

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

КАФЕДРА СИСТЕМ ТА ТЕХНОЛОГІЙ КІБЕРБЕЗПЕКИ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Технологія інтеграції систем збору та обробки логів і метрик на базі Grafana Alloy»

зі спеціальності

125 Кібербезпека та захист інформації

(код, найменування спеціальності)

освітньо-професійної програми

Інформаційна та кібернетична безпека

(назва програми)

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

Вадим СКИБИЦЬКИЙ

(підпис)

Виконав: здобувач вищої освіти групи БСДМ-61
СКИБИЦЬКИЙ Вадим

(прізвище, ім'я)

Керівник

д.ф. Собчук Андрій

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

Рецензент

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

ЗМІСТ

	Стор.
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	9
ВСТУП	10
1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ІНТЕГРАЦІЇ СИСТЕМ ЗБОРУ ТА ОБРОБКИ ТЕЛЕМЕТРІЇ.....	13
1.1 Дослідження проблеми розрізненості даних.....	13
1.2 Дослідження концепції спостережуваності та її компонентів.....	19
1.3 Дослідження концепції стандартизації збору телеметрії та викликів інтеграції.....	22
2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ІНТЕГРАЦІЇ СИСТЕМ ЗБОРУ ТА ОБРОБКИ ЛОГІВ І МЕТРИК НА БАЗІ GRAFANA ALLOY.....	27
2.1 Аналіз архітектури та історії розвитку Grafana Alloy.....	27
2.2 Аналіз компонентних моделей та декларативної мови конфігурації	31
2.3 Порівняльний аналіз з альтернативними рішеннями.....	40
3 РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ВПРОВАДЖЕННЯ GRAFANA ALLOY	45
3.1 Розробка стратегії розгортання, масштабування та високої доступності для систем збору та обробки телеметрії.....	45
3.2 Розробка ключових моментів при інтеграції Grafana Alloy в інфраструктуру.....	56
3.3 Розробка рекомендацій щодо практичної реалізації пайплайнів збору логів та метрик	68
ВИСНОВКИ	83
ПЕРЕЛІК ПОСИЛАНЬ	85
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)	86

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ACL – Access Control List.

API – Application Programming Interface.

CI/CD – Continuous Integration / Continuous Deployment.

CNCF – Cloud Native Computing Foundation.

CRI – Container Runtime Interface.

ECS – Elastic Common Schema.

ELK – Elasticsearch, Logstash та Kibana.

gRPC – Google Remote Procedure Call.

HA – High Availability.

IaC – Infrastructure as Code.

MQTT – Message Queuing Telemetry Transport.

MTTD – Mean Time To Detection.

MTTR – Mean Time To Resolution.

OPC UA – Open Platform Communications Unified Architecture.

OTel – OpenTelemetry.

OTLP – OpenTelemetry Protocol.

PromQL – Prometheus Query Language.

SDK – Software Development Kit.

SLA – Service Level Agreement.

SNMP – Simple Network Management Protocol.

SRE – Site Reliability Engineering.

WAL – Write-Ahead Logging.

ВСТУП

Актуальність дослідження. Однією з ключових особливостей сучасних інформаційно-технологічних систем є експоненційне зростання їх складності. Цей процес зумовлений постійно зростаючою потребою в переході від монолітних архітектур, де всі компоненти об'єднані в одну неподільну систему, до розподілених моделей, побудованих на мікросервісах, широким використанням технологій контейнеризації та оркестрації, таких як Docker та Kubernetes, а також масовою міграції інфраструктур в орендовані хмарні середовища.

Таке масове розширення кількості сервісів, що використовуються, призводить до збільшення об'єму телеметричних даних – логів, метрик та трасувань. В таких умовах звичні методи моніторингу, орієнтовані на статичні хости та заздалегідь визначені показники є недостатньо ефективними для забезпечення надійності та продуктивності.

Внаслідок такої складності систем з'являється одна з ключових проблем – феномен інформаційних силосів. Цей феномен пояснюється так: інформаційні силоси – це ізольовані колекції даних, які перешкоджають обміну даними між різними відділами, системами та підрозділами. Коли дані стають ізольованими, організаціям може бути складно підтримувати їхню якість і приймати рішення на основі даних. Логи, метрики та трасування часто можуть збиратися, оброблятися та зберігатися за допомогою окремих, неінтегрованих між собою інструментів та стеків, що призводить до збільшення середніх значень MTTD та MTTR інцидентів.

Прикладом малоефективного способу обробки даних може бути ситуація, коли логи надходять до стеку ELK (Elasticsearch, Logstash, Kibana), метрики до Prometheus, а трасування до Jaeger. Така розрізненість створює значні перешкоди для інженерних команд, оскільки унеможливорює отримання цілісної картини стану системи, а також безпосередньо впливає на бізнес-показники та досвід кінцевих користувачів.

Для того щоб усунути цю проблему організації переходять від концепції звичайного моніторингу до більш широкої ідеї спостереження. Моніторинг

відповідає на прості, завчасно визначені питання, такі як: «Яке поточне навантаження процесора?», в той час як спостереження надає можливість поставити несподівані запитання про внутрішній стан системи, наприклад: «Чи перевищує навантаження процесора 90 %?», спираючись на зовнішні вихідні дані – телеметрію. Це не лише допомагає з'ясувати, що сталося, а й дозволяє ретельно проаналізувати причини цих подій у складних і динамічних умовах.

Об'єкт дослідження – процес збору, обробки та інтеграції телеметричних даних у сучасних розподілених ІТ-системах.

Предмет дослідження – технологія інтеграції процесів обробки та інтеграції телеметричних даних на базі програмного продукту Grafana Alloy, його архітектура, функціональні можливості та практичне застосування.

Мета роботи – провести аналіз технології інтеграції систем збору та обробки телеметричних даних на базі Grafana Alloy та розробити рекомендації щодо її впровадження для побудови ефективної системи спостережуваності.

Наукові завдання:

дослідити проблему розрізненості телеметричних даних та теоретичні основи концепції спостережуваності;

проаналізувати існуючі підходи та інструменти для збору логів та метрик;

проаналізувати архітектуру, компонентну модель та функціональні особливості Grafana Alloy;

розробити практичний сценарій інтеграції системи збору логів та метрик на базі Grafana Alloy;

сформулювати рекомендації щодо впровадження та експлуатації Grafana Alloy з урахуванням аспектів масштабування, продуктивності та безпеки.

Методи дослідження – опрацювання літератури за даною темою, аналіз експлуатаційної документації, проведення тестів за допомогою наявних інструментів.

Практичне значення одержаних результатів: розробка методичних рекомендацій та практичних прикладів конфігурації, що дозволять інженерним командам ефективно впроваджувати Grafana Alloy для створення уніфікованих

систем спостережуваності, скорочуючи середній час виявлення та вирішення інцидентів у складних мікросервісних архітектурах.

1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ ІНТЕГРАЦІЇ СИСТЕМ ЗБОРУ ТА ОБРОБКИ ТЕЛЕМЕТРІЇ

1.1. Аналіз проблеми розрізненості даних

У сучасних інформаційних системах спостерігається постійне зростання обсягів даних, що надходять з різноманітних джерел: додатків, серверів, мережевого обладнання, контейнеризованих сервісів, а також інфраструктури віртуалізації та хмарних платформ. Кожен компонент такої системи створює унікальний набір журналів подій, метрик і подій, які відображають стан як програмного забезпечення, так і апаратного забезпечення, що задіяне у процесах.

Проте, здебільшого, ця інформація збирається окремо, застосовуючи різноманітні формати, протоколи та критерії, що спричиняє роздробленість області даних. Такий стан розрізненості унеможлиблює повноцінний аналіз, ускладнює створення уніфікованих моделей моніторингу та сповільнює автоматизацію ухвалення рішень у межах підтримки функціональності інформаційних комплексів.

Проблема розрізненості даних у системах збору та обробки логів і метрик має багатофакторний характер. Вона виникає внаслідок технічної, організаційної, семантичної та часової несумісності джерел телеметрії, які розвивалися незалежно одне від одного. Нижче розглянемо основні причини детальніше.

Одна з головних причин розрізненості полягає у великій технологічній різноманітності систем, які формують і передають телеметричні дані.

У межах однієї інфраструктури можуть співіснувати операційні системи різних типів – Linux, Windows, macOS, кожна з яких має власну систему логування (наприклад, journald, Event Viewer або syslog), сервіси та додатки, що використовують різні бібліотеки для запису логів (Log4j, Serilog, NLog, Python logging тощо), мережеве обладнання, яке генерує дані у власних форматах (наприклад, SNMP traps, NetFlow, sFlow) та системи моніторингу, що застосовують

власні агенти та протоколи збору метрик, наприклад, Prometheus exporters, Zabbix agents, Telegraf, Collectd, StatsD.

Через таку розрізненість навіть такі елементарні складові як, наприклад, “час події”, “ідентифікатор вузла” або “рівень критичності” – можуть мати різне представлення (timestamp vs. time, host vs. hostname, severity vs. level).

Інтеграція цих джерел потребує додаткових перетворень і фільтрації даних, що ускладнює побудову єдиного інформаційного простору.

Другим важливим моментом є різноманітність форматів.

Більшість систем логування не мають чітко визначеної схеми запису, що призводить до появи напівструктурованих або неструктурованих даних.

Типові приклади:

Логи системного рівня подаються у вигляді рядків тексту без єдиної структури (наприклад, Jan 01 12:00:01 server1 kernel: CPU overheating).

Інші системи передають JSON-структури (`{"timestamp":"2025-11-04T12:00:01Z", "host":"server1", "event":"CPU overheating"}`).

Метрики часто зберігаються як key-value пари або у форматі time series (`cpu_usage{host="server1"} 0.78 1730726400`).

Навіть у межах однієї платформи формат може змінюватися з оновленнями версій або конфігурацій.

Така варіативність змушує розробників створювати окремі парсери, регулярні вирази або шаблони розбору логів, що підвищує складність обробки й знижує надійність системи.

Не зважаючи на те, що існують різноманітні засоби стандартизації телеметрії, зокрема OpenTelemetry, OpenMetrics, W3C Trace Context та інші, більшість організацій використовує власні методи.

Ця ситуація зумовлена низкою факторів: legacy-рішення, що не підтримують сучасні стандарти, необхідністю у специфічних форматах для окремих галузей (наприклад, промислових або медичних), а також обмеженнями апаратного забезпечення (наприклад, у мережевих пристроях).

Відсутність узгоджених стандартів спричиняє неузгодженість у роботі навіть тих систем, котрі виконують подібні завдання. Наприклад, Prometheus та InfluxDB використовують схожі концепції часових рядів, але механізми збору та отримання цих даних суттєво відрізняються (Pull vs. Push моделі). В наслідок цього, прямий обмін даними між ними виявляється неможливим, якщо не використовувати додатковий шар інтеграції.

Отже, розрізненість даних виникає через цілу низку чинників, починаючи від технічних аспектів і закінчуючи організаційними моментами. Неузгодженість стандартів, різниця в часі передачі даних та відмінності у форматах початкових даних формують уривчасте середовище, де інформацію неможливо використовувати належним чином без попереднього приведення до єдиного вигляду. Здолання цих бар'єрів є обов'язковою умовою для створення єдиної системи збору логів та метрик, що гарантує єдність, узгодженість у часі та достовірність відомостей для аналізу.

Головним наслідком розрізненості даних є нездатність сформувати цілісний погляд на усю інфраструктуру. Це тягне за собою те, що технічні спеціалісти оперують лише фрагментарною інформацією, бачачи логи якогось конкретного сервера чи обмежений набір метрик, але без розуміння ширшого системного контексту. Крім того, виникають затримки у виявленні інцидентів, а також серйозні труднощі із масштабуванням. Адже у відсутності уніфікованої структури даних, нарощування кількості джерел телеметрії обертається неконтрольованим розростанням системи, де збільшення обсягів інформації не підкріплене дієвими методами її обробки.

Розрізненість даних – це не абстрактна концепція, а цілком реальна проблема, що виявляється у процесі роботи практично будь-якого сучасного ІТ-середовища. Найбільш відчутною ця складова стає у масивних, децентралізованих установах, де одночасно працює значна кількість компонентів для збору логів і метрик, які були розроблені різними командами чи інтегровані у різний часовий проміжок.

Одним із найтипівіших прикладів є веб-інфраструктури корпоративного рівня. У таких системах логи веб-серверів, додатків, баз даних і балансувальників

навантаження зберігаються у різних місцях і форматах. Наприклад, журнали доступу веб-сервера Nginx можуть бути записані у звичайних текстових файлах на кожному вузлі, тоді як сервер застосунків передає свої події до централізованого syslog-сервера, а база даних формує власні журнали у внутрішньому форматі. У випадку інциденту, коли спостерігається збільшення часу відповіді або збої у роботі, аналітик стикається з необхідністю вручну зіставити події між різними джерелами. Відсутність узгодженого формату, різниця в часових мітках та семантична неоднозначність полів суттєво ускладнюють встановлення причинно-наслідкових зв'язків. Як наслідок, виявлення першопричини проблеми може тривати годинами, хоча всі необхідні дані формально наявні [1].

Не менш показовим є приклад з експлуатації хмарних та контейнеризованих середовищ, таких як Kubernetes. У таких системах кожен контейнер, вузол кластера й оркестратор мають власні джерела телеметрії. Контейнери генерують логи додатків, kubelet і kube-proxy формують системні повідомлення, а компоненти оркестратора записують події керування. Додатково агенти моніторингу збирають метрики використання ресурсів, таких як процесор, пам'ять і мережевий трафік. Проблема полягає в тому, що ці потоки даних часто не узгоджені між собою. Подія у журналі контейнера може бути зафіксована раніше за метрику, яка її спричинила, або взагалі не мати спільного ідентифікатора, який дозволив би зв'язати їх між собою. У результаті оператору складно відстежити життєвий цикл події: від зміни навантаження до збою в додатку. Така ситуація особливо критична у динамічних середовищах, де контейнери створюються та видаляються протягом хвилин, а джерела логів постійно змінюються.

Проблема розрізненості даних чітко проявляється також у промислових IoT-системах. Наприклад, на підприємствах, де використовується велика кількість сенсорів для збору показників температури, тиску, вологості чи вібрацій, дані можуть надходити через різні протоколи: Modbus, MQTT, OPC UA або навіть через застарілі власні рішення. Частина датчиків передає значення у цифровому вигляді, частина у аналоговому, а деякі пристрої можуть відправляти лише усереднені показники через певні інтервали часу. У таких умовах неможливо одразу

інтегрувати всі потоки у єдине сховище, оскільки відрізняються не лише формати, а й частоти оновлення, одиниці виміру та навіть точність даних. Без уніфікації цієї телеметрії подальший аналіз стає малоефективним: статистичні моделі втрачають точність, а система моніторингу не може коректно виявляти відхилення чи потенційні інциденти.

Ще один практичний сценарій спостерігається в інформаційній безпеці. Системи SIEM покликані об'єднувати логи з різних джерел, таких як сервери, мережеве обладнання, антивіруси, засоби автентифікації тощо. Однак на практиці інтеграція таких джерел часто потребує тривалого ручного налаштування парсерів та правил нормалізації. Наприклад, один міжмережевий екран може маркувати подію як “Intrusion Attempt”, інший як “Attack detected”, а антивірусна система – “Threat found”. Для людини ці записи очевидно пов'язані, але для машини вони є трьома різними типами подій. Без попереднього узгодження семантики даних автоматичний аналіз або кореляція подій стають ненадійними, що знижує ефективність виявлення загроз у реальному часі.

Проблема розрізненості є особливо відчутною у великих корпораціях зі складною IT-архітектурою, де кожний департамент відповідає за свою систему моніторингу. Як приклад, команда розробки використовує власні засоби для агрегації логів та тестування продуктивності, операційний відділ – окремий софт для спостереження за серверами, а відділ кіберзахисту – незалежну SIEM-систему. Через відсутність єдиного підходу до управління телеметрією, комунікація між цими рішеннями відсутня, що породжує ізольовані одне від одного набори даних. Наслідком є те, що навіть несуттєві відхилення можуть пройти повз увагу на загальному рівні інфраструктури, оскільки кожна група бачить лише фрагмент загальної ситуації.

Таким чином, у всіх згаданих ситуаціях простежується одна спільна риса – велика кількість окремих джерел, що продукують інформаційні потоки, які, хоч і є логічно поєднаними, проте технічно не узгоджуються між собою. Це, своєю чергою, унеможливорює проведення всебічної аналітики та гальмує швидкість реагування на будь-які інциденти. Саме ці реально спостережувані наслідки

роз'єднаності даних чітко вказують на нагальну потребу у формуванні уніфікованого про шарку для інтеграції. Метою такого шару має стати забезпечення єдності форматів, приведення часових позначок до спільного знаменника та надання належного контексту зібраним телеметричним даним задля їх подальшого опрацювання та дослідження.

Хоча розрізненість даних залишається однією з ключових проблем у побудові телеметричних систем, сучасна практика пропонує кілька напрямів її мінімізації:

- Використання форматів типу JSON-Lines, OpenTelemetry або Parquet для уніфікації логів і метрик дозволяє спростити подальшу аналітику.
- Застосування проміжного рівня (middleware), який агрегує, нормалізує та маршрутизує дані від різних джерел у спільне сховище.
- Кожен запис телеметрії має містити інформацію про джерело, тип події, середовище, версію ПЗ тощо – це дозволяє формувати єдиний контекст для аналізу.

Розрізненість даних у системах збору логів і метрик є однією з головних перешкод для створення ефективної, масштабованої та аналітично спроможної інфраструктури моніторингу.

Вона має комплексний характер – технічний, семантичний і організаційний – та призводить до зниження якості спостережуваності (observability) у сучасних ІТ-середовищах.

Подолання цієї проблеми потребує впровадження єдиної технологічної основи для уніфікації, маршрутизації та зберігання телеметричних даних. Саме така основа є передумовою для побудови інтегрованої системи збору й обробки логів і метрик, що буде розглянуто у наступних розділах.

1.2. Дослідження концепції спостережуваності та її компонентів

Спостережуваність (observability) – це здатність отримувати глибокі відомості про внутрішній устрій системи, що дозволяє організаціям виявляти та усувати проблеми до того, як вони перетворяться на серйозні збої [2].

На відміну від традиційного моніторингу, що довгий період часу вважався основним інструментом для відстеження стану ІТ-систем, сучасні розподілені системи рухаються в бік спостережуваності. Це дозволяє їм виходити за рамки звичного моніторингу, отримуючи інформацію не тільки про відомі проблеми та показники, а й надаючи можливості для вивчення властивостей системи та виявлення закономірностей в ній, що дозволяє реактивно реагувати на проблеми, що можуть бути виявлені найближчим часом.

Основою спостережуваності є три фундаментальні типи телеметричних даних, що часто називають «трима стовпами». Цими компонентами є:

1. Логи – детальна інформація про події та стан системи.
2. Метрики – кількісні показники ефективності.
3. Трасування – відстеження наскрізних запитів у системі [3].

Розглянемо детально кожен з компонентів.

Логи – це детальні записи про конкретні події, що відбулися в системі. Вони є найстарішим і найбільш детальним джерелом інформації, що надають поглиблений контекст про події, надаючи дані про те, чому сталась та чи інша подія.

Логи поділяються на два основні види: структуровані та неструктуровані.

Неструктуровані логи – це простий текст, зручний для читання людиною, але його складно використовувати для автоматизованого аналізу подій чи їх пошуку.

Структуровані логи в свою чергу надають організовані дані (зазвичай в форматі JSON) у вигляді пар «ключ-значення», що робить їх зручними та гнучкими для машинного парсингу, пошуку та аналізу.

Метрики – це числові значення, що представляють стан системи в певний момент часу, зібрані через регулярні проміжки. Вони зберігаються в форматі часових рядів та підходять для загального моніторингу системи, виявлення аномалій, трендів та налаштування автоматичних сповіщень. Аналіз метрик надає можливість проаналізувати, що відбувається в системі.

Основні типи метрик [4]:

1. Лічильники – значення, що монотонно зростають, наприклад, кількість оброблених запитів та помилок.
2. Вимірювачі – значення, що можуть як зростати, так і спадати, наприклад, поточне використання пам'яті або кількість активних з'єднань.
3. Гістограми – вимірюють розподіл значень у певних інтервалах, наприклад, час, необхідний на обробку конкретного запиту.

Головним недоліком метрик є те, що вони надають конкретне уявлення про проблему, але не надають детального контексту про її причину.

Трасування – це метод, що дозволяє відстежувати повний шлях одного запиту через усі компоненти розподіленої системи. Воно дозволяє отримати візуальні залежності між сервісами, вимірюючи затримки на кожному етапі та надаючи можливість знаходити місця, де в системі виникла проблема.

Основними елементами трасування є:

1. Trace ID: унікальний ідентифікатор для кожного запиту.
2. Spans: окремі операції в рамках трейсу.
3. Context: інформація про взаємозв'язки між спанами.
4. Візуалізація: представлення шляху запиту через систему [3].



Рис. 1.1 Три стовпи спостережуваності

Головна відмінність спостережуваності від моніторингу розкривається не в ізолюваному використанні кожного з елементів, а в можливості їх інтегрувати та доповнювати одне одним. Кожен зі стовпів надає унікальну інформацію про систему, що, комбінуючись, дозволяє проаналізувати проблему повністю.

Шлях, за яким працює спостережуваність, зазвичай виглядає так [5]:

1. Вперше проблема визначається завдяки метрикам. Система моніторингу може спрацювати, сповіщаючи про аномалію: різке зростання частоти помилок, збільшення часу відповіді, або падіння пропускну здатності для певного сервісу. Оскільки метрики дають швидкий, високорівневий сигнал про те, що щось пішло не так, вони дозволяють локалізувати проблему до певного сервісу або часового проміжку. Однак, вони рідко пояснюють, чому це сталося.

2. Маючи інформацію від метрик, переходимо до аналізу розподілених трасувань за цей період часу. Трасування показує повний шлях запитів через усі залучені мікросервіси. Аналізуючи окремі span'и в межах повільних або невдалих трас, можна точно визначити, де саме виникла проблема – який конкретний виклик сервісу, запит до бази даних чи інша операція спричиняє затримку або помилку.

3. Визначивши конкретний span, де виникла проблема, використовуються логи для отримання детального контексту. Сучасні системи спостережуваності дозволяють корелювати логи з трасуванням за допомогою унікальних ідентифікаторів (`trace_id`, `span_id`), які додаються до лог-записів. Це дає можливість миттєво відфільтрувати лише ті лог-повідомлення, що стосуються конкретної проблемної операції в межах конкретного запиту. Саме в логах знаходиться відповідь на питання "чому": детальне повідомлення про помилку, `stack trace`, значення змінних, несподівані вхідні дані або будь-яка інша інформація, яку розробник передбачив залогувати.

Цей плавний перехід між метриками, трасуванням та логами можливий лише за умови їхньої кореляції. Додавання ідентифікаторів трасування до логів та метрик, а також можливість легко переходити між різними типами даних в інтерфейсі платформи спостережуваності є ключовою особливістю спостережуваності. Цей зв'язок між різними типами телеметрії створює контекст, і саме він перетворює розрізнені дані на дані про стан системи.

Таким чином, синергія компонентів спостережуваності дозволяє командам швидко й ефективно рухатись від виявлення симптому (метрики) до локалізації проблеми (трасування) і, нарешті, до розуміння її першопричини (логи).

1.3. Дослідження концепції стандартизації збору телеметрії та викликів інтеграції

Постійне збільшення об'ємів телеметричних даних у сучасних ІТ-системах призвело до необхідності розроблення єдиних підходів до їх збору, передачі та обробки. Логи, метрики, трасування запитів і події безпеки утворюють комплексне середовище для спостереження, де ефективність аналізу безпосередньо залежить від рівня стандартизації форматів і протоколів обміну інформацією.

Без спільних стандартів дані залишаються розрізненими, а процес інтеграції між різними платформами перетворюється на складну інженерну задачу. Саме тому концепція стандартизації збору телеметрії стала ключовим напрямом розвитку систем моніторингу, аналітики та автоматизації активів інфраструктури.

Під стандартизацією збору телеметрії розуміють формування єдиних принципів опису, структурування та передачі даних про стан систем і процесів. Це включає не лише визначення форматів і протоколів, а й розробку уніфікованих моделей даних, схем метаданих і політик часової синхронізації.

Завдання стандартизації полягає в тому, щоб телеметрична інформація, отримана з різних частин ІТ-інфраструктури, була взаємозамінною, уніфікованою та здатною до спільного функціонування (інтероперабельною). Якщо говорити інакше, немає значення, який саме елемент генерує цю інформацію, чи то серверний сервіс, мережевий комутатор, чи ізольований контейнер – усі ці джерела зобов'язані надавати телеметрію у формі, що дозволяє будь-якому наступному інструменту аналізу чи відображення даних однозначно її зрозуміти.

Одним із базових напрямів стандартизації є уніфікація форматів даних. Для логів це означає перехід від довільних текстових записів до структурованих форматів, таких як JSON або Protobuf. Для метрик застосовуються формати часових рядів із чітко визначеними назвами, мітками та одиницями виміру. Уніфікований формат дозволяє застосовувати єдині механізми парсингу, зберігання та фільтрації, що значно спрощує інтеграцію між компонентами.

Іншим важливим напрямом є стандартизація протоколів обміну. Серед найбільш поширених ініціатив варто виділити OpenTelemetry, OpenMetrics, Prometheus exposition format, W3C Trace Context та Elastic Common Schema (ECS) [8]. Вони визначають правила передачі телеметрії між агентами, брокерами та системами обробки, забезпечуючи єдиний контекст подій, узгодження часових міток і підтримку розподіленого трасування.

Таблиця 1.1.

Порівняння таблиця стандартів обміну телеметрією

Стандарт / ініціатива	Типи телеметрії	Основна мета	Ключові особливості	Підтримувані інструменти / екосистема
OpenTelemetry (OTel)	Логи, метрики, траси	Уніфікація збору та передачі телеметрії	Відкрита специфікація; SDK для більшості мов; спільна модель даних; підтримка розподіленого трасування	Grafana Alloy, Jaeger, Prometheus, Tempo, Loki
OpenMetrics	Метрики	Стандартизація формату експозиції метрик	Ґрунтується на Prometheus exposition format; акцент на часових рядах і мітках; простота інтеграції	Prometheus, VictoriaMetrics, Thanos

Таблиця 1.1. (Продовження)

Elastic Common Schema (ECS)	Логи, події безпеки	Уніфікація полів логів для спільної аналітики	Визначений набір стандартних полів (host.name, event.action, user.id); сумісність із JSON	Elastic Stack (Elasticsearch, Logstash, Kibana)
------------------------------------	---------------------	---	---	---

W3C Trace Context	Траси (розподілені запити)	Стандартизація контексту трасування між сервісами	Визначає заголовки traceparent та tracestate; забезпечує кореляцію запитів у мікросервісах	OpenTelemetry, Zipkin, Jaeger
StatsD / DogStatsD	Метрики	Спрощена передача числових показників	Легкий UDP-протокол; обмежений набір типів даних; відсутність контексту	Datadog, Telegraf, Graphite

Аналіз наведених стандартів показує, що тенденція розвитку систем збору телеметрії рухається у напрямі інтеграції різних типів даних у спільну модель. Найбільш повним і гнучким рішенням наразі є OpenTelemetry, яке об'єднує концепції OpenMetrics і W3C Trace Context. Саме цей підхід лежить в основі таких інтеграційних технологій, як Grafana Alloy, що демонструють практичну реалізацію принципів стандартизації у масштабованих середовищах [6].

Зокрема, OpenTelemetry сьогодні є наймасштабнішою та найперспективнішою ініціативою у сфері стандартизації збору телеметрії. Вона об'єднує під одним API збір логів, метрик і трасувань, надає відкриту специфікацію для інструментів різних постачальників і створює універсальний формат передачі даних через мережу.

На базі цієї концепції формується ціла екосистема інструментів, сумісних між собою без додаткових адаптерів. Одним із сучасних прикладів такої реалізації є Grafana Alloy, інтеграційна технологія, яка використовує OpenTelemetry Collector як основу для збору, нормалізації та маршрутизації телеметрії. Вона демонструє практичну реалізацію принципів стандартизації: підтримку уніфікованих форматів даних, єдиний конвеєр обробки логів і метрик, а також сумісність із різними системами моніторингу. Таким чином, Grafana Alloy уособлює еволюцію від розрізнених агентів збору даних до стандартизованої платформи інтеграції.

Окремим напрямом є стандартизація семантики даних. Вона передбачає уніфікацію назв параметрів, класифікацію подій за рівнем важливості, типом ресурсу або контекстом виконання. Наприклад, у межах OpenTelemetry існують

семантичні конвенції – стандартизовані імена для метрик (`cpu.usage`, `memory.available`), атрибутів запитів (`http.status_code`, `db.system`) та ідентифікаторів компонентів. Завдяки цьому дані з різних джерел можуть бути інтерпретовані у спільному контексті без втрати змісту.

бути інтерпретовані у спільному контексті без втрати змісту.

Таким чином, сучасна тенденція стандартизації телеметрії спрямована на створення універсальної екосистеми збору та аналізу даних, де різні інструменти можуть працювати спільно, використовуючи відкриті протоколи та спільні формати опису подій.

Попри очевидні переваги, впровадження стандартів збору телеметрії супроводжується низкою викликів, які мають як технічний, так і організаційний характер.

Одним із ключових викликів є спадковість. Багато підприємств мають системи, розроблені десятиліття тому, які не підтримують сучасні формати або протоколи. Їх модернізація потребує значних ресурсів, тому часто доводиться розробляти проміжні адаптери чи шлюзи, що виконують перетворення даних до уніфікованого вигляду. Це збільшує складність архітектури та створює додаткові точки потенційних відмов.

Другий виклик – різноманітність технологічних стеків. Сучасні ІТ-системи поєднують мікросервісну, контейнерну, серверну та хмарну архітектури, кожна з яких має власні інструменти спостереження. У результаті виникає необхідність у створенні єдиної інтеграційної шини, здатної приймати дані у різних форматах, з різною частотою та затримками.

Третім суттєвим викликом є питання продуктивності та обсягів даних. Стандартизація часто передбачає збір ширшого набору метрик і контекстуальних атрибутів, що збільшує навантаження на мережу, дискові ресурси та системи обробки. У великих розподілених середовищах обсяг телеметрії може сягати терабайтів на добу, тому потрібні високопродуктивні механізми агрегації, компресії та маршрутизації.

Не менш важливим є виклик семантичної узгодженості. Навіть при використанні стандартів, різні команди або розробники можуть по-різному інтерпретувати значення полів чи рівні критичності подій. Це породжує ситуації, коли формально стандартизовані дані залишаються неповністю узгодженими. Для подолання цього потрібна чітка політика керування метаданими та контроль якості телеметрії.

Окрім технічних аспектів, існують і організаційні виклики. Впровадження стандартів потребує узгодження між різними командами, зміни процесів розробки та експлуатації, а також навчання персоналу. Часто стандартизація сприймається як додаткове навантаження, що уповільнює впровадження нових функцій, тому виникає опір з боку розробників або адміністраторів. Вирішення цих проблем вимагає поступового підходу, поетапної інтеграції стандартів і демонстрації їхньої практичної користі.

Впровадження стандартів у сфері телеметрії є основою для побудови інтегрованих платформ збору, зберігання та аналізу даних. Завдяки єдиним форматам і протоколам забезпечується прозорий обмін інформацією між компонентами, незалежно від постачальника чи архітектури. Це відкриває можливості для централізації спостереження, автоматизованого виявлення аномалій, глибокої аналітики та прогнозування стану систем.

2 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ІНТЕГРАЦІЇ СИСТЕМ ЗБОРУ ТА ОБРОБКИ ЛОГІВ І МЕТРИК НА БАЗІ GRAFANA ALLOY

2.1. Аналіз архітектури та історії розвитку Grafana Alloy

Grafana Alloy – це сучасна інтеграційна платформа для збору, обробки та маршрутизації телеметрії, розроблена компанією Grafana Labs як частина Grafana Observability Stack. Її поява стала логічним етапом еволюції інструментів моніторингу, які до цього часу розвивалися окремими напрямками: окремо для логів (Grafana Loki), метрик (Prometheus, Mimir) та трасувань (Tempo). Впровадження Grafana Alloy дозволило об'єднати ці компоненти в єдину інтеграційну систему збору телеметрії, що забезпечує уніфікований підхід до обробки даних та їх подальшого аналізу.

Історично Grafana Alloy є продовженням розвитку OpenTelemetry Collector – відкритого стандарту, розробленого спільнотою CNCF для забезпечення сумісності між різними системами збору телеметрії. Хоча OpenTelemetry забезпечував універсальність, його конфігурація вимагала значних зусиль, а впровадження часто супроводжувалося складністю узгодження компонентів. Компанія Grafana Labs, маючи великий досвід у створенні рішень для спостережуваності, розробила власну модифікацію колектора з розширеною функціональністю, покращеною керованістю та інтеграцією з власними сервісами – цим рішенням і стала Grafana Alloy.

Однією з ключових відмінностей Grafana Alloy від попередніх поколінь колекторів є декларативна архітектура [10]. Усі компоненти системи визначаються у вигляді декларацій у конфігураційних файлах (переважно у форматі YAML). Користувач описує не окремі кроки виконання, а структуру потоку даних (pipeline), що проходить через набір модулів – від приймання телеметрії до її збереження чи відправлення. Такий підхід спрощує адміністрування, дозволяє швидко змінювати схеми збору даних та інтегрувати нові джерела телеметрії без перезапуску всієї системи.

Архітектура Grafana Alloy побудована за трирівневою моделлю обробки телеметрії, що складається з рівнів приймання (receiving), обробки (processing) та експорту (exporting) [9].

Кожен рівень представлений набором незалежних компонентів, які взаємодіють між собою через уніфіковані інтерфейси даних.

1. Рівень приймання (Receiver Layer)

Цей рівень відповідає за отримання даних з різноманітних джерел. Alloy підтримує десятки типів receivers, які можуть збирати системні логи з journald, syslog або файлів, метрики з Prometheus endpoints (HTTP scrape), трасування через OTLP [7], Jaeger або Zipkin-протоколи, дані з контейнерних середовищ (Docker, Kubernetes), події з агентів або SDK OpenTelemetry.

Receivers працюють у асинхронному режимі, що дозволяє обробляти великий потік вхідних даних без втрати продуктивності.

Вони також можуть бути мультипротокольними, тобто приймати одночасно різні типи телеметрії, що особливо корисно для мікросервісних систем, де лог і метрика можуть генеруватись у межах одного контейнера.

2. Рівень обробки (Processor Layer)

На цьому рівні здійснюється перетворення, нормалізація та фільтрація даних. Саме процесори формують логіку того, як обробляється телеметрія перед передачею у сховище. Основні типи процесорів:

- Batch Processor – об'єднує потоки подій у пакети для ефективнішого експорту;
- Attributes Processor – додає або змінює атрибути записів (наприклад, хост, датацентр, сервіс);
- Filter Processor – видаляє непотрібні або тестові дані;
- Metrics Transform Processor – виконує агрегацію, обчислення похідних метрик;
- Resource Processor – додає контекстні метадані (версія сервісу, середовище, регіон).

Кожен процесор може бути використаний у кількох пайплайнах одночасно, що дозволяє гнучко налаштовувати потоки даних. Наприклад, логи з вебсерверів можуть проходити через фільтрацію та нормалізацію, тоді як метрики – через агрегацію, перш ніж потрапити до сховища.

3. Рівень експорту (*Exporter Layer*)

Останній рівень відповідає за передачу обробленої телеметрії до систем зберігання чи аналітики.

Grafana Alloy має готові експортери для більшості популярних інструментів спостережуваності:

- Loki Exporter – передає структуровані логи у Grafana Loki;
- OTLP Exporter – універсальний експортер для будь-яких систем, сумісних з OpenTelemetry;
- Prometheus Remote Write – передача метрик у Prometheus або Mimir;
- Tempo Exporter – відправлення трасувань у Grafana Tempo;
- File / HTTP Exporters – для збереження або тестового аналізу локально.

Alloy може одночасно експортувати ті самі дані у кілька напрямів, наприклад, у Prometheus і Grafana Cloud, що забезпечує резервування та аналітичну гнучкість.

Окрім трьох базових шарів, архітектура Grafana Alloy включає допоміжні рівні, які забезпечують керованість, масштабованість і спостережуваність самої системи:

Control Layer (керування) – відповідає за централізоване розгортання, перевірку конфігурацій та моніторинг стану агентів. Alloy підтримує оновлення без простоїв і має вбудовані health-check ендпоїнти.

Configuration Layer – реалізує декларативне керування через YAML, що дає змогу визначати складні топології збору телеметрії як код (IaC-підхід).

Observability Layer – надає власні метрики про стан системи Alloy (кількість прийнятих, оброблених, відхилених записів, затримки, час експорту тощо), що дозволяє здійснювати самоспостереження (self-monitoring).

Повний цикл обробки телеметрії в Grafana Alloy можна описати як послідовність:

1. Receiver отримує дані з джерела (наприклад, логи з journald або метрики з Prometheus).
2. Processor виконує попередню обробку: очищення, нормалізацію, додавання атрибутів.
3. Exporter передає результати у цільову систему: Loki, Mimir, Tempo чи зовнішню платформу моніторингу.

Кожен етап працює незалежно, що дозволяє реалізовувати різні конвеєри в межах одного агента, наприклад, окремий пайплайн для логів та метрик, які потім зливаються у спільний дашборд Grafana.

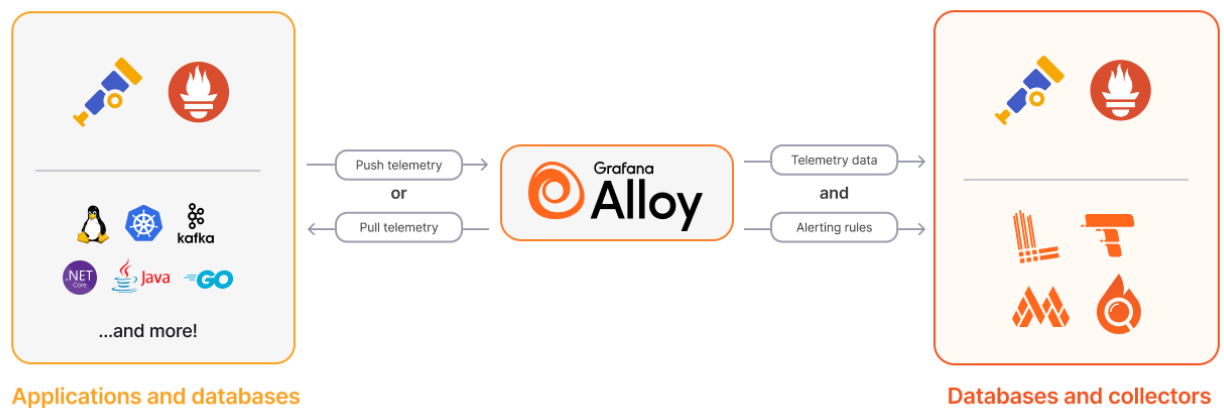


Рис. 2.1. Трирівнева модель Grafana Alloy

Особливої уваги заслуговує інтеграція Grafana Alloy з екосистемою Grafana. Вона підтримує пряме підключення до Loki для логів, Tempo для трасувань і Mimir або Prometheus для метрик. Це дозволяє використовувати Alloy як універсальний шар обробки телеметрії, який автоматично маршрутизує дані у відповідні системи, забезпечуючи спільний контекст і узгоджені часові мітки. Такий підхід суттєво спрощує налаштування комплексної спостережуваності, оскільки всі дані можуть бути візуалізовані у Grafana без додаткових адаптерів.

З технічного погляду Grafana Alloy підтримує високу продуктивність і розподіленість. Система побудована з урахуванням принципів горизонтального масштабування: кілька екземплярів колектора можуть працювати паралельно з балансуванням навантаження, а у випадку відмови одного вузла трафік

автоматично перенаправляється на інший. Це дозволяє забезпечити високу доступність (High Availability) та відмовостійкість, що є критичним для великих розподілених середовищ.

Ще однією сильною стороною Grafana Alloy є її сумісність із відкритими стандартами. Вона повністю підтримує OTLP-протокол (OpenTelemetry Protocol), що забезпечує взаємодію з будь-якими сумісними клієнтами чи агентами. Крім того, Alloy може приймати дані у форматах JSON, Syslog, Prometheus exposition format, що дозволяє інтегруватися навіть зі старими системами моніторингу.

Таким чином, Grafana Alloy можна охарактеризувати як уніфікований інтеграційний рушій для телеметрії, що об'єднує різноманітні потоки даних у єдину систему. Її архітектура поєднує гнучкість OpenTelemetry, простоту конфігурування Promtail та масштабованість сучасних хмарних сервісів. Завдяки цьому Alloy виступає центральним елементом побудови спостережуваності, який спрощує керування телеметрією, мінімізує дублювання агентів і забезпечує цілісне бачення стану IT-інфраструктури.

2.2. Аналіз компонентних моделей та декларативної мови конфігурації

Компонентна (модульна) модель організації потоків телеметрії розглядає систему як набір автономних блоків, кожен з яких виконує чітку функцію: прийом даних (receiver), обробку/перетворення (processor) або відправку/збереження (exporter). Пайплайн (pipeline) – це композиція таких блоків у заданому порядку або у вигляді графа, де дані можуть кілька разів розгалужуватись (fan-out) або зливатися (fan-in). Такий поділ відповідає принципу «одна відповідальність – один модуль» і робить систему гнучкою, тестованою та розширюваною.

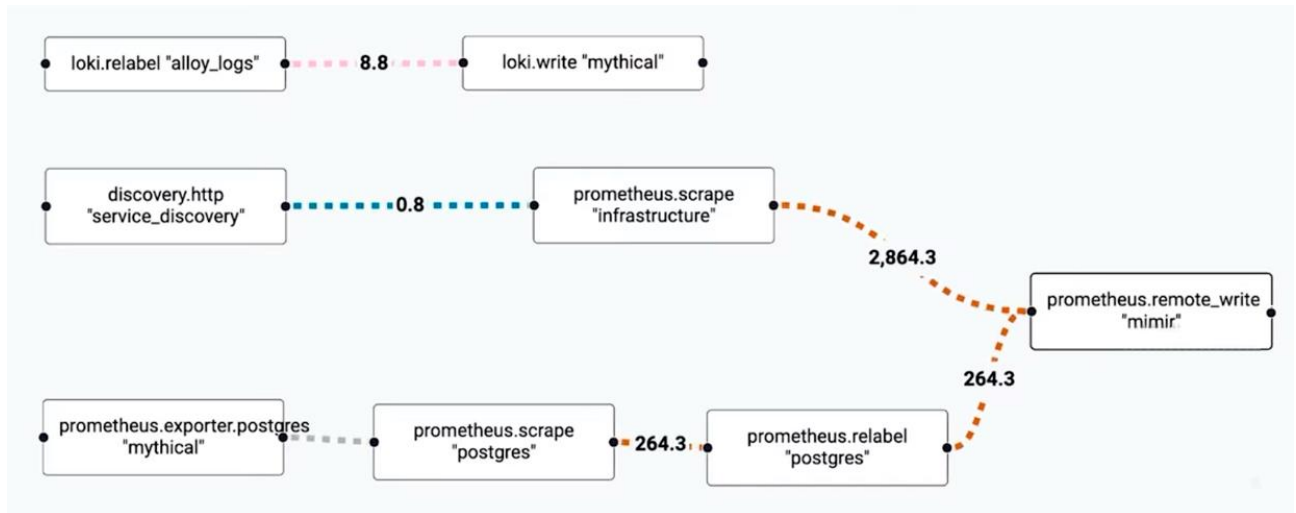


Рис. 2.2. Графічне зображення роботи модулів

Розглянемо детально роботу призначення кожного з компонентів.

Receivers відповідають за взаємодію з джерелом телеметрії: отримують сирі події, повідомлення, метрики чи траси і перетворюють їх у внутрішнє представлення платформи. Receivers зазвичай працюють асинхронно, буферизують вхідні події та видають їх у вигляді потоків у processor layer. Вони мають механізми backpressure (притискання) або local buffering для уникнення втрат при пікових навантаженнях. Також receivers часто виконують базовий парсинг (наприклад JSON-розбір, tokenization) і первинну нормалізацію (додавання часових міток, визначення source/host).

Processors забезпечують семантичну підготовку даних: нормалізацію, фільтрацію, агрегацію, збагачення метаданими і т.д. Вони реалізують основну «логіку» пайплайна. Processors можна комбінувати, поміщати у різні пайплайни, застосовувати умовно (наприклад: якщо service == "auth", можна додати певні атрибути). Завдяки цьому одна й та сама логіка обробки легко перевикористовується та тестується окремо.

Exporters передають оброблені дані в цільові системи зберігання або аналітики: time-series DB, лог-сховища, SIEM, хмарні сервіси. Exporters звичайно підтримують retry-політики, backoff, підтвердження доставки і, за потреби, персистентне локальне буферування (disk queue) для забезпечення доставки при тимчасових збоях мережі/бекенду.

У структурі Grafana Alloy поняття пайплайна (pipeline) відіграє центральну роль, оскільки саме він визначає логіку руху телеметрії від моменту її збору до збереження чи аналізу в системі призначення. Пайплайн – це не просто послідовність компонентів, а цілісний маршрут обробки даних, який може мати різний ступінь складності, від простого лінійного до динамічно розгалуженого з умовною маршрутизацією. Завдяки такій гнучкій архітектурі Grafana Alloy здатна адаптуватися під найрізноманітніші сценарії збору логів, метрик і трасувань у сучасних мікросервісних середовищах.

Найпростішою формою побудови пайплайна є однорядковий або лінійний потік, у якому дані проходять послідовно через усі етапи – приймання, обробку та експорт. У цьому випадку телеметрія, отримана від певного джерела (наприклад, метрики з Prometheus або логи з journald), спочатку надходить до компонента receiver, далі передається до одного чи кількох processor для фільтрації, нормалізації чи збагачення атрибутами, після чого потрапляє до exporter, який відправляє її у кінцеву систему – Grafana Loki, Prometheus або Grafana Tempo. Подібні лінійні конвеєри прості в реалізації та зручні для локального моніторингу або невеликих проєктів. Їх перевага полягає у передбачуваності поведінки системи, простоті налагодження та мінімальній затримці при передачі даних. Якщо виникає помилка чи збій у роботі одного з компонентів, її легко локалізувати, оскільки потік має чітку і зрозумілу послідовність.

Однак із розширенням інфраструктури, коли один і той самий потік телеметрії може бути корисним для різних систем спостереження, виникає потреба у розгалужених або fan-out пайплайнах. У такій топології один приймач даних може передавати потоки одночасно кільком процесорам і експортерам. Наприклад, дані, отримані від OTLP-приймача, можуть бути оброблені одним процесором для додавання тегів служби, після чого паралельно спрямовані в Loki для аналізу логів, у S3 для архівного зберігання та у SIEM-систему для контролю безпеки. Таким чином, замість дублювання агентів чи повторного збору даних із різних джерел, Grafana Alloy дозволяє здійснювати одноразовий збір і подальше мультиспрямування телеметрії. Це суттєво зменшує навантаження на

інфраструктуру та забезпечує постійність даних – усі системи отримують однакові оброблені події, що мають узгоджений набір метаданих.

Ще більш гнучким підходом є умовна маршрутизація потоків або *conditional routing*, яка дозволяє вибірково спрямовувати дані залежно від їхніх властивостей, змісту чи контексту. Такий підхід особливо важливий для великих систем, де обсяги телеметрії надзвичайно високі, а дані мають різне призначення. Умовна маршрутизація в Grafana Alloy базується на аналізі атрибутів кожного запису, таких як назва сервісу, рівень важливості повідомлення, середовище виконання або регіон. Завдяки цьому можна створювати складні, але ефективні схеми розподілу потоків. Наприклад, логи рівня «error» можуть автоматично надсилатися до системи безпеки, тоді як звичайні інформаційні повідомлення потрапляють лише до Loki; трейси з сервісу «payment» можуть спрямовуватись у Grafana Tempo, а дані з «frontend», до іншої аналітичної платформи; метрики зі stage-середовища зберігаються у тестовому Prometheus, а production-метрики – у основному кластері. Такі маршрути можуть бути описані декларативно у YAML-конфігурації, що робить систему гнучкою, динамічною та легкою в управлінні без зміни коду.

У практиці великих організацій часто використовується гібридна топологія пайплайнів, де поєднуються елементи лінійних, розгалужених і умовних маршрутів. Це дозволяє створювати складні графи потоків даних, у яких різні типи телеметрії обробляються одночасно, але за власною логікою. Наприклад, єдиний приймач може збирати дані з Kubernetes-кластерів, після чого один потік відфільтровується за рівнем серйозності та спрямовується у Loki, інший агрегує метрики і надсилає їх до Prometheus, а третій виконує вибірку обробку трас і передає їх у Tempo. Такий підхід дозволяє централізувати збір телеметрії, зменшити кількість агентів, забезпечити єдину точку контролю та підвищити узгодженість між різними джерелами даних.

Особливістю архітектури Grafana Alloy є також реалізація механізмів надійності та керування навантаженням усередині пайплайнів. Кожен експортер у системі працює з асинхронними чергами, що гарантує безпеку передачі навіть у випадках тимчасової недоступності цільових систем. Якщо бекенд недосяжний,

дані накопичуються локально, а після відновлення зв'язку відправляються повторно згідно з політикою повторних спроб. У середовищах із великою кількістю експортерів Grafana Alloy забезпечує також балансування навантаження, розподіляючи обробку між кількома вузлами, що значно підвищує продуктивність і стійкість системи.

Завдяки своїй модульній побудові пайплайни Grafana Alloy можна розглядати як направлений граф потоків телеметрії, у якому кожен вузол є незалежним компонентом з власною логікою, метриками та чергами обробки. У конфігураційних файлах YAML такі зв'язки описуються декларативно, що дозволяє легко створювати нові маршрути, додавати або змінювати пункти збору даних без зміни архітектури системи. Декларативний підхід робить процес налаштування зрозумілим і передбачуваним, а також дозволяє використовувати методологію “Infrastructure as Code”, коли вся телеметрійна інфраструктура описується у вигляді коду, що можна версіювати та розгортати автоматизовано.

Таким чином, архітектура пайплайнів у Grafana Alloy є глибоко продуманою і здатною підтримувати як прості, так і надзвичайно складні топології збору даних. Вона забезпечує поєднання стабільності лінійних маршрутів, гнучкості розгалужених потоків і адаптивності умовної маршрутизації. Завдяки цьому система дозволяє організаціям централізовано керувати телеметрією, оптимізувати витрати на збір і зберігання, а також гарантувати узгодженість і повноту даних, що є критично важливим для сучасних підходів до спостережуваності в інфраструктурі підприємств.

Однією з найважливіших особливостей Grafana Alloy, що відрізняє її від класичних агентів збору телеметрії, є використання декларативного підходу до конфігурації. Цей підхід базується на ідеї, що користувач описує не покрокову послідовність дій системи, а кінцевий бажаний стан, тобто що саме потрібно збирати, обробляти та куди передавати. Усе інше – порядок виконання, ініціалізація компонентів, взаємозв'язки між ними, система визначає самостійно. Такий принцип протиставляється імперативним конфігураціям, де адміністратор повинен задавати конкретні інструкції «як» це робити.

У Grafana Alloy декларативна конфігурація представлена у форматі YAML, зрозумілому людині синтаксисі, який зберігає структурованість даних і легко піддається автоматизації. Конфігураційний файл визначає логіку збору телеметрії через набір секцій, які описують компоненти системи – приймачі, процесори, експортери і пайплайни, що з'єднують їх у єдину систему обробки даних.

Принцип роботи декларативної мови Grafana Alloy полягає в тому, що система під час запуску читає конфігураційний файл, перевіряє його на структурну узгодженість, будує граф залежностей між компонентами й автоматично створює внутрішній конвеєр потоків даних. Цей граф можна уявити як динамічно побудовану мережу з компонентів і каналів передачі. Завдяки цьому користувачеві не потрібно вручну вказувати, в якому порядку запускати окремі модулі, Alloy самостійно визначає правильну послідовність ініціалізації. Якщо в конфігурації вказано кілька незалежних пайплайнів, система виконує їх паралельно, ізольовано один від одного, що гарантує стабільність навіть при збої одного з маршрутів.

У декларативній конфігурації Grafana Alloy основним будівельним елементом є блоки опису компонентів, які мають унікальні ідентифікатори та параметри. Кожен блок визначає тип компоненту, його функціональні параметри, протоколи, фільтри чи шляхи з'єднання. Завдяки цьому створюється чітка ієрархія, яку легко масштабувати або змінювати.

Створити базовий шаблон налаштування можна на сайті Alloy Configuration Generator, де є можливість налаштувати деталі приймачів, процесорів, експортерів і пайплайнів, що використовуються.

Типовий приклад декларативного файлу конфігурації Grafana Alloy можна подати так:

```
prometheus.remote_write "local" {
  endpoint {
    url = "http://mimir:9009/api/v1/push"
  }
}

loki.write "local" {
```

```

    endpoint {
      url = "http://loki:3100/loki/api/v1/push"
    }
  }

otelcol.exporter.otlp "local" {
  client {
    endpoint = "tempo:4317"
  }
}

prometheus.scrape "linux_node" {
  targets = prometheus.exporter.unix.node.targets
  forward_to = [
    prometheus.remote_write.local.receiver,
  ]
}

prometheus.exporter.unix "node" {
}

loki.relabel "journal" {
  forward_to = []

  rule {
    source_labels = ["__journal__systemd_unit"]
    target_label = "unit"
  }
}

loki.source.journal "read" {
  forward_to = [
    loki.write.local.receiver,
  ]
  relabel_rules = loki.relabel.journal.rules
  labels = {
    "job" = "integrations/node_exporter",
  }
}

```

Ключовим принципом декларативної конфігурації Grafana Alloy є незалежність компонентів. Кожен елемент існує окремо й не залежить від конкретного місця у файлі, важливо лише, щоб його назва збігалася з посиланнями

у пайплайнах. Це дозволяє динамічно підключати або відключати окремі компоненти, не змінюючи логіку решти системи. Наприклад, можна додати новий експортер для тестового середовища, не порушуючи основного конвеєра, або створити додатковий процесор для попередньої фільтрації даних.

Ще однією важливою рисою декларативного підходу є ідемпотентність конфігурацій, тобто стабільність результату при повторному розгортанні. Якщо файл не змінюється, то незалежно від кількості повторних запусків система створюватиме однакову структуру пайплайнів. Це дає змогу застосовувати автоматизовані інструменти керування конфігураціями, такі як Terraform, Ansible або GitOps-підходи, коли всі зміни конфігурацій зберігаються в системі контролю версій і розгортаються через CI/CD-процеси. У результаті Grafana Alloy легко інтегрується у сучасні DevOps-практики, забезпечуючи принцип «конфігурація як код».

Декларативна модель має також значні переваги у сфері масштабування та стандартизації. У великих організаціях, де десятки або сотні агентів збору телеметрії працюють у різних середовищах, дуже важливо мати уніфікований шаблон конфігурації. Grafana Alloy дозволяє створювати спільні базові файли, які потім успадковуються або перевизначаються залежно від середовища, наприклад, змінюються лише адреси бекендів або додаткові теги середовища (`environment: production, region: eu-west`). Це значно зменшує ризики помилок і полегшує підтримку інфраструктури, адже конфігурація стає передбачуваною, повторно використовуваною та зрозумілою навіть для нових інженерів у команді.

Особливе місце у декларативній системі Grafana Alloy займають умовні та динамічні конструкції, які дозволяють реалізовувати складну маршрутизацію без написання логічних скриптів. За допомогою параметрів можна описувати, наприклад, фільтри за рівнем логів, середовищем або сервісом. Декларативна логіка дає змогу визначити, що логи рівня “error” мають бути спрямовані в один експортер, а решта – в інший, або що певний пайплайн активується лише при наявності конкретного джерела. Такі умови визначаються через параметри

конфігурації та реалізуються всередині компонентів без додаткового коду з боку користувача.

Серед важливих особливостей також варто відзначити самодокументованість декларативних конфігурацій. Завдяки структурованому формату YAML будь-який фахівець може швидко зрозуміти логіку роботи системи, просто переглянувши файл. Це значно підвищує прозорість і керованість великих систем спостережуваності. У разі необхідності зміни поведінки, наприклад, додавання нового типу експорту або модифікації обробки даних, інженер може зробити це без ризику порушення інших процесів, оскільки всі залежності чітко визначені у структурі файлу.

Ще одним важливим аспектом є гнучкість інтеграції декларативних конфігурацій із зовнішніми системами. Оскільки Grafana Alloy підтримує стандарти OpenTelemetry, конфігураційні файли можуть бути використані спільно з іншими компонентами екосистеми, що забезпечує сумісність і переносимість налаштувань. Наприклад, один і той самий конфігураційний шаблон можна використати як для локального агента, так і для колектора в хмарному середовищі, змінивши лише параметри підключення до бекендів.

У підсумку, декларативна мова конфігурації Grafana Alloy не просто спрощує процес налаштування агентів збору телеметрії, а створює цілісну методологію управління інфраструктурою спостережуваності. Вона забезпечує передбачуваність, узгодженість, легкість масштабування й глибоку інтеграцію з DevOps-підходами. Завдяки цьому Grafana Alloy стає не просто інструментом збору даних, а універсальним каркасом для побудови стандартизованих, керованих і відтворюваних систем моніторингу, у яких будь-яка зміна описується декларативно, зберігається у версійному контролі та застосовується автоматизовано без втручання в роботу системи.

2.3. Порівняльний аналіз з альтернативними рішеннями

Система Grafana Alloy з'явилася як відповідь на зростаючу складність сучасних екосистем моніторингу, де необхідно поєднати збір логів, метрик та трас одночасно, зберігаючи гнучкість, масштабованість і сумісність зі стандартами OpenTelemetry. Для повноти розуміння її місця на ринку важливо провести порівняльний аналіз з іншими поширеними рішеннями, такими як Prometheus, Grafana Agent (Classic), OpenTelemetry Collector, Fluent Bit та Vector від Datadog. Кожен із цих інструментів має власну архітектурну модель, підхід до обробки даних та набір компромісів, що визначають їхню придатність до певних сценаріїв використання.

Традиційно системи збору телеметрії будувалися за принципом спеціалізації: одні інструменти фокусувалися виключно на метриках, інші – на логах або трасуванні. Наприклад, Prometheus історично став стандартом для збору метрик завдяки потужній моделі “pull” і гнучкій мові запитів PromQL. Проте Prometheus не передбачає прямого збору логів і не забезпечує обробку даних перед передаванням – для цього зазвичай потрібні додаткові агенти, такі як Fluent Bit чи Loki Agent. Таким чином, архітектура Prometheus вимагає побудови окремих конвеєрів для кожного типу телеметрії, що ускладнює інтеграцію й масштабування.

Інший поширений підхід представлений Fluent Bit – легковаговим агентом для збору та маршрутизації логів. Його основна перевага полягає у високій продуктивності та мінімальному споживанні ресурсів, що робить його популярним у контейнерних середовищах. Однак Fluent Bit не призначений для збору метрик або трас, і тому потребує додаткових компонентів для повноцінного моніторингу. Крім того, його конфігураційна модель є переважно імперативною – користувачеві потрібно явно визначати порядок виконання фільтрів, парсерів і виходів. Це створює ризики помилок у складних середовищах і обмежує масштабованість рішень.

Особливої уваги заслуговує OpenTelemetry Collector, який став основою для багатьох сучасних агентів, зокрема й для Grafana Alloy. Колектор є універсальною платформою для збору, обробки та експорту телеметрії, підтримує модульну архітектуру компонентів і працює за відкритим стандартом OTLP. Проте базовий

OpenTelemetry Collector має деякі обмеження у зручності конфігурації та відсутність інтегрованих інструментів для візуалізації й централізованого управління. Для практичного використання вимагається додаткова обгортка або інструментарій для спрощення його налаштування, а також інтеграція з системами моніторингу, як-от Prometheus або Grafana.

Саме на базі цієї архітектури з'явився Grafana Agent, а пізніше – Grafana Alloy, які розвивають ідеї OpenTelemetry, але пропонують спрощену та більш керовану модель конфігурації. Якщо Grafana Agent мав класичну, частково імперативну структуру, де конфігураційні секції часто дублювали одна одну для різних типів даних, то Grafana Alloy об'єднує все це в єдиний декларативний підхід. Він дозволяє визначати пайплайни обробки незалежно від типу даних, що уніфікує підхід до збору логів, метрик і трас. У цьому полягає одна з головних переваг Alloy над попередніми рішеннями: він не лише підтримує багатоканальний збір, а й забезпечує єдину логіку конфігурації та спільну компонентну базу.

Ще одним важливим конкурентом є Vector – сучасна платформа від Datadog, яка також застосовує концепцію конвеєрів і підтримує збір логів та метрик. Однак Vector фокусується переважно на ефективності обробки потоків даних, а не на повній інтеграції з екосистемою спостережуваності. Хоча його конфігурації також декларативні, система має власну специфічну мову, несумісну з OTLP, що ускладнює інтеграцію зі сторонніми інструментами, такими як Grafana або Tempo. Таким чином, Vector ефективний у сценаріях високонавантажених систем логування, але не забезпечує універсальності, яку пропонує Grafana Alloy.

Порівняльний аналіз також демонструє різницю у підходах до розгортання та масштабування. Prometheus та Fluent Bit часто потребують ручного керування конфігураціями або використання сторонніх засобів оркестрації. Натомість Grafana Alloy завдяки декларативному підходу легко інтегрується у процеси CI/CD, підтримує динамічне розгортання контейнерів через Helm, Kubernetes або Terraform. Це дає змогу централізовано оновлювати конфігурації та забезпечувати однаковий результат незалежно від кількості повторних розгортань.

Grafana Alloy має ще одну очевидну перевагу: він побудований повністю на базі специфікацій OpenTelemetry, підтримує OTLP як основний протокол обміну, а також сумісний із Prometheus, Loki, Tempo та іншими інструментами екосистеми Grafana. Такий підхід забезпечує єдину точку входу для всіх типів телеметрії, що зменшує складність інтеграції та підвищує спостережуваність у розрізнених середовищах.

Крім того, важливою перевагою Grafana Alloy є єдина декларативна мова конфігурації, яка забезпечує самодокументованість і повторюваність системи. На відміну від Fluent Bit або OpenTelemetry Collector у чистому вигляді, де конфігурації можуть виглядати доволі складно, Alloy надає зрозумілий, узгоджений формат YAML, який легко читається й може бути використаний у великих інфраструктурах без значних модифікацій. Це спрощує адміністрування та зменшує кількість людських помилок під час розгортання.

Отже, у результаті порівняльного аналізу можна зробити висновок, що Grafana Alloy поєднує найкращі риси своїх попередників – відкритість і стандартизацію OpenTelemetry Collector, гнучкість Prometheus, продуктивність Fluent Bit, і водночас усуває їхні слабкі сторони завдяки уніфікованій компонентній моделі, декларативному підходу та нативній інтеграції з екосистемою Grafana. Це робить Alloy не просто агентом збору даних, а платформою для побудови цілісної, масштабованої та стандартизованої системи спостережуваності, здатної ефективно функціонувати у мультисервісних і контейнеризованих середовищах.

Таблиця 2.1.

Порівняльний аналіз систем збору телеметрії

Критерій	Prometheus	Fluent Bit	OpenTelemetry Collector	Grafana Agent (Classic)	Vector	Grafana Alloy

Тип даних	Метрики	Логи	Логи, метрики, трейси	Логи, метрики, трейси	Логи, метрики	Логи, метрики, трейси
Архітектура	Монолітна	Модульна	Компонентна	Компонентна	Конвеєрна	Компонентна
Модель конфігурації	Імперативна	Імперативна	YAML-конфігурації з фіксованими секціями	Напівімперативна	Декларативна	Повністю декларативна
Підтримка OpenTelemetry	Часткова (через експортери)	Обмежена	Повна	Повна	Часткова (власний формат)	Повна, нативна підтримка OTLP
Гнучкість маршрутизації	Низька	Середня	Висока	Середня	Висока	Висока
Продуктивність	Висока для метрик	Дуже висока	Висока	Висока	Дуже висока	Висока при обробці різних типів даних
Простота налаштування	Середня	Складна при великій кількості джерел	Середня	Вища за середню	Середня	Висока
Масштабованість	Обмежена горизонтально	Висока	Висока	Висока	Висока	Дуже висока

Таблиця 2.1. (Продовження)

Інтеграція з Grafana / Tempo / Loki	Часткова	Через зовнішні плагіни	Потребує додаткових налаштувань	Тісна	Часткова	Повна, нативна інтеграція
-------------------------------------	----------	------------------------	---------------------------------	-------	----------	---------------------------

Основний сценарій використання	Збір метрик	Логування контейнерів	Універсальний колектор	Гібридне рішення Grafana Labs	Швидкі конвеєри логів	Єдиний агент для всієї телеметрії
---------------------------------------	-------------	-----------------------	------------------------	-------------------------------	-----------------------	-----------------------------------

Проведений аналіз методів та засобів інтеграції систем збору й обробки логів і метрик показує, що еволюція таких інструментів рухається у напрямі від спеціалізованих до універсальних агентів спостережуваності, здатних працювати з усіма типами телеметрії одночасно. Традиційні рішення, як-от Prometheus чи Fluent Bit, залишаються ефективними у вузьких сценаріях, але демонструють обмеження у масштабуванні та гнучкості інтеграції. OpenTelemetry Collector став ключовим етапом розвитку, забезпечивши модульну архітектуру та підтримку відкритих стандартів, однак потребує розширень для практичного використання у великих інфраструктурах.

Grafana Alloy, як спадкоємець і розвиток ідей Grafana Agent та OpenTelemetry, поєднує найкращі характеристики попередників і усуває їхні недоліки. Його декларативна мова конфігурації, універсальна компонентна модель та нативна підтримка OTLP роблять Alloy одним із найбільш гнучких і стандартизованих рішень сучасного рівня. Завдяки цьому він здатен виконувати роль централізованого агента спостережуваності, який однаково ефективно збирає, обробляє й маршрутизує логи, метрики та трейси, незалежно від середовища розгортання.

Таким чином, результати порівняльного аналізу свідчать, що Grafana Alloy є логічною еволюцією екосистеми інструментів моніторингу та найоптимальнішим вибором для побудови інтегрованої системи збору телеметрії у сучасних мультихмарних, контейнеризованих і мікросервісних архітектурах.

3 РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ВПРОВАДЖЕННЯ GRAFANA ALLOY

3.1. Розробка стратегії розгортання, масштабування та високої доступності для систем збору та обробки телеметрії

Побудова сучасної системи спостереження за інфраструктурою та додатками ґрунтується на вимогах до масштабованості, надійності та гнучкості, оскільки обсяг та різноманітність телеметричних даних постійно зростають. Grafana Alloy, який поєднує функціональність декількох класичних інструментів, від Prometheus Agent до OpenTelemetry Collector і Fluent Bit, і надає універсальну платформу, яку можна застосовувати як у малих, так і у високонавантажених розподілених середовищах. Основним принципом побудови архітектури, що базується на Alloy, є модульність і декларативність: усі пайплайни збору, обробки та експорту телеметрії визначаються у вигляді окремих компонентів, які легко змінювати, масштабувати або комбінувати відповідно до потреб організації.

У фундаменті архітектури лежить підхід до побудови безстанової, або stateless, інфраструктури. Alloy за своєю природою не прив'язаний до локального стану, не зберігає історичних даних та може бути безпечно перезапущений у будь-який момент. Завдяки цьому стає можливим горизонтальне масштабування шляхом додавання нових інстансів без складних механізмів шардінгу чи узгодження станів між вузлами. Stateless-підхід також забезпечує безперервність роботи при оновленнях, швидке розгортання та високу стійкість до збоїв, оскільки відмова одного інстансу практично не впливає на доставку телеметрії.

Архітектура системи на базі Grafana Alloy передбачає чітке розділення рівнів збору та агрегації даних. На рівні окремих вузлів – серверів, контейнерів або віртуальних машин працюють легкі агенти, які виконують мінімальну, але необхідну обробку: парсинг логів, додавання базових метаданих, нормалізацію та попередній семплінг. Такий підхід дозволяє мінімізувати навантаження на сервісні машини та не дублювати складну логіку на кожному хості. Натомість більш об'ємні операції, включно з агрегацією, кореляцією між сигналами, розширеним семплінгом, enrichment-процедурами та fan-out маршрутизацією, виконуються на централізованому рівні виконуються у шарі агрегаторів. Це спрощує управління всією телеметричною інфраструктурою та дає змогу застосовувати уніфіковані правила для всієї організації.

Важливим принципом при проєктуванні архітектури є максимальна уніфікація протоколів та форматів даних. Alloy спроектований таким чином, щоб основний обмін даними здійснювався за допомогою OTLP – відкритого стандарту OpenTelemetry, який підтримує роботу через gRPC та HTTP. Використання OTLP як універсального транспортного механізму істотно спрощує інтеграцію з будь-якими сторонніми сервісами, дозволяє уникати залежностей від конкретних постачальників та робить архітектуру гнучкою, придатною для розгортання у багатохмарних середовищах. Одночасно з цим Alloy зберігає сумісність із Prometheus remote_write та Loki API, що важливо у випадку, коли в інфраструктурі вже працюють існуючі рішення.

Ще одним ключовим принципом є перенесення максимальної логічної складності з рівня окремих вузлів у центр обробки. Це дозволяє не тільки зменшити ресурсне навантаження на робочі машини, але й уніфікувати правила обробки та маршрутизації телеметрії, забезпечивши їхню передбачуваність та легке керування. Централізований рівень може виконувати також контроль навантаження, динамічне зниження обсягу даних у разі пікових навантажень та застосовувати ефективні стратегії повторної доставки даних при збоях у бекенд-системах.

Особливе місце посідає аспект надійності та стійкості системи. Alloy підтримує механізми повторної доставки, адаптивні черги, експоненціальну затримку при помилках та тимчасове буферизаційне збереження даних. Уся система проєктується таким чином, щоб у випадку недоступності Loki, Tempo або Mimir не відбувалося втрати телеметрії, а після відновлення бекенду всі дані доставлялися автоматично. До цього додаються вбудовані health-ендпоїнти, за допомогою яких оркестратори можуть визначати реальний стан агента та приймати рішення про його перезапуск або виведення з балансування.

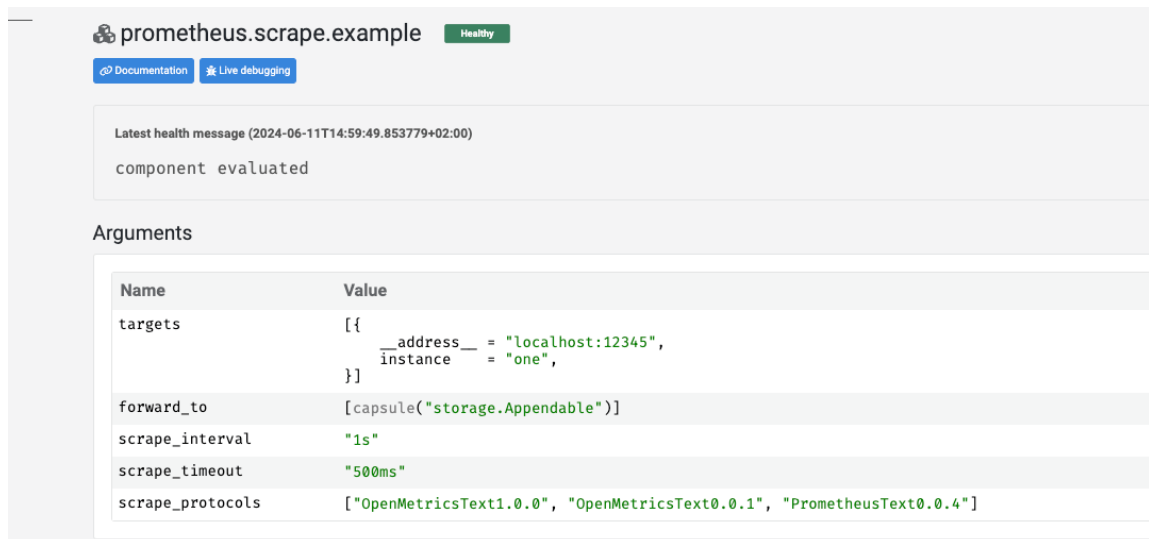


Рис. 3.1. Health-check окремого ендпоїнта

Останнім загальним принципом є забезпечення логічної ізоляції між командами або сервісами. Завдяки гнучкій системі маршрутизації можна будувати окремі пайплайни телеметрії для різних департаментів, продуктів чи середовищ (production, staging, sandbox), використовуючи різні retention-політики та правила доступу. Це значно підвищує безпеку й дозволяє уникати змішування різних потоків даних.

У сукупності всі ці принципи створюють стійку, гнучку та придатну до масштабування архітектуру, яка може ефективно працювати в умовах постійного зростання обсягів телеметрії. Grafana Alloy у такому середовищі виступає універсальним і водночас потужним елементом, здатним об'єднати метрики, логи й трасування в єдину, чітко структуровану екосистему спостереження.

Розгортання Grafana Alloy може здійснюватися у різних архітектурних моделях, і вибір конкретного варіанта визначається масштабом інфраструктури, вимогами до продуктивності, доступності та ступенем централізованості процесів обробки телеметрії. Гнучкість Alloy дозволяє адаптувати його як для невеликих систем з обмеженими ресурсами, так і для розподілених корпоративних середовищ, де обіг телеметрії обчислюється терабайтами на добу. Основним завданням при визначенні моделі розгортання є досягнення оптимального балансу між локальною обробкою, централізованим контролем та ефективним використанням мережесхемних і обчислювальних ресурсів.

Одним із найпоширеніших варіантів є розгортання Alloy у вигляді локального агента на кожному вузлі інфраструктури. У такій моделі кожен сервер або контейнерний вузол має власний інстанс Alloy, який безпосередньо збирає метрики, логи та трасування з локальних джерел. Цей підхід забезпечує мінімальні затримки та дозволяє негайно обробляти події там, де вони виникають. Локальні агенти ефективні для середовищ, де важливе детальне спостереження за індивідуальними компонентами, а також у випадках, коли потрібно забезпечити високу стійкість до збоїв: навіть якщо центральні системи тимчасово недоступні, агенти можуть продовжувати накопичувати та відправляти дані після відновлення зв'язку. Окрім того, локальний підхід підтримує гнучке налаштування різних пайплайнів для окремих хостів або груп сервісів.

Другим варіантом є побудова централізованих кластерів Alloy, які виконують роль агрегаторів або ретрансляторів телеметрії. На відміну від локальних інстансів, що обмежуються обробкою вузького набору локальних даних, агрегатори відповідають за злиття потоків із різних джерел, їхню стандартизацію та маршрутизацію до бекенд-систем. Такий підхід особливо ефективний у великих інфраструктурах, де необхідно уніфікувати логіку семплінгу, нормалізації або збагачення даних. Централізовані інстанси здатні одночасно обробляти великі обсяги телеметрії та зменшують навантаження на кожний окремий сервер, оскільки складна логіка перенесена у центральний шар. Цей підхід добре підходить для організацій, де керування телеметрією здійснюють окремі команди, а контроль політик має здійснюватися з однієї точки.

Особливим випадком централізованого підходу є горизонтальне масштабування агрегаторів у кластер, коли Alloy запускається в кількох ідентичних екземплярах за балансувальником навантаження. Оскільки Alloy є безстановим, такий кластер легко масштабувати відповідно до поточної кількості даних: при зростанні обсягу телеметрії додаються нові екземпляри, а при зменшенні можуть бути вимкнені без ризику для системи. Цей варіант часто застосовується у сценаріях з високими вимогами до доступності та необхідністю приймати дані від

великої кількості агентів, наприклад у глобальних розподілених системах або багатокластерних Kubernetes-середовищах.

У контейнеризованих середовищах, зокрема у Kubernetes, Alloy найчастіше розгортається у вигляді DaemonSet, що забезпечує автоматичний запуск окремого агента на кожному вузлі кластера. DaemonSet-підхід забезпечує природну відповідність топології кластера ідентичній кількості агентів, що ідеально підходить для збору логів та метрик контейнерів. У складніших сценаріях Alloy може також запускатися як Deployment для виконання ролі кластерного агрегатора або проміжного ретранслятора між сервісами та зовнішніми бекенд-системами. Завдяки гнучкому механізму конфігурації Helm-чартів та можливості динамічного масштабування цей варіант є найбільш придатним для хмарних, динамічних і високонавантажених систем [11].

Таким чином, універсальність підходів до розгортання Grafana Alloy дозволяє будувати архітектури будь-якої складності – від невеликих систем з кількома вузлами до масштабних корпоративних середовищ, що містять тисячі сервісів та мільйони телеметричних подій на секунду. Гнучкість, безстановість і сумісність Alloy з відкритими стандартами робить його фундаментальним елементом сучасної інфраструктури спостереження, здатним адаптуватися під будь-які технологічні та операційні вимоги.

Масштабування Grafana Alloy є ключовим елементом побудови надійної та продуктивної системи спостереження, оскільки обсяг телеметрії в сучасних інфраструктурах зростає нерівномірно та інколи непередбачувано. Alloy, завдяки своїй безстановій природі та принципам модульності, дозволяє проєктувати гнучкі стратегії масштабування, що можуть адаптуватися під змінні робочі навантаження, топологію інфраструктури та вимоги до затримок і доступності. Основна мета масштабування полягає у тому, щоб забезпечити стабільну обробку великого потоку даних без деградації продуктивності та без втрати інформації, водночас оптимізуючи використання ресурсів.

В основі стратегії масштабування Alloy лежить розмежування функцій збору та агрегації. Локальні агенти, встановлені на кожному вузлі, масштабуються

природним чином разом із розширенням інфраструктури, адже кожен новий сервер або контейнерний хост автоматично отримує власний інстанс Alloy. Це дозволяє без додаткового навантаження на центральні системи збирати повний обсяг телеметрії, знижуючи ризик утворення мережових вузьких місць. Така модель є фундаментально ефективною, оскільки масштабування відбувається горизонтально та органічно, без необхідності ускладнювати конфігурацію.

Водночас центральний рівень, що виконує агрегацію, маршрутизацію або збагачення даних, потребує окремого підходу. Як правило, масштабування цього рівня здійснюється шляхом запуску декількох однотипних інстансів Alloy за балансувальником навантаження. Оскільки Alloy є безстановим і не потребує координації станів між вузлами, додавання нових екземплярів дозволяє лінійно збільшувати пропускну здатність. Такий підхід мінімізує ризики перевантаження під час пікових навантажень і створює основу для побудови високодоступних систем. Застосування механізмів автоматичного масштабування, зокрема у Kubernetes, дозволяє оперативно реагувати на зміну потоку телеметрії та підвищувати рівень SLA.

Одним із ключових елементів масштабування є оптимізація конфігурацій пайплайнів. Надмірна кількість процесорів у ланцюзі обробки може негативно вплинути на пропускну здатність та збільшити затримки. Тому для великих навантажень рекомендується розподіляти складні обчислювальні операції по декількох рівнях, наприклад, злегка обробляти дані на локальних агентах, а глибшу нормалізацію або кореляцію виконувати лише на центральних агрегаторах. Такий підхід дозволяє зменшити дублювання роботи та уникнути перенавантаження окремих інстансів.

У великих і розподілених системах також необхідно враховувати географічне масштабування. Розміщення Alloy ближче до джерел телеметрії зменшує затримку та навантаження на транзитні канали. Зазвичай це реалізується через створення регіональних шарів Alloy, які виконують збір даних у конкретному регіоні та передають їх до глобальних бекенд-систем. Така модель є особливо важливою для

транснаціональних організацій, де централізація збору телеметрії може створити критичні мережеві перевантаження.

Стратегія масштабування повинна враховувати й економічні аспекти. Оскільки Alloy може обробляти великі обсяги даних, постає питання оптимізації витрат на ресурси та передачу. У таких випадках застосовується селективна фільтрація або семплінг даних на агентському рівні. Наприклад, можна зменшувати частку трасувань, що відправляються до бекендів, або відсіювати надлишкові логи. Alloy дозволяє гнучко застосовувати різні політики для різних типів телеметрії чи сервісів, що дає змогу масштабувати систему в межах наявного бюджету.

Окремої уваги потребує інтеграція Alloy із зовнішніми системами, такими як Loki, Tempo чи Mimir. Оскільки саме бекенд-сервіси визначають швидкість прийому даних, масштабування Alloy повинно бути синхронізоване з їхніми можливостями. Якщо бекенди не справляються з вхідними потоками, необхідно або збільшувати їх потужність, або застосовувати політики обмеження навантаження у Alloy. Це дозволяє уникнути втрати телеметрії та зберегти передбачувану поведінку системи.

Таким чином, стратегія масштабування Grafana Alloy будується на принципах горизонтального розширення, гнучкої оптимізації пайплайнів та адаптації до обмежень бекенд-систем. Alloy дозволяє створювати багаторівневі архітектури, які вирізняються стійкістю, масштабованістю та передбачуваністю, що є критично важливим для сучасних систем спостереження, здатних ефективно працювати в умовах постійного зростання обсягів телеметрії та збільшення складності інфраструктури.

Забезпечення високої доступності (High Availability, HA) для системи збору та обробки телеметрії на базі Grafana Alloy є критичним аспектом побудови надійної інфраструктури спостереження, у якій безперервність передачі даних має першочергове значення. У сучасних розподілених середовищах навіть короткочасна зупинка агентів чи проміжних елементів обробки може призвести до втрати важливої діагностичної інформації, що ускладнить відновлення сервісів та

знизить загальний рівень оперативності реагування. Тому архітектура на основі Alloy повинна бути спроектована з урахуванням принципів відмовостійкості, надмірності та самовідновлення компонентів.

 Clustering List of clustering peers			
Node Name	Advertised Address	Current State	Local Node
node-a	127.0.0.1:12345	participant	
node-b	127.0.0.1:12346	participant	☒
node-c	127.0.0.1:12347	participant	

Рис 3.2. Приклад налаштованих кластерів для забезпечення НА

Однією з ключових переваг Alloy є його безстановість, що істотно спрощує побудову механізмів високої доступності. Оскільки окремі екземпляри Alloy не зберігають критичного внутрішнього стану, вони можуть бути легко замінені, перезапущені або дубльовані без ризику втрати узгодженості. Це дозволяє розгортати кілька паралельних інстансів, які виконують однакові функції, забезпечуючи мінімізацію часу простою у випадку відмови окремого вузла. Такий підхід формує основу для побудови кластерів високо доступних агрегаторів або маршрутизаторів телеметрії, розміщених за балансувальником навантаження.

Балансування трафіку відіграє ключову роль у гарантуванні постійної доступності сервісу. Оскільки потоки телеметрії можуть бути дуже нерівномірними, важливо застосовувати механізми розподілу навантаження, які забезпечують стабільний розподіл обробки між інстансами Alloy. У більшості випадків трафік направляється через L4/L7 балансувальники, що дозволяє гнучко перемикає потоки даних у реальному часі. Використання health-checks та механізмів автоматичного виключення неефективних вузлів забезпечує оперативну реакцію на збої, зводячи час недоступності фактично до нуля.

Важливою складовою забезпечення НА є підвищення надійності на рівні агентів, встановлених на кінцевих серверах або контейнерних вузлах. Оскільки телеметрія генерується розподілено, зупинка агентського процесу на окремому хості може призвести до повної втрати даних з цього вузла. Для запобігання подібним ситуаціям використовуються стандартні механізми відмовостійкості операційної системи та оркестраторів, наприклад, systemd-units із автоматичним

перезапуском, Kubernetes DaemonSets із політиками Always, або інструменти моніторингу, що забезпечують самовідновлення. Застосування таких стратегій гарантує, що агентські компоненти Alloy залишаються працездатними навіть за умов локальних збоїв.

Суттєву роль у контексті НА відіграють бекенд-сервіси, до яких надсилає телеметрію Alloy: Loki, Tempo, Mimir, Prometheus Remote Write та інші. Оскільки доступність цих систем визначає загальну надійність процесу збору даних, важливо забезпечити наявність резервних точок прийому. Alloy дозволяє дублювати налаштування експортерів, налаштовувати fallback-механізми, направляти дані одночасно у кілька кластерів чи застосовувати черги доставки. Така гнучкість у маршрутизації є одним із найпотужніших інструментів для підвищення загального рівня відмовостійкості.

Не менш важливим є використання політик локального буферизаційного кешування. У випадку тимчасової недоступності бекенду Alloy може зберігати дані локально та надсилати їх пізніше, коли сервіс стане доступним. Це особливо корисно для віддалених або нестабільних регіонів, де мережеві збої можуть бути частими. Механізм локального кешу значно зменшує ризик втрати телеметрії та дозволяє підтримувати цілісність історичних даних.

У підсумку, забезпечення високої доступності Grafana Alloy базується на поєднанні архітектурних рішень, механізмів дублювання, інструментів автоматичного самовідновлення та інтелектуального маршрутизаційного підходу. Alloy дозволяє побудувати відмовостійку систему спостереження, яка зберігає працездатність навіть за умов часткових або повних збоїв окремих компонентів, забезпечуючи безперервність передачі телеметрії та стабільність інфраструктури в цілому.

Базовим принципом побудови рекомендованої продакшн-архітектури є розділення рівнів збору та рівнів агрегації даних. На кожному хості, незалежно від того, чи це фізичний сервер, віртуальна машина або контейнерний вузол у Kubernetes, має бути розгорнутий локальний агент Alloy, який відповідає за безпосереднє отримання метрик, логів і трасувань від застосунків та системних

сервісів. Такий підхід забезпечує максимальну близькість до джерела телеметрії й мінімізує ризик втрати даних у разі тимчасових збоїв мережі або сторонніх сервісів. Кожен екземпляр агента працює ізольовано й виконує базову нормалізацію, фільтрацію та перетворення даних.

Другий важливий елемент архітектури – рівень проміжних агрегаторів або роутерів телеметрії. Ці екземпляри Alloy не прив'язані до конкретних хостів, а працюють як спільний шар обробки. Вони приймають великий обсяг даних від численних агентів, виконують додаткову маршрутизацію, enrich-процеси (наприклад, додавання тегів, кореляцію сигналів або конвертацію форматів), а також забезпечують доставку даних до бекенд-систем – Loki, Tempo, Prometheus/Mimir або сторонніх інструментів. Використання окремого рівня агрегаторів дає змогу масштабуватися поступово, залежно від збільшення кількості агентів або підвищення інтенсивності телеметричного трафіку.

З метою забезпечення відмовостійкості агрегаторів рекомендується розгортати їх не менш ніж у двох екземплярах у кожній зоні доступності або дата-центрі, встановлюючи перед ними балансувальники навантаження. Подібна конфігурація гарантує, що збій одного вузла не призведе до порушення доставки телеметрії й не спричинить втрату даних. Завдяки тому, що Alloy є безстановим, інстанси можуть бути з легкістю масштабовані вгору або вниз, а також замінені новими без будь-яких ризиків порушення узгодженості.

Третій компонент рекомендованої архітектури включає інтеграцію з централізованими системами зберігання телеметрії. Метрики, логи й трейсинг мають направлятися до відповідних бекендів, оптимізованих під конкретний тип даних: Loki – для логів, Tempo – для трасувань, Mimir або Prometheus – для метрик. У продакшні доцільно використовувати окремі кластери для цих систем, бажано з власною реплікацією та шардінгом, що забезпечує оптимальну продуктивність і стійкість навіть при дуже великих обсягах даних. Alloy у цьому ланцюгу виконує роль надійного транспортного рівня, який гарантує доставку даних навіть за умов тимчасової недоступності бекенд-сервісів завдяки механізмам локального кешування та повторної доставки.

Окрему увагу в продакшн-архітектурі слід приділити безпеці. Оскільки телеметрія містить потенційно чутливу інформацію щодо внутрішньої роботи систем, рекомендується обов'язково використовувати TLS-шифрування для всіх каналів зв'язку між агентами, агрегаторами та бекендами. Крім того, доцільно налаштовувати механізми автентифікації, такі як консистентні ключі доступу або мікросервісні аутентифікаційні шлюзи. Alloy підтримує такі підходи нативно, що дозволяє будувати безпечну транспортну модель, не обмежуючи продуктивність або масштабованість системи.

Ще одним елементом рекомендованої архітектури є централізоване управління конфігураціями Alloy. У великих середовищах, де можуть бути сотні або тисячі агентів, ручне оновлення конфігурацій є неприйнятним. Тому доцільно застосовувати GitOps-підхід, систему шаблонізації (наприклад, Helm, Kustomize або Terraform) або централізовані механізми управління конфігураціями (Ansible, Puppet, SaltStack). Завдяки уніфікованій структурі конфігурацій Alloy легко адаптується до таких підходів, що дозволяє забезпечувати консистентність правил обробки телеметрії по всій інфраструктурі.

Географічний аспект продакшн-розгортання також відіграє істотну роль. Якщо інфраструктура розподілена по кількох регіонах, доцільно застосовувати локальні агрегатори в кожному регіоні з можливістю подальшої передачі даних у глобальний кластер бекендів. Така стратегія мінімізує затримки, знижує навантаження на міжрегіональні канали та додатково захищає систему від регіональних інцидентів. Alloy уживається в подібних сценаріях органічно, забезпечуючи плавну інтеграцію локальних та глобальних потоків телеметрії.

У підсумку рекомендована продакшн-архітектура Grafana Alloy передбачає багаторівневу організацію: локальні агенти на кожному хості, високоступні агрегатори з балансуванням навантаження, окремі безпечні канали доставки, централізовані системи зберігання телеметрії зі шардінгом і реплікацією, а також зріла система управління конфігураціями. Такий підхід дозволяє створити масштабовану, гнучку й відмовостійку інфраструктуру спостережуваності, здатну

забезпечувати стабільне функціонування навіть у складних динамічних умовах виробничого середовища.

Стратегія розгортання Grafana Alloy у продакшн-середовищах повинна спиратися на чітко визначені принципи, які забезпечують масштабованість, відмовостійкість і передбачуваність системи спостережуваності. Розмежування ролей між локальними агентами, агрегаторами телеметрії та бекенд-системами дозволяє сформувати гнучку багаторівневу архітектуру, здатну ефективно працювати як у невеликих інсталяціях, так і у великих розподілених інфраструктурах. Alloy у цьому контексті виступає універсальним компонентом, що стандартизує збір, обробку й маршрутизацію телеметрії, забезпечуючи стабільну доставку даних навіть у разі збоїв окремих вузлів.

Продакшн-підхід передбачає виконання низки ключових вимог: правильно організоване масштабування, використання високодоступних конфігурацій із балансуванням навантаження, захищені канали передачі, а також централізоване управління конфігураціями для підтримки узгодженості всієї системи. Рекомендована архітектура демонструє, що Alloy може ефективно інтегруватися з Loki, Tempo, Prometheus/Mimir та іншими сервісами, формуючи цілісне середовище спостережуваності. У результаті впровадження такої стратегії організація отримує надійну, стабільну та гнучку телеметричну інфраструктуру, здатну адаптуватися до зростання навантажень та вимог бізнесу.

3.2. Розробка ключових моментів при інтеграції Grafana Alloy в інфраструктуру

Перш ніж інтегрувати Grafana Alloy у будь-яке середовище, надзвичайно важливо провести глибокий і всебічний аналіз наявної телеметричної інфраструктури. Це не просто технічна формальність, а ключовий етап, який визначає, наскільки гладко й ефективно проходитиме подальше впровадження. Аналіз дозволяє зрозуміти, як рухаються метрики, логи та трасування всередині організації, які джерела даних є критичними, які процеси вже автоматизовані, а які

все ще працюють у ручному режимі або взагалі не контролюються. У багатьох випадках саме на цьому етапі виявляється хаотичність або надмірна фрагментація телеметрії, що часто виникає через роки поступового нарощування інструментів і відсутність централізованої стратегії спостережуваності.

Одним із першочергових завдань є визначення того, які агенти та колектори вже працюють на серверах, кластерах чи контейнерних середовищах. У деяких інфраструктурах можуть одночасно працювати кілька різних агентів, наприклад, `node_exporter` для метрик, `Fluent Bit` для логів, `OpenTelemetry Collector` для трасувань, а також внутрішні або сторонні рішення для специфічних потреб. Така гібридність може бути функціонально робочою, але часто призводить до дублювання навантаження, конфліктів за порти, неконсистентних форматів даних і важкості в подальшому обслуговуванні. Аналіз дозволяє зрозуміти, які компоненти можна оптимізувати або повністю замінити Alloy, а які краще залишити як є, якщо вони виконують вузькоспеціалізовану роль.

Наступним кроком є детальне дослідження маршрутів телеметрії, від моменту, коли дані з'являються на джерелі, до їх кінцевого призначення. Дуже важливо зрозуміти, як саме дані проходять через різні рівні інфраструктури: чи обробляються вони локально на вузлах, чи надсилаються одразу в центральну систему, чи фільтруються вони перед передачею, хтось виконує нормалізацію, чи існують проміжні буфери або черги. Саме тут часто виявляються вузькі місця, наприклад, коли всі логи необроблено надсилаються в один великий бекенд, що призводить до перевитрати ресурсів і зниження продуктивності. Alloy може значно змінити ці процеси, тому варто заздалегідь визначити, де саме він зможе принести найбільшу користь.

Окрему увагу потрібно приділити формату телеметрії. Сьогодні багато організацій працюють у змішаних середовищах, де частина сервісів генерує структуровані OTLP-трасування і JSON-логи, а інші – старі plain-text формати або нестандартні метрики. Це створює виклики при подальшій інтеграції та кореляції даних між собою. Під час аналізу необхідно визначити, які формати підтримуються, які потрібно конвертувати, де необхідна нормалізація та яку

частину цього процесу може виконати Grafana Alloy без зміни кодових баз сервісів.

І завершальним важливим аспектом аналізу є виявлення ризиків та технічного боргу. Часто під час дослідження телеметрії виявляється, що деякі сервіси давно не оновлювалися, в деяких конфігураційних файлах накопичилися неактуальні директиви, існують старі агенти, які ніхто не підтримує, або деякі потоки логів дублюються у два різні бекенди, створюючи зайві витрати. Це той момент, коли перехід на Grafana Alloy може стати не лише технічним оновленням, але й можливістю провести загальну санацію телеметричної інфраструктури.

Таким чином, детальний аналіз існуючої телеметричної інфраструктури є фундаментом успішної інтеграції Grafana Alloy. Він дозволяє побачити повну картину поточного стану, виявити недоліки, визначити правильну стратегію переходу, уникнути поширених помилок і створити таку архітектуру спостережуваності, яка буде масштабованою, зрозумілою та керованою у довгостроковій перспективі.

Першим кроком у плануванні міграції є визначення чітких цілей переходу. Необхідно зрозуміти, що саме очікує організація від Alloy: зменшення навантаження на існуючу телеметричну інфраструктуру, спрощення керування агентами, покращення продуктивності, розширення функціональних можливостей чи перехід на OpenTelemetry-стандартизовані формати. Важливо сформулювати ці цілі завчасно, оскільки вони визначатимуть архітектурні рішення та обсяг змін у майбутніх етапах.

Наступним етапом є складання карти залежностей, тобто чітке визначення того, які сервіси, кластери чи вузли будуть охоплені переходом та як саме вони взаємодіють з телеметрією. Це включає перевірку всіх точок збору даних, способів їхнього транспортування, бекендів, куди вони передаються, і обробки, що відбувається по дорозі. Таке картування дозволяє виявити місця, де встановлення Alloy може порушити існуючі ланцюги, а також визначити вузли, де перехід потрібно проводити найобережніше – наприклад, системи з великим навантаженням або критичні сервіси. Саме тут часто виявляється необхідність у

тестовому середовищі, яке дозволяє перевірити роботу Alloy без ризику для продакшну.

Далі необхідно розробити детальний план міграції, який включає поетапне впровадження. Найкращою практикою є створення кількості етапного підходу: спочатку встановити Alloy тільки на тестових серверах, потім застосувати на обмеженій групі некритичних продакшн-вузлів, і лише після цього плавно розширювати застосування на решту інфраструктури. Такий підхід дозволяє рано виявити помилки конфігурації, розбіжності у форматах даних або проблеми продуктивності, не впливаючи на основні сервіси.

Особливо важливо розробити план сумісного функціонування Alloy з уже існуючими агентами. Повний одномоментний перехід зазвичай неможливий, оскільки одні та ті самі дані можуть тимчасово збиратися двома різними компонентами. Тому необхідно передбачити перехідний період, упродовж якого старі агенти поступово виводяться з експлуатації, а Alloy бере на себе відповідні обов'язки. Це потребує ретельного порівняння даних, щоб переконатися, що нові пайплайни працюють коректно і не створюють дублювання або втрат інформації.

Завершальним етапом планування є підготовка документації, як внутрішньої для DevOps-команди, так і експлуатаційної для SRE або розробників. Вона має містити ключові принципи міграції, опис архітектурних рішень, схеми та інструкції, щоб перехід не залежав від одного спеціаліста і міг бути відтворений будь-ким із команди.

Таким чином, планування переходу на Grafana Alloy це комплексний процес, який потребує технічної точності, стратегічного бачення та врахування багатьох залежностей всередині інфраструктури. Правильно спланований перехід дозволить організації мінімізувати ризики, уникнути простоїв і вивести систему спостережуваності на новий рівень ефективності, керованості та стандартизації.

Під час інтеграції з Loki найчастіше використовують локальні джерела на сервері або в контейнерному середовищі, такі як файлові журнали або `journald`. Alloy у цьому випадку зчитує лог-рядки, застосовує необхідні трансформації –

наприклад, видалення зайвих метаданих, нормалізацію міток чи додавання інформації про сервіс – і передає їх у Loki за допомогою HTTP-пушу.

Інтеграція з Tempo будується навколо протоколу OTLP – це стандартний спосіб передачі трасувань. Alloy може як приймати OTLP-трафік від застосунків, так і пересилати його в Tempo. Це дає можливість розташовувати Alloy поруч із сервісами, які генерують трасування, і мінімізувати затримку.

Тут Alloy приймає трасування як через gRPC, так і через HTTP, після чого передає їх у Tempo. Це особливо корисно для середовищ, де застосунки використовують різні реалізації OpenTelemetry SDK, або коли потрібно підтримувати кілька протоколів одночасно. Alloy також можна налаштувати на sampling, фільтрацію чи обмеження трафіку, що дозволяє зменшити навантаження на Tempo без втрати ключової інформації.

Збирання метрик інтегрується найчастіше через Prometheus-сумісний scraping або через remote_write. Alloy здатний як виставляти власні експортери, так і збирати метрики зі сторонніх застосунків. Після цього дані можуть надсилатися безпосередньо у Prometheus або Mimir. Конфігурація remote_write є найпростішим способом передати метрики у центральне сховище, особливо коли потрібно уникнути локальних інстансів Prometheus.

Такий підхід дозволяє не тільки збирати власні метрики Alloy, але й централізовано відправляти будь-які інші прометеусові метрики. Alloy може працювати як lightweight collector, що підписується на експортери або scrape-мітки, об'єднує метрики з різних джерел і передає їх у бекенд уже в узгодженому вигляді.

Важливо враховувати, що інтеграція всіх цих компонентів передбачає узгодження схем маршрутів даних, форматів, політик безпеки та можливостей масштабування. Наприклад, якщо використовуються OTLP-receivers, необхідно впевнитися, що порти прослуховуються лише потрібними інтерфейсами й захищені TLS або мережевою сегментацією. При інтеграції з Loki потрібно переконатися, що релабелінг відбувається на рівні агента, щоб уникнути надмірного зростання навантаження на Loki. А при роботі з Prometheus важливо

налаштувати queue-параметри `remote_write` таким чином, щоб Alloy гарантовано витримував пікові навантаження.

Однією з ключових переваг Grafana Alloy у порівнянні з класичними агентами або окремими OpenTelemetry Collector, Promtail чи Fluent Bit, є можливість значно спростити конфігурацію всієї телеметричної системи. Замість того щоб підтримувати десятки різних конфігурацій у різних форматах і з різною логікою, Alloy пропонує єдиний підхід до роботи з логами, метриками та трасуваннями. Це не просто полегшує адміністрування, це змінює сам підхід до побудови інфраструктури: телеметрія стає зрозумілішою, передбачуваною та легшою у відлагодженні.

Зазвичай у середовищах, де використовуються класичні агенти, виникає ситуація, коли один компонент відповідає за збирання метрик, інший за логування, третій за трасування, а четвертий за їх маршрутизацію. У такій архітектурі кожен агент має власний синтаксис, власні правила обробки даних та власні підходи до логування та моніторингу. Це призводить до того, що конфігурації зростають у розмірах, а зміни в одному місці можуть зачіпати інші компоненти. Alloy, навпаки, дозволяє уніфікувати всі ці процеси й створити конфігурацію, яка логічно цілісна, легко читається і підтримується в межах одного формату.

Одним із ключових викликів при інтеграції Grafana Alloy у вже існуючу інфраструктуру є уніфікація та стандартизація протоколів телеметрії. У багатьох організаціях історично накопичуються різні агенти, формати логів і протоколи експорту даних: Prometheus exposition format для метрик, Fluentd/Fluent Bit або syslog для логів, Jaeger/Zipkin для трасування, власні формати для внутрішніх сервісів. У результаті телеметрія збирається хаотично, а підтримка системи ускладнюється із кожним новим компонентом. Перехід на Grafana Alloy дозволяє створити єдиний рівень збору телеметрії саме завдяки використанню протоколу OpenTelemetry як базового стандарту, що замінює або інтегрується з більшістю альтернативних підходів.

OpenTelemetry вирішує проблему розрізнених протоколів тим, що надає спільну модель даних для метрик, логів і трасування, а також уніфіковані

протоколи передачі цих даних через OTLP/HTTP і OTLP/gRPC. У порівнянні з класичним Prometheus-підходом, де метрики збираються шляхом pull-опитування endpointів, OpenTelemetry робить акцент на push-моделі, що дозволяє уникнути проблем зі складними топологіями мереж, NAT-ом або сервісами, які працюють не за статичними адресами. Водночас Alloy добре підтримує обидва підходи, як OTLP, так і Prometheus exposition format, дозволяючи органічно поєднувати push і pull без необхідності переписувати телеметрію всіх сервісів одразу.

Стандартизація протоколів також спрощує міжсервісну взаємодію в мікросервісних інфраструктурах. Наприклад, сервіси в Kubernetes можуть генерувати метрики у Prometheus-форматі, трейси у OTLP, а логи писати у stdout, після чого Alloy єдиним конфігом збирає все й уніфікує. Такі підходи практично неможливо реалізувати через Prometheus, який не підтримує логи та трейси, або через Fluent Bit, який не має нормальної роботи з метриками та трейсами. Таким чином, OpenTelemetry разом із Grafana Alloy формує сучасний стандарт телеметрії, який забезпечує максимальну сумісність, розширюваність і зручність супроводу.

Загалом вибір OpenTelemetry як головного протоколу в рамках інтеграції Grafana Alloy не означає відмову від існуючих інструментів, скоріше це створює універсальний прошарок, здатний приймати дані з різних джерел та перетворювати їх у єдиний формат. Це робить телеметрію прозорою, логічно узгодженою та готовою до масштабування, а сам Alloy стає центральною точкою стандартизації. Саме тому під час переходу до Alloy варто розглядати OpenTelemetry не як заміну, а як стратегічну основу для майбутньої, більш чистої, однорідної та гнучкої архітектури збору телеметрії.

Оптимізація продуктивності Grafana Alloy у процесі інтеграції з існуючою інфраструктурою стає одним із ключових завдань, оскільки саме від цього залежить наскільки швидко телеметрія буде збиратися, оброблятися та передаватися далі в системи зберігання. У багатьох організацій телеметричний трафік зростає пропорційно масштабуванню інфраструктури: збільшення кількості сервісів, мікросервісів, контейнерів, віртуальних машин або edge-вузлів неминуче приводить до збільшення логів, метрик та трас. І якщо на початкових етапах Alloy

може працювати з типовими конфігураціями без помітних затримок, то при навантаженні у сотні мегабайт логів на секунду або десятки тисяч spans у хвилину починають проявлятися вузькі місця, які потребують системного підходу до оптимізації.

Перш за все оптимізація продуктивності зосереджується на правильній організації пайплайнів в Alloy. Це означає, що всі процесори, які застосовуються до логів або трас, повинні бути розташовані в логічній і максимально ефективній послідовності. Наприклад, якщо вхідні логи потребують парсингу, то краще застосовувати більш легкі фільтри або дропи перед важкими парсерами, щоб зменшити навантаження. В Alloy це особливо важливо, оскільки він дозволяє комбінувати функції Promtail, OpenTelemetry Collector і Prometheus Agent в одному процесі. Неправильний порядок процесорів може призвести до надлишкового CPU-споживання або навіть до backpressure, коли відправка даних у Loki, Tempo або Prometheus Remote Write починає накопичуватись у чергах.

Другим значущим аспектом є контроль параметрів черг і буферизації. Alloy, як і OTel Collector, підтримує механізми batch-обробки, які значно впливають на продуктивність. Збільшення розміру batch у телеметрії дозволяє зменшити кількість запитів до Loki або Tempo, що знижує накладні витрати на мережу та serialization/deserialization. Але при надмірно великих batch-розмірах зростає ризик збільшення latency у доставці даних, що критично для метрик, які повинні бути актуальними для alerting. Тому оптимізація завжди проходить шляