	app	1.0.0	http://app-service	🔗	⟳ ⚙️ 📁 🗑️

Слика 12: Преглед на распоредувања

Корисникот во секој момент има можност да ги гледа логовите од својата апликација, доколку платформата успешно ги креира ресурсите и апликацијата е започната логовите се видливи од страна на корисникот, за оваа цел не се зачувува состојбата на логовите во базата на податоци и секогаш се земаат во реално време.

Секогаш кога апликацијата добие нови логови со користејќи отворена конекција преку WebSocket протоколот корисничкиот интерфејс чека од серверот одговор и добива нов лог од самата платформа.

Ова им овозможува на корисниците евиденција во своите апликации, доколку настанат грешки корисниците можат да ги откријат со користење на логови.

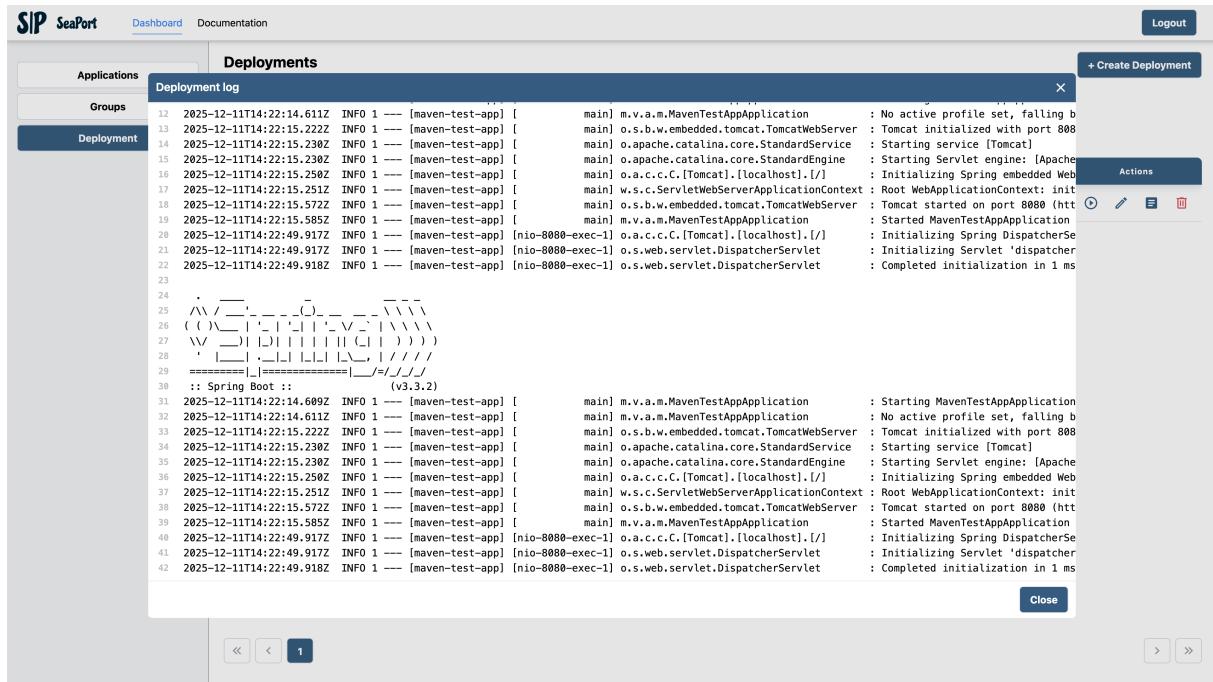
Преглед од логовите од една апликација се прикажани на слика 13.

Некогаш тоа не е доволно за дебагирање на самата апликација, но тоа е една лимитација од користењето на оваа платформа, главната цел би била едноставност за хостирање, со тоа се жртвува прегледноста за грешки во апликациите кои се хостираат.

4.7 Креирање на распоредувања за хостирање на апликации

Прегледот за креирање на распоредувања за хостирање на апликации е претставен на слика 14, на оваа слика се прикажани полинјата кои се потребни за Kubernetes платформата да се хостира една апликација.

Од страна на корисникот се бара претходно да има креирано група во која ќе биде сместено ова хостирање, и да има обезбедено изградена верзија од апликацијата



Слика 13: Преглед на логови од хостирана апликација

која сака да ја хостира, потоа се бара од нивна страна поставки за самата платформа, односно како ќе се вика самиот сервис, од ова зависи интерната патека до тој сервис.

Корисникот има можност да одбери својата хостирана апликацијата да биде изложена на надворешниот свет, при тоа за оваа цел како дел од изложената патека се користи името на именскиот простор за тој корисник, а останатиот дел од патеката го одредува корисникот, оваа лимитација е воведена во случај на два корисници да одберат именски простор или патека што се поклопува, доколку се случи ова би дошло до грешка од страна на платформата, за да се спречи ова се генерира еден именски простор за секој корисник, и е дел од самата патека.

За крај корисникот има можност да специфицира конфигурациски полиња за својата околина, овие конфигурации подоцна се испраќаат до апликацијата, каде апликацијата може да одреди како ќе ги прочита и користи, за оваа цел за полесно организирање на овие конфигурациски полиња постои и дијалог со подобар преглед за прилагодување на овие полиња, тоа е прикажано на слика 15.

4.8 Основни шаблони за градење на апликации

На слика 16. е прикажан преглед на основните шаблони за градење на апликации, овие шаблони се користат за да се олесни градењето на апликации, при што корисникот има можност да избере од некој од овие шаблони и да го користи за градење на својата апликација.

Администратор има целосна контрола врз овие шаблони, при тоа една од лимитациите е тоа што е потребно да се направи шаблон за секоја верзија на програмскиот јазик или алатката за градење на апликации, исто така потребно е за секоја алатка да се прави посебен шаблон.

Screenshot of the SIP SeaPort interface showing the 'Create Deployment' form. The 'Deployment' tab is selected in the sidebar. The main form shows fields for Group (g2), Application name (app), Application version (1.0.0), Name (app-deployment), Port (8080), Service (app-service), External checkbox (checked), Name (app-ingress), Path (/4e414cd2-3405-4a11-98de-283548459e51/app), and Environment Properties (NAME: testing, PROP: /v1/test).

