МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

Клім Віталій Володимирович

Безпека контейнеризованих середовищ та Kubernetes / Security of Containerized Environments and Kubernetes

спеціальність: 125 – Кібербезпека та захист інформації

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБзм -21

Клім Віталій Володимирович

Науковий керівник
к.т.н., доцент Т.Г. Цаволик

Кваліфікаційну роботу
допущено до захисту:

«____» _____ ___2025 р.

Завідувач кафедри
_____ **В.В. Яцків**

ТЕРНОПІЛЬ - 2025

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В. Яцків
«____» _____ 20___ року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Клім Віталій Володимирович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Безпека контейнеризованих середовищ та Kubernetes / Security of Containerized Environments and Kubernetes
керівник роботи: к.т.н., доцент Т.Г. Цаволик
затверджені наказом по університету 20 грудня 2024 року № 938

2. Строк подання студентом закінченої кваліфікаційної роботи 5 грудня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:
– описати можливості та особливості контейнеризації та платформи Kubernetes;
– розглянути типи контейнеризованих застосунків та конфігурації кластерів Kubernetes;
– запропонувати політики безпеки для захисту контейнеризованих застосунків;
– запропонувати механізми контролю доступу та ізоляції ресурсів у Kubernetes;
– розглянути засоби моніторингу та виявлення аномалій ;
– розробити алгоритм реагування на інциденти безпеки в Kubernetes-кластері;
– описати інтеграцію запропонованих рішень із системами моніторингу;
– описати основні загрози та вразливості контейнеризованих середовищ;
– оцінити ефективність запропонованих політик та механізмів захисту;
– сформувати рекомендації щодо впровадження розроблених рішень;

5. Перелік графічного матеріалу у роботі:
– простори імен розгорнуті у кластері minikube;
– стан ресурсів kyverno,opa та falco;
– індекси elastic;
– стан кластера minikube;
– маніфест файл для створення простору імен;
– структурна схема застосування політики Gatekeeper.;
– приклад власних правил для ConfigMap;
– фрагмент конфігурації Filebeat;
– інформаційна панель Kibana;
– результат логів від час порушння політи безпеки;
– отриманні логи під час порушення політие безпеки всередині контейнера.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання: 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1		12.2024 р. – 03.2025 р.	
2		03.2025 р. – 06.2025 р.	
3		06.2025 р. – 11.2025 р.	

Студент _____ Клім В.В

Керівник роботи _____ к.т.н., доц. Т.Г. Цаволик

АНОТАЦІЯ

Кваліфікаційна робота на тему «Безпека контейнеризованих середовищ та Kubernetes» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека та захист інформації» написана обсягом 98 сторінок і містить 15 ілюстрацій, 5 таблиць, 2 додатки та 27 джерел.

Метою роботи є підвищення рівня захищеності контейнеризованих середовищ і кластерів Kubernetes шляхом удосконалення механізмів політик безпеки та виявлення аномалій.

Методи досліджень. Аналіз загроз і вразливостей контейнеризованих інфраструктур, моделювання атак, проектування політик безпеки, поведінкова аналітика подій, засоби моніторингу Elastic Stack.

Результати дослідження: сформовано багаторівневий підхід до застосування політик безпеки; розроблено інтегровану систему, що поєднує контроль політик з автоматизованим виявленням аномалій; змодельовано типові атаки та оцінено ефективність запропонованого рішення. Отримані результати підтверджують можливість підвищення точності й своєчасності виявлення інцидентів у Kubernetes.

Результати роботи можуть бути використані для посилення безпеки контейнеризованих середовищ у хмарних та гібридних інфраструктурах і вдосконалення процесів моніторингу та реагування на інциденти.

Ключові слова: KUBERNETES, КОНТЕЙНЕРИЗАЦІЯ, ПОЛІТИКИ БЕЗПЕКИ, АНОМАЛІЇ, ELASTIC STACK.

ANNOTATION

The qualification thesis titled “Security of Containerized Environments and Kubernetes” submitted for obtaining the Master’s degree in specialty 125 “Cybersecurity and Information Protection” consists of 98 pages and includes 15 figures, 5 tables, 2 appendix, and 27 referenced sources. The purpose of the thesis is to enhance the security of containerized environments and Kubernetes clusters by improving mechanisms of dynamic security policy enforcement and anomaly detection

Research methods include analysis of threats and vulnerabilities of containerized infrastructures, attack modelling, security policy design, behavioral event analytics, and monitoring approaches based on the Elastic Stack

Research results: a multilayer approach to applying security policies in Kubernetes has been developed; an integrated system combining policy control with automated anomaly detection has been implemented; typical attack scenarios have been modelled, and the effectiveness of the proposed solution has been evaluated in terms of accuracy and timeliness of incident detection

The results may be applied to strengthening the security of containerized environments in cloud and hybrid infrastructures and improving monitoring and incident response processes in organizations operating Kubernetes-based systems

Key words: KUBERNETES, CONTAINERIZATION, SECURITY POLICIES, ANOMALIES, ELASTIC STACK

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface, програмний інтерфейс прикладного програмування

Kubernetes (K8s) – система оркестрації контейнеризованих застосунків з відкритим кодом

VM (ВМ) – Virtual Machine, віртуальна машина

Pod – мінімальна логічна одиниця розгортання в Kubernetes, що може містити один або декілька контейнерів

Namespace (NS) – логічно ізольований простір імен у кластері Kubernetes

RBAC – Role-Based Access Control, модель керування доступом на основі ролей

ABAC – Attribute-Based Access Control, модель керування доступом на основі атрибутів

DAC – Discretionary Access Control, дискреційна модель керування доступом, у якій власник ресурсу визначає права доступу

MAC – Mandatory Access Control, мандатна модель керування доступом, що ґрунтується на централізованій політиці та мітках безпеки

PSA – Pod Security Admission, вбудований механізм контролю безпеки подів у Kubernetes

PSS – Pod Security Standards, стандартизовані профілі безпеки подів у Kubernetes

CIS – Center for Internet Security, організація, що розробляє стандарти та рекомендації з кібербезпеки (зокрема CIS Kubernetes Benchmark)

NIST – National Institute of Standards and Technology, розробник стандартів і рекомендацій у сфері інформаційної безпеки (зокрема NIST SP 800-190)

CVE – Common Vulnerabilities and Exposures, публічний реєстр ідентифікаторів відомих вразливостей

TLS – Transport Layer Security, криптографічний протокол захисту мережевих з’єднань

eBPF – extended Berkeley Packet Filter, технологія розширюваного фільтрування та трасування подій у ядрах Linux

OPA – Open Policy Agent, універсальний рушій політик для хмарних і контейнеризованих середовищ

Rego – декларативна мова опису політик для рушія Open Policy Agent

Gatekeeper – компонент інтеграції OPA з Kubernetes як admission-контролера

Kyverno – Kubernetes-native рушій політик, у якому політики описуються у вигляді ресурсів Kubernetes

CRD – CustomResourceDefinition, механізм розширення API Kubernetes за рахунок визначення власних типів ресурсів

CI/CD – Continuous Integration / Continuous Delivery, безперервна інтеграція та безперервне постачання

GitOps – підхід до керування інфраструктурою та конфігураціями

PaC – Policy-as-Code, підхід, за якого політики безпеки, доступу й комплаєнсу описуються у вигляді коду й керуються через системи контролю версій

CPU – Central Processing Unit, центральний процесор

RAM – Random Access Memory, оперативна пам’ять з довільним доступом

ВСТУП

Актуальність роботи. Активне впровадження контейнеризації та мікросервісної архітектури призвело до широкого використання платформи Kubernetes для розгортання та масштабування програмних систем. Разом із перевагами гнучкості та автоматизації зростає кількість кіберзагроз, пов’язаних із неправильною конфігурацією кластерів, відсутністю контролю доступу та недостатнім моніторингом подій безпеки. Умови швидких змін у DevOps-процесах роблять традиційні статичні моделі захисту недостатньо ефективними, тому актуальним є застосування динамічних політик безпеки та засобів виявлення аномалій у реальному часі.

Мета і завдання дослідження. Метою роботи є підвищення ефективності захисту Kubernetes-кластерів шляхом розроблення та експериментальної перевірки комплексного підходу, що поєднує динамічне застосування політик безпеки, аналіз вразливостей контейнерних образів і виявлення аномалій у реальному часі.

Для досягнення поставленої мети необхідно розв’язати такі **завдання**:

– проаналізувати архітектуру Kubernetes, типові загрози та вразливості контейнеризованих середовищ;

– дослідити вбудовані механізми безпеки Kubernetes;

– проаналізувати підходи до динамічного застосування політик безпеки та обґрунтувати вибір інструментів Kyverno, Falco, Trivy та Elastic Stack;

– розробити модель політик безпеки для різних класів робочих навантажень у Kubernetes-кластері;

– спроєктувати архітектуру тестового середовища Kubernetes з інтеграцією обраних інструментів;

– реалізувати інтегровану систему динамічного застосування політик безпеки та виявлення аномалій;

– змоделювати типові сценарії атак на контейнеризовані застосунки й оцінити ефективність запропонованого рішення;

– виконати оцінку залишкових ризиків та сформувати практичні рекомендації щодо впровадження системи захисту.

Об’єкт дослідження – контейнеризовані обчислювальні середовища, що функціонують у Kubernetes-кластерах.

Предмет дослідження – методи, засоби та механізми забезпечення безпеки Kubernetes-кластерів на основі динамічного управління політиками безпеки й моніторингу подій у реальному часі.

Методи дослідження. У роботі використано аналіз технічної документації та наукових джерел, системний аналіз архітектури Kubernetes-кластерів, моделювання атак, експериментальні дослідження параметрів безпеки, методи логування та візуалізації подій, а також порівняльне оцінювання ефективності засобів захисту й аналіз залишкових ризиків.

Наукова новизна одержаних результатів. Удосконалено підхід до захисту Kubernetes-кластерів шляхом інтеграції механізмів динамічного застосування політик безпеки з поведінковим моніторингом і виявленням аномалій у реальному часі. Запропоновано модель взаємодії політик безпеки та системи виявлення аномалій, що забезпечує автоматизоване виявлення небезпечних відхилень у роботі контейнеризованих сервісів та ініціювання коригувальних дій на рівні кластера.

Практичне значення отриманих результатів. Розроблено та апробовано інтегровану систему захисту Kubernetes-кластерів, яку можна застосовувати як типове рішення для підвищення рівня безпеки контейнеризованих застосунків у корпоративних та хмарних інфраструктурах. Описані приклади політик безпеки, правила виявлення аномалій та дашборди в Elastic Stack можуть бути використані й адаптовані фахівцями з кібербезпеки в практичній діяльності.

Публікації та апробація КР.

1. Віталій КЛІМ, Тарас ЦАВОЛИК Архітектура Систем безпеки Kubernetes. Збірник матеріалів науково-практичного симпозіуму «Захист інформації'2025», Тернопіль, 2025. – С 44-45.

2. Віталій КЛІМ, Тарас ЦАВОЛИК Архітектура Систем безпеки Kubernetes : Збірник матеріалів науково-практичного симпозіуму «Технології Інтернету речей: системи та рішення» (ТІР:СТ - 2025), Тернопіль, 2025. .– С 22-23.

РОЗДІЛ 1. АНАЛІЗ СТАНУ БЕЗПЕКИ КОНТЕЙНЕРИЗОВАНИХ СЕРЕДОВИЩ У KUBERNETES

1.1 Основи контейнеризації та архітектура Kubernetes

У сучасному IT-середовищі контейнеризація стала однією з ключових технологій, що змінила підхід до розробки, розгортання та супроводу програмного забезпечення. Її поява та поширення стали відповіддю на потребу у створенні стандартизованих, легковагих і портативних середовищ виконання, які працюють однаково в різних інфраструктурах — від локального середовища розробника до масштабованих хмарних платформ .

Контейнер можна визначити як ізольоване середовище виконання, яке містить застосунок разом з усіма необхідними бібліотеками, конфігураціями та залежностями. На відміну від віртуальних машин, контейнери спільно використовують ядро операційної системи, що робить їх легшими з точки зору споживання ресурсів і швидшими під час запуску. Основними перевагами контейнеризації є висока швидкість розгортання, економія апаратних ресурсів, зменшення накладних витрат, а також підвищення узгодженості між середовищами розробки, тестування та промислової експлуатації.

Історичний розвиток контейнеризації бере початок від механізму chroot в UNIX (1979 рік), який вперше дозволив ізолювати файлову систему процесу. Надалі з’явилися такі рішення, як FreeBSD Jails, Solaris Zones, Linux namespaces і cgroups, OpenVZ та LXC, які поступово вдосконалювали механізми ізоляції процесів і ресурсів. Проривом у масовому впровадженні контейнерів стала поява платформи Docker у 2013 році, яка надала зручний інструментарій для створення, керування та поширення контейнерних образів [1].

Однак використання лише Docker або подібних платформ ускладнює ефективне керування великою кількістю контейнерів у розподілених системах. Виникла потреба в оркестраторах контейнерів, які автоматизують розгортання,

масштабування, оновлення та моніторинг контейнеризованих застосунків. Найбільш поширеним рішенням стала система Kubernetes (K8s) з відкритим кодом, спочатку розроблена компанією Google і передана під управління спільноти Cloud Native Computing Foundation (CNCF) [2].

Kubernetes (K8s) є платформою для автоматизованого керування контейнеризованими робочими навантаженнями та сервісами, яка реалізує декларативну модель керування. Користувач описує бажаний стан системи у конфігураційних файлах (зазвичай у форматі YAML), а Kubernetes за допомогою контрольних циклів (control loops) самостійно приводить поточний стан кластера у відповідність до заданого та підтримує цю відповідність у часі . Платформа забезпечує самовідновлення, автоматичне масштабування, оновлення без простоїв та балансування навантаження.

Архітектура Kubernetes складається з двох основних рівнів: Control Plane та Worker Nodes.

Control Plane відповідає за загальне управління кластером і прийняття рішень щодо його стану. До його основних компонентів належать:

- kube-apiserver — центральна точка доступу до кластера через REST API; усі зовнішні та внутрішні взаємодії з Kubernetes здійснюються через цей компонент;
- etcd — розподілене сховище типу «ключ–значення», у якому зберігаються конфігурація та поточний стан усіх об’єктів кластера;
- kube-scheduler — аналізує доступні ресурси та обмеження, приймає рішення, на яких вузлах розміщувати нові Pod;
- kube-controller-manager — виконує контрольні цикли (контролери), які стежать за тим, щоб фактичний стан об’єктів відповідав описаному у деклараціях;
- cloud-controller-manager — забезпечує інтеграцію з хмарними провайдерами та винесення логіки, специфічної для хмарної інфраструктури, за межі основних контролерів.

Worker Node — це фізичний або віртуальний сервер, який безпосередньо виконує контейнери та надає обчислювальні ресурси. Основні компоненти вузла:

- kubelet — агент на вузлі, який отримує завдання від API-сервера, запускає відповідні Pod і постійно відстежує їхній стан;
- kube-proxy — мережевий проксі, що реалізує правила маршрутизації та мережеві політики для забезпечення доступу до сервісів всередині кластера;
- container runtime — середовище виконання контейнерів, відповідальне за запуск і керування контейнерами.

У сучасних кластерах Kubernetes зазвичай використовуються containerd або CRI-O, тоді як Docker Engine історично відіграв значну роль і нині, як правило, використовується опосередковано через containerd або окремий адаптер.

Базовою одиницею розгортання у Kubernetes є Pod — мінімальна логічна структура, яка може містити один або кілька тісно пов’язаних контейнерів. Контейнери всередині одного Pod спільно використовують мережевий простір, IP-адресу та, за потреби, томи для зберігання даних [5]. Pod-и розміщуються на вузлах і керуються контролерами вищого рівня. Над Pod-ами в ієрархії Kubernetes розташовані такі об’єкти:

1. ReplicaSet — об’єкт, який гарантує підтримку заданої кількості реплік Pod у робочому стані;
2. Deployment — контролер, що надає декларативну модель керування життєвим циклом застосунків, у тому числі оновлення без простоїв та можливість відкату до попередніх версій;
3. Service — абстракція, яка забезпечує стабільну точку доступу (віртуальну IP-адресу та DNS-ім’я) до набору Pod незалежно від їх конкретного розміщення та поточної кількості;
4. Namespace — механізм логічного поділу ресурсів кластера між командами, проєктами або середовищами (наприклад, dev, staging, prod), що спрощує ізоляцію та керування доступом.

Ключовою особливістю роботи Kubernetes є механізм контрольного циклу (reconciliation loop). Контролери постійно спостерігають за фактичним станом об’єктів у кластері та порівнюють його з бажаним станом, описаним у маніфестах. У разі виявлення розбіжностей система автоматично вживає заходів для їх усунення, наприклад перестворює видалений Pod, переміщує робочі навантаження з несправного вузла або масштабує кількість реплік.

Конфігураційні файли Kubernetes зазвичай описуються у форматі YAML і містять параметри бажаного стану: кількість реплік, образи контейнерів, змінні середовища, політики доступу, обмеження ресурсів тощо . Такий підхід дозволяє описувати інфраструктуру як код (Infrastructure as Code), що є одним з ключових принципів сучасних DevOps-практик.

Kubernetes також підтримує розширюваність через Custom Resource Definitions (CRD) — механізм додавання нових типів ресурсів до API Kubernetes. CRD дозволяють розширювати стандартний набір об’єктів, надаючи можливість описувати доменно-специфічні сутності, які керуються за тими самими принципами, що й вбудовані ресурси . У поєднанні з операторами (operator pattern), які інкапсулюють складну логіку керування станом цих ресурсів, CRD забезпечують можливість автоматизації розгортання та супроводу складних систем у кластері, зберігаючи уніфіковану модель взаємодії через Kubernetes API.

Таким чином, Kubernetes надає потужний набір інструментів для управління життєвим циклом контейнеризованих застосунків. Його архітектура базується на принципах модульності, декларативного опису стану, подієво-орієнтованої обробки змін та механізмах самовідновлення. Глибоке розуміння цих основ є передумовою для подальшого аналізу викликів безпеки, які виникають у динамічних контейнерних середовищах Kubernetes-кластерів.

1.2 Безпекові механізми Kubernetes-кластерів

Складність та динамічність Kubernetes-кластерів створюють суттєві виклики у сфері кібербезпеки. Для ефективного захисту інфраструктури необхідно враховувати архітектурні особливості платформи, взаємодію її компонентів, принципи конфігурації, а також сучасні моделі атак. Kubernetes містить низку вбудованих механізмів безпеки, які за умови коректного налаштування дозволяють значно зменшити ризики: автентифікацію та авторизацію доступу до API, ізоляцію на рівні namespace та pod, мережеві політики, керування конфіденційними даними, аудит, контроль параметрів контейнерів, оновлення компонентів та інтеграцію із засобами моніторингу й виявлення аномалій [3].

Kubernetes використовує кілька механізмів автентифікації доступу до API-сервера: клієнтські сертифікати X.509, токени, службові облікові записи, статичні файли токенів, а також інтеграцію з зовнішніми постачальниками ідентичності через OpenID Connect (OIDC), webhook-автентифікацію або аутентифікуючий проксі.

Адміністратор кластера визначає набори користувачів і сервісних облікових записів, які діють від імені застосунків. З кожним запитом до kube-apiserver передається маркер автентифікації і сервер перевіряє його дійсність. Інтеграція з корпоративними системами керування ідентичністю (LDAP, SAML, IdP) реалізується опосередковано — через OIDC-провайдери, webhook чи проксі-рішення, а не як «нативна» функція самого Kubernetes.

Після успішної автентифікації Kubernetes повинен визначити, чи може суб’єкт виконати запитану дію над конкретним ресурсом. Для цього використовуються різні режими авторизації: RBAC (Role-Based Access Control), Node, ABAC (Attribute-Based Access Control) та Webhook. На практиці основним і рекомендованим механізмом є RBAC.

Модель RBAC базується на використанні об’єктів Role та ClusterRole, які визначають множину дозволених дій, а також об’єктів RoleBinding і ClusterRoleBinding, що прив’язують ці ролі до конкретних користувачів, груп або облікових записів ServiceAccount. Наприклад, одна роль може надавати лише права на читання об’єктів Pod у вибраному просторі імен, тоді як інша — дозволяти змінювати об’єкти Deployment у межах усього кластера. Ключові рекомендації:

- застосовувати принцип найменших привілеїв (least privilege);
- розмежовувати ролі адміністраторів, розробників та системних сервісів;
- уникати використання надмірно широких ClusterRole без нагальної потреби;
- по можливості не використовувати ABAC у нових кластерах як застарілий та менш керований підхід.

