ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації

(повне найменування інституту, назва факультету)

Кафедра електронної техніки

(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри

Олександр

ВОВНА

(підпис) (ім’я, прізвище)

«___»_____2022 р.

Випускна кваліфікаційна робота

магістра

(освітній ступінь)

на тему «Засоби автоматизації розгортання монолітних та мікросервісних додатків»

Виконав: студент 2 курсу, групи КІм-21

(шифр групи)

спеціальності 123 «Комп’ютерна інженерія»

(шифр і назва спеціальності)

Дар’я ПОНОМАРЕНКО

(ім’я та прізвище) (підпис)

Керівник доц. каф. ЕТ, к.т.н., доц. Віталій ШАМАЄВ

(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензенти В.о. зав.каф., доц. каф. ПМІ, к.т.н., доц. Наталія МАСЛОВА

(посада, науковий ступінь, вчене звання, ім’я та прізвище) (підпис)

Зав.каф., доц. каф. ЕІ, к.т.н.,доц. Олександр КОЛЛАРОВ

(посада, науковий ступінь, вчене звання, ім’я та прізвище) (підпис)

Нормоконтроль:

(підпис) доц.каф.ЕТ,к.т.н.,доц.Віталій ШАМАЄВ

Засвідчую, що у цій дипломній роботі немає запозичень з праці інших авторів без відповідних посилань.

Студент _____

(підпис)

Луцьк – 2022 р.

ДВНЗ «Донецький національний технічний університет»
Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра електронної техніки
Освітній ступінь магістр
Спеціальність 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Вовна
«____»_____2022 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Пономаренко Дар'ї Володимирівни
(прізвище, ім'я, по батькові)

1. Тема роботи «Засоби автоматизації розгортання монолітних та мікросервісних додатків»
керівник роботи Шамаєв Віталій Віталійович, к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від 28.09 2022 року № 446

2. Строк подання студентом роботи 2 грудня 2022 року

3. Вихідні дані до роботи: результати науково-дослідної роботи, переддипломної практики

4.Зміст пояснювальної записки (перелік питань, які потрібно розробити):
1 Постановка задачі
1.1 Проблема розробки програмного забезпечення
1.2 Основні етапи та моделі розробки ПЗ
1.3 Визначення мети і завдань роботи
2 Проектування системи
2.1 Опис та вимоги до системи
2.2 Інтеграція та розгортання
2.3 Структура та компоненти системи
2.4 Висновки до другого розділу
3 Реалізація системи
3.1 Вибір інструментальних засобів
3.2 Налаштування компонентів системи
3.3 Налаштування інфраструктури системи
3.4 Висновки до третього розділу
4 Тестування й впровадження системи
4.1 Вимоги до платформи та оточення

4.2 Інструкція до використання
4.3 Тестування системи
4.4 Висновки до четвертого розділу
5. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сергієнко О.І.		

6. Дата видачі завдання 22.09.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	28.09.2022-05.10.2022	
2	Постановка завдання	06.10.2022-09.10.2022	
3	Огляд тематики роботи, актуальність роботи, формулювання мети на задач	10.10.2022-14.10.2022	
4	Проектування алгоритмічного забезпечення	15.10.2022-20.10.2022	
5	Проектування програмного забезпечення	21.10.2022-31.10.2022	
6	Розробка програмного забезпечення	01.11.2022-15.11.2022	
7	Оформлення пояснювальної записки	01.11.2022-15.11.2022	
8	Нормоконтроль пояснювальної записки та додаткових матеріалів	16.11.2022-30.11.2022	
9	Рецензування роботи	01.12.2022-10.12.2022	
10	Захист роботи	згідно графіку	

Студент _____ Пономаренко Д.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шамаєв В.В.
(підпис) (прізвище та ініціали)

ABSTRACT

Ponomarenko D.V. Tools for automating the deployment of monolithic and microservice applications / Final qualifying work for obtaining an educational degree «Master» in specialty 123 «Computer Engineering». – DonNTU, Lutsk, 2022.

The purpose of the work consists in increasing the quality of the source code and speeding up the process of developing a software product by implementing a system of automated testing and assembly of programs.

The object of work is methods of developing, analyzing, testing, and compiling applications.

The subject of the work is a cloud-based system for automated testing and application development.

The following main tasks are solved in the qualification work:

1) An analysis of existing problems in the field of software development was carried out.

2) An overview of the software market was carried out, the selection and comparison of relevant solutions was carried out, the optimal options were selected for use in work.

3) Software products have been configured for system operation.

4) The software settings were transferred to the helm configuration files to automate the deployment of programs.

5) Instructions for using the project have been created.

The practical result of the qualification work is a system of automated testing and assembly of programs.

Keywords: VCS, AWS, Kubernetes, Helm, Jenkins, Nexus, Sonarqube, IaC, Cloud, CI.

АНОТАЦІЯ

Пономаренко Д.В. Засоби автоматизації розгортання монолітних та мікросервісних додатків / Випускна кваліфікаційна робота на здобуття освітнього ступеня «магістр» за спеціальністю 123 Комп'ютерна інженерія. – ДВНЗ ДонНТУ, Луцьк, 2022.

Мета роботи полягає у підвищенні якості вихідного коду та прискорення процесу розробки програмного продукту шляхом впровадження системи автоматизованого тестування та збирання програм.

Об'єктом роботи є методи розробки, аналізу, тестування та складання додатків.

Предметом роботи є хмарна система автоматизованого тестування та складання додатків.

В кваліфікаційній роботі вирішені такі основні задачі:

1) Проведено аналіз існуючих проблем у сфері створення програмного забезпечення.

2) Проведено огляд ринку ПЗ, проведено відбір та порівняння відповідних рішень, вибрано оптимальні варіанти для використання в роботі.

3) Проведено налаштування програмних продуктів для роботи системи.

4) Проведено перенесення налаштувань ПЗ у конфігураційні файли helm для автоматизації розгортання програм.

5) Створено інструкцію щодо використання проекту.

Практичним результатом кваліфікаційної роботи є система автоматизованого тестування та збирання програм.

Ключові слова: VCS, AWS, Kubernetes, Helm, Jenkins, Nexus, Sonarqube, IaC, Cloud, CI.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Лінтер – Статичний аналізатор коду

ПЗ – Програмне забезпечення

API – Application Programming Interface

ASG – Auto scaling group

AWS – Amazon web services

CD – Continuous deployment

Chart – Набір конфігурацій kubernetes

CI – Continuous integration

Cloud – Доступність системних ресурсів комп’ютера за вимогою

CPU – Central processing unit

Deployment – Kind in kubernetes

DNS – Domain name server

EC2 – Elastic computer cloud

ELB – Elastic load balancer

Git – Розподілена система управління версіями

IaC – Infrastructure as code

Node – Physical machine in Kubernetes cluster

Open Source – ПЗ з відкритим вихідним кодом

Proxy – Система або маршрутизатор, який забезпечує шлюз між користувачами та Інтернетом

RAM – Random Access Memory

Storage – Disc size

URL – Uniform Resource Locator

VCS – Version Control System

Yaml – Формат серіалізації даних[i]

ВСТУП

Актуальність роботи. У другому десятилітті XXI-го сторіччя все складніше стає розробляти програмне забезпечення, у зв'язку з тим, що високий рівень пропозицій на ринку ПЗ підняв рівень вимог кінцевого користувача до продукту. Велика кількість мов програмування, безліч різних пристроїв, підвищені вимоги до інформаційної безпеки, глобальна популяризація хмарних технологій призвели до того, що зараз недостатньо мати хорошу ідею і написати якісний код. Крім розробки свого продукту, його потрібно постійно тестувати, перезбирати, постійно вносити зміни у відповідності до вимог ринку.

У зв'язку з ситуацією актуальним питанням є складання, тестування, впровадження та оновлення програмного забезпечення. У цій дипломній роботі проведено аналіз цих проблем і зроблено висновки щодо ситуації, яка склалася.

Мета роботи полягає у підвищенні якості вихідного коду та прискорення процесу розробки програмного продукту шляхом впровадження системи автоматизованого тестування та збирання програм.

Об'єктом роботи є методи розробки, аналізу, тестування та складання додатків.

Предметом роботи є хмарна система автоматизованого тестування та складання додатків.

Практична цінність роботи полягає у створенні гнучкої хмарної системи автоматизації перевірки вихідного коду та складання додатків, що можуть бути інтегровані на підприємства будь-якої форми власності, а також використовуватися приватними особами.

Структура та обсяг роботи. Випускна кваліфікаційна робота обсягом 148 сторінки складається зі вступу, чотирьох розділів, висновка, переліка посилань та 4 додатків. Робота містить 35 рисунків, 7 таблиць.

У **першому розділі** розглядаються основні проблеми розробки програмного забезпечення. Описано основні етапи та моделі розробки.

Визначено недоліки у існуючих моделях та методологіях. Сформульовано мету та основні завдання кваліфікаційної роботи.

У **другому розділі** проводиться проектування системи. Описано основні вимоги, структуру та компоненти системи. Описано як проходитиме інтеграція та розгортання.

У **третьому розділі** проведено реалізацію системи. Вибрано технології та інструменти для реалізації, описано налаштування компонентів системи. Проведено налаштування інфраструктури системи.

У **четвертому розділі** було проведено тестування та впровадження системи, описано технічні вимоги. Наведено інструкцію до використання системи та проведено її тестування.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Проблема розробки програмного забезпечення

У сучасному світі кожна людина має смартфон, особистий або робочий комп'ютер, ноутбук, і всі ми користуємося величезною кількістю програм на наших пристроях. Глобальна цифровізація торкнулася всіх ринків, починаючи від навчання і закінчуючи космічною галуззю. Тенденція популяризації веде до того, що у багатьох галузях конкуренція зростає занадто швидко, що призводить до підвищених вимог до кінцевого продукту, що ускладнює процес розробки програмного забезпечення.

Сучасний процес розробки програмного забезпечення докорінно відрізняється від того, що було 20–30 років тому. Якщо раніше розробники ПЗ могли місяцями розробляти певну функцію і не бачити, як вона працює в програмі, то зараз все доходить до того, що компанії намагаються випускати оновлення свого продукту якнайчастіше, а такі лідери ринку як Facebook або Amazon роблять по кілька релізів продукту в день. Ці умови призводять до того, що розробники-початківці, які не мають великих бюджетів не в змозі відповідати вимогам ринку, що викликають труднощі при розробці їх продуктів.

Виходячи з вищевикладеної інформації слід, що розробникам потрібен комплект додаткового програмного забезпечення для складання, тестування і розгортання своїх додатків, що вимагає додаткової кваліфікації, тому що жоден продукт на сучасному ринку не вміє працювати «з коробки».

В умовах постійно зростаючого попиту на різні програмні продукти збільшується і кількість компаній і незалежних розробників, які бажають задовольнити цей попит, що швидко росте. Існує підвищена конкуренція у цій сфері, яка змушує випускати програмне забезпечення швидше та у більшій кількості, ніж конкуренти. Коли в такій великій гонці головною метою є прибуток для розробників, найважливішим стає встигнути випустити продукт у термін.

1.2 Основні етапи та моделі розробки ПЗ

З того часу, як у розробці ПЗ бере участь не один розробник, а цілі команди, що займаються аналізом і плануванням, розробкою та тестуванням, менеджмент розглядає різні підходи для поліпшення процесів управління командою.

З найпопулярніших підходів до розробки програмного забезпечення, родоначальником методології розробки була модель Waterfall. На рис. 1.1 зображено схему цього підходу до розробки програмного забезпечення.

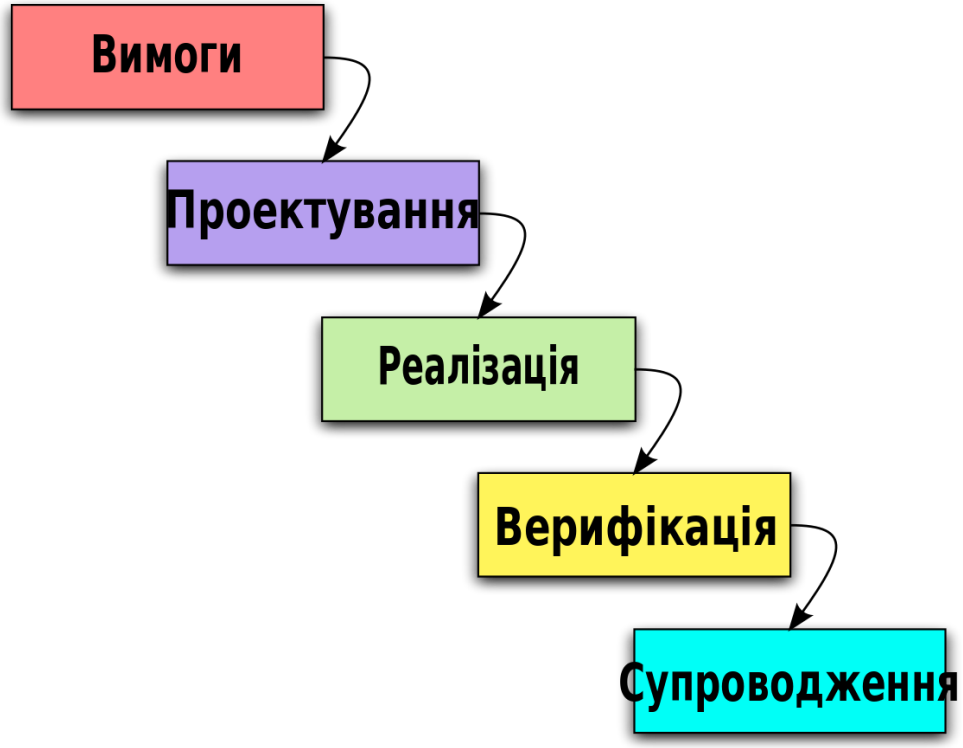

Рисунок 1.1 – Приклад методології Waterfall

До неї входять такі етапи як аналіз вимог до підсумкового програмного продукту та системний дизайн, розробка, тестування, використання, підтримка.

На першому кроці відповідальний за реалізацію проекту, як правило, це менеджер проекту, збирає всі необхідні вимоги від замовника додатка, проводить їх узгодження з технічною командою, та узгодження правок із

замовником, планування витрат та строків для кожного з етапів, планування ризиків, підготовку документації. У цій методології більшість успіху проекту залежить від цього, наскільки ретельно було проведено етап планування. Далі, після узгодження всіх технічних питань, команда переходить до розробки архітектури майбутньої програми: вибираються відповідні технології для кожного з напрямків роботи.

Наступний етап один із основних, хоч і не найбільш тривалих – розробка продукту за підготовленою документацією.

Далі проводяться різні типи тестування програми для підтвердження того, що кінцевий продукт немає помилок і його робота прогнозована. Тестувальники з документації перевіряють, що вся функціональність відповідає заявленій, проводять тестові сценарії. Після успішного тестування здійснюється впровадження додатку – його збирають, налаштовують та готують до роботи на обладнанні замовника. Як правило, додаток, що працює, залишається в технічній підтримці команди, яка його розробляла.

Це основні етапи притаманні даної методології, також можливе додавання додаткових кроків, якщо цього вимагає проект.

Потрібно зауважити, що не всі етапи проходят ідеально та так я заплановано спочатку. Незважаючи на відсутність гнучкості на початку планування розробки проекту можуть бути зауваження та неточності, які потребують нових ітерацій.

На першому етапі замовники описують свої вимоги до кінцевого продукту та своє бачення щодо функцій. Фахівець обробляє це, приводить до формульованого виду і складає документацію. Далі спеціаліст з написання документації повинен узгодити це з замовниками і якщо є недоліки знову обробити зауження, скласти нову документацію і знову нести на узгодження з замовниками. Після цього етапу треба розписати вимоги до програмного продукту для розробників цього продукту. Для них теж формулюється документація більш формальною мовою, з більш чітким описанням що і як повинно бути. Коли фахівець впорався з документацією для команди

розробників, він вже узгоджує це з технічним лідером та з командою розробників і також, якщо є зауваження, то потрібно ще раз обробити документацію і довести до потрібного стану.

Так робиться з кожним етапом, все потрібно перевіряти, узгоджувати, перероблювати якщо є зауваження та додаткові вимоги.

Основний недолік цієї методології – повна відсутність зробити коригування до ПЗ, що розробляється, після етапу узгодження документації між компанією та замовником. Також додавання нових функцій можливе тільки після впровадження продукту і означає новий цикл розробки поточної методології. Однак, у деяких випадках, коли всі вимоги чітко відомі, а бюджет занадто великий, ця методологія найбільше підходить, наприклад в аерокосмічній сфері. Інші переваги та недоліки наведені в табл. 1.1.

Таблиця 1.1 – Переваги та недоліки методології Waterfall

Переваги	Недоліки
Стійкість до змін у колективі	Дуже багато документів
Дисциплінує	Користувача та замовника повністю ізолюють
Гнучка на ранній стадії	Усі вимоги мають бути одразу відомі
Прозора	Тестування відбувається лише наприкінці

Розглянемо більш докладно про перваги та недоліки цієї методології.

Першою перевагою та одною з найважливіших, я вважаю, це стійкість до кадрових змін у колективі. Завдяки тому, що на етапі розробки документації все описується якнайбільш точно та з усіма деталями проекту дана методологія не боїться змін у штаті працівників. Учасники команди можуть приходити чи покидати проект, але це не відразиться на самій розробці. Нові фахівці швидко зможуть зорієнтуватися на новому місці та почати свою роботу. Також ця перевага дає нам відсутність додаткових витрат на комунікацію в команді. Якщо прийде новий співробітник він зможе розібратися сам бо існує докладна

документація, та скоріш приступить до роботи над задачами: для усіх процесів описані правила.

Ця методологія також дисциплінує тому що притримується чіткого плану та послідовності виконання розробки, а також завдяки суворому менеджменту. Менеджер проекту ретельно слідкує за працівниками, чи є у кожного фахцівця завдання, чи вчасно виконується робота, можливо у когось виникли питання чи з’явився блокер.

Наступні перевага яку ми розглянемо – це гнучкість на раніх етапах та прозорість. До початку етапа безпосередньо самої розробки можна легко вносити зміни до проекту, додавати чи вбирати вимоги до кінцевого продукту та інші змін на усіх попередніх етапах. Також про прозорість – це перевага, бо заздалегіть зрозуміло що буде виконуватися та відбуватися на кожному з етапів розробки програмного продукту. Завдяки цьому в рази легше спрогнозувати фінансові витрати на проект, які фахівці і в якій кількості потрібні та скільки часу це може зайняти враховуючи бюджет та команду.

Теперь ми більш детальніше зупинимося на недоліках цієї методології. Перш за все, що потрібно згадати це велика кількість документації. Якщо нам потрібно розписати все якнайбільш докладно ми повинні мати на увазі, що документів буде дуже багато. Також цю велику кількість документів потрібно постійно актуалізувати, доповнювати чи вносисти зміни згідно з реальною ситуацією на проекті.

У якийсь момент люди зрозуміли, що існуючих методів розробки недостатньо для світу, що постійно змінюється. Було зрозуміло, що waterfall не може задовольнити потреби і були чітко визначені недоліки даної методології. На зміну йому прийшли нові методології, гнучкіші, мінливіші, які дозволяють швидко і якісно підлаштовуватися під зміни та вимоги замовників до кінцевого продукту. Через це робота над програмним продуктом перетворюється в справжню бюрократію. Поки не узгодиш кожну зміну, кожне додавання до документації і не пропишешь усе докладно – нічого не сдвинеться з міста.

Наступний момент – це те, що замовника та користувача повністю ізолюють один від одного. Це має поганий вплив на розробку, бо замовник не може вносити корегування вже на етапі розробки якщо щось йде не так, а користувач не може поступово звикати до програмного продукту. Користувач отримує вже повністю готовий продукт, с великою кількістю функцій та з великою імовірністю натикнутися на помилки та баги.

Наступне, та не менш важливе те, що усі вимоги до проекту мають бути відомі одразу. Неможливо усе передбачити заздалегіть. Замовник може і сам не до кінця розуміти всіх вимог та потреб, всіх функцій та можливостей, що мають бути в кінцевому продукті. Дуже складною роботою є з’ясувати кінцевий список вимог до проекту. І неможливість вносити корегування на етапі розробки робить неможливим додавання нових функцій.

Важливим буде сказати про те, що тестування продукту відбуватиметься лише в кінці роботи над проектом, після етапу розробки. Також треба мати на увазі що тестування можут проводити некомпетентні фахівці. Враховучи все це можна зрузуміти, що тестування може зайняти великі втрати часу, бо помилку складніше буде знайти у великій кількості коду. Ця помилка чи баг інколи може з’являтися лише у тестовому середовище, але може не бути у продакшн версії. Тож ці помилки можуть наспіх закривати заплатками, а не робити якісні зміни в програмному продукті бо це несе за собою нову ітерацію розробки, збільшення витрат та інше.

Підводячи підсумок можна сказати за яких умов підходить ця методолгія розробки.

У цій роботі розглянуто лише деякі найпопулярніші та широко використовувані методології розробки програмного забезпечення у світі.

Agile[1] - це сімейство методів розробки програмного забезпечення, яке використовується як невеликими командами, так і великими компаніями, наприклад Google, Spotify, Netflix, і воно не описує якихось підходів, а засноване на agile manifesto[2]. На даний момент використовується не тільки у

сфері IT, а й багатьох інших. На рис. 1.2 зображено основні етапи даної методології.

Рисунок 1.2 – Приклад методології Agile

Сам процес розробки програмного забезпечення розбивається на невеликі цикли, ітерації, які тривають два-три тижні[3]. На кожен цикл є певний перелік завдань, які потрібно виконати та протестувати. Наприкінці кожної ітерації проводитиметься аналіз виконаної роботи та робиться постановка та коригування завдань на наступний цикл[4].

Найважливіше у цій методології розробки програмного забезпечення це акцент на результат, яким має стати робочий продукт. Гнучкість дозволяє вносити коригування і зміни підлаштовуючись таким чином під вимоги замовника і кінцевих споживачів, що часто змінюються. Також дуже важливим пунктом є безперервна комунікація команди між собою та із замовником, що дозволяє вчасно виправляти помилки, вносити необхідні зміни та розширювати

можливості програмного забезпечення[5]. При цьому збільшується якість спільної роботи команди. Переваги та недоліки продемонстровані у табл. 1.2.

Таблиця 1.2 – Переваги та недоліки методології Agile

Переваги	Недоліки
Збільшення прибутку	Найменша передбачуваність
Гарантія якості з тестуванням на кожному етапі	Більше часу та зобов'язань
Дослідження досвіду користувача на всіх етапах	Підвищені вимоги до розробників та замовників
Продукти виходять на ринок швидше у вигляді "сирої", але працюючої версії	Відсутність необхідної документації

Розглянемо докладніше переваги та недоліки цієї методології. Тож першим плюсом методології розробки Agile – це те, що використання гнучкої методології на раніх стадіях дає можливість тестувати проект невеликими частинами. Завдяки цьому легше знаходити помилки в програмі та виправити їх вчасно і правильно. Також в рази простіше знайти та виправити баг, коли програмного коду не так багато, як наприкінці роботи над проектом. Це гарантує підвищену якість самого тестування, бо це роблять досить часто.

Наступне, на чому потрібно зупинитися – це збільшення прибутку компанії-розробника за рахунок того, що продукт можна випустити раніше, за відсутності якихось функцій чи з невеличкими багами, а далі випускати патчі з виправленням помилок і оновлення з новими функціями. Таким чином з’являється користувацький досвід та самі користувачі можуть робити зауваження чи пропонувати зміни та додаткові функції. Тож програма вже починає заробляти гроші для компанії не будучи ще в готовому стані. А тим часом розробники можуть і далі працювати над покращенням продукту.

Виходячи з цього є наступна перевага, програмні продукти виходять на ринок раніше, чим коли ми будемо використовувати методологію Waterfall. Таким чином ми можемо випускати більше продуктів та регулярно. Завдяки цьому ми маємо можливість скоріш повертати свої інвестиції в проекти.

Наступна перевага – це покращена прозорсть між замовником программного забезпечення та кінцевим користувачем. Бо на усіх етапах розробки і замовник і користувач можут пропонувати зміни та корегувати програмний продукт.

І останнє на чому я хочу зупинитися в перевагах методології Agile – це зниження рисків тому що команда знаходить і виправляє будь-які проблеми та помилки на раніх етапах розробки.

А тепер потрібно більш докладно зупинитися на недоліках цієї методології, бо ідеальних методів не існує. Поперше це те, що дана методологія менш передбачувана. В деяких проектах дуже важко оцінити скільки зусиль, часу та грошей знадобиться для реалізації цього проекту. Особливо важко оцінити різни змінні на початку розробки великого програмного продукту.

Ще одним недоліком є те, що ця методологія потребує більшого часу та обов’язків. Замовники, розробники, дизайнери, тестувальники та інші члени команди повинні постійно та тісно взаємодіяти один з одним. Це може включати у себе багатокількістні бесіди у різній конфігурації, оскільки це є найкращою формою взаємодії та взаєморозуміння. Користувачі повинні бути доступними у будь-який проміжок часу для тестування та підписання на кожному етапі розробки програмного продукту, щоб розробники могли відмітити етап як завершений і тільки після цього переходити до наступного кроку розробки. Завдяки цьому підходу можна гарантувати те, що продукт зможе задовільнити потреби користувачів, але він є обтяжливим і потребує багато часу.

Наступне про що треба зауважити це підвищені вимоги до розробників та замовників. Принципи методології Agile натякають на тісне співробітництво та активну роботу користувачів. Незважаючи на те що Agile це дуже корисна, цікава та гнучка система, вона потребує великого вкладу, обов'язків від команди та замовників, великої віддачі. Тож клієнти повинні пройти навчання, щоб мати змогу допомогти в розробці програмного продукту.

Також потрібно мати на увазі, що при використанні цієї методології може бути відсутність необхідної документації. Оскільки вимогу до проекту уточнюються в момент початку розробки, документація стає менш докладною. Тож коли новий працівник з'являється в команді та приєднується до розробки проекту, він не знає про нові функції, зміни в проекти чи зміни напрямку деяких функцій у програмному продукті. Це збільшую час адаптації нових фахівців у проекті та створює непорозуміння та труднощі.

Останнє що треба описати це те, що проект легко збивається зі шляху. Використовуючи дану методологію розробки програмного забезпечення ми маємо на увазі, що проект буде зазнавати змін при розробці на кожному етапі підлаштовуючись до наявних вимог. Метод потребує дуже невеликого планування для початкку роботи над проектом і припускає що потреби споживача постійно змінюються. Надалі, якщо споживач некоректно описав свої побажання та неправильно сформулюва відгук, розробникам буде складніше продовжити роботу та вони можут зосередитися на невірному напрямку розробки. Також у методолгії Agile є потенціал для безперервного чи неконтрольованого зростання об'єма проекту, та постіно мінливий продукт може стати вічнотривалим проектом.

Ще одна методологія була винайдена в Японії на початку 1940-х років. Тоді цей метод був розроблений для оптимізації виробництва на заводі Toyota. Основний підхід у тому, що на кожному етапі не має бути недоробок і переробок, процеси мають виконуватися строго відповідно за планом.

На початку 2000-х років цей підхід був запозичений та доопрацьований для використання в управлінні процесами створення програмного забезпечення.

Канбан[7] найчастіше реалізують як дошку, в якій існує кілька колонок з різними етапами розробки програмного забезпечення. Кількість колонок та їх призначення визначає менеджер проекту. Є план завдань, які потрібно виконати. Відбувається сортування завдань за пріоритетом і в будь-який момент часу можна поміняти пріоритет, внести зміни до завдання так, як цього вимагає проект.

Так само в кожній колонці має бути обмежена кількість завдань, на це впливає кількість осіб, які перебувають на позиції, котра займається конкретною специфікою робіт. Завдання, в ході його виконання, переміщається з колонки в колонку, пересуваючись по етапах процесу розробки програмного забезпечення з одного стану в інший. Так само обмеженість кількості завдань у кожному блоці допомагає відстежити, з якими проблемами команда стикається найчастіше, на якому етапі потрібно більше фахівців, а на якому можна обійтися і меншою кількістю спеціалістів. Важливою умовою є те, що не можна брати в роботу нове завдання, якщо не закінчено роботу над попереднім. Якщо задача не може бути виконана її переміщують у спеціально виділену для таких завдань колонку, у якій описується, що конкретно заважає виконанню даних робіт.

Як описано вище найголовнішим інструментом у методології Канбан[8] є дошка з картками. Дошка може бути як фізичною, так і виконаною у програмному вигляді. Основна умова, що доступ до неї повинен бути абсолютно у кожного учасника команди, що працює над проектом.

Найчастіше на Канбан дошці може бути декілька колонок, але три основні є завжди. Перша колонка це беклог, там знаходиться список завдань, який відсортований за пріоритетом. Друга колонка – це завдання,

які знаходяться в роботі на даний момент. Третя колонка – це виконані завдання.

На одній дошці можуть бути картки навіть з різних проектів, а для того, щоб розрізнити завдання одного від завдань іншого проекту, використовують різні кольори для карток із завданнями. На кожній картці можна переглянути корисну інформацію, що стосується завдання: опис завдання, кому вона призначена, терміни виконання, пріоритет тощо. Приклад такої дошки Канбан зображено на рис. 1.3.

Рисунок 1.3 – Приклад дошки Kanban

Канбан методологія має такі принципи. Насамперед потрібно візуалізувати роботу над проектом. Розділити завдання по етапах розробки програмного продукту. Це допомагає надалі відстежувати та коригувати роботу для збільшення продуктивності команди та якості продукту.

Наступне, що потрібно виконати – це створити необхідну кількість колонок із назвами етапів розробки даного проекту. Під кожен проект може бути своя кількість колонок та відповідні назви процесів. Завдання мають бути контрольовані, щоб у разі затримки на певному етапі вчасно проаналізувати причину затримки і вирішити її. Також потрібно постійно оновлювати статус

самої задачі, на якому етапі вона перебуватиме, що вже було з нею зроблено і що ще потрібно виконати. Це дає можливість орієнтуватися скільки часу може бути витрачено на виконання цього завдання у конкретного спеціаліста. А коли співробітник із якихось причин залишає команду, це допомагає не загубитися в чужому завданні і швидко влитися в його вирішення.

Цінності даного методу полягають у тому, що він допомагає ділиться інформацією в реальному часі, дає змогу досягати рівноваги між навантаженням, яке лягає на команду та її можливостями. Канбан налаштований на тісне співробітництво та бесперервне вдосконалення комунікацій між учасниками команди. Метод також загострює увагу потребам замовника і кінцевого споживача програмного продукту. Дуже важливим є те, що всі учасники команди повинні ставитись один до одного з повагою, тут немає місця яскраво вираженої ієрархії, всі спілкуються на рівних, але при цьому кожен має намагатися стати прикладом для своїх колег за проектом. Потрібно ставитися з розумінням до цілей проекту та узгоджено рухатися до кращого результату. Додатково, у табл. 1.3 є переваги та недоліки даного методу.

Таблиця 1.3 – Переваги та недоліки методології Kanban

Переваги	Недоліки
Гнучкість планування	Не підходить для довгострокового планування
Контроль термінів виконання	Не підходить для великих команд
Підвищення ефективності роботи	Відсутність чітких термінів може знизити її корисність
Наочність просування роботи	Може не підходити в середовищі, що дуже швидко змінюється.

Після наведеної таблиці потрібно зупинитися більш детальніше на преевагах та недоліках.

Поперше, методологія Канбан побудована таким чином, що команда концентрується на єдиній задачі, не дивлячись на те, що цих задач може бути декілька. Менеджер проекту слідкує за виконанням, він може змінювати пріорітет задачі не впливаючи на сам процес роботи над проектом. Після того як команда завершує одну задачу вони робить наступну. Це показує нам гнучкість данної системи. На увазі мається не завжди одна задача на цілу команду, тут скоріш про те, що кожен фахівець бере собі в роботу одну задачу і працює з нею до того моменту, як виконає її чи менеджер не відміть її за непотрібністю.

Контроль термінів виконання – це теж одна з важливих переваг.

Ця методологія дозволяє менеджеру проекту та іншим членам команди відстежувати на якому етапі зараз знаходиться задача, робочий процес відкрит для усіх членів команди. Тож керівникам проекту легше оптимізувати тривалість проекту та прогнозувати час, який вимагається для рішення майбутніх задач.

Підвищена ефективності роботи досягається за рахунок того, що фахівець концентрується на виконанні однієї задачі. Багатозадачність значно знижає якість роботи та сповільнює просування проекту вперед. Чим більш задач, які в стані застою тим частіше працівник буде перемикатися з однії задачі на іншу. Наведений метод допогає швидко знаходити та вирішувати слабкі моменти тому допогає запобігати таких застоїв. Це значно скоротшує час виділений на роботу над завданням та підвищує якість результату виконаної роботи.

Однією з найголовніших переваг при роботі за допогою методології Канбан є наочність просування роботи. Ми можемо побачити на якому етапі зараз знаходиться завдання, це доступно для кожного члена команди. Завдяки цьому легше виявити на якому етапі виникають труднощі.

Також потрібно відмітити що ця методолгія проста та зрозуміла інтуітивно. Всі члени команди, які раніше не працювали за данною методологією швидко розуміють принцип роботи з нею. Ціна обслуговування

та можливість швидко впровадити методологію Канбан у проект є вагомою перевагою.

Недоліками дошки Канбан можна назвати те що вона не підходить для довгострокового планування. Методологія розрахована на досягненя короткочасних цілей. Робота над проектом вибудовується таким чином, що задачі виконуються в порядку актуальності завдання, при цьому приорітетність завдань у міру роботи може змінюватися в залежності від багатьох факторів.

Другий недолік дошки, що вона не підходить для великих команд. Кількість членів команди впливає на можливість відстежувати прогрес кожного над проектом. Тому, краще за всього, щоб в одній команді було не більше десяти чоловік. Але на практиці методологію використовують і для декілька більшої кількості людей.

І останнє на чому треба приділити увагу це те, що методологія не зосереджена на досягнені цілей та стратегії проекту.

Scrum[9] – ця методологія розробки програмного забезпечення допомагає кожному члену команди вести щільну спільну роботу. Підхід даної методики в тому, що над вирішенням завдань працюють усі, а не так що замовник поставив завдання і просто чекає, коли команда все зробить. З використанням цього методу управління проектами спеціалісти у команді мають конкретну роль.

Product owner або власник продукту – особа, яка представляє інтереси компанії щодо цього проекту. Він знає, який має бути продукт розробки, його призначення та функції. Ця людина зацікавлена у якісному результаті. Product owner не працює саме в команді, а скоріше знаходиться на стороні клієнта, замовника продукту, але працює безпосередньо з усією командою, позначає деякі завдання, їх пріоритет.

Scrum-майстер – це не керівник стандартного розуміння цього слова. Його мета полягає в тому, щоб зробити команду більш самостійною, мотивувати її, усувати проблеми, які гальмують процес розробки або призводять до конфліктів між співробітниками. Також

скрам-майстер виступає сполучною ланкою в комунікації між власником продукту та командою розробників. Зазвичай скрам-команда складається з невеликої кількості людей. На великих проектах, де існує кілька скрам-команд, проводять зустрічі для комунікації скрам-майстрів усіх команд.

Наступна роль - це Scrum-команда. Scrum-команда складається із невеликої кількості людей, приблизно 9-10 осіб[10]. Команда обов'язково має складатися з усіх фахівців, необхідних для роботи над проектом. Якісну роботу команди забезпечують за рахунок того, що кожен спеціаліст бажано повинен бути зайнятий тільки одним проектом, склад команди по можливості залишають незмінним. Ці умови важливі, щоб нічого не відволікало команду від розробки програмного забезпечення, це покращує якість та швидкість самої розробки і так простіше людям, які працюють над проектом. Так само немає керівника, який би становив беклог завдань, це робить сама команда, розставляє пріоритети, створює завдання, розподіляє галузі відповідальності кожного фахівця. Команда розробників забезпечує якість виконаних завдань та продукту загалом.

Сам метод працює так, ділиться на спринти, які тривають 2-4 тижні, якою тривалістю будуть спринти вирішується на самому початку проекту. Під час одного спринту виконується обмежена кількість завдань, які були заплановані на цей спринт. Також зазвичай проходять щоденні збори, на яких кожен фахівець розповідає над яким завданням зараз працює, що було виконано з моменту попередніх зборів, що планує робити далі, чи є якісь блокери, проблеми чи питання. Також кілька разів на тиждень відбуваються збори фахівців одного напряму для вирішення технічних питань. Це допомагає контролювати просування завдань та проекту загалом. Наприкінці кожного спринту робиться план завдань наступного, проводиться обговорення проблем, що виникли протягом спринту. Приклад роботи метода на рис. 1.4

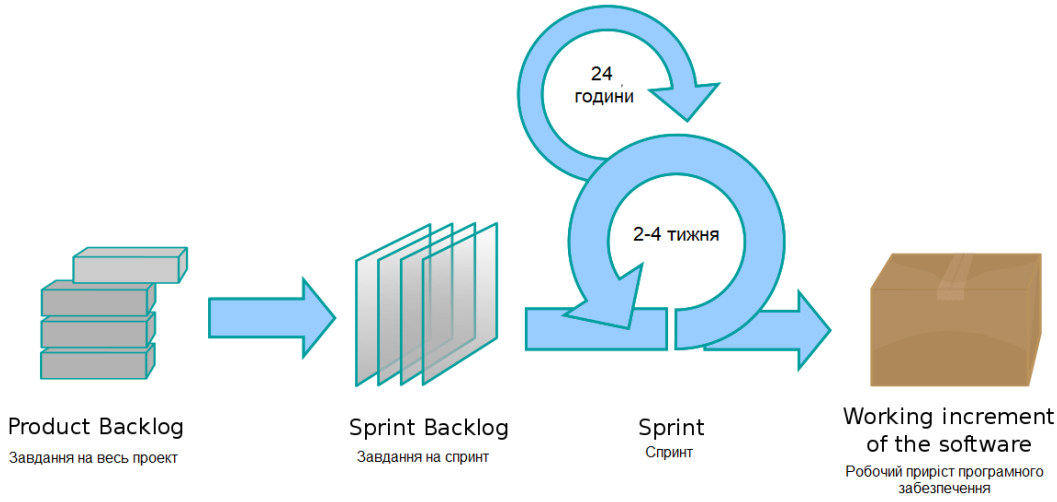

Рисунок 1.4 – метод розробки Scrum

Існує таке поняття як product backlog та sprint backlog. Перше – це ті завдання, виконавши які ми отримуємо готовий продукт. Другий - це перелік завдань саме для даного спринту, який складає команда і погоджує з власником товару[11].

Отже, як працює цей метод. На початку відбувається планування спринту. На цій події повинні бути присутніми всі, власник продукту, скрам-майстер і всі члени команди розробників. Власник продукту висловлює завдання та їхню пріоритетність, що, на його думку, має бути виконано за час спринту. Команда розробників обговорює це, оцінює час, який займе те чи інше завдання та висловлює, що справді можна встигнути виконати за час спринту. Поставлені на планування спринту завдання повинні бути виконані, і не тільки на словах, що зроблено «це і це», а саме готовий результат, готовий продукт або функція.

Зупинку спринту роблять у дуже поодиноких випадках. Спринт може зупинити власник продукту через непотрібність поставлених завдань на даному спринті, також це може зробити команда розробників, якщо розуміє, що не встигає виконати завдання вчасно. У такому разі збирається вся команда і вирішуються причини зупинки спринту, якщо це якісь критичні проблеми, то обговорюють, як уникнути такого у майбутньому. Обов'язковою умовою для цієї методології є наявність

збору команди для обговорення отриманих результатів після закінчення спринту та зборів команди для обговорення проблемних моментів, що трапилися з командою протягом спринту. Як правило, такі збори проводяться в останній день спринту. На них команди обговорюють усі досягнуті результати, іноді команди документують такі збори, як звітність, а також обговорюють технічні та нетехнічні проблеми, які виникли на останньому спринті та обговорюють дії для уникнення таких ситуацій у майбутньому. Після обговорення цього призначають завдання на новий спринт і команда починає sprint. Недоліки та переваги даного методу показані у табл. 1.4.

Таблиця 1.4 – Переваги та недоліки методології Scrum

Переваги	Недоліки
Швидка окупність та зниження витрат	Не підходить для великих команд
Гнучкість	Не підходить для великих проектів
Творчий підхід та мотивація	Не підходить для продуктів, які розробляють за строгим планом
Тісна комунікація з власником продукту	Потрібен досвід роботи у scrum-команді

Зупиняючись на цій методології ми маємо ряд переваг як для команди, так і для компанії в цілому. Ознайомимося докладніше з основними сильними сторонами скрам.

Орієнтація на клієнта та потреби бізнесу. Продукт постійно покращується, додаються нові функції та корисні деталі, збільшується цінність програмного продукту для його користувачів. А за цим одраз йде швидка окупність, бо є можливість швидкого запуск продукту «сирим» з найбільш важлими функціями та в повністю робочому стані. Інші функції додаються далі по мірі їх розробки та впровадження в проект.

Гнучкість та адптивність. Можливість швидко вносити зміни, чуйно реагувати на зміни на ринку чи інші важливі обставини та потреби

користувачів. В цій методолгії немає чіткого та невідступного слідування плану та розкладу, але є вміння пріоритезувати задачі в залежності від потреб та вимог. Команда працює короткими етапами, та на кожному з них визначає цілі та шлях їх досягнення. Це дуже прискорює процес. Також у цьому допомагає робота над різними задачами на проекті одночасно.

Керівництво з методології Скрам займає 17 сторінок тому зрозуміло, що ця методологія проста для впровадження і не потребує великих витрат часу, людського ресурсу та фінансів. Також вона практично не потребує менеджерської роботи, для налаштування усях процесів є скрам-майстер. Команда може самоорганізовуватися і враховується думка кожного члена команди.

У цій методології присутній відкритий обмін інформацією, що робить процес максимально прозорим. Кожен член команди може підствітити якусь проблему чи звернути увагу колег на важливу частину проекту.

Розглянемо деякі мінуси. В методолгії Scram так само як і в методолгії Agile недостатньо пророблена документація, не завжди достатньо часу чи невідомо про деяки деталі заздалегіть. Якщо на проекті важливо вести ідеально точну та детальну документацію потрібно вводити для цього конкретну роль, одного фахівця, котрий буде займатися виключно написанням документації.

Не підходить для великих проектів, великих команд. В такому разі вводять декілька скрам команд. Можуть бути складнощі з відстежуванням прогресу, хто на якому етапі роботи знаходиться та з якими проблемами стикнувся при роботі над своїм завданням та з комунікацією між членами команди також будуть великі складнощі.

Повинна бути активна взаємодія та віддача клієнта з процесом розробки, без цього буде швидко та сильно знижатися ефективність усієї команди. Замовник забеспечує своєчасне формування продукт-бєклогу.

Ґрунтуючись на інформації, викладеної вище, бачимо, що незалежно від обраного підходу до розробки програмного продукту, є обов'язкові кроки, такі як аналіз вихідного коду, складання, тестування та впровадження.

1.3 Визначення мети і завдань роботи

У цьому розділі було виконано такі кроки:

- Проведено оцінку вимог до розробників програмного забезпечення. Проаналізовано основні кроки та проблеми, з якими стикаються розробники ПЗ.
- Проведено оцінку вимог до розробників програмного забезпечення. Проаналізовано основні кроки та проблеми, з якими стикаються розробники ПЗ. Проведено їх глибокий аналіз із виділенням сильних і слабких сторін, описано для яких типів команд підходять ті чи інші методології.
- Виділено економічну складову у розробці програмного забезпечення, а також її вплив на технічну команду.
- Було виділено та оглянуто основні етапи у розробці програмного продукту, детально описаний кожен етап, проблеми які можуть виникнути на них.

Отже головною метою роботи є створення простого у використанні та налаштуванні середовища для централізації та автоматизації процесу збирання, зберігання, тестування, розгортання та керування монолітними та мікросервісними додатками.

Для досягнення мети необхідно виконати наступні задачі:

1) Провести детальний та чіткий опис усіх вимог до створюваної системи. Якщо система буде складатися з кількох компонентів, позначити рівні їх взаємодії та залежностей.
2) Скласти архітектуру системи, яка дозволяє спростити роботу розробників програмного забезпечення.
3) Вибрати відповідні інструменти, які забезпечують оптимальну роботу системи та повне покриття вимог до системи, що розробляється, її компоненти повинні повністю інтегруватися між собою.

4) Здійснити налаштування системи. Налаштувати незалежні системи окремо, а потім зробити конфігурацію їхньої взаємодії один з одним. Створити документ із описом роботи кожної системи та їх взаємодії.
5) У сучасному світі велика різноманітність гаджетів та пристроїв для виконання розробки, велика кількість конфігурацій. Ми не можемо передбачити, на якому пристрої розгортатиметься програмне забезпечення, тому необхідно реалізувати можливість запуску системи незалежно від того, на якому апаратному забезпеченні вона буде встановлена. Провести опис мінімальних технічних вимог апаратної частини для функціонування системи.
6) Незважаючи на те, що ми не можемо передбачити на якому пристрої розгортатиметься програмний продукт і які мови програмування та фреймворки будуть використовуватися надалі треба враховувати це при розробці системи та підтримувати всі найпопулярніші мови програмування.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Опис та вимоги до системи

При опису системи, що розробляється треба зауважити, що на вході розробники мають лише репозиторій з проектом. Система для допомоги розробки программного забезпечення має такі налаштувані і такі системи, які самі зберуть додаток у готовому вигляді.

Тож у цьому проекті злагоджена робота декількох незалежних систем які запускаються одна за одною, виконуючи свої функції для кожного етапу в розробці програмного забеспечення.

Для досягнення мети ми проводимо налаштування спочатку окремо кожної незалежної системи, а після цього налаштовуємо порядок їх виконання і як вони будуть взаємодіяти одна з іншою.

Автоматизація розгортання монолітних та мікросервісних додатків – це дуже складний процес який потребує від фахівців велокиї кількості знань та досвіду. Саме для цього ми ведемо розробку систему для обглешення цієї задачі.

Система незалежить від того яка мова програмування чи фреймворк використовуються при розробці програмного забезпечення.

Отже, для розробки цієї системи ми можемо скласти список вимог, необхідних для створення повноцінної системи, які допомагають при розробці програмного забезпечення.

Так як у сучасному світі багата кількість різноматніх мов та фреймворків програмування ми не можемо передбачити на якій мові буде написане програмне забезпечення та яка мова чи фреймворк стануть популярними через декілька років. Тому система не повинна прив’язуватися до якоїсь однієї технології.

При розробці програмного забезпеченя використовується множина различніх фреймворків кожний з котрих має свої унікальні збиральники, параметри збірки та правила написання коду. Тому система повинна

незалежати від інструментів розробників і вміти робити усі описані вище кроки самостійно без втручання фахівців по розробці.

Так як у процесі розробки будь-якого програмного забезпечення бере участь чимала кількість розробників, то всі використовують систему контролю версій для відстеження стану та паралельної робити над одним проектом. Отже, система повинна вміти працювати з найпопулярнішими системами контролю версій.

Також ми не можемо передбачити на скільки масштабні чи ні програмні продукти будуть розробюлятися за допомогою цієї системи, отже, вона повинна вміти розширятися для роботи з різними навантаженнями.

В різних проектах існуєть різни вимоги до якості вихідного коду та правил написання коду тому користувачи системи повинні мати можливість для тонкого налаштування правил за якими буде перевірятися якість написаного коду.

Так як програмний продукт це індивідуальна власність компанії чи розробників то вони повинні мати змогу зберігати зібрані артефакти недоступними для публічного доступ та мати можливість версіонувати їх. Отже у системи повинна бути присутня підсистема зберігання зібраних артефактів.

За складеними вимогами можна визначити основні компоненти системи, що розробляється. В основі майбутньої системи буде знаходитись CI/CD система, яка стежитиме за зміною у вихідному коді, займатиметься запуском аналізу коду, складанням додатка, його централізованого зберігання, запуском оточення для тестування зібраного артефакту, а також розгортання виробничого середовища. Вихідний код буде зберігатися та вилучатися із системи контролю версій.

Важливим компонентом на кількох кроках є QG. Цей елемент дозволяє контролювати належну якість вихідного коду та програми на різних етапах розробки програмного забезпечення.

Оскільки середовище, що розробляється для роботи з різними мовами програмування і фреймворками, то для складання артефакту потрібно підготувати різні оточення.

Для спрощення керуванням тестовим та робочим оточенням, найкраще використовувати технологію контейнеризації. Для зберігання зібраного артефакту (бінарного файлу, набору файлів, контейнера) найкраще використовувати приватні сховища, щоб сторонні особи не отримали доступу до різних версій програми.

На рис. 2.1 зображено загальну схему роботи системи тестування та збирання додатків.

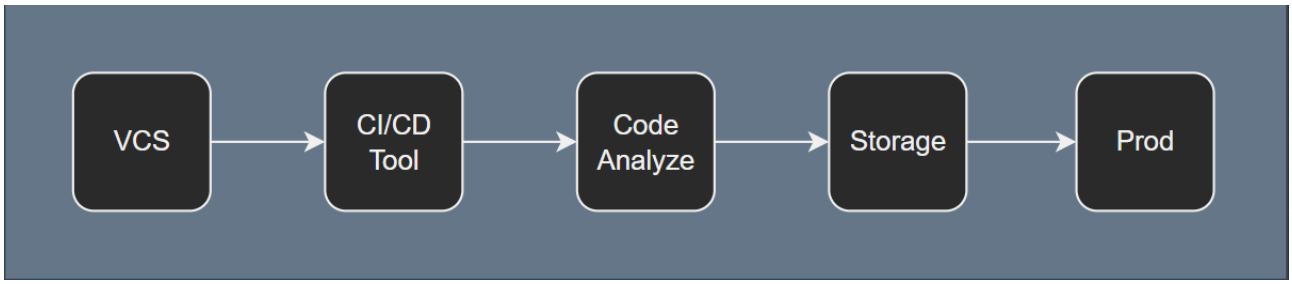

Рисунок 2.1 – Загальна схема роботи системи тестування та збирання додатків

Таким чином, виходячи з проведеного аналізу, робота розроблюваної системи знадобляться такі незалежні системи, які будуть взаємодіяти:

- Система контролю версій. Допогає працювати одночасно багатьом розробникам та не заважати при цьому іншим. Кожен виділяє свою гілку від основної та робить зміни у ній, не заважаючи цим іншим та без погроз зламати основний код;
- Система CI/CD. Для автоматичного запуску збірки програмного продукту. Запускає одну систему за іншою та робить процес безперервним та автоматичним;
- Сховище артефактів. Воно потрібно для того, щоб зберігати артефакти програмного забезпечення в приватному сховищі та зберігати артефакти проекту у різних версіях;

- Статичний аналізатор коду. Він буде виконувати перевірку відповідності коду, який є у проекті згідно до правил написання коду;
- Система оркестрації контейнерами.

2.2 Інтеграція та розгортання

CI/CD – це досить нова концепція розробки програмного продукту, яка поєднує кілька етапів DevOps. Дана методологія збільшує швидкість складання, зводить до мінімуму помилки і покращує якість продукту, що розробляється.

Основні принципи CI/CD полягають у тому, що учасники команди розробників та кінцевий споживач продукту працюють спільно. Дизайнери роблять зміни у зовнішньому вигляді, розробники проектують різні функції для продукту, девопси налагоджують складання та розгортання продукту, а споживачі мають дати відгуки про використання цього програмного забезпечення.

CI – безперервна інтеграція, методологія, яка під час впровадження у проект покращує якість продукту в рази. За рахунок того, що розробники часто роблять комміти, зазвичай прийнято робити коментарі до своїх змін у коді щодня, таким чином зручно відстежувати помилки відразу. За таких малих змін простіше знайти те, що призводить до проблеми, які дефекти та помилки з'явилися. Якщо код пишеться відразу великою частиною, потім знайти в чому проблема набагато складніше і потрібно переглядати безліч рядків коду, витрачати багато часу і окремо тестувати кожну маленьку частину. Робота короткими частинами допомагає уникнути зміни одного і того ж коду декількома розробниками, що захищає нас від проблем у майбутньому при злитті гілок.

CD має два значення Continuous Delivery і Continuous Deployment – безперервна доставка та безперервне розгортання. Ця частина проводить автоматичну доставку та автоматичне розгортання версії продукту на будь-які

середовища оточення: production-серверах, або в середовищі тестування та розробки.

Дана методологія підходить для тих компаній і проектів, в яких вносяться часті зміни розробки програмного продукту, але при цьому регулярно випускають нові стабільні релізи. Розробникам можна повністю зосередитися на розробці та поліпшенні якості програмного забезпечення, що розробляється, оскільки вони отримують автоматично налаштовані процеси тестування, що допомагають виловлювати помилки та проблеми на ранніх етапах, а також автоматичні процеси розгортання продукту розробки.

2.3 Структура та компоненти системи

Основними елементами у процесі розробки програмного забезпечення з допомогою методології CI/CD можна назвати такі частини.

Одним з найважливіших компонентів у програмному комплексі, що розробляється, є система контролю версій. Система контролю версій (VCS)[12] або система керування версіями – централізоване сховище вихідного коду. На даний момент більш ніж 99% VCS, які використовуються для зберігання вихідного коду, є системи засновані на Git. Крім зберігання файлів вихідного коду, такі системи дозволяють версіонувати кожен файл, зберігати стан змін, створювати запити на зміну, додавати коментарі до змін (що позитивно впливає на процес розробки), тегувати зміни.

Крім версіонування, важливим компонентом будь-якої системи контролю версій є створення незалежних гілок, за допомогою яких кожен користувач може створювати необмежену кількість незалежних версій вихідного коду для подальшого з'єднання з основним кодом.

Сучасні системи контролю версій намагаються не відставати від трендів популярних технологій і запроваджують у своїх продуктах додаткову функціональність, наприклад GitLab пропонує своїм користувачам додавати

CI/CD функціональність, пов'язану з тим чи іншим репозиторієм. У загальному вигляді схема роботи із системою зберігання версій наведена рис. 2.2.

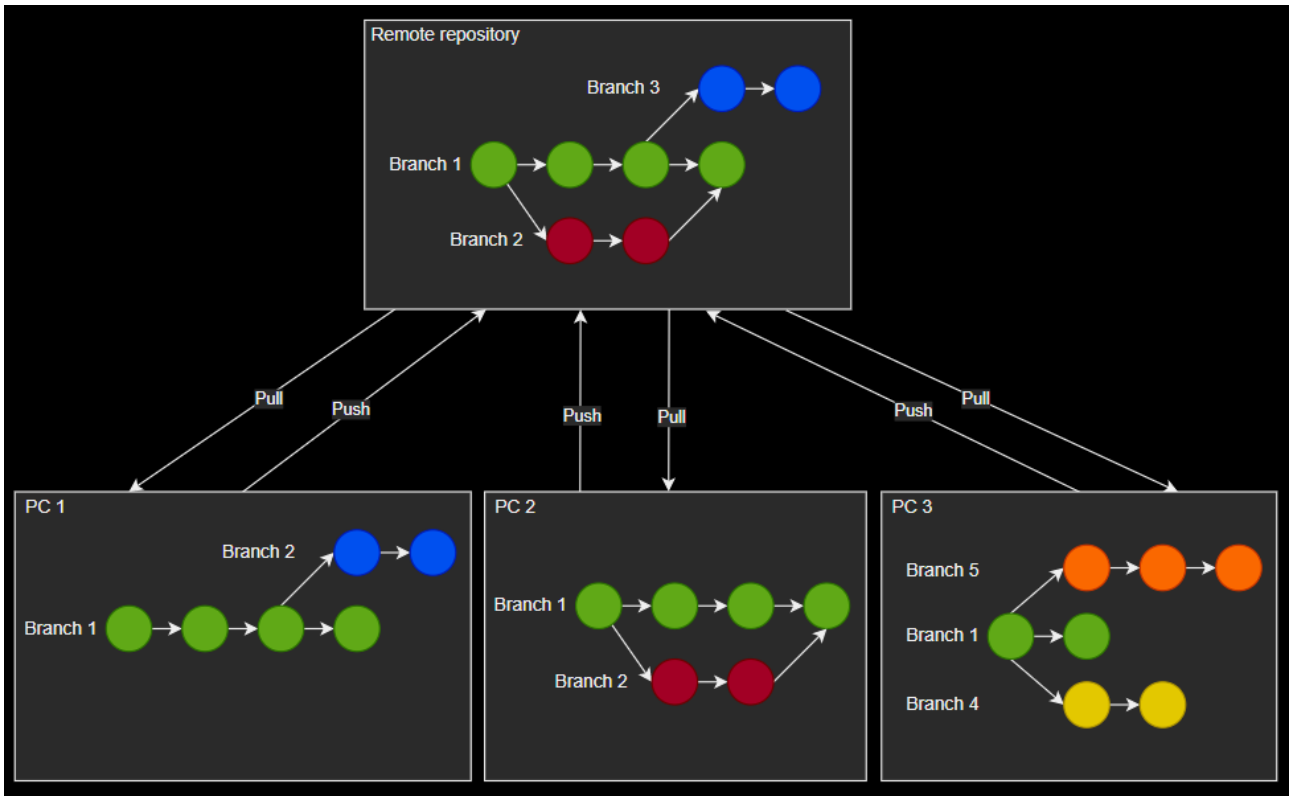

Рисунок 2.2 – Схема роботи із системою зберігання версій

Основна ідея в тому, що є можливість створювати гілки – незалежні розділи, до яких вносяться зміни. Кожен розробник має доступ до гілок у центральному репозиторії, і навіть може створювати свої, завантажувати в центральний репозиторій і з'єднувати гілки між собою.

Наступний компонент, що розробляється, один з найважливіших, тому що саме він буде в центрі всієї системи, займатися запуском всіх компонентів і передачею їм необхідних параметрів - CI/CD утиліта.

Наступний компонент системи – статичний аналізатор коду. Етап аналізу коду дозволяє перевірити, що створюваний продукт повністю відповідає стандартам, не містить помилок, критичних вразливостей. На цьому етапі програма не компілюється, аналізатор перевіряє код, порівнюючи його із заздалегідь визначеними правилами. Правила поділяються за мовами

програмування, тут можна налаштувати різні патерни свого проекту. Після перевірки, як правило, результат порівнюється зі встановленими стандартами і на виході ми отримуємо звіт, в якому є детальна інформація та відповідь – чи пройшов код перевірку. Крім статичних аналізаторів коду, існують лінтери, для перевірки певних типів файлів.

Сховище артефактів можна використовувати як мережеве сховище (папку мережі, віддалений сервер), так і спеціалізоване програмне забезпечення. Основна умова для сховища – воно має підтримувати зберігання бінарних файлів та контейнерів, а також мати можливість версіонування для можливості перемикання між різними патчами.

Останній компонент системи – контейнери та система оркестрації контейнерами. Перші спроби контейнерних технологій зустрічалися ще 1982 року у реалізації chroot в операційних системах UNIX. Популярність ця технологія набула в минулому десятилітті в реалізації у вигляді програмного забезпечення Docker. Головна причина популярності цієї технології – збільшення кількості залежностей у програмних продуктах, що призвело до проблем настроювання оточення та сильного ускладнення міграції додатків з одного сервера на інший. Ще однією перевагою є швидке та легке масштабування додатків як вгору, так і вниз.

Контейнеризація дозволяє зробити ізоляцію на рівні простору імен, сховища та мережі. Використовуючи контейнерні технології, немає необхідності щоразу встановлювати необхідне ПЗ та налаштовувати залежності, досить просто запустити контейнер, який ніяк не впливає на оточення запуску, він однаково прогнозовано запуститься та відпрацює на різних операційних системах незалежно від їх настроювання.

Використання системи оркестрації контейнерами не є обов'язковим елементом, проте вона сильно спрощує керування системою контейнерів і дозволяє легко масштабуватись, взявши на себе більшість завдань.

По-перше, у традиційному розумінні один контейнер - одна програма, це дозволяє домогтися безпеки, за рахунок обмежування спілкування між

додатками, а також легко і швидко змінювати версії програм, просто перезапустивши контейнер, з новою версією. По-друге, якщо потрібно буде швидко додати кілька нових контейнерів, для збільшення продуктивності системи - не потрібно буде створювати їх вручну, а просто передати команду для додавання деякої кількості контейнерів, або налаштувати автоматичне масштабування, при збільшенні навантаження на контейнер.

Підсумовуючи, можна дійти висновку, що у загальному вигляді алгоритм та послідовність кроків виглядають так.

1) На початку в системі контролю версій розробники мають репозиторій, у якому знахотяться всі необхідні файли проекту.
2) Розробник робить зміну в коді та створює запит на зміну в системи контролю версій.
3) Спрацьовує webhook, який запускає роботу сценарію з перевірки вихідного коду, який перевіряє на правильність написання коду згідно до існуючих правил.
4) Після перевірки вихідного коду в системі контролю версій додається оцінка перевірки, тобто бачимо пройшов код перевірку чи ні.
5) Після підтвердження запиту на зміну спрацьовує webhook, який запускає сценарій складання програми (включаючи етап тестів інтегрованих у ту чи іншу мову програмування).
6) Після успішного збирання отриманий артефакт міститься в сховищі артефактів, в систему контролю версій додається тег з поточною версією програми. Версіонування важливо для того, щоб розробники знали в якій версії які функції вже інтегровані.
7) Після закінчення збирання, за тригером запускається сценарій тестування (у новому оточенні запускаються сценарії інтеграційного тестування).
8) Після успішного тестування, спрацьовує тригер на розгортання програми у виробничому середовищі.

Усі описані кроки та залежності представлені рис. 2.3

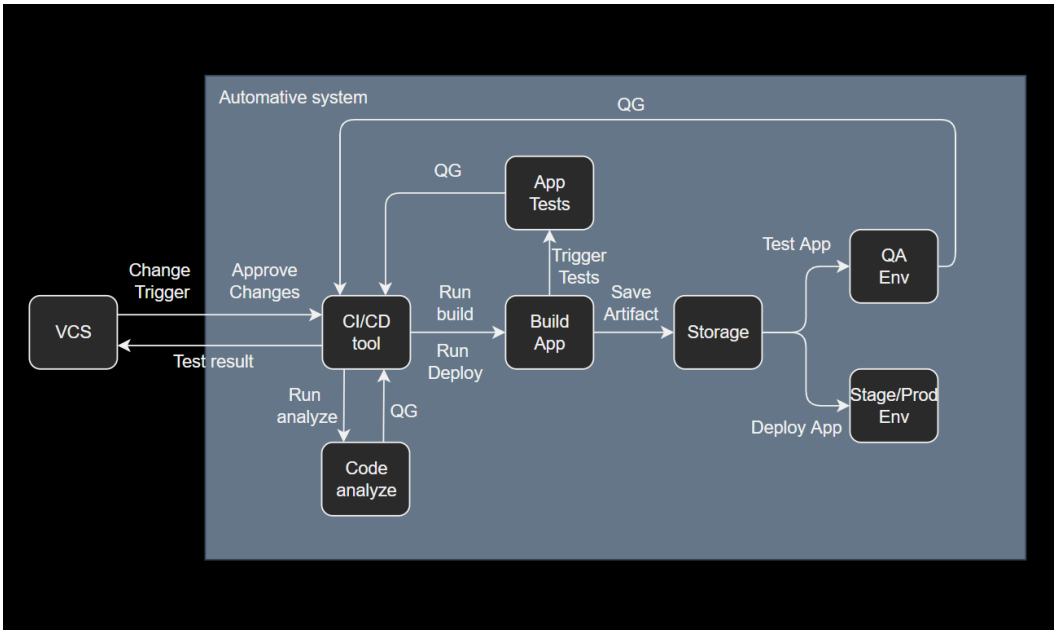

Рисунок 2.3 – Схема роботи системи тестування та збирання додатків

2.4 Висновки до другого розділу

В другому розділі було розроблено глобальний концептуальний аналіз проектованої системи.

Опис та вимоги до системи. Система повинна спрощувати життя розробників, робити процесс розробки монолітних та мікросервісних додатків простіше та якісніше при меншій досвідченості персоналу по розробці програмного забезпечення. Зараз існує проблема з кадрами, бо потрібно знати багато інформації та вміти використовувати багато інструментів та технологій для досягнення необхідного результату. Основні вимоги до системи це просте використання системи та робота декількох незалежних систем для розробки програмного продукту.

Проведене описання компонентів системи, які мають бути у проекті. Тож для розробки цієї системи потрібні: система контролю версій, система CI/CD, сховище артефактів, статичний аналізатор коду та система оркестрації контейнерами.

Описано структуру системи. Більше детально розглянуто як незалежні системи повинні взаємодіяти одна з одною, в якоій послідовності вони запускаються і для чого потрібні.

Детально розглянуто для чого потрібна CI/CD система та етапи її роботи. Описано окремо про безперервну інтеграцію, безперервну доставку та безперервне розгортання програмних продуктів, для чого це потрібно і як працює.

Розглянуто аналізатор коду ті його робота. Для якісного та коректно працюючого коду не обійтись без цього компоненту системи. Він аналізує розроблюваний продукт на відповідність стандартам, цо він не містить помилок та критичних вразливостей. Також він аналізує код на відповідність до правил, які поділяють в залежності від мови програмування.

Детально описано для чого існує контейнери та система оркестрації. Контейнеризація – це технологія, яка дозволяє без прив’язки до апаратного чи програмного забезпеченя розробляти програмне забезпечення, запускати його прогнозовано та не впливаючи на пристирій на якому це запущено.

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Вибір інструментальних засобів

У попередньому розділі було проведено аналіз необхідних для роботи системи компонентів, і обрані основні елементи, ґрунтуючись на яких можна зробити вибір програмних засобів, для реалізації системи. Однак, варто відзначити, що зараз на ринку безліч засобів реалізацій програмного забезпечення для вирішення кожного завдання, а глобальна стандартизація продуктів призвела до того, що більшість ПЗ має схожу функціональність, тому при виборі того чи іншого продукту все здебільшого зводиться до переваг фахівця, яка займається розробкою архітектури, і тому, наскільки він знайома з цим додатком. Список продуктів, необхідний для вирішення завдань, поділений за категоріями наведено рис. 3.1

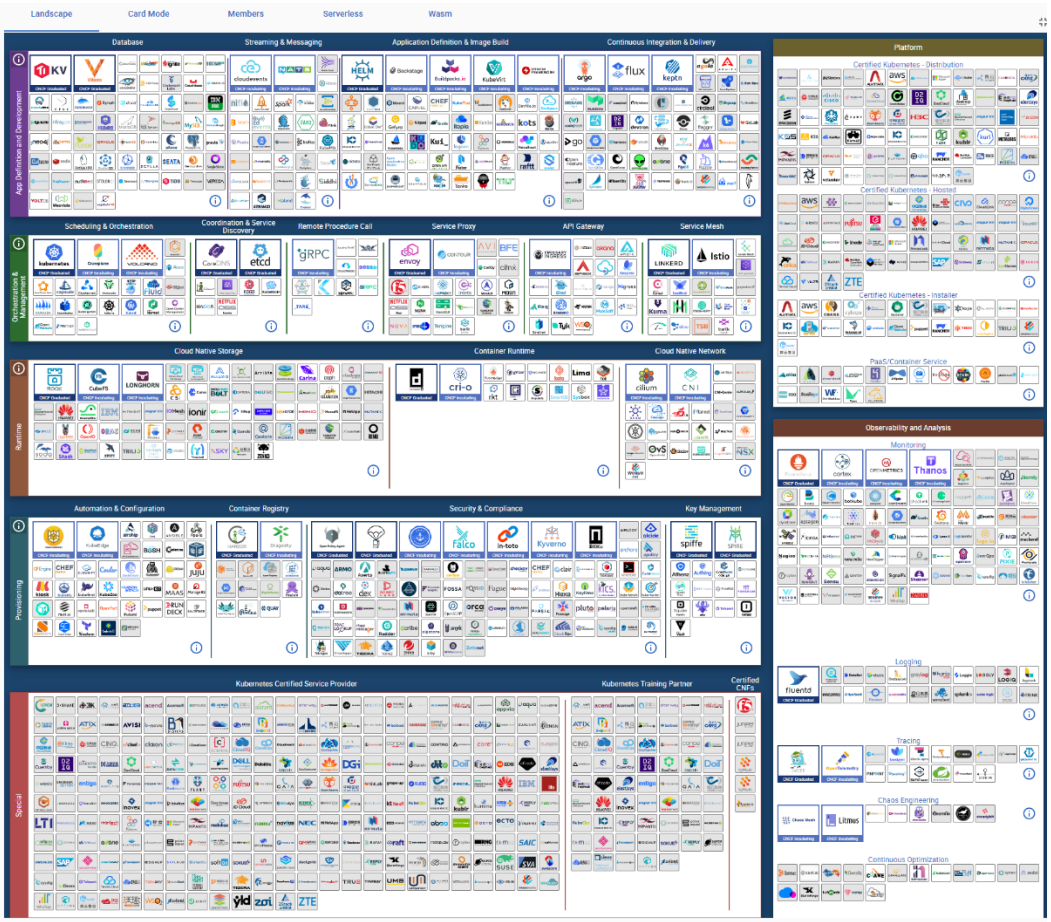

Рисунок 3.1 – Інструменти системного інженера

На даний момент на ринку ПЗ є 2 основних представники, що займаються технологією контейнеризації Docker і Kubernetes CRI[13].

При виборі кінцевого продукту для застосування в системі, варто зазначити, що процеси, які використовуються при роботі з контейнерними технологіями у Docker та Kubernetes CRI[14] дуже схожі,, вони зображені на рис. 3.2.

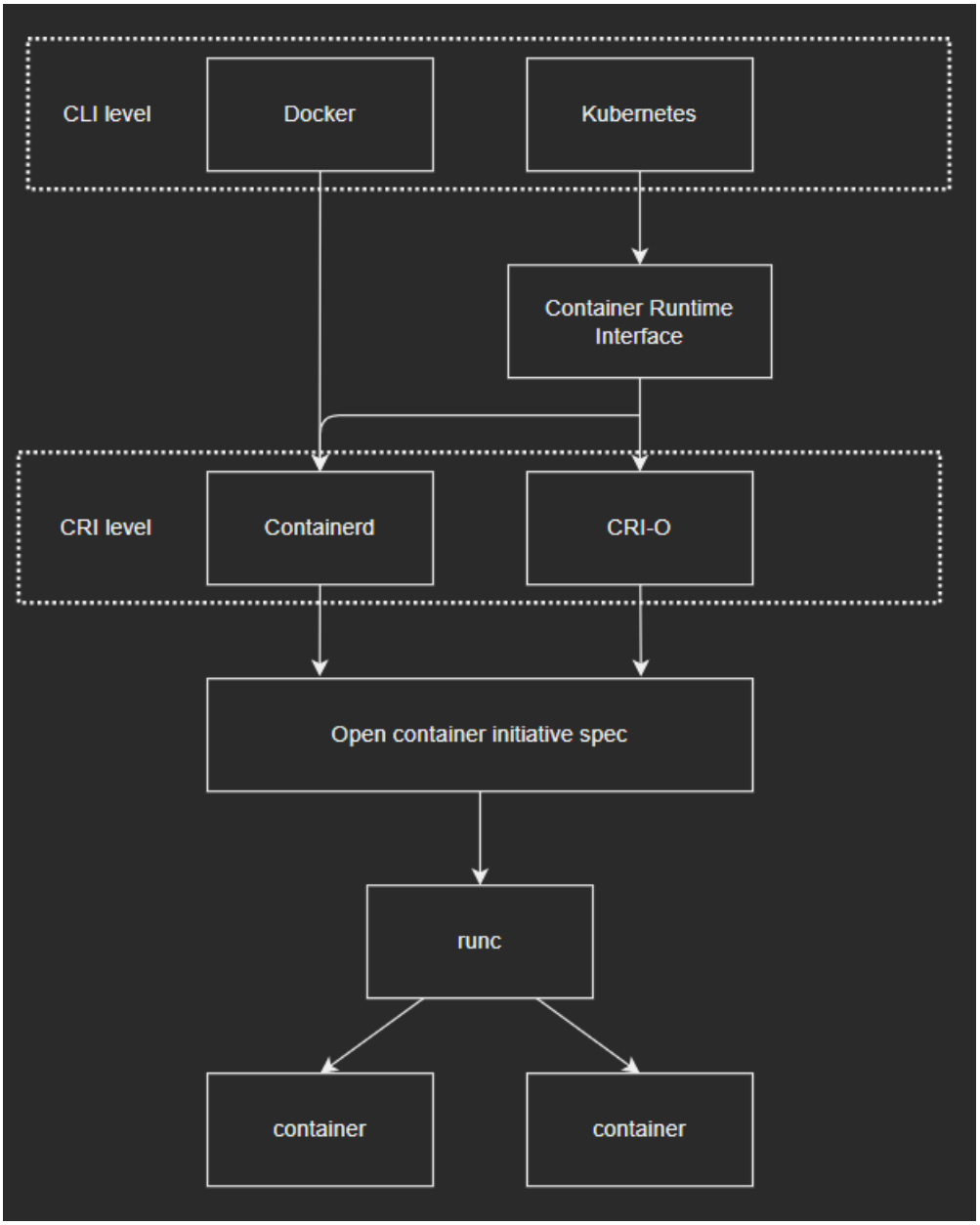

Рисунок 3.2 – Порівняння роботи docker та Kubernetes CRI

Docker складається з декількох компонентів: docker-cli - утиліта для роботи з docker, приймає команди з CLI і передає їх у двигун docker, containerd - демон для роботи з контейнерами, в його завдання входить завантаження з віддалених репозиторіїв та в репозиторії, передача команд на запуск контейнерів та runc який займається запуском контейнерів. Також компанія docker має інструмент docekrhub – мережеве сховище образів контейнерів, з можливістю завантажувати свої модифіковані контейнери, що є явною перевагою серед конкурентів.

Kubernetes CRI тривалий час звертався до контейнерів використовуючи своє API як прошарок, проте в останній версії вони відступили від цього методу і повністю відмовилися від підтримки свого API та движка docker, тепер Kubernetes CRI безпосередньо взаємодіятиме з containerd.

Новий підхід дозволить прибрати 2 додаткових кроки, які використовувалися до цього моменту, бо це було досить незручно, тому що компанія docker не надавала API або інші методи для роботи зі своїми технологіями. Починаючи з патча 1.23 kubernetes припиняє підтримку docker у своєму продукті, і це дозволило прискорити роботу їхнього програмного забезпечення при використанні з великою кількістю контейнерів.

У сфері рішень оркестрації контейнерів на ринку програмного забезпечення є 2 продукту: docker swarm – нативне рішення від компанії docker і Kubernetes – open source продукт від компанії Google.

Docker swarm – це нативний інструмент від компанії Docker, створений для оркестрації контейнерів. Він здатний запускати, контролювати, керувати, знищувати та балансувати навантаження між групою контейнерів. Kubernetes – open source платформа від компанії Google, що дозволяє створювати кластери повного життєвого циклу – створення, видалення, керування, моніторинг, масштабування, автоматизація процесів. У Kubernetes структура кластера складніша, ніж у docker swarm. Перелік переваг та недоліків визначено у табл. 3.1.

Таблиця 3.1 – порівняння docker swarm та Kubernetes

Docker swarm	Kubernetes
Переваги	
Зручне встановлення	Ведення журналу та моніторинг
Сумісність	Швидкість
Швидкість	Декларативна конфігурація системи
Хороша документованість	Масштабування інфраструктури
Контроль версій	Сховище
Недоліки	
Залежність від платформи	Налаштування
Сховище	Міграція
Моніторинг	Сумісність

Грунтуючись на наведених порівнянні та вимогах до продукту, найкращим рішенням у даному випадку буде поєднати переваги описаних вище технологій: використовувати Kubernetes разом з helm як систему оркестрації контейнерами і зберігати готові образи контейнерів у docker hub або хмарному провайдері.

Так як Kubernetes[14] зберігає всю конфігурацію в yaml файлах і кількість вузлів зазвичай занадто велика, то управління такою кількістю інформації буває складним, для цього був створений інструмент helm. Helm – це пакетний менеджер для конфігурацій Kubernetes. Він дозволяє налаштувати залежності, користуватися змінними, зберігати, відстежувати та перемикатися між різними версіями додатків та конфігурацій, що значно спрощує роботу з кластерами Kubernetes.

Далі варто визначитися з вибором хмарного провайдера, це не обов'язкова умова, адже Kubernetes вміє працювати і локально за допомогою утиліти minicube, проте це рішення більше підходить для тестування деяких функцій, ніж повноцінне використання у робочому середовищі, до того ж усі основні представники хмарних провайдерів мають вбудовану підтримку Kubernetes кластерів. На даний момент на ринку присутні 3 основні хмарні провайдери:

- AWS
- GCP
- AZURE

На момент 2022 року, їх функціональність здебільшого ідентична, і для вирішення проблем продукту, що розробляється, підходять всі 3, тому питання вибору буде зводитися до порівняння їх тарифів за використання ресурсів.

Порівнювати необхідно за компонентами, які використовуватимуться у роботі системи. Порівняння вартості сервісів подано у табл. 3.2

Таблиця 3.2 – порівняння вартості сервісів у хмарних провайдерів

Type	Metric	AWS	GCP	AZURE
LoadBalancer	1 hour	0.008	0.007	0.008
Instance	4 CPU, 16Gb mem	0.152	0.18	0.137
Network trafic	1 Gb	0.01	0.01	0.01
Kubernetes Cluster	1 hour	1	1	1
Container storage	1 Gb	0.1	0.11	0.2
Computer storage	1 Gb	0.01	0.01	0.01
Monitoring	1 request	0.01	0.0075	0.0125
Total		1.29	1.3245	1.3775

Виходячи з отриманих даних у табл. 3.2, можна дійти висновку, що оптимальний варіант буде використання хмарного сервісу AWS.

Після закінчення вибору технологій, на яких буде побудовано платформу, необхідно вибрати програмні засоби, на базі яких працюватиме система. Першим компонентом буде інструмент CI/CD. Як вже писалося на початку поточного розділу, ринок інструментів для вирішення тих чи інших завдань системних інженерів перенасичений і часто вибір робиться на користь програм, з якими був позитивний досвід роботи. Однак одні з лідерів популярності на ринку CI/CD інструментів на даний момент є:

- Jenkins;

- Circle CI;
- Gitlab CI;
- Bamboo;
- TeamCity.

Основні відмінності даних інструментів можна побачити у табл. 3.3

Таблиця 3.3 – Основні відмінності даних інструментів

	Jenkins	CircleCI	TeamCity	Bamboo	Gitlab CI
Open Source	+	-	-	-	-
Складність конфігурації	3/5	3/5	3/5	3/5	3/5
Вбудовані функції	2/5	4/5	4/5	4/5	4/5
Інтеграція	5/5	3/5	4/5	3/5	4/5
Хостинг	Локально, Хмара	Локально, Хмара	Локально	Локально, Хмара, Bitbucket	Локально, Хмара
Безкошт.	+	+	+	+	+
Вартість агента	Безк.	Від 39$/міс	Від 200$ Одноразово	Від 200$ Одноразово	Від 4$/міс
Підтрим. ОС	Windows, MacOS, UNIX like systems	Linux, MacOS	Windows, MacOS, Linux	Windows, MacOS, Linux	Linux

Ґрунтуючись на даних табл. 3.3, практично по всіх пунктах кращі або рівні параметри утиліти Jenkins. У таблиці у неї найменший бал у графі «Вбудовані функції», і справді у неї найменша функціональність після інсталяції, проте той факт, що це програмне забезпечення є open source, призвело до того, що у Jenkins дуже велика і активна спільнота. Люди та корпорації розробили безліч плагінів для різних потреб, за допомогою яких можна створити повноцінний цикл CI/CD, провести інтеграцію із системами

контролю версій та іншими інструментами для роботи з кодом, такими як Sonarqube, Azure, AWS, GCP.

Далі слід вибрати програмне забезпечення для статичного аналізу коду та перевірки різних файлів на відповідність стандартам. На даний момент на ринку ПЗ у цій сфері є універсальне рішення, яке використовується у всіх проектах – Sonarqube. Sonarqube – це проект із відкритим вихідним кодом, який проводить статичний аналіз коду, здатний працювати з усіма сучасними мовами програмування, використовуючи вбудовані правила[15].

Основна перевага даного продукту – можливість повної конфігурації та створення правил, щоб тестові сценарії підходили під певний проект. Ви можете налаштувати кількість і строгість правил для всіх мов програмування, що використовуються в проекті. На рис. 3.3 наведено приклад вікна створення правил у Sonarqube.

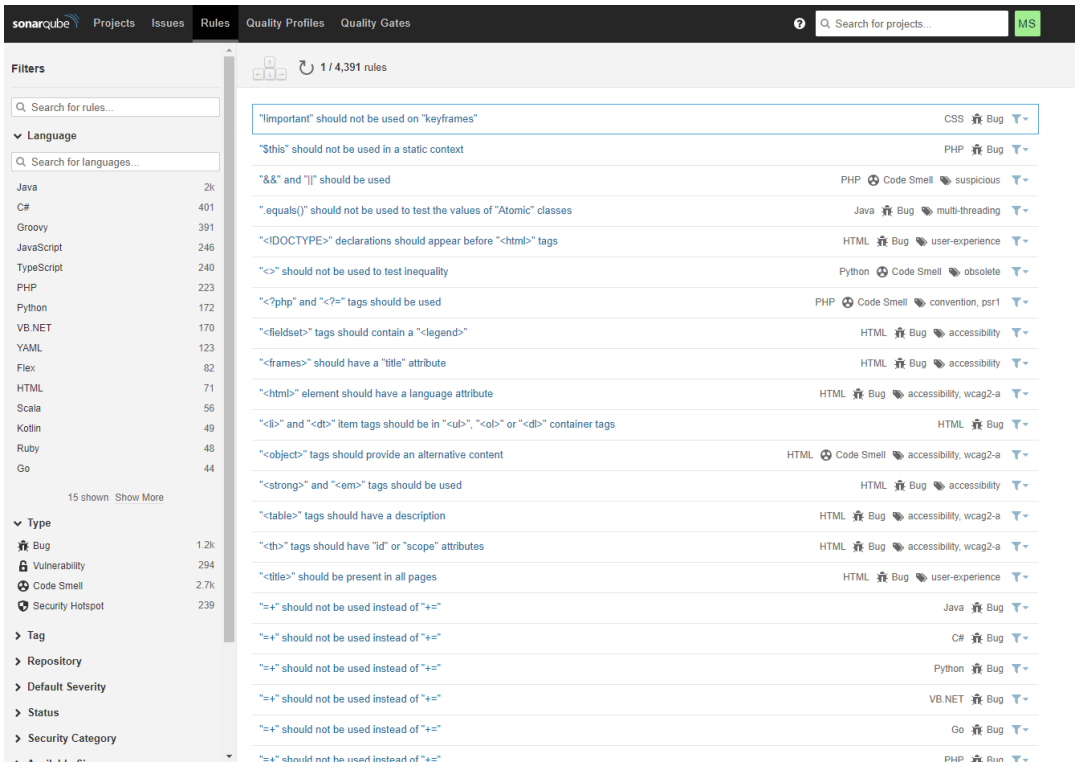

Рисунок 3.3 – Вікно додавання політик sonarqube

Як видно на зображенні, в самому ПЗ можна додати правило, вибрати тип помилки, якщо частина коду не пройде перевірку (баг, вразливість, проблема безпеки), а також критичність, тег та інше.

Також, оскільки в проекті будуть використовуватися технології контейнеризації docker і helm, то будуть присутні докерфайли та хелмфайли, для них існують спеціальні лінтери, які перевіряють чи відповідає формат документів прийнятим стандартам для написання таких типів файлів – dockerlint та helm-lint.

Останнім компонентом у цій системі буде сховище артефактів. Як уже описувалося в пункті 2.2, основна вимога до цього програмного забезпечення – воно має вміти зберігати бінарні версії додатків та докер-образи. Таким рішенням є програмне забезпечення від компанії Sonartype – Nexus. Воно дозволяє зберігати різні типи ресурсів – бінарні файли, архіви та набори папок та файлів, пакети apt, pip, yum, helm chart, докер образи та інші.

3.2 Налаштування компонентів системи

На етапі налаштування компонентів буде здійснено будуть створені та налаштовані всі сценарії, аналізатори кода, агенти для збирання різних типів додатків, сховища, інтеграції із системами контролю версій, а також внутрішні інтеграції між додатками.

Налаштування варто почати з утиліти Jenkins[16], оскільки саме вона буде центральним і сполучним елементом системи між усіма її вузлами. Після інсталяції (етап пропущений, тому що буде описаний у пункті 3.3), перед користувачем з'являється головне вікно програми, зображене на рис. 3.4.

Тут можна додати нові сценарії роботи та перейти до меню конфігурації. Для початку потрібно зробити початкове налаштування - встановити плагіни, налаштувати запуск на необхідних агентах, додати секрети для авторизації у зовнішніх ресурсах.

