ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА СИСТЕМ ТА ТЕХНОЛОГІЙ КІБЕРБЕЗПЕКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Технологія впровадження DevSECOps з використанням Sonarqube»

зі спеціальності *125 Кібербезпека та захист інформації*
(код, найменування спеціальності)

освітньо-професійної програми *Інформаційна та кібернетична безпека*
(назва програми)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Поліщук Артем
(підпис)

Виконав: здобувач вищої освіти групи БСДМ-61
Поліщук Артем
(прізвище, ім'я)

Керівник Марченко Віталій
(науковий ступінь, вчене звання, прізвище, ім'я)

Рецензент _____
(науковий ступінь, вчене звання, прізвище, ім'я)

Київ 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CI/CD – Continuous Integration / Continuous Deployment (безперервна інтеграція / безперервне розгортання)

SAST – Static Application Security Testing (статичне тестування безпеки додатків)

DAST – Dynamic Application Security Testing (динамічне тестування безпеки додатків)

IAST – Interactive Application Security Testing (інтерактивне тестування безпеки додатків)

DevSecOps – Development, Security, and Operations (розробка, безпека та експлуатація, інтегровані в єдиний процес)

IDE – Integrated Development Environment (інтегроване середовище розробки)

API – Application Programming Interface (інтерфейс програмування додатків)

XSS – Cross-Site Scripting (міжсайтовий скриптинг)

SCA – Software Composition Analysis (аналіз складу програмного забезпечення)

SOAR – Security Orchestration, Automation, and Response (оркестрація, автоматизація та реагування в області безпеки)

ВСТУП

Актуальність дослідження. Сучасне програмне забезпечення є основою ефективної діяльності більшості організацій, а вимоги до його якості та безпеки постійно зростають. Традиційні методи захисту, що застосовуються після завершення розробки, вже не забезпечують належного рівня безпеки в умовах стрімкого збільшення складності додатків та еволюції кіберзагроз. На цьому тлі методологія DevSecOps, яка інтегрує безпеку на всіх етапах життєвого циклу розробки, стає критично важливою для створення надійних і захищених програмних продуктів. Впровадження DevSecOps дозволяє не лише підвищити рівень безпеки, а й забезпечити оперативність та ефективність процесів розробки та розгортання. Інтеграція безпеки в конвеєри безперервної інтеграції та розгортання (CI/CD) є складним завданням, яке вимагає детального аналізу методів та інструментів, зокрема таких як SonarQube, GitLab та Jenkins. Недостатня увага до цього аспекту здатна призвести до вразливостей, які можуть бути використані зловмисниками, спричиняючи фінансові втрати та підрив репутації компаній.

Об'єкт дослідження – процеси забезпечення безпеки програмного забезпечення в умовах безперервної інтеграції та розгортання.

Предмет дослідження – методи та інструменти DevSecOps, а також підходи до їх ефективної інтеграції в конвеєри CI/CD.

Мета роботи – дослідити теоретичні засади та практичні аспекти впровадження DevSecOps із використанням інструментів SonarQube, GitLab та Jenkins, а також розробити рекомендації щодо ефективної інтеграції безпеки в процеси безперервної інтеграції та розгортання програмного забезпечення.

Наукові завдання:

вивчити теоретичні основи DevSecOps та визначити його відмінності від традиційних підходів до розробки й безпеки;

проаналізувати методи та засоби, що використовуються в DevSecOps для забезпечення безпеки на всіх етапах життєвого циклу ПЗ;

дослідити інструменти для інтеграції безпеки в процеси CI/CD (зокрема SonarQube, GitLab та Jenkins) й оцінити їх ефективність;

розробити рекомендації щодо впровадження DevSecOps у практику розробки програмного забезпечення з акцентом на інтеграції безпеки в конвеєри CI/CD.

Методи дослідження – аналіз літературних джерел та актуальної документації з тематики DevSecOps; вивчення технічних можливостей та вимог інструментів SonarQube, GitLab, Jenkins; моделювання сценаріїв інтеграції безпеки в CI/CD; оцінка та порівняння отриманих результатів.

Практичне значення одержаних результатів: формування методичних рекомендацій щодо впровадження DevSecOps із використанням SonarQube, GitLab та Jenkins. Запропоновані рішення дозволять організаціям підвищити рівень безпеки програмного забезпечення, скоротити час реагування на виявлені вразливості та мінімізувати ризик кіберзагроз, покращуючи при цьому ефективність процесів розробки та розгортання.

Апробація результатів. Основні висновки та результати дослідження були оприлюднені на Всеукраїнській науковій конференції «Актуальні проблеми кібербезпеки» (м. Київ, 02 листопада 2024 р.), де отримали позитивну оцінку та рекомендації до подальшого розвитку.

1 АНАЛІЗ ПРОБЛЕМИ ІНТЕГРАЦІЇ DEVSECOPS

1.1 Аналіз методів та засобів DevSecOps

У сучасному світі інформаційних технологій швидкість розробки та впровадження програмного забезпечення є критично важливою для конкурентоспроможності організацій. Однак, зі зростанням складності систем та кількості кіберзагроз, забезпечення безпеки стає невід'ємною частиною процесу розробки. У цьому контексті виникає підхід **DevSecOps**, який інтегрує безпеку в усі етапи життєвого циклу програмного забезпечення.

Методи DevSecOps

DevSecOps об'єднує практики розробки, експлуатації та безпеки, інтегруючи їх у єдиний безперервний процес. Це дозволяє забезпечити швидке та безпечне впровадження програмного забезпечення [1].

Інтеграція безпеки на ранніх етапах розробки (Shift Left)

Принцип "Shift Left" передбачає перенесення заходів з безпеки на найбільш ранні стадії життєвого циклу розробки програмного забезпечення. Це означає, що безпека враховується вже на етапах планування, проектування та написання коду.

Визначення вимог безпеки: Інтеграція вимог безпеки в специфікації та технічні завдання.

Моделювання загроз: Аналіз потенційних загроз та вразливостей під час проектування системи.

Навчання розробників: Підвищення обізнаності команди щодо принципів безпечного програмування.

Переваги:

- **Раннє виявлення вразливостей**: Виявлення та усунення проблем безпеки на ранніх етапах значно знижує вартість їх виправлення.

- **Зниження ризиків**: Проактивний підхід до безпеки зменшує ймовірність появи вразливостей у продакшн-середовищі.
- **Визначення вимог безпеки**: Інтеграція вимог безпеки в специфікації та технічні завдання.
- **Моделювання загроз**: Аналіз потенційних загроз та вразливостей під час проектування системи.
- **Навчання розробників**: Підвищення обізнаності команди щодо принципів безпечного програмування.

Безперервна інтеграція та розгортання з вбудованою безпекою

Інтеграція безпеки в конвеєри CI/CD забезпечує автоматизоване та безпечне постачання програмного забезпечення.

Налаштування конвеєрів з інтегрованими перевірками безпеки: Включення етапів тестування безпеки в CI/CD.

Використання Quality Gates: Автоматичне прийняття рішень про розгортання на основі результатів перевірок безпеки.

Контейнеризація та оркестрація: Використання Docker та Kubernetes з врахуванням найкращих практик безпеки.

Переваги:

- **Швидке впровадження змін:** Оновлення можуть бути швидко розгорнуті без компромісів у безпеці.

Впровадження політик та відповідності (Compliance as Code)

Compliance as Code передбачає автоматизацію перевірки відповідності програмного забезпечення та інфраструктури встановленим політикам безпеки та нормативним вимогам шляхом опису цих вимог у вигляді коду.

Опис політик у вигляді коду: Використання спеціалізованих мов та інструментів для формалізації політик безпеки.

Автоматизоване тестування відповідності: Інтеграція перевірок відповідності в конвеєри CI/CD.

Документування та аудит: Зберігання результатів перевірок для аудиту та звітності.

Переваги:

- **Швидке виявлення невідповідностей:** Автоматизовані перевірки дозволяють оперативно виявляти порушення політик.
- **Підтримка відповідності нормативам:** Допомагає дотримуватися стандартів, таких як GDPR, HIPAA, PCI DSS.

Засоби DevSecOps

DevSecOps базується на інтеграції безпеки в усі етапи життєвого циклу розробки та експлуатації програмного забезпечення. Для досягнення цієї мети використовуються різноманітні інструменти та технології, які забезпечують автоматизацію процесів, моніторинг безпеки та співпрацю між командами.

Інструменти статичного аналізу коду (SAST)

Статичний аналіз коду дозволяє виявляти вразливості та недоліки в коді без його виконання. Ці інструменти аналізують вихідний код на предмет потенційних проблем безпеки, порушень стандартів кодування та поганих практик.

Основні інструменти:

SonarQube: Платформа для безперервної інспекції коду з підтримкою багатьох мов програмування. Забезпечує глибокий аналіз коду, виявлення вразливостей та рекомендації щодо їх виправлення [4].

Fortify Static Code Analyzer: Комерційний інструмент для детального аналізу коду з акцентом на безпеку.

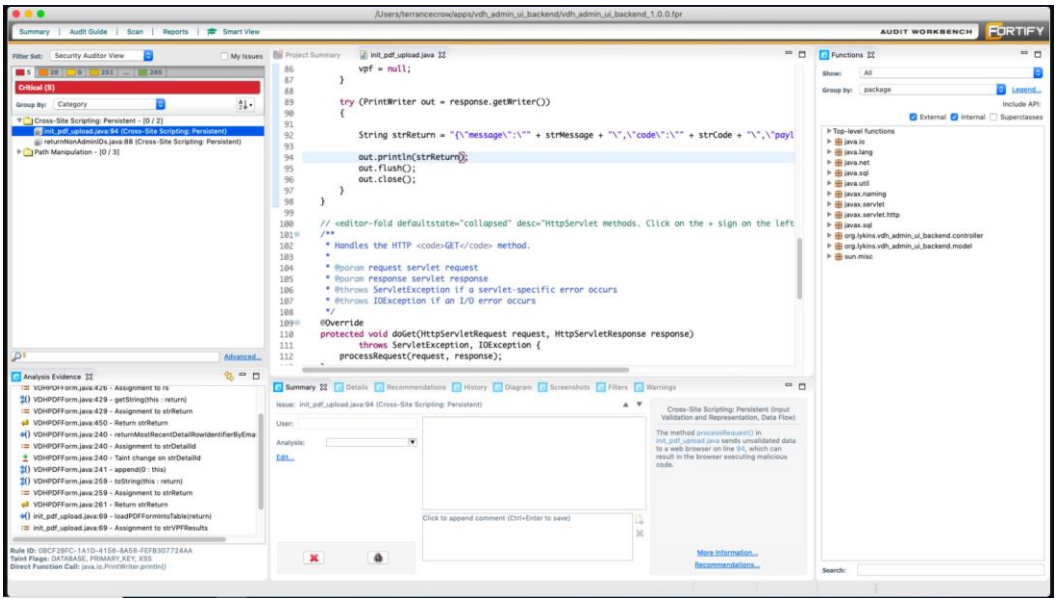

Рис.1.1. Fortify Static Code Analyzer

Checkmarx: Рішення для SAST, що інтегрується з різними IDE та платформами CI/CD.

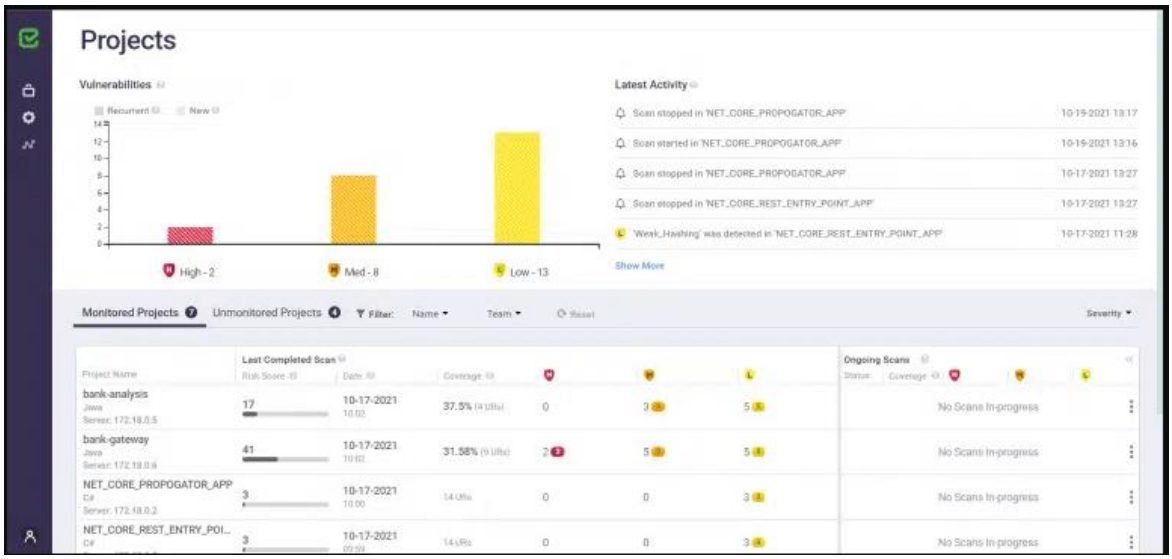

Рис.1.2. Checkmarx

Переваги:

- Раннє виявлення вразливостей на етапі розробки.
- Підвищення якості коду та дотримання стандартів.
- Інтеграція з конвеєрами CI/CD для автоматизації процесів.

Інструменти динамічного аналізу безпеки (DAST)

DAST-інструменти тестують працюючі додатки, імітуючи атаки з боку зловмисників. Вони допомагають виявити вразливості, які можуть бути пропущені під час статичного аналізу.

Основні інструменти:

OWASP ZAP (Zed Attack Proxy): Безкоштовний інструмент з відкритим кодом для автоматизованого DAST-тестування веб-додатків [7].

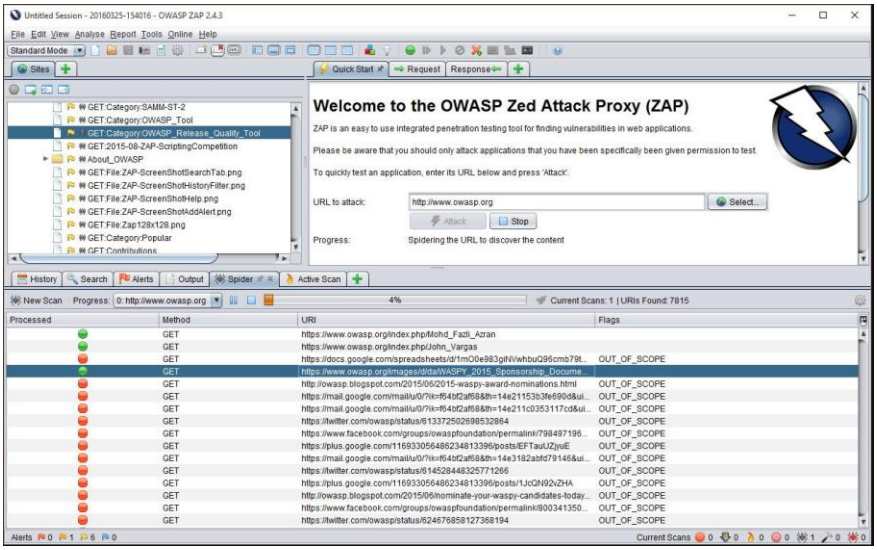

Рис.1.3. OWASP ZAP **(**Zed Attack Proxy**)**

Burp Suite: Потужний комерційний інструмент для тестування безпеки веб-додатків з можливістю автоматизації та розширення функціональності.

Acunetix: Автоматизований сканер безпеки веб-додатків з підтримкою різних технологій.

Переваги:

- Виявлення вразливостей у реальних умовах виконання додатку.
- Автоматизація тестування та інтеграція з CI/CD конвеєрами.
- Підтримка різних типів атак та вразливостей.