Для нестандартних сценаріїв може застосовуватися Webhook Authorization, коли Kubernetes делегує прийняття рішень зовнішньому сервісу авторизації. Namespace надає логічну ізоляцію ресурсів кластера: застосунків, ролей, квот, подій, секретів тощо. Це особливо критично для багатокористувацьких середовищ, а також для розділення етапів життєвого циклу (наприклад, dev / staging / prod).

Починаючи з версії 1.25, у Kubernetes стабільно підтримується механізм Pod Security Admission, який замінив PodSecurityPolicy, що був остаточно вилучений з ядра платформи [4]. PSA реалізований як admission controller і забезпечує застосування Pod Security Standards на рівні namespace. Стандарт визначає три профілі безпеки:

- Privileged — практично без обмежень; застосовується лише для довірених системних робочих навантажень;
- Baseline — блокує очевидно небезпечні практики, але дозволяє поширені сценарії розгортання;

- Restricted — максимально жорсткий профіль, який вимагає запуску контейнерів від не-root користувачів, мінімального набору привілеїв, обмежених монтованих томів тощо.

Політика задається через мітки pod-security.kubernetes.io/mode: level на namespace, де mode — enforce, audit або warn, a level — privileged, baseline або restricted. Pod, який не відповідає встановленій політиці для режиму enforce, буде відхилено. Це дозволяє централізовано застосовувати базові вимоги до безпеки на рівні кластерної політики.

За замовчуванням мережна модель Kubernetes дозволяє Pod-ам вільно встановлювати з'єднання один з одним. Об'єкт NetworkPolicy дозволяє явно визначати, який трафік дозволено між Pod-ами та зовнішніми мережами, використовуючи селектори міток, IP-діапазони, порти та протоколи.

Застосування NetworkPolicy потребує CNI-плагіна, який підтримує їх реалізацію (Calico, Cilium, Weave Net, Antrea тощо). Типові рекомендації:

- проєктувати політики за принципом «deny by default», відкриваючи лише необхідні напрямки трафіку;
- використовувати правила ingress/egress для контролю як вхідних, так і вихідних з'єднань;
- уникати використання динамічно змінних міток для визначення критичних політик доступу;
- комбінувати мережеві політики із сегментацією за namespace.

Мережеві політики дозволяють суттєво зменшити поверхню атаки, обмеживши бічний рух (lateral movement) всередині кластера.

Об'єкти Secret призначені для зберігання конфіденційних даних: паролів, токенів, ключів API, TLS-сертифікатів тощо. За замовчуванням значення Secret кодуються у base64 та зберігаються в etcd без шифрування, тому компрометація сховища etcd означає компрометацію секретів.

Рекомендовані практики:

- увімкнути шифрування секретів «на спокої» (at rest) за допомогою EncryptionConfiguration, обравши сучасні стійкі алгоритми шифрування ;
- жорстко обмежити доступ до операцій читання Secret через RBAC;
- по можливості уникати передачі секретів як змінних середовища, віддаючи перевагу монтованим томам (volume) із файлами-секретами;
- інтегрувати Kubernetes з зовнішніми системами керування секретами (HashiCorp Vault, AWS Secrets Manager, Azure Key Vault тощо), якщо потрібне централізоване керування та ротація ключів.

Компонент kube-apiserver є центральною і критичною ланкою архітектури Kubernetes, оскільки саме через нього проходять усі запити до кластера та здійснюється доступ до його ресурсів. Захист цього компонента забезпечується застосуванням TLS-сертифікатів для шифрування трафіку й автентифікації сторін, жорстким обмеженням мережевого доступу до API за допомогою VPN, міжмережевих екранів і мережевих списків контролю доступу з розмежуванням зовнішніх і внутрішніх точок входу, увімкненням та коректним налаштуванням журналювання аудиту (audit logging) для фіксації всіх викликів API, вимкненням або істотним обмеженням застарілих і непотрібних admission-плагінів та анонімного доступу, а також суворим контролем доступу до kubeconfig-файлів і сертифікатів адміністраторів. Некоректна експозиція API-сервера у зовнішні мережі без належних механізмів автентифікації та авторизації є однією з найпоширеніших причин компрометації Kubernetes-кластерів.

Параметри securityContext, які задаються як для Pod, так і для окремих контейнерів, дають змогу визначати умови та рівень привілеїв, з якими запускаються процеси всередині контейнеризованого середовища. Зокрема, за їх допомогою можна примусово виконувати процеси від імені користувача, що не має прав суперкористувача, обмежувати набір системних можливостей Linux до необхідного мінімуму, у випадках, коли це допустимо, використовувати кореневу

файлову систему лише для читання, застосовувати профілі seccomp для звуження переліку дозволених системних викликів, а також забороняти запуск контейнерів у привілейованому режимі. Сукупність таких налаштувань суттєво зменшує потенційні наслідки експлуатації вразливостей ядра операційної системи й спроб виходу за межі контейнера та логічно доповнює вимоги політик Pod Security Admission і стандартів Pod Security Standards.

Kubernetes має вбудований механізм audit logging, який фіксує запити до API-сервера відповідно до заданої політики аудиту. Audit policy визначає, які події потрібно логувати, а також рівень деталізації.

Журнали аудиту відіграють ключову роль у забезпеченні безпеки Kubernetes-кластера, оскільки саме на їх основі здійснюється розслідування інцидентів, аналізуються дії адміністраторів та сервісних облікових записів, підтверджується виконання вимог стандартів інформаційної безпеки, таких як ISO 27001 чи PCI DSS, а також виявляється підозріла активність під час інтеграції з SIEM-платформами для реагування на подію.

Рівень захищеності кластера значною мірою визначається актуальністю його компонентів і правильністю налаштувань. До цього належать регулярне оновлення kube-apiserver, kubelet, etcd, мережевих і сховищних плагінів, а також контролерів ingress, розмежування вузлів керування та вузлів із робочими навантаженнями, періодичне використання інструментів на зразок kube-bench для перевірки відповідності рекомендаціям CIS Kubernetes Benchmark, а також інтеграція автоматизованих перевірок безпеки й сканування конфігурацій у процеси CI/CD. Такі заходи дають змогу своєчасно виявляти небезпечні або некоректні налаштування та зменшувати імовірність успішної атаки.

Kubernetes забезпечує можливість інтеграції з інструментами runtime security та моніторингу, які аналізують поведінку контейнерів і компонентів кластера в реальному часі. До таких засобів належать:

- Falco — open source рушій для виявлення аномальної поведінки в контейнеризованих середовищах; аналізує системні виклики, події ядра та події Kubernetes і дозволяє описувати правила виявлення підозрілої активності [16];
- інструменти, що використовують eBPF, для глибокого спостереження за мережевими та системними подіями з мінімальним оверхедом;
- стек моніторингу на базі Prometheus і Alertmanager, який дозволяє відстежувати ключові метрики (частота перезапусків Pod, сплески використання ресурсів, помилки в застосунках) та надсилати сповіщення в Slack, email чи інші системи оповіщення.

Поєднання платформних механізмів безпеки Kubernetes (RBAC, PSA, NetworkPolicy, Secrets, audit logging) з інструментами runtime-аналізу та моніторингу формує багаторівневу модель захисту кластера.

1.3 Типові вразливості та поверхня атаки у контейнерних середовищах

Контейнеризовані середовища, попри численні переваги, формують складну поверхню атаки. Потенційними цілями зловмисників є контейнерні образи, запущені контейнери та Pod-и, мережеві з’єднання, Kubernetes API, компоненти Control Plane, вузли кластера, а також елементи ланцюга постачання програмного забезпечення. Помилки конфігурації на будь-якому з цих рівнів можуть призвести до ескалації прав, бічного руху всередині кластера, витоку даних або повної компрометації інфраструктури.

У динамічному Kubernetes-середовищі поверхня атаки суттєво залежить від параметрів конфігурації. Типовими точками входу для атак є:

- вразливі або неконтрольовані контейнерні образи;
- контейнери з надмірними привілеями чи доступом до ресурсів хоста;
- відсутність або некоректна реалізація мережевої ізоляції;
- відкритий або слабо захищений доступ до kube-apiserver;

- слабкі механізми автентифікації та авторизації;
- надмірні права CI/CD-пайплайнів та сервісних облікових записів.

У сукупності ці фактори розширюють можливості зловмисника — від локальної компрометації окремого контейнера до доступу до Control Plane та даних усієї організації.

Контейнери, побудовані на основі певних базових образів, автоматично успадковують усі притаманні цим образам вразливості [5]. На практиці це проявляється у використанні застарілих пакетів і бібліотек з відомими вразливостями (CVE), наявності в образі зайвих утиліт загального призначення (наприклад, bash, curl, пакетних менеджерів), які можуть бути використані зловмисником після компрометації, зберіганні всередині образу облікових даних, ключів доступу або конфігураційних файлів із секретною інформацією, а також у запуску застосунків від імені користувача root без додаткових обмежень. Для зменшення таких ризиків доцільно застосовувати мінімалістичні базові образи, фіксувати конкретні версії образів замість використання тегу latest, регулярно виконувати їх сканування на наявність вразливостей за допомогою спеціалізованих інструментів, таких як Trivy, а також аналізувати склад образів за допомогою програмних відомостей про компоненти.

Надмірне надання прав контейнерам належить до найбільш критичних загроз для безпеки кластерного середовища. Якщо контейнер запускається в привілейованому режимі або отримує прямий доступ до важливих каталогів вузла, зокрема /var/run/docker.sock чи системних директорій, процеси всередині нього фактично наближаються за можливостями до процесів, що виконуються безпосередньо на хості. У такій ситуації контейнер здатен читати й змінювати файлову систему вузла, втручатися в роботу мережевих інтерфейсів та правил, керувати контейнерним рантаймом і запускати додаткові контейнери в обхід визначених політик безпеки, а також отримувати доступ до локальних сокетів, через які можна впливати на kubelet та інші сервісні компоненти. Зменшення

зазначених ризиків досягається завдяки коректному налаштуванню параметрів securityContext із повною відмовою від привілейованого режиму, жорстким обмеженням набору Linux capabilities, запуском процесів від імені користувачів без прав суперкористувача та, де це прийнятно, використанням кореневої файлової системи лише для читання у поєднанні із застосуванням профілів Pod Security Standards рівнів Baseline та Restricted через механізм Pod Security Admission.

Чимало інцидентів у Kubernetes пов’язані не стільки з експлойтами в програмному коді, скільки з некоректними або занадто слабким налаштуваннями кластера. До типових прикладів належать відсутність або надто прості NetworkPolicy, які дозволяють Pod-ам безконтрольно обмінюватися трафіком, надмірно широкі ролі та прив’язки в RBAC, відкритий доступ до kube-apiserver з публічних мереж без обмеження джерел трафіку та використання багатофакторної автентифікації, зберігання Secret-об’єктів у etcd без шифрування та слабкий контроль їх читання через RBAC, а також сервіси типу NodePort або LoadBalancer, доступні з Інтернету без додаткових рівнів захисту. Для виявлення подібних помилок конфігурації та зіставлення їх із рекомендаціями CIS Kubernetes Benchmark доцільно застосовувати інструменти на зразок kube-bench, kube-hunter, а також спеціалізовані платформи безпеки для комплексного аналізу стану кластера.

Сучасні атаки часто націлені не лише на запущений кластер, а й на ланцюг постачання програмного забезпечення. Для контейнеризованих середовищ це означає ризики на етапах побудови образів, роботи CI/CD-пайплайнів та розгортання у кластері :

- зберігання облікових даних до реєстрів образів, Kubernetes-кластера або хмарних сервісів у відкритому вигляді в CI-конфігураціях;
- відсутність перевірки образів (signing, валідація підписів, сканування на CVE) перед розгортанням;
- сервісні облікові записи, які мають надмірні дозволи в кластері.

Рекомендованими заходами є запровадження підходу policy-as-code для контролю маніфестів та пайплайнів, використання інструментів для перевірки образів (Trivy, cosign тощо), розподіл середовищ (dev/staging/prod) та обмеження прав доступу CI/CD-систем до кластера за принципом найменших привілеїв.

Типова відкрита мережева модель Kubernetes робить обмін трафіком всередині кластера простим, але водночас відкриває додаткові можливості для атак. Якщо мережеві обмеження не налаштовані, зловмисник може переміщуватися між сервісами всередині кластера, перехоплювати трафік між компонентами (атаки типу Man-in-the-Middle), сканувати внутрішні сервіси та користуватися службами, що працюють без автентифікації, наприклад Redis або внутрішні HTTP API.

Щоб зменшити ці ризики, у кластері задають NetworkPolicy і чітко визначають, які Pod-и можуть обмінюватися трафіком між собою та ззовні, вмикають шифрування з використанням TLS або mTLS для взаємодії між сервісами, а за потреби впроваджують service mesh, який бере на себе шифрування трафіку, перевірку взаємодії між сервісами та централізоване керування правилами доступу.

Урахування таких загроз ще на етапі проєктування дає змогу одразу закладати потрібні механізми захисту. Практичний підхід до безпеки Kubernetes поєднує використання перевірених контейнерних образів, мінімально необхідні привілеї для процесів, жорсткі обмеження доступу, захищені мережеві з’єднання між компонентами та контроль усього шляху потрапляння застосунку в кластер, що дозволяє будувати Kubernetes-кластери, стійкі до реальних атак [6].

1.4 Порівняння безпеки віртуалізації та контейнеризації

Контейнеризація та віртуалізація є двома поширеними підходами до ізоляції середовищ виконання застосунків. Обидві технології забезпечують розподіл

ресурсів і гнучке керування інфраструктурою, однак мають суттєві відмінності, які впливають на модель загроз, рівень ізоляції та підходи до захисту[6].

У разі віртуалізації кожна віртуальна машина працює на власній гостьовій операційній системі, ізольованій від інших. Гіпервізор віртуалізує апаратні ресурси та формує чітку межу між віртуальною машиною і хостом. Така модель забезпечує високий рівень ізоляції, оскільки процеси в одній VM не мають прямого доступу до пам’яті та ядра інших віртуальних машин чи хостової системи.

У контейнеризації всі контейнери спільно використовують ядро хостової операційної системи. Ізоляція досягається за допомогою механізмів Linux — namespaces, cgroups, а також модулів безпеки на прикладі AppArmor та SELinux . Хоча за умови коректної конфігурації вони забезпечують достатньо сильну логічну ізоляцію, спільне використання ядра робить контейнери потенційно більш вразливими до атак типу container breakout, коли зловмисник отримує доступ до хост-системи через вразливість у ядрі або помилки конфігурації.

У разі компрометації окремої VM можливості зловмисника зазвичай обмежуються цією ВМ: для переходу до інших ВМ або до гіпервізора необхідно експлуатувати окремі вразливості гіпервізора, які зустрічаються значно рідше та, як правило, оперативно виправляються. Таким чином, гіпервізор виступає потужною, але відносно малою за площею, «лінією оборони».

У контейнерних середовищах неправильна конфігурація суттєво підвищує ризик переходу з контейнера на хост. У разі успішного container breakout зловмисник отримує доступ до всієї вузлової системи, а через неї — до інших контейнерів, спільних томів і, потенційно, до компонентів кластера.

У класичній віртуалізації основними елементами поверхні атаки є гіпервізор, інтерфейси керування та гостьові ОС. Взаємодія між VM і хостом обмежена, а стандартні моделі розгортання зазвичай формують відносно стабільну конфігурацію.

У середовищах, де використовуються контейнери, можливостей для атаки зазвичай більше, ніж у класичній віртуалізації. До загальної інфраструктури додаються додаткові елементи, такі як реєстри образів, платформи CI/CD, а також компоненти керування кластером Kubernetes, зокрема API server, kubelet та різні контролери. Часто контейнери будують на основі публічних базових образів, і це збільшує ризик того, що в систему ще на етапі ланцюга постачання потраплять уже вразливі пакети. Крім того, у кластері існує велика кількість об’єктів, наприклад Pod, Deployment, Service, Job, і численні зв’язки між ними, що підвищує ймовірність помилок у конфігурації.

У моделях на базі віртуальних машин критично важливим елементом є гіпервізор, і його компрометація може фактично вивести з ладу все середовище. У випадку контейнеризації подібну роль відіграє ядро операційної системи та компоненти, які відповідають за роботу контейнерів і кластера, тому їхній захист стає ключовим завданням.

Контейнеризація передбачає декларативне керування станом та автоматизацію життєвого циклу робочих навантажень. У разі інциденту можна швидко:

- перезапустити або переназначити Pod на іншому вузлі;
- відкотити застосунок до попередньої версії;
- замінити уразливий образ на оновлений, не змінюючи цілісно інфраструктуру.

Це суттєво скорочує час відновлення і дозволяє оперативно реагувати на виявлені вразливості.

Віртуальні машини зазвичай повільніше створюються, копіюються та переносяться, що ускладнює масове відновлення після інциденту. Водночас такий підхід краще підходить для повної ізоляції чутливих або легасі-систем, які не можуть бути адаптовані до контейнерного формату.

У віртуалізованих середовищах аудит переважно фокусується на подіях усередині гостьових ОС та на подіях управління віртуальною інфраструктурою. Деталізація подій, пов’язаних із мережею та міжсервісною взаємодією, часто потребує додаткових рішень.

Контейнерні середовища, зокрема Kubernetes, надають розширені можливості аудиту: audit logging дозволяє фіксувати кожен запит до API-сервера, зміни конфігураційних об’єктів і дії контролерів. У поєднанні з подіями кластера (events), логами контейнерів і метриками це створює детальну картину стану середовища, яка може використовуватися в DevSecOps-процесах для виявлення аномалій та інцидентів у реальному часі.

У контейнерних середовищах питання довіри до базових образів і всього ланцюга постачання програмного забезпечення є особливо важливим. Уразливості можуть потрапити в промислове середовище через використання неперевірених публічних образів, відсутність регулярного сканування образів на вразливості, слабкий контроль змін у CI/CD-процесах, а також через нехтування підписуванням образів і перевіркою підписів під час розгортання.

Сучасні підходи до безпеки контейнеризації передбачають обов’язкове сканування образів, використання мінімалістичних і попередньо посилених базових образів, застосування policy-as-code для контролю розгортання та перевірку підписів контейнерних образів. У класичній віртуалізації також використовують підготовлені шаблони віртуальних машин, однак ланцюг постачання там зазвичай менш динамічний і змінюється рідше. Порівняння основних аспектів безпеки віртуалізації та контейнеризації подано в таблиці 1.1.

Таблиця 1.1 – Порівняння безпекових характеристик віртуалізації та контейнеризації.

Аспект	Віртуалізація	Контейнеризація
Рівень ізоляції	Високий рівень ізоляції завдяки гіпервізору та окремим гостьовим ОС	Логічна ізоляція на рівні ядра ОС та механізмів Linux (namespaces, cgroups); спільне ядро для всіх контейнерів
Поверхня атаки	Відносно менша: гіпервізор, інтерфейси керування, гостьові ОС	Ширша: реєстри образів, CI/CD, Kubernetes control plane, численні API та об’єкти кластера
Відновлення після інциденту	Відновлення повільніше; створення та міграція ВМ потребують більше часу	Швидке та автоматизоване перерозгортання контейнерів, відкат версій, перезапуск Pod-ів
Аудит та моніторинг	Логування переважно на рівні гостьових ОС і платформи віртуалізації	Розширений аудит: журнали викликів API, події кластера, метрики, інтеграція з DevSecOps-процесами
Ризик при компрометації	Компрометація однієї ВМ здебільшого локалізована (за винятком рідкісних експлойтів гіпервізора)	Компрометація привілейованого контейнера може призвести до захоплення вузла й подальшої ескалації в усьому кластері
Типові сценарії використання	Ізоляція чутливих / легасі-систем, випадки, коли потрібна «жорстка» ізоляція	Масштабовані мікросервіси, динамічні робочі навантаження, DevOps/DevSecOps-процеси

Таким чином, віртуалізація забезпечує сильнішу ізоляцію та краще підходить для розміщення особливо чутливих або легасі-систем, тоді як контейнеризація надає вищу гнучкість, швидкість і масштабованість, але вимагає ретельної конфігурації, формалізованих політик безпеки та постійного моніторингу . У наступних підрозділах буде проаналізовано сучасні стандарти й практики, які дозволяють підвищити захищеність Kubernetes-середовищ і частково компенсувати вищу поверхню атаки контейнерних платформ.