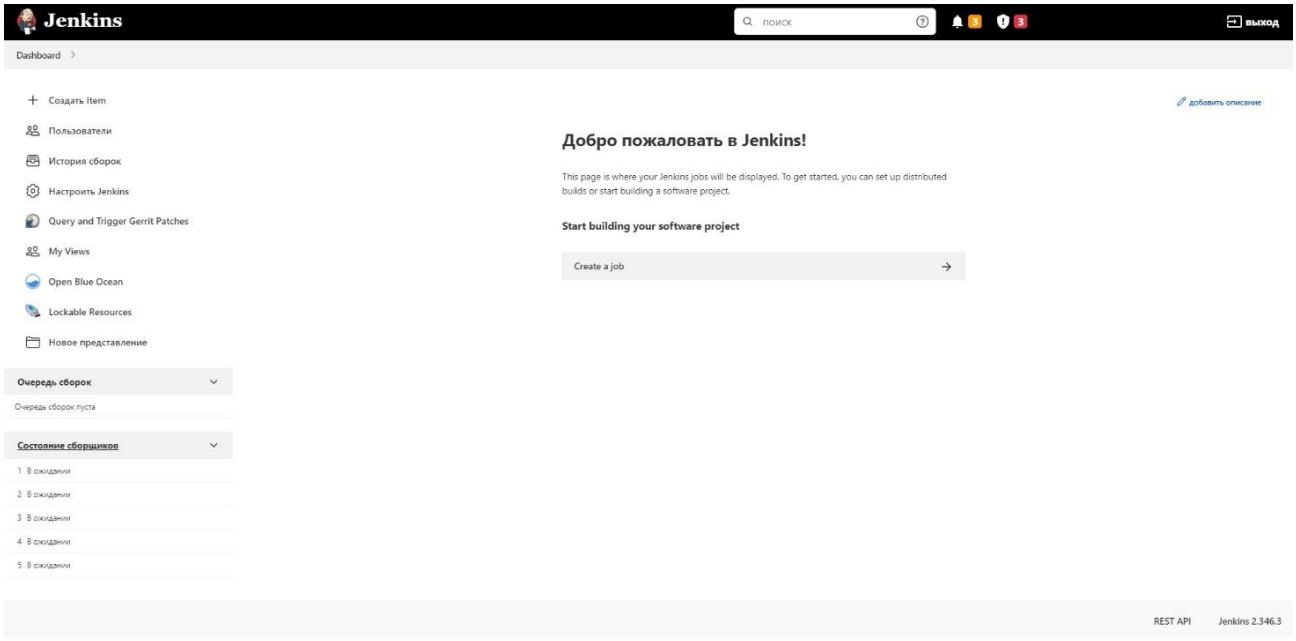

Рисунок 3.4 – Головне меню Jenkins

Для інтеграції з усіма компонентами необхідно встановити такі плагіни:

– JCASC (Jenkins Configuration as Code) – утиліта для запису всіх конфігурацій системи у вигляді yaml файлу як backup інструменту, та завантаження конфігурації з yaml файлу як restore компонент. Корисна функція при використанні Jenkins усередині контейнерів, дозволяє підготувати конфігурації, і щоразу запускати налаштовану майстер ноду;

– JCASC-Jobs – додаток до плагіну jcasc, що дозволяє підключати сценарії під час завантаження агента;

– Gitlab та Github – 2 плагіни для створення інтеграцій системи Jenkins із системами контролю версій gitlab та github. Дозволяють налаштувати тригери для автоматичного запуску сценаріїв за попередньо налаштованими подіями, додавати теги до VCS, додавати статуси кроків виконання завдань до системи контролю версій;

– Gitlab webhook triger дозволяє налаштовувати поведінку вебхуків для роботи з системами контролю версій;

– Active choice parameter – дозволяє параметризувати сценарії, додаючи скрипти як параметри запуску, корисно для створення параметрів на основі даних, що динамічно змінюються (наприклад версії збірок, які

генеруються автоматично і підставляються в сценарій після зчитування даних з репозиторію);

– Kubernetes – плагін дозволяє додавати динамічних агентів, що створюються всередині кластера Kubernetes, налаштовувати технічні характеристики кожного агента, їх властивості, параметри запуску;

– Pipeline – дозволяє створювати настроюванні сценарії з поділом по кроках;

– Credentials – дозволяє зберігати секрети в Jenkins, для подальшого використання їх у сценаріях користувача;

– Sonarqube – дозволяє додати інтеграцію з компонентом sonarqube, для авторизації, та зіставлення перевірок коду зі сценаріями користувача.

Далі необхідно зробити конфігурації встановлених плагінів та базове налаштування системи. Попередньо необхідно створити секрети в меню credentials для систем github, gitlab, sonarqube. Для цього необхідно перейти в меню configure jenkins і далі в меню system configuration. У цьому розділі налаштовуються базові елементи та всі плагіни, крім тих, що відповідають за налаштування безпеки.

Для початку варто заповнити інформацію для плагіна "gitlab integration". Тут можна додати необмежену кількість з'єднань:

– connection name - асоціативне ім'я, яке буде використовуватися в пайплайнах;

– gitlab host URL - кореневе ім'я gitlab сервера, з яким налаштовуватиметься інтеграція;

– credentials - ім'я токена, який доданий в дженкінс, для доступу до gitlab.

Далі налаштування плагіна "github". Тут також можна додати необмежену кількість з'єднань:

– name - асоціативне ім'я для гіт сервера;

– api URL - кореневе ім'я для github сервера;

– credentials - ім'я токена для доступу до github серверу.

Останнім буде налаштовано плагін Sonarqube:

– Name - асоціативне ім'я, для використання в пайплайнах;
– Server url - кореневе ім'я sonarqube сервера;
– Server authentication token - API токен для доступу до sonarqube серверу.

Далі слід додати динамічні агенти для запуску пайплайнів. Це в першу чергу необхідно для зняття великих навантажень з майстер агента, в такій конфігурації майстер нода займатиметься лише прослуховуванням вебхуків, тригером завдань та запуском необхідних агентів з передачею їм параметрів пайплайну. Запуск пайплайнів на окремих динамічних вузлах також допомагає підготувати конфігурацію вузлів відповідно до вимог того чи іншого оточення і запускати пайплайни на вже заготовлених агентах.

Конфігурація здійснюється в меню "configure Jenkins" - "nodes" - "configure clouds". Тут необхідно заповнити поля для інтеграції з кластером Kubernetes і запуском агентів як під Kubernetes.

– Name – асоціативне ім'я;
– Kunernetes url – адреса Kubernetes кластера;
– Kubernetes server certificate key – сертифікат для інтеграції з kubernetes;
– Kubernetes namespace – ім'я простору імен, у якому запускатимуться агенти;
– Credentials – секрети для підключення до кластера;
– Jenkins URL – адреса Jenkins, за якою Kubernetes надсилатиме дані;
– Jenkins tunnel – порт агентів;
– Pod Templates – тут додаються агенти, додати можна необмежену кількість
– Pod Template/Name – ім'я шаблону;
– Labels – мітки, за якими викликаються агенти усередині пайплайнів;
– Usage – використання при всіх стартах пайплайнів або тільки якщо агент викликаний по полю label;

- Docker image – ім'я репозиторію та ім'я образу, який буде використовуватися для запуску агента;
- Working directory – робоча директорія для пайплайнів;
- Environment Variable – поле для змінних оточення контейнера.

Після завершення всіх конфігурацій можна приступити до створення та налаштування пайплайнів. Для роботи системи необхідно створити 3 пайплайни для кожної мови програмування: перевірка, збирання, публікація. На даному етапі будуть створені повні цикли для java8, java11, python, go, npm. На першому етапі необхідно створити шаблон пайплайну для перевірки коду. Приклади всіх пайплайнів, сценаріїв тощо будуть для java11, решта коду приведена у додатоку Б. У ньому будуть присутні такі кроки:

- Init – підготовчий крок, тут будуть створені всі необхідні дерикторії, підготовлені змінні оточення, налаштовані залежності;
- Checkout – клонування git-репозиторію з проектом у робочу директорію Jenkins;
- Git version – зчитування файлу pom.xml, пошук змінної з номером версії, запис версії у змінну оточення;
- Sonar – запуск команди для сканування робочої директорії проекту статичним аналізатором коду sonarqube;
- Docker-lint – лінтинг докерфайлу;
- Helm-lint – лінтинг хелм чарту;
- Check-Commit – опціональний крок, перевірка коміт повідомлення на відповідність правилам проекту;
- ProjectStatus - відправка в систему контролю версій тега, зі станом сценарію збірки, що пройшов, в залежності від того, успішний пайплайн чи ні.

Крок підготовки представлений рис. 3.5

```
stages {
    stage('Init') {
        agent {
            any
        }
        steps {
            updateGitlabCommitStatus(name:'Init', state:'running')
            echo '------------------------------Init------------------------------'
            sh 'rm -rf *'
            sh 'mkdir project'
            updateGitlabCommitStatus(name:'Init', state:'success')
        }
    }
```

Рисунок 3.5 – Крок підготовки Init

Крок клонування та валідації коміт повідомлення представлений рис. 3.6

```
stage('Checkout') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Checkout', state:'running')
        echo '------------------------------CHECKOUT------------------------------'
        dir('project') {
            git branch: 'main', url: 'git@{GIT_URL}:{PROJECT_OWNER}/{PROJECT_NAME}.git', credentialsId: '{GIT_CREDENTIALS}'
        }
        updateGitlabCommitStatus(name:'Checkout', state:'success')
    }
}
stage('CheckCommit') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Check-Commit', state:'running')
        echo '------------------------------Check-Commit------------------------------'
        dir('project')
        {
            def pattern = COMMIT_MSG_PATTERN
            def message = getLastCommitMessage
            if (!commitMessage(pattern, message)) {
                script.error "Commit message not valid!"
            }
        }
        updateGitlabCommitStatus(name:'Check-Commit', state:'success')
    }
}
```

Рисунок 3.6 – Крок клонування

Крок зчитування версії та перевірки утилітою sonarqube представлений рис. 3.7

```
stage('Git version') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Git version', state:'running')
        echo '------------------------------Git version------------------------------'
        dir('project')
        {
            version = script.sh(
            script: "cat pom.xml | grep -Poh '<parent.version>' || echo \"\"",
            returnStdout: true
            )
        }
        updateGitlabCommitStatus(name:'Git version', state:'success')
    }
}
stage('Sonar') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Sonar', state:'running')
        echo '------------------------------Sonar------------------------------'
        dir('project')
        {
            withSonarQubeEnv(installationName: 'sq1') {
                sh 'mvn clean verify sonar:sonar'
            }
        }
        updateGitlabCommitStatus(name:'Sonar', state:'success')
    }
}
```

Рисунок 3.7 - Крок зчитування версії та перевірки утилітою sonarqube

Крок лінтингу докерфайлу та helm чарту представлені на рис. 3.8

```
stage('Docker-lint') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Docker-lint', state:'running')
        echo '-----------------------------Docker-lint-------------------------------'
        dir('project')
        {
            script.sh "hadolint Dockerfile"
        }
        updateGitlabCommitStatus(name:'Docker-lint', state:'success')
    }
}
stage('Helm-lint') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Helm-lint', state:'running')
        echo '-----------------------------Helm-lint-------------------------------'
        dir('project')
        {
            script.sh 'ct lint --charts deploy-templates/'
        }
        updateGitlabCommitStatus(name:'Helm-lint', state:'success')
    }
}
```

Рисунок 3.8 – Крок лінтингу докерфайлу та helm чарту

Останній крок з тегуванням поточної версії представлений на рис. 3.9

```
post {
    success{
        echo '-----------------------------POST SUCCESS-------------------------------'
        git tag -m 'Review success'
        updateGitlabCommitStatus(name:'ProjectStatus', state:'success')
    }
    failure{
        echo '-----------------------------POST FAILURE-------------------------------'
        git tag -m 'Review failure'
        updateGitlabCommitStatus(name:'ProjectStatus', state:'failed')
    }
}
```

Рисунок 3.9 – Крок з тегуванням поточної версії

Після завершення написання пайплайну для перевірки вихідного коду потрібно перевірити його, зробивши тестовий запуск, і здійснити аналогічні пайплайни для інших мов програмування, необхідно замінити дії в кроках git

version, sonar. Приклад роботи пайплайну для збирання представлений на рис. 3.10.

Рисунок 3.10 – Робота пайплайну

Далі потрібно скласти пайплайн для складання артефакту, відправлення його у файлове сховище, складання докеру образу і надсилання його в nexsus. Пайплайн складатиметься з таких кроків:

- Init – підготовчий крок, тут будуть створені всі необхідні дерикторії, підготовлені змінні оточення, налаштовані залежності;
- Checkout – клонування git-репозиторію в робочу дерикторію;
- Git version – зчитування файлу pom.xml, пошук змінної з номером версії, запис версії у змінну оточення;
- Build – етап компіляції програми для отримання артефакту;
- Build-image – етап складання докеру образу із зібраним артефактом;
- Push-to-nexus – відправка артефакту до сховища nexus;
- Push-image-nexus – надсилання образу в сховище nexus.

Крок підготовки, клонування представлений на рис. 3.11

```
stage('Init') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Init', state:'running')
        echo '------------------------------Init------------------------------'
        sh 'rm -rf *'
        sh 'mkdir project'
        updateGitlabCommitStatus(name:'Init', state:'success')
    }
}
stage('Checkout') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Checkout', state:'running')
        echo '------------------------------CHECKOUT------------------------------'
        dir('project') {
            git branch: 'main', url: 'git@{GIT_URL}:{PROJECT_OWNER}/{PROJECT_NAME}.git', credentialsId: '{GIT_CREDENTIALS}'
        }
        updateGitlabCommitStatus(name:'Checkout', state:'success')
    }
}
stage('Git version') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Git version', state:'running')
        echo '------------------------------Git version------------------------------'
        dir('project')
        {
            version = script.sh(
            script: "cat pom.xml | grep -Poh '<parent.version>' || echo \"\"",
            returnStdout: true
            )
        }
        updateGitlabCommitStatus(name:'Git version', state:'success')
    }
}
```

Рисунок 3.11 – Крок підготовки, клонування

Крок складання артефакту та образу представлений на рис. 3.12

```
stage('Build') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Build', state:'running')
        echo '------------------------------Build------------------------------'
        dir('project')
        {
            script.sh 'mvn clean'
            script.sh 'mvn clean install'
        }
        updateGitlabCommitStatus(name:'Build', state:'success')
    }
}
stage('Build-image') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Build-image', state:'running')
        echo '------------------------------Build-image------------------------------'
        dir('project')
        {
            script.sh "/kaniko/executor --context `pwd` --destination {REPO_NAME}/{PROJECT}:latest --destination {REPO_NAME}/{PROJECT}:${IMAGE_TAG}"
        }
        updateGitlabCommitStatus(name:'Build-image', state:'success')
    }
}
```

Рисунок 3.12 – Крок складання артефакту та образу

Крок відправлення артефакту представлений на рис. 3.13

```
def push_artifact(list) {
    list.each { item ->
        echo "Pushing ${item}"
        pom_root = readMavenPom file: "pom.xml";
        pom = readMavenPom file: "${item}/pom.xml";
        filesByGlob = findFiles(glob: "${item}/target/*.${pom.packaging}");
        echo "${filesByGlob[0].name} ${filesByGlob[0].path} ${filesByGlob[0].directory} ${filesByGlob[0].length} ${filesByGlob[0].lastModified}"
        artifactPath = filesByGlob[0].path;
        artifactExists = fileExists artifactPath;
        if(artifactExists) {
        echo "*** File: ${artifactPath}, group: ${pom_root.groupId}, packaging: ${pom.packaging}, version ${pom_root.version}";
        nexusArtifactUploader(
            nexusVersion: NEXUS_VERSION,
            protocol: NEXUS_PROTOCOL,
            nexusUrl: NEXUS_URL,
            groupId: pom_root.groupId,
            version: pom_root.version,
            repository: NEXUS_REPOSITORY,
            credentialsId: NEXUS_ID_CRED,
            artifacts: [
                [artifactId: pom.artifactId,
                classifier: '',
                file: artifactPath,
                type: pom.packaging]
                ]);
            } else {
            error "*** File: ${artifactPath}, could not be found";
        }
    }
}
stage('Push-to-nexus') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Push-to-nexus', state:'running')
        echo '------------------------------Push-to-nexus------------------------------'
        dir('project')
        {
            script {
                push_artifact("${MODULES}".split(' '))
            }
        }
        updateGitlabCommitStatus(name:'Push-to-nexus', state:'success')
    }
}
```

Рисунок 3.13 – Крок відправлення артефакту

Крок відправлення образу представлений на рис. 3.14

```
stage('Push-image-nexus') {
    agent {
        any
    }
    steps {
        updateGitlabCommitStatus(name:'Push-image-nexus', state:'running')
        echo '------------------------------Push-image-nexus------------------------------'
        dir('project')
        {
            script {
                nexusArtifactUploader(
                nexusVersion: NEXUS_VERSION,
                protocol: NEXUS_PROTOCOL,
                nexusUrl: NEXUS_URL,
                groupId: pom_root.groupId,
                version: pom_root.version,
                repository: NEXUS_REPOSITORY_DOCKER,
                credentialsId: NEXUS_ID_CRED,
                artifacts: [
                    [artifactId: pom.artifactId,
                    classifier: '',
                    file: dockerimage,
                    type: image]
                    ]);
            }
        }
        updateGitlabCommitStatus(name:'Push-image-nexus', state:'success')
    }
}
```

Рисунок 3.14 – Крок відправлення образу

Приклад роботи пайплайну для складання представлений на рис. 3.15

Рисунок 3.15 – Робота пайплайну для складання

Після створення пайплайну для збирання образу необхідно змінити крок для збирання артефакту та образу під інші фреймворки і створити пайплайни для інших мов програмування.

3.3 Налаштування інфраструктури системи

Після того, як ми створили та налаштували CI/CD утиліту, можна приступати до налаштування інфраструктури. У роботі буде використовуватися підхід IaC – інфраструктура як код. Всі компоненти системи будуть запускатися всередині кластера kubernetes, відповідно, вся інфраструктура буде описана як файли kubernetes конфігурацій. Для спрощення роботи з kubernetes та параметризації проекту разом із kubernetes використовуватиметься helm[17].

Всередині kubernetes кластера є кілька типів ресурсів, які необхідно налаштувати: deployment – основний компонент, відповідає за створення контейнерів, їх конфігурацію, масштабування, service – компонент, що

відповідає за мережеву взаємодію контейнерів між собою, ingress – вбудований сервер DNS для маршрутизації запитів усередині кластера.

Першим компонентом для налаштування буде sonarqube. Для роботи з ним потрібно завантажити helm chart з публічного репозиторію, налаштувати конфігурацію у value файлі. Всередині deployment визначаються контейнери, їх змінні оточення, аргументи запуску, монтовані директорії та диски, налаштовується replica set або stateful set, які порти будуть відкриті і контейнери, визначаються label для зіставлення з іншими типами компонентів. Можна визначати кілька контейнерів для 1 ресурсу типу deployment, наприклад, 1-й контейнер запустить скрипт і проведе початкове налаштування оточення, а другий контейнер буде з працюючим додатком[18]. Це зроблено для поділу повноважень та економії місця при масштабуванні. Приклад deployment як файлу конфігурації Kubernetes програми наведено на рис 3.16.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: sonarqube
5    labels:
6      app: sonarqube
7  spec:
8    replicas: 1
9    selector:
10     matchLabels:
11       app: sonarqube
12   template:
13     metadata:
14       labels:
15         app: sonarqube
16     spec:
17       containers:
18       - name: sonarqube-scaner
19         image: sonarqube/sonarqube:1.14.2
20         ports:
21         - containerPort: 9000
```

Рисунок 3.16 – Файл конфігурації Kubernetes програми

Однак при використанні helm chart усі аргументи параметризуються, створюючи залежності, і успадковуються або від файлу із загальною конфігурацією для чарту або успадковуються від інших файлів. Частина конфігураційного файлу компонента deployment для sonarqube представлена на рис 3.17.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ template "sonarqube.fullname" . }}
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
    app.kubernetes.io/name: {{ template "sonarqube.name" . }}-{{ template "sonarqube.fullname" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
    app.kubernetes.io/part-of: sonarqube
    app.kubernetes.io/component: {{ template "sonarqube.fullname" . }}
    app.kubernetes.io/version: {{ tpl .Values.image.tag . | quote }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ template "sonarqube.name" . }}
      release: {{ .Release.Name }}
{{- if .Values.deploymentStrategy }}
  strategy:
{{ toYaml .Values.deploymentStrategy | indent 4 }}
{{- end }}
  template:
    metadata:
      labels:
        app: {{ template "sonarqube.name" . }}
        release: {{ .Release.Name }}
{{- with .Values.podLabels }}
{{ toYaml . | indent 8 }}
{{- end }}
      annotations:
        checksum/init-sysctl: {{ include (print $.Template.BasePath "/init-sysctl.yaml") . | sha256sum }}
        checksum/plugins: {{ include (print $.Template.BasePath "/install-plugins.yaml") . | sha256sum }}
        checksum/config: {{ include (print $.Template.BasePath "/config.yaml") . | sha256sum }}
        checksum/secret: {{ include (print $.Template.BasePath "/secret.yaml") . | sha256sum }}
{{- if .Values.annotations}}
      {{- range $key, $value := .Values.annotations }}
        {{ $key }}: {{ $value | quote }}
      {{- end }}
{{- end }}
```

Рисунок 3.17 – Частина конфігураційного файлу компонента deployment для sonarqube

Частина конфігурації вхідних параметрів, що використовуються в чарті sonarqube, наведена на рис 3.18.

```
deploymentType: "StatefulSet"

# There should not be more than 1 sonarqube instance connected to the same databa
replicaCount: 1

# This will use the default deployment strategy unless it is overriden
deploymentStrategy: {}
# Uncomment this to scheduler pods on priority
# priorityClassName: "high-priority"

## Use an alternate scheduler, e.g. "stork".
## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-sc
##
# schedulerName:

## Is this deployment for OpenShift? If so, we help with SCCs
OpenShift:
  enabled: false
  createSCC: true

edition: "community"

image:
  repository: sonarqube
  tag: 9.7.1-{{ .Values.edition }}
  pullPolicy: IfNotPresent
  # If using a private repository, the imagePullSecrets to use
  # pullSecrets:
  #   - name: my-repo-secret
```

Рисунок 3.18 – Частина конфігурації вхідних параметрів

Далі необхідно також підготувати helm chart для централізованого сховища nuxus, завантаживши його чарт та встановивши його параметри у value файлі.

Останнім компонентом для налаштування буде перенесення запуску та передналаштування утиліти Jenkins. У першу чергу необхідно підготувати список плагінів, які потрібні для всіх сценаріїв, а також самого Jenkins, на сторінці з описом плагіна знайти ім'я плагіна для установки і задати цей параметр в конфігурації запуску Jenkins. Основні плагіни, які використовуються в Jenkins:

- Configuration-as-code дозволяє описувати глобальну конфігурацію Jenkins як файл в yaml форматі;
- Docker дозволяє запускати динамічних агентів у докер контейнерах, з можливістю тонкого налаштування конфігурації контейнерів;
- Kubernetes дозволяє запускати динамічних агентів усередині кластера kubernetes;
- Configuration-as-code-jobs дозволяє описувати сценарії запуску в Jenkins у вигляді файлу в форматі yaml. Для роботи необхідний плагін configuration-as-code.

Після підготовки образу старту Jenkins можна налаштувати запуск. Всі конфігурації будуть здійснюватися за допомогою двох плагінів configuration-as-code, які конфігурують базові налаштування Jenkins – налаштування системи, налаштування плагінів та налаштування безпеки та плагін configuration-as-code-jobs – настроюючі сценарії при першому старті програми. Налаштування записуються у форматі yaml у Kubernetes конфігурації типу configmap, далі при запуску події ця конфігурація підключається як примонтований розділ, і вказується в якій директорії цей розділ буде розміщений. На рис. 3.19 представлена часткова конфігурація базового налаштування Jenkins, за допомогою плагіна configuration-as-code.

```yaml
apiVersion: v2
kind: ConfigMap
metadata:
  name: jenkins-startup
data:
  jenkins.conf: |
    jenkins:
      agentProtocols:
      - "JNLP4-connect"
      - "Ping
      crumbIssuer:
        standard:
          excludeClientIPFromCrumb: false
      disableRememberMe: false
      labelAtoms:
      - name: "built-in"
      markupFormatter: "plainText"
      mode: NORMAL
      myViewsTabBar: "standard"
      numExecutors: 2
      primaryView:
        all:
          name: "all"
      projectNamingStrategy: "standard"
      quietPeriod: 5
      remotingSecurity:
        enabled: true
      scmCheckoutRetryCount: 0
```

Рисунок 3.19 – Часткова конфігурація базового налаштування Jenkins

Після того, як базова конфігурація була перенесена у файл конфігурації, необхідно перенести всі сценарії. За допомогою плагіна configuration-as-code-jobs, в цьому ж файлі описуються сценарії, а так як Kubernetes використовуватиметься спільно з helm, то за допомогою конструкції «if .Values.java-maven.enabled» можна обмежити створення сценаріїв лише для тих фреймворків, які будуть використовуватися в Jenkins. На рис. 3.20 Наведено приклад конфігурації сценарію для фреймворку java-maven.

```
jobs:
  script: |
    {{ if .Values.java-maven.enabled }}
    pipeline {
      environment {
        JAVA_TOOL_OPTIONS = ">
                            -Duser.home=/var/maven
                            /tmp/maven:/var/maven/.m2
                            MAVEN_CONFIG=/var/maven/.m2
        "
      SONAR_USER_HOME = "/var/maven/.m2"
      DOCKER = credentials('Docker')
    }
    agent any

    options {
      gitLabConnection("git")
      gitlabBuilds(builds: ['Init', 'Checkout', 'Git version', 'Build ', 'Build-image', 'Push-to-nexus', 'Push-image-nexus', 'Push-ECR'])
    }

    triggers {
      gitlab(triggerOnPush: false, triggerOnMergeRequest: true, branchFilterType: 'NameBasedFilter', includeBranchesSpec: "{source.branch}")
```

Рисунок 3.20 – Конфігурація сценарію для фреймворку java-maven

Останнім кроком у підготовці чарту для Jenkins буде створення файлу із конфігурацією – values. Тут необхідно описати базові змінні, змінні оточення та залежності, а також налаштувати усі фреймворки. Частина конфігурації файлу values представлена на рис. 3.21.

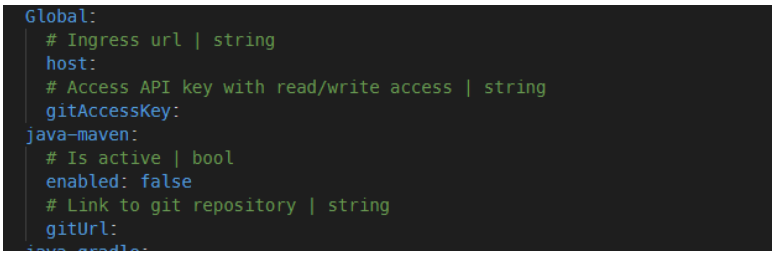

Рисунок 3.21 – Частина конфігурації файлу values

Після завершення створення та налаштування всіх чартів, необхідно підготувати файлову структуру. У кореневій папці потрібно створити папку templates і перемістити всі створені helm chart в неї, створити в корені файл values, в який перенести всі параметри з конфігураційного файлу в кожному чарті, а також створити файл Chart.yaml в якому вказати залежності - створені чарти. На поточний момент, після заповнення файлу values.yaml у корені проекту валідними даними, після виконання команди: helm install project ./project –values values.yaml, всередині кластера Kubernetes розгортається весь проект.

3.4 Висновки до третього розділу

У цьому розділі було розглянуто такі питання та вирішено такі проблеми:

1) Проведено аналіз ринку програмного забезпечення. Вибрано найпопулярніші та відповідні за функціональністю додатки для вирішення кожної із завдань. Проведено порівняння сильних і слабких сторін усіх додатків, після аналізу привілеїв та недоліків були обрані найвідповідніші додатки для вирішення завдань, поставлених до системи. Вибрано оптимальні інструменти для вирішення кожного завдання.

2) Зроблено почергове налаштування кожного компонента системи. Детально описаний кожен крок роботи субкомпонентів, описано як кожен компонент системи працюватиме залежно від різних вхідних даних (мов програмування та фреймворків).
3) Здійснено налаштування інтеграції між усіма компонентами системи. Описано взаємозв'язок компонентів і те, як вони працюватимуть між собою, передавати дані та отримувати відповіді.
4) Автоматизоване розгортання всіх сервісів із підготовлених шаблонів для різних ситуацій.
5) Зроблено підготовку для автоматичного розгортання всіх компонентів з готовими конфігураціями та інтеграціями. Створено хельм чарт, який на вході отримує файл з конфігураціями (посилання на гіт репозиторій, ключі для авторизації) і залежно від отриманих параметрів за допомогою команди запуску розгорнеться оточення з готовою для роботи системою.

Для подальшої роботи системи необхідно підготувати мануал, з інформацією про те, які системи існують, мінімальні системні вимоги до апаратного забезпечення, необхідний для її стабільної роботи, а також створити інструкцію для використання даної системи та інструкції з тестування системи та компонентів, які будуть запускатися на ній.

4 ТЕСТУВАННЯ Й ВПРОВАДЖЕННЯ СИСТЕМИ

4.1 Вимоги до платформи та оточення

Грунтуючись на обраній архітектурі для реалізації поточного проекту, всі додатки будуть запускатися в докер контейнерах в кластері Kubernetes, які попередньо налаштовані, в них встановлені всі необхідні компоненти, основні та додаткові утиліти та програми, налаштовані змінні оточення, що звільняє від необхідності додаткових налаштувань, так що в цьому розділі будуть описані системні вимоги для запуску використовуваних програм, а також наявність обов'язкових компонентів, які не були описані в попередніх розділах.

Для запуску програми Nexus необхідно

Мінімальні системні вимоги:

- CPU: 1 ядро;
- RAM: 1 Gb;
- Storage: 5 Gb.

Рекомендовані системні вимоги:

- CPU: 1 ядро;
- RAM: 2 Gb;
- Storage: 30 Gb.

Для запуску програми Sonar необхідно

Мінімальні системні вимоги:

- CPU: 1 ядро;
- RAM: 1 Gb;
- Storage: 10 Gb.

Рекомендовані системні вимоги:

- CPU: 2 ядро;
- RAM: 2 Gb;
- Storage: 15 Gb.

Для запуску програми Jenkins-master необхідно

Мінімальні системні вимоги:

- CPU: 1 ядро;
- RAM: 1 Gb;
- Storage: 10 Gb.

Рекомендовані системні вимоги:

- CPU: 2 ядро;
- RAM: 2 Gb;
- Storage: 30 Gb.

Для запуску програми Jenkins-agent необхідно

Мінімальні системні вимоги:

- CPU: 1 ядро;
- RAM: 1 Gb;
- Storage: 1 Gb.

Рекомендовані системні вимоги:

- CPU: 2 ядро;
- RAM: 4 Gb;
- Storage: 5 Gb.

Для роботи в кластері Kubernetes, його необхідно попередньо запустити, це можливо зробити локально використовуючи утиліту minicube або встановивши kublet на своєму сервері, створити кластер в хмарних ресурсах. Поточну реалізацію було зроблено всередині хмарного провайдера Amazon Web Services. Для роботи з кластером необхідно зайти в консоль управління обліковим записом, перейти до розділу elastic kubernetes cluster і додати новий кластер. Також кластер можна створити за допомогою утиліти aws cli.

Для роботи поточного проекту знадобляться щонайменше 2 ноди – для Kubernetes master та для Kubernetes agent. Велика кількість нод дозволить працювати більш ефективно, за рахунок того, що ресурси не навантажуватимуться так сильно і з'явиться можливість розділити навантаження на декількох машинах, але це також призведе до того, що

збільшаться витрати на утримання апаратури, тому кожен клієнт повинен сам вибирати оптимальні для нього співвідношення продуктивності та фінансових витрат.

Для коректної роботи всіх компонентів, з урахуванням мінімальних системних вимог для запуску кластера kubernetes підійдуть щонайменше такі типи інстансів:

- t2.small і краще;
- t3.small і краще;
- t4g.medium і краще;
- t4g.medium і краще;
- m5, m6a, m6i.

Так як більшість сервісів в AWS[19] платні, і багато хто з них знімає плату за використаний час, то хорошим рішенням буде налаштувати розклад роботи кластера для економії витрат. Для цього є кілька підходів, які можна вибрати за зручністю використання. Один з варіантів налаштування кластера розкладу є створити правила в меню Auto Scaling Group. Користувач вказує кількість машин, які будуть запускатися, час роботи, типи машин і безпосередньо саму конфігурацію машин. В іншому варіанті, користувач може написати Lamda функцію і додати її до Amazon Web Services облікового запису, яка контролюватиме роботу хмарного кластера. Найдешевший, але рутинний варіант - створювати та видаляти інстанси мануально, на вимогу щоразу коли це необхідно.

Ще одна хороша можливість знизити фінансові витрати - використовувати спотові інстанси замість інстансів on demand. Такі типи машин коштують набагато дешевше, що за великої кількості віртуальних машин та великих тимчасових навантажень значно зменшать витрати на використання інфраструктури. Однак варто врахувати, що такі типи машин підійдуть тільки для розробки та тестування, і здебільшого не підходять для використання у виробничому середовищі, тому що надаються вони тільки у випадку, якщо в регіоні є машини, що не використовуються, на вимогу, і в

деяких ситуаціях може не бути потрібної кількості машин, або не бути їх зовсім.

Також для роботи системи необхідно створити запис у сервісі Route53 – вбудованих dns у хмарному провайдері amazon web services. Працює як звичайний dns сервіс, підтримує найпопулярніші типи записів, такі як A record, CNAME, AAAA, NS і дозволяє перенаправляти трафік на внутрішні ресурси AWS[20], в даному випадку на балансувальник навантаження, створюваний усередині helm chart.

4.2 Інструкція до використання

Для роботи з системою необхідні такі встановлені та підготовлені сервіси:

- Kubernetes cluster з двома або більше нодами;
- Ingress контролер;
- Запис Route53 для ingress контролера;
- Встановлена утиліта kubectl;
- Встановлена утиліта helm.

Користувачеві системи необхідно завантажити helm chart собі на пристрій або додати його в свої helm репозиторії. Наступним кроком буде підготовка файлу зі змінними, для запуску самого чарту, приклад із необхідними змінними та їх описом знаходиться всередині чарту. Після цього необхідно запустити команду для розгортання чарту. Потім, як helm завершить свою роботу, в консоль користувача буде виведена інформація про стан запуску, а також технічна інформація для роботи з компонентами системи. Користувач зможе перевірити, що всі компоненти запустилися за допомогою консолі та утиліти kubectl або за допомогою будь-якої утиліти для роботи з Kubernetes кластером, наприклад Lens. Приклад виведення в консоль для запуску чарту наведено на рис. 4.1.

```
STATUS: deployed
REVISION: 1
NOTES:
Your domain - daria-ponomarenko.pp.ua

Jenkins     - https://jenkins.daria-ponomarenko.pp.ua
Nexus       - https://nexus.daria-ponomarenko.pp.ua
Sonarqube   - https://sonar.daria-ponomarenko.pp.ua
```

Рисунок 4.1 – Виведення в консоль для запуску чарту

Далі користувач може перевірити роботу основних компонентів. Для запуску сценарію перевірки вихідного коду – створити pull request у тому git репозиторії, який був вписаний у налаштуваннях helm чарту. Після створення pull request за допомогою webhook здійснюється запуск сценарію на перевірку коду. Відпрацьовують кроки завантаження репозиторію в робочу директорію, перевіряє коміт на відповідність стандартам, проводить лінтинг докерфайлу і helm чарту, перевірка файлів вихідного коду за допомогою аналазатора коду sonar. Результатом роботи цього сценарію має бути pipeline, представлений на рис. 4.2.

Рисунок 4.2 – Результат роботи сценарію

У вікні сценарію користувач може переглянути загальний стан сценарію, для кожного фреймворку, переглянути логи для одного з визначених кроків або відкрити загальний лог виконання сценарію для дебага, перейти з посилання на sonarqube і переглянути статус перевірки коду. В утиліті sonarqube користувач може переглянути результат статичного аналізу вихідного коду. Тут буде загальна статистика з виконаного сканування, з оцінками та показниками, а також є можливість перейти всередину поточного сканування для отримання більш детальної інформації, такої як кількість знайдених невідповідностей встановленим правилам, кількість багів у коді, кількість вразливостей та помилок, переглянути знайдені вразливості безпеки та інші показники. На рис. 4.3 представлено вікно статичного аналізатора коду sonarqube.

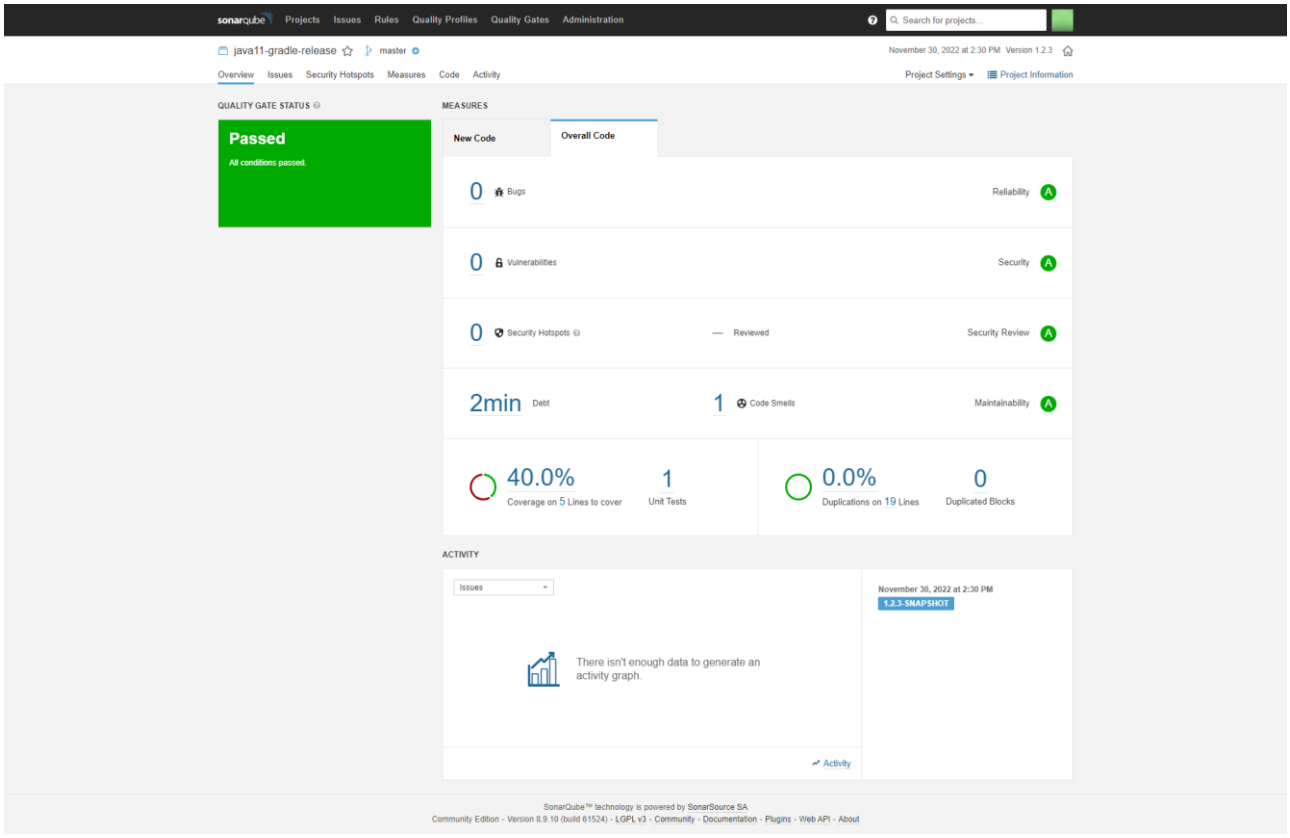

Рисунок 4.3 – Статус перевірки коду

Далі, якщо сценарій успішно відпрацьовано – до системи зберігання версії буде додано тег, що буде прив'язаний до коміту, який перевірявся у сценаріях, із коментарем версії.

Після цього, якщо користувач підтвердить запит на злиття в git репозиторіях, спрацює webhook за подією push, і він запустить наступний сценарій для складання артефакту програми. У цьому сценарії виконуються кроки підготовки робочої директорії проекту, скачування вихідного коду, який був злитий в кінцеву гілку в репозиторії, складання певного артефакту, залежно від використовуваного фреймворку, відправка артефакту в систему зберігання додатків Nexus, складання докеру образу, його тегування та відправлення образу у систему Nexus. Після поточного сценарію користувача системи буде 2 додатки - зібраний артефакт і зібраний докер образ, які він може використовувати в тестових або продакшен оточеннях. Також система зберігання артефактів Nexus дозволяє зберігати різні версії додатків, що дозволяє легко перемикатися між необхідними версіями і вести контроль кожної з них. Приклад сценарію складання для java додатків maven наведено на рис. 4.4.

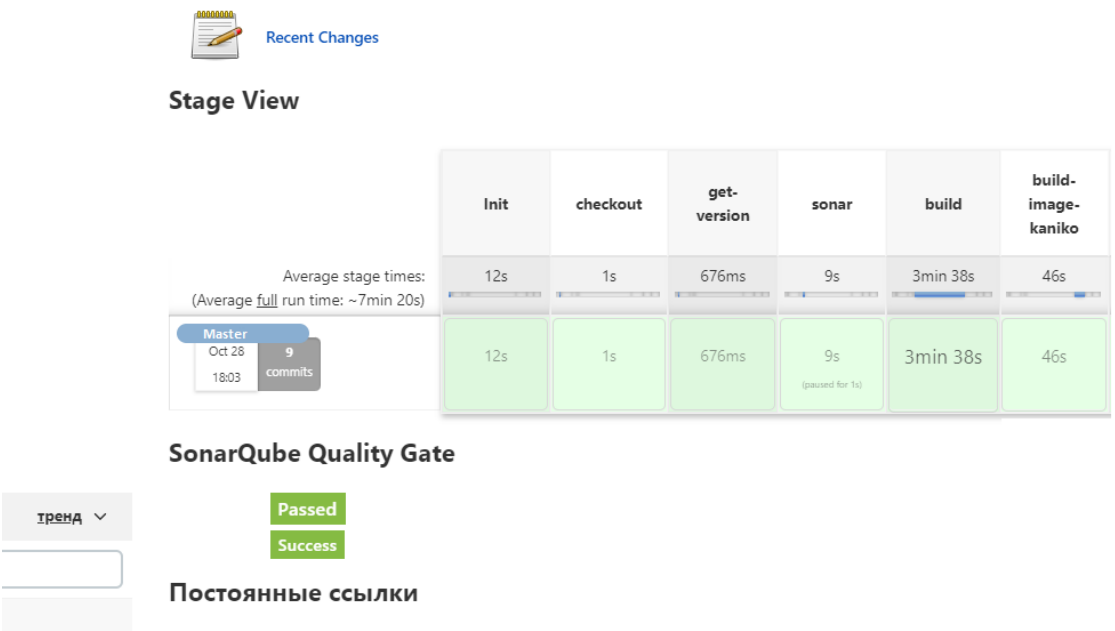

Рисунок 4.4 – Сценарій складання для java додатків maven

Всередині кожного сценарію також можна переглянути всі запуски, вивчити його детальну інформацію про кожен розгляд, логи, посилання на нексусне сховище. Після завершення сценарію складання користувач може перейти в Nexus і переконатися у наявності нового артефакту. На рис. 4.5 приклад подання артефактів з 1-м дослідженням артефакту всередині.

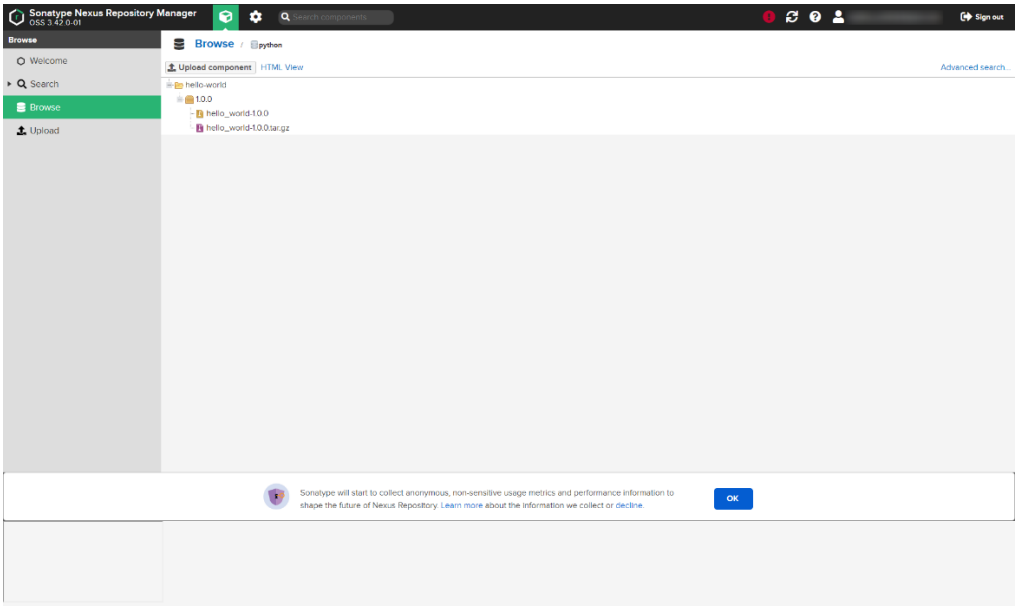

Рисунок 4.5 – Подання артефактів з 1-м дослідженням артефакту всередині

4.3 Тестування системи

У ході роботи з автоматизованою системою важливим кроком є тестування працездатності системи та її компонентів та моніторинг роботи всіх елементів системи. Для роботи з kubernetes кластерами існує безліч утиліт, розглянемо 2 з них:

– Kubectl – утиліта командного рядка для роботи з Kubernetes API. З її допомогою можна створювати, переглядати, змінювати та видаляти компоненти Kubernetes кластера. Має повну підтримку всіх функцій для роботи з кластером Kubernetes, єдиний недолік при роботі з великими навантаженнями - в командному рядку незручно відстежувати деякі елементи.

– Lens – програмне забезпечення для роботи з Kubernetes кластером. Має свій графічний інтерфейс, за допомогою якого можливо проводити моніторинг ресурсів усередині кластера у зручному форматі, дозволяє налаштовувати відображення та подання, зручний при використанні з кількома кластерами. Також за допомогою lens можна змінювати та видаляти всі типи ресурсів усередині кластера.

При вході в Lens у головному вікні необхідно вибрати з яким кластером працювати, після чого відкривається вікно з переліком ресурсів та їх вмістом. На рис. 4.6 показано бокове меню із переліком ресурсів kubernetes кластера.

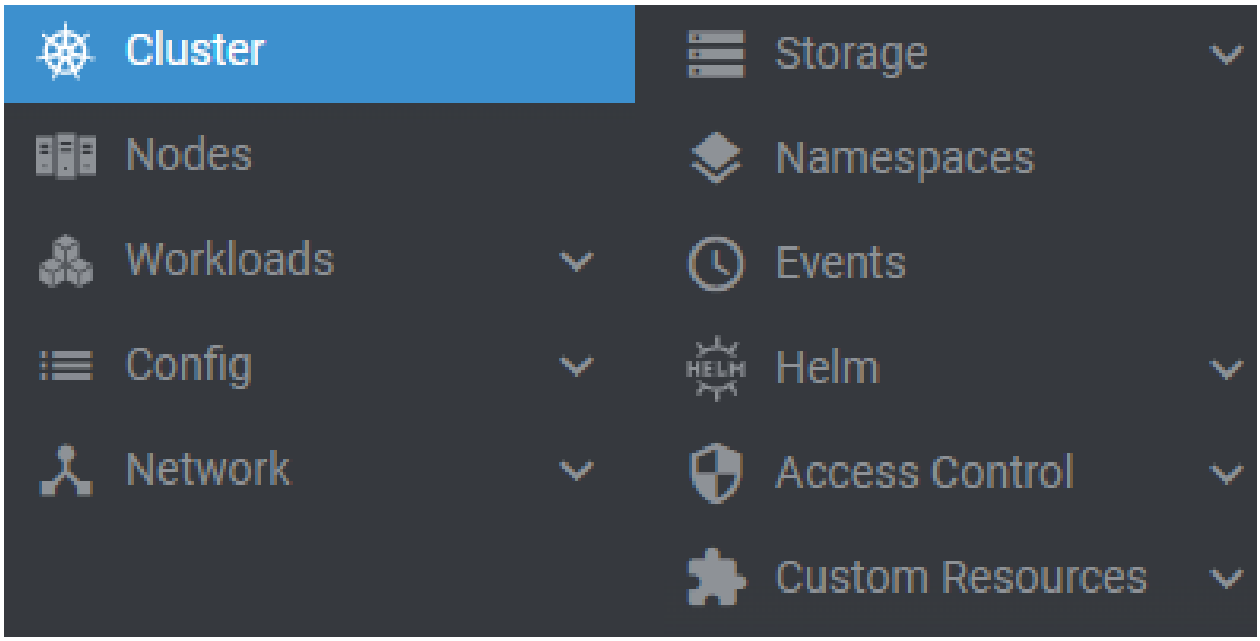

Рисунок 4.6 – Бокове меню kubernetes кластера

Вибравши необхідний тип ресурсу, на екрані буде відображено всі наявні компоненти цього типу. У Kubernetes є 2 типи ресурсів кластерні та рівня простору імен, якщо був вибраний ресурс 2-го типу, тобто можливість відфільтрувати висновок за певним простором імен. У таких ресурсах як pod можна подивитися логи в реальному часі. У всіх ресурсах є можливість переглянути конфігурацію у вбудованому редакторі та за необхідності внести зміни. Також у кожному вікні можна переглянути стан того чи іншого процесу. На рис. 4.7 показано вікно Lens з ресурсами типу поди, також у правій частині

екрана відкрито бічне меню з інформацією про один поде і внизу відкриті логи цього поди.

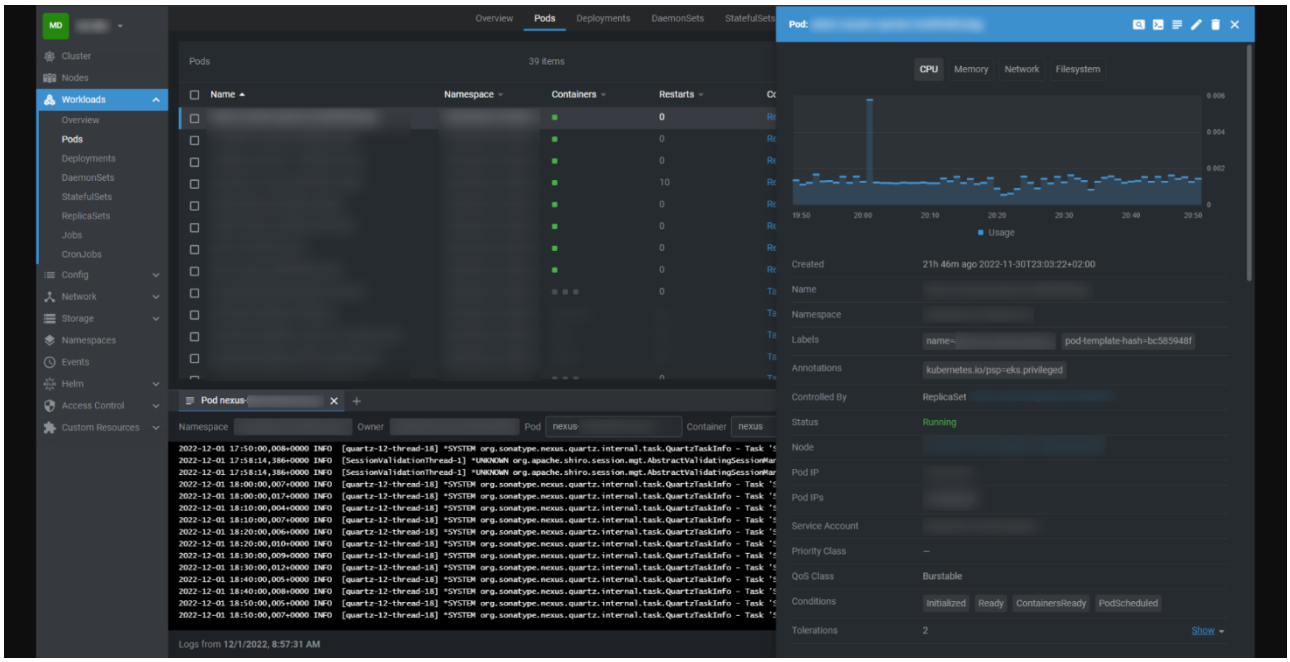

Рисунок 4.7 – Вікно програми Lens

Таким чином можна сказати, що lens є зручним та функціональним інструментом для моніторингу стану ресурсів усередині kubernetes кластера, а також корисний при тестуванні для пошуку несправностей роботи компонентів.

4.4 Висновки до четвертого розділу

У цьому розділі було вирішено такі питання:

1) У цьому розділі описано мінімальні системні вимоги апаратного забезпечення для коректного запуску системи. Були наведені різні конфігурації, щоб кожен міг підібрати оптимальний варіант для використання у своїй ситуації, оскільки різні програми використовують різні навантаження на систему. У деяких випадках, користувачі можуть використовувати апаратні засоби меншої продуктивності, ніж ті, що

наведені в технічному описі, проте немає гарантій на коректну роботу систему.

2) Незважаючи на те, що в посібнику описані вимоги та інструкції для запуску системи в хмарному провайдері Amazon Web Services, перевагою системи є те, що при необхідності користувачі можуть запустити систему на своєму апаратному забезпеченні. Для цього необхідно на своїх серверах встановити кубелет. Далі робота буде така ж як із кластером у хмарному провайдері. Це дозволить додати додаткові рівні ізоляції для проектів із підвищеними вимогами до безпеки.
3) Проведено аналіз витрат з економічної точки зору, запропоновано різні шляхи для мінімізації витрат користувачів під час використання публічних провайдерів хмарних сервісів.
4) Наведено детальну інструкцію з усіма попередніми кроками, для тих користувачів у яких ще немає у використанні Kubernetes кластера або інших ресурсів необхідних для роботи системи.
5) Описано кроки щодо створення кластера Kubernetes усередині хмарного провайдера Amazon Web Services.
6) Описано процес роботи з системою, на прикладі запуску та використання всіх стадій та кроків java maven програми. Наведено зображення того, як мають виглядати сценарії та компоненти робочої системи.
7) Наведено інструкцію з прикладами додатків та описом роботи з ними для тестування роботи системи та додатків, які вона збирає. Наведено приклади вікон пропонованих рішень, а також повний опис кожного елемента програми для повного розуміння процесу тестування.

ВИСНОВКИ

У даній магістерській роботі було проведено аналіз, спроектовано та створено систему для спрощення процесу розробки програмного забезпечення. Для реалізації системи було використано такі продукти – Jenkins, Sonarqube, Nexus, Kubernetes, Helm. За підсумком виконання роботи було досягнуто таких результатів:

1) Проведено огляд ринку розробки програмного забезпечення, за результатами якого було виділено основні труднощі та проблеми, з якими стикаються розробники програмного забезпечення.
2) Виділено основні методології та підходи для розробки ПЗ. Проведено огляд їх переваг та недоліків, на підставі якого визначено вимоги до процесів розробки програмного забезпечення.
3) Грунтуючись на отриманій інформації, було складено список вимог до системи, визначено основні компоненти, які повинні бути присутніми в ній.
4) За складеним списком вимог до системи, було розроблено архітектуру системи, виділено основні субкомпоненти, позначено їх роль та взаємозв'язок між собою.
5) Здійснено налаштування кожного окремого компонента, з детальним описом вироблених налаштувань, здійснено інтеграцію всіх елементів системи між собою.
6) Всі елементи системи були об'єднані в єдиний helm чарт для простого запуску за допомогою однієї команди. Основні елементи конфігурації системи винесені у файл зі змінними, для централізованого управління станом додатків.
7) Складено інструкцію з покроковими діями для запуску системи. Визначено мінімальні вимоги до апаратної частини для коректної роботи системи. Наведено різні конфігурації систем для роботи у публічному хмарному провайдері Amazon Web Services.

8) Оскільки основним споживачем системи будуть розробники-початківці, окрема увага приділена економії фінансів на витрати обслуговування обладнання. Наведено рекомендації для зниження витрат на продукти, що використовуються.

Перспективами розвитку додатку є:

1) Створення програми з інтерфейсом користувача, для централізованого управління всіма етапами з єдиного інтерфейсу.
2) Додавання SSO з можливістю розмежування ролей для зручної роботи в командах.
3) Внести можливість додавати користувацькі кроки та фреймворки до проекту.

ПЕРЕЛІК ПОСИЛАНЬ

1. Podeswa H. Agile Guide to Business Analysis and Planning, The: From Strategic Plan to Continuous Value Delivery: 1st Edition, 2021. 800 p.

2. Apke L. Understanding the Agile Manifesto, 2015. 74p.

3. Apke L. Understanding The Agile Manifesto: A Brief & Bold Guide to Agile, 2014. 46p.

4. Apke L. Agile Machine Learning: Effective Machine Learning Inspired by the Agile Manifesto: 1st ed. Edition, 2019. 265p.

5. Todaro D. The Epic Guide to Agile: More Business Value on a Predictable Schedule with Scrum, 2019. 518p.

6. Anderson J.D. Kanban: Successful Evolutionary Change for Your Technology Business, 2010. 278p.

7. Brechner E. Agile Project Management with Kanban (Developer Best Practices): 1st Edition, 2015. 160p.

8. Leopold K. Practical Kanban: From Team Focus to Creating Value, 2017. 353p.

9. A Scrum Book: The Spirit of the Game: 1st Edition / [Sutherland J. and etc.], 2019. 574p.

10. Ockerman S. Mastering Professional Scrum: A Practitioners Guide to Overcoming Challenges and Maximizing the Benefits of Agility (The Professional Scrum Series): 1st Edition / S. Ockerman, S. Reindl, 2019. 224p.

11. Press T. Scrum Basics: A Very Quick Guide to Agile Project Management, 2015. 108p.

12. Loeliger J. Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development: 3rd Edition / J. Loeliger, M. McCullough, 2022. 546p.

13. Poulton N. The Kubernetes Book: 2022 Edition Kindle Edition, 2022. 313p.

14. Hohn A. The Book of Kubernetes: A Complete Guide to Container Orchestration, 2022. 384p.

15. Papapetrou P. SonarQube in Action:1st Edition, Kindle Edition, 2013. 392p.

16. Jenkins Administrator's Guide: Install, manage, and scale a CI/CD build and release system to accelerate your product life cycle / [Park C.S. and etc.], 2021. 436p.