Слика 14: Преглед за креирање на распоредувања за хостирање

Screenshot of the SIP SeaPort interface showing the 'Create Deployment' form with an open 'Environment' modal. The modal lists environment properties: NAME: testing and PROP: /v1/test. The main deployment form shows the same configuration as in Screenshot 14.

Слика 15: Преглед за прилагодување на конфигурациски полиња

Но главна предност е тоа што администратор може да додава шаблони за технологии за кои иницијално ова решение не е прилагодено.

Корисник одбира еден шаблон од овие за да ја гради својата апликација, за оваа цел се користи алатка наречена kaniko, која е специјално направена за градење на слики за контејнери, оваа алатка лесно се интегрирана во Kubernetes платформата и овозможува градење на слики без потреба од Docker инсталација, или посебна алатка

за оваа цел.

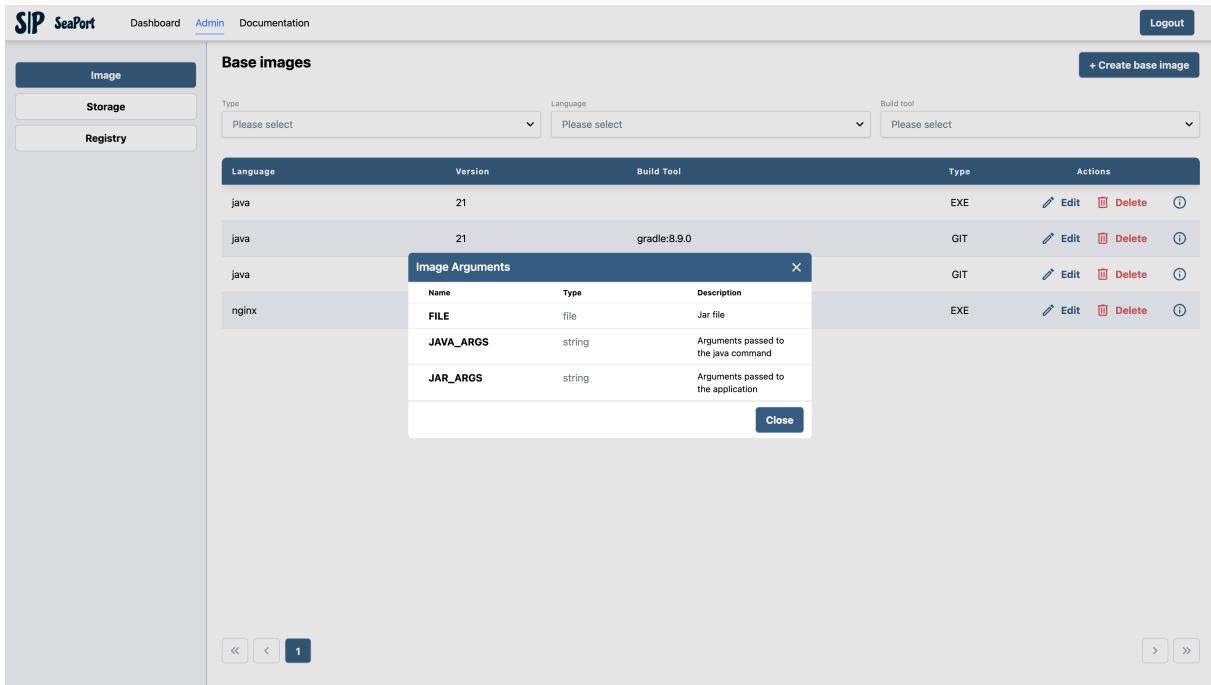
Language	Version	Build Tool	Type	Actions
java	21		EXE	
java	21	gradle:8.9.0	GIT	
java	21	maven:3.9.8	GIT	
nginx	1.27.1		EXE	

Слика 16: Преглед на основните шаблони за градење на апликации

Администратор има можност да ги прилагоди потребните параметри за градење на апликацијата, овие се во целосна контрола на администратор и се претставуваат на корисникот како што се дефинирани, полинјата може да се видат на слика 17. и слика 18. Во слика 17. се прикажани полинјата потребни за градење на Java апликација со извршна датотека, каде администраторот им дава избор за прикачување на датотека на корисникот, додека на корисничка страна добива соодветно поле за оваа намена, а останатите се аргументите кои ќе бидат пратени кон самата апликација, ова служи за во случај на корисникот да му е потребно да ја прилагоди Java виртуелната машина (JVM), бидејќи тоа не би можел да го направи во наредниот чекор при хостирање.

На слика 18. се прикажани полинјата потребни за градење на Java апликација со користење на алатка за изградба на извршна датотека, оваа слика конкретно се однесува на Gradle алатката, но сликата е доста специфична и од корисникот се бара git репозиториум, но самиот шаблон не имплементира автентификација, ова би можело да се додади како дополнително функционалност од страна на администраторот, при што би се додале дополнителни полинја за оваа намена, потоа во зависност од имплементацијата на шаблонот, администраторот исто така им дозволува на корисниците да испратат аргументи при градење на апликација, со што се дозволува поголема флексибилност при градење на апликации, и исто како и претходниот случај се дава на корисникот можност да се пратат аргументи со кои се прилагодува однесувањето на Java виртуелната машина.

На слика 19. е прикажан прегледот за прилагодување на шаблонот, при тоа на оваа страница администраторот има можност да внесе јазик, доколку не постои може да внесе било каква вредност, откако се зачува новиот јазик или новата верзија, корисниците автоматски имаат пристап до новиот јазик или верзија внесена од страна на



Слика 17: Преглед на опциите потребни за градење на Java апликација со извршна датотека

администраторот.