1.5 Сучасні стандарти безпеки: CIS Benchmark, NIST 800-190

Забезпечення безпеки контейнеризованих середовищ неможливе без орієнтації на загальновизнані стандарти та рекомендації. Вони слугують основою для оцінювання поточного стану безпеки, побудови політик та вдосконалення процедур захисту у Kubernetes-кластерах і контейнерних платформах. Серед найбільш авторитетних документів у цій сфері виділяють CIS Kubernetes Benchmark та NIST SP 800-190 “Application Container Security Guide” [7].

CIS Kubernetes Benchmark розроблений Center for Internet Security (CIS) як набір детальних рекомендацій щодо безпечної конфігурації Kubernetes. Бенчмарк випускається у версіях, прив’язаних до конкретних версій Kubernetes, і містить конкретні перевірки для компонентів керування (control plane), сховища etcd, вузлів кластера, а також політик автентифікації, авторизації, журналювання та мережевої безпеки.

Кожна рекомендація в CIS Benchmark містить опис, обґрунтування, рівень важливості (наприклад, Level 1 / Level 2, іноді з позначеннями High/Medium/Low у супровідних матеріалах), інструкції для перевірки (Audit) та кроки для виправлення (Remediation) . Наприклад, перевірка параметра --anonymous-auth=false для kube-apiserver гарантує, що анонімні запити до API не будуть прийматися без автентифікації.

Для автоматизованої оцінки відповідності кластерів рекомендаціям CIS широко використовується інструмент kube-bench від Aqua Security. Він запускає набір тестів, що відповідають певній версії CIS Kubernetes Benchmark, аналізує конфігурацію компонентів і формує звіт про відповідність . Це дозволяє виявляти критичні конфігураційні вразливості та відстежувати прогрес hardening-заходів.

Документ NIST SP 800-190 “Application Container Security Guide”, опублікований Національним інститутом стандартів і технологій США (NIST), має

ширший фокус: він описує загальні принципи безпеки контейнеризованих застосунків незалежно від конкретної платформи оркестрації [5].

У NIST SP 800-190 розглядаються:

- типові загрози й ризики, властиві контейнеризації;
- захист середовища виконання (runtime) і хостової операційної системи;
- безпечне створення та доставка образів (image provenance), включно з процесами побудови та публікації образів;
- управління секретами та конфігурацією;
- моніторинг, виявлення аномалій і реагування на інциденти .

NIST виокремлює ключові елементи екосистеми контейнерів, такі як: образи, реєстри, оркестратори, середовище виконання, хост-ОС і мережа. Для кожного з них описуються типові вразливості та рекомендовані контрзаходи — від жорсткості конфігурацій (hardening) до процесних і організаційних заходів.

Окрему увагу в NIST SP 800-190 приділено ланцюгу постачання ПЗ. Документ рекомендує:

- забезпечувати image provenance — простежуваність походження образів;
- використовувати цифровий підпис образів контейнерів і валідацію цілісності під час завантаження та запуску;
- виконувати систематичне сканування образів на наявність відомих вразливостей;
- впроваджувати механізми policy enforcement у пайплайнах CI/CD, які блокують розгортання образів, що не відповідають політикам безпеки;
- застосовувати безперервний моніторинг та збір подій безпеки для аналізу та реагування.

Хоча обидві ініціативи стосуються безпеки контейнерних середовищ, їх фокус відрізняється (табл. 1.2).

Таблиця 1.2 – Порівняння CIS Kubernetes Benchmark та NIST SP 800-190

Критерій	CIS Kubernetes Benchmark	NIST SP 800-190
Сфера застосування	Безпечна конфігурація кластерів Kubernetes	Контейнеризовані середовища та платформи оркестрації (незалежно від конкретної технології)
Основний фокус	Параметри компонентів платформи, конфігурація кластера, заходи з підвищення стійкості (hardening)	Увесь життєвий цикл контейнерів, організація процесів і контрзаходи безпеки
Рівень деталізації	Високий: конкретні технічні дії, параметри, флаги та команди для перевірки	Середній: акцент на процесах, політиках і управлінні ризиками
Тип рекомендацій	Переважно технічні настанови щодо конфігурацій, контролю доступу та розмежування прав	Поєднання технічних, організаційних і процедурних рекомендацій
Інструменти для перевірки	Наявні спеціалізовані інструменти оцінювання (зокрема kube-bench)	Відсутні «офіційні» інструменти; документ визначає вимоги та принципи

Узагальнюючи, CIS Kubernetes Benchmark можна розглядати як детальний чек-ліст технічної конфігурації кластера, тоді як NIST SP 800-190 задає ширший

процесний контекст і вимоги до організації безпеки контейнерних робочих навантажень у межах життєвого циклу.

На практиці рекомендації CIS Kubernetes Benchmark і NIST SP 800-190 використовують разом, оскільки вони закривають різні, але пов’язані аспекти безпеки. CIS Kubernetes Benchmark зосереджується на тому, як налаштувати компоненти Kubernetes, щоб зменшити ризик типових конфігураційних помилок, посилити захист control plane, вузлів і механізмів доступу. Документ NIST SP 800-190, своєю чергою, описує, як побудувати цілісний процес безпечної роботи з контейнерами: від розробки та збирання образів до їх розгортання, моніторингу та реагування на інциденти в рамках DevOps чи DevSecOps.

Якщо впроваджувати обидва підходи разом, це дає змогу системно перевіряти й виправляти конфігурації, демонструвати відповідність зовнішнім вимогам на кшталт ISO 27001, SOC 2 або вимог окремих замовників, чітко визначати політики безпеки, ролі та зони відповідальності в командах DevOps і SRE, а також вбудовувати автоматичні перевірки конфігурацій та контейнерних образів у CI/CD-процеси за допомогою інструментів типу kube-bench, сканерів вразливостей і систем контролю політик.

1.6 Аналіз реальних інцидентів безпеки у Kubernetes

Попри розвинену екосистему інструментів захисту та активну підтримку спільноти, Kubernetes залишається привабливою ціллю для атак. У більшості зафіксованих інцидентів ключову роль відіграють помилки конфігурації, недостатні механізми контролю доступу, відсутність належної мережевої сегментації та використання вразливих або недовірених образів . Аналіз реальних випадків дає змогу виокремити типові вектори атак і сформувати вимоги до побудови безпечної архітектури Kubernetes-кластерів.

У 2018 році в середовищі Tesla було виявлено несанкціоноване розгортання контейнерів для майнінгу криптовалют[8]. Первинний доступ до інфраструктури зловмисники отримали через публічно доступний інтерфейс Kubernetes Dashboard, який не вимагав автентифікації [38 – 40]. Після цього було здійснено доступ до сховища etcd, де зберігалися облікові дані хмарної інфраструктури, та розгорнуто контейнер із майнером.

Для ускладнення виявлення використовувалися приватні майнінгові пули, маскування трафіку та контрольоване споживання ресурсів, що не створювало очевидних аномалій у навантаженні.

З погляду безпеки цей випадок демонструє критичну важливість:

- обмеження доступу до адміністративних інтерфейсів (Dashboard, API-сервер) та відмови від анонімного доступу;
- застосування механізмів RBAC і мережевих політик для розмежування доступу до ключових компонентів (зокрема etcd);
- впровадження механізмів поведінкового моніторингу для виявлення нетипових обчислювальних та мережевих патернів.

В одному з публічно описаних випадків, пов’язаних із хмарною інфраструктурою Shopify, дослідники продемонстрували можливість отримання розширеного доступу до Kubernetes Secrets через поєднання серверної вразливості (Server-Side Request Forgery, SSRF) та надмірних прав сервісних облікових записів і хмарних IAM-ролей.

Наявність прав на читання секретів у кількох namespaces створювала потенційну можливість витоку конфіденційних даних у разі компрометації одного сервісу.

Цей інцидент підкреслює необхідність:

- послідовного застосування принципу найменших привілеїв як у RBAC-політиках Kubernetes, так і в IAM-конфігураціях хмарних провайдерів;

- регулярного аудиту ролей і прив’язок на предмет надмірного доступу до Secret-об’єктів та інших критичних ресурсів;
- використання шифрування секретів та сегментації доступу до них на рівні namespaces.

Інцидент, відомий як Azurescape, стосувався керованого сервісу Azure Container Instances (ACI). Дослідники Palo Alto Networks (Unit 42) виявили можливість виходу з контейнера за межі ізоляційного середовища та отримання токенів привілейованого Kubernetes service account, що потенційно дозволяло впливати на інші контейнери в багатотенантному середовищі.

Ця уразливість була наслідком комбінації помилок у внутрішній реалізації сервісу та конфігурації привілейованих облікових записів. Вона показує, що:

- багатотенантні контейнерні платформи потребують особливо суворих вимог до ізоляції;
- сервісні облікові записи мають бути максимально обмежені за обсягом повноважень;
- навіть у керованих хмарних сервісах необхідний додатковий контроль з боку користувача (журнали, політики, моніторинг спроб перетину меж між тенантами).

Резонансний інцидент у компанії Capital One у 2019 році став прикладом успішної експлуатації ланцюга вразливостей: використання SSRF у веб-додатку, доступ до AWS Instance Metadata Service (IMDS), отримання тимчасових IAM-креденшалів і подальший доступ до S3-бакетів з конфіденційними даними.

У цьому випадку частина сервісів працювала в контейнерах, а слабкий поділ мережі та відсутність жорстких обмежень доступу до сервісу метаданих IMDS додатково посилили наслідки атаки. Це показує, що в подібних середовищах потрібно чітко обмежувати доступ до IMDS (зокрема застосовувати IMDSv2, мережеві фільтри та проксування запитів), уважно проєктувати IAM-ролі за принципом мінімально необхідних прав, а також впроваджувати захист від SSRF,

зокрема валідацію адрес у запитах, контроль вихідного трафіку та логічну сегментацію внутрішніх службю.

Низка досліджень у 2020–2024 роках виявила значну кількість шкідливих контейнерних образів у публічних реєстрах (зокрема Docker Hub), що містили криптомайнери, бекдори та інший шкідливий код . Частина таких образів маскувалася під популярні базові образи, а деякі містили шкідливу логіку у нижніх шарах, що ускладнювало її виявлення.

Додатково, окремі supply-chain інциденти (наприклад, пов’язані з уразливостями в бібліотеках чи системних компонентах) демонструють, що небажаний код може потрапляти в контейнерні образи ще на етапі побудови.

Ці випадки вказують на необхідність:

- використання приватних або керованих реєстрів образів та перевірених офіційних базових образів;
- обов’язкового сканування образів на вразливості та шкідливу активність перед розгортанням;
- застосування цифрових підписів та політик admission control, які блокують непідписані, недовірені або несертифіковані образи;
- постійного моніторингу контейнерних середовищ на предмет нетипових мережевих з’єднань і ознак криптомайнінгу.

Узагальнення розглянутих інцидентів показує, що найчастіше до успішних атак призводить набір повторюваних чинників ризику. Серед них відкриті або недостатньо захищені інтерфейси керування, зокрема Kubernetes Dashboard, API-сервер і служби метаданих хмарних інстансів, помилки, застосування вразливих або недовірених контейнерних образів з публічних реєстрів без належного сканування та перевірки походження, а також слабкий рівень спостереження за подіями, коли журнали аудиту, події безпеки й телеметрія або не збираються, або не аналізуються автоматизованими засобами для виявлення аномалій.

З огляду на це підвищення стійкості Kubernetes-середовищ до атак передбачає впровадження багаторівневої автентифікації та чітко визначених політик доступу на основі RBAC, використання мережевої сегментації та NetworkPolicy з обмеженням доступу до хмарних сервісів метаданих, максимальне скорочення привілеїв Pod-ів і контейнерів із забороною привілейованого режиму та контролем параметрів securityContext, застосування лише перевірених, підписаних і регулярно просканованих контейнерних образів з інтеграцією контролю ланцюга постачання у CI/CD-процеси, розгортання систем журналювання та поведінкового моніторингу, таких як Falco, інструменти на основі eBPF і рішення класу SIEM з кореляцією подій, а також періодичну оцінку конфігурації кластерів відповідно до рекомендацій CIS Kubernetes Benchmark і NIST SP 800-190.

Таким чином, аналіз реальних інцидентів демонструє, що більшість компрометацій у Kubernetes зумовлені не відсутністю засобів безпеки як таких, а їх неповним або некоректним застосуванням. Формалізація вимог, регулярний аудит конфігурацій та інтеграція контролів безпеки в життєвий цикл контейнеризованих застосунків є ключовими умовами побудови надійної системи захисту Kubernetes-кластерів.

РОЗДІЛ 2. ДИНАМІЧНЕ ЗАСТОСУВАННЯ ПОЛІТИК БЕЗПЕКИ У KUBERNETES

2.1 Моделі контролю доступу в Kubernetes: RBAC, ABAC, Pod Security Admission

Класичні моделі контролю доступу – дискреційна (DAC), мандатна (MAC), рольова (RBAC) та атрибутивна (ABAC) – описують різні підходи до того, як співвідносяться суб’єкти (користувачі, процеси), об’єкти (ресурси) та дозволені дії. У Kubernetes вони проявляються не як абстрактні теоретичні конструкції, а як конкретні механізми авторизації та контролю конфігурації подів: RBAC використовується для регулювання доступу до API, ABAC застосовувався як альтернативний авторизатор, а Pod Security Admission реалізує політики безпеки на рівні опису подів [9]. Саме поєднання цих механізмів визначає, наскільки гнучко кластер може адаптувати політики до змін середовища, навантажень та загроз.

Рольовий контроль доступу (RBAC) у Kubernetes базується на чіткій декомпозиції: роль описує, що дозволено робити, а прив’язка ролі – кому саме надаються ці дозволи. На практиці це означає, що адміністратор оперує не окремими «ручними» дозволами для кожного користувача, а наборами правил у вигляді об’єктів Role/ClusterRole та їх прив’язок RoleBinding/ClusterRoleBinding [10].

На відміну від статичних ACL, RBAC у Kubernetes є повністю декларативним: політики доступу описуються у вигляді маніфестів і зберігаються в etcd поряд з іншими об’єктами кластера. Це дозволяє:

- версіонувати конфігурацію прав доступу в системах керування версіями (Git);
- застосовувати принцип policy as code, коли зміни політик проходять той самий життєвий цикл, що й зміни застосунків (рев’ю, тестування, розгортання);

- інтегрувати керування доступом у GitOps-процеси, при яких реальний стан кластера автоматично приводиться у відповідність до стану, зафіксованого в репозиторії.

З погляду динамічного застосування політик безпеки RBAC відіграє подвійну роль. По-перше, він визначає, хто має право змінювати самі політики, включно з налаштуваннями Pod Security Admission, мережевих політик та секретів. По-друге, за рахунок декларативності й інтеграції в CI/CD він дозволяє адаптувати рівень доступу до поточних потреб: тимчасово розширювати права для операційного втручання, застосовувати обмежувальні політики у відповідь на інцидент, сегментувати доступ між командами та середовищами.

З точки зору подальших розділів, RBAC задає «рамки дозволеного» для всіх інструментів динамічного моніторингу та реагування. Наприклад, механізм виявлення аномалій може працювати від імені сервісного облікового запису, що має право лише читати журнали та генерувати події, але не може змінювати конфігурацію кластера без явного погодження політики.

Атрибутивна модель контролю доступу (ABAC) у загальному вважається більш гнучкою, ніж RBAC, оскільки рішення про доступ приймається на основі набору атрибутів і правил, що їх комбінують [11]. Теоретично це дозволяє задавати політики на зразок: «дозволити змінювати конфігурацію лише розробникам сервісу X у робочий час із корпоративної мережі» – без необхідності створювати окремі ролі для кожної умову доступу.

Реалізація ABAC у Kubernetes була виконана через окремий авторизатор API-сервера, який використовує JSON-файл політик. Кожен запис у цьому файлі описує умови, за яких певному обліковому запису дозволяється виконувати дію над ресурсом. Однак у реальних кластерах така реалізація виявилася малозручною з кількох причин:

- файл політик є статичним: його зміна потребує перезапуску API-сервера, що суперечить ідеям безперервної доставки та ускладнює динамічне керування доступом;
- зі зростанням кількості користувачів, сервісних облікових записів та сервісів політики стають громіздкими і важко перевіряються на коректність їх написання;
- відсутня інтеграція з декларативною моделлю Kubernetes: політики ABAC не є звичайними об’єктами кластера, їх важко версіонувати та застосовувати через маніфести.

У підсумку в офіційній документації Kubernetes механізм ABAC вважається застарілим і не рекомендується для нових розгортань, тоді як основна увага приділяється RBAC. При цьому концепції ABAC наразі здебільшого впроваджуються через зовнішні механізми політик, наприклад Open Policy Agent (OPA), або через власні webhook-авторизатори, які взаємодіють з RBAC та admission-контролерами (таблиця 2.1).

Таблиця 2.1 – Порівняння RBAC і ABAC

RBAC	ABAC
Рольова модель	Атрибутивна модель
Статичні дозволи	Деталізований контроль доступу
Менш гнучкий	Дуже гнучкий
Важко працює з великою кількістю ролей	Краще масштабується у великих організаціях
Простіше у впровадженні	Складніше у впровадженні

Pod Security Admission (PSA) не є моделлю контролю доступу в класичному розумінні, однак він реалізує політики допустимих конфігурацій для подів і фактично працює як спеціалізований механізм авторизації для опису робочих

навантажень. На відміну від RBAC, який відповідає на запитання «хто може виконати операцію », PSA відповідає на запитання «чи допустима з погляду безпеки конфігурація пода, який хочуть створити».

Pod Security Admission ґрунтується на стандартизованих профілях Pod Security Standards (PSS) – privileged, baseline та restricted, що задають різні рівні жорсткості обмежень [14]. Політика прив’язується до простору імен через використання міток, тож перехід між профілями зводиться до зміни невеликої кількості декларативних параметрів. Це створює умови для гнучкого й динамічного керування безпекою:

- у середовищах розробки доцільно використовувати профіль baseline із помірними вимогами;
- у промислових просторах імен використовується профіль restricted, який забороняє запуск привілейованих контейнерів, вимикає небезпечні можливості та примушує запускати процеси від непривілейованих користувачів усередині контейнерів;
- у виняткових випадках для окремих системних компонентів може бути дозволений профіль privileged, але лише в спеціально виділених і ізольованих просторах імен.

Додаткову гнучкість забезпечують режими enforce, audit та warn. Завдяки їм можливо:

- спочатку вмикати політики у режимі аудиту, збираючи статистику про те, які саме застосунки не відповідають вимогам;
- переводити окремі середовища в режим попереджень, не блокуючи створення об’єктів, але інформуючи розробників про порушення;
- лише після адаптації маніфестів переходити до жорсткого режиму enforce, де невідповідні поди взагалі не допускаються до розгортання.

Таким чином, PSA виконує роль конфігураційного «фільтра безпеки», який працює ще до появи пода в кластері й тісно інтегрується з декларативною моделлю

Kubernetes. Для динамічного застосування політик це особливо важливо: змінивши профіль безпеки на рівні простору імен, організація одночасно впливає на всі нові розгортання в цьому середовищі, не змінюючи окремі маніфести.

2.2 Admission controllers як механізм динамічного застосування політик

У першому розділі було показано, що кожен запит до Kubernetes API проходить послідовні етапи: автентифікацію, авторизацію та подальшу обробку на рівні admission-плагінів. Саме admission controllers виступають тим рівнем, де політики безпеки та відповідності можуть застосовуватись динамічно – без зміни вихідного коду застосунків і без ручного втручання адміністраторів при кожній операції над ресурсами [12].

Admission controller – це компонент, що перехоплює запити до kube-apiserver після успішної автентифікації та авторизації, але до моменту збереження об’єкта в сховище etcd. Він застосовується до операцій створення, оновлення й видалення об’єктів, а в окремих випадках – і до «нестандартних» дій на кшталт підключення до Pod через API-проксі. Водночас вони не впливають на операції читання, що дає змогу розділити контроль конфігурацій і контроль доступу до даних.

Набір увімкнених admission-плагінів налаштовується через параметр kube-apiserver --enable-admission-plugins, а додаткову гнучкість забезпечують механізми динамічного admission control – webhook та декларативні політики, які описуються у формі типових об’єктів Kubernetes (Рисунок 2.1).

Рисунок 2.1 – Місце admission controllers у конвеєрі обробки запитів Kubernetes.