17. Learning Helm: Managing Apps on Kubernetes: 1st Edition / Butcher M., Farina M., Dolitsky J., 2021. 214p

18. Block A. Learn Helm: Improve productivity, reduce complexity, and speed up cloud-native adoption with Helm for Kubernetes: Kindle Edition / A. Block, A. Dewey, 2020. 487p.

19. Culkin J. AWS Cookbook: Recipes for Success on AWS: 1st Edition / J. Culkin, M. Zazon, 2021. 358p.

20. Potdar S.V. AWS Automation With Terraform: Infrastructure as Code: Kindle Edition, 2022. 198p.

ДОДАТОК А – Заява на затвердження теми і керівника роботи

Завідувачу кафедри
електронної техніки
Вовна Олександру Володимировичу
студентки 2 курсу, групи КІм-21,
напряму підготовки 123 «Комп'ютерна інженерія»
Пономаренко
Дар'ї Володимирівни

ЗАЯВА

Прошу призначити керівником моєї роботи магістра к.т.н., доц., доцента кафедри електронної техніки Шамаєв Віталій Віталійович
(вчена ступінь, вчене звання, посада, П.І.Б викладача)

і закріпити за мною наступну тему: «Засоби автоматизації розгортання монолітних та мікросервісних додатків»
(найменування теми)

«___»_____ 2022 р.
(підпис студента)

ПОГОДЖЕНО:

Науковий керівник _____ Шамаєв В.В.
(підпис) (прізвище та ініціали)

ДОДАТОК Б – Охорона праці

Аналіз умов праці відділу розробки програмного забезпечення для планування на промисловому підприємстві

Відділ розробки програмного забезпечення для планування на промисловому підриємстві розташовується на п’ятому поверсі п’ятиповерхового будинку. Розмір приміщення офісу наступний: довжина 5м; ширина 3м; висота 2,5м.

Виходячи з цих даних маємо:

- Площа приміщення

$$Sп = 5 * 4 = 20 м^2$$

- Об’єм приміщення

$$Vп = 5 * 4 * 2,5 = 50 м^3$$

Таким чином на одного програміста припадає площа в 20 $м^2$ і об’єм в 50 $м^3$ на людину.

Стіни мають шпалери бежевого кольору, коефіцієнт відбиття для стін 75%, для стелі – 50%. Висота столів над рівнем підлоги – 0,75м.

Для освітлення використовуються світильники типу ЛБ-80, які розміщуються на стелі. Вони розміщуються на відстані 0,15 м від стелі. Сумарна площа віконних прорізів - 2м $^{2.}$

Відділ займається розробкою, удосконаленням і технічною підтримкою програмного забезпечення для планування на промисловому підприємстві. У відділі працює один програміст.

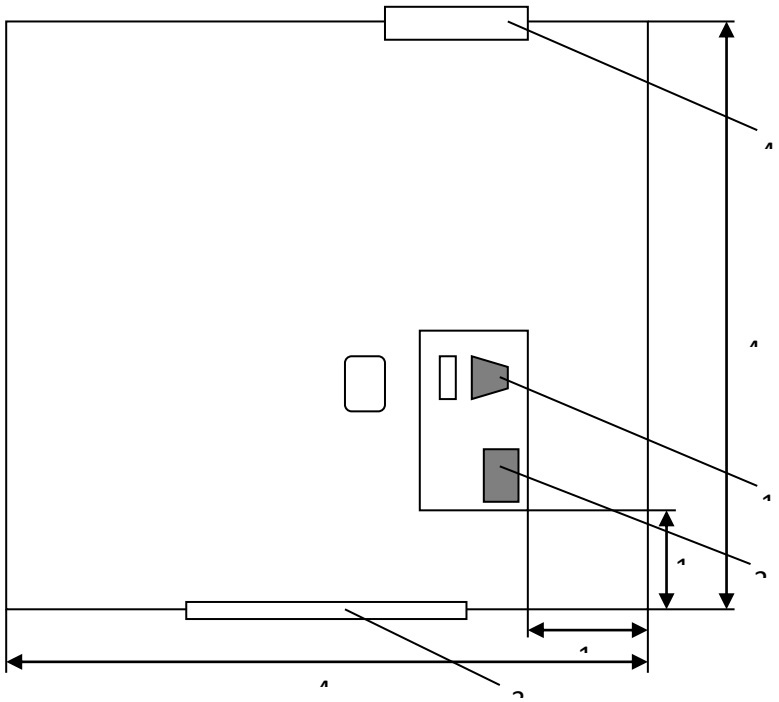

Рисунок Б.1 План промислового приміщення(1 - робоче місце, 2 - принтер, 3 - віконний проріз, 4 - дверний проріз)

Виконуються стандарти щодо розміщення меблі в приміщенні:

– мінімум 1м між комп’ютером та стіною з вікном;

– мінімум 1,2 м між боковими сторонами моніторів;

– мінімум 2,5 м між тильною стороною монітора та іншого комп’ютера або робочого місця;

– мінімум 1 м прохід.

Всі заземлені металеві предмети, такі як радіатори опалення, водопровід і газопровід, повинні бути закриті діелектричним щитом, щоб уникнути контакт з ними. У нас в приміщенні є радіатори опалення і вони закриті діелектричним щитом.

Робоче місце програміста знаходиться в офісі, а основна діяльність програміста пов'язана з роботою за комп'ютером. Таким чином, основні шкідливі фактори, що впливають на працівника офісу, пов'язані з комп'ютерами.

Стандарт 12.0.003-74 ССБТ «Небезпечні та шкідливі виробничі фактори» дає класифікацію небезпечних і шкідливих факторів на виробництві.

Можна виділити виробничі фактори, які виникають внаслідок роботи комп'ютера: підвищений рівень шуму; іонізація повітря; електричні і магнітні поля; підвищена яскравість світла; фізичні перевантаження (кисті рук, спина); психологічні перевантаження.

Всі ці шкідливі виробничі фактори викликають професійні захворювання у співробітника офісу, до яких відносяться наступні хвороби: ослаблення пам'яті; головні болі; серцево-судинні захворювання; ослаблення зору. Крім того, розумове напруження може спровокувати різні психологічні захворювання.

Розрахунок системи загального рівномірного освітлення з лампами розжарювання для приміщення, в якому використовуються зорові роботи високої точності

Офісне приміщення має розміри 5х4х2,5 м; коефіцієнт відбиття стін 70%, стелі - 50%; висота столів над рівнем підлоги - 0,75 м; відстань від світильника до стелі - 0,15 м.

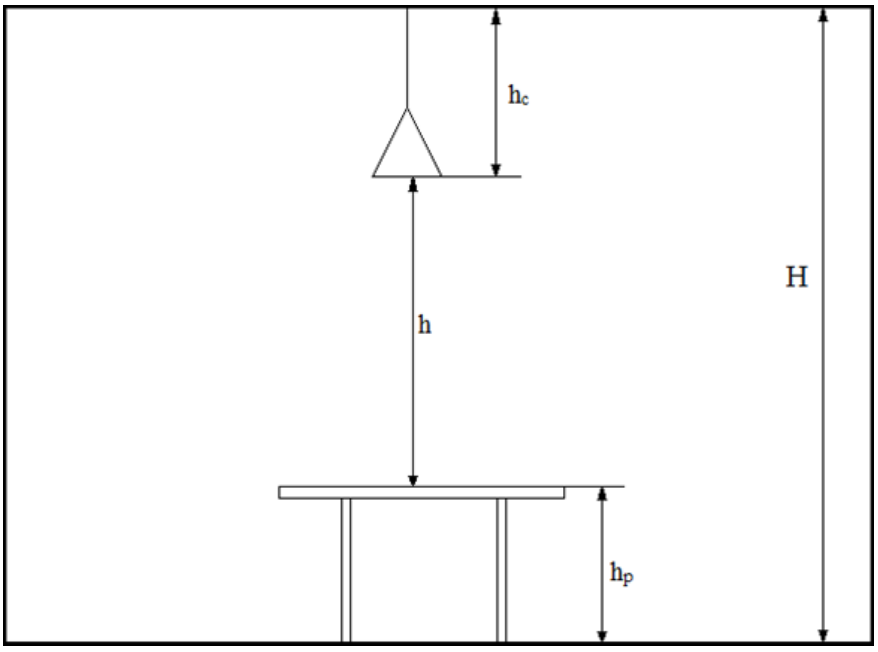

Рисунок Б.2 – Розрахунок індексу приміщення

Згідно СНіП II-4-79 «Будівельні норми і правила. Природне і штучне освітлення. Норми проектування », мінімальна освітленість складає Е = 400-500лк.

Насамперед визначимо висоту підвісу світильника над робочим місцем:

$$H_c = H - h_C - h_p = 2,5 - 0,15 - 0,75 = 1,6 \text{ м}$$

Далі визначимо рекомендовану відстань між світильниками за такою формулою:

$$L = 1,4\ H_c = 1,4 * 1,6 = 2,24 \text{ м}$$

Світловий потік світильника визначається такою формулою:

$$\Phi_л = \frac{E \cdot K_З \cdot S \cdot Z}{N \cdot n \cdot \eta} \text{ЛМ},$$

де Е - нормативна освітленість, лк;

$K_з$ - коефіцієнт запасу, що враховує зниження освітленості в результаті забруднення та старіння ламп $K_з = 1,3$

S - площа приміщення, що освітлюється, $м^2$;

Z - коефіцієнт нерівномірності освітлення для ламп розжарювання (=1,15);

N - кількість світильників;

n - кількість ламп у світильнику;

η - коефіцієнт використання світового потоку, який визначається за світлотехнічними таблицями залежно від показника приміщення (і) та коефіцієнтів відбиття стін та стелі.

Для світильників, що мають світловий потік $Ф_1 = 4320$ лм. (При і = 1,09, $Р_{стелі} = 70\%$, $Р_{стін} = 50\%$), коефіцієнт використання становить η = 0,5.

Обчислимо світловий потік одного світильника:

$Ф_л = (E * S * K_З * Z)/ \eta = (400 * 20 * 1,3 * 1,15) / 0,5 = 23920$ лм.

Кількість необхідних світильників можна порахувати, розділивши нормативний світловий потік на світловий потік, створюваний одним світильником:

$N = Ф_л / Ф$

N = 23920 / 4320 = 6 шт

Таким чином, для освітлення приміщення необхідно наявність 6 світильників типу ЛБ-80.

Комп'ютер, розташований в офісі, є джерелом тепла, таким чином, необхідно визначити умови його вентиляції.

Наступна формула дозволяє знайти повітрообмін по теплу:

$$L = \frac{Q_{над}}{c \times p(t_в - t_н)},$$

де $Q_{над}$ - надлишкове виділення тепла в робочому приміщенні, ккал/год.;

с - теплоємкість повітря (0,237 ккал/кг°С);

р - обсягова вага повітря (1,226 кг/ м^3);

$t_в$ - температура витяжного повітря (30°С);

$t_н$ - температура приточного повітря (20°С).

Надмірне надходження тепла можна обчислити за формулою:

$$Q_{над} = Q_{уст} + Q_{пер} + Q_{осв} + Q_{ср}$$

де $Q_{уст}$ - виділення тепла від устаткування;

$Q_{пер}$ - виділення тепла робітниками;

$Q_{осв}$ - надходження тепла від електричного освітлення;

$Q_{ср}$ - надходження тепла від сонячної радіації через вікна.

Виділення тепла від установок обчислюється за такою формулою:

$$Q_{уст} = Р \times К_а \times К_б \times 860,$$

де Р - сумарна потужність устаткування, кВт/год;

$К_а$ - коефіцієнт установленої потужності (0,95);

$К_б$ - коефіцієнт одночасної роботи (1,0).

В офісі маємо одне робоче місце, з 1 системним блоком і 1 монітором. Крім того, в офісі є один принтер. Потужність системного блоку 0,35 КВт, потужність монітора 0,04 КВт, потужність принтера 0,3. Таким чином, обчислюємо Р:

Р = 0,35 + 0,04 + 0,3 = 0,69 кВт/год

Таким чином, отримуємо:

$Q_{уст} = 0,69 \times 0,95 \times 1 \times 860 = 563,73$ ккал/год

Визначимо виділення тепла працівниками за такою формулою:

$Q_{пер} = n \times g$,

де n - кількість працюючих;

g - кількість тепла, що виділяє один працівник за годину (100 ккал/год.).

Таким чином, отримуємо:

$Q_{пер} = 1 \times 100 = 100$ ккал/год

Тепер, визначимо виділення тепла від електричного освітлення

$Q_{осв} = E_м \times g_1 \times S$ ккал/год

де $E_м$ - нормована освітленість для цієї зорової роботи приймаємо рівним 400 лк;

g_1 - питоме тепловиділення на 1 м^2 підлоги при 1 лк освітленості (для люмінесцентних ламп - 0,05 ккал/год.).

S - площа приміщення, м^2.

Підставивши числові значення, отримаємо:

$Q_{осв} = 400 \times 0,05 \cdot \times 20 = 400$ ккал/год

Далі визначимо тепло, створюване сонячною радіацією, яка потрапляє через вікна:

$Q_{ср} = F \cdot g_2 \cdot K_{осл}$,

де F - площа віконних прорізів (2 м^2);

g_2 - кількість тепла, що надходить через 1 м^2 віконного прорізу

(65 ккал/год.);

$K_{осл}$ - коефіцієнт ослаблення, приймаємо - 0,4.

Підставивши числові значення, маємо:

$Q_{ср} = 2 \cdot 65 \cdot 0,4 = 52$ ккал/год

Порахуємо сумарне значення надлишкового тепла:

$Q_{над} = 563,73 + 100 + 400 + 52 = 1115,73$ ккал/год

Визначимо витрати повітря в приміщенні:

L = 1115,73 / (0,237 * 1,226 * (30-20)) = 384,73 $м^3$/год

Система вентиляції, встановлена в приміщенні, має потужність 1000$м^3$/год, таким чином, вона задовольняє нормативам.

Параметри мікроклімату на робочих місцях регламентуються ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень». Відповідно до даних санітарних норм:

У холодні періоди року: температура повітря 21-23 градуса по Цельсію; швидкість руху повітря 0.1 метра в секунду; відносна вологість 40-60 %.

У теплі періоди року: температура повітря 22-24 градуса по Цельсію; швидкість руху повітря 0.2 метра в секунду; відносна вологість 40-60 %.

Вище зазначені норми цілком відповідають фактичним даним приміщення.

ДОДАТОК В – Лістинг програмного забезпечення

/sonarqube/chart/templates/change-admin-password-hook.yml

```
{{- if .Values.account }}
{{- if or .Values.account.adminPassword .Values.account.adminPasswordSecretName}}
apiVersion: batch/v1
kind: Job
metadata:
  name: {{ template "sonarqube.fullname" . }}-change-admin-password-hook
  labels:
    app: {{ template "sonarqube.name" . }}
    heritage: {{ .Release.Service }}
    release: {{ .Release.Name }}
    helm.sh/chart: "{{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}"
  {{- range $key, $value := .Values.service.labels }}
    {{ $key }}: {{ $value | quote }}
  {{- end }}
  annotations:
    "helm.sh/hook": post-install
    "helm.sh/hook-delete-policy": hook-succeeded
  {{- range $key, $value := .Values.adminJobAnnotations }}
    {{ $key }}: {{ $value | quote }}
  {{- end }}
spec:
  template:
    metadata:
      name: {{ template "sonarqube.fullname" . }}-change-admin-password-hook
      labels:
        app: {{ template "sonarqube.name" . }}
```

```
      heritage: {{ .Release.Service }}
      release: {{ .Release.Name }}
    {{- range $key, $value := .Values.service.labels }}
      {{ $key }}: {{ $value | quote }}
    {{- end }}
  spec:
    restartPolicy: OnFailure
    {{- if or .Values.image.pullSecrets .Values.image.pullSecret }}
    imagePullSecrets:
      {{- if .Values.image.pullSecret }}
      - name: {{ .Values.image.pullSecret }}
      {{- end }}
      {{- if .Values.image.pullSecrets }}
{{ toYaml .Values.image.pullSecrets | indent 8 }}
      {{- end }}
    {{- end }}
    serviceAccountName: {{ template "sonarqube.serviceAccountName" . }}
  {{- if .Values.tolerations }}
    tolerations:
{{ toYaml .Values.tolerations | indent 8 }}
  {{- end }}
  {{- if .Values.affinity }}
    affinity:
{{ toYaml .Values.affinity | indent 8 }}
  {{- end }}
    containers:
    - name: {{ template "sonarqube.fullname" . }}-change-default-admin-password
      image: {{ default "curlimages/curl:latest" .Values.curlContainerImage }}
      {{- if $securityContext := .Values.account.securityContext }}
      securityContext:
```

```
{{ toYaml $securityContext | indent 12 }}
      {{- end }}
      command: ["sh", "-c", 'until curl -v --connect-timeout 100 {{ template "sonarqube.fullname" . }}:{{ default 9000 .Values.service.internalPort }}{{ default "/" .Values.account.sonarWebContext }}api/system/status | grep -w UP; do sleep 10; done; curl -v --connect-timeout 100 -u admin:$CURRENT_ADMIN_PASSWORD -X POST "{{ template "sonarqube.fullname" . }}:{{ default 9000 .Values.service.internalPort }}{{ default "/" .Values.account.sonarWebContext }}api/users/change_password?login=admin&previousPassword=$CURRENT_ADMIN_PASSWORD&password=$ADMIN_PASSWORD"']
      env:
      - name: ADMIN_PASSWORD
        valueFrom:
          secretKeyRef:
            {{- if .Values.account.adminPassword }}
            name: {{ template "sonarqube.fullname" . }}-admin-password
            {{- else }}
            name: {{ .Values.account.adminPasswordSecretName }}
            {{- end }}
            key: password
      - name: CURRENT_ADMIN_PASSWORD
        valueFrom:
          secretKeyRef:
            {{- if .Values.account.adminPassword }}
            name: {{ template "sonarqube.fullname" . }}-admin-password
            {{- else }}
            name: {{ .Values.account.adminPasswordSecretName }}
            {{- end }}
            key: currentPassword
      resources:
```

```
{{ toYaml (default .Values.resources .Values.account.resources) | indent 10 }}
{{- end }}
{{- end }}
/sonarqube/chart/templates/config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ template "sonarqube.fullname" . }}-config
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
data:
  sonar.properties: |
  {{- range $key, $val := .Values.sonarProperties }}
    {{ $key }}={{ $val }}
  {{- end }}
  {{- if not .Values.elasticsearch.bootstrapChecks }}
    sonar.es.bootstrap.checks.disable=true
  {{- end }}
  {{- if .Values.sonarSecretKey }}
    sonar.secretKeyPath={{ .Values.sonarqubeFolder }}/secret/sonar-secret.txt
  {{- end }}
/sonarqube/chart/templates/deployment.yaml
{{- if eq .Values.deploymentType "Deployment"}}
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ template "sonarqube.fullname" . }}
```

```
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
    app.kubernetes.io/name: {{ template "sonarqube.name" . }}-{{ template "sonarqube.fullname" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
    app.kubernetes.io/part-of: sonarqube
    app.kubernetes.io/component: {{ template "sonarqube.fullname" . }}
    app.kubernetes.io/version: {{ tpl .Values.image.tag . | quote }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      app: {{ template "sonarqube.name" . }}
      release: {{ .Release.Name }}
{{- if .Values.deploymentStrategy }}
  strategy:
{{ toYaml .Values.deploymentStrategy | indent 4 }}
{{- end }}
  template:
    metadata:
      labels:
        app: {{ template "sonarqube.name" . }}
        release: {{ .Release.Name }}
{{- with .Values.podLabels }}
{{ toYaml . | indent 8 }}
{{- end }}
```

```
      annotations:
        checksum/init-sysctl: {{ include (print $.Template.BasePath "/init-sysctl.yaml")
. | sha256sum }}
        checksum/plugins: {{ include (print $.Template.BasePath "/install-
plugins.yaml") . | sha256sum }}
        checksum/config: {{ include (print $.Template.BasePath "/config.yaml") . |
sha256sum }}
        checksum/secret: {{ include (print $.Template.BasePath "/secret.yaml") . |
sha256sum }}
{{- if .Values.annotations}}
      {{- range $key, $value := .Values.annotations }}
        {{ $key }}: {{ $value | quote }}
      {{- end }}
{{- end }}
    spec:
    {{- if .Values.schedulerName }}
      schedulerName: {{ .Values.schedulerName }}
    {{- end }}
      serviceAccountName: {{ template "sonarqube.serviceAccountName" . }}
      securityContext:
{{ toYaml .Values.securityContext | indent 8 }}
      {{- if or .Values.image.pullSecrets .Values.image.pullSecret }}
      imagePullSecrets:
        {{- if .Values.image.pullSecret }}
        - name: {{ .Values.image.pullSecret }}
        {{- end }}
        {{- if .Values.image.pullSecrets}}
{{ toYaml .Values.image.pullSecrets | indent 8 }}
        {{- end }}
      {{- end }}
```

```
    initContainers:
    {{- if .Values.extraInitContainers }}
{{ toYaml .Values.extraInitContainers | indent 8 }}
    {{- end }}
    {{- if .Values.caCerts.enabled }}
     - name: ca-certs
       image: {{ default "adoptopenjdk/openjdk11:alpine" .Values.caCerts.image }}
       imagePullPolicy: {{ .Values.image.pullPolicy  }}
       command: ["sh"]
       args: ["-c", "cp -f \"${JAVA_HOME}/lib/security/cacerts\" /tmp/certs/cacerts;
if [ \"$(ls /tmp/secrets/ca-certs)\" ]; then for f in /tmp/secrets/ca-certs/*; do keytool -
importcert -file \"${f}\" -alias \"$(basename \"${f}\")\" -keystore /tmp/certs/cacerts -
storepass changeit -trustcacerts -noprompt; done; fi;"]
       {{- if $securityContext := .Values.initContainers.securityContext }}
       securityContext:
{{ toYaml $securityContext | indent 12 }}
       {{- end }}
       resources:
{{ toYaml .Values.initContainers.resources | indent 12 }}
       volumeMounts:
         - mountPath: /tmp/certs
           name: sonarqube
           subPath: certs
         - mountPath: /tmp/secrets/ca-certs
           name: ca-certs
       {{- with .Values.env }}
       env:
         {{- . | toYaml | trim | nindent 12 }}
       {{- end }}
    {{- end }}
```

```
    {{- if or .Values.initSysctl.enabled .Values.elasticsearch.configureNode }}
      - name: init-sysctl
        image: {{ default "busybox:1.32" .Values.initSysctl.image }}
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        {{- if $securityContext := (default .Values.initContainers.securityContext .Values.initSysctl.securityContext) }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml (default .Values.initContainers.resources .Values.initSysctl.resources) | indent 12 }}
        command: ["sh",
          "-e",
          "/tmp/scripts/init_sysctl.sh"]
        volumeMounts:
          - name: init-sysctl
            mountPath: /tmp/scripts/
        {{- with .Values.env }}
        env:
          {{- . | toYaml | trim | nindent 12 }}
        {{- end }}
    {{- end }}
    {{- if .Values.plugins.install }}
      - name: install-plugins
        image: {{ default "curlimages/curl:7.76.1" .Values.plugins.image }}
        imagePullPolicy: {{ .Values.image.pullPolicy }}
        command: ["sh",
          "-e",
          "/tmp/scripts/install_plugins.sh"]
```

```
        volumeMounts:
          - mountPath: {{ .Values.sonarqubeFolder }}/extensions/plugins
            name: sonarqube
            subPath: extensions/plugins
          - name: install-plugins
            mountPath: /tmp/scripts/
          {{- if .Values.plugins.netrcCreds }}
          - name: plugins-netrc-file
            mountPath: /root
          {{- end }}
        {{- if $securityContext := .Values.initContainers.securityContext }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml (default .Values.initContainers.resources .Values.plugins.resource) | indent 12 }}
        env:
          - name: http_proxy
            value: {{ default "" .Values.plugins.httpProxy }}
          - name: https_proxy
            value: {{ default "" .Values.plugins.httpsProxy }}
          - name: no_proxy
            value: {{ default "" .Values.plugins.noProxy }}
        {{- with .Values.env }}
          {{- . | toYaml | trim | nindent 12 }}
        {{- end }}
    {{- end }}
    {{- if or .Values.sonarProperties .Values.sonarSecretProperties .Values.sonarSecretKey (not .Values.elasticsearch.bootstrapChecks) }}
```

```
      - name: concat-properties
        image: {{ default "busybox:1.32" .Values.initContainers.image }}
        imagePullPolicy: {{ .Values.image.pullPolicy  }}
        command:
        - sh
        - -c
        - |
          #!/bin/sh
          if [ -f /tmp/props/sonar.properties ]; then
            cat /tmp/props/sonar.properties > /tmp/result/sonar.properties
          fi
          if [ -f /tmp/props/secret.properties ]; then
            cat /tmp/props/secret.properties > /tmp/result/sonar.properties
          fi
          if [ -f /tmp/props/sonar.properties -a -f /tmp/props/secret.properties ]; then
            awk 1 /tmp/props/sonar.properties /tmp/props/secret.properties > /tmp/result/sonar.properties
          fi
        volumeMounts:
        {{- if or .Values.sonarProperties .Values.sonarSecretKey (not .Values.elasticsearch.bootstrapChecks) }}
          - mountPath: /tmp/props/sonar.properties
            name: config
            subPath: sonar.properties
        {{- end }}
        {{- if .Values.sonarSecretProperties }}
          - mountPath: /tmp/props/secret.properties
            name: secret-config
            subPath: secret.properties
        {{- end }}
```

```
          - mountPath: /tmp/result
            name: concat-dir
        {{- if $securityContext := .Values.initContainers.securityContext }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml .Values.initContainers.resources | indent 12 }}
        {{- with .Values.env }}
        env:
          {{- . | toYaml | trim | nindent 12 }}
        {{- end }}
    {{- end }}
    {{- if .Values.postgresql.enabled }}
      - name: "wait-for-db"
        image: {{ default "busybox:1.32" .Values.initContainers.image }}
        imagePullPolicy: {{ .Values.image.pullPolicy  }}
        {{- if $securityContext := .Values.initContainers.securityContext }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml .Values.initContainers.resources | indent 12 }}
        command: ["/bin/sh", "-c", "for i in $(seq 1 200); do nc -z -w3 {{ .Release.Name}}-postgresql 5432 && exit 0 || sleep 2; done; exit 1"]
    {{- end }}
    {{- if .Values.priorityClassName }}
    priorityClassName: {{ .Values.priorityClassName }}
    {{- end }}
  {{- if .Values.nodeSelector }}
```

```
      nodeSelector:
{{ toYaml .Values.nodeSelector | indent 8 }}
    {{- end }}
    {{- if .Values.hostAliases }}
      hostAliases:
{{ toYaml .Values.hostAliases | indent 8 }}
    {{- end }}
    {{- if .Values.tolerations }}
      tolerations:
{{ toYaml .Values.tolerations | indent 8 }}
    {{- end }}
    {{- if .Values.affinity }}
      affinity:
{{ toYaml .Values.affinity | indent 8 }}
    {{- end }}
      containers:
      {{- if .Values.extraContainers }}
        {{- toYaml .Values.extraContainers | nindent 8 }}
      {{- end }}
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:{{ tpl .Values.image.tag . }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: {{ .Values.service.internalPort }}
              protocol: TCP
          env:
            - name: SONAR_WEB_JAVAOPTS
              value: {{ template "sonarqube.jvmOpts" . }}
            - name: SONAR_CE_JAVAOPTS
```

```
            value: {{ template "sonarqube.jvmCEOpts" . }}
          - name: SONAR_JDBC_PASSWORD
            valueFrom:
              secretKeyRef:
                name: {{ template "jdbc.secret" . }}
                key: {{ template "jdbc.secretPasswordKey" . }}
          - name: SONAR_WEB_SYSTEMPASSCODE
            valueFrom:
              secretKeyRef:
              {{- if and .Values.monitoringPasscodeSecretName
.Values.monitoringPasscodeSecretKey }}
                name: {{ .Values.monitoringPasscodeSecretName }}
                key: {{ .Values.monitoringPasscodeSecretKey }}
              {{- else }}
                name: {{ template "sonarqube.fullname" . }}-monitoring-passcode
                key: SONAR_WEB_SYSTEMPASSCODE
              {{- end }}
          {{- with .Values.env }}
          {{- . | toYaml | trim | nindent 12 }}
          {{- end }}
        envFrom:
          - configMapRef:
              name: {{ template "sonarqube.fullname" . }}-jdbc-config
{{- range .Values.extraConfig.secrets }}
          - secretRef:
              name: {{ . }}
{{- end }}
{{- range .Values.extraConfig.configmaps }}
          - configMapRef:
              name: {{ . }}
```

```
{{- end }}
        livenessProbe:
          exec:
            command:
            - sh
            - -c
            - |
              host="$(hostname -i || echo '127.0.0.1')"
              reply=$(wget -qO- --header="X-Sonar-Passcode: $SONAR_WEB_SYSTEMPASSCODE" http://${host}:{{ .Values.service.internalPort }}{{ .Values.livenessProbe.sonarWebContext }}api/system/liveness 2>&1)
              if [ -z "$reply" ]; then exit 0; else exit 1; fi
          initialDelaySeconds: {{ .Values.livenessProbe.initialDelaySeconds }}
          periodSeconds: {{ .Values.livenessProbe.periodSeconds }}
          failureThreshold: {{ .Values.livenessProbe.failureThreshold }}
        readinessProbe:
          httpGet:
            path: {{ .Values.readinessProbe.sonarWebContext }}api/system/status
            port: http
          initialDelaySeconds: {{ .Values.readinessProbe.initialDelaySeconds }}
          periodSeconds: {{ .Values.readinessProbe.periodSeconds }}
          failureThreshold: {{ .Values.readinessProbe.failureThreshold }}
        {{- if .Values.containerSecurityContext }}
        securityContext:
{{- toYaml .Values.containerSecurityContext | nindent 12 }}
        {{- end }}
        volumeMounts:
{{- if .Values.persistence.mounts }}
{{ toYaml .Values.persistence.mounts | indent 12 }}
```

```
{{- end }}
        {{- if or .Values.sonarProperties .Values.sonarSecretProperties
.Values.sonarSecretKey (not .Values.elasticsearch.bootstrapChecks) }}
        - mountPath: {{ .Values.sonarqubeFolder }}/conf/
          name: concat-dir
        {{- end }}
        {{- if .Values.sonarSecretKey }}
        - mountPath: {{ .Values.sonarqubeFolder }}/secret/
          name: secret
        {{- end }}
        {{- if .Values.caCerts }}
        - mountPath: {{ .Values.sonarqubeFolder }}/certs
          name: sonarqube
          subPath: certs
        {{- end }}
        - mountPath: {{ .Values.sonarqubeFolder }}/data
          name: sonarqube
          subPath: data
        {{- if .Values.persistence.enabled }}
        - mountPath: {{ .Values.sonarqubeFolder }}/extensions
          name: sonarqube
          subPath: extensions
        {{- else if .Values.plugins.install }}
        - mountPath: {{ .Values.sonarqubeFolder }}/extensions/plugins
          name: sonarqube
          subPath: extensions/plugins
        {{- end }}
        - mountPath: {{ .Values.sonarqubeFolder }}/temp
          name: sonarqube
          subPath: temp
```

```
          - mountPath: {{ .Values.sonarqubeFolder }}/logs
            name: sonarqube
            subPath: logs
          - mountPath: /tmp
            name: tmp-dir
        resources:
{{ toYaml .Values.resources | indent 12 }}
    volumes:
{{- if .Values.persistence.volumes }}
{{ tpl (toYaml .Values.persistence.volumes | indent 6) . }}
{{- end }}
    {{- if or .Values.sonarProperties .Values.sonarSecretKey ( not
.Values.elasticsearch.bootstrapChecks) }}
    - name: config
      configMap:
        name: {{ template "sonarqube.fullname" . }}-config
        items:
        - key: sonar.properties
          path: sonar.properties
    {{- end }}
    {{- if .Values.sonarSecretProperties }}
    - name: secret-config
      secret:
        secretName: {{ .Values.sonarSecretProperties }}
        items:
        - key: secret.properties
          path: secret.properties
    {{- end }}
    {{- if .Values.sonarSecretKey }}
    - name: secret
```

```
  secret:
    secretName: {{ .Values.sonarSecretKey }}
    items:
    - key: sonar-secret.txt
      path: sonar-secret.txt
{{- end }}
{{- if .Values.caCerts }}
- name: ca-certs
  secret:
    secretName: {{ .Values.caCerts.secret }}
{{- end }}
{{- if .Values.plugins.netrcCreds }}
- name: plugins-netrc-file
  secret:
    secretName: {{ .Values.plugins.netrcCreds }}
    items:
    - key: netrc
      path: .netrc
{{- end }}
- name: init-sysctl
  configMap:
    name: {{ template "sonarqube.fullname" . }}-init-sysctl
    items:
      - key: init_sysctl.sh
        path: init_sysctl.sh
- name: install-plugins
  configMap:
    name: {{ template "sonarqube.fullname" . }}-install-plugins
    items:
      - key: install_plugins.sh
```

```
            path: install_plugins.sh
      - name: sonarqube
        {{- if .Values.persistence.enabled }}
        persistentVolumeClaim:
          claimName: {{ if .Values.persistence.existingClaim }}{{ .Values.persistence.existingClaim }}{{- else }}{{ template "sonarqube.fullname" . }}{{- end }}
        {{- else }}
        emptyDir: {{- toYaml .Values.emptyDir | nindent 10 }}
        {{- end  }}
      - name : tmp-dir
        emptyDir: {{- toYaml .Values.emptyDir | nindent 10 }}
        {{- if or .Values.sonarProperties .Values.sonarSecretProperties .Values.sonarSecretKey ( not .Values.elasticsearch.bootstrapChecks) }}
      - name : concat-dir
        emptyDir: {{- toYaml .Values.emptyDir | nindent 10 -}}
        {{- end }}
{{- end }}
/sonarqube/chart/templates/ingress.yaml
{{- if .Values.ingress.enabled -}}
{{- $serviceName := include "sonarqube.fullname" . -}}
{{- $servicePort := .Values.service.externalPort -}}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ template "sonarqube.fullname" . }}
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
```

```
    heritage: {{ .Release.Service }}
{{- if .Values.ingress.labels }}
{{ .Values.ingress.labels | toYaml | trimSuffix "\n"| indent 4 -}}
{{- end}}
{{- if .Values.ingress.annotations}}
  annotations:
    {{- range $key, $value := .Values.ingress.annotations }}
    {{ $key }}: {{ $value | quote }}
    {{- end }}
{{- end }}
spec:
  {{- if .Values.ingress.ingressClassName }}
  ingressClassName: {{ .Values.ingress.ingressClassName }}
  {{- end }}
  rules:
  {{- range .Values.ingress.hosts }}
  - host: {{ printf "%s" .name }}
    http:
      paths:
      - backend:
          service:
            name: {{ default $serviceName .serviceName }}
            port:
              number: {{ default $servicePort .servicePort }}
        path: {{ .path}}
        pathType: {{ default "ImplementationSpecific" .pathType }}
  {{- end }}
  {{- if .Values.ingress.tls }}
  tls:
{{ toYaml .Values.ingress.tls | indent 4 }}
```

```
  {{- end -}}
{{- end }}
/sonarqube/chart/templates/init-fs.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ template "sonarqube.fullname" . }}-init-fs
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
data:
  init_fs.sh: |-
    {{- if .Values.persistence.enabled }}
    chown -R {{ .Values.persistence.uid }}: {{ .Values.sonarqubeFolder }}
    {{- end }}
/sonarqube/chart/templates/init-sysctl.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ template "sonarqube.fullname" . }}-init-sysctl
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
data:
  init_sysctl.sh: |-
    {{- if .Values.initSysctl.vmMaxMapCount }}
```

```
  if [[ "$(sysctl -n vm.max_map_count)" -lt {{ .Values.initSysctl.vmMaxMapCount
}} ]]; then
    sysctl -w vm.max_map_count={{ .Values.initSysctl.vmMaxMapCount }}
  fi
  {{- end }}
  {{- if .Values.initSysctl.fsFileMax }}
  if [[ "$(sysctl -n fs.file-max)" -lt {{ .Values.initSysctl.fsFileMax }} ]]; then
    sysctl -w fs.file-max={{ .Values.initSysctl.fsFileMax }}
  fi
  {{- end }}
  {{- if .Values.initSysctl.nofile }}
  if [[ "$(ulimit -n)" != "unlimited" ]]; then
    if [[ "$(ulimit -n)" -lt {{ .Values.initSysctl.nofile }} ]]; then
      echo "ulimit -n {{ .Values.initSysctl.nofile }}"
      ulimit -n {{ .Values.initSysctl.nofile }}
    fi
  fi
  {{- end }}
  {{- if .Values.initSysctl.nproc }}
  if [[ "$(ulimit -u)" != "unlimited" ]]; then
    if [[ "$(ulimit -u)" -lt {{ .Values.initSysctl.nproc }} ]]; then
      echo "ulimit -u {{ .Values.initSysctl.nproc }}"
      ulimit -u {{ .Values.initSysctl.nproc }}
    fi
  fi
  {{- end }}
/sonarqube/chart/templates/install-plugins.yaml
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: {{ template "sonarqube.fullname" . }}-install-plugins
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
data:
  install_plugins.sh: |-
    {{- if .Values.plugins.install }}
      [ -e {{ .Values.sonarqubeFolder }}/extensions/plugins/* ] && rm {{ .Values.sonarqubeFolder }}/extensions/plugins/*
      cd {{ .Values.sonarqubeFolder }}/extensions/plugins
      {{- range $index, $val := .Values.plugins.install }}
      curl {{ if $.Values.plugins.noCheckCertificate }}--insecure{{ end }} {{ if $.Values.plugins.netrcCreds }}--netrc-file /root/.netrc{{ end }} -fsSLO {{ $val | quote }}
      {{- end }}
    {{- end }}
/sonarqube/chart/templates/jdbc-config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ template "sonarqube.fullname" . }}-jdbc-config
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
data:
  SONAR_JDBC_USERNAME: {{ template "jdbc.username" . }}
```

```
{{- if .Values.jdbcOverwrite.enable }}
  SONAR_JDBC_URL: {{ .Values.jdbcOverwrite.jdbcUrl | trim | quote }}
{{- else if and .Values.postgresql.service.port .Values.postgresql.postgresqlDatabase
}}
  SONAR_JDBC_URL: "jdbc:postgresql://{{ template "postgresql.hostname" .
}}:{{- .Values.postgresql.service.port -}}/{{- .Values.postgresql.postgresqlDatabase
-}}"
{{- end }}
```