Платформи безперервної інтеграції та розгортання (CI/CD)

CI/CD платформи забезпечують автоматизацію процесів побудови, тестування та розгортання програмного забезпечення з інтеграцією безпеки [1].

Основні інструменти:

Jenkins: Популярний інструмент з відкритим кодом для автоматизації процесів CI/CD з великою кількістю плагінів.

GitLab CI/CD: Вбудована в GitLab система для безперервної інтеграції та розгортання з підтримкою функцій безпеки.

Azure DevOps: Хмарна платформа від Microsoft з інтегрованими інструментами для DevOps та DevSecOps.

Переваги:

- Автоматизація процесів розробки та розгортання.
- Інтеграція з інструментами безпеки для виконання перевірок на кожному етапі.
- Підтримка різних мов програмування та платформ.

1.2.Теоретичні основи DevSecOps

DevSecOps є еволюційним розвитком методології DevOps, який інтегрує безпеку в усі етапи життєвого циклу розробки програмного забезпечення. Розуміння теоретичних основ DevSecOps є критично важливим для успішного впровадження цієї методології в організаціях, що прагнуть підвищити швидкість та якість розробки, одночасно забезпечуючи високий рівень безпеки [1].

Витоки та еволюція DevSecOps

Поява DevSecOps пов'язана з розвитком Agile та DevOps методологій, які націлені на прискорення процесу розробки та доставки програмного забезпечення. DevOps, що поєднує команди розробки (Development) та експлуатації (Operations), сприяв підвищенню ефективності та швидкості випуску продуктів. Однак, зростання кіберзагроз та посилення вимог до безпеки виявили необхідність інтеграції безпеки (Security) безпосередньо в ці процеси.

Традиційно безпека розглядалася як окремий етап, що відбувався після розробки та перед розгортанням. Такий підхід призводив до затримок та підвищення вартості виправлення вразливостей, виявлених на пізніх стадіях. DevSecOps зміщує фокус на інтеграцію безпеки з самого початку розробки, що дозволяє виявляти та виправляти проблеми на ранніх етапах, знижуючи ризики та витрати [2].

Принципи DevSecOps

Основні принципи DevSecOps ґрунтуються на інтеграції безпеки в усі аспекти розробки та експлуатації, сприяючи створенню безпечного та якісного програмного забезпечення. Нижче наведено ключові принципи DevSecOps:

Безпека як невід'ємна частина культури організації

DevSecOps передбачає, що безпека є спільною відповідальністю всіх членів команди. Це означає, що розробники, операційні інженери та спеціалісти з безпеки працюють разом для досягнення спільної мети. Формування культури, де безпека інтегрована в повсякденну діяльність, сприяє підвищенню обізнаності та відповідальності кожного співробітника [2].

Інтеграція безпеки на ранніх етапах (Shift Left Security)

Перенесення заходів безпеки на ранні етапи життєвого циклу розробки дозволяє виявляти вразливості ще до того, як вони стануть критичними. Це досягається через моделювання загроз, визначення вимог безпеки та проведення код-рев'ю з акцентом на безпеку. Такий підхід знижує вартість виправлення помилок та підвищує загальний рівень захищеності продукту.

Автоматизація процесів безпеки

Автоматизація є ключовим елементом DevSecOps. Вона дозволяє інтегрувати перевірки безпеки безпосередньо в конвеєри безперервної інтеграції та розгортання (CI/CD). Автоматизовані інструменти для статичного та динамічного аналізу коду, а також для аналізу складу програмного забезпечення, забезпечують швидке виявлення вразливостей без уповільнення процесу розробки.

Інфраструктура як код (IaC) з вбудованою безпекою

Використання підходу IaC дозволяє автоматизувати розгортання та управління інфраструктурою за допомогою коду. Це спрощує інтеграцію безпеки в процеси управління інфраструктурою.

Управління версіями та секретами

Безпечне управління конфігураціями та конфіденційними даними є критично важливим. DevSecOps передбачає використання інструментів для безпечного зберігання та передачі секретів, а також контроль версій.

Вплив принципів DevSecOps на процес розробки:

- **Прискорення випуску продуктів:** Автоматизація та інтеграція безпеки дозволяють швидше доставляти оновлення без компромісів у захищеності.
- **Покращення якості та надійності:** Раннє виявлення та виправлення вразливостей підвищують загальну якість продукту.
- **Зниження ризиків та витрат:** Усунення проблем на ранніх стадіях є менш витратним і знижує ризик успішних атак.
- **Підвищення довіри клієнтів та відповідності нормативам:** Безпечне програмне забезпечення відповідає вимогам регуляторів та підвищує репутацію компанії.

1.3. Аналіз інструментів для інтеграції безпеки в CI/CD

Інтеграція безпеки в конвеєри безперервної інтеграції та розгортання (CI/CD) є ключовим елементом методології DevSecOps. Це дозволяє виявляти та усувати вразливості на ранніх стадіях розробки, забезпечуючи швидке та безпечне постачання програмного забезпечення. У цьому розділі розглянемо загальні підходи та інструменти для інтеграції безпеки в процеси CI/CD, проаналізуємо їхні можливості та обговоримо практичні аспекти їх застосування.

Статичний та динамічний аналіз коду (SAST і DAST) в CI/CD

Статичний аналіз коду (SAST) здійснюється без виконання програми і спрямований на виявлення вразливостей у вихідному коді на ранніх стадіях розробки. Інтеграція SAST в CI/CD дозволяє автоматично аналізувати код при кожній побудові або коміті, допомагаючи розробникам оперативно виявляти та виправляти потенційні проблеми безпеки. Використання інструментів, таких як **SonarQube** або **Fortify Static Code Analyzer**, дає змогу налаштовувати правила перевірки відповідно до вимог проекту та встановлювати Quality Gates для контролю метрик якості та безпеки [2].

Динамічний аналіз безпеки (DAST) проводиться під час виконання програми і спрямований на виявлення вразливостей, які проявляються лише в робочому середовищі. Інтеграція DAST у CI/CD передбачає автоматичне сканування додатків після їх розгортання в тестовому середовищі. Інструменти, такі як **OWASP ZAP** або **Burp Suite**, можуть бути налаштовані для автоматизованого сканування, виявляючи проблеми, такі як SQL-ін'єкції або міжсайтовий скриптинг (XSS). Результати сканування інтегруються з системами управління завданнями, що дозволяє команді швидко реагувати на виявлені вразливості.

Поєднання SAST та DAST у процесах CI/CD забезпечує комплексний підхід до безпеки, охоплюючи як статичні, так і динамічні аспекти програмного

забезпечення. Раннє виявлення проблем завдяки SAST знижує вартість їх виправлення, оскільки усунення вразливостей на етапі розробки є менш витратним, ніж після розгортання. DAST, у свою чергу, допомагає виявити вразливості, пов'язані з конфігурацією середовища або взаємодією компонентів, які неможливо виявити статичним аналізом.

Ключові аспекти інтеграції SAST та DAST:

- **Автоматизація процесів:** Забезпечує безперервний контроль безпеки без додаткового навантаження на команду.
- **Налаштування оповіщень:** Дозволяє розробникам оперативно отримувати інформацію про виявлені вразливості.
- **Встановлення порогових значень:** *Quality Gates* допомагають приймати автоматичні рішення щодо продовження або зупинки побудови залежно від результатів аналізу.

Виклики при впровадженні SAST та DAST:

- **Збільшення часу побудови:** Додаткові перевірки можуть уповільнювати процес CI/CD. Оптимізація конвеєрів та використання паралельного виконання завдань можуть мінімізувати цей ефект.
- **Навчання персоналу:** Розробники повинні розуміти роботу цих інструментів і правильно інтерпретувати їхні результати.

Переваги інтеграції SAST та DAST:

- **Підвищення якості коду:** SAST допомагає виявляти та виправляти помилки на ранніх стадіях, зменшуючи кількість вразливостей у продакшн.
- **Виявлення реальних загроз:** DAST виявляє вразливості, що можуть бути експлуатовані в реальних умовах, включаючи проблеми з конфігурацією.
- **Зниження ризиків та витрат:** Раннє усунення вразливостей зменшує потенційні витрати на виправлення та наслідки інцидентів безпеки.

Інструменти статичного аналізу коду (SAST)

SonarQube є одним з найпопулярніших інструментів для статичного аналізу коду. Це платформа з відкритим кодом, яка підтримує понад 25 мов програмування, включаючи Java, C#, JavaScript, Python та інші. SonarQube дозволяє виявляти баги, вразливості та проблеми з покриттям коду тестами. Інтеграція SonarQube з CI/CD конвеєрами, такими як Jenkins або GitLab CI/CD, дозволяє автоматично запускати аналіз при кожній побудові або коміті. Встановлення *Quality Gates* допомагає контролювати ключові метрики якості та приймати рішення щодо продовження або зупинки процесу побудови на основі результатів аналізу [7].

Fortify Static Code Analyzer від Micro Focus є комерційним рішенням для глибокого аналізу коду на наявність вразливостей. Він підтримує широкий спектр мов програмування та може інтегруватися з різними інструментами розробки та CI/CD платформами. Fortify надає детальні звіти про виявлені вразливості та рекомендації щодо їхнього виправлення. Це рішення особливо підходить для великих організацій, які потребують високого рівня захисту та відповідності нормативним вимогам.

Checkmarx є ще одним потужним інструментом SAST, який пропонує комплексний підхід до безпеки коду. Він забезпечує глибокий аналіз коду з можливістю налаштування правил та інтеграцію з популярними IDE, такими як Visual Studio та Eclipse. Checkmarx може бути інтегрований у CI/CD конвеєри для автоматичного аналізу та надання зворотного зв'язку розробникам у режимі реального часу [11].

Veracode Static Analysis є хмарним рішенням для статичного аналізу коду, яке не вимагає встановлення програмного забезпечення на локальних машинах. Це забезпечує швидкий старт та масштабованість. Veracode підтримує інтеграцію з різними CI/CD інструментами та надає детальні звіти про вразливості з пріоритетами та рекомендаціями щодо їх усунення.

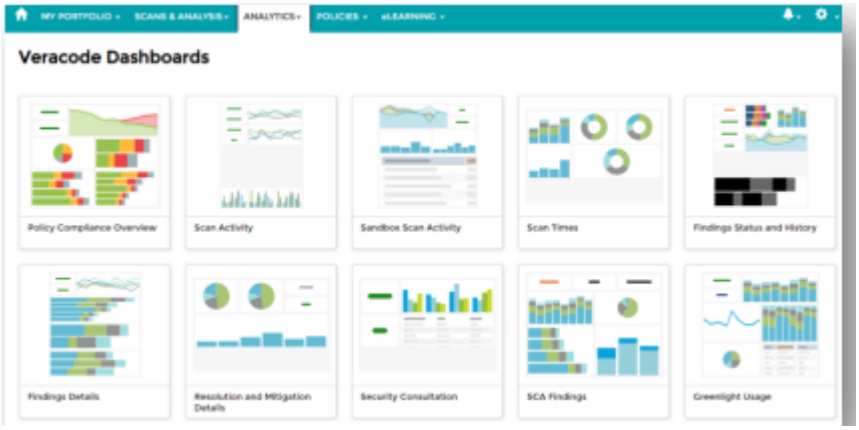

Рис.1.4. Veracode Static Analysis

При виборі SAST-інструменту важливо враховувати такі фактори:

- **Підтримувані мови програмування**: Переконатися, що інструмент підтримує всі мови, які використовуються в проектах.
- **Можливості інтеграції**: Оцінити, наскільки легко інструмент інтегрується з існуючими CI/CD конвеєрами та інструментами розробки та чи взагалі на таке здатен.
- **Якість виявлення вразливостей**: Перевірити, наскільки ефективно інструмент виявляє реальні вразливості та мінімізує хибнопозитивні повідомлення.
- **Масштабованість та продуктивність**: Оцінити, чи може інструмент справлятися з великими кодовими базами без значного впливу на час побудови.

Практичні аспекти використання SAST-інструментів у CI/CD включають налаштування автоматичного запуску аналізу при кожній побудові або коміті. Це забезпечує постійний контроль за станом безпеки коду та дозволяє розробникам швидко реагувати на виявлені проблеми. Важливо також налаштувати систему оповіщень, щоб відповідальні особи отримували повідомлення про критичні вразливості. Використання *Quality Gates* допомагає автоматизувати прийняття рішень щодо продовження процесу побудови, зупиняючи його у разі виявлення серйозних проблем.

Використання SAST-інструментів сприяє підвищенню загальної якості коду, оскільки вони допомагають виявляти не лише вразливості, але й проблеми з кодовою базою, такі як дублювання коду та недоліки архітектури. Це сприяє підвищенню продуктивності команди розробників та покращенню програмного забезпечення.

Важливим аспектом є навчання команди щодо використання SAST-інструментів та інтерпретації їхніх результатів. Розробники повинні розуміти, як виправляти виявлені вразливості та уникати їх у майбутньому. Проведення тренінгів та семінарів з безпечного програмування допоможе підвищити обізнаність та компетентність команди в питаннях безпеки.

Одним із викликів при використанні SAST-інструментів є можливість генерування великої кількості хибнопозитивних спрацювань. Це може відволікати команду від реальних проблем та знижувати ефективність. Для мінімізації цього ефекту рекомендується налаштовувати правила аналізу відповідно до специфіки проекту та періодично переглядати їх. Деякі інструменти мають можливість навчання на основі попередніх результатів, що допомагає покращити точність виявлення вразливостей.

Інструменти динамічного аналізу коду (DAST)

Динамічний аналіз безпеки (DAST. На відміну від статичного аналізу коду (SAST), який досліджує вихідний код без його виконання, DAST-інструменти тестують додаток під час його реального виконання. Це дозволяє виявляти вразливості, які проявляються тільки в робочому середовищі, зокрема ті, що пов'язані з конфігурацією, взаємодією компонентів та поведінкою додатку під час обробки реальних запитів.

OWASP ZAP (Zed Attack Proxy) є одним з найпопулярніших DAST-інструментів з відкритим кодом. Розроблений проєктом OWASP, ZAP призначений для автоматизованого та ручного сканування веб-додатків на наявність вразливостей. Інструмент може бути інтегрований у процеси CI/CD, що дозволяє

автоматично запускати сканування після розгортання додатку в тестовому середовищі. OWASP ZAP підтримує функції проксі-сервера, що дає можливість перехоплювати та аналізувати трафік між клієнтом та сервером, імітуючи різні атаки, такі як SQL-ін'єкції, міжсайтовий скриптинг (XSS) та інші.

Burp Suite від компанії PortSwigger є потужним комерційним інструментом для тестування безпеки веб-додатків. Він пропонує широкий спектр функціональності, включаючи автоматизоване сканування, аналіз трафіку, модифікацію запитів та відповідей, а також різні модулі для специфічних типів атак. Burp Suite може бути інтегрований у CI/CD конвеєри, що дозволяє автоматизувати процес сканування та отримувати звіти про виявлені вразливості безпосередньо в процесі розробки.

Acunetix є іншим популярним комерційним DAST-інструментом, який спеціалізується на скануванні веб-додатків та API. Він підтримує широкий спектр технологій та платформ, включаючи JavaScript, HTML та односторінкові додатки (SPA). Acunetix надає детальні звіти з рекомендаціями щодо виправлення виявлених вразливостей, а також може інтегруватися з системами управління завданнями для автоматичного створення тикетів.

При інтеграції DAST-інструментів у процеси CI/CD важливо враховувати наступні аспекти:

- **Автоматизація запуску сканування**. DAST-інструменти можуть бути налаштовані для автоматичного запуску сканування після успішної побудови та розгортання додатку в тестовому середовищі. Це забезпечує своєчасне виявлення вразливостей перед тим, як додаток потрапить у продакшн.
- **Налаштування параметрів сканування**. Щоб зменшити час сканування та уникнути перевантаження тестового середовища, можна налаштувати область сканування, визначити пріоритетні зони додатку або використовувати інкрементальні сканування.