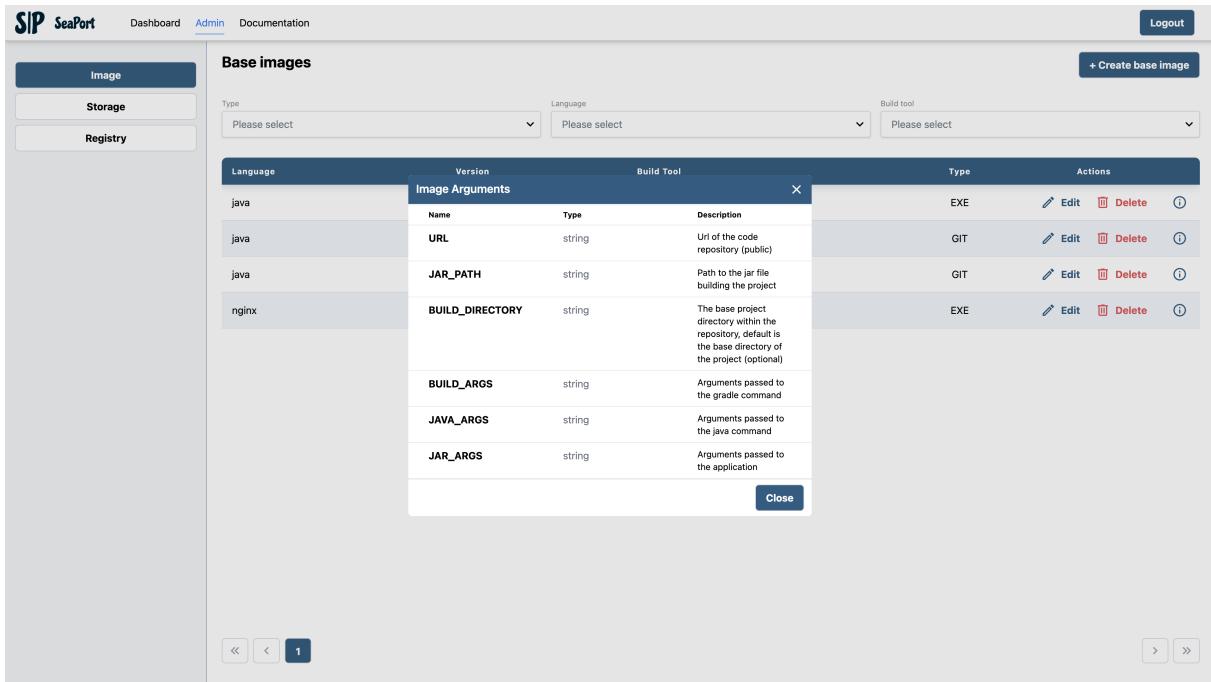
Администраторот има можност да додава и одзема полиња од шаблонот, при што сите овие промени се рефлектираат кај корисниците, кои при следното градење на апликација ќе ги добијат новите полиња, сите полиња кои се наведени од администратор може да со истото име да се користат во самиот шаблон за градење, како што е прикажано во код 2, овие вредности се праќаат директно до алатката за градење на апликации.

Во случај да администраторот направи промени, сите постоечки апликации кои веќе се изградени си остануваат да постојат, но секоја наредна верзија од истата апликација што би се градела од страна на корисникот ќе се користи новиот шаблон, со што корисниците потребно е да ги внесат променетите аргументи, заради оваа причина треба од страна на администраторот да се внимава при правење на промени во постоечки шаблони.

Прикажано со код 2 е една целина за градење на Java апликација со користење на Gradle алатката, овој шаблон е прилагоден да ги користи аргументите кои се наведени на слика 18. притоа може да се забележи дека градењето е поделено на четири делови:

1. Преземање на изворниот код од git репозиториумот.
2. Градење на извршната датотека со користење на Gradle.
3. Градење на минимална Java извршна околина (JRE) со користење на jlink.
4. Креирање на финална слика за контејнер со вметнување на извршната датотека и Java извршната околина.

Овие чекори се дефинирани од страна на администратор и не се задолжително,



Слика 18: Преглед на опциите потребни за градење на Java апликација со користење на алатка за изградба на извршна датотека

доколку администраторот одлучи да го промени шаблонот основните слики што се користат може да ги замени со други, исто така одлуката дали ќе се изгради Java извршна околина или би се користела Java развојната околина (JDK) е во целосна контрола на администратор.

```

1  # 1.
2  FROM alpine/git:2.45.2 as src
3
4  WORKDIR /data
5
6  ARG HASH
7  ARG URL
8
9  ENV HASH=${HASH}
10 RUN git clone $URL src
11
12 # 2.
13 FROM gradle:8.9.0-jdk21-alpine as builder
14
15 ARG JAR_PATH
16 ARG BASE_DIRECTORY=". "
17 ARG BUILD_ARGS=""
18
19 WORKDIR /data
20
21 COPY --from=src /data/src src
22
23 WORKDIR /data/src/$BASE_DIRECTORY
24
25 RUN gradle $BUILD_ARGS clean build
26 RUN mv $JAR_PATH /data/app.jar
27

```

```

28  # 3.
29  FROM eclipse-temurin:21-alpine as jrebuilder
30
31 WORKDIR build
32
33 RUN jlink \
34   --add-modules ALL-MODULE-PATH \
35   --strip-debug \
36   --no-man-pages \
37   --no-header-files \
38   --output jre
39
40  # 4.
41  FROM alpine:3.20.2
42
43 COPY --from=jrebuilder /build/jre jre
44 COPY --from=builder /data/app.jar app.jar
45
46 ENV JAVA_ARGS ""
47 ENV JAR_ARGS ""
48
49 ENV JAVA_HOME /jre
50 ENV PATH $JAVA_HOME/bin:$PATH
51
52 CMD java $JAVA_ARGS -jar $JAR_ARGS app.jar

```

Код 2: Шаблон за изградба на Java извршна датотека користејќи Gradle

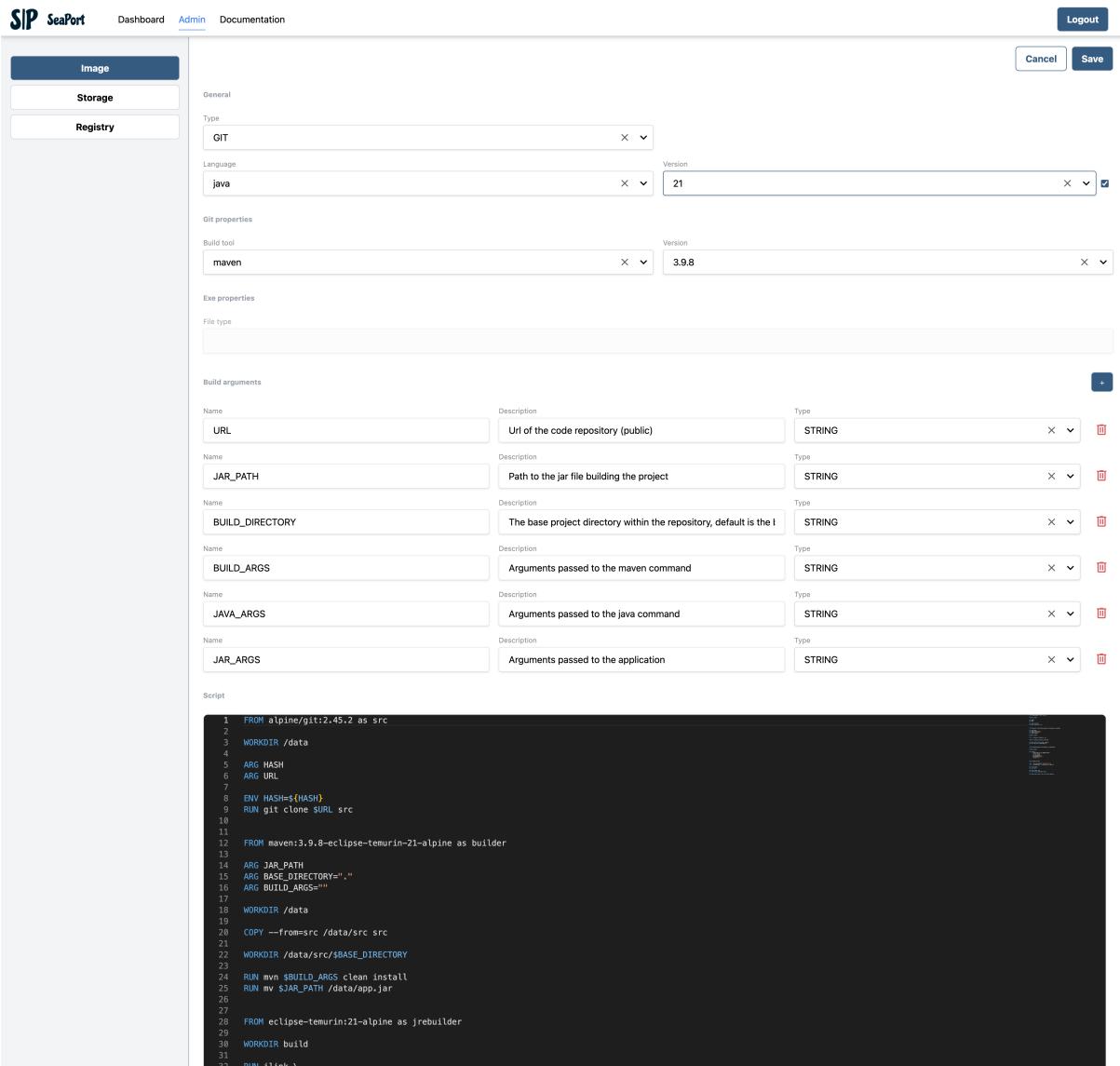
4.9 Администрација на извршни датотеки

Прегледот за администрација на извршни датотеки е прикажан на слика 20. Овој преглед е достапен само до администраторот на платформата, при што администраторот има можност да ги брише сите извршни датотеки кои се зачувани во системот, овие извршни датотеки се користат при градење на слики за контејнери, и откако една слика е изградена извршната датотека повеќе не е потребна, заради оваа причина администраторот има можност да ги брише овие извршни датотеки и да ослободи простор на дискот.

Исто така во позадина апликацијата секој ден извршува чистење на извршни датотеки кои се постари од 1 ден, со што се овозможува автоматско одржување на системот и ослободување на простор на дискот. Ова е возможно бидејќи секоја извршна датотека се користи само при градење на слика за контејнер, и потоа не е возможно да се искористи од страна на корисникот. Со што оваа датотека останува на системот, При тоа заради една употреба не би имало смисла да се чува подолго време.

Но оваа функционалност е овозможена во случај да некој корисник постави извршна датотека која е препоголема и прави проблеми, администраторот веднаш би можел да ја избрише таа извршна датотека и да ослободи простор на дискот.

Откако се искористи извршната датотека за градење на сликата потоа е невозможно истата да се види од кој корисник доаѓа, но за таа цел се користи патеката на извршната датотека, при секое зачувување на извршна датотека истата се зачува во папка која го содржи уникатниот идентификатор на корисникот, при тоа полето за пребарување дозволува пребарување по патеката на извршната датотека,



Слика 19: Преглед за прилагодување на шаблонот за градење на апликација

што овозможува администраторот да ја најде извршната датотека која сака да ја избрише според уникатниот идентификатор или името на апликацијата.

Голем предизвик тута беше што датотеките се менацираат од страна на алатката која се користи за чување на датотеки, а тоа е WebDAV серверот, при што не постои директен пристап до датотеките, туку се користи WebDAV клиент^[14] за оваа намена, што го отежнува процесот на менацирање на овие извршни датотеки.

4.10 Администрација на регистарот за слики

На следниот преглед прикажан на слика 21. е прикажан прегледот за администрација на регистарот на слики, овој преглед е достапен само до администраторот на платформата, при што администраторот има можност да ги брише сите слики кои се зачувани во регистарот на слики, овие слики се користат при хостирање на апликации, но откако сликата се избриши поради какви било причини, таа слика

The screenshot shows the 'Storage' section of the SIP SeaPort Admin interface. On the left, there's a sidebar with 'Image', 'Storage' (which is selected and highlighted in blue), and 'Registry'. The main area is titled 'Stored resources' and contains a table with one entry:

Name	Path	Size (bytes)	Actions
app-1.0.0.jar	/day/4e414cd2-3405-4a11-98de-283548459e51/app/app-1.0.0.jar	20303685	Delete

Below the table are navigation buttons: '<<', '< >', and '>>'.

Слика 20: Преглед за администрација на извршни датотеки

повеќе не е достапна за користење, и доколку дојде до рестарт на апликациите што ја хостираат оваа слика апликациите повеќе нема да можат да се стартираат. Но ова би било видливо и од страна на корисникот, при што од негова страна нема да се појави избришаната слика во прегледот на изградените слики 9.

The screenshot shows the 'Registry' section of the SIP SeaPort Admin interface. On the left, there's a sidebar with 'Image', 'Storage', and 'Registry' (which is selected and highlighted in blue). The main area is titled 'Registry' and contains a table with one entry:

Name	User	App	Namespace	Latest	Releases	Actions
app	f7aab97a-730e-42a8-96ee-03eaf973b17			1.0.0	1	Delete Details

Below the table are navigation buttons: '<<', '< >', and '>>'.

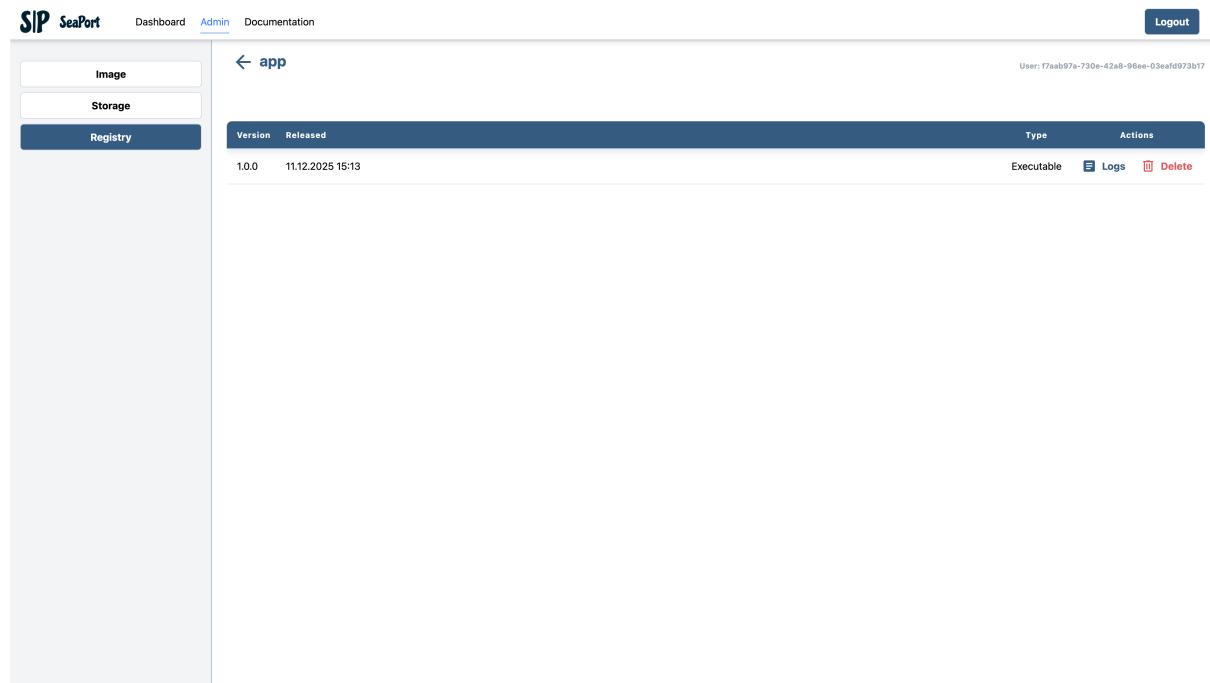
Слика 21: Преглед за администрација на регистарот на слики

На овој преглед администраторот може да ги види апликациите или сликите на

сите корисници на апликацијата, и тука се добива исто така, доколку е достапно и името на корисникот кој ја има креирано оваа слика, ова е прикажано на слика 22 и слика 21.

За овој преглед се користат истите податоци што се претставени за апликацијата или сликите на корисникот прикажани на слика 9, при што администраторот има можност да ги брише сите слики, без разлика кој корисник ја има креирано истата, но обичен корисник има можност да ја избрижи единствено својата слика.

При бришење на слика од регистарот на слики, главна цел не беше да се избриши сликата само од базата на податоци, туку и да се избриши сликата од страна на регистарот, за оваа цел се користи интерфејсот на искористениот регистар за слики, притоа комуникацијата се одвиваше со користење на Spring WebFlux клиент^[11].



Version	Released	Type	Actions
1.0.0	11.12.2025 15:13	Executable	Logs Delete

Слика 22: Преглед за администрација на регистарот на слики

4.11 Преглед за документација

Прегледот за документација е прикажан на слика 23. Овој преглед е достапен за сите корисници на апликацијата, при што корисниците имаат можност да ја видат документацијата за користење на апликацијата, дополнително како да конфигурираат Java апликации, системите за градење на Java апликации, како и објаснување за тоа што се сервиси и како се поврзуваат повеќе апликации.

4.12 Позадина на апликацијата

Апликацијата во позадина исто така нуди апликациски интерфејс (API) кој овозможува целосно управување со сите функционалности кои се достапни преку корисничкиот интерфејс, но нуди и поддршка за покомплицирани сценарија, како што се креирање на повеќе подови во рамки на едно распоредување, или користење на посебни конфигурациски датотеки за самата апликација.

The screenshot shows the Seaport Documentation page for Java Applications. The left sidebar includes links for Overview, Java Applications, Configuration, Services & Ingress, Examples, and FAQ. The main content area is titled "Seaport Hosting Documentation" and "Java Applications". It contains sections on Java apps (including Spring Boot) working best as executable JARs, typical build steps for Gradle and Maven, and recommended application properties for dynamic ports. A code block shows a Gradle build script and another shows a YAML configuration file for a Kubernetes service.

Слика 23: Преглед за документација

Апликацијата е обезбедена со системот за автентикација keycloak, при што за одредени барања е потребно корисникот да биде автентициран, за оваа цел се користи OpenID Connect протоколот, при што корисникот добива токен кој го користи за автентикација на барањата, овој протокол го користи OAuth2 при што апликацијата со користење на OAuth2 сервиси пристапува до keycloak и се добива токен со кој се проверува до што точно има пристап одреден корисник. Одредени барања се исклучиво достапни само до администраторот на апликацијата, при што се користи улогата која е доделена на корисникот за да се одреди дали корисникот има пристап до одредени ресурси.

Апликацијата поддржува автоматско чистење на ресурси кои не се користат, при што секој ден се бришат извршни датотеки кои се постари од 1 ден, со што се овозможува ослободување на простор на дискот. Исто така доколку е избришана слика од регистарот на слики, во регистарот остануваат податоци кои не се чистат автоматски, и се повикува команда од платформата за хостирање на апликации (Kubernetes) кон регистарот за да се избришат сите несакани податоци.