/sonarqube/chart/templates/networkpolicy.yaml

```
{{- if .Values.networkPolicy.enabled }}
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: {{ template "sonarqube.fullname" . }}-network-policy
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  podSelector:
    matchLabels:
      app: {{ template "sonarqube.name" . }}
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
```

```
      - podSelector:
          matchLabels:
            app: {{ template "sonarqube.name" . }}
            release: {{ .Release.Name }}
    ports:
      - port: {{ .Values.service.internalPort }}
{{ if .Values.prometheusExporter.enabled }}
  - from:
      - namespaceSelector:
          matchLabels:
            networking/namespace: {{ .Values.networkPolicy.prometheusNamespace }}
    ports:
      - port: {{ .Values.prometheusExporter.ceBeanPort }}
        protocol: TCP
      - port: {{ .Values.prometheusExporter.webBeanPort }}
        protocol: TCP
{{ end }}
 egress:
  - to:
      - namespaceSelector:
          matchLabels:
            networking/namespace: kube-system
        podSelector:
          matchLabels:
            k8s-app: kube-dns
    ports:
      - port: 53
        protocol: UDP
```

```
{{- if .Values.postgresql.enabled }}
    - to:
        - podSelector:
            matchLabels:
              app.kubernetes.io/name: postgresql
      ports:
        - port: 5432
          protocol: TCP
{{- end }}
    - to:
        - ipBlock:
            cidr: 0.0.0.0/0
{{- end -}}

{{ if and .Values.postgresql.enabled .Values.networkPolicy.enabled }}
---
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: {{ template "sonarqube.fullname" . }}-database
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  podSelector:
    matchLabels:
```

```
      app.kubernetes.io/name: postgresql
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: {{ template "sonarqube.name" . }}
      ports:
        - port: 5432
  egress:
    - to:
        - namespaceSelector: {}
          podSelector:
            matchLabels:
              k8s-app: kube-dns
      ports:
        - port: 53
          protocol: UDP
{{- end }}


{{- if and .Values.networkPolicy.enabled
.Values.networkPolicy.additionalNetworkPolicys }}
---
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: {{ template "sonarqube.fullname" . }}-additional-network-policy
```

```
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
{{- with .Values.networkPolicy.additionalNetworkPolicys -}}
{{ toYaml . | nindent 2 }}
{{- end }}
{{- end -}}
/sonarqube/chart/templates/prometheus-ce-config.yaml
{{- if .Values.prometheusExporter.enabled }}
apiVersion: v1
kind: ConfigMap
metadata:
  name: {{ template "sonarqube.fullname" . }}-prometheus-ce-config
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
data:
  prometheus-ce-config.yaml: |-
{{ .Values.prometheusExporter.ceConfig | default
.Values.prometheusExporter.config | toYaml | indent 8 }}
{{- end }}
/sonarqube/chart/templates/prometheus-config.yaml
{{- if .Values.prometheusExporter.enabled }}
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: {{ template "sonarqube.fullname" . }}-prometheus-config
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
data:
  prometheus-config.yaml: |-
{{ toYaml .Values.prometheusExporter.config | indent 8 }}
{{- end }}
```

/sonarqube/chart/templates/pvc.yaml

```
{{- if and .Values.persistence.enabled (not .Values.persistence.existingClaim) }}
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: {{ template "sonarqube.fullname" . }}
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: "{{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}"
    release: "{{ .Release.Name }}"
    heritage: "{{ .Release.Service }}"
{{ if .Values.persistence.annotations}}
  annotations:
    {{- range $key, $value := .Values.persistence.annotations }}
    {{ $key }}: {{ $value | quote }}
    {{- end }}
```

```
{{- end }}
spec:
  accessModes:
    - {{ .Values.persistence.accessMode | quote }}
  resources:
    requests:
      storage: {{ .Values.persistence.size | quote }}
{{- if .Values.persistence.storageClass }}
{{- if (eq "-" .Values.persistence.storageClass) }}
  storageClassName: ""
{{- else }}
  storageClassName: "{{ .Values.persistence.storageClass }}"
{{- end }}
{{- end }}
{{- end }}
/sonarqube/chart/templates/route.yaml
---
{{- if not (or .Values.postgresql.enabled .Values.postgresql.existingSecret
.Values.jdbcOverwrite.jdbcSecretName)}}
apiVersion: v1
kind: Secret
metadata:
  name: {{ template "sonarqube.fullname" . }}
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
type: Opaque
```

```
data:
  {{ template "jdbc.secretPasswordKey" . }}: {{ template
"jdbc.internalSecretPasswd" . }}
{{- end }}
---
{{- if .Values.monitoringPasscode}}
apiVersion: v1
kind: Secret
metadata:
  name: {{ template "sonarqube.fullname" . }}-monitoring-passcode
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
type: Opaque
data:
  SONAR_WEB_SYSTEMPASSCODE: {{ .Values.monitoringPasscode | b64enc |
quote }}
{{- end }}
---
{{- if .Values.account }}
{{- if .Values.account.adminPassword }}
apiVersion: v1
kind: Secret
metadata:
  name: {{ template "sonarqube.fullname" . }}-admin-password
  labels:
    app: {{ template "sonarqube.name" . }}
```

```
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
type: Opaque
stringData:
  password: {{ .Values.account.adminPassword | urlquery | quote }}
  currentPassword: {{ default "admin" .Values.account.currentAdminPassword | urlquery | quote }}
{{- end }}
{{- end }}
```

/sonarqube/chart/templates/service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ template "sonarqube.fullname" . }}
  labels:
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
  {{- range $key, $value := .Values.service.labels }}
    {{ $key }}: {{ $value | quote }}
  {{- end }}
{{ if .Values.service.annotations}}
  annotations:
    {{- range $key, $value := .Values.service.annotations }}
    {{ $key }}: {{ $value | quote }}
    {{- end }}
{{- end }}
```

```
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.externalPort }}
      targetPort: http
      protocol: TCP
      name: http
      {{- if .Values.service.nodePort }}
      nodePort: {{ .Values.service.nodePort }}
      {{- end }}
  selector:
    app: {{ template "sonarqube.name" . }}
    release: {{ .Release.Name }}
  {{- if eq .Values.service.type "LoadBalancer"}}
  {{- if .Values.service.loadBalancerSourceRanges }}
  loadBalancerSourceRanges:
  {{- range .Values.service.loadBalancerSourceRanges }}
  - {{ . }}
  {{- end }}
  {{- end -}}
  {{- if .Values.service.loadBalancerIP}}
  loadBalancerIP: {{.Values.service.loadBalancerIP}}
  {{- end }}
  {{- end }}
/sonarqube/chart/templates/serviceaccount.yaml
{{- if .Values.serviceAccount.create -}}
---
apiVersion: v1
```

```
kind: ServiceAccount
metadata:
{{- if .Values.serviceAccount.name }}
  name: {{ .Values.serviceAccount.name }}
{{- else  }}
  name: {{ include "sonarqube.fullname" . }}
{{- end }}
{{- if .Values.serviceAccount.annotations }}
  annotations:
{{ toYaml .Values.serviceAccount.annotations | indent 4 }}
{{- end }}
automountServiceAccountToken: {{ .Values.serviceAccount.automountToken |
default "false" }}
{{- end -}}
/sonarqube/chart/templates/{{- if and (.Values.OpenShift.enabled)
(.Values.OpenShift.createSCC) }}

# This SCC allows any user ID but restricts capabilties and host access
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: "allows pod to run as root, privileged and run sysctl"
    "helm.sh/hook": pre-install
  name: {{ .Release.Name }}-privileged-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
```

```
allowHostPorts: false
allowPrivilegedContainer: true
allowPrivilegeEscalation: true
allowedCapabilities: []
allowedFlexVolumes: []
allowedUnsafeSysctls: []
defaultAddCapabilities: []
defaultAllowPrivilegeEscalation: true
fsGroup:
  type: RunAsAny
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SETGID
runAsUser:
  type: RunAsAny
# This can be customized for your host machine
seLinuxContext:
  type: MustRunAs
# seLinuxOptions:
#   level:
#   user:
#   role:
#   type:
supplementalGroups:
  type: RunAsAny
```

```
# This can be customized for your host machine
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
# If you want a priority on your SCC -- set for a value more than 0
priority: 11
users:
{{- if .Values.serviceAccount.name }}
- system:serviceaccount:{{ .Release.Namespace }}:{{ .Values.serviceAccount.name
}}
{{- else  }}
- system:serviceaccount:{{ .Release.Namespace }}:{{ .Release.Name }}-sonarqube
{{- end }}
{{- if .Values.postgresql.securityContext.enabled }}
- system:serviceaccount:{{ .Release.Namespace }}:{{ .Release.Name }}-postgresql
{{- end }}


{{- end }}
/sonarqube/chart/templates/sonarqube-sts.yaml
{{- if eq .Values.deploymentType "StatefulSet"}}
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: {{ template "sonarqube.fullname" . }}
  labels:
```

```
    app: {{ template "sonarqube.name" . }}
    chart: {{ .Chart.Name }}-{{ .Chart.Version | replace "+" "_" }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
    app.kubernetes.io/name: {{ template "sonarqube.name" . }}-{{ template
"sonarqube.fullname" . }}
    app.kubernetes.io/instance: {{ .Release.Name }}
    app.kubernetes.io/managed-by: {{ .Release.Service }}
    app.kubernetes.io/part-of: sonarqube
    app.kubernetes.io/component: {{ template "sonarqube.fullname" . }}
    app.kubernetes.io/version: {{ tpl .Values.image.tag . | quote }}
spec:
  replicas: {{ .Values.replicaCount }}
  serviceName: {{ template "sonarqube.fullname" . }}
  selector:
    matchLabels:
      app: {{ template "sonarqube.name" . }}
      release: {{ .Release.Name }}
  template:
    metadata:
      labels:
        app: {{ template "sonarqube.name" . }}
        release: {{ .Release.Name }}
{{- with .Values.podLabels }}
{{ toYaml . | indent 8 }}
{{- end }}
      annotations:
        checksum/init-sysctl: {{ include (print $.Template.BasePath "/init-sysctl.yaml")
. | sha256sum }}
```

```
      checksum/init-fs: {{ include (print $.Template.BasePath "/init-fs.yaml") . | sha256sum }}
      checksum/plugins: {{ include (print $.Template.BasePath "/install-plugins.yaml") . | sha256sum }}
      checksum/config: {{ include (print $.Template.BasePath "/config.yaml") . | sha256sum }}
      checksum/secret: {{ include (print $.Template.BasePath "/secret.yaml") . | sha256sum }}
{{- if .Values.prometheusExporter.enabled }}
      checksum/prometheus-config: {{ include (print $.Template.BasePath "/prometheus-config.yaml") . | sha256sum }}
      checksum/prometheus-ce-config: {{ include (print $.Template.BasePath "/prometheus-ce-config.yaml") . | sha256sum }}
{{- end }}
{{- if .Values.annotations}}
    {{- range $key, $value := .Values.annotations }}
      {{ $key }}: {{ $value | quote }}
    {{- end }}
{{- end }}
   spec:
    securityContext:
{{ toYaml .Values.securityContext | indent 8 }}
    {{- if or .Values.image.pullSecrets .Values.image.pullSecret }}
    imagePullSecrets:
      {{- if .Values.image.pullSecret }}
      - name: {{ .Values.image.pullSecret }}
      {{- end }}
      {{- if .Values.image.pullSecrets}}
{{ toYaml .Values.image.pullSecrets | indent 8 }}
      {{- end }}
```

```
    {{- end }}
    initContainers:
    {{- if .Values.extraInitContainers }}
{{ toYaml .Values.extraInitContainers | indent 8 }}
    {{- end }}
    {{- if .Values.postgresql.enabled }}
      - name: "wait-for-db"
        image: {{ default "busybox:1.32" .Values.initContainers.image }}
        imagePullPolicy: {{ .Values.image.pullPolicy  }}
        {{- if $securityContext := .Values.initContainers.securityContext }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml .Values.initContainers.resources | indent 12 }}
        command: ["/bin/sh", "-c", "for i in $(seq 1 200); do nc -z -w3 {{
.Release.Name}}-postgresql 5432 && exit 0 || sleep 2; done; exit 1"]
    {{- end }}
    {{- if .Values.caCerts.enabled }}
      - name: ca-certs
        image: {{ default "adoptopenjdk/openjdk11:alpine" .Values.caCerts.image }}
        imagePullPolicy: {{ .Values.image.pullPolicy  }}
        command: ["sh"]
        args: ["-c", "cp -f \"${JAVA_HOME}/lib/security/cacerts\" /tmp/certs/cacerts;
if [ \"$(ls /tmp/secrets/ca-certs)\" ]; then for f in /tmp/secrets/ca-certs/*; do keytool -
importcert -file \"${f}\" -alias \"$(basename \"${f}\")\" -keystore /tmp/certs/cacerts -
storepass changeit -trustcacerts -noprompt; done; fi;"]
        {{- if $securityContext := .Values.initContainers.securityContext }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
```

```
        {{- end }}
        resources:
{{ toYaml .Values.initContainers.resources | indent 12 }}
        volumeMounts:
          - mountPath: /tmp/certs
            name: sonarqube
            subPath: certs
          - mountPath: /tmp/secrets/ca-certs
            name: ca-certs
        {{- with .Values.env }}
        env:
          {{- . | toYaml | trim | nindent 12 }}
        {{- end }}
    {{- end }}
    {{- if or .Values.initSysctl.enabled .Values.elasticsearch.configureNode }}
      - name: init-sysctl
        image: {{ default "busybox:1.32" .Values.initSysctl.image }}
        imagePullPolicy: {{ .Values.image.pullPolicy  }}
        {{- if $securityContext := (default .Values.initContainers.securityContext
.Values.initSysctl.securityContext) }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml (default .Values.initContainers.resources .Values.initSysctl.resources) |
indent 12 }}
        command: ["sh",
          "-e",
          "/tmp/scripts/init_sysctl.sh"]
```

```
        volumeMounts:
          - name: init-sysctl
            mountPath: /tmp/scripts/
        {{- with .Values.env }}
        env:
          {{- . | toYaml | trim | nindent 12 }}
        {{- end }}
      {{- end }}


      {{- if or .Values.sonarProperties .Values.sonarSecretProperties
.Values.sonarSecretKey (not .Values.elasticsearch.bootstrapChecks) }}
        - name: concat-properties
          image: {{ default "busybox:1.32" .Values.initContainers.image }}
          imagePullPolicy: {{ .Values.image.pullPolicy  }}
          command:
          - sh
          - -c
          - |
            #!/bin/sh
            if [ -f /tmp/props/sonar.properties ]; then
              cat /tmp/props/sonar.properties > /tmp/result/sonar.properties
            fi
            if [ -f /tmp/props/secret.properties ]; then
              cat /tmp/props/secret.properties > /tmp/result/sonar.properties
            fi
            if [ -f /tmp/props/sonar.properties -a -f /tmp/props/secret.properties ]; then
              awk 1 /tmp/props/sonar.properties /tmp/props/secret.properties >
/tmp/result/sonar.properties
            fi
```

```
        volumeMounts:
        {{- if or .Values.sonarProperties .Values.sonarSecretKey (not
.Values.elasticsearch.bootstrapChecks) }}
          - mountPath: /tmp/props/sonar.properties
            name: config
            subPath: sonar.properties
        {{- end }}
        {{- if .Values.sonarSecretProperties }}
          - mountPath: /tmp/props/secret.properties
            name: secret-config
            subPath: secret.properties
        {{- end }}
          - mountPath: /tmp/result
            name: concat-dir
        {{- if $securityContext := .Values.initContainers.securityContext }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml .Values.initContainers.resources | indent 12 }}
        {{- with .Values.env }}
        env:
          {{- . | toYaml | trim | nindent 12 }}
        {{- end }}
    {{- end }}

    {{- if .Values.prometheusExporter.enabled }}
      - name: inject-prometheus-exporter
```

```
        image: {{ default "curlimages/curl:7.76.1" .Values.prometheusExporter.image
}}
        imagePullPolicy: {{ .Values.image.pullPolicy  }}
        {{- if $securityContext := (default .Values.initContainers.securityContext
.Values.prometheusExporter.securityContext) }}
        securityContext:
{{ toYaml $securityContext | indent 12 }}
        {{- end }}
        resources:
{{ toYaml (default .Values.initContainers.resources
.Values.prometheusExporter.resources) | indent 12 }}
        command: ["/bin/sh","-c"]
        args: ["curl -s '{{ template "prometheusExporter.downloadURL" . }}' {{ if
$.Values.prometheusExporter.noCheckCertificate }}--insecure{{ end }} --output
/data/jmx_prometheus_javaagent.jar -v"]
        volumeMounts:
          - mountPath: /data
            name: sonarqube
            subPath: data
        env:
          - name: http_proxy
            value: {{ default "" .Values.prometheusExporter.httpProxy }}
          - name: https_proxy
            value: {{ default "" .Values.prometheusExporter.httpsProxy }}
          - name: no_proxy
            value: {{ default "" .Values.prometheusExporter.noProxy }}
        {{- with .Values.env }}
          {{- . | toYaml | trim | nindent 12 }}
        {{- end }}
    {{- end }}
```

```
    {{- if and .Values.persistence.enabled .Values.initFs.enabled }}
     - name: init-fs
       image: {{ default "busybox:1.32" .Values.initFs.image }}
       imagePullPolicy: {{ .Values.image.pullPolicy  }}
       {{- if $securityContext := (default .Values.initContainers.securityContext
.Values.initFs.securityContext) }}
       securityContext:
{{ toYaml $securityContext | indent 12 }}
       {{- end }}
       resources:
{{ toYaml (default .Values.initContainers.resources .Values.initFs.resources) | indent
12 }}
       command: ["sh",
        "-e",
        "/tmp/scripts/init_fs.sh"]
       volumeMounts:
        - name: init-fs
          mountPath: /tmp/scripts/
{{- if .Values.persistence.mounts }}
{{ toYaml .Values.persistence.mounts | indent 12 }}
{{- end }}
        {{- if .Values.caCerts.enabled }}
        - mountPath: {{ .Values.sonarqubeFolder }}/certs
          name: sonarqube
          subPath: certs
        {{- end }}
        - mountPath: {{ .Values.sonarqubeFolder }}/data
          name: sonarqube
          subPath: data
```

```
      {{- if .Values.persistence.enabled }}
      - mountPath: {{ .Values.sonarqubeFolder }}/extensions
        name: sonarqube
        subPath: extensions
      {{- else if .Values.plugins.install }}
      - mountPath: {{ .Values.sonarqubeFolder }}/extensions/plugins
        name: sonarqube
        subPath: extensions/plugins
      {{- end }}
      - mountPath: {{ .Values.sonarqubeFolder }}/temp
        name: sonarqube
        subPath: temp
      - mountPath: {{ .Values.sonarqubeFolder }}/logs
        name: sonarqube
        subPath: logs
      - mountPath: /tmp
        name: tmp-dir
{{- end }}
{{- if .Values.plugins.install }}
  - name: install-plugins
    image: {{ default "curlimages/curl:7.76.1" .Values.plugins.image }}
    imagePullPolicy: {{ .Values.image.pullPolicy  }}
    command: ["sh",
      "-e",
      "/tmp/scripts/install_plugins.sh"]
    volumeMounts:
      - mountPath: {{ .Values.sonarqubeFolder }}/extensions/plugins
        name: sonarqube
```