- **Інтеграція результатів сканування**. Результати DAST-сканування повинні бути доступними для команди розробників та спеціалістів з безпеки. Це може бути реалізовано через автоматичне створення звітів, розсилку сповіщень або інтеграцію з системами управління завданнями, такими як Jira чи Azure Boards.

Одним із викликів при використанні DAST-інструментів є тривалість сканування. Оскільки DAST-сканування проводиться в реальному часі та охоплює широкий спектр тестів, воно може займати значний час, що впливає на швидкість конвеєра CI/CD. Для вирішення цієї проблеми можна:

- Виконувати сканування паралельно з іншими завданнями конвеєра.
- Планувати повні сканування поза робочим часом або використовувати більш швидкі, цільові сканування для критичних компонентів.
- Оптимізувати налаштування сканування, виключаючи малозначущі або недоступні для сканування області додатку.

Хибнопозитивні спрацювання є ще однією проблемою, з якою можуть стикатися користувачі DAST-інструментів. Автоматизовані сканери можуть генерувати помилкові спрацьовування, що потребує додаткового аналізу з боку спеціалістів з безпеки. Для мінімізації рекомендується:

- Регулярно оновлювати інструмент та його бази знань.
- Налаштовувати правила сканування відповідно до специфіки додатку.
- Проводити ручну перевірку критичних вразливостей перед вжиттям заходів.
- Безпека тестового середовища є важливим аспектом при використанні DAST. Оскільки сканери можуть виконувати потенційно шкідливі запити, важливо забезпечити, щоб тестове середовище було ізольованим від середовища розробки та не містило реальних даних користувачів [1].

Практичні рекомендації щодо використання DAST-інструментів у CI/CD:

- **Аутентифікація**. Налаштуйте інструмент для проходження аутентифікації, щоб сканер мав доступ до захищених областей додатку. Це забезпечить більш повне охоплення тестуванням.
- **Попереднє налаштування сесій**. Використовуйте файли cookie або токени автентифікації, щоб уникнути проблем з обмеженнями доступу або блокуванням сканера.
- **Використання API-сканування**. Якщо додаток має API, використовуйте функції сканування API, надаючи інструменту специфікації, такі як OpenAPI або Swagger.
- **Навчання команди**. Розробники та спеціалісти з безпеки повинні розуміти результати сканування та знати, як виправляти виявлені вразливості. Проведення спільних аналізів результатів та тренінгів допоможе підвищити ефективність.

Переваги використання DAST-інструментів включають виявлення вразливостей, які неможливо знайти статичним аналізом коду. Це, зокрема, проблеми, пов'язані з:

- Некоректною обробкою даних користувача на рівні сервера.
- Неправильним управлінням сесіями та автентифікацією.
- Помилками конфігурації серверів та сервісів.
- Вразливостями, що виникають через взаємодію з іншими компонентами або сервісами.

Інструменти інтерактивного аналізу безпеки (IAST)

Інтерактивний аналіз безпеки додатків (IAST) поєднує в собі переваги статичного (SAST) та динамічного (DAST) аналізу, забезпечуючи більш глибоке та точне виявлення вразливостей у реальному часі. IAST-інструменти працюють всередині додатку під час його виконання, аналізуючи код та поведінку програми, що дозволяє виявляти вразливості з високою точністю та мінімізувати кількість хибнопозитивних спрацювань [2].

Як працює IAST

IAST-агенти інтегруються безпосередньо в середовище виконання додатку. Під час виконання програми, зокрема під час автоматизованих тестів або реального використання, агент відстежує потоки даних, виклики методів та взаємодії з іншими компонентами. Це дозволяє виявляти вразливості, які можуть бути пропущені статичним або динамічним аналізом.

На відміну від DAST, який імітує атаки ззовні, IAST має доступ до внутрішніх даних та логіки додатку. Це дозволяє йому більш точно визначати, чи є певна поведінка вразливістю, чи це нормальна функціональність.

Переваги використання IAST

- **Висока точність виявлення**: Завдяки доступу до внутрішніх даних та контексту, IAST-інструменти можуть більш точно ідентифікувати вразливості, зменшуючи кількість помилок.
- **Швидка інтеграція**: IAST легко інтегрується в існуючі процеси розробки та тестування, не вимагаючи значних змін у конвеєрах CI/CD.
- **Автоматизація**: Інструменти IAST можуть працювати в режимі реального часу під час виконання автоматизованих тестів, забезпечуючи безперервний моніторинг безпеки.
- **Детальні звіти**: Надають розробникам конкретну інформацію про місце в коді та умови, за яких виникає вразливість, що спрощує процес виправлення.

Популярні IAST-інструменти

- **Contrast Security**: Забезпечує інтерактивний аналіз безпеки для різних мов програмування та фреймворків, таких як Java, .NET, Node.js та Python.
- **Seeker від Synopsys**: Підтримує аналіз додатків на різних платформах, надаючи детальні звіти та рекомендації.
- **HCL AppScan (раніше IBM Security AppScan)**: Пропонує IAST-рішення, які поєднують статичний та динамічний аналіз з інтерактивним підходом.

Порівняння інструментів статичного і динамічного аналізу безпеки коду

1. Загальний огляд SAST та DAST

Статичний аналіз безпеки коду (SAST):

- **Методологія:** Аналізує вихідний код без її виконання.
- **Мета:** Виявлення вразливостей на ранніх стадіях розробки, ще до виконання додатка.
- **Підхід:** Білий ящик (white-box testing), оскільки має доступ до внутрішньої структури коду.
- **Приклади інструментів:** SonarQube, Checkmarx, Fortify Static Code Analyzer, Veracode Static Analysis.

Динамічний аналіз безпеки коду (DAST):

- **Методологія:** Тестує працюючий додаток шляхом відправки шкідливих запитів та аналізу відповідей.
- **Мета:** Виявлення вразливостей, які проявляються під час виконання додатка.
- **Підхід:** Чорний ящик (black-box testing), оскільки не має доступу до вихідного коду.
- **Приклади інструментів:** OWASP ZAP, Burp Suite, Acunetix, Netsparker.

2. Порівняння за ключовими параметрами

Таблиця 1.1.

Порівняння SAST та DAST за ключовими параметрами

Параметр	SAST	DAST
Етап застосування	На ранніх стадіях розробки, під час написання коду	Після розгортання додатка в тестовому або продуктивному середовищі
Тип аналізу	Статичний аналіз вихідного коду	Динамічне тестування працюючого додатка
Підхід до тестування	Білий ящик	Чорний ящик
Доступ до коду	Необхідний доступ до вихідного або байт-коду	Не потребує доступу до коду
Виявлення вразливостей	Виявляє потенційні вразливості на рівні коду (наприклад, SQL-ін'єкції, XSS, неправильне управління пам'яттю)	Виявляє вразливості, які проявляються під час виконання (наприклад, некоректна автентифікація, сесійні проблеми, конфігураційні помилки)
Хибні спрацьовування	Може генерувати велику кількість хибних позитивних спрацьовувань	Зазвичай менше хибних позитивних спрацьовувань, але може пропустити деякі вразливості
Складність впровадження	Вимагає інтеграції з процесами розробки та навчання команди	Може бути запущений незалежно від процесів розробки
Час виявлення проблем	Виявлення під час розробки	Виявлення після розгортання додатка

3. Переваги та недоліки

SAST:

- **Переваги:**
  - Раннє виявлення вразливостей, що знижує вартість їх виправлення.

- Глибокий аналіз коду, включаючи логіку програми та структуру даних.
- Можливість інтеграції в IDE та CI/CD конвеєри для безперервного аналізу.

- **Недоліки:**

- Може генерувати велику кількість хибних позитивних спрацьовувань, що потребує ручної перевірки.
- Не виявляє вразливостей, які проявляються тільки під час виконання додатка.
- Потребує доступу до вихідного коду, що може бути проблемою для сторонніх компонентів.

DAST:

- **Переваги:**

- Виявляє вразливості, які проявляються тільки під час виконання додатка.
- Не потребує доступу до вихідного коду, що дозволяє тестувати сторонні додатки або компоненти.
- Імітує дії зловмисника, що дозволяє виявляти реальні загрози.
- Менше хибних позитивних спрацьовувань порівняно з SAST.

- **Недоліки:**

- Виявляє вразливості пізніше у життєвому циклі розробки.
- Не може виявити вразливості, приховані в недоступних для тестування ділянках коду.
- Обмежений у виявленні логічних помилок та внутрішніх проблем коду.

4. Приклади виявлених вразливостей

SAST:

- Неправильне використання функцій або бібліотек.
- Небезпечні конструкції коду (наприклад, використання eval() з неперевіреними даними).
- Витік конфіденційної інформації через коментарі або повідомлення про помилки.

- Потенційні переповнення буфера або неправильне управління пам'яттю.

DAST:

- Неправильна обробка вхідних даних, що призводить до SQL-ін'єкцій або XSS.
- Проблеми з автентифікацією та авторизацією.
- Некоректне управління сесіями (наприклад, вразливість до атак повторного використання сесій).
- Витік інформації через неправильні HTTP-заголовки або повідомлення про помилки.

5. Сумісність SAST та DAST

SAST та DAST не є взаємовиключними, а навпаки, доповнюють один одного. Поєднання обох підходів дозволяє забезпечити більш повне охоплення вразливостей та підвищити загальний рівень безпеки додатка.

- **SAST** забезпечує аналіз коду на ранніх стадіях, що допомагає розробникам запобігати впровадженню вразливостей.
- **DAST** дозволяє виявляти вразливості, які проявляються лише під час виконання, а також перевіряти безпеку розгорнутих додатків у реальних умовах.

6. Вибір інструментів SAST та DAST

Критерії вибору:

- **Підтримувані мови та технології:** Переконатись, що інструмент підтримує мови програмування та фреймворки, які використовуються.
- **Можливості інтеграції:** Інструменти повинні легко інтегруватися з CI/CD конвеєрами та іншими інструментами.
- **Якість та точність аналізу:** Оцінити рівень хибних позитивних та спрацьовувань.
- **Вартість:** Розглянути бюджет та можливості безкоштовних або комерційних рішень.
- **Підтримка:** Наявність активної підтримки та регулярних оновлень.

Вибір інструментів для DevSecOps: SonarQube, GitLab, Jenkins

SonarQube є потужним інструментом для статичного аналізу коду (SAST). Він дозволяє автоматично перевіряти вихідний код на наявність вразливостей, помилок та порушень стандартів кодування. Підтримуючи понад 25 мов програмування, SonarQube забезпечує глибокий аналіз коду та надає розробникам зворотний зв'язок у режимі реального часу. Інтеграція в процеси CI/CD дозволяє автоматизувати перевірки безпеки та якості коду при кожній побудові або коміті.

GitLab — це комплексна платформа для управління життєвим циклом розробки програмного забезпечення. Вона поєднує в собі систему контролю версій, засоби для управління завданнями, а також вбудовані можливості для безперервної інтеграції та розгортання (CI/CD). GitLab надає інструменти для інтеграції безпеки, включаючи SAST, DAST та аналіз складу програмного забезпечення (SCA), що дозволяє виявляти вразливості на різних етапах розробки. Крім того, він забезпечує централізоване управління доступом та контролем версій, що сприяє підвищенню безпеки та прозорості процесів.

Jenkins є популярним інструментом для автоматизації процесів CI/CD з відкритим кодом. Завдяки своїй розширюваності та великій кількості плагінів, Jenkins може бути налаштований для виконання різноманітних завдань, включаючи інтеграцію інструментів безпеки, таких як SonarQube та різні DAST-сканери. Jenkins дозволяє створювати складні конвеєри побудови, тестування та розгортання, забезпечуючи автоматизацію процесів та інтеграцію безпеки в кожному етапі.

Вибір цих інструментів обумовлений наступними факторами:

- **Сумісність та інтеграція.** SonarQube, GitLab та Jenkins легко інтегруються між собою. Наприклад, можна налаштувати Jenkins для автоматичного запуску SonarQube-аналізу при кожній побудові, а результати аналізу відображати безпосередньо в GitLab. Це забезпечує безперервний контроль якості та безпеки коду в єдиному робочому просторі.

- **Гнучкість та масштабованість.** Jenkins та GitLab підтримують широкий спектр плагінів та інтеграцій з іншими інструментами, що дозволяє адаптувати їх під специфічні потреби проекту. SonarQube також надає можливість налаштовувати правила аналізу та додавати підтримку нових мов програмування.
- **Підтримка спільноти.** Всі три інструменти мають активну спільноту користувачів та розробників, що забезпечує постійне оновлення, швидке виправлення помилок та наявність великої кількості ресурсів для навчання та вирішення проблем.

Практичне застосування цих інструментів у DevSecOps може виглядати наступним чином:

1. **Розробка коду та контроль версій з GitLab.** Розробники пишуть код та здійснюють коміти в репозиторії GitLab. Використання GitLab забезпечує централізоване управління кодом та контроль доступу, що підвищує безпеку та впорядкованість процесів.

2. **Автоматизація побудови та тестування з Jenkins.** При кожному коміті або за розкладом Jenkins автоматично запускає процеси побудови та тестування. Завдяки розширюваності Jenkins, можна налаштувати конвеєри для виконання різноманітних завдань, включаючи статичний та динамічний аналіз безпеки.

3. **Статичний аналіз коду з SonarQube.** В рамках конвеєрів Jenkins інтегрується SonarQube для автоматичного аналізу коду. Результати аналізу передаються в GitLab, де розробники можуть переглядати звіти та отримувати зворотний зв'язок. Це сприяє швидкому виявленню та виправленню вразливостей та проблем з якістю коду.

4. **Безперервне розгортання та моніторинг.** Після успішного проходження всіх етапів тестування та аналізу безпеки, Jenkins може автоматично розгортати додаток у продакшн-середовище. Використання інструментів моніторингу та логування дозволяє відстежувати стан додатку та оперативно реагувати на інциденти.

Переваги використання цієї комбінації інструментів у DevSecOps:

- **Цілісність процесів.** Інтеграція SonarQube, GitLab та Jenkins забезпечує безперервний та автоматизований процес розробки, тестування, аналізу безпеки та розгортання.
- **Підвищення продуктивності.** Автоматизація рутинних завдань та швидкий зворотний зв'язок дозволяють розробникам зосередитися на написанні якісного коду.
- **Покращення безпеки.** Виявлення вразливостей на ранніх стадіях та інтеграція безпеки в кожен етап життєвого циклу розробки знижує ризик появи критичних проблем у продакшн-середовищі.
- **Масштабованість та адаптивність.** Ці інструменти можуть бути налаштовані під потреби проектів будь-якого розміру та складності, а також легко інтегруються з іншими системами та сервісами.

Висновок до розділу 1:

У першому розділі було здійснено комплексний огляд концепції DevSecOps, теоретичних засад та ключових методів, що лежать в основі інтеграції безпеки у процеси розробки та експлуатації програмного забезпечення. Було визначено основні принципи DevSecOps, зокрема «Shift Left Security», , автоматизацію перевірок та постійне вдосконалення підходів до захисту. Розглянуто методи статичного (SAST), динамічного (DAST) та інтерактивного (IAST) аналізу безпеки коду, а також наголошено на важливості управління конфігураціями, секретами та відповідністю (Compliance as Code).

Окрему увагу приділено огляду інструментів, які можуть бути впроваджені в конвеєри CI/CD для забезпечення безпеки, таких як SonarQube, GitLab та Jenkins. Зокрема, було показано, як ці інструменти, разом з іншими засобами SAST, DAST та IAST, сприяють ранньому виявленню вразливостей, підвищенню якості та надійності програмного забезпечення.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ІНТЕГРАЦІЇ DEVSECOPS

У цьому розділі буде розглянуто процес встановлення, налаштування та інтеграції інструментів SonarQube, GitLab та Jenkins у контексті DevSecOps. Крок за кроком процес встановлення кожного інструменту, їх налаштування, а також інтеграцію між ними для створення ефективного та безпечного конвеєра CI/CD.

2.1 SonarQube як інструмент статичного аналізу коду

Встановлення SonarQube Community edition

Системні вимоги:

- Операційна система: Linux, Windows, macOS
- Java: Oracle JRE 11 або OpenJDK 11,17

Завантаження SonarQube:

Перейти на офіційний сайт SonarQube та завантажити останню версію для операційної системи. Завантажити та розпакувати архів, після цього запустити файл StartSonar.bat, щоб встановити Sonarqube.Крім цього потрібно встановити Sonar-scanner та Java 11 версії або вище, після цього потрібно перейти до налаштувань перемінних та внести зміни в системну перемінну ”Path“, після чого зберегти зміни [7].

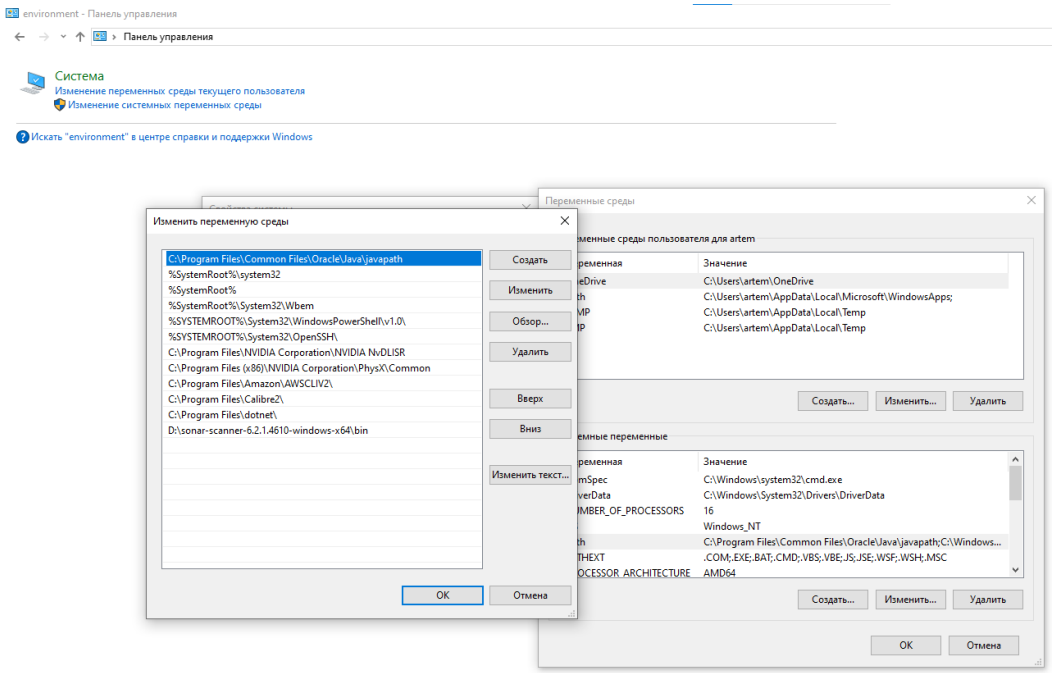

Рис.2.1. Встановлення перемінної Javapath

У випадку якщо необхідно аби сонар був доступний з інших локальних машин в мережі, потрібно змінити в файлі sonar.properties параметр “sonar.web.host”, якщо потрібно то можна змінити порт змінивши параметр “web.port”, за замовчуванням використовується 9000.

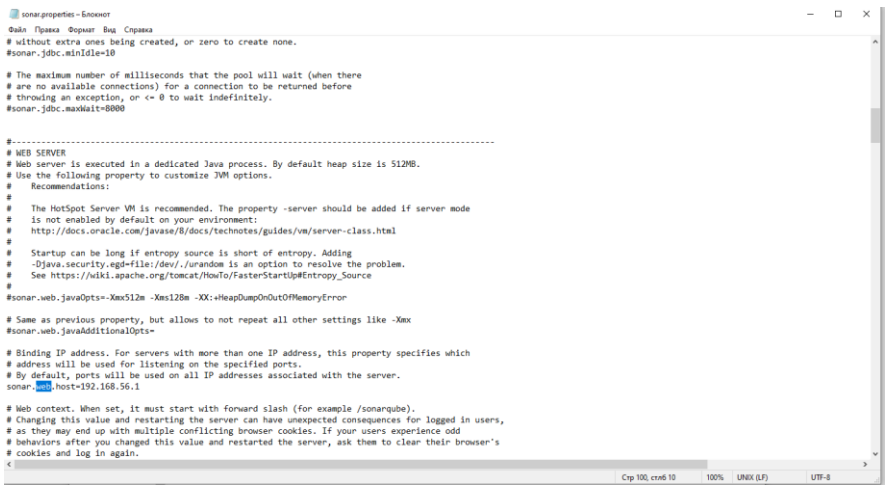

Рис.2.2. Налаштування файлу sonar.properties щоб зробити sonar доступним через локальну мережу

Після встановлення необхідно перейти на веб сторінку по http://ip-address:9000, та ввести admin admin, сонар запропонує змінити пароль, після чого базове налаштування Sonarqube завершене.

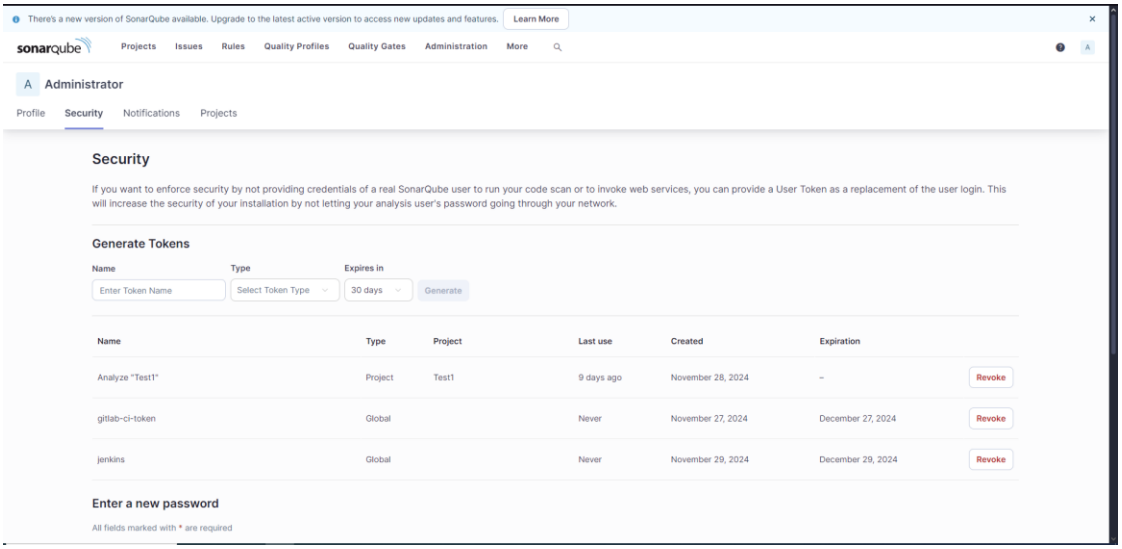

Рис.2.3. Налаштування Sonarqube завершено

2.2 Інтеграція GitLab у DevSecOps процеси

Системні вимоги:

- Операційна система: Linux (рекомендовано Ubuntu або CentOS), в цій роботі використовувалась Ubuntu, на сайті розробника вказано які дистрибутиви підходять. Детальніше на сайті https://about.gitlab.com/install/.
- Пам'ять: Мінімум 4 ГБ RAM
- Процесор: 2 ядра або більше

```
sudo apt-get update
sudo apt-get install -y curl openssh-server ca-certificates tzdata perl
sudo apt-get install -y postfix
```

Додати репозиторій пакетів GitLab і встановити пакет

curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo bash

Далі треба налаштувати зовнішню адресу завдяки якій можна буде працювати з Sonarqube та Jenkins.

sudo EXTERNAL_URL="https://gitlab.example.com" apt-get install gitlab-ee

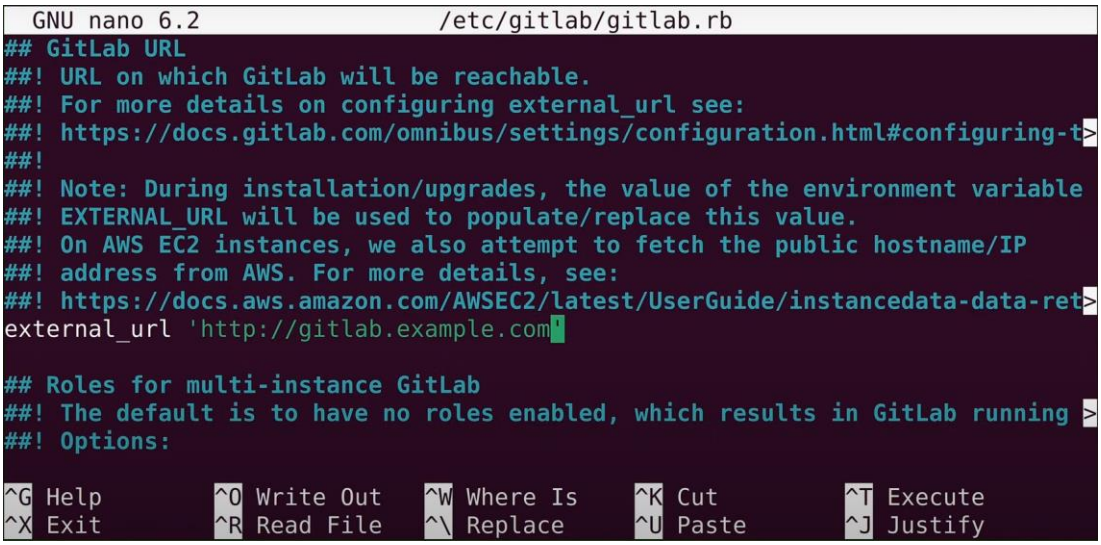

Рис.2.3. Налаштування зовнішньої IP адреси Gitlab

Також це можна зробити пізніше через налаштування файлу gitlab.rb

install gitlab-ee=16.2.3-ee.0

Якщо ви не вказали спеціальний пароль під час встановлення, пароль буде згенеровано випадковим чином і зберігатиметься протягом 24 годин у /etc/gitlab/initial_root_password. Використовуйте цей пароль з іменем користувача root для входу.

```
jay@gitlab:~$ sudo cat /etc/gitlab/initial_root_password
# WARNING: This value is valid only in the following conditions
#          1. If provided manually (either via `GITLAB_ROOT_PASSWORD` environm
ent variable or via `gitlab_rails['initial_root_password']` setting in `gitlab
.rb`, it was provided before database was seeded for the first time (usually,
the first reconfigure run).
#          2. Password hasn't been changed manually, either via UI or via comm
and line.
#
#          If the password shown here doesn't work, you must reset the admin p
assword following https://docs.gitlab.com/ee/security/reset_user_password.html
#reset-your-root-password.

Password: [redacted]

# NOTE: This file will be automatically deleted in the first reconfigure run a
fter 24 hours.
```

Рис.2.4. Отримання тимчасового паролю

Після цього необхідно зайти в gitlab та змінити пароль.

Налаштування GitLab CI/CD

Створення проекту:

- Увійти в GitLab як адміністратор.
- Створити нову групу та проект.
- НалаштуватиGitLab Runner:
- Встановити GitLab Runner: curl -LJO https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64

sudo mv gitlab-runner-linux-amd64 /usr/local/bin/gitlab-runner

sudo chmod +x /usr/local/bin/gitlab-runner

sudo useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash

sudo gitlab-runner install --user=gitlab-runner --working-directory=/home/gitlab-runner

sudo gitlab-runner start

Після цього можна створювати нові runner для проектів напряму через gitlab.

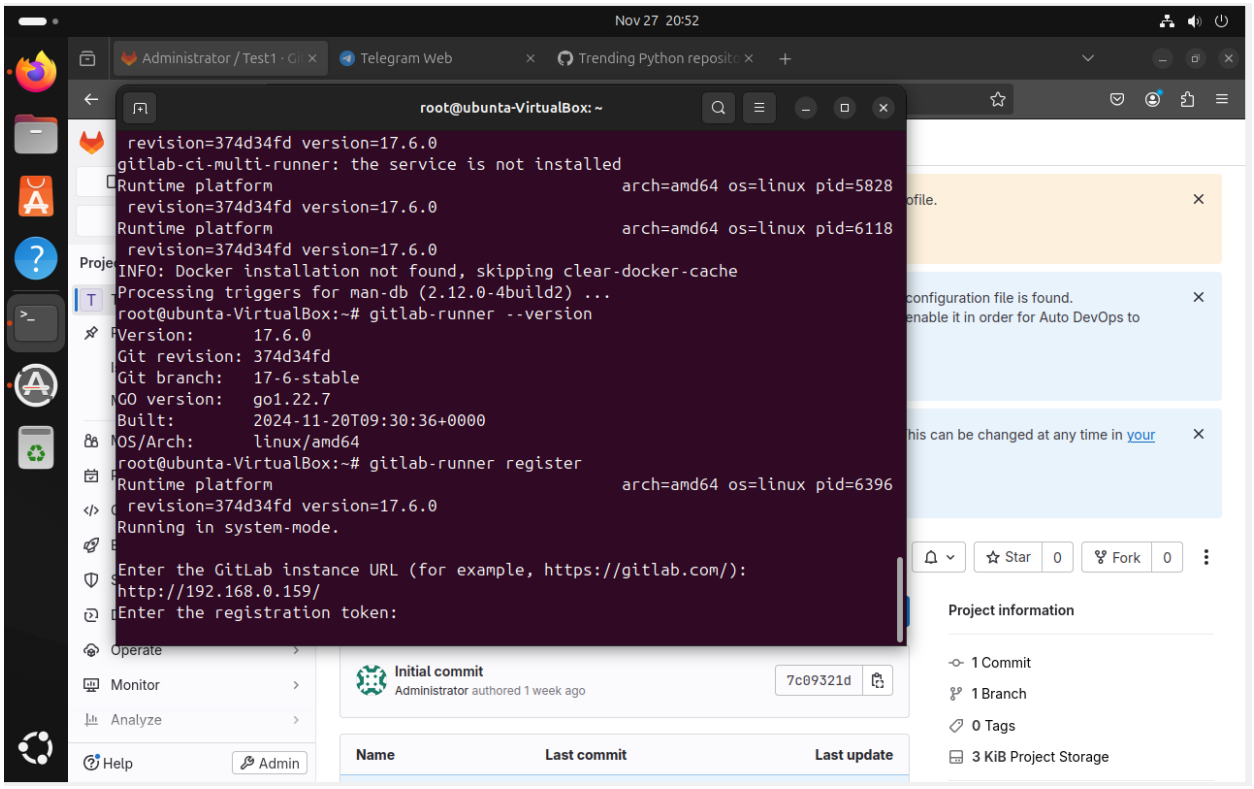

Рис.2.5. Реєстрація runner

Реєстрація Runner:

sudo gitlab-runner register

- Ввести URL вашого GitLab (http://gitlab.example.com).
- Обрати реєстраційний токен (його можна знайти в налаштуваннях проекту під "Settings" -> "CI/CD" -> "Runners").
- Обрати тип Runner (наприклад, shell).