За характером впливу на запит виділяють два основні типи admission controllers: mutating (мутуючі) та validating (валідаційні).

Mutating admission controllers можуть змінювати об’єкт перед його збереженням: додавати значення за замовчуванням, інжектувати додаткові контейнери чи анотації, коригувати параметри ресурсів і налаштування безпеки тощо. Поширений приклад – автоматичне додавання resource limits або необхідних анотацій до Pod, навіть якщо розробник не зазначив їх у маніфесті.

Validating admission controllers не змінюють об’єкт, а лише ухвалюють рішення «дозволити / відхилити» на основі заданих правил. Якщо хоча б один валідаційний контролер відхиляє запит, об’єкт не потрапляє до кластера.

У Kubernetes admission-процес відбувається в два етапи: спочатку послідовно виконуються всі mutating-контролери, після чого над уже зміненим об’єктом паралельно працюють validating-контролери. Такий поділ важливий з точки зору безпеки: правила, які залежать від остаточного стану об’єкта, повинні реалізовуватись саме як валідаційні політики.

Базовий дистрибутив Kubernetes включає набір admission-плагінів, вбудованих у kube-apiserver та відповідальних за основні системні функції. Серед них виокремлюють три «розширювальні» контролери: MutatingAdmissionWebhook; ValidatingAdmissionWebhook; ValidatingAdmissionPolicy (Рисунок 2.2).

Перші два забезпечують інтеграцію з динамічними webhook, а третій дозволяє реалізовувати декларативні політики перевірки на основі виразів CEL без необхідності звернення до зовнішніх HTTP-сервісів [13].

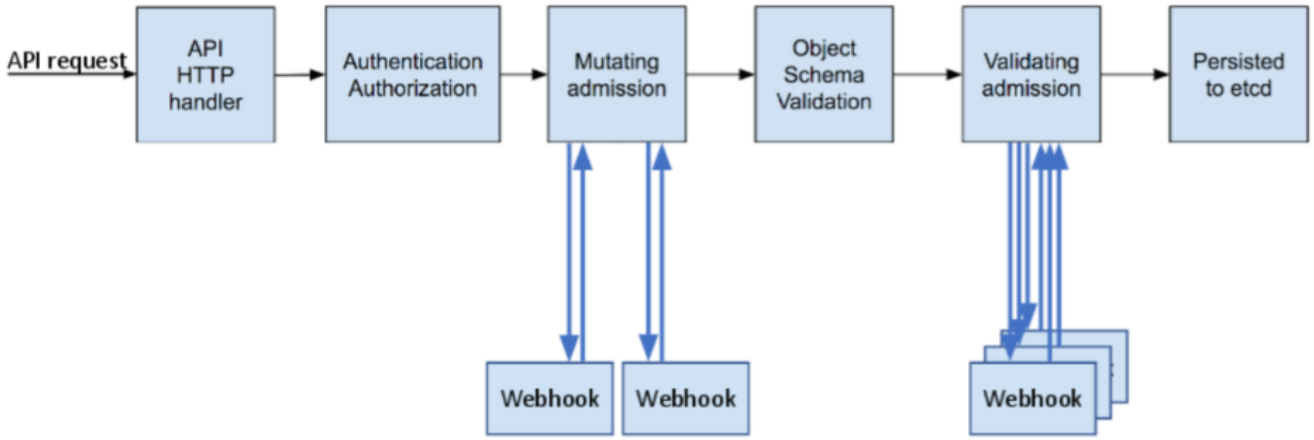

Рисунок 2.2 – Послідовність виклику mutating та validating admission controllers при обробці запиту до Kubernetes API.

На відміну від статичних рекомендацій або одноразового сканування конфігурацій, admission controllers працюють синхронно з кожним запитом, що змінює стан кластера. Тому вони є природною точкою впровадження динамічних політик безпеки: будь-яка зміна маніфестів повинна пройти через ці політики, і невідповідні об’єкти просто не потрапляють до кластера.

Ключові властивості admission controllers, які роблять їх придатними для динамічного застосування політик:

- контекстність – у рішенні можуть одночасно враховуватись атрибути запиту (користувач, групи, service account), тип ресурсу, простір імен, мітки, а також додаткові дані, отримані з зовнішніх систем (реєстри образів, системи ідентичності, SIEM тощо);
- декларативність та розширюваність – політики можуть описуватись як Kubernetes-об’єкти, що дозволяє керувати ними через Git, CI/CD і GitOps-підхід [19; 20];
- незалежність від сервісів – правила безпеки централізовані й не залежать від конкретної реалізації мікросервісів, що критично в багатокомандних середовищах.

Сучасні policy-механізми, зокрема Open Policy Agent / Gatekeeper та Kyverno, функціонують як динамічні admission controllers: вони отримують від kube-apiserver запити у форматі JSON-об’єктів, звіряють їх із наборами політик і повертають рішення про дозвіл, відхилення або модифікацію об’єкта. Унаслідок цього політики безпеки можуть еволюціонувати разом із середовищем: за появи нових вимог достатньо змінити декларацію політики, не вносячи змін до самих застосунків

У типових Kubernetes-кластерах admission controllers використовуються для реалізації широкого спектру політик, які стосуються не лише безпеки, а й керування ресурсами та відповідності стандартам:

- примусове задання обмежень ресурсів – автоматичне додавання requests/limits для CPU і пам’яті, блокування подів без цих параметрів;
- контроль реєстрів образів – дозволення розгортання контейнерів лише з довірених реєстрів або з образів, підписаних певним ключем; блокування невідомих або публічних реєстрів;
- обмеження конфігурацій безпеки – заборона привілейованих контейнерів, монтованих томів hostPath, запуску від root, а також обмеження Linux capabilities; частина таких вимог може дублюватися або деталізуватися відносно профілів Pod Security Standards;
- уніфікація метаданих – примусове додавання міток і анотацій (наприклад, team, environment, owner, cost-center), що спрощує подальший моніторинг та аналіз інцидентів;
- заборона небезпечних або застарілих API-версій – блокування створення об’єктів із використанням API, які вважаються вразливими або вилученими в нових версіях кластера;
- відповідність стандартам та політикам організації – реалізація вимог CIS Benchmark, NIST 800-190 чи внутрішніх стандартів, де описано, які саме параметри мають бути присутні або заборонені в маніфестах.

Важливо, що всі ці політики можуть оновлюватися в режимі реального часу: після зміни CRD-політик або webhook-конфігурацій нові вимоги починають застосовуватися до кожної наступної операції створення чи оновлення ресурсів, не потребуючи перезапуску кластера (Рисунок 2.3).

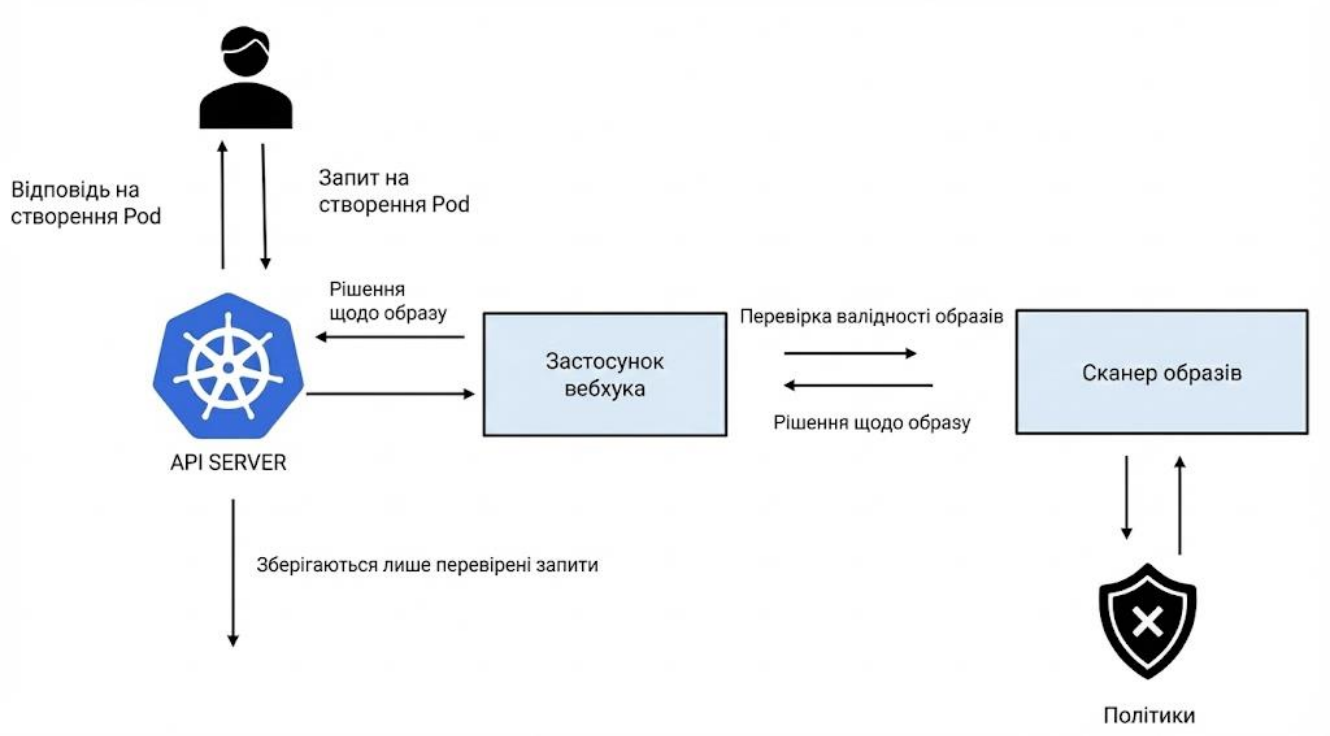

Рисунок 2.3 – Приклади політик на рівні admission controllers.

Це дозволяє будувати сценарії адаптивного захисту, коли реакція на виявлені інциденти включає також оперативне посилення admission-політик.

Попри очевидні переваги, застосування admission controllers супроводжується низкою викликів, які слід враховувати під час проєктування безпечного Kubernetes-середовища:

- Затримки та продуктивність. Кожен mutating або validating webhook додає затримку до обробки запитів до API. Велика кількість webhook’ів, складні політики або повільні зовнішні сервіси можуть призвести до відчутного сповільнення операцій керування кластером. Тому рекомендації Kubernetes прямо

вказують на необхідність оптимізації латентності admission-webhook’ів і обмеження їх кількості.

- Відмовостійкість і режим «fail-open/fail-closed». Якщо зовнішній webhook недоступний, адміністратор має обрати, чи блокувати всі запити (fail-closed), чи дозволяти їх без перевірки (fail-open). Обидва варіанти несуть ризики: у першому випадку можливий простій кластера, у другому – тимчасове обходження політик.
- Складність політик. Непродумане поєднання великої кількості правил, що перекривають одне одного, може призвести до непередбачуваних конфліктів, коли законні операції блокуються без зрозумілої причини. Це підкреслює важливість поетапного впровадження політик (режими audit/warn) та їхнього тестування в окремих середовищах.
- Обмеженість погляду на рантайм. Admission controllers бачать лише опис бажаного стану (маніфести, запити до API), але не реальну поведінку застосунків під час виконання. Тому вони повинні доповнюватися засобами моніторингу й виявлення аномалій у рантаймі (Falco, eBPF-інструменти), які аналізують системні виклики, мережеву активність і події на вузлах.

2.3 Концепція policy-as-code у безпеці контейнеризованих середовищ Kubernetes

Концепція policy-as-code у хмарних та контейнеризованих середовищах означає вираження вимог безпеки, доступу й відповідності у вигляді формальних правил, записаних як код і керованих тими самими інструментами, що й прикладне програмне забезпечення [15]. Політики описуються у зрозумілій для людини формі, зберігаються у системі контролю версій, проходять рецензування, тестування і автоматично застосовуються засобами CI/CD та GitOps. Такий підхід перетворює

політику безпеки з статичного документа на активний компонент інфраструктури, який безпосередньо впливає на кожну зміну конфігурації кластера.

У середовищі Kubernetes policy-as-code є логічним продовженням декларативної моделі керування ресурсами. Якщо YAML-маніфести описують бажаний стан об'єктів, то політики, виражені як код, визначають, які з цих конфігурацій вважаються припустимими з точки зору безпеки та внутрішніх стандартів організації. Це дозволяє реалізувати контроль безпеки не як одноразову перевірку, а як безперервний процес, інтегрований в усі етапи життєвого циклу контейнеризованих застосунків: від розробки та рев'ю маніфестів до їх розгортання в кластері й подальшої експлуатації.

У Kubernetes реалізація policy-as-code зазвичай спирається на спеціалізовані механізми політик, які інтегруються з API-сервером як admission-контролери. Найбільш поширеним універсальним механізмом є Open Policy Agent (OPA), що використовує декларативну мову Rego й дає змогу описувати політики для різних систем – від Kubernetes до API-шлюзів і CI-конвеєрів. У поєднанні з компонентом Gatekeeper OPA виступає як валідуючий admission-контролер: кожен об'єкт, який створюється або оновлюється в кластері, перевіряється на відповідність наборам обмежень, що зберігаються в Git як код.

Паралельно розвиваються орієнтовані на Kubernetes рішення, зокрема Kyverno – механізм політик, у якому самі політики представлені у вигляді стандартних Kubernetes-ресурсів і описуються мовою YAML. Такий підхід знижує поріг входу для платформних команд, оскільки дозволяє працювати з політиками у звичному форматі, без необхідності вивчати окрему мову запитів. Kyverno поєднує перевірку, мутування та генерацію ресурсів: одні правила блокують небезпечні конфігурації, інші автоматично доповнюють маніфести потрібними параметрами.

У практичних рекомендаціях щодо захисту Kubernetes-кластерів policy-as-code розглядається як базовий механізм керування ризиками: політики дозволяють систематично застосовувати найкращі практики, забезпечуючи їхнє автоматичне

виконання у кожному середовищі – від розробки до промислової експлуатації. Особливу увагу приділяють інтеграції з GitOps-підходом: політики зберігаються в репозиторіях разом із маніфестами, зміни проходять рев'ю, а спеціальні оператори відстежують відповідність реального стану кластера зафіксованим правилам [16].

Для безпеки контейнеризованих середовищ Kubernetes policy-as-code виконує одразу кілька функцій. По-перше, це механізм превентивного контролю конфігурацій: недопустимі об'єкти блокуються ще до появи в кластері. По-друге, це спосіб формалізації вимог до безпеки, які можуть надалі коригуватися на основі результатів моніторингу та виявлення аномалій. По-третє, це інструмент узгодження роботи команд розробки, експлуатації та безпеки в єдиному процесі, де політики розглядаються як такий самий важливий артефакт, як і вихідний код сервісів. Саме на цьому фундаменті будуються підходи до динамічного застосування політик та поєднання статичних обмежень з аналізом поведінки контейнерів у реальному часі.

2.4 OpenPolicy Agent та мова Rego для опису політик безпеки

Open Policy Agent (OPA) є загальнопризначеним механізмом політик, призначеним для уніфікації застосування політик на різних рівнях програмно-апаратного стеку [17]. OPA відокремлює процес ухвалення рішень від компонентів, що ці рішення реалізують: прикладні сервіси, інфраструктурні елементи або платформи контейнеризації передають до механізму структуровані дані про запит, а у відповідь отримують лаконічне рішення у форматі, наприклад, «дозволити/заборонити» чи перелік виявлених порушень. Така декомпозиція дає змогу змінювати й розвивати політики безпеки незалежно від бізнес-логіки системи.

Архітектура OPA передбачає наявність локального сховища політик і даних, інтерфейсу для отримання запитів та механізму оцінювання правил. Політики

завантажуються до OPA у вигляді окремих файлів, а допоміжні довідкові відомості зберігаються як дані. Після надходження запиту OPA обчислює значення визначених вихідних змінних згідно з правилами політики та повертає результат у стандартизованому форматі.

Політики в OPA описуються мовою Rego – спеціалізованою декларативною мовою, призначеною для роботи з ієрархічними структурами даних у форматах JSON та YAML [18]. Rego дозволяє формулювати правила у вигляді логічних виразів над полями вхідного документа, зосереджуючись на тому, який стан вважається допустимим, а не на покроковому алгоритмі перевірки. Типовою практикою є визначення логічної змінної, значення якої встановлюється в істинне, якщо виконуються всі умови безпечної конфігурації. За відсутності таких умов діє принцип «заборони за замовчуванням», що є бажаним з точки зору безпеки.

У середовищі Kubernetes OPA використовується насамперед як основа для реалізації policy-as-code на рівні admission-контролю. Найпоширенішим варіантом інтеграції є зв'язка OPA з компонентом Gatekeeper, який реєструється в кластері як веб-хуки перевірки й отримує від API-сервера об'єкти, що створюються або змінюються. Gatekeeper передає опис ресурсу разом з контекстною інформацією до OPA, де політики Rego оцінюються щодо конкретного запиту [19]. Якщо хоча б одна політика вказує на порушення, операція відхиляється, а до користувача повертається повідомлення з описом причини.

Політики, які застосовує OPA у зв'язці з Gatekeeper, зазвичай структуруються у вигляді шаблонів обмежень і конкретних обмежень, що застосовуються до вибраних груп ресурсів або просторів імен. Наприклад, один шаблон може описувати загальну логіку перевірки використання привілейованих контейнерів, а конкретні обмеження – визначати, до яких namespace слід застосовувати цю перевірку, та які саме параметри вважати припустимими. Таким чином мова Rego використовується не лише для простих булевих рішень, а й для формування структурованих звітів про порушення або списків необхідних змін.

Характерною особливістю Rego є можливість комбінувати кілька джерел даних у межах однієї політики. Механізм може одночасно аналізувати опис Kubernetes-об’єкта, довідкові таблиці довірених реєстрів, результати попереднього сканування вразливостей або іншу додаткову інформацію, що зберігається у сховищі даних OPA. Це дозволяє формулювати політики, які враховують не лише статичні властивості конфігурації, а й метадані про компонент, його критичність чи належність до певної бізнес-функції. Для безпеки контейнеризованих середовищ це відкриває можливість деталізованого контролю за розгортанням сервісів залежно від їх ролі й контексту.

Практичні приклади застосування OPA та Rego у Kubernetes охоплюють контроль використання привілеїв, обмеження доступу до hostPath-томів, примусове задання ресурсних лімітів, фіксацію вимог до міток і анотацій, контроль використання дозволених реєстрів образів, а також заборону небезпечних або застарілих параметрів конфігурації. У поєднанні з підходом policy-as-code політики записуються як код у репозиторіях, версіонуються, тестуються й автоматично розгортаються в кластерах, що забезпечує послідовність вимог безпеки в різних середовищах.

У контексті безпеки контейнеризованих середовищ Kubernetes OPA та Rego доцільно розглядати як гнучкий механізм формалізації політик, який легко інтегрується з наявною інфраструктурою та процесами розробки. Політики, виражені мовою Rego, можуть бути пов’язані з результатами моніторингу й виявлення аномалій, що дозволяє коригувати правила безпеки у відповідь на нові загрози. Оновлення таких політик через системи контролю версій і подальше їх застосування на рівні admission-контролерів створюють передумови для динамічного управління ризиками в Kubernetes-кластерах.

2.5 Kyverno як Kubernetes-native policy engine

Kyverno є спеціалізованим механізмом політик, орієнтованим на Kubernetes і призначеним для автоматизації вимог безпеки, відповідності та операційних практик у межах кластера [20]. На відміну від універсальних механізмів політик, які вимагають використання окремої мови опису правил, Kyverno працює з уже звичним для адміністраторів форматом – маніфестами Kubernetes у вигляді YAML-ресурсів. Самі політики представлені як об'єкти Kubernetes, що полегшує їх вбудовування в наявні процеси керування кластером і дає змогу управляти ними так само, як іншими кластерами ресурсів.

Ключовою особливістю Kyverno є його повна відповідність моделі Kubernetes. Kyverno розгортається в кластері як набір контролерів і динамічних admission webhook'ів, які перехоплюють запити до API-сервера, що змінюють стан кластера. Для кожної операції створення, оновлення або видалення ресурсу Kyverno отримує структуру запиту, зіставляє її з наявними політиками та ухвалює рішення: дозволити операцію, заблокувати її, змінити ресурс (mutate) або згенерувати додаткові об'єкти. Таким чином забезпечується безперервний контроль безпеки на шляху будь-якої зміни конфігурації контейнеризованих навантажень.