Додека апликацијата гради слики за контејнери, апликацијата ги складира сите податоци за одредена изградба на сликата во кеш меморија на самата апликација, поради тоа овозможено е следење на статусот на градењето, како и прикажување на логови во реално време до корисникот, овие податоци се чуваат само додека трае процесот на градење, откако сликата е изградена овие податоци се бришат од кеш меморијата, но логовите од градењето се зачувуваат во базата на податоци за понатамошно прегледување од страна на корисникот. Сликите се градат асинхроно, при што корисникот не мора да чека додека сликата се изгради, туку може да продолжи со користење на апликацијата и да прегледува други делови од апликацијата, додека во позадина се гради сликата^[15].

Апликацијата исто така менаџира распоредување во кеш меморијата, односно откако корисник креира распоредување на апликација, одредени податоци се чуваат во кеш меморијата за пристап до статусот на апликацијата. Потоа овие податоци се претставуваат на корисник со користење на веб сокети, при што корисникот добива известување во реално време за статусот на својата апликација.

5 Предизвици при развој на апликацијата

Главен предизвик при развој на оваа апликација беше да се овозможи лесно хостирање на апликации, при што корисникот не би требало да има големи познавања за работење со специфични платформи, туку едноставно да може да ја хостира својата апликација со неколку кликови. За ова решение потребно беше да се користи одредена платформа, за тоа беше избрана Kubernetes платформата, која овозможува лесно хостирање на апликации во контејнери и менаџирање на ресурсите. Платформата дозволува преку дефинираниот интерфејс и потребните ресурси да се хостира апликација. За поврзување на апликацијата со Kubernetes платформата беше користена библиотеката `java-kubernetes-client` која овозможува лесно управување со ресурсите во рамки на кластерот, дава можност за интеграција со кластерот со цел да се имплементираат повеќе комплексни функционалности потребни на корисниците за преглед и за менаџирање на нивните апликации.

5.1 Овозможување на преглед и следење низ процесот за хостирање

Друг предизвик беше како да се овозможи преглед на распоредувањата на апликации, при тоа корисник да може да види што апликации се хостирали, за оваа цел беше искористена база на податоци за чување на сите потребни информации за градење на ресурсите кои платформата ги очекува за хостирање на апликации, информациите во базата на податоци ги следат релациите помеѓу ресурсите на платформата, и со ова се дозволува апликацијата да биде флексибилна во тоа како се хостираат апликации, при тоа корисникот има можност да ги менаџира своите апликации преку корисничкиот интерфејс, и заради тоа беше потребно да се чуваат сите информации за состојбата на ресурсите од платформата, бидејќи голема лимитација е што некои работи како Kubernetes конфигурациски мапи не се достапни веднаш до апликацијата откако се ажурира вредноста, при тоа за апликацијата да ги добие овие вредности потребно е да се направи рестарт на истата, но исто така некои вредности во ресурсите не е возможно да се ажурираат без да се избриши тој ресурс, за надминување на оваа лимитација се следи во позадина што точно ажурирал корисникот, и доколку е очекувана промена за која сме сигурни дека е возможна се праќа оваа промена до кластерот и се прави ажурирање на засегнатиот ресурс, но доколку се детектира промена која не ги задоволува ни еден од условите се прави целосно рекреирање на ресурсите, ова би значело дека корисничката апликација се рестартира.

Основна функционалност за корисници е можноста за следење на логови и статус на апликацијата во реално време, како и можноста за управување со ресурсите и можност да направи рестарт или ажурирање на сликата која се користи, за оваа цел беше потребно да се користат веб сокети кои овозможуваат двонасочна комуникација помеѓу клиентот и серверот, при што серверот во реално време испраќа информа-

ции до клиентот за статусот на апликацијата, како и логови од апликацијата, ова овозможува корисникот да има преглед на својата апликација во реално време и да може да реагира доколку нешто не е во ред со неговата апликација. За следење на овие апликации беше потребно да се креира кеш за секоја претплата за специфичните информации кои корисникот ги бара, и секој веб сокет креира своја претплата за одредени апликации, за оваа цел сите делови кои имаат отворена интеракција со кластерот за добивање на некакви информации од страна на платформата се чува информација до самиот кеш, доколку некој друг корисник ја побара истата информација не би се креирала нова врска до платформата, туку би се користела веќе постоечката врска, доколку никој не слуша на оваа врска таа се брише од кешот и се ослободуваат ресурсите.

5.2 Градење на слики за контејнери

За корисникот да добие можност да хостира апликација потребно беше да се овозможи на корисник да може да гради слики за контејнери, кои ги очекува Kubernetes платформата, за оваа цел најлесен начин беше да се користи алатката kaniko која овозможува градење на слики во рамки на самата платформа, без потреба од посебен Docker демон, главен предизвик тута беше како да се овозможи корисникот да може да ги прилагодува параметрите за градење на slikата, при тоа да не се ограничува само на еден начин или една технологија за градење, но исто така беше потребно да се овозможи на корисник поедноставен пристап, и не би било соодветно на корисник да се дозволи да внесува целосен Dockerfile, бидејќи тоа би барало од корисникот да има познавања за градење на слики, познавање од самата алатка која ги гради slikите. Како решение беше да се овозможи администраторот на апликацијата да креира шаблони за градење на слики, при што администраторот има можност да ги дефинира сите потребни параметри за градење на slikата, како и кои полиња ќе бидат достапни до корисникот за прилагодување, при тоа корисникот има можност да ги прилагодува само тие полиња кои се дефинирани од страна на администраторот, со што се овозможува флексибилност во градењето на slikите, но исто така се олеснува процесот за корисникот кој не би требало да има големи познавања за градење на слики. Голема придобивка беше тоа што корисникот добива само неколку полиња за прилагодување, и како едноставна опција им е понудено да прикачат извршна датотека, од која се креира slikата, при тоа за конфигурација се праќаат потребни параметри.

Градењето на слики за контејнери беше имплементирано со прикачување на извршна датотека од страна на корисникот, но за истата да се прати до WebDAV серверот беше потребно да се додади поддршка за прикачување на датотеки но истовремено и да се пратат потребните параметри за градење на slikата, но доколку не беше потребно да се праќа извршна датотека, туку само праќање на параметри, за ова беше потребно да се овозможи и таква опција. Затоа овие два дела остануваат посебни во апликацијата, и е дефиниран посебен повик доколку се праќа извршна датотека, и посебен повик во случајот каде не се испраќа датотеката, туку само параметри.

5.3 Простор за чување на извршни датотеки и слики

Друг проблем беше каде да се чуваат овие извршни датотеки, бидејќи овие датотеки е потребно да се пристапат во рамки на кластерот од страна на алатката

за градење на слики, но не беше возможно да се праќаат директно до алатката за градење, за оваа цел беше искористен WebDAV сервер кој е хостиран исто така во рамки на платформата, овој сервер е скриен за корисници, и тие немаат директна интеракција со истиот, интеракцијата се врши преку апликацијата, за комуникација со WebDAV серверот се користи WebDAV клиент за Java наречен **sardine**, овој клиент целосно го имплементира WebDAV протоколот.

Изградените слики се чуваат во регистар за слики, при што беше потребно да се пристапат овие слики во рамки на кластерот, за оваа цел беше искористен Docker registry кој е хостиран во рамки на платформата, овој регистар овозможува чување на слики и пристап до истите од страна на кластерот, за да се овозможи чистење на сликтите беше потребно да се комуницира со регистарот преку неговиот интерфејс, за ова се користи библиотеката **spring-webflux** која овозможува лесно праќање на HTTP барања и примање на одговори од регистарот.

5.4 Развој на апликацијата и локално тестирање

Најголем предизвик беше развојот на апликацијата бидејќи апликацијата е длабоко интегрирана со Kubernetes платформата, затоа за тестирање на апликацијата беше потребно да се има пристап до кластер, за оваа цел беше искористен kind кој овозможува креирање на локален кластер, но тоа не беше доволно, исто така за пољесен развој апликацијата содржи две можности за започнување на истата, првата можност беше да се користи од внатрешноста на кластерот, за оваа цел постои Spring профил наречен **cluster** во кој се конфигурираат сервисите преку нивните внатрешни адреси, при што апликацијата се стартира како дел од кластерот и се користат сите сервиси кои се хостирани во рамки на кластерот. Втората можност беше да се користи Spring профил наречен **external**, при што сите сервиси се пристапуваат преку нивните надворешни адреси, ова овозможува развој на апликацијата од локална машина, при што апликацијата комуницира со сервисите кои се хостирани во рамки на кластерот и со самиот кластер, но самата апликација не е дел од кластерот.

За да се овозможи комуникација од надвор од кластерот се користи Nginx ingress контролер кој овозможува пристап до сервисите од надворешната мрежа, и потребно беше истиот да се интегрира во рамки на кластерот. Дополнително потребно беше да се интегрираат сите дополнителни сервиси потребни за развој на апликацијата, како што се базата на податоци, WebDAV серверот, keycloak серверот, и регистарот за слики, сите овие сервиси беа интегрирани во рамки на кластерот за да се овозможи целосно тестирање на апликацијата во реални услови. За интеграција со кластерот се користи апликацискиот интерфејс на кластерот кога апликацијата се наоѓа во рамки на кластерот потребно беше да се конфигурираат улогите кои апликацијата ги добива, за оваа цел беше искористен RBAC системот на Kubernetes кој овозможува дефинирање на улоги и дозволи за пристап до ресурсите во рамки на кластерот.

За тестирање на градењето на сликтите се користеше kubectl^[16] алатка која овозможува интеракција со кластерот, при тоа за менадирање на потребните ресурси за локално тестирање, доколку една алатка не е соодветно наместена апликацијата не може да функционира правилно, ова е голем проблем бидејќи апликацијата има многу екстерни зависности, за оваа цел беше направена скрипта која овозможува лесно поставување на сите потребни сервиси во рамки на кластерот, при тоа се овозможува брзо поставување на целата околина за тестирање на апликацијата.

Во случај како што се WebDAV или регистарот за слики, се користеше алатка како curl за тестирање на овие сервиси^[17].

5.5 Автентикација и авторизација

Автентикацијата и авторизацијата беше имплементирана со помош на Keycloak серверот, кој овозможува лесно управување со корисници и улоги, за оваа цел беше потребно да се интегрира Keycloak серверот во рамки на кластерот, и да се конфигурираат сите потребни клиенти и улоги за апликацијата. Главен предизвик беше како да се овозможи контрола на пристап на различни ресурси во апликацијата за оваа цел беше искористен OpenID Connect протоколот кој овозможува авторизација и автентикација помеѓу корисникот и серверот. Дополнително беше потребно да се овозможи интеграција на Keycloak серверот со самата апликација, за оваа цел беше искористен Spring Security кој овозможува лесна интеграција со OAuth2 протоколот и Keycloak серверот, Додека на корисничкиот интерфејс беше искористена библиотеката што ја развиваат луѓето што работат на Keycloak наречена `keycloak-js`^[18] која овозможува лесна интеграција на Keycloak серверот со веб апликации.

6 Подобрувања и осврт кон иднината

Апликацијата беше развиена со основните функционалности во предвид, при тоа главен фокус беше да се овозможи лесно хостирање на апликации во рамки на Kubernetes платформа, но постојат неколку подобрувања кои би можеле да се направат во иднина за да се подобри корисничкото искуство и функционалностите на апликацијата, едно такво подобрување би било да се овозможи поддршка за сервиси како што се Kafka, RabbitMQ, бази на податоци, додека во моментот ова е можно да се изведи со користење на апликацискиот интерфејс наменет за кориснички апликации, но посветено решение за ова би овозможило поголема контрола, и полесно управување од страна на администраторите и корисниците.

Дополнително би можело да се овозможи поддршка за повеќе платформи за хостирање на апликации, како што се Docker Swarm, OpenShift, при што администратор би имал можност да избере која платформа сака да ја користи за хостирање и да го вметни решението на истата платформа. Исто така една лимитација е тоа што градењето на слики е врзано со самата платформа, едно подобрување би било апликацијата да овозможи поддршка за прикачување на слики до користениот регистар, при што корисникот би имал можност да гради слики на свој начин и да ги прикачува до регистарот, наместо да се користи вградената функционалност за градење на слики.

Друга можност за подобрување би било да се овозможи контрола врз страниците со документација, каде што на администратор би се дозволило да управува со содржината, со ова на корисниците би се овозможило подобро искуство при користење, затоа што администраторот ги познава своите корисници и знае повеќе што би им било потребно.