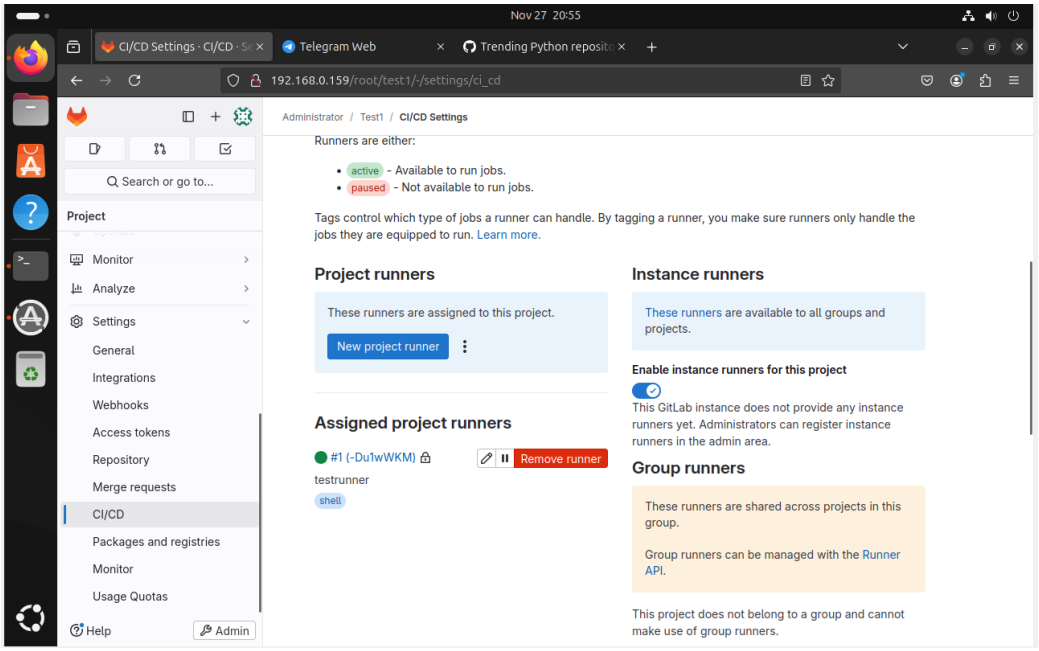

Рис.2.6. Створений runner

Також можна це зробити через CI/CD розділ проекту де після створення раннеру буде вказана послідовність команд після виконання яких буде сторено новий runner для виконання CI/CD.

Налаштування Gitlab завершено.

2.3 Інтеграція Jenkins у DevSecOps процеси

Для того щоб встановити Jenkins необхідно перейти на офіційний сайт розробника та обрати необхідну версію Jenkins на вашу ОС (в даному випадку Windows).

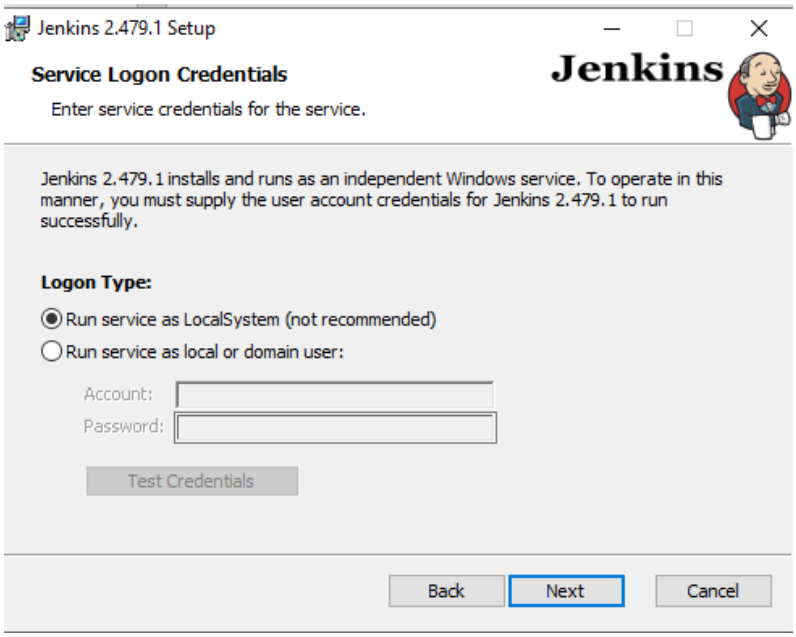

Рис.2.7. Встановлення jenkins

За замовчуванням Jenkins використовує порт 8080, але його можна замінити якщо він вже зайнятий.

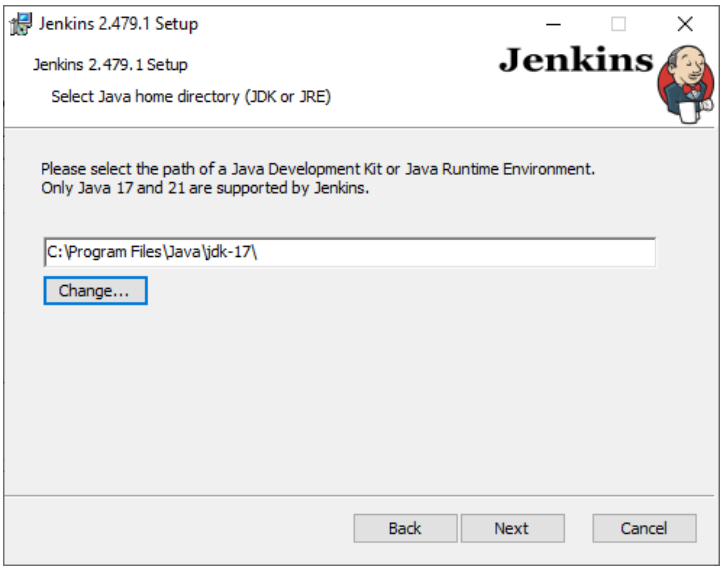

Рис.2.8. Встановлення jenkins

Після цього необхідно вказати шлях до JDK та обрати встановити.

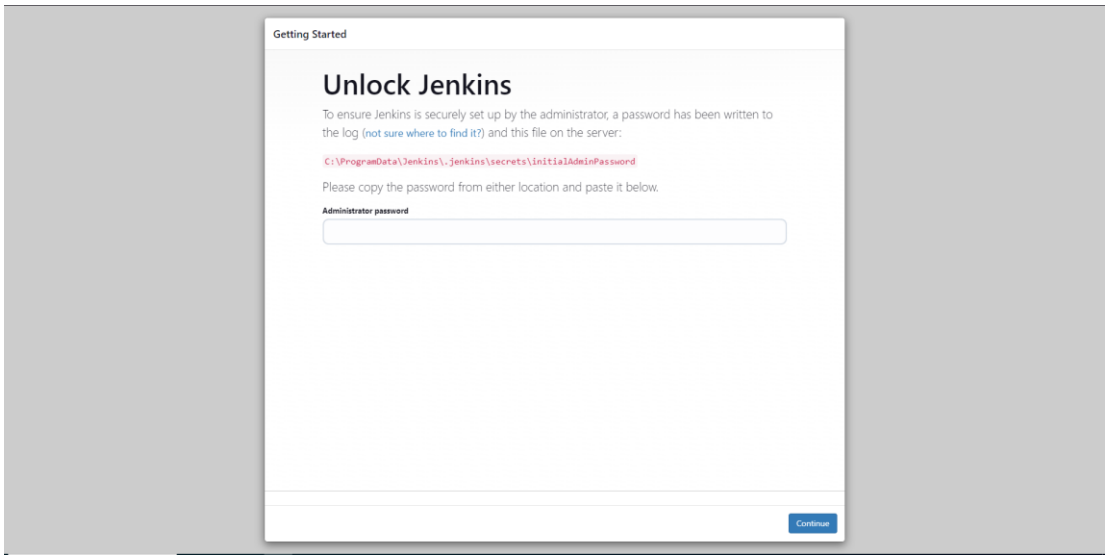

Рис.2.9. Встановлення шляху до JDK

Далі необхідно слідувати інструкціям з налаштування. Рекомендується встановити спочатку тільки базові плагіни і вже потім довстановити за потреби необхідні.

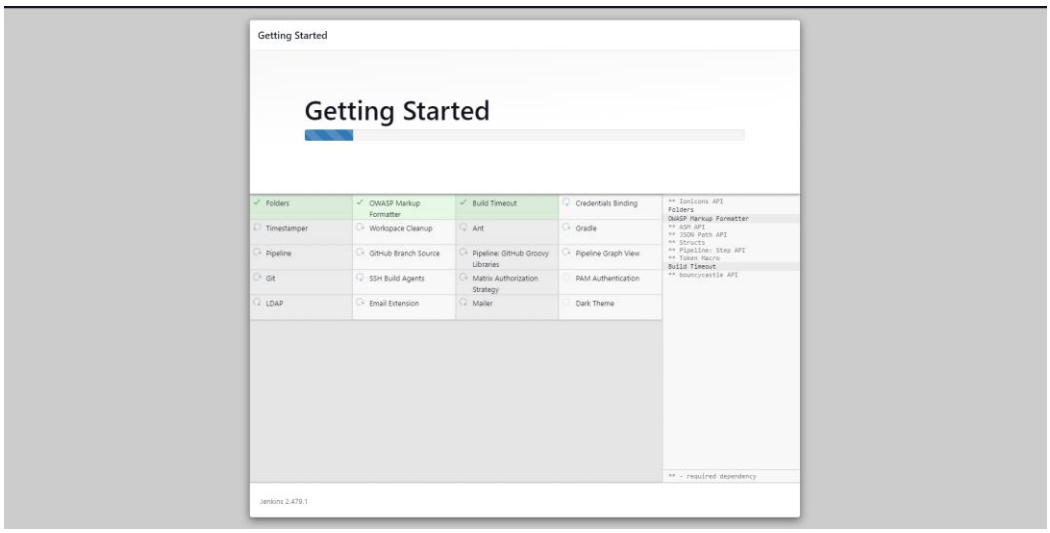

Рис.2.10. Встановлення плагінів jenkins

Jenkins встановлено, для роботи з Sonarqube та Gitlab необхідні плагіни серії Gitlab, а також плагін з назвою Sonarqube-scanner, який дозволить інтегрувати разом Sonarqube та Gitlab.

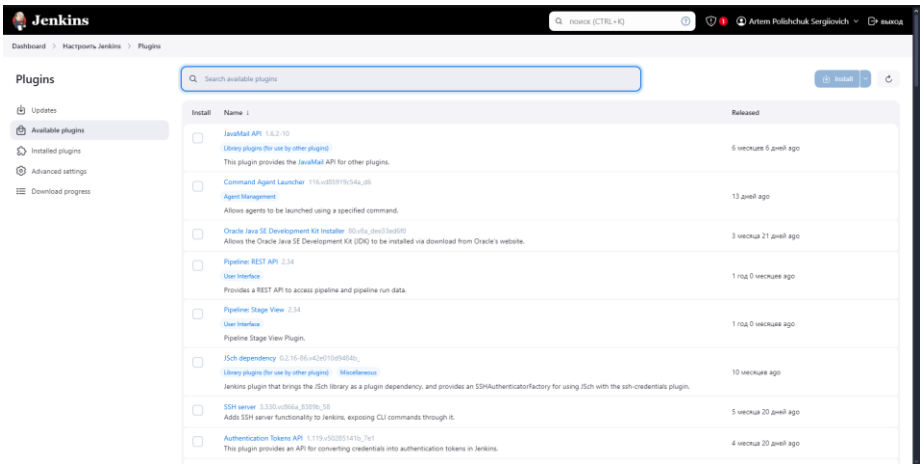

Рис.2.11. Встановлення додаткових плагінів

Висновок до розділу 2:

У другому розділі було проведено детальне дослідження практичних аспектів впровадження DevSecOps, зокрема інтеграції безпеки в конвеєри безперервної інтеграції та розгортання (CI/CD) із застосуванням інструментів SonarQube, GitLab та Jenkins. Метою цього етапу було визначити оптимальні методи та підходи до комплексної інтеграції безпеки на всіх етапах розробки, що сприяло підвищенню якості, надійності та захищеності програмного забезпечення.

Спочатку було проаналізовано процес планування впровадження DevSecOps, включно з оцінкою наявної інфраструктури та вимог безпеки. Далі увага зосереджувалася на встановленні та налаштуванні обраних інструментів. Зокрема, було детально проаналізовано інтеграцію SonarQube для автоматизованого статичного аналізу коду, GitLab для централізованого управління репозиторіями, моніторингу змін коду та запуску CI/CD конвеєрів, а також Jenkins для побудови складних процесів автоматизації, включаючи запуск тестів безпеки та сканувань.

3 ТЕХНОЛОГІЯ ВПРОВАДЖЕННЯ DEVSECOPS З ВИКОРИСТАННЯМ SONARQUBE

3.1 Топологія мережі

Мережа складається з 1 комп’ютера та 2 віртуальних машин, на одній встановлена операційна система Windows 10 з Jenkins, на іншій Ubuntu з Gitlab. На самому комп’ютері встановлено Sonarqube.

Також варто додати, що в процессі написання коду можна використовувати додаток до Visual Studio під назвою SonarLint (перейменовано в SonarQube for IDE: Visual Studio Code).

На відміну від більш масштабних рішень на зразок SonarQube, SonarLint працює локально та надає зворотний зв’язок негайно, без потреби вивантажувати зміни до репозиторію або запускати зовнішні аналізатори.

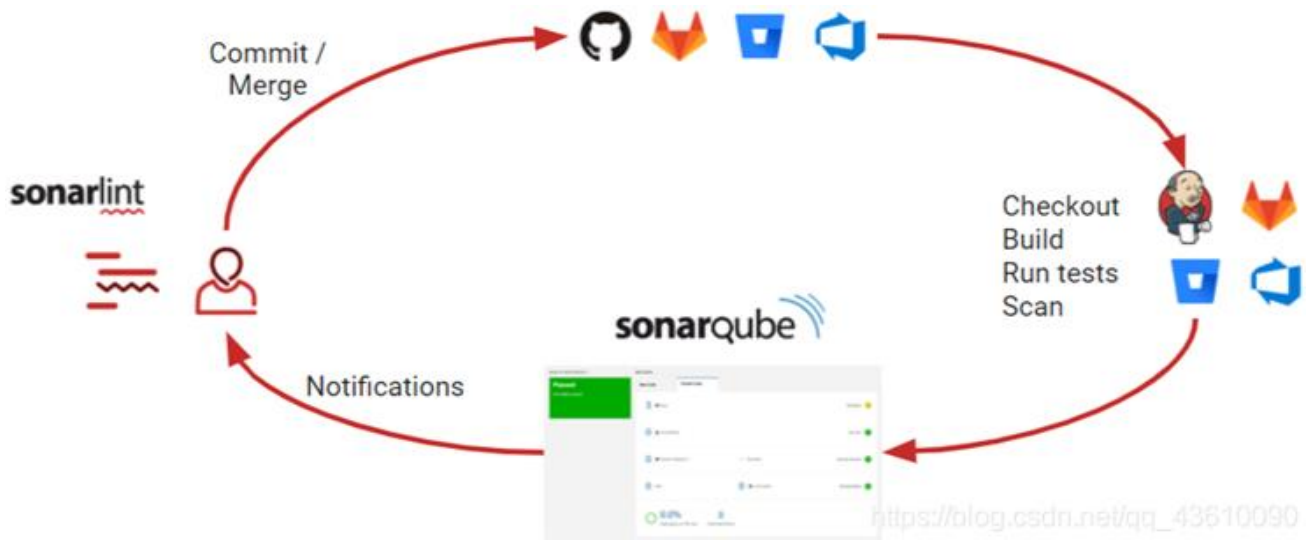

Рис. 3.1. Схема роботи

3.2 Впровадження DevSecOps в реальному проекті

Перш за все потрібно налаштувати інтеграцію програм між собою. Для того аби налаштувати зв’язок між sonarqube та gitlab необхідно:

1. Створення Персонального Токена в GitLab

Увійти у обліковий запис GitLab та перейти до User Settings > Access Tokens.

Налаштувати токен:

Назва (Token Name)

Термін дії (Expiration Date): Вибрати дату або залишити пустим, якщо не потрібен термін дії.

Scopes: Обрати api (мінімальний рівень доступу).

Create Personal Access Token.

Скопіювати токен — він відображається лише один раз.

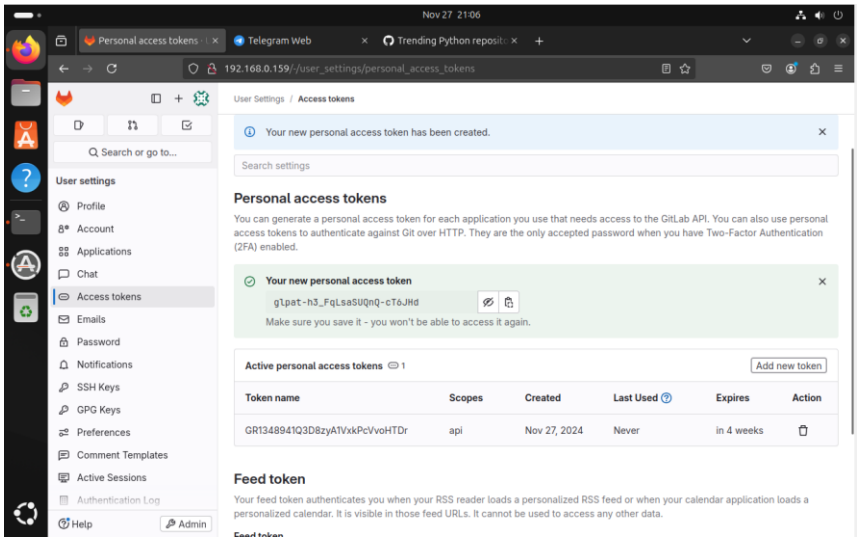

Рис. 3.2. Створений Access token

2. Інтеграція Sonarqube з Gitlab

Для налаштування інтеграції необхідно створити токен в розділі Application, для зв’язку з Sonarqube. При правильному налаштування Sonarqube з’єднається з gitlab.

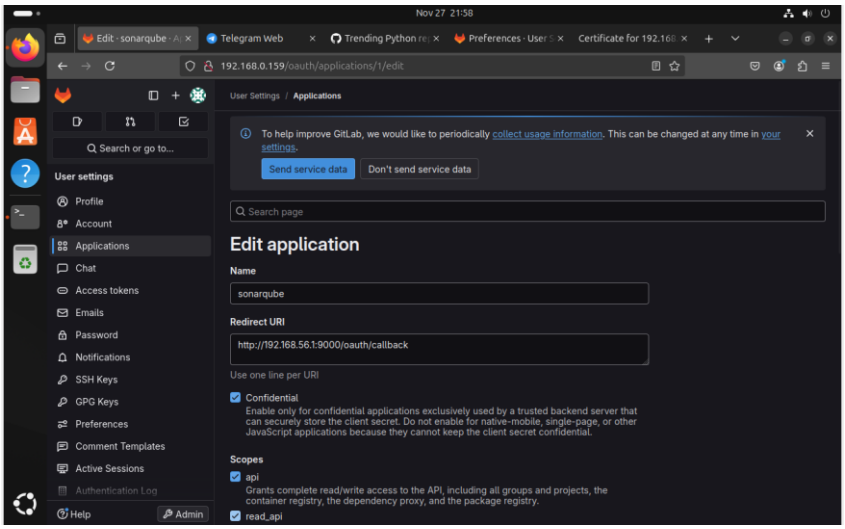

Рис. 3.3.Створення токену Sonarqube у розділі Application

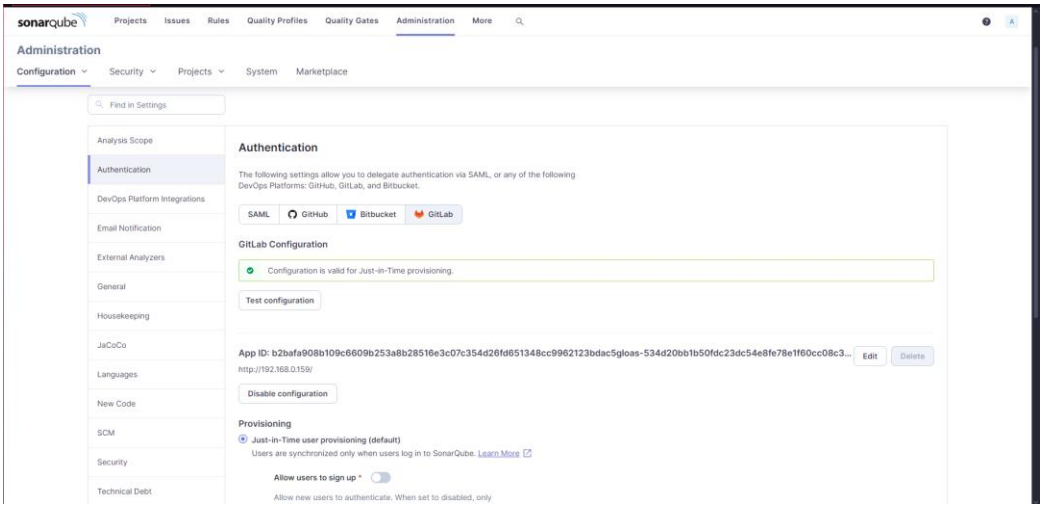

Рис. 3.4. Встановлення токену у Sonarqube та підствердження з’єднання

Наступним кроком буде створення токену для конкретного проекта та вкладення його у sonar.

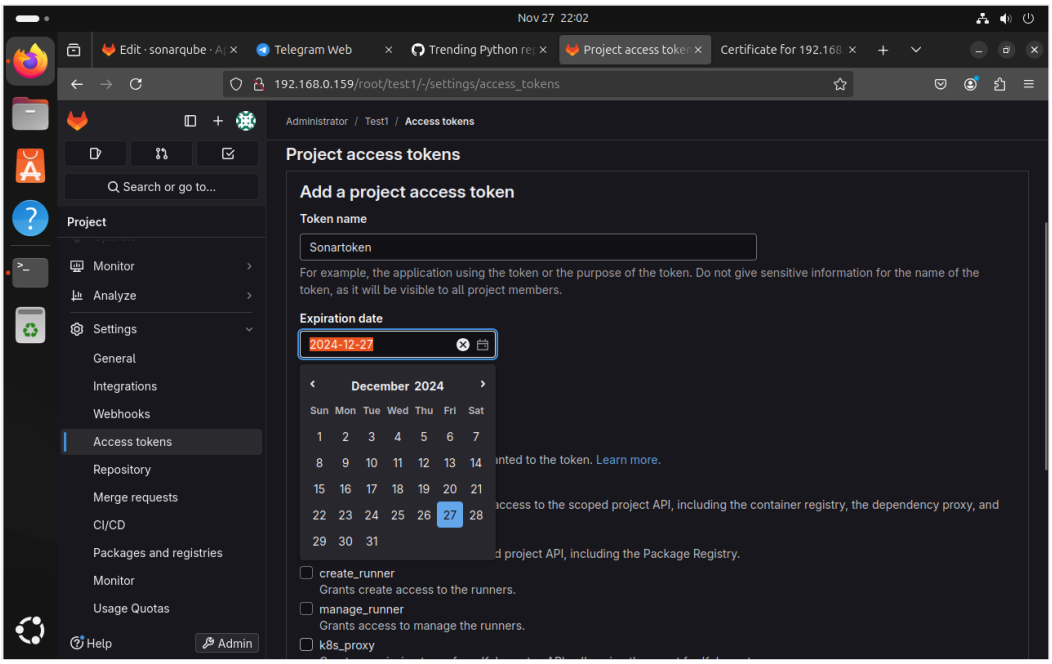

Рис. 3.5. Створення токену проекту для подальшого встановлення

Тепер можна імпортувати потрібний репозиторій.

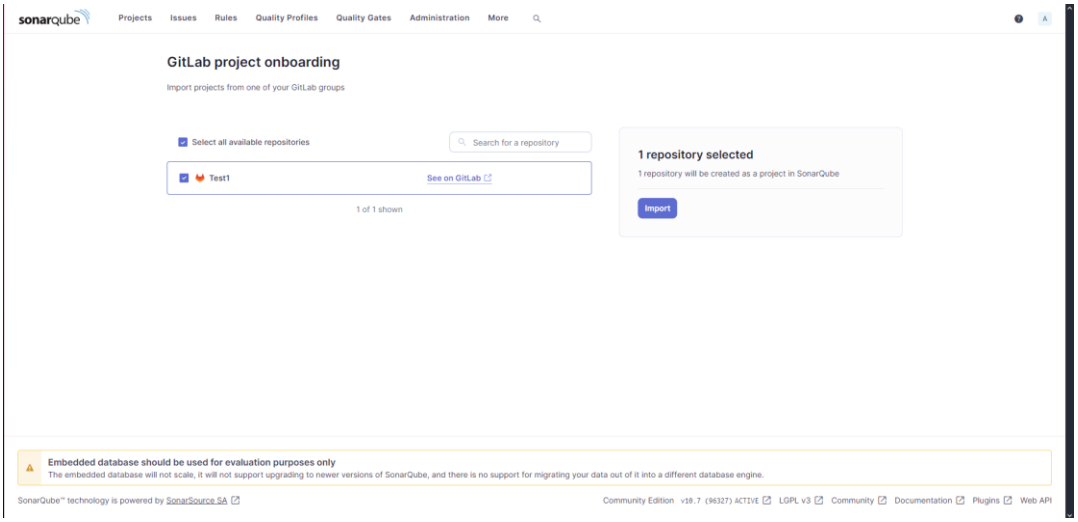

Рис. 3.6. Імпорт проекту в Sonar

Далі для конкретного проекту потрібно слідувати інструкціям які надає sonarqube для налаштування.

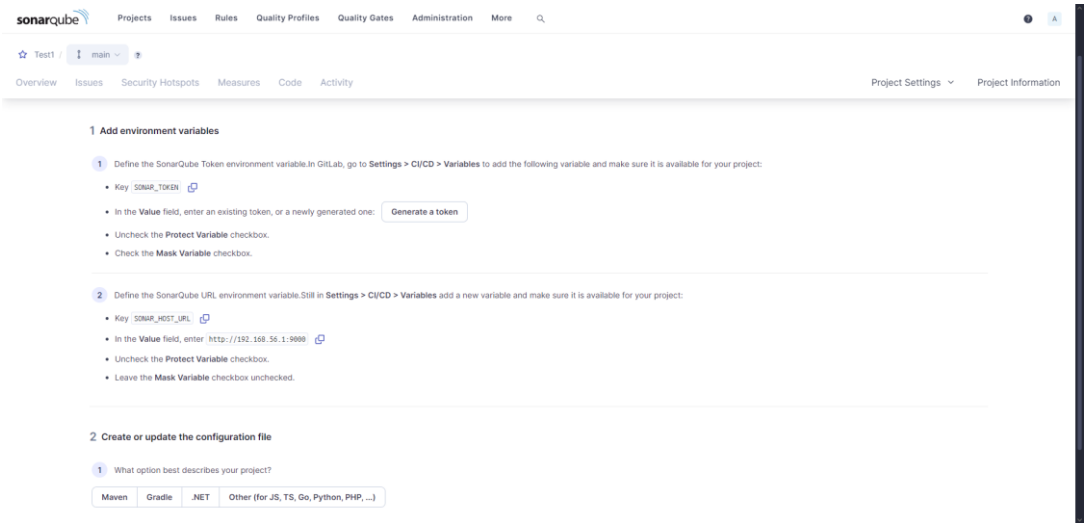

Рис. 3.7. Інструкції Sonarqube

Результатом буде закінчена перевірка. В Sonarqube буде висвітлено журнал де можна переглянути всі вразливості та помилки:

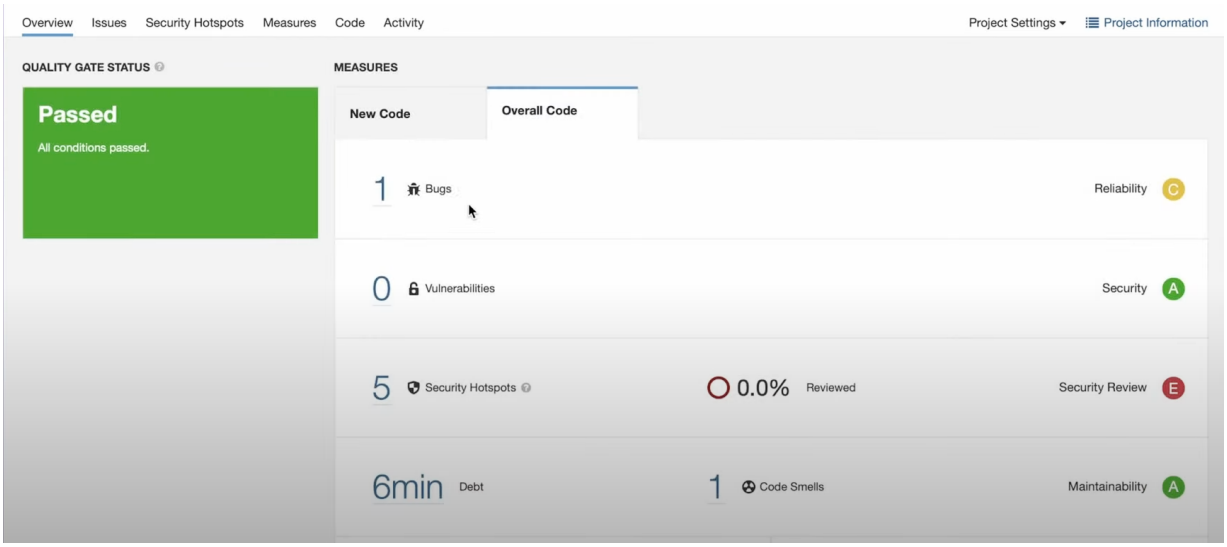

Рис. 3.8. Результати сканування

В даному випадку знайшло 5 вразливостей та 1 баг.

3. Інтеграція з Jenkins

Тепер необхідно все автоматизувати за допомогою Jenkins.

Налаштування зв’язку з Gitlab

Створення Персонального Токена в GitLab через Applications, що було в розділі про інтеграцію Gitlab.

Додавання облікових даних в Jenkins:

У Jenkins перейти до "Credentials" (або "Manage Jenkins" -> "Manage Credentials").

Рис. 3.9. Вікно Credentials

Додати нові "Credentials" типу "GitLab API Token":

Scope: Global

ID: gitlab-token

Description: Token for GitLab

API Token: Вставити згенерований токен з GitLab.

Рис. 3.10. Створення Credentials для jenkins

Налаштування GitLab сервера в Jenkins

Перейти до "Manage Jenkins" -> "Configure System". В секції "GitLab" додати новий сервер:

Name: GitLab Server

GitLab Host URL: https://gitlab.example.com (змінити на свій URL)

Credentials: gitlab-token, створений раніше.

Перевірити з'єднання, натиснувши "Test Connection".

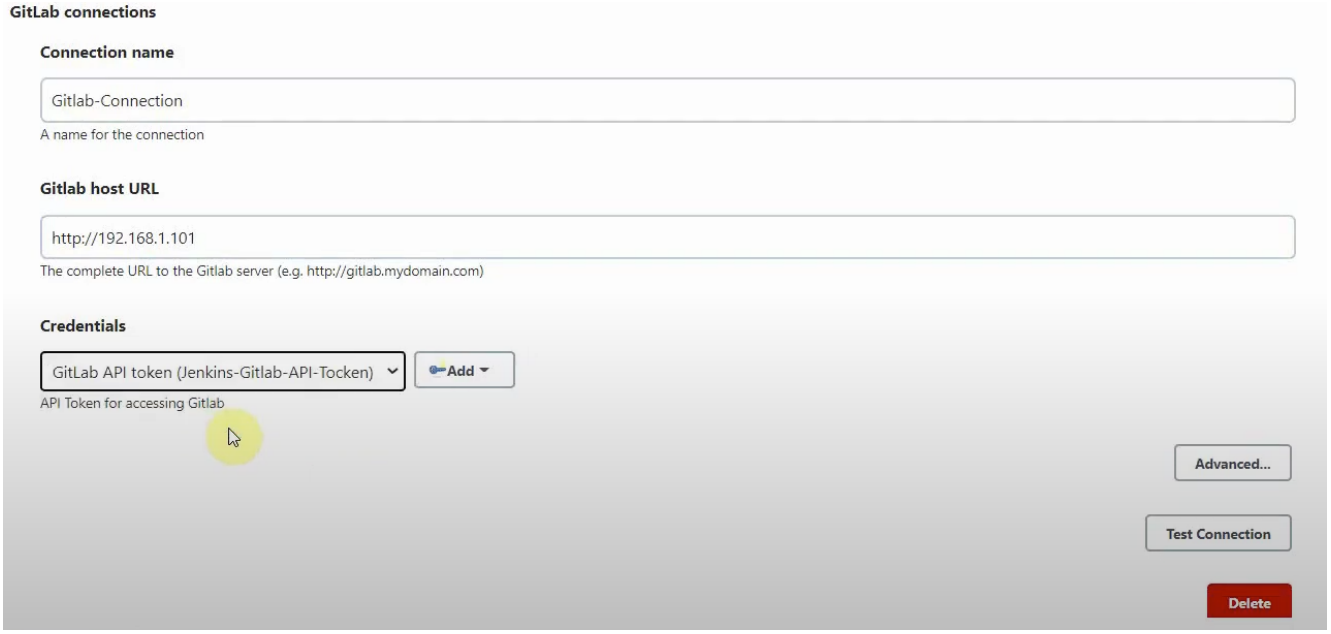

Рис. 3.11. Інтеграція Jenkins з Gitlab

Налаштування Sonarqube для інтеграції з Jenkins.

Крок 1: Встановлення плагіну SonarQube Scanner

У Jenkins перейти до "Manage Jenkins" -> "Manage Plugins". В розділі доступних плагінів встановити плагін sonar scanner після чого, цей крок завершено.

Крок 2: Налаштування SonarQube в Jenkins

"Manage Jenkins" -> "Configure System".

В секції "SonarQube servers" додати новий сервер:

Name: SonarQube Server

Server URL: http://sonarqube.example.com (змінити на свій SonarQube URL)

Server authentication token: Додати нові облікові дані типу "Secret Text" та вставити токен доступу з SonarQube.

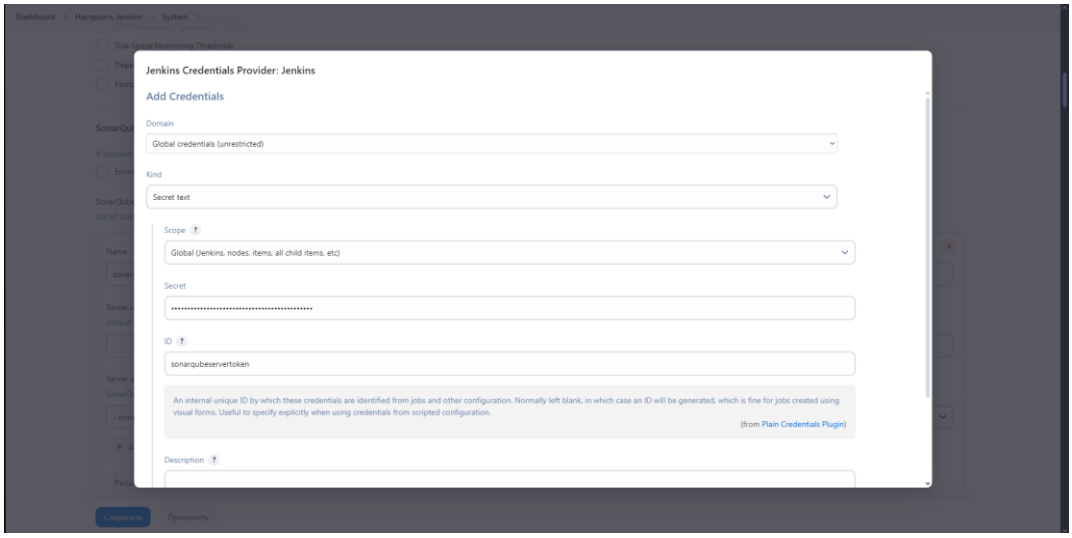

Рис. 3.12. Налаштування взаємодії Sonarqube server з jenkins

Крок 3: Налаштування SonarQube Scanner

Перейти до "Manage Jenkins" -> "Global Tool Configuration", знайти секцію "SonarQube Scanner" та додати новий сканер:

Name: SonarQube Scanner

Install automatically.

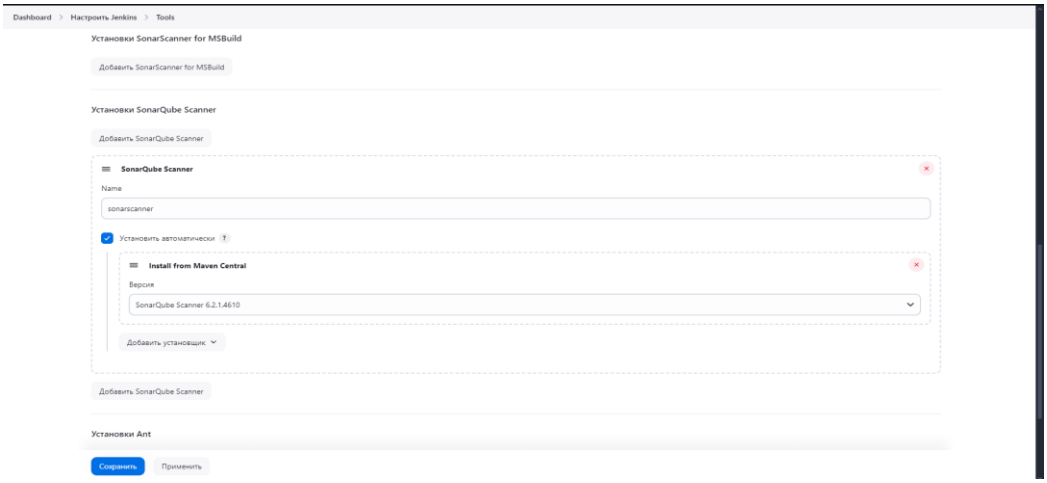

Рис. 3.12. інтеграція Sonarqube scanner у jenkins

Створення Jenkins Pipeline з інтеграцією GitLab та SonarQube

Крок 1: Створення нового Pipeline-проекту

У Jenkins обрати "New Item", ввести назву проекту та обрати тип "Pipeline". Одним з варіантів налаштування конвеєру є скрипт основні складові якого, наведено в наступному кроці.

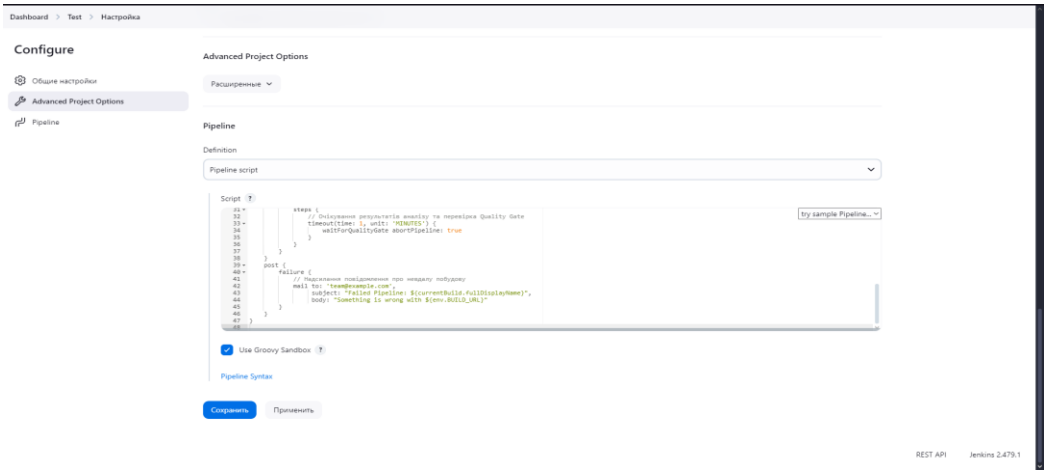

Рис. 3.13. Встановлення скрипта для конвеєру

Крок 2: Налаштування Jenkinsfile

environment: Встановлює змінні середовища, такі як облікові дані для GitLab.

stage('Checkout'): Витягує код з репозиторію GitLab.

stage('Build'): Компілює та будує проект.

stage('SonarQube Analysis'): Виконує аналіз коду за допомогою SonarQube.

stage('Quality Gate'): Чекає на результати Quality Gate та приймає рішення про продовження.

post: Дії після завершення конвеєра, наприклад, надсилання повідомлень про помилки.

Налаштування вебхуків у GitLab для Jenkins

Щоб Jenkins автоматично запускав побудову при змінах у репозиторії GitLab, необхідно налаштувати вебхуки.

Кроки налаштування:

У GitLab:

Перейти до проекту.

Виберіть "Settings" -> "Webhooks".

Додати новий вебхук:

URL: http://jenkins.example.com/project/your_project (URL Jenkins проекту)

Secret Token: (необов'язково) для додаткової безпеки.

Trigger: Обрати події, які повинні запускати побудову, наприклад, Push Events, Merge Request Events.

У Jenkins:

У налаштуваннях проекту переконатись, що опція "Build when a change is pushed to GitLab" увімкнена.

Встановити "Secret Token" (якщо використовували в GitLab).

Результати збірок будуть висвітлюватись на панелі build history.

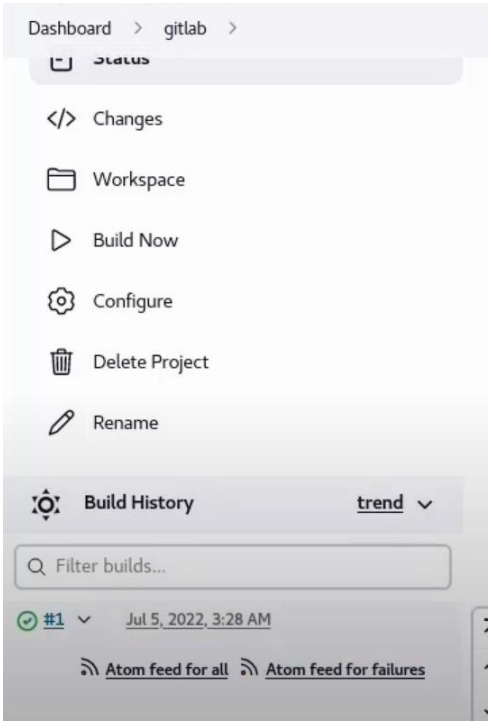

Рис. 3.14. Конвеєр працює

Налаштування завершено.

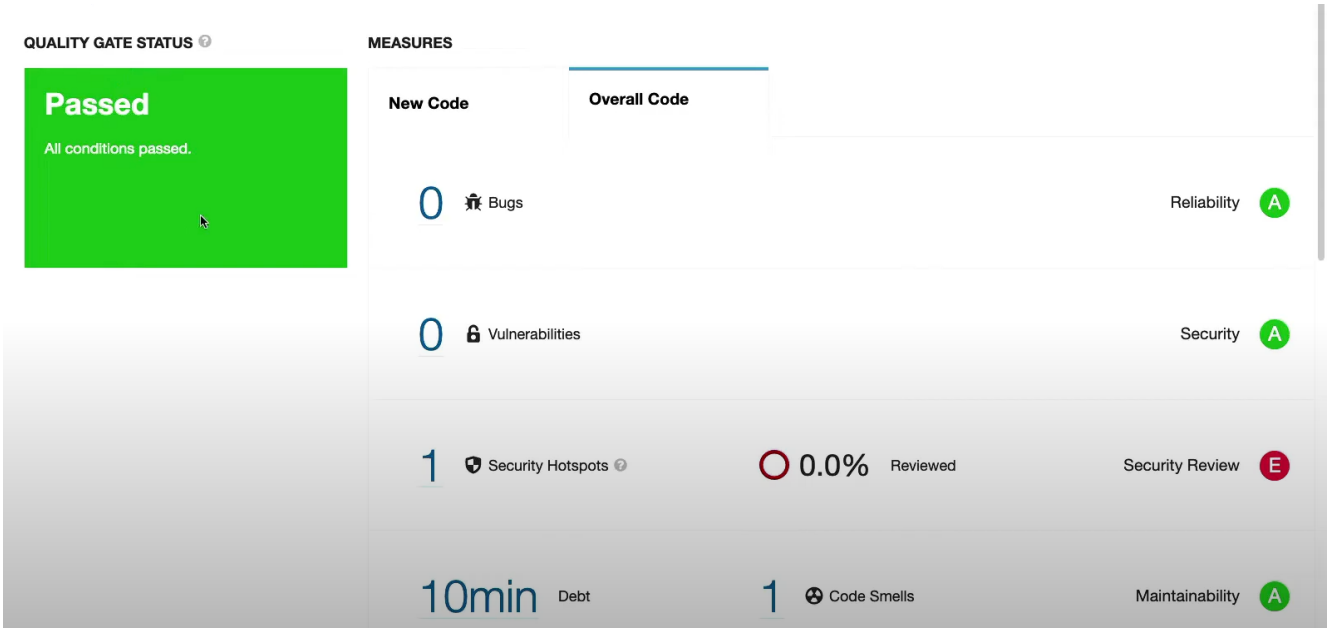

Рис. 3.15. Результати перевірки іншого проекту через повну інтеграцію

3.3 Рекомендації щодо впровадження DevSECOps з використанням Sonarqube

Впровадження DevSecOps з використанням SonarQube є важливим кроком для підвищення рівня безпеки та якості програмного забезпечення в організації. При цьому необхідно оцінити ефективність інтеграції безпеки в процеси розробки та визначити, на якому рівні знаходяться потенційні ризики, пов'язані з вразливостями в коді.

Необхідно налаштувати правила та профілі якості в SonarQube відповідно до вимог проекту. Це забезпечить фокусування на найбільш критичних вразливостях та допоможе уникнути хибнопозитивних спрацювань, які можуть відволікати команду. Регулярний перегляд та оновлення цих правил сприятиме актуальності аналізу та відповідності сучасним стандартам безпеки.

Навчання команди розробників принципам безпечного програмування та використання інструментів статичного аналізу є невід'ємною частиною успішного впровадження DevSecOps. Розуміння важливості безпеки та вміння ефективно

використовувати SonarQube підвищує загальний рівень захищеності програмного забезпечення. Проведення тренінгів та семінарів сприятиме підвищенню компетенції команди.

Також важливо забезпечити постійний моніторинг та швидке реагування на виявлені вразливості. Інтеграція з системами управління завданнями та автоматичні сповіщення дозволять оперативно виправляти проблеми та мінімізувати ризики. Впровадження таких практик сприятиме створенню культури безпеки в організації.