У сучасних версіях Kyverno політики описуються через набір спеціалізованих типів ресурсів, серед яких ClusterPolicy, ValidatingPolicy, ImageValidatingPolicy, MutatingPolicy, GeneratingPolicy, CleanupPolicy [21]. Кожна політика містить один або кілька правил, що визначають: до яких ресурсів вона застосовується (через селектори назв, міток, типів і версій API), які дії слід виконати (валідація, модифікація, генерація, перевірка підписів образів), а також у якому режимі працює політика – наприклад, лише аудит чи жорстке блокування. Така модель дозволяє водночас реалізувати як запобіжний контроль (enforce), так і поступове впровадження вимог у режимі спостереження.

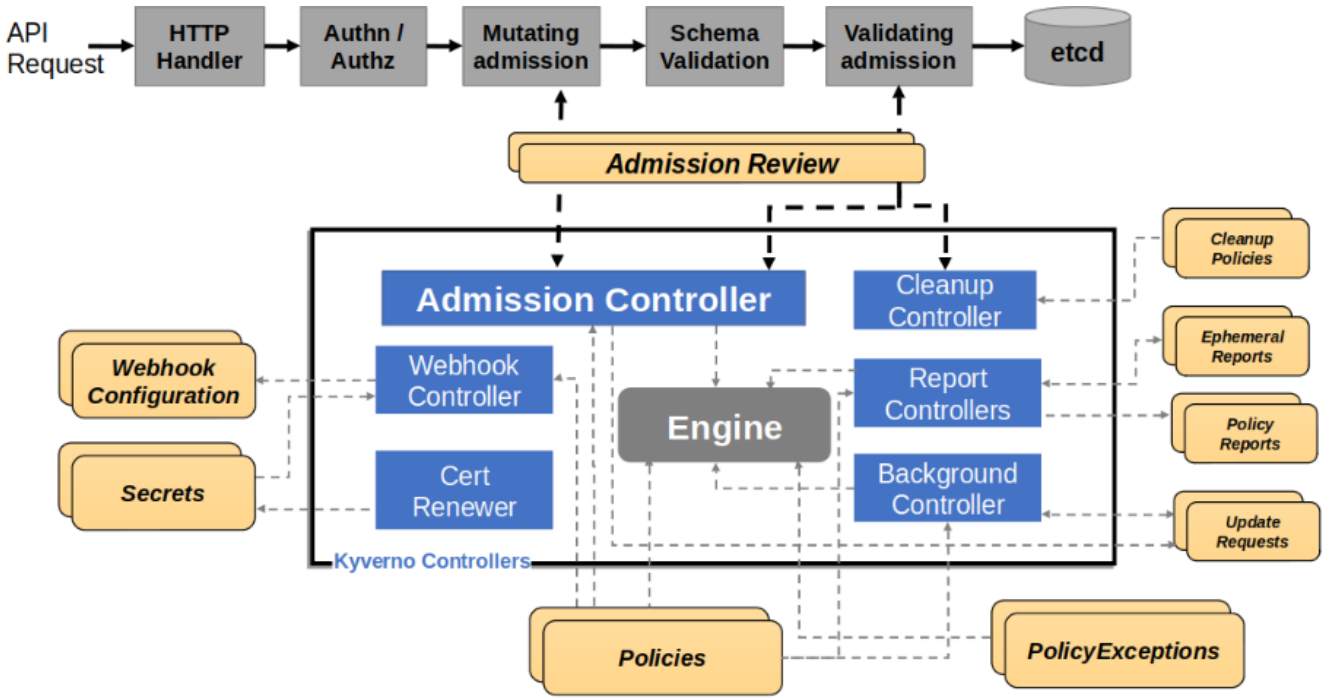

Рисунок 2.4 – Взаємодія Kyverno з Kubernetes.

У контексті безпеки контейнеризованих середовищ Kyverno підтримує кілька ключових можливостей. Передусім політики валідації дають змогу блокувати конфігурації, що створюють ризики або не відповідають вимогам: запуск привілейованих контейнерів, використання hostPath-томів, відсутність лімітів ресурсів, виконання процесів від користувача root, застосування застарілих API-версій. Далі, завдяки політикам мутування система може автоматично доповнювати маніфести необхідними параметрами – встановлювати requests/limits для CPU й пам’яті, додавати обов’язкові мітки та анотації, примусово налаштовувати securityContext. Нарешті, політики генерації дають можливість створювати пов’язані ресурси у відповідь на події розгортання, що забезпечує послідовність конфігурацій без ручного копіювання.

Окремий напрямок функціональності Kyverno пов’язаний із перевіркою контейнерних образів та підтримкою вимог до ланцюга постачання програмного забезпечення. Політики типу verifyImages дозволяють перевіряти підписи контейнерних образів, атестації та метадані, обмежувати використання образів

лише довіреними реєстрами і блокувати невідомі або неперевірені джерела. Це безпосередньо пов’язано з безпекою контейнеризованих середовищ, оскільки дозволяє автоматично запобігати розгортанню потенційно небезпечних образів у кластері.

Kyverno тісно поєднується з підходом policy-as-code. Політики описуються в YAML, зберігаються у системах контролю версій, проходять рецензування та автоматично застосовуються до кластерів через конвеєри CI/CD і GitOps-інструменти. Така інтеграція дозволяє забезпечити єдині стандарти безпеки для різних середовищ та різних кластерів, а також швидко поширювати оновлені політики на всі цільові інфраструктури. Додатково надається CLI-інструментарій, який дозволяє перевіряти ресурси та політики поза межами кластера, зокрема в рамках етапів статичного аналізу в CI.

Важливою складовою Kyverno є можливість проведення фонових сканувань уже наявних ресурсів та формування звітів про порушення політик. Результати перевірок агрегуються у вигляді об’єктів звітів, які можуть бути інтегровані з системами моніторингу, візуалізації та оповіщення, зокрема через спеціалізовані компоненти для побудови дашбордів і надсилання сповіщень. Це забезпечує не лише превентивний контроль на етапі admission, а й періодичну оцінку фактичного стану кластера щодо визначених стандартів безпеки.

Kyverno розвивається як проєкт з відкритим кодом і підтримується спільнотою, що забезпечує наявність великої кількості готових прикладів політик для типових сценаріїв безпеки в Kubernetes. У публічних репозиторіях доступні шаблони для реалізації рекомендацій Pod Security Standards, обмеження запуску привілейованих контейнерів, контролю мережевих налаштувань, захисту критично важливих ресурсів від випадкового видалення та розв’язання інших типових завдань. Наявність таких шаблонів полегшує початкове впровадження Kyverno та дає змогу зосередитися на адаптації політик до специфічних вимог організації.

Попри свої переваги, використання Kyverno потребує ретельного проєктування політик і врахування питань продуктивності. Як і у випадку з іншими механізмами, що працюють на рівні admission, велика кількість складних правил може збільшувати затримку під час обробки запитів до API-сервера, особливо в кластерах із високим навантаженням. Рекомендовано впроваджувати політики поетапно: спочатку використовувати режим аудиту, проаналізувати отримані звіти й лише потім поступово переводити критичні правила в режим примусового застосування. За дотримання цих підходів Kyverno виступає як потужний Kubernetes-native інструмент керування політиками, що підтримує динамічне застосування вимог безпеки в контейнеризованих середовищах.

2.6 Приклади політик безпеки для контролю конфігурацій (Rego / Kyverno)

Політики безпеки, реалізовані за допомогою Rego та Kyverno, дозволяють формалізувати типові вимоги до конфігурацій контейнеризованих навантажень і зробити їх обов’язковими для всіх команд, що працюють із кластером Kubernetes. На практиці це реалізується як набір правил, які блокують небезпечні конфігурації, автоматично доповнюють маніфести відсутніми параметрами або генерують супровідні ресурси. Нижче наведено кілька типових категорій політик, важливих для безпеки контейнеризованих середовищ.

Поширеним прикладом є політики, що забороняють запуск привілейованих контейнерів та використання небезпечних можливостей ядра. У випадку Rego така політика перевіряє значення полів securityContext і capabilities у специфікації контейнерів та формує порушення, якщо виявлено privileged: true або включення чутливих можливостей, наприклад NET_ADMIN чи SYS_ADMIN. Аналогічна політика в Kyverno описується у вигляді правила валідації, яке застосовується до ресурсів типу Pod і шаблонів розгортань; у разі невідповідності запит відхиляється

з поясненням причини. Таким чином забезпечується примусовий контроль за тим, щоб робочі навантаження запускалися у максимально обмеженому режимі.

Важливу роль відіграють політики, що гарантують коректне використання ресурсів. Для цього формулюються правила, які вимагають наявності параметрів resources.requests і resources.limits для CPU та оперативної пам'яті у всіх контейнерах певних просторів імен. Реалізація на Rego може трактувати відсутність цих полів як порушення, тоді як Kyverno, крім валідації, здатне автоматично додавати значення за замовчуванням за допомогою політик мутування. Такий підхід дозволяє одночасно запобігати неконтрольованому споживанню ресурсів і спрощувати роботу команд розробки, які можуть покладатися на узгоджені дефолтні значення.

Окремо виділяють політики, орієнтовані на контроль походження контейнерних образів. У випадку Rego поширеною практикою є правило, яке звіряє значення поля image в описі контейнерів із переліком дозволених реєстрів, збереженим у сховищі даних OPA. Якщо виявлено образ із невідомого чи забороненого реєстру, політика формує відмову. У Kyverno аналогічна функціональність реалізується за допомогою директив verifyImages, що дозволяють обмежити використання образів визначеним набором реєстрів і виконувати перевірку криптографічних підписів або атестацій. Таким чином, до кластера допускаються лише сервіси з образами з довірених джерел, що істотно зменшує ризик використання підроблених контейнерів.

З точки зору забезпечення комплаєнсу та зручності керування корисними є політики, що стандартизують метадані. До них належать правила, які зобов'язують додавати обов'язкові мітки на рівні Pod або Deployment, наприклад app, team, environment, owner. У Rego така вимога реалізується шляхом аналізу секції metadata.labels і фіксації порушення за відсутності хоча б однієї з необхідних міток. У Kyverno подібний підхід може бути втілений у формі валідаційних або мутуючих політик, які автоматично доповнюють об'єкти стандартним набором міток. У

результаті спрощуються задачі логування, моніторингу, білінгу та аналізу інцидентів, адже кожен об’єкт у кластері має чітко визначені атрибути приналежності.

Важливим напрямком є політики, пов’язані з Pod Security Standards та іншими рекомендаціями щодо «зміцнення» контейнерів. У практиці використання Rego для Kubernetes поширені правила, які перевіряють відповідність маніфестів вимогам профілю restricted: заборона hostNetwork, hostPID, hostIPC, використання непривілейованих користувачів, обмеження томів типу hostPath та інші параметри. Kyverno, зі свого боку, має готові набори політик, що реалізують вимоги профілів baseline та restricted, і може застосовувати їх до обраних просторів імен у різних режимах – від аудиту до жорсткого блокування. Це дозволяє інституціоналізувати стандарти безпеки на рівні кластера.

Ще один клас політик стосується захисту критичних конфігурацій і секретів. У Rego можуть бути сформульовані правила, що забороняють розгортання ресурсів із прямим вбудовуванням конфіденційних даних у поля env або configMap, вимагаючи використання механізму Secret та відповідних посилань. Kyverno дозволяє блокувати операції видалення або змінення певних чутливих ресурсів (наприклад, конфігурацій кластерних компонентів або просторов імен для системних сервісів) і генерувати додаткові захисні об’єкти, такі як політики мережевої безпеки, при створенні нових сервісів.

У сукупності такі політики демонструють, як Rego та Kyverno можуть використовуватися для побудови системи конфігураційного контролю, де вимоги безпеки та комплаєнсу виражені у вигляді формальних правил, інтегрованих у механізми admission-контролю Kubernetes. Формалізація типових вимог – від обмеження привілеїв і контролю ресурсів до перевірки джерел образів і уніфікації метаданих – створює основу для динамічного застосування політик, здатних еволюціонувати разом із середовищем і реагувати на нові загрози.

2.7 Інтеграція політик безпеки у GitOps-процеси та CI/CD-пайплайни

Побудова інтеграції політик безпеки з GitOps та CI/CD-пайплайнами ґрунтується на концепції, за якою всі суттєві елементи роботи кластера – інфраструктура, налаштування сервісів і правила безпеки – описуються декларативно та підтримуються в системі контролю версій як єдиному джерелі істини [23]. Git у цьому контексті є вузловою ланкою, де зберігається бажаний стан, що включає маніфести Kubernetes, політики OPA/Rego, політики Kyverno та додаткові конфігурації. Будь-які зміни до цього стану проходять через коміти та запити на злиття, що дає змогу використовувати усталені процеси рецензування й повного аудиту історії.

У GitOps-моделі оператори типу Argo CD або Flux відстежують репозиторій із маніфестами й політиками, періодично порівнюють вміст Git зі станом кластера та автоматично приводять середовище у відповідність до описаного бажаного стану [24]. Це створює безперервний цикл узгодження, в межах якого будь-яка зміна в репозиторії (оновлення версії образу, додавання нового сервісу, зміна політики безпеки) призводить до послідовних автоматизованих дій: запуску CI-перевірок, оновлення цільових гілок, синхронізації GitOps-оператором та застосування конфігурацій у кластері. Політики безпеки при цьому розглядаються як невід’ємна частина декларативного стану, а не як окремий ручний етап.

На етапі безперервної доставки GitOps-оператор розгортає в кластері як прикладні ресурси, так і об’єкти політик. Для OPA це можуть бути шаблони обмежень і самі обмеження у Gatekeeper, для Kyverno – ClusterPolicy та інші типи політик, які контролюють конфігурації подів, розгортань, конфігураційних карт і секретів. Політики потрапляють у кластер тим самим шляхом, що й звичайні маніфести, тому будь-яка зміна вимог безпеки фіксується в Git, проходить рев’ю й автоматично розгортається у всіх цільових середовищах. Це забезпечує

послідовність і відтворюваність політик у багатокластерних і мультихмарних сценаріях.

Важливу роль відіграє поєднання GitOps-механізмів із admission-контролем Kubernetes. Навіть якщо певна зміна потрапила до кластера в обхід стандартного GitOps-шляху, валідуючі й мутуючі admission-контролери, побудовані на базі OPA або Kyverno, застосовують політики до кожного запиту, що змінює стан кластера. Це створює додатковий рівень захисту: політики, керовані через Git, фактично блокують небажані конфігурації вже на межі кластера, незалежно від джерела запиту. У практичних рекомендаціях описується модель «двох циклів», де GitOps відповідає за узгодження бажаного стану, а політики – за узгодження цього стану з вимогами безпеки.

Ще один аспект інтеграції стосується захисту самого GitOps-ланцюжка. Для репозиторіїв із маніфестами й політиками застосовуються окремі вимоги: обмеження доступу, захист гілок, обов’язковий code review, підпис комітів та артефактів, контроль цілісності через сканування історії змін і перевірку цифрових підписів. Додатково рекомендується розділяти репозиторії за середовищами, запроваджувати різні рівні доступу до production-гілок і використовувати механізми перевірки секретів, щоб уникати випадкового розміщення конфіденційних даних у Git. Усе це розглядається як частина моделі загального управління ризиками для GitOps-процесів.

У контексті CI/CD-пайплайнів ключову роль відіграє чітко вибудований ланцюжок перевірок. На початкових стадіях виконуються статичні тести політик щодо інфраструктурного коду; після їх успішного завершення зміни потрапляють у гілку, яку надалі використовує GitOps-оператор. Під час етапу синхронізації оператор доставляє конфігураційні оновлення в кластер, де політики Kyverno або OPA повторно застосовуються як admission-контролери. У низці сценаріїв результати перевірок політик (наприклад, звіти Kyverno чи Gatekeeper) додатково

передаються в системи моніторингу й оповіщення, створюючи ефективний канал зворотного зв’язку для команд розробки та безпеки (таблиця 2.2).

Таблиця 2.2 - Порівняльна характеристика інструментів GitOps Flux CD та Argo CD

Характеристика	Flux CD	Argo CD
Модель розгортання	GitOps з підходом pull на основі контролерів Kubernetes	GitOps з підходом pull з можливістю ручної синхронізації
Інтерфейс та спостережуваність	Немає власного вебінтерфейсу, CLI та дашборди через Grafana і Prometheus	Повнофункціональний вебінтерфейс з візуалізацією стану, журналами і відмінностями конфігурацій
Багатокористувацький режим і доступ	Простори імен Kubernetes і стандартний RBAC	AppProject і гнучкий RBAC, що налаштовується через інтерфейс або файли
Підтримка Helm та розширюваність	Вбудований контролер Helm, додаткові контролери для розширення	Вбудований рендеринг Helm і плагіни керування конфігурацією
Робота з секретами	Вбудована підтримка SOPS	Зовнішні інструменти через плагіни
Кілька кластерів	Окреме встановлення Flux CD у кожен кластер	Єдиний інтерфейс для централізованого керування кількома кластерами
Статус і спільнота	Проєкт CNCF рівня graduated, під керівництвом Weaveworks	Проєкт CNCF рівня incubating, під керівництвом Intuit

Референсні архітектури показують модель, у якій політики безпеки охоплюють увесь ланцюг постачання: від перевірки конфігурацій у CI та GitOps-контролю цілісності репозиторіїв до admission-перевірок у кластері й подальшого моніторингу аномалій у рантаймі. У межах такого підходу зміна політики – скажімо, посилення Pod Security або звуження переліку дозволених джерел образів – реалізується як модифікація коду, фіксується в Git, проходить код-рев'ю та автоматично впливає на всі майбутні розгортання. Це дає змогу підтримувати динамічні політики безпеки, що розвиваються разом із системою й водночас залишаються керованими та прозорими.

2.8 Інструменти тестування та валідації політик

Динамічне застосування політик безпеки можливе лише тоді, коли самі політики є перевіреними й прогнозованими. Для цього використовують спеціалізовані інструменти тестування та валідації, які дозволяють ставитися до політик так само, як до коду: писати для них "юніт-тести", запускати регресійні перевірки та інтегрувати їх у CI/CD-процеси [25]. Такі інструменти дають змогу локально перевірити логіку правил, протестувати їх на типових прикладах конфігурацій і лише після цього вмикати жорстке застосування політик у кластері. Це зменшує ризик випадкового блокування легітимних змін і спрощує еволюцію політик у часі.

Одним з найпоширеніших інструментів для тестування політик на основі Rego є conftest. Це консольна утиліта, яка використовує механізм Open Policy Agent та дозволяє перевіряти будь-які структуровані конфігураційні файли – зокрема Kubernetes-маніфести, Terraform-конфігурації, Dockerfile та інший інфраструктурний код . Політики описуються мовою Rego, а команда conftest test виконує їх щодо наданих файлів і повертає, які правила виконані, а які порушені. Завдяки підтримці різних форматів даних conftest зручно використовувати в CI-

пайплайнах: кожен коміт, що змінює маніфести або IaC-код, супроводжується автоматичним запуском політичних тестів, які блокують зміни, що суперечать вимогам безпеки чи комплаєнсу.

Kyverno CLI є орієнтованим саме на Kubernetes інструментом для локальної перевірки політик Kyverno та поведінки цих політик щодо конкретних ресурсів. Команди test та apply дозволяють проганяти політики на наборі YAML-маніфестів поза межами кластера, порівнювати фактичний результат із очікуваним, а також перевіряти коректність самих політик. Для опису очікуваного результату використовують окремі тестові маніфести, де зазначається, який ресурс має бути дозволений, заблокований або змінений у результаті застосування політики. Такий підхід дає змогу створювати регресійні набори тестів для політик: при кожній зміні правил запускається kyverno CLI, і якщо результати відрізняються від очікуваних, зміна політики не проходить перевірку. Kyverno CLI активно застосовується у CI/CD-процесах для попередньої валідації політик та маніфестів перед тим, як їх буде передано GitOps-оператору або застосовано до кластера.

Для OPA-політик, крім утиліти conftest, передбачено вбудований механізм перевірки за допомогою команди opa test. Вона дозволяє оформлювати тести до Rego-політик у вигляді окремих файлів і запускати їх як модульні тести, порівнюючи фактичні значення логічних змінних або результатів запитів з очікуваними. Це спрощує й пришвидшує розробку політик, а також дає змогу виявляти помилки в логіці до етапу інтеграції з Kubernetes чи іншими компонентами. Для інтерактивного опрацювання та відлагодження політик використовують веб-інструменти на кшталт Rego Playground, які дають можливість експериментувати з правилами та вхідними даними без побудови повної інфраструктури.