```
          subPath: extensions/plugins
        - name: install-plugins
          mountPath: /tmp/scripts/
        {{- if .Values.plugins.netrcCreds }}
        - name: plugins-netrc-file
          mountPath: /root
        {{- end }}
      {{- if $securityContext := (default .Values.initContainers.securityContext
.Values.plugins.securityContext) }}
      securityContext:
{{ toYaml $securityContext | indent 12 }}
      {{- end }}
      resources:
{{ toYaml (default .Values.initContainers.resources .Values.plugins.resource) |
indent 12 }}
      env:
        - name: http_proxy
          value: {{ default "" .Values.plugins.httpProxy }}
        - name: https_proxy
          value: {{ default "" .Values.plugins.httpsProxy }}
        - name: no_proxy
          value: {{ default "" .Values.plugins.noProxy }}
      {{- with .Values.env }}
        {{- . | toYaml | trim | nindent 12 }}
      {{- end }}
  {{- end }}
  containers:
  {{- if .Values.extraContainers }}
    {{- toYaml .Values.extraContainers | nindent 8 }}
```

```
    {{- end }}
     - name: {{ .Chart.Name }}
       image: "{{ .Values.image.repository }}:{{ tpl .Values.image.tag . }}"
       imagePullPolicy: {{ .Values.image.pullPolicy }}
       ports:
        - name: http
          containerPort: {{ .Values.service.internalPort }}
          protocol: TCP
        {{- if .Values.prometheusExporter.enabled }}
        - name: monitoring-web
          containerPort: {{ .Values.prometheusExporter.webBeanPort }}
          protocol: TCP
        - name: monitoring-ce
          containerPort: {{ .Values.prometheusExporter.ceBeanPort }}
          protocol: TCP
        {{- end }}
       resources:
{{ toYaml (default .Values.resources .Values.resource) | indent 12 }}
       env:
        - name: SONAR_WEB_JAVAOPTS
          value: {{ template "sonarqube.jvmOpts" . }}
        - name: SONAR_CE_JAVAOPTS
          value: {{ template "sonarqube.jvmCEOpts" . }}
        - name: SONAR_JDBC_PASSWORD
          valueFrom:
            secretKeyRef:
              name: {{ template "jdbc.secret" . }}
              key: {{ template "jdbc.secretPasswordKey" . }}
```

```
          - name: SONAR_WEB_SYSTEMPASSCODE
            valueFrom:
              secretKeyRef:
              {{- if and .Values.monitoringPasscodeSecretName
.Values.monitoringPasscodeSecretKey }}
                name: {{ .Values.monitoringPasscodeSecretName }}
                key: {{ .Values.monitoringPasscodeSecretKey }}
              {{- else }}
                name: {{ template "sonarqube.fullname" . }}-monitoring-passcode
                key: SONAR_WEB_SYSTEMPASSCODE
              {{- end }}
          {{- with .Values.env }}
          {{- . | toYaml | trim | nindent 12 }}
          {{- end }}
        envFrom:
          - configMapRef:
              name: {{ template "sonarqube.fullname" . }}-jdbc-config
{{- range .Values.extraConfig.secrets }}
          - secretRef:
              name: {{ . }}
{{- end }}
{{- range .Values.extraConfig.configmaps }}
          - configMapRef:
              name: {{ . }}
{{- end }}
        livenessProbe:
          exec:
            command:
            - sh
```

```
        - -c
        - |
          host="$(hostname -i || echo '127.0.0.1')"
          reply=$(wget -qO- --header="X-Sonar-Passcode:
$SONAR_WEB_SYSTEMPASSCODE" http://${host}:{{
.Values.service.internalPort }}{{ .Values.livenessProbe.sonarWebContext
}}api/system/liveness 2>&1)
          if [ -z "$reply" ]; then exit 0; else exit 1; fi
        initialDelaySeconds: {{ .Values.livenessProbe.initialDelaySeconds }}
        periodSeconds: {{ .Values.livenessProbe.periodSeconds }}
        failureThreshold: {{ .Values.livenessProbe.failureThreshold }}
      readinessProbe:
        exec:
          command:
          - sh
          - -c
          - |
            #!/bin/bash
            # A Sonarqube container is considered ready if the status is UP,
DB_MIGRATION_NEEDED or DB_MIGRATION_RUNNING
            # status about migration are added to prevent the node to be kill while
sonarqube is upgrading the database.
            host="$(hostname -i || echo '127.0.0.1')"
            if wget --proxy off -qO- http://${host}:{{ .Values.service.internalPort }}{{
.Values.readinessProbe.sonarWebContext }}api/system/status | grep -q -e
'"status":"UP"' -e '"status":"DB_MIGRATION_NEEDED"' -e
'"status":"DB_MIGRATION_RUNNING"'; then
                exit 0
            fi
            exit 1
        initialDelaySeconds: {{ .Values.readinessProbe.initialDelaySeconds }}
```

```
          periodSeconds: {{ .Values.readinessProbe.periodSeconds }}
          failureThreshold: {{ .Values.readinessProbe.failureThreshold }}
        startupProbe:
          httpGet:
            scheme: HTTP
            path: {{ .Values.readinessProbe.sonarWebContext }}api/system/status
            port: http
          initialDelaySeconds: {{ .Values.startupProbe.initialDelaySeconds }}
          periodSeconds: {{ .Values.startupProbe.periodSeconds }}
          failureThreshold: {{ .Values.startupProbe.failureThreshold }}
        {{- if .Values.containerSecurityContext }}
        securityContext:
{{- toYaml .Values.containerSecurityContext | nindent 12 }}
        {{- end }}
        volumeMounts:
{{- if .Values.persistence.mounts }}
{{ toYaml .Values.persistence.mounts | indent 12 }}
{{- end }}
          {{- if or .Values.sonarProperties .Values.sonarSecretProperties
.Values.sonarSecretKey (not .Values.elasticsearch.bootstrapChecks) }}
          - mountPath: {{ .Values.sonarqubeFolder }}/conf/
            name: concat-dir
          {{- end }}
          {{- if .Values.sonarSecretKey }}
          - mountPath: {{ .Values.sonarqubeFolder }}/secret/
            name: secret
          {{- end }}
          {{- if .Values.caCerts.enabled }}
          - mountPath: {{ .Values.sonarqubeFolder }}/certs
```

```
  name: sonarqube
  subPath: certs
{{- end }}
- mountPath: {{ .Values.sonarqubeFolder }}/data
  name: sonarqube
  subPath: data
{{- if .Values.persistence.enabled }}
- mountPath: {{ .Values.sonarqubeFolder }}/extensions
  name: sonarqube
  subPath: extensions
{{- else if .Values.plugins.install }}
- mountPath: {{ .Values.sonarqubeFolder }}/extensions/plugins
  name: sonarqube
  subPath: extensions/plugins
{{- end }}
- mountPath: {{ .Values.sonarqubeFolder }}/temp
  name: sonarqube
  subPath: temp
- mountPath: {{ .Values.sonarqubeFolder }}/logs
  name: sonarqube
  subPath: logs
- mountPath: /tmp
  name: tmp-dir
{{- if .Values.prometheusExporter.enabled }}
- mountPath: {{ .Values.sonarqubeFolder }}/conf/prometheus-config.yaml
  subPath: prometheus-config.yaml
  name: prometheus-config
- mountPath: {{ .Values.sonarqubeFolder }}/conf/prometheus-ce-config.yaml
```

```
            subPath: prometheus-ce-config.yaml
            name: prometheus-ce-config
          {{- end }}
    {{- if .Values.nodeSelector }}
      nodeSelector:
{{ toYaml .Values.nodeSelector | indent 8 }}
    {{- end }}
    {{- if .Values.hostAliases }}
      hostAliases:
{{ toYaml .Values.hostAliases | indent 8 }}
    {{- end }}
    {{- if .Values.tolerations }}
      tolerations:
{{ toYaml .Values.tolerations | indent 8 }}
    {{- end }}
    {{- if .Values.affinity }}
      affinity:
{{ toYaml .Values.affinity | indent 8 }}
    {{- end }}
      serviceAccountName: {{ template "sonarqube.serviceAccountName" . }}
      volumes:
{{- if .Values.persistence.volumes }}
{{ tpl (toYaml .Values.persistence.volumes | indent 6) . }}
{{- end }}
      {{- if or .Values.sonarProperties .Values.sonarSecretKey ( not
.Values.elasticsearch.bootstrapChecks) }}
      - name: config
        configMap:
          name: {{ template "sonarqube.fullname" . }}-config
```

```
    items:
    - key: sonar.properties
      path: sonar.properties
{{- end }}
{{- if .Values.sonarSecretProperties }}
- name: secret-config
  secret:
    secretName: {{ .Values.sonarSecretProperties }}
    items:
    - key: secret.properties
      path: secret.properties
{{- end }}
{{- if .Values.sonarSecretKey }}
- name: secret
  secret:
    secretName: {{ .Values.sonarSecretKey }}
    items:
    - key: sonar-secret.txt
      path: sonar-secret.txt
{{- end }}
{{- if .Values.caCerts.enabled }}
- name: ca-certs
  secret:
    secretName: {{ .Values.caCerts.secret }}
{{- end }}
{{- if .Values.plugins.netrcCreds }}
- name: plugins-netrc-file
  secret:
```

```
    secretName: {{ .Values.plugins.netrcCreds }}
    items:
    - key: netrc
      path: .netrc
{{- end }}
- name: init-sysctl
  configMap:
    name: {{ template "sonarqube.fullname" . }}-init-sysctl
    items:
      - key: init_sysctl.sh
        path: init_sysctl.sh
- name: init-fs
  configMap:
    name: {{ template "sonarqube.fullname" . }}-init-fs
    items:
      - key: init_fs.sh
        path: init_fs.sh
- name: install-plugins
  configMap:
    name: {{ template "sonarqube.fullname" . }}-install-plugins
    items:
      - key: install_plugins.sh
        path: install_plugins.sh
{{- if .Values.prometheusExporter.enabled }}
- name: prometheus-config
  configMap:
    name: {{ template "sonarqube.fullname" . }}-prometheus-config
    items:
```

```
          - key: prometheus-config.yaml
            path: prometheus-config.yaml
      - name: prometheus-ce-config
        configMap:
          name: {{ template "sonarqube.fullname" . }}-prometheus-ce-config
          items:
          - key: prometheus-ce-config.yaml
            path: prometheus-ce-config.yaml
      {{- end }}
      - name: sonarqube
        {{- if .Values.persistence.enabled }}
        persistentVolumeClaim:
          claimName: {{ if .Values.persistence.existingClaim }}{{
.Values.persistence.existingClaim }}{{- else }}{{ template "sonarqube.fullname" .
}}{{- end }}
        {{- else }}
        emptyDir: {{- toYaml .Values.emptyDir | nindent 10 }}
        {{- end  }}
      - name : tmp-dir
        emptyDir: {{- toYaml .Values.emptyDir | nindent 10 }}
        {{- if or .Values.sonarProperties .Values.sonarSecretProperties
.Values.sonarSecretKey ( not .Values.elasticsearch.bootstrapChecks) }}
      - name : concat-dir
        emptyDir: {{- toYaml .Values.emptyDir | nindent 10 -}}
        {{- end }}
{{- end }}
```

/sonarqube/chart/values.yaml

```
# Default values for sonarqube.
# This is a YAML-formatted file.
```

```
# Declare variables to be passed into your templates.

# If the deployment Type is set to Deployment sonarqube is deployed as a replica set.
deploymentType: "StatefulSet"

# There should not be more than 1 sonarqube instance connected to the same database. Please set this value to 1 or 0 (in case you need to scale down programmatically).
replicaCount: 1

# This will use the default deployment strategy unless it is overriden
deploymentStrategy: {}
# Uncomment this to scheduler pods on priority
# priorityClassName: "high-priority"

## Use an alternate scheduler, e.g. "stork".
## ref: https://kubernetes.io/docs/tasks/administer-cluster/configure-multiple-schedulers/
##
# schedulerName:

## Is this deployment for OpenShift? If so, we help with SCCs
OpenShift:
  enabled: false
  createSCC: true

edition: "community"

image:
```

```
  repository: sonarqube
  tag: 9.7.1-{{ .Values.edition }}
  pullPolicy: IfNotPresent
  # If using a private repository, the imagePullSecrets to use
  # pullSecrets:
  #   - name: my-repo-secret

# Set security context for sonarqube pod
securityContext:
  fsGroup: 1000

# Set security context for sonarqube container
containerSecurityContext:
  # Sonarqube dockerfile creates sonarqube user as UID and GID 1000
  runAsUser: 1000

# Settings to configure elasticsearch host requirements
elasticsearch:
  # DEPRECATED: Use initSysctl.enabled instead
  configureNode: true
  bootstrapChecks: true

service:
  type: ClusterIP
  externalPort: 9000
  internalPort: 9000
  labels:
  annotations: {}
```

```
  # May be used in example for internal load balancing in GCP:
  # cloud.google.com/load-balancer-type: Internal
  # loadBalancerSourceRanges:
  #   - 0.0.0.0/0
  # loadBalancerIP: 1.2.3.4

# Optionally create Network Policies
networkPolicy:
  enabled: false

  # If you plan on using the jmx exporter, you need to define where the traffic is coming from
  prometheusNamespace: "monitoring"

  # If you are using a external database and enable network Policies to be created
  # you will need to explicitly allow egress traffic to your database
  # expects https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.21/#networkpolicyspec-v1-networking-k8s-io
  # additionalNetworkPolicys:

# also install the nginx ingress helm chart
nginx:
  enabled: false

ingress:
  enabled: false
  # Used to create an Ingress record.
  hosts:
    - name: sonarqube.{{ .Values.global.global.domainName }}
```

```
    # Different clouds or configurations might need /* as the default path
    path: /
    # For additional control over serviceName and servicePort
    # serviceName: someService
    # servicePort: somePort
    # the pathType can be one of the following values:
Exact|Prefix|ImplementationSpecific(default)
    # pathType: ImplementationSpecific
  annotations: {}
  # kubernetes.io/ingress.class: nginx
  # kubernetes.io/tls-acme: "true"
  # This property allows for reports up to a certain size to be uploaded to SonarQube
  # nginx.ingress.kubernetes.io/proxy-body-size: "8m"

  # Set ingressClassName if kubernetes version is >= 1.18
  # Reference: https://kubernetes.io/blog/2020/04/02/improvements-to-the-ingress-
api-in-kubernetes-1.18/
  # ingressClassName: nginx

# Additional labels for Ingress manifest file
  # labels:
  #  traffic-type: external
  #  traffic-type: internal
  tls: []
  # Secrets must be manually created in the namespace. To generate a self-signed
certificate (and private key) and then create the secret in the cluster please refer to
official documentation available at https://kubernetes.github.io/ingress-nginx/user-
guide/tls/#tls-secrets
  # - secretName: chart-example-tls
  #   hosts:
```

```
  #     - chart-example.local


route:
  enabled: false
  host: ""
  # Add tls section to secure traffic. TODO: extend this section with other secure route settings
  # Comment this out if you want plain http route created.
  tls:
    termination: edge


  annotations: {}
  # See Openshift/OKD route annotation
  # https://docs.openshift.com/container-platform/4.10/networking/routes/route-configuration.html#nw-route-specific-annotations_route-configuration
  # haproxy.router.openshift.io/timeout: 1m


  # Additional labels for Route manifest file
  # labels:
  #  external: 'true'


# Affinity for pod assignment
# Ref: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity
affinity: {}


# Tolerations for pod assignment
# Ref: https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/
# taint a node with the following command to mark it as not schedulable for new pods
```

```
# kubectl taint nodes <node> sonarqube=true:NoSchedule
# The following statement will tolerate this taint and as such reverse a node for sonarqube
tolerations: []
#  - key: "sonarqube"
#    operator: "Equal"
#    value: "true"
#    effect: "NoSchedule"

# Node labels for pod assignment
# Ref: https://kubernetes.io/docs/user-guide/node-selection/
# add a label to a node with the following command
# kubectl label node <node> sonarqube=true
nodeSelector: {}
#  sonarqube: "true"

# hostAliases allows the modification of the hosts file inside a container
hostAliases: []
# - ip: "192.168.1.10"
#   hostnames:
#   - "example.com"
#   - "www.example.com"

readinessProbe:
  initialDelaySeconds: 60
  periodSeconds: 30
  failureThreshold: 6
  # If an ingress *path* other than the root (/) is defined, it should be reflected here
  # A trailing "/" must be included
```

```
  sonarWebContext: /
  # sonarWebContext: /sonarqube/

livenessProbe:
  initialDelaySeconds: 60
  periodSeconds: 30
  failureThreshold: 6
  # If an ingress *path* other than the root (/) is defined, it should be reflected here
  # A trailing "/" must be included
  sonarWebContext: /
  # sonarWebContext: /sonarqube/
  # If an ingress *path* is defined, it should be reflected here
  # sonar.web.context: /sonarqube

startupProbe:
  initialDelaySeconds: 30
  periodSeconds: 10
  failureThreshold: 24
  # If an ingress *path* other than the root (/) is defined, it should be reflected here
  # A trailing "/" must be included
  sonarWebContext: /
  # sonarWebContext: /sonarqube/

initContainers:
  # image: busybox:1.32
  # We allow the init containers to have a separate security context declaration
because
  # the initContainer may not require the same as SonarQube.
  # securityContext: {}
```

```
  # We allow the init containers to have a separate resources declaration because
  # the initContainer does not take as much resources.
  resources: {}

# Extra init containers to e.g. download required artifacts
extraInitContainers: {}

## Array of extra containers to run alongside the sonarqube container
##
## Example:
## - name: myapp-container
##   image: busybox
##   command: ['sh', '-c', 'echo Hello && sleep 3600']
##
extraContainers: []

## Provide a secret containing one or more certificate files in the keys that will be added to cacerts
## The cacerts file will be set via SONARQUBE_WEB_JVM_OPTS and SONAR_CE_JAVAOPTS
##
caCerts:
  enabled: false
  image: adoptopenjdk/openjdk11:alpine
  secret: your-secret

initSysctl:
  enabled: true
  vmMaxMapCount: 524288
```

```
  fsFileMax: 131072
  nofile: 131072
  nproc: 8192
  # image: busybox:1.32
  securityContext:
    privileged: true
  # resources: {}

initFs:
  enabled: true
  # image: busybox:1.32
  securityContext:
    privileged: true

prometheusExporter:
  enabled: true
  # jmx_prometheus_javaagent version to download from Maven Central
  version: "0.16.0"
  # Alternative full download URL for the jmx_prometheus_javaagent.jar (overrides prometheusExporter.version)
  # downloadURL: ""
  # if you need to ignore TLS certificates for whatever reason enable the following flag
  noCheckCertificate: false

  # Ports for the jmx prometheus agent to export metrics at
  webBeanPort: 8000
  ceBeanPort: 8001
```

```
  config:
    rules:
      - pattern: ".*"
  # Overrides config for the CE process Prometheus exporter (by default, the same rules are used for both the Web and CE processes).
  # ceConfig:
  #   rules:
  #     - pattern: ".*"
  # image: curlimages/curl:7.76.1
  # For use behind a corporate proxy when downloading prometheus
  # httpProxy: ""
  # httpsProxy: ""
  # noProxy: ""
  # Setting the security context to the default sonarqube user 1000/1000
  securityContext:
    runAsUser: 1000
    runAsGroup: 1000


# List of plugins to install.
# For example:
# plugins:
#  install:
#    - "https://github.com/AmadeusITGroup/sonar-stash/releases/download/1.3.0/sonar-stash-plugin-1.3.0.jar"
#    - "https://github.com/SonarSource/sonar-ldap/releases/download/2.2-RC3/sonar-ldap-plugin-2.2.0.601.jar"
#
plugins:
  install: []
```

```
  # For use behind a corporate proxy when downloading plugins
  # httpProxy: ""
  # httpsProxy: ""
  # noProxy: ""

  # image: curlimages/curl:7.76.1
  # resources: {}

  # .netrc secret file with a key "netrc" to use basic auth while downloading plugins
  # netrcCreds: ""

  # Set to true to not validate the server's certificate to download plugin
  noCheckCertificate: false
  securityContext:
    runAsUser: 1000
    runAsGroup: 1000

## Values to add to SONARQUBE_WEB_JVM_OPTS
##
# jvmOpts: "-Djava.net.preferIPv4Stack=true"
jvmOpts: ""

## Values to add to SONAR_CE_JAVAOPTS
jvmCeOpts: ""

## a monitoring passcode needs to be defined in order to get reasonable probe results
# not setting the monitoring passcode will result in a deployment that will never be
ready
```

```
monitoringPasscode: "define_it"

# Alternatively, you can define the passcode loading it from an existing secret specifying the right key

# monitoringPasscodeSecretName: "pass-secret-name"

# monitoringPasscodeSecretKey: "pass-key"


## Environment variables to attach to the pods

##

# env:

#   # If you use a different ingress path from /, you have to add it here as the value of SONAR_WEB_CONTEXT

#   - name: SONAR_WEB_CONTEXT

#     value: /sonarqube

#   - name: VARIABLE

#     value: my-value


# Set annotations for pods

annotations: {}


## We usually don't make specific ressource recommandations, as they are heavily dependend on

## The usage of SonarQube and the surrounding infrastructure.

## Adjust these values to your needs, but make sure that the memory limit is never under 4 GB

resources:

  limits:

    cpu: 800m

    memory: 4Gi

  requests:

    cpu: 400m
```

```
    memory: 2Gi


persistence:
  enabled: false
  ## Set annotations on pvc
  annotations: {}


  ## Specify an existing volume claim instead of creating a new one.
  ## When using this option all following options like storageClass, accessMode and size are ignored.
  # existingClaim:


  ## If defined, storageClassName: <storageClass>
  ## If set to "-", storageClassName: "", which disables dynamic provisioning
  ## If undefined (the default) or set to null, no storageClassName spec is
  ##   set, choosing the default provisioner.  (gp2 on AWS, standard on
  ##   GKE, AWS & OpenStack)
  ##
  storageClass:
  accessMode: ReadWriteOnce
  size: 5Gi
  uid: 1000


  ## Specify extra volumes. Refer to ".spec.volumes" specification : https://kubernetes.io/fr/docs/concepts/storage/volumes/
  volumes: []
  ## Specify extra mounts. Refer to ".spec.containers.volumeMounts" specification : https://kubernetes.io/fr/docs/concepts/storage/volumes/
  mounts: []
```

```
# In case you want to specify different resources for emptyDir than {}
emptyDir: {}
  # Example of resouces that might be used:
  # medium: Memory
  # sizeLimit: 16Mi

# A custom sonar.properties file can be provided via dictionary.
# For example:
# sonarProperties:
#   sonar.forceAuthentication: true
#   sonar.security.realm: LDAP
#   ldap.url: ldaps://organization.com

# Additional sonar properties to load from a secret with a key "secret.properties" (must be a string)
# sonarSecretProperties:

# Kubernetes secret that contains the encryption key for the sonarqube instance.
# The secret must contain the key 'sonar-secret.txt'.
# The 'sonar.secretKeyPath' property will be set automatically.
# sonarSecretKey: "settings-encryption-secret"

## Override JDBC values
## for external Databases
jdbcOverwrite:
  # If enable the JDBC Overwrite, make sure to set `postgresql.enabled=false`
  enable: false
  # The JDBC url of the external DB
```

```
  jdbcUrl: "jdbc:postgresql://myPostgress/myDatabase?socketTimeout=1500"
  # The DB user that should be used for the JDBC connection
  jdbcUsername: "sonarUser"
  # Use this if you don't mind the DB password getting stored in plain text within the values file
  jdbcPassword: "sonarPass"
  ## Alternatively, use a pre-existing k8s secret containing the DB password
  # jdbcSecretName: "sonarqube-jdbc"
  ## and the secretValueKey of the password found within that secret
  # jdbcSecretPasswordKey: "jdbc-password"

## Configuration values for postgresql dependency
## ref: https://github.com/bitnami/charts/blob/master/bitnami/postgresql/README.md
postgresql:
  # Enable to deploy the bitnami PostgreSQL chart
  enabled: true
  ## postgresql Chart global settings
  # global:
  #   imageRegistry: ''
  #   imagePullSecrets: ''
  ## bitnami/postgres image tag
  # image:
  #   tag: 11.7.0-debian-10-r9
  # existingSecret Name of existing secret to use for PostgreSQL passwords
  # The secret has to contain the keys postgresql-password which is the password for postgresqlUsername when it is
  # different of postgres, postgresql-postgres-password which will override postgresqlPassword,
```

```
  # postgresql-replication-password which will override replication.password and
postgresql-ldap-password which will be
  # used to authenticate on LDAP. The value is evaluated as a template.
  # existingSecret: ""
  #
  # The bitnami chart enforces the key to be "postgresql-password". This value is only
here for historic purposes
  # existingSecretPasswordKey: "postgresql-password"
  postgresqlUsername: "sonarUser"
  postgresqlPassword: "sonarPass"
  postgresqlDatabase: "sonarDB"
  # Specify the TCP port that PostgreSQL should use
  service:
    port: 5432
  resources:
    limits:
      cpu: 2
      memory: 2Gi
    requests:
      cpu: 100m
      memory: 200Mi
  persistence:
    enabled: true
    accessMode: ReadWriteOnce
    size: 20Gi
    storageClass:
  securityContext:
    # For standard Kubernetes deployment, set enabled=true
```

```
    # If using OpenShift, enabled=false for restricted SCC and enabled=true for
anyuid/nonroot SCC
    enabled: true
    # fsGroup specification below are not applied if enabled=false. enabled=false is the
required setting for OpenShift "restricted SCC" to work successfully.
    # postgresql dockerfile sets user as 1001
    fsGroup: 1001
  containerSecurityContext:
    # For standard Kubernetes deployment, set enabled=true
    # If using OpenShift, enabled=false for restricted SCC and enabled=true for
anyuid/nonroot SCC
    enabled: true
    # runAsUser specification below are not applied if enabled=false. enabled=false is
the required setting for OpenShift "restricted SCC" to work successfully.
    # postgresql dockerfile sets user as 1001
    runAsUser: 1001
  volumePermissions:
    # For standard Kubernetes deployment, set enabled=false
    # For OpenShift, set enabled=true and ensure to set
volumepermissions.securitycontext.runAsUser below.
    enabled: false
    # if using restricted SCC set runAsUser: "auto" and if running under
anyuid/nonroot SCC - runAsUser needs to match runAsUser above
    securityContext:
      runAsUser: 0
  shmVolume:
    chmod:
      enabled: false
  serviceAccount:
    ## If enabled = true, and name is not set, postgreSQL will create a serviceAccount
```

```
    enabled: false
    # name:

# Additional labels to add to the pods:
# podLabels:
#   key: value
podLabels: {}
# For compatibility with 8.0 replace by "/opt/sq"
# For compatibility with 8.2, leave the default. They changed it back to /opt/sonarqube
sonarqubeFolder: /opt/sonarqube

tests:
  image: bitnami/minideb-extras
  enabled: true
  resources: {}
  initContainers:
    image: bats/bats:1.2.1
    resources: {}

# For OpenShift set create=true to ensure service account is created.
serviceAccount:
  create: false
  # name:
  # automountToken: false # default
  ## Annotations for the Service Account
  annotations: {}

# extraConfig is used to load Environment Variables from Secrets and ConfigMaps
```

```
# which may have been written by other tools, such as external orchestrators.
#
# These Secrets/ConfigMaps are expected to contain Key/Value pairs, such as:
#
# apiVersion: v1
# kind: ConfigMap
# metadata:
#   name: external-sonarqube-opts
# data:
#   SONARQUBE_JDBC_USERNAME: foo
#   SONARQUBE_JDBC_URL: jdbc:postgresql://db.example.com:5432/sonar
#
# These vars can then be injected into the environment by uncommenting the following:
#
# extraConfig:
#   configmaps:
#     - external-sonarqube-opts

extraConfig:
  secrets: []
  configmaps: []

# account:
# The values can be set to define the current and the (new) custom admin passwords at the startup (the username will remain "admin")
#   adminPassword: admin
#   currentAdminPassword: admin
```

```
# The above values can be also provided by a secret that contains "password" and "currentPassword" as keys. You can generate such a secret in your cluster
# using "kubectl create secret generic admin-password-secret-name --from-literal=password=admin --from-literal=currentPassword=admin"
#   adminPasswordSecretName: ""
#   securityContext: {}
#   resources:
#     limits:
#       cpu: 100m
#       memory: 128Mi
#     requests:
#       cpu: 100m
#       memory: 128Mi
# curlContainerImage: curlimages/curl:latest
# adminJobAnnotations: {}
# sonarWebContext: /

terminationGracePeriodSeconds: 60
```

ДОДАТОК Г – Перелік зауважень нормоконтролера

Сторінка	Позначення	Зміст зауваження

[i]