Висновок до розділу 3:

У третьому розділі було об’єднано результати дослідження інструментів DevSecOps та перевірено їх практичне застосування в побудові CI/CD конвеєрів із впровадженою безпекою. Зокрема, розглянуто автоматизоване тестування коду за допомогою SonarQube, інтегроване в роботу GitLab і Jenkins, що дало змогу виявляти та виправляти вразливості на ранніх етапах розробки. Проведений аналіз підтвердив, що поєднання цих інструментів дозволяє підвищити рівень захищеності програмного забезпечення, скоротити час реагування на виявлені проблеми та оптимізувати взаємодію між командами розробки й безпеки. Запропоновані рекомендації щодо налаштування та інтеграції DevSecOps-практик у процеси CI/CD становлять основу для успішного впровадження підходу в організаціях різного масштабу та сприяють більш надійному функціонуванню програмних продуктів.

ВИСНОВКИ

У цій роботі було проаналізовано ключові аспекти інтеграції DevSecOps з використанням інструменту SonarQube, а також запропоновано рекомендації щодо впровадження практик DevSecOps у процес розробки програмного забезпечення.

Дослідження показало, що статичний аналіз коду за допомогою SonarQube є ефективним методом для раннього виявлення вразливостей, кодових запахів та недоліків у коді. Інтеграція SonarQube в CI/CD конвеєри, такі як Jenkins та GitLab CI/CD, дозволяє автоматизувати процес перевірки коду та оперативно надавати зворотний зв’язок розробникам. Водночас, важливим є налаштування Quality Gates та профілів якості, що забезпечують стандартизацію вимог до коду.

Запропоновані рекомендації включають інтеграцію SonarQube в усі етапи життєвого циклу розробки, налаштування інструменту відповідно до специфіки проекту, навчання команди принципам безпечного програмування та автоматизацію реагування на виявлені вразливості. Особливий акцент зроблено на співпраці команд розробки, експлуатації та безпеки для досягнення спільних цілей.

Важливим аспектом впровадження DevSecOps є постійне вдосконалення процесів, оновлення інструментів та адаптація до нових викликів у сфері кіберзагроз. Використання SonarQube у поєднанні з іншими інструментами аналізу безпеки, такими як SAST, DAST та SCA, забезпечує всебічний підхід до виявлення вразливостей.

Отже, впровадження DevSecOps з використанням SonarQube є ефективним кроком для забезпечення високої якості та безпеки програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Поліщук А. С. Технологія впровадження DevSecOps з використанням Sonarqube. Всеукраїнська наукова конференція «Актуальні проблеми кібербезпеки». Державний університет інформаційно-комунікаційних технологій. 25 жовтня 2024. Тези доповідей. С. 144–145. URL: https://duikt.edu.ua/uploads/p_2661_48963150.pdf (дата звернення: 26.11.2024).

2. Kim G., Humble J., Debois P., Willis J. The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations. Portland, OR : IT Revolution Press, 2016. 480 p.

3. Bass L., Weber I., Zhu L. DevOps: A Software Architect’s Perspective. Boston, MA : Addison-Wesley Professional, 2015. 320 p.

4. Kerner S. M. DevSecOps: A Quick Start Guide. Birmingham : Packt Publishing, 2019. 250 p.

5. SonarSource SA. SonarQube Documentation : веб-сайт. URL: https://docs.sonarqube.org (дата звернення: 01.10.2024).

6. GitLab Inc. GitLab CI/CD Documentation : веб-сайт. URL: https://docs.gitlab.com/ee/ci (дата звернення: 01.10.2024).

7. Jenkins Project. Jenkins User Documentation : веб-сайт. URL: https://www.jenkins.io/doc (дата звернення: 01.10.2024).

8. OWASP Zed Attack Proxy (ZAP) : веб-сайт. URL: https://www.zaproxy.org (дата звернення: 02.10.2024).

9. Snyk Ltd. Snyk User Documentation : веб-сайт. URL: https://docs.snyk.io (дата звернення: 02.10.2024).

10. Aqua Security. Aqua Platform Documentation : веб-сайт. URL: https://docs.aquasec.com (дата звернення: 04.10.2024).

11. Micro Focus. Fortify Static Code Analyzer Documentation : веб-сайт. URL: https://www.microfocus.com/documentation/fortify-static-code-analyzer (дата звернення: 07.10.2024).

12. Checkmarx. Checkmarx Documentation : веб-сайт. URL: https://checkmarx.com/resource-center/documentation (дата звернення: 06.11.2024).

13. Veracode. Veracode Documentation : веб-сайт. URL: https://docs.veracode.com (дата звернення: 01.11.2024).

14. National Institute of Standards and Technology (NIST). Framework for Improving Critical Infrastructure Cybersecurity : веб-сайт. URL: https://www.nist.gov/cyberframework (дата звернення: 12.10.2024).

15. Puppet & SANS Institute. State of DevOps Report : веб-сайт. URL: https://puppet.com/resources/report/state-of-devops-report (дата звернення: 05.10.2024).

16. HashiCorp. Terraform Documentation : веб-сайт. URL: https://www.terraform.io/docs (дата звернення: 05.10.2024).

17. HashiCorp. Vault Documentation : веб-сайт. URL: https://www.vaultproject.io/docs (дата звернення: 05.10.2024).

18. Docker Inc. Docker Documentation : веб-сайт. URL: https://docs.docker.com (дата звернення: 02.11.2024).

19. Kubernetes. Kubernetes Documentation : веб-сайт. URL: https://kubernetes.io/docs (дата звернення: 05.11.2024).

20. Ansible by Red Hat. Ansible Documentation : веб-сайт. URL: https://docs.ansible.com (дата звернення: 03.11.2024).

21. Open Policy Agent (OPA). OPA Documentation : веб-сайт. URL: https://www.openpolicyagent.org/docs (дата звернення: 03.11.2024).

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)

2

Мета, Об’єкт, Завдання дослідження та предмет дослідження

Актуальність: сучасне програмне забезпечення є ключовою складовою діяльності більшості організацій, проте традиційні методи захисту, що впроваджуються після завершення розробки, вже не гарантують належного рівня безпеки. Кіберзагрози постійно змінюються, а невпинне зростання складності додатків збільшує ризик витоків даних, фінансових втрат та репутаційних збитків. У відповідь виникає DevSecOps, який дає змогу інтегрувати безпеку на початок життєвого циклу розробки та швидко виявляти вразливості.

Мета дослідження: дослідити теоретичні засади та практичні аспекти впровадження DevSecOps із використанням SonarQube, GitLab та Jenkins, а також сформувати рекомендації щодо інтеграції безпеки в процеси CI/CD для підвищення рівня захищеності та якості програмного забезпечення.

Предмет дослідження: методи та інструменти DevSecOps, а також підходи до їх ефективної інтеграції в конвеєри CI/CD.

Об'єкт дослідження: процеси забезпечення безпеки програмного забезпечення в умовах безперервної інтеграції та розгортання.

Завдання дослідження:

- Дослідити теоретичні засади DevSecOps та визначити його ключові відмінності від традиційних підходів до розробки й безпеки.
- Проаналізувати методи й засоби DevSecOps, що забезпечують інтеграцію безпеки на всіх етапах життєвого циклу програмного забезпечення.
- Оцінити ефективність використання SonarQube, GitLab і Jenkins для впровадження DevSecOps у процеси безперервної інтеграції та розгортання (CI/CD).
- Розробити рекомендації щодо практичного застосування DevSecOps, зокрема інтеграції безпеки в CI/CD, і перевірити дієвість цих рекомендацій на реальних прикладах.

3

Що таке DevSecOps та його відмінності від DevOps

- DevSecOps — це еволюція DevOps, яка інтегрує безпеку (Security) в усі етапи розробки та експлуатації програмного забезпечення. Якщо DevOps зосереджується на прискоренні випуску продуктів через тісну співпрацю команд розробки та експлуатації, то DevSecOps додає до цього підходу безперервний контроль безпеки. Такий підхід передбачає, що захист даних та виявлення вразливостей відбуваються з самого початку життєвого циклу ПЗ, а не окремо чи після завершення розробки.

4

Рішення для DevSecOps

- **Статичний аналіз коду (SAST)** дозволяє досліджувати вихідний код без його фактичного виконання. Такий підхід дає змогу розробникам виявляти потенційні помилки та вразливості ще під час написання коду. Інструменти на зразок SonarQube перевіряють синтаксис і структуру вихідних файлів та співставляють їх із базами відомих уразливостей або правил безпечного кодування. Це дає змогу зменшити кількість проблем безпеки, котрі могли би потрапити в продакшн-середовище, а також підвищити загальну якість і підтримуваність проєктів. Водночас, статичний аналіз обмежений тим, що не відображає реальної поведінки додатку під час взаємодії з базами даних, хмарними сервісами чи сторонніми API.
- **Динамічний аналіз безпеки (DAST)** полягає в перевірці працюючого додатку, коли він вже розгорнутий у тестовому середовищі. Інструменти, такі як OWASP ZAP, імітують дії потенційних зловмисників і надсилають специфічні запити з метою виявлення вразливостей, пов'язаних з некоректною обробкою даних, помилками конфігурації або логікою взаємодії компонентів. Подібний підхід краще відображає реальні умови роботи додатку, дозволяючи виявити проблеми, які неможливо знайти статичним аналізом. Проте, оскільки DAST реалізує стратегію «чорної скриньки», інструмент не має доступу до коду і не може виявити проблем, що не проявляються під час зовнішніх запитів.
- **Інтерактивний аналіз безпеки (IAST)** поєднує принципи статичного та динамічного підходів. У випадку IAST в додатку запускається спеціальний агент, який відслідковує потоки даних та взаємодію ззовні під час виконання або автоматизованих тестів. Такий метод дає змогу більш точного виявлення уразливостей і знижує ймовірність хибнопозитивних результатів, оскільки агент розуміє контекст внутрішньої логіки додатку. Втім, IAST вимогливий до налаштувань і може потребувати більше ресурсів, бо вимагає встановлення агентів і тісної інтеграції з середовищем виконання.

sonarqube

OWASP Zed Attack Proxy

CONTRAST SECURITY

5

Схема роботи Sonarlint, Gitlab, Jenkins, Sonarqube

Крім Gitlab, Sonarqube та Jenkins, можна використовувати також Sonarlint. На відміну від більш масштабних рішень на зразок SonarQube, SonarLint працює локально та надає зворотний зв'язок негайно, без потреби вивантажувати зміни до репозиторію або запускати зовнішні аналізатори.

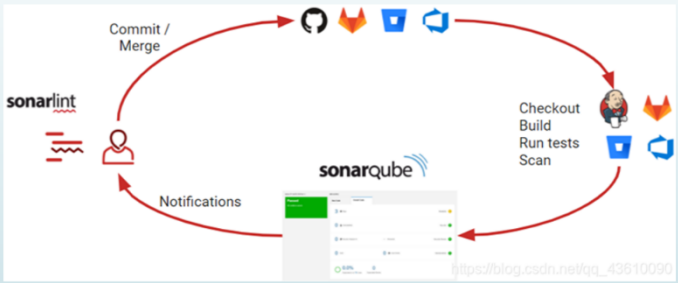

Рис. 5.1. Схема роботи.

6

Sonarqube

- **SonarQube** — це платформа для статичного аналізу коду, яка автоматично виявляє баги та вразливості у вихідному коді. Вона підтримує різні мови програмування й легко інтегрується з CI/CD конвеєрами (наприклад, Jenkins та GitLab). Результати аналізу відображаються у вигляді метрик та звітів, а «Quality Gates» забезпечують автоматичний контроль безпеки і якості коду. Таким чином, SonarQube сприяє ранньому виявленню проблем, знижує вартість їх виправлення й підвищує надійність програмних продуктів.

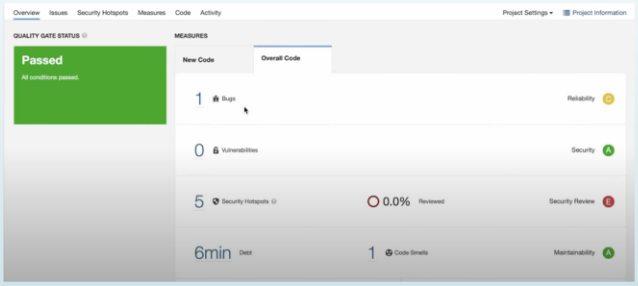

Рис. 6.1. Звіт з роботи Sonarqube.

7

Gitlab

- **GitLab** — це комплексна платформа для керування репозиторіями, планування та безперервної інтеграції/розгортання (CI/CD). Вона поєднує систему контролю версій Git із засобами автоматичного збирання та тестування коду, а також надає вбудовані інструменти безпеки (SAST, DAST, SCA). Такий підхід створює єдиний робочий простір для розробників і спеціалістів з безпеки, забезпечуючи пришвидшене постачання та вищий рівень захищеності програмного забезпечення.

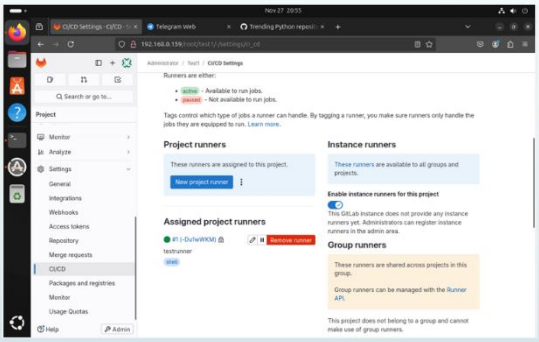

Рис. 7.1 Створення одного з runner для проектів

8

Jenkins

- **Jenkins** — це відкрита платформа для організації безперервної інтеграції та розгортання (CI/CD). Вона приймає завдання (job) з репозиторію або іншого джерела, виконує збирання та тестування коду, а також управляє розгортанням. Завдяки системі плагінів Jenkins легко адаптується до специфіки проєкту, інтегрується з різними інструментами безпеки й мовами програмування, що дозволяє ефективно автоматизувати процес розробки та швидко реагувати на виявлені вразливості.

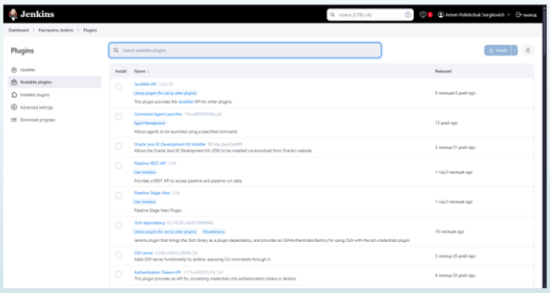

Рис. 8.1 Вікно вибору плагінів Jenkins

9

РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ВПРОВАДЖЕННЯ DevSECOps З ВИКОРИСТАННЯМ Sonarqube

- **Оцінити ризики та налаштувати.** Спершу визначити найбільш критичні місця проєкту і відповідно налаштувати Quality Gates та профілі аналізу в SonarQube. Це дасть змогу уникнути хибнопозитивних спрацювань і зосередитись на реальних загрозах.
- **Інтегрувати SonarQube в процеси розробки.** Включити статичний аналіз коду як етап у CI/CD, щоб автоматично виконувати перевірки безпеки під час кожної збірки чи злиття гілок. Це сприяє ранньому виявленню та швидкому виправленню проблем.
- **Регулярно оновлювати політики та профілі.** Періодично переглядати та актуалізувати правила аналізу відповідно до нових вимог безпеки і типів загроз. Це допомагає підтримувати ефективність SonarQube, адже застарілі політики можуть призвести до пропуску небезпечних проблем або генерації надлишкових хибнопозитивів.

10

Висновки

- В роботі було розглянуто підходи інтеграції DevSecOps з використанням SonarQube для підвищення безпеки та якості програмного забезпечення.
- Проведено аналіз ефективності інструментів статичного аналізу, а також розроблено рекомендації щодо налаштування та впровадження їх у конвеєри CI/CD.
- Та розроблено рекомендації щодо впровадження DevSecOps з використанням Sonarqube.

11

Апробація результатів

Всеукраїнська наукова конференція «Актуальні проблеми кібербезпеки». 25 жовтня 2024