7 Заклучок

Решението претставува целосен систем за хостирање на апликации во облак, кој овозможува лесно управување и распоредување на апликации преку интуитивен кориснички интерфејс. Притоа со користење на современи технологии како Kubernetes и Docker, се обезбедува независност од инфраструктурата и се овозможува голема флексибилност на решението, доколку би било потребно, решението дозволува проширување со какви било технологии кои се потребни за одредени апликации.

Главна предност на ова решение е едноставноста за користење, при што од страна на еден корисник се бара минимален број на информации, додека при градење на слики од апликациите од корисникот потребно е само извршна датотека која има можност истиот корисник да ја изгради на свој начин, независно од решението. Ова овозможува брзо хостирање на апликација без потреба од сложени системи за градење на истите, но истовремено доколку е потребно исто така има можност за градењето слики да се прилагоди според потребите на корисниците, каде што се овозможува користење на системот за градење како CI/CD процесот (цевководот).

Апликацијата го решава проблемот што настанува при хостирање на апликации, каде што на корисниците им е потребно големо познавање од повеќе технологии, платформи, и концепти за да можат да ги хостираат своите апликации, со ова решение се овозможува лесно хостирање на апликации без потреба од познавање на сите тие концепти, додека истовремено се овозможува доволно можности за прилагодување и конфигурација за понапредните корисници.

Истовремено беше изработен интуитивен интерфејс кој овозможува лесно управување со апликации, кој што за корисниците е поделен на повеќе делови, како што се, градење на слики, организација во групи и распоредување на апликации. Интерфејсот истовремено нуди следење на апликациите во реално време, овозможувајќи на корисниците да ги следат логовите и статусот на нивните апликации во процесот на градење и распоредување. Додека за администраторите беше изработен посебен интерфејс кој овозможува контрола врз шаблоните за градење, контрола врз ресурсите на платформата, исто така со изборот на технологии се овозможува менаџирање на корисниците и нивните улоги со користење на Keycloak. Со што се минимизира потребата од интеракција со базата или со платформата со користење на корисничкиот интерфејс.

Една голема предност на решението е исто така самото хостирање, за да се хостира апликацијата (решението) не се потребни екстерни сервиси кои работат надвор од самата платформа, туку сите потребни сервиси можи да се дел од платформата, како и самата апликација што подоцна менаџира останати апликации и распоредувања. Ова исто така би овозможило лесна промена на околина.

Додека решението нуди едноставност и истовремено голема флексибилност на корисниците, исто така е присутна комплексноста која доаѓа со флексибилноста, првично апликацијата нуди поддршка за градење на Java апликации, доколку е потребна промена на технологијата за градење на слики тоа е веќе понудено од страна на апликацијата, но од страна на администраторот се бара познавање на инфраструктурата и зависностите за да има можност да креира нови шаблони за градење.

8 Библиографија

- [1] Gradle Inc. Gradle, 2026. URL: <https://gradle.org/>.
- [2] Spring Community. Building web applications with spring boot and kotlin, 2026. URL: <https://spring.io/guides/tutorials/spring-boot-kotlin>.
- [3] The Svelte team. Svelte documentation, 2026. URL: <https://svelte.dev/docs/svelte/overview>.
- [4] The Svelte team. Sveltekit documentation, 2026. URL: <https://svelte.dev/docs/kit/introduction>.
- [5] Cloud Native Computing Foundation (CNCF). kubernetes, 2026. URL: <https://kubernetes.io/>.
- [6] Kubernetes team. Kubernetes java client, 2026. URL: <https://github.com/kubernetes-client/java?tab=readme-ov-file>.
- [7] The PostgreSQL Global Development Group. Postgresql, 2026. URL: <https://www.postgresql.org/>.
- [8] Cloud Native Computing Foundation (CNCF). Keycloak, 2026. URL: <https://www.keycloak.org/>.
- [9] Kaniko team. Kaniko, 2026. URL: <https://github.com/GoogleContainerTools/kaniko>.
- [10] Cloud Native Computing Foundation (CNCF). Container registry api v2, 2026. URL: <https://distribution.github.io/distribution/spec/api/>.
- [11] Rod Johnson , Juergen Hoeller , Keith Donald , Colin Sampaleanu , Rob Harrop , Thomas Risberg , Alef Arendsen , Darren Davison , Dmitriy Kopylenko , Mark Pollack , Thierry Templier , Erwin Vervaat , Portia Tung , Ben Hale , Adrian Colyer , John Lewis , Costin Leau , Mark Fisher , Sam Brannen , Ramnivas Laddad , Arjen Poutsma , Chris Beams , Tareq Abedrabbo , Andy Clement , Dave Syer , Oliver Gierke , Rossen Stoyanchev , Phillip Webb , Rob Winch , Brian Clozel , Stephane Nicoll , Sebastien Deleuze. Spring framework - spring webflux, 2026. URL: <https://docs.spring.io/spring-framework/reference/web/webflux/new-framework.html>.
- [12] Greg Stein, Jim Whitehead. Webdav, 2026. URL: <http://www.webdav.org/>.
- [13] Rod Johnson , Juergen Hoeller , Keith Donald , Colin Sampaleanu , Rob Harrop , Thomas Risberg , Alef Arendsen , Darren Davison , Dmitriy Kopylenko , Mark Pollack , Thierry Templier , Erwin Vervaat , Portia Tung , Ben Hale , Adrian Colyer , John Lewis , Costin Leau , Mark Fisher , Sam Brannen , Ramnivas Laddad , Arjen Poutsma , Chris Beams , Tareq Abedrabbo , Andy Clement , Dave Syer , Oliver Gierke , Rossen Stoyanchev , Phillip Webb , Rob Winch , Brian Clozel , Stephane Nicoll , Sebastien Deleuze. Spring framework - websocket api, 2026. URL: <https://docs.spring.io/spring-framework/reference/web/websocket/server.html>.
- [14] lookfirst. Sardine, 2026. URL: <https://github.com/lookfirst/sardine>.
- [15] Spring Community. Creating asynchronous methods, 2026. URL: <https://spring.io/guides/tutorials/spring-boot-kotlin>.

- [16] The Kubernetes Authors. Command line tool (kubectl), 2026. URL: <https://kubernetes.io/docs/reference/kubectl/>.
- [17] Using Curl commands with Webdav. Postgresql, 2026. URL: <https://www.postgresql.org/>.
- [18] The Keycloak Authors. Keycloak js, 2026. URL: <https://www.npmjs.com/package/keycloak-js>.