У сучасних практиках DevSecOps інструменти тестування політик інтегруються в усі етапи життєвого циклу змін. На ранніх стадіях розробки політики перевіряються локально (opa test, conftest, kyverno CLI), далі ті самі

перевірки вбудовуються в CI-пайплайни, де виконуються при кожному коміті, а у production-середовищах політики вже застосовуються через admission-контролери Kubernetes. Така багаторівнева перевірка зменшує ризик того, що помилка в політиці призведе до збоїв у кластері, і одночасно забезпечує можливість швидко адаптувати правила до нових загроз. У поєднанні з механізмами виявлення аномалій інструменти тестування та валідації політик дозволяють будувати динамічну, але контрольовану систему керування безпекою в контейнеризованих середовищах Kubernetes [26].

2.9 Порівняльний аналіз підходів OPA/Gatekeeper та Kyverno

У сучасних контейнеризованих середовищах OPA/Gatekeeper і Kyverno виступають двома ключовими інструментами реалізації політик для Kubernetes, що втілюють підхід policy-as-code та здійснюють контроль конфігурацій на рівні admission-контролю. В обох випадках вимоги безпеки задаються у формі формалізованих правил, які дають змогу відхиляти некоректні маніфести до того, як вони будуть застосовані в кластері. При цьому підходи помітно різняться за концепцією, мовними засобами опису політик, архітектурною організацією та типовими сценаріями застосування.

OPA/Gatekeeper базується на загальнопризначеному механізмі Open Policy Agent, у якому політики описуються декларативною мовою Rego. Gatekeeper забезпечує глибоку інтеграцію з Kubernetes за допомогою набору CRD (ConstraintTemplate, Constraint), контролерів і webhook’ів, що реалізують валідацію та аудит конфігурацій. Така архітектура дає змогу використовувати єдиний механізм роботи з політиками в різних доменах – Kubernetes, API-шлюзах, CI/CD-пайплайнах, Terraform – і вибудовувати спільну модель політик для всієї хмарної інфраструктури.

Kyverno, навпаки, від початку проєктувався як Kubernetes-native policy engine, в якому політики описуються у форматі YAML як звичайні ресурси кластера. Логіка правил максимально наближена до структури Kubernetes-об’єктів, а дії політик поділяються на валідацію, мутування, генерацію та «очищення» ресурсів (таблиця 2.3).

Таблиця 2.3 – Порівняння інструментів політик Open Policy Agent (OPA) Gatekeeper і Kyverno

Характеристика	Kyverno	Open Policy Agent (OPA)
Мова політик	YAML	Rego (спеціалізована предметно-орієнтована мова, DSL)
Простота використання	Нативний для Kubernetes інструмент із простими політиками на основі YAML	Потребує вивчення мови Rego
Типи політик	Валідація, модифікація (mutation), генерація ресурсів	Валідація, модифікація (mutation)
Інтеграція	Працює нативно всередині Kubernetes	Потребує додаткового налаштування та інтеграції
Продуктивність	Легковаговий механізм, оптимізований для Kubernetes	Для складних політик може вносити додаткову затримку
Типові сценарії використання	Безпека, відповідність вимогам (compliance), управління (governance)	Безпека, відповідність вимогам (compliance), загальне забезпечення дотримання політик
Складність упровадження	Порівняно простий для користувачів Kubernetes	Вимагає глибшого розуміння Rego та інтеграцій

З точки зору виразності політик OPA/Rego забезпечує ширші можливості для складної логіки, комбінування різних джерел даних і реалізації нетривіальних умов доступу чи комплаєнсу. Rego дає змогу будувати політики, які одночасно аналізують структуру ресурсу, довідкові таблиці, результати сканування вразливостей та інші контекстні дані. Це робить OPA/Gatekeeper привабливим у сценаріях, де необхідно уніфікувати політики на рівні всієї організації й узгоджувати їх не лише з Kubernetes, але й з іншими системами. Kyverno натомість пропонує менш універсальну, проте більш інтуїтивну модель, орієнтовану на типові задачі безпеки та керування конфігураціями Kubernetes-кластерів.

Порівняння з операційної точки зору показує, що обидва рішення використовують механізм admission-контролю й підтримують аудиторські перевірки вже наявних ресурсів, однак реалізують це по-різному. Gatekeeper надає окремий аудит-контролер, який періодично перевіряє ресурси на відповідність обмеженням і формує звіти. Kyverno, крім перевірок на admission-шляху, виконує фонові сканування та генерує об’єкти звітів, а також має вбудовані механізми генерації додаткових ресурсів (наприклад, NetworkPolicy або ConfigMap) [27]. У практичних оглядах відзначається, що OPA/Gatekeeper краще підходить для організацій із сильною орієнтацією на єдиний механізм політик для всієї інфраструктури, тоді як Kyverno зручніший як платформиний інструмент для команд, що будують стандартизоване Kubernetes-середовище.

З точки зору інтеграції у GitOps і CI/CD-процеси обидва підходи підтримують сценарії policy-as-code. OPA/Gatekeeper природно поєднується з інструментами на кшталт conftest та opa test для перевірки Rego-політик у пайплайнах, а самі ConstraintTemplate і Constraint зберігаються в репозиторіях разом з іншими маніфестами. Kyverno пропонує власний CLI для тестування політик і маніфестів, а політики у форматі YAML органічно вписуються у GitOps-схеми синхронізації кластерів через Argo CD або Flux. У публічних порівняннях підкреслюється, що у випадку, коли політики мають виходити за межі Kubernetes, перевагу зазвичай

віддають OPA, тоді як для суто Kubernetes-кластерів простішим у впровадженні є Kyverno.

Підсумовуючи, можна стверджувати, що OPA та Kyverno не є взаємовиключними рішеннями, а радше інструментами, націленими на різні пріоритети. У випадку, коли важливо створити єдину систему керування політиками для кількох середовищ і сервісів із високим ступенем гнучкості та повторного використання правил, більш виправданим є вибір OPA/Gatekeeper. Якщо ж ключовою метою є швидке запровадження Kubernetes-native політик, які є інтуїтивно зрозумілими для DevOps і платформних команд без додаткового навчання, логічніше обрати Kyverno. Обидва рішення дають змогу реалізувати динамічні політики безпеки; остаточний вибір визначається архітектурними вимогами, зрілістю команд і тим, наскільки критичною є потреба виходити з політиками за межі одного кластера.

РОЗДІЛ 3. ІНТЕГРОВАНА СИСТЕМА ДИНАМІЧНОГО ЗАСТОСУВАННЯ ПОЛІТИК ТА АВТОМАТИЧНОГО ВИЯВЛЕННЯ АНОМАЛІЙ У KUBERNETES

3.1 Архітектура тестового середовища та інтегрованої системи

Архітектуру тестового середовища спроєктовано так, щоб, його структура максимально відтворювала конфігурацію кластера Kubernetes, який використовується у реальному середовищі експлуатаці. На основі такого підходу реалізовано цілісну систему, що поєднує динамічне застосування політик безпеки, виявлення аномалій у режимі реального часу та централізований аналіз подій.

Основою середовища є дві віртуальна машина з операційною системою Rocky Linux 9.6, на одній за допомогою minikube розгорнуто одновузловий кластер Kubernetes, а на іншій було налаштовано кластер Elastic Stack , що виконує роль сховища й аналітичної платформи.

Логічна структура кластера організована через поділ на простори імен як показано на рисунку 3.1.

```
[root@rockyserver ~]# minikube kubectl -- get namespaces
NAME                   STATUS   AGE
applications           Active   9h
default                Active   14d
kube-node-lease        Active   14d
kube-public            Active   14d
kube-system            Active   14d
kubernetes-dashboard   Active   14d
observability          Active   9h
security-platform      Active   9h
```

Рисунок 3.1 – Простори імен розгорнуті у кластері minikube.

Виділено три основні простори імен:

- security-platform — простір, у якому розгорнуті всі компоненти платформи безпеки: механізми політик Kyverno та OPA Gatekeeper, система виявлення аномалій Falco, службові конфігурації та допоміжні сервіси
- applications — простір для тестових застосунків, що імітують реальні сервіси. Частина з них навмисно конфігурується з потенційно небезпечними параметрами або вразливостями для моделювання атак і перевірки спрацювань системи.
- observability — простір для агентів збору логів та метрик, які забезпечують транспортування подій у зовнішню підсистему моніторингу й аналітики.

Такий поділ дозволяє чітко відокремити бізнес-навантаження від інфраструктури безпеки й спостережуваності, а також застосовувати різні політики та режими контролю до кожного простору імен.

Ядром динамічного застосування політик виступають Kyverno та OPA Gatekeeper, розгорнуті як admission-контролери. Kyverno використовується для декларативних політик, що описуються у вигляді Kubernetes-ресурсів. OPA Gatekeeper забезпечує реалізацію більш складних правил на основі Rego, де важлива перевірка контексту. Обидва механізми працюють на рівні Kubernetes API: будь-які спроби створити або змінити ресурси проходять через цей шар контролю, де приймається рішення про допуск або блокування.

Підсистема виявлення аномалій у режимі реального часу побудована на базі Falco. Він розгортається як DaemonSet у просторі security-platform і на кожному вузлі кластера запускає агент, який використовує eBPF для перехоплення системних викликів ядра, пов’язаних із контейнерами. Отримані події збагачуються Kubernetes-метаданими (namespace, pod, node, образ контейнера, лейбли) та порівнюються з набором правил. Таким чином реалізується поведінковий рівень контролю, який доповнює декларативні політики: навіть якщо небезпечна

конфігурація пройшла admission-контроль або з’явилась унаслідок помилки, Falco дозволяє виявити підозрілу активність безпосередньо під час виконання (Рисунок 3.2).

```
[root@rockyserver ~]# minikube kubectl -- get pods -n security-platform
NAME                                            READY   STATUS    RESTARTS
falco-hhqfq                                     2/2     Running   2 (7h1m ago)
gatekeeper-audit-86f7c844f6-2rqqr               1/1     Running   10
gatekeeper-controller-manager-f7cbf498-g8wc2    1/1     Running   2 (5h46m ago)
gatekeeper-controller-manager-f7cbf498-kh27d    1/1     Running   1 (7h4m ago)
gatekeeper-controller-manager-f7cbf498-w24f8    1/1     Running   0
kyverno-admission-controller-747f7dffc9-sln7s   1/1     Running   25 (33s ago)
kyverno-background-controller-74fbc6479f-fgs29  1/1     Running   24 (33s ago)
kyverno-cleanup-controller-c5d8dc59d-l4hpp      1/1     Running   33 (34s ago)
kyverno-reports-controller-576566fb98-brrjs     1/1     Running   25 (33s ago)
```

Рисунок 3.2 – Стан ресурсів kyverno,opa та falco.

Централізоване збирання, зберігання та аналіз подій здійснюється за допомогою Elastic Stack. Elasticsearch і Kibana. Агенти збору логів, розгорнуті в просторі observability, забирають:

- журнали Falco;
- журнали Kyverno та OPA;
- журнали Kubernetes API;
- журнали тестових застосунків у просторі applications.

Потоки логів нормалізуються й індексуються в Elasticsearch у спеціалізованих індексних просторах (Рисунок 3.3), що дозволяє будувати інформаційні панелі, кореляційні запити та правила сповіщень у Kibana. У результаті всі рівні захисту — політики Kubernetes, поведінковий моніторинг на рівні ядра та журнали застосунків — зводяться в один аналітичний центр.

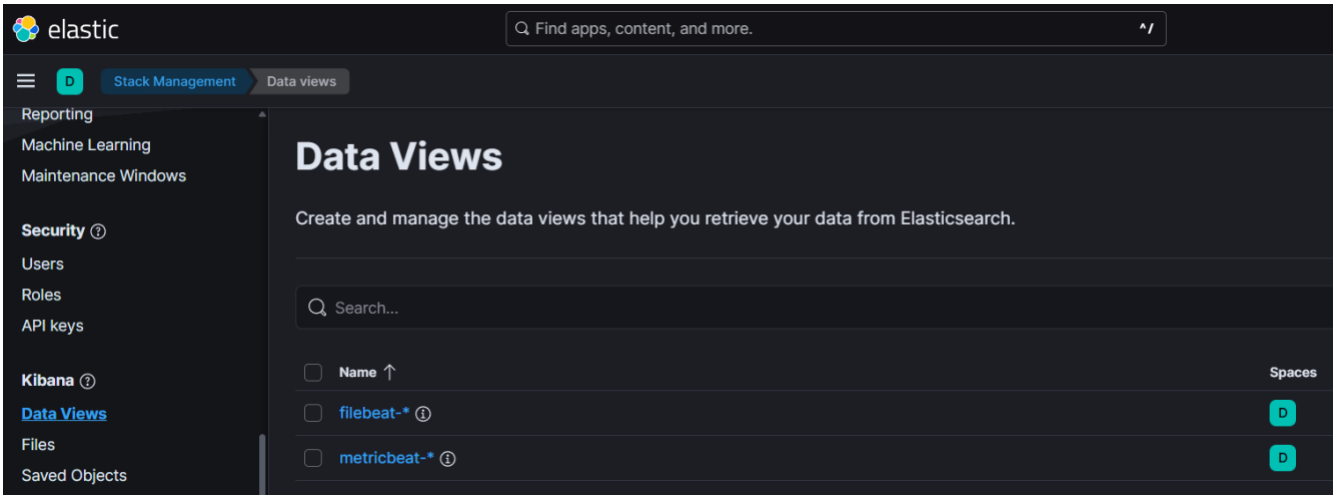

Рисунок 3.3 – Індекси elastic.

Зовні до цієї архітектури підключається CI/CD-конвеєр, який реалізує підхід policy-as-code. Репозиторій коду містить Kubernetes-маніфести, політики Kyverno та OPA, конфігурації Falco й агентів логування. Під час змін у репозиторії запускаються етапи лінтингу, перевірки маніфестів та сканування безпеки, після чого успішно перевірені конфігурації доставляються до кластера. Таким чином, одна й та сама логіка безпеки реалізується як на етапі розробки, так і на етапі виконання, а результати роботи всієї системи можна побачити через Elastic Stack.

У підсумку архітектура тестового середовища представляє собою цілісну інтегровану систему:

- Kubernetes-кластер на базі minikube як платформа для розгортання навантажень;
- Kyverno та OPA Gatekeeper як шар динамічного застосування політик;
- Falco з eBPF як шар поведінкового моніторингу контейнерів;
- Elastic Stack як шар аналітики, візуалізації та сповіщень;
- CI/CD-зв’язка як механізм керування конфігураціями та політиками у форматі «як код».

Саме в такій конфігурації система використовується далі для реалізації конкретних політик, моделювання атак і кількісної оцінки ефективності підходу.

3.2 Реалізація системи динамічного застосування політик і виявлення аномалій у Kubernetes-кластері

Для реалізації запропонованої системи було розгорнуто локальне тестове середовище на базі одновузлового кластера Kubernetes, створеного за допомогою Minikube в операційній системі Rocky Linux 9.6 (Рисунок 3.4). Minikube дає змогу швидко розгорнути кластер Kubernetes на локальній машині для цілей розроблення й тестування, що спрощує проведення експериментів та відлагодження компонентів безпеки. Rocky Linux 9.6 обрано як базову операційну систему хоста завдяки стабільності дистрибутива та сучасній версії ядра Linux, яка підтримує використання eBPF-технологій, необхідних для роботи системи Falco. У цьому середовищі було встановлено Minikube та ініціалізовано кластер Kubernetes.

```
[root@rockyserver ~]# minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Рисунок 3.4 – Стан кластера minikube.

У межах кластера Minikube були створені простори імен для розділення функціональних компонентів. У цих просторах імен розгорнуто Kyverno, OPA Gatekeeper, Falco та агенти Beats. Така організація забезпечила можливість ізолювати логіку політик, збір телеметрії та реалізацію механізмів виявлення аномалій, а також мінімізувати вплив експериментальних змін на базову інфраструктуру кластера. Маніфест файл продемонстровано на рисунку 3.5.

Рисунок 3.5 – Маніфест файл для створення простору імен.

Для динамічного застосування політик під час створення й модифікації ресурсів Kubernetes використано два механізми контролю допуску: Kyverno та OPA Gatekeeper. Обидва інструменти інтегруються з API-сервером Kubernetes як веб-хуки перевірки і аналізують запити на створення або зміну об’єктів до моменту їхнього збереження в etcd. Такий підхід дозволяє блокувати потенційно небезпечні або некоректно налаштовані ресурси ще на етапі подання запиту, що відповідає концепції «shift-left» у сфері безпеки.

У межах дослідження було налаштовано політику Kyverno, яка вимагає наявності обов’язкової мітки в кожному новому просторі імен. Політика реалізована у вигляді об’єкта ClusterPolicy з правилом типу validate, яке перевіряє, чи містять метадані Namespace мітку з назвою kyverno. Якщо під час створення простору імен така мітка відсутня, запит відхиляється з відповідним повідомленням про помилку. Конфігурація політики має вигляд:

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-labels
spec:
  validationFailureAction: enforce
  background: true
  rules:
    - name: check-for-labels
      match:
        resources:
          kinds: ["Namespace"]
      validate:
        message: "Простір імен повинен мати мітку `kyverno`."
        pattern:
          metadata:
            labels:
              kyverno: "?*"
```

Параметр validationFailureAction: enforce задає обов’язковість дотримання правила: ресурси, що не відповідають вимозі, не допускаються до кластера. Опція

background: true забезпечує додаткову перевірку вже існуючих просторів імен у фоновому режимі. Шаблон у секції pattern перевіряє наявність мітки kyverno з будь-яким непорожнім значенням.

Open Policy Agent Gatekeeper використано як другий механізм політик, що ґрунтується на рушії OPA та мові запитів Rego. Gatekeeper встановлюється в кластер як набір контролерів і реєструє власний ValidatingWebhook, який отримує копії об'єктів, що створюються або змінюються. На відміну від Kyverno, у Gatekeeper політики описуються в два кроки: спочатку створюється шаблон ConstraintTemplate з Rego-логікою перевірки та описом параметрів, далі на його основі — один або кілька об'єктів типу Constraint, які визначають конкретні вимоги до ресурсів. У роботі було використано шаблон K8sRequiredLabels, що перевіряє наявність обов'язкових міток, після чого створено обмеження, яке вимагає, щоб кожен простір імен містив мітку gatekeeper. У разі відсутності хоча б однієї обов'язкової мітки створення ресурсу блокується, а в логах фіксується порушення. Структурну схему застосування політики на основі ConstraintTemplate і Constraint наведено на рисунку 3.6.

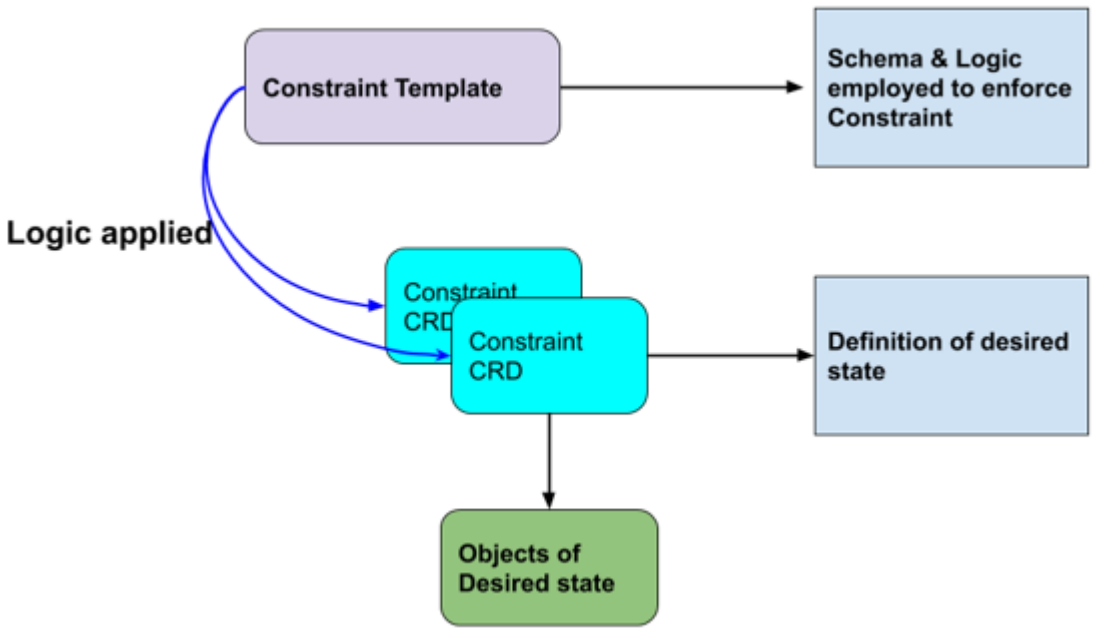

Рисунок 3.6 – Структурна схема застосування політики Gatekeeper.

Спільне використання Kyverno та Gatekeeper в одному кластері дало змогу поєднати переваги обох інструментів. Kyverno застосовано для швидкого опису політик у звичному YAML-форматі, зокрема для простих перевірок наявності міток і базових вимог до конфігурації. Gatekeeper використано для експериментів із більш складними умовами, що потребують виразної логіки на Rego. У процесі налаштування було розмежовано сфери відповідальності політик, що дозволило уникнути конфліктів між інструментами та забезпечити узгоджене застосування вимог безпеки до об’єктів Kubernetes.

Для моніторингу безпеки під час виконання у кластері було розгорнуто систему Falco. Falco було встановлено в кластері у вигляді DaemonSet за допомогою офіційного Helm-чарта. Це забезпечило запуск окремого екземпляра агента на кожному вузлі кластера та уніфіковане керування параметрами конфігурації. У файлі falco.yaml було активовано вивід подій у форматі JSON та увімкнено запис журналу у файл із вказанням шляху до файлу /var/log/falco/falco-events.json. Одночасно з цим збережено вивід подій у стандартний потік stdout, що полегшує початкову діагностику налаштувань.

Окрім типового набору правил, було створено декілька власних правил, адаптованих до специфіки тестового середовища. Зокрема, визначено правило, яке фіксує запуск пакетних менеджерів (apt, dnf) усередині контейнерів, що не є типовим для штатного робочого навантаження й може свідчити про спробу встановлення додаткових утиліт після компрометації. Також було додано правила для відстеження нетипових мережевих з’єднань контейнерів і доступу до привілейованих шляхів файлової системи. Власні правила розміщено в окремому ConfigMap як зображено на рисунку 3.7, який підключено до DaemonSet Falco, що спростило їхню підтримку та модифікацію.

```
kind: ConfigMap
metadata:
  name: falco-custom-rules
  namespace: security-platform
data:
  custom_rules.yaml: |-
    - rule: PackageManagerExecutionInContainer
      desc: >
        Виявлення запуску пакетних менеджерів (apt, apt-get, dnf, yum)
        усередині контейнера.
      condition: >
        spawned_process and container and
        proc.name in ("apt", "apt-get", "dnf", "yum")
      output: >
        [FALCO] Пакетний менеджер у контейнері
        (user=%user.name container=%container.id
        image=%container.image.repository cmdline=%proc.cmdline)
      priority: WARNING
      tags: [container, package_manager, custom]

    - rule: SuspiciousOutboundConnectionToAdminPorts
      desc: >
        Нетипове вихідне мережеве з'єднання з контейнера до адміністративних портів.
      condition: >
        outbound and container and
        evt.type = connect and
        fd.dport in (22, 3306, 5432, 6379)
      output: >
        [FALCO] Нетипове мережеве з'єднання з контейнера
        (user=%user.name container=%container.id
        image=%container.image.repository dst=%fd.sip:%fd.dport)
      priority: WARNING
      tags: [container, network, custom]

    - rule: AccessToPrivilegedFilesystemPaths
      desc: >
        Доступ контейнера до привілейованих шляхів файлової системи
        (/etc, /root, /var/log).
      condition: >
        open_read and container and
        (fd.name startswith "/etc" or
         fd.name startswith "/root" or
         fd.name startswith "/var/log")
      output: >
        [FALCO] Доступ до привілейованих шляхів файлової системи
        (user=%user.name container=%container.id
        image=%container.image.repository file=%fd.name)
      priority: WARNING
      tags: [container, filesystem, custom]
```

Рисунок 3.7 – Приклад власних правил для ConfigMap.

Для централізованого збору та аналізу подій безпеки було реалізовано інтеграцію з Elastic Stack, де Elasticsearch виконує роль сховища даних, та Kibana як інтерфейс для візуалізації й аналітики. У кластері Kubernetes Filebeat і Metricbeat розгорнуто у вигляді DaemonSet, що забезпечує запуск відповідних агентів на кожному вузлі.

Filebeat налаштовано для збирання журналів, пов’язаних із безпекою. Основним джерелом є файл із подіями Falco, який агент читає як вхідний лог-файл типу log. У конфігурації filebeat.yml для цього джерела задано додаткові поля, що дозволяє в подальшому відокремлювати ці події від інших записів у сховищі (Рисунок 3.8). Додатково Filebeat збирає аудиторські журнали kube-apiserver, журнали kubelet і журнали компонентів Kyverno та Gatekeeper. Усі зібрані записи передаються за протоколом Beats до Elasticsearch, де зберігаються в окремих індексах.

```
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /var/log/falco/falco-events.json
    json.keys_under_root: true
    json.add_error_key: true
    fields:
      source: falco
      k8s.security_component: falco
    fields_under_root: true

  - type: log
    enabled: true
    paths:
      - /var/log/containers/*kube-apiserver*.log
      - /var/log/containers/*kubelet*.log
      - /var/log/containers/*kyverno*.log
      - /var/log/containers/*gatekeeper*.log
    processors:
      - add_kubernetes_metadata:
          in_cluster: true
    fields:
      source: k8s-control-plane
    fields_under_root: true

output.elasticsearch:
  hosts:
    - "https://elasticsearch.example.local:9200"
  username: "elastic"
  password: "${ELASTIC_PASSWORD}"
  ssl.verification_mode: "none"
```

Рисунок 3.8 – Фрагмент конфігурації Filebeat.

Metricbeat використано для збирання метрик стану кластера. У конфігурації модуля Kubernetes налаштовано підключення до API kubelet на кожному вузлі (порт 10250) та ввімкнено збирання метрик типів node, system, pod, container, volume. Це

дозволило отримувати дані про завантаженість процесора й пам’яті, кількість та стани подів і контейнерів, використання дискових ресурсів тощо. Для доступу до kubelet Metricbeat використовує службовий обліковий запис із необхідними правами. Зібрані метрики передаються в Elasticsearch до індексів metricbeat-* і надалі відображаються в Kibana. Типову інформаційну панель із показниками роботи вузла Minikube та контейнерів подано на рисунку 3.9.

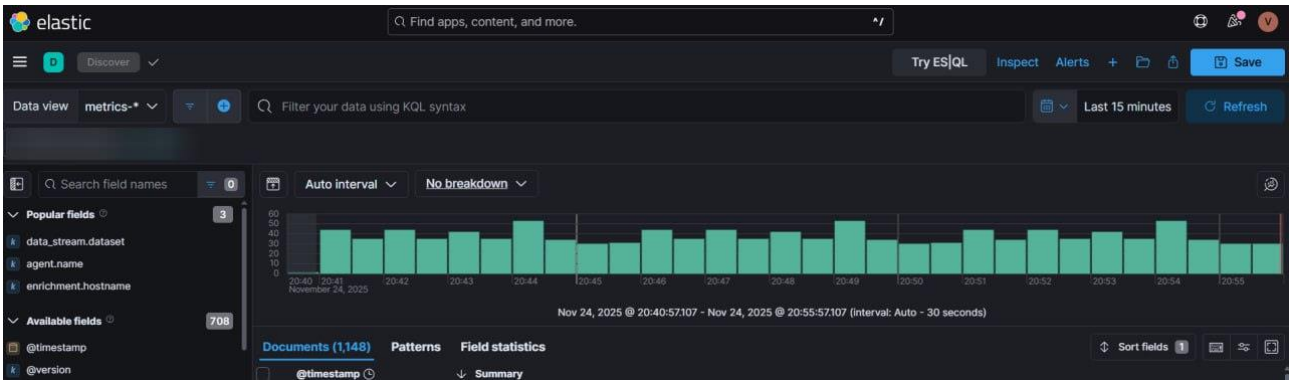

Рисунок 3.9 – Інформаційна панель Kibana.

Після початку надходження даних у Kibana були налаштовані шаблони індексів та інформаційні панелі для подій Falco й метрик Kubernetes. Це забезпечило можливість одночасно спостерігати потік подій безпеки, що генеруються Falco, та відповідні зміни стану інфраструктури, вимірювані Metricbeat. Така інтеграція створила єдиний центр спостереження за безпекою кластера: у разі спрацювання правила Falco відповідний запис одразу відображається в Kibana, а адміністратор може перевірити, чи супроводжується інцидент змінами навантаження або іншими аномаліями. На основі цієї інформації в подальших етапах реалізації системи формуються правила виявлення аномалій і сценарії автоматизованого реагування.

3.3 Моделювання атак та проведення експериментів у тестовому середовищі

Метою моделювання атак у побудованому тестовому середовищі було перевірити узгоджену роботу всіх компонентів інтегрованої системи — механізмів динамічного застосування політик (Kyverno, OPA Gatekeeper), системи виявлення аномалій на рівні ядра (Falco) та підсистеми централізованої аналітики (Elastic Stack). Сценарії атак добиралися таким чином, щоб охопити як помилки конфігурації на рівні Kubernetes-ресурсів, так і зловмисну активність усередині контейнерів, а також забезпечити можливість відтворення кожного експерименту.

Перед моделюванням атак було сформовано базовий фон нормальної роботи системи. У кластері розгорнуто тестові застосунки у просторі імен applications, налаштовано стандартні політики Kyverno та Gatekeeper, увімкнено збір журналів і метрик за допомогою Filebeat та Metricbeat. На цьому етапі Falco працював із типовим набором правил, а в Kibana були побудовані базові інформаційні панелі для відстеження стану вузла, подів і контейнерів, а також для огляду потоку подій Falco. Такий підхід дав змогу виділити «нормальний» профіль подій і метрик, відносно якого надалі оцінювалася аномальна активність.

Подальше моделювання атак виконувалося серією сценаріїв, згрупованих за рівнем, на якому виникає порушення:

- порушення політик безпеки на рівні декларативних конфігурацій Kubernetes;
- зловмисна активність усередині контейнерів під час виконання;
- комбіновані сценарії, де некоректна конфігурація створює передумови для подальшої атаки на рівні виконання.

Кожен сценарій запускався окремо, з фіксацією часу початку та завершення, а між сценаріями проводилося повернення системи до початкового стану. Це дозволило уникнути накладання подій різних експериментів і забезпечити прозорий аналіз результатів для кожного окремого випадку.

Першу групу експериментів присвячено порушенню політик безпеки на рівні Kubernetes-ресурсів (Рисунок 3.10). У цих сценаріях свідомо створювалися або модифікувалися об’єкти, що не відповідали вимогам, які задають Kyverno та OPA Gatekeeper. А саме, було змодельовано:

- спробу створення простору імен без обов’язкової мітки, яку вимагає політика Kyverno;
- спробу створення простору імен без мітки, визначеної в обмеженні Gatekeeper;
- розгортання пода з привілейованим контейнером та доступом до hostPath;
- створення деплойменту без визначених ресурсних лімітів для контейнерів.

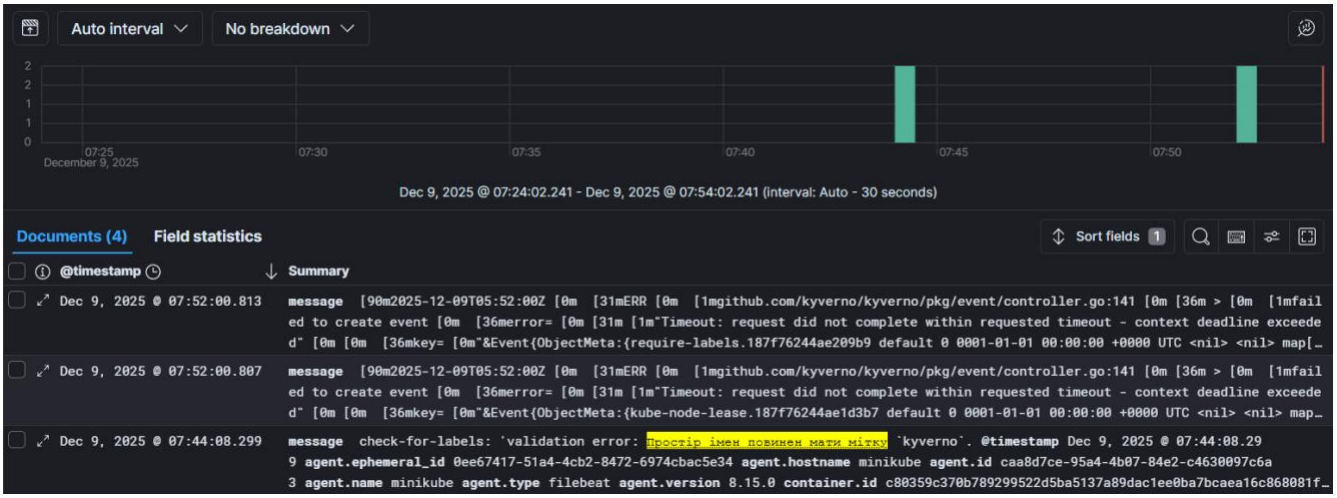

Рисунок 3.10 – Результат логів від час порушння політи безпеки.

У кожному з цих випадків перевірялися дві ключові характеристики роботи системи: коректність реакції admission-контролерів та повнота фіксації події в журналах, які надходять до Filebeat і далі до Elasticsearch. У Kibana для кожного сценарію формувався фільтр за часовим інтервалом експерименту, за типом подій і за відповідними полями (namespace, ім’я ресурсу, тип порушення).

Другу групу експериментів спрямовано на виявлення аномальної активності усередині контейнерів за допомогою Falco (Рисунок 3.11). Для цього в кластері було розгорнуто декілька тестових подів, що імітують компрометовані сервіси. Усередині таких подів виконувалися дії, які в штатному режимі не повинні відбуватися, а саме:

- запуск інтерактивної оболонки контейнера з виконанням команд системного адміністрування;
- запуск пакетних менеджерів (apt, dnf) для встановлення додаткових утиліт;
- спроби доступу до чутливих директорій файлової системи (/etc, /root, /var/log);
- ініціювання нетипових мережевих з'єднань та сканування портів інших подів у кластері.

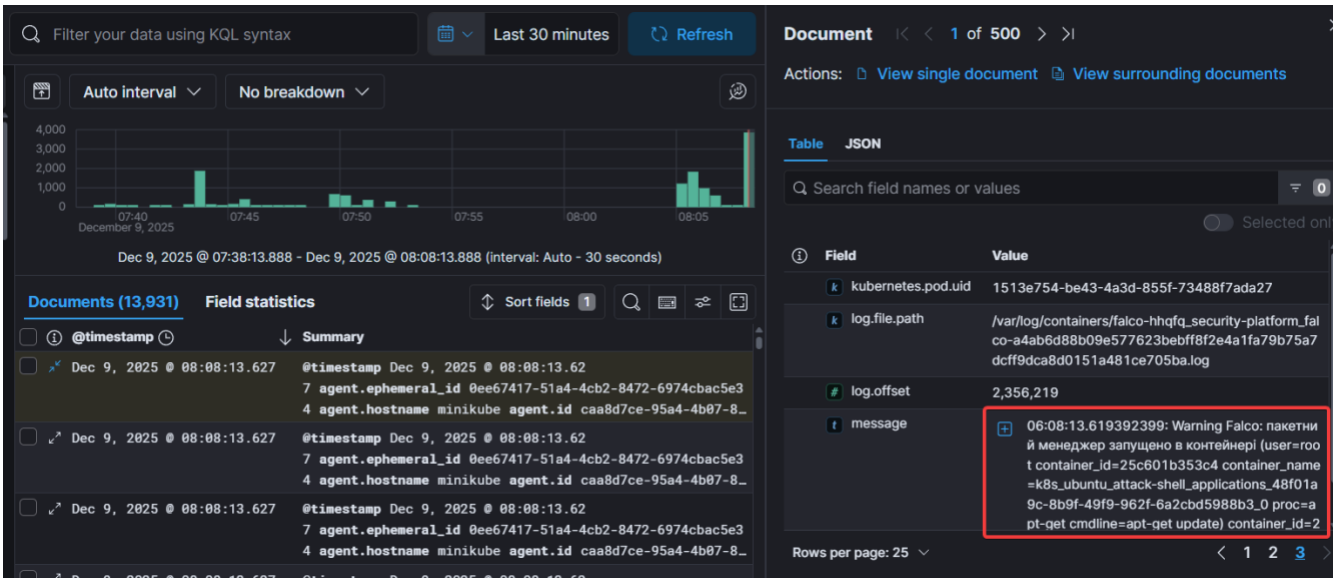

Рисунок 3.11 – Отриманні логи під час порушення політие безпеки всередині контейнера.

Для кожного з перелічених сценаріїв було попередньо підготовлено відповідні правила Falco, включно з власними, доданими до окремого ConfigMap. Під час виконання атаки Falco генерував події, які одночасно потрапляли до stdout

і в JSON-журнал, що зчитується Filebeat. Далі ці події індексувалися в Elasticsearch з додатковими полями, що значно полегшувало фільтрацію та кореляцію.

Окремо було змодельовано мережеві сценарії, пов’язані з розвідкою усередині кластера. У цих експериментах тестовий контейнер виконував спроби послідовного підключення до діапазону портів інших сервісів у просторі applications. Власне правило Falco відстежувало підозрілу частоту мережевих з’єднань з одного контейнера до різних портів за короткий інтервал часу. Паралельно Metricbeat фіксував зміну навантаження на мережу та процесор контейнера, що дало змогу в Kibana побудувати єдину картину — з подією Falco про можливе мережеве сканування та відповідними змінами метрик.

Третю групу експериментів становили комбіновані сценарії, у яких некоректна конфігурація інфраструктури створювала передумови для подальшої атаки на рівні виконання. Наприклад, навмисне послаблення політики дозволяло допустити до кластера под із небажаними параметрами конфігурації. Після цього в контейнері виконувалися дії з другої групи сценаріїв. У результаті в журналах фіксувався повний цикл інциденту: від моменту потрапляння небажаного ресурсу в кластер до спрацювань Falco та змін у метриках інфраструктури. Такі сценарії особливо важливі для демонстрації інтегрованого характеру системи, в якій окремі компоненти не працюють ізольовано, а доповнюють одне одного.

У всіх експериментах застосовувався єдиний підхід до фіксації результатів:

- фіксація початкової конфігурації;
- документування кроків моделювання атаки;
- збір та маркування релевантних подій у Kibana за часовими, просторовими та функціональними ознаками;
- коротке текстове резюме для кожного сценарію з акцентом на тому, який компонент системи першим виявив порушення та як інші компоненти доповнюють картину інциденту.

3.4 Аналіз результатів експериментів та рекомендації щодо впровадження в промислових кластерах

Результати експериментів, проведених у тестовому середовищі, підтвердили доцільність інтегрованого підходу до захисту Kubernetes-кластерів, у якому поєднуються динамічне застосування політик на рівні API-сервера, поведінковий моніторинг контейнерів під час виконання та централізована аналітика подій. Аналіз отриманих спрацювань показав, що кожен із компонентів системи закриває свою частину загроз, а їхня взаємодія дозволяє побудувати цілісну «картину інциденту» від моменту спроби порушення політики до фіксації підозрілої активності на рівні ядра та інфраструктурних метрик.

Під час моделювання порушень політик на рівні Kubernetes-ресурсів Kyverno та OPA Gatekeeper стабільно блокували створення об’єктів, які не відповідали заданим вимогам . Це дозволило в Kibana швидко відфільтровувати події, пов’язані з порушеннями політик, і співставляти їх із діями адміністратора чи розробника, який ініціював створення ресурсу. Важливим спостереженням є те, що застосування політик у режимі enforce практично не впливає на затримку створення ресурсів: у межах тестового середовища реакція контролерів допуску була для користувача близькою до миттєвої, що свідчить про можливість використання такого підходу в промислових кластерах без помітного погіршення продуктивності операцій із Kubernetes API.

Аналіз роботи Falco показав ефективність поведінкового контролю під час виконання, особливо в сценаріях, коли некоректна або надто «м’яка» конфігурація політик допускає до кластера потенційно небезпечні поди. Під час запуску пакетних менеджерів, виконання інтерактивних оболонок, спроб доступу до критичних директорій та симульованого мережевого сканування Falco стабільно генерував події з чітким описом виявленого порушення, контекстом пода й контейнера та часом спрацювання. Після переадресації цих подій до Elasticsearch через Filebeat

вони ставали доступними для аналізу в Kibana практично в режимі реального часу. Це підтвердило, що Falco здатен не лише доповнювати декларативні політики, а й виявляти атаки, які відбуваються вже після успішного створення ресурсу в кластері

У Kibana така послідовність подій дозволяла відстежити, який саме компроміс між зручністю розгортання й жорсткістю політик призвів до появи вразливого ресурсу, та як це відобразилося на поведінці контейнерів. Цей ефект є ключовим для практичного застосування інтегрованої системи.

Узагальнюючи результати проведеного аналізу, можна зробити висновок, що запропонована інтегрована система динамічного застосування політик та автоматичного виявлення аномалій у Kubernetes забезпечує підвищений рівень прозорості та керованості безпеки кластера. Вона може бути адаптована до промислових умов за умови поетапного впровадження політик, ретельного налаштування правил виявлення аномалій і належної організації процесів взаємодії між командами, які відповідають за інфраструктуру та безпеку.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальну задачу підвищення рівня безпеки контейнеризованих середовищ та кластерів Kubernetes шляхом поєднання механізмів динамічного застосування політик та автоматизованого виявлення аномалій. У процесі виконання роботи отримано такі результати:

1. Проведено комплексний аналіз контейнеризованих середовищ та Kubernetes, визначено ключові загрози, вразливості, типові помилки конфігурацій, особливості моделі порушника та обмеження вбудованих засобів безпеки. Показано, що статичні налаштування безпеки не відповідають умовам динамічної мікросервісної інфраструктури.

2. Обґрунтовано необхідність динамічного застосування політик безпеки. Сформовано підхід до побудови багаторівневої системи політик, що охоплює рівні кластера, просторів імен та окремих сервісів. Визначено механізми автоматизованої реакції на порушення політик та їхню інтеграцію в робочі процеси Kubernetes.

3. Розроблено та реалізовано інтегровану систему безпеки, яка поєднує динамічний контроль політик з модулем поведінкового аналізу подій на базі Elastic Stack. Налаштовано збір, нормалізацію та кореляцію журналів контейнерів і вузлів, забезпечено можливість виявлення аномальної активності в режимі, наближеному до реального часу.

4. Змодельовано типові атаки на контейнеризовані середовища, зокрема запуск привілейованих контейнерів, спроби доступу до файлової системи хоста, несанкціоновані мережеві з’єднання та аномальні внутрішні процеси в контейнерах. Показано, що запропонована система ефективно фіксує такі події та формує інформативні журнали для подальшого аналізу.

5. Проведено експериментальне оцінювання ефективності системи, що підтвердило можливість підвищення точності та своєчасності виявлення інцидентів у Kubernetes-кластерах, а також доцільність інтеграції політик безпеки з автоматизованою аналітикою поведінкових відхилень.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Docker Documentation [Електронний ресурс]. – Режим доступу: https://docs.docker.com/get-started/overview/ (дата звернення: 10.09.2025)
2. Kubernetes Documentation. What is Kubernetes? [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ (дата звернення: 12.09.2025)
3. Kubernetes Components [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/overview/components/ (дата звернення: 14.09.2025)
4. Kubernetes Security Overview [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/security/overview/ (дата звернення: 16.09.2025)
5. Kubernetes Authentication [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/reference/access-authn-authz/authentication/ (дата звернення: 19.09.2025)
6. Kubernetes Authorization (RBAC) [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/reference/access-authn-authz/authorization/ (дата звернення: 21.09.2025)
7. Kubernetes Audit Logging [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/tasks/debug/debug-cluster/audit/ (дата звернення: 23.09.2025)
8. Pod Security Admission [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/security/pod-security-admission/ (дата звернення: 25.09.2025)
9. Kubernetes Network Policies [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/services-networking/network-policies/ (дата звернення: 28.09.2025)

10. Souppaya M., Morello J., Scarfone K. Application Container Security Guide. NIST SP 800-190 [Електронний ресурс]. – Режим доступу: https://csrc.nist.gov/pubs/sp/800/190/final (дата звернення: 30.09.2025)

11. Rice L. Container Security: Fundamental Technology Concepts that Protect Containerized Applications. – Sebastopol: O'Reilly Media, 2020. – 210 p.

12. CIS Kubernetes Benchmark [Електронний ресурс]. – Режим доступу: https://www.cisecurity.org/benchmark/kubernetes (дата звернення: 02.10.2025)

13. Sysdig Blog. Detecting Cryptomining Attacks "in the Wild" [Електронний ресурс]. – Режим доступу: https://www.sysdig.com/blog/detecting-cryptomining-attacks-in-the-wild/ (дата звернення: 04.10.2025)

14. Controlling Access to the Kubernetes API [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/security/controlling-access/ (дата звернення: 07.10.2025)

15. Using RBAC Authorization [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/reference/access-authn-authz/rbac/ (дата звернення: 09.10.2025)

16. Using ABAC Authorization [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/reference/access-authn-authz/abac/ (дата звернення: 11.10.2025)

17. Kubernetes Admission Controllers [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/ (дата звернення: 13.10.2025)

18. Dynamic Admission Control (Admission Webhooks) [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/ (дата звернення: 15.10.2025)

19. Pod Security Standards [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/concepts/security/pod-security-standards/ (дата звернення: 20.10.2025)

20. GitGuardian Blog. What is Policy-as-Code? An Introduction to Open Policy Agent [Електронний ресурс]. – Режим доступу: https://blog.gitguardian.com/what-is-policy-as-code-an-introduction-to-open-policy-agent/ (дата звернення: 22.10.2025)

21. CNCF TAG Security. Kubernetes Policy Management Whitepaper [Електронний ресурс]. – Режим доступу: https://github.com/kubernetes/sig-security (дата звернення: 25.10.2025)

22. Open Policy Agent Documentation. Introduction [Електронний ресурс]. – Режим доступу: https://openpolicyagent.org/docs/ (дата звернення: 27.10.2025)

23. Open Policy Agent Documentation. Policy Language Rego [Електронний ресурс]. – Режим доступу: https://openpolicyagent.org/docs/policy-language/ (дата звернення: 29.10.2025)

24. Kubernetes Blog. OPA Gatekeeper: Policy and Governance for Kubernetes [Електронний ресурс]. – Режим доступу: https://kubernetes.io/blog/2019/08/06/opa-gatekeeper-policy-and-governance-for-kubernetes/ (дата звернення: 31.10.2025)

25. Kyverno Documentation. Introduction [Електронний ресурс]. – Режим доступу: https://kyverno.io/docs/introduction/ (дата звернення: 03.11.2025)

26. Kyverno Documentation. Policy Types [Електронний ресурс]. – Режим доступу: https://kyverno.io/docs/policy-types/ (дата звернення: 05.11.2025)

27. Kyverno Sample Policies [Електронний ресурс]. – Режим доступу: https://kyverno.io/policies/ (дата звернення: 07.11.2025)

28. CNCF Blog. Policy-as-Code in the Software Supply Chain [Електронний ресурс]. – Режим доступу: https://www.cncf.io/blog/2024/02/14/policy-as-code-in-the-software-supply-chain/ (дата звернення: 09.11.2025)

29. Argo CD Documentation [Електронний ресурс]. – Режим доступу: https://argo-cd.readthedocs.io/ (дата звернення: 12.11.2025)

30. OPA Documentation. Policy Testing [Електронний ресурс]. – Режим доступу: https://www.openpolicyagent.org/docs/latest/policy-testing/ (дата звернення: 14.11.2025)

31. Conftest Documentation [Електронний ресурс]. – Режим доступу: https://www.conftest.dev/ (дата звернення: 16.11.2025)

32. Kyverno CLI Documentation [Електронний ресурс]. – Режим доступу: https://kyverno.io/docs/kyverno-cli/ (дата звернення: 18.11.2025)

33. Kyverno vs OPA Gatekeeper – Kubernetes Policy-as-Code Showdown [Електронний ресурс]. – Режим доступу: https://aws.plainenglish.io/part-3-kyverno-vs-opa-gatekeeper-kubernetes-policy-as-code-showdown-c1a38397c2ae (дата звернення: 21.11.2025)

ДОДАТОК А

Копії публікації

ЗМІСТ

УДК 004.056.5

Віталій КЛІМ, Тарас ЦАВОЛИК

Західноукраїнський національний університет

АРХІТЕКТУРА СИСТЕМИ БЕЗПЕКИ KUBERNETES

Вступ. Стрімке поширення контейнеризації та використання Kubernetes як домінуючої платформи оркестрації призводить до суттєвого ускладнення вимог до кібербезпеки. Динамічність кластерів, автоматичне масштабування, часті оновлення контейнерних образів та велика кількість взаємодіючих мікросервісів створюють середовище, у якому традиційні статичні політики безпеки втрачають ефективність. Контроль доступу, перевірка конфігурацій та моніторинг активності повинні реагувати на зміни у режимі реального часу, інакше в інфраструктурі виникають критичні прогалини безпеки. Сучасні системи DevOps та GitOps вимагають автоматизованого й адаптивного управління політиками, що забезпечує їх узгодженість у всіх середовищах. Динамічне застосування політик стає ключовим елементом захисту Kubernetes, оскільки дозволяє мінімізувати людський фактор, забезпечити відповідність конфігурацій та автоматично блокувати потенційні загрози без втручання адміністратора. Такий підхід дозволяє не лише підвищити рівень безпеки, а й підтримувати стабільність сервісів у системах високого навантаження. Додатково впровадження політик дає можливість стандартизувати процеси безпеки, що особливо важливо при роботі великих команд та у розподілених інфраструктурах.

Мета. Метою дослідження є обґрунтування концепції динамічного застосування політик безпеки у Kubernetes та визначення механізмів, які забезпечують адаптивність, автоматичне реагування на аномалії та підвищення стійкості контейнеризованих інфраструктур. Одним із ключових завдань є встановлення взаємозв'язку між поведінковим моніторингом, автоматизованими інструментами контролю та інтелектуальними системами аналізу загроз, що дозволяють сформувати сучасну модель кіберзахисту.

1. Основи реалізації динамічних політик безпеки.

Механізм політик у Kubernetes реалізується через Admission Controllers – модулі, які перехоплюють запити до API-серверу перед створенням або модифікацією об'єктів. Додаткове розширення можливостей надають інструменти Kyverno та Open Policy Agent (OPA) Gatekeeper, що дозволяють формувати політики на мові декларативних правил [1].

Динамічне застосування політик передбачає зміну правил залежно від:

– подій у системі моніторингу (наприклад, Falco або eBPF-сенсор фіксує аномалію);

– контексту запиту (namespace, роль користувача, тип застосунку);

– аналітики поведінки контейнерів (виявлення нетипових дій, спроб доступу до host-ресурсів).

Така архітектура забезпечує принцип Policy-as-Code, коли всі політики зберігаються у Git-репозиторії та оновлюються автоматично через GitOps-процеси (ArgoCD або Flux). Це мінімізує людський фактор і гарантує узгодженість конфігурацій у всіх середовищах.

2. Інтеграція політик з системами виявлення аномалій.

Динамічні політики безпеки ефективні лише у зв'язці з механізмами моніторингу. Інструменти, що базуються на технології eBPF, дозволяють аналізувати системні виклики ядра й миттєво ініціювати реакцію – наприклад, блокування контейнера, що перевищив дозволені права.

Типовий цикл роботи системи такий:

1. Сенсор Falco фіксує подію (наприклад, exec у привілейованому контейнері);

2. Подія передається у систему автоматизації (Kyverno Controller або OPA Webhook);

3. Динамічна політика змінюється - блокує запуск подібних контейнерів у майбутньому;

4. Зміни синхронізуються у Git-репозиторії, забезпечуючи audit-trace.

Таблиця 1 - Взаємозв'язок між рівнем реагування та типом події

Тип події	Реакція політики	Інструмент реалізації
Підозріла активність у контейнері	Блокування, alert	Falco, Kyverno
Спроба зміни конфігурації	Автоматичне відхилення	OPA Gatekeeper
Вразливий контейнерний образ	Оновлення політики CI/CD	Trivy, ArgoCD
Нетипова мережева активність	Заборона доступу	Cilium, Calico

Висновок. Динамічні політики безпеки є ключовою складовою сучасної моделі захисту Kubernetes. Вони забезпечують автоматичне реагування на зміни у кластері, адаптацію політик у реальному часі, інтеграцію з поведінковим моніторингом та значно зменшують ризики, пов'язані з людським фактором. Завдяки поєднанню Policy-as-Code, eBPF-моніторингу, GitOps-підходів та SIEM-аналітики формуються самонавчальні системи кіберзахисту. У майбутньому очікується інтеграція машинного навчання, яке дозволить прогнозувати загрози та формувати політики автоматично, що зробить Kubernetes платформою з автономним рівнем безпеки. Крім того, розвиток поведінкових моделей дозволить створювати системи, здатні передбачати потенційні атаки та блокувати їх до фактичного здійснення, що істотно підвищить рівень кіберзахисту підприємств.

Перелік використаних джерел.

1. Kyverno Policy Engine for Kubernetes. [Електронний ресурс]. - Режим доступу: https://kyverno.io/

2. Open Policy Agent Gatekeeper. [Електронний ресурс]. - Режим доступу: https://openpolicyagent.org/

3. Falco Runtime Security Project. [Електронний ресурс]. - Режим доступу: https://falco.org/

4. ArgoCD GitOps Continuous Delivery. [Електронний ресурс]. - Режим доступу: https://argo-cd.readthedocs.io/

5. Trivy Vulnerability Scanner. [Електронний ресурс]. - Режим доступу: https://aquasecurity.github.io/trivy/

ЗАХИСТ'ІНФОРМАЦІЇ'2025

матеріали
науково-практичного симпозіуму

2025

ЗМІСТ

УДК 004.056

Віталій КЛІМ, Тарас ЦАВОЛИК

Західноукраїнський національний університет

АРХІТЕКТУРА СИСТЕМИ БЕЗПЕКИ KUBERNETES

Вступ. Сучасні організації дедалі частіше впроваджують контейнеризовані середовища для забезпечення масштабованості, ефективності та автоматизації розгортання додатків.

Однак разом із перевагами контейнеризації з'являються нові вектори атак, які потребують розроблення спеціалізованих рішень безпеки. Kubernetes, як домінуюча система оркестрації контейнерів, вимагає чітко спроєктованої архітектури безпеки, що забезпечує конфіденційність, цілісність і доступність сервісів у динамічному середовищі.

Мета. Метою дослідження є аналіз архітектури системи безпеки Kubernetes, визначення ключових компонентів і механізмів захисту, а також розроблення концептуальної моделі інтегрованої безпеки з урахуванням сучасних викликів кіберзагроз.

1. Компоненти архітектури безпеки Kubernetes

Система безпеки Kubernetes ґрунтується на багаторівневій моделі, що охоплює безпеку кластера, вузлів і контейнерів [1].

Основні її складові:

1. Authentication та Authorization – механізми перевірки автентичності користувачів і сервісних облікових записів із подальшим контролем доступу через Role-Based Access Control (RBAC).

2. Admission Controllers – політики, які перехоплюють запити до API Server та застосовують правила безпеки (наприклад, заборона привілейованих контейнерів, перевірка labels, namespace-ізоляція).

3. Network Policies – контроль взаємодії між Pods через Cilium або Calico згідно з Zero Trust-підходом.

4. Secrets Management – безпечне зберігання та передача конфіденційних даних за допомогою Kubernetes Secrets і інтеграції з зовнішніми KMS (наприклад, HashiCorp Vault).

5. Audit Logging – фіксація усіх подій у кластері для подальшого аналізу та виявлення аномалій.

Важливою умовою ефективного функціонування є інтеграція всіх цих компонентів через єдиний центр моніторингу (SIEM або Elastic Stack), що дозволяє здійснювати кореляцію подій у реальному часі.

2. Інтеграція динамічних політик та автоматизація захисту

У сучасних DevOps-процесах статичні налаштування безпеки є недостатніми, оскільки середовище Kubernetes змінюється динамічно. Тому використовуються інструменти Kyverno та OPA (Gatekeeper), які забезпечують політикоорієнтований контроль і автоматичне оновлення правил відповідно до контексту системи [2].

Наприклад, якщо контейнер намагається запуститися з надмірними привілеями, система може автоматично заблокувати цей процес. Додатково інструменти Falco та Tracee, що використовують eBPF, дозволяють відстежувати поведінку контейнерів на рівні ядра, виявляючи спроби несанкціонованого доступу або ескалації прав.

Інтеграція таких рішень у CI/CD-конвеєри реалізує концепцію "security as code", де безпека є невід'ємною частиною життєвого циклу розробки програмного забезпечення. Це забезпечує постійний контроль і мінімізацію ризиків уразливостей на всіх етапах розгортання.

Таблиця 1 – Основні компоненти системи безпеки Kubernetes

Компонент	Призначення	Приклад інструменту
RBAC	Контроль доступу до ресурсів	Kubernetes API Server
Network Policy	Обмеження трафіку між Pods	Calico, Cilium
Admission Controller	Динамічне застосування правил	Kyverno, OPA Gatekeeper
Runtime Security	Моніторинг аномалій	Falco, Tracee

Висновок. Архітектура системи безпеки Kubernetes повинна розглядатися як єдина багаторівнева структура із динамічним застосуванням політик і автоматизованим виявленням аномалій. Поєднання RBAC, Network Policies, Admission Controllers та інструментів на основі eBPF забезпечує комплексний захист кластерів і створює передумови для побудови систем самоадаптивної кібербезпеки в контейнеризованих середовищах.

Подальший розвиток архітектури безпеки передбачає впровадження механізмів машинного навчання для прогнозування потенційних атак і адаптацію політик безпеки до змін середовища в режимі реального часу.

Перелік використаних джерел:

1. Kubernetes Documentation – Security Overview. [Електронний ресурс]. Режим доступу: https://kubernetes.io/docs/concepts/security/

2. Kyverno Policy Engine for Kubernetes. [Електронний ресурс]. Режим доступу: https://kyverno.io/

3. Falco Runtime Security Project. [Електронний ресурс]. Режим доступу: https://falco.org/

4. OPA Gatekeeper – Policy Controller for Kubernetes. [Електронний ресурс]. Режим доступу: https://openpolicyagent.org/

5. HashiCorp Vault – Secrets Management. [Електронний ресурс]. Режим доступу: https://www.vaultproject.io/

ДОДАТОК Б

Лістинг основних правил для виявлення аномальної поведінки

```
- rule: Interactive shell in container
  desc: Виявлення запуску інтерактивної оболонки всередині контейнера
  condition: >
    container
    and spawned_process
    and proc.name in (bash, sh, zsh, ksh)
  output: >
    FALCO ALERT: інтерактивна оболонка в контейнері
    (user=%user.name container_id=%container.id container_name=%container.name proc=%proc.name cmdline=%proc.cmdline)
  priority: WARNING
  tags: [k8s, container, shell, anomaly]

- rule: Package manager in container
  desc: Виявлення запуску пакетних менеджерів
  condition: >
    container
    and spawned_process
    and proc.name in (apt, apt-get, dnf, yum)
  output: >
    FALCO ALERT: запуск пакетного менеджера в контейнері
    (user=%user.name container_id=%container.id container_name=%container.name proc=%proc.name cmdline=%proc.cmdline)
  priority: WARNING
  tags: [k8s, container, package, anomaly]

- rule: Sensitive directory access in container
  desc: Доступ до чутливих директорій
  condition: >
    container
    and evt.type in (open, openat, creat)
    and (fd.name startswith /etc or fd.name startswith /root or fd.name startswith /var/log)
  output: >
    FALCO ALERT: доступ до чутливої директорії в контейнері
    (user=%user.name file=%fd.name cmdline=%proc.cmdline container_id=%container.id container_name=%container.name)
  priority: WARNING
  tags: [k8s, container, filesystem, anomaly]

- rule: Suspicious network activity in container
  desc: Нетипові вихідні мережеві з’єднання з контейнера
  condition: >
    container
    and evt.type = connect
    and fd.type = inet
  output: >
    FALCO ALERT: підозріла мережева активність у контейнері
    (user=%user.name container_id=%container.id container_name=%container.name
     fd.sip=%fd.sip fd.sport=%fd.sport fd.dip=%fd.dip fd.dport=%fd.dport cmdline=%proc.cmdline)
  priority: WARNING
  tags: [k8s, container, network, anomaly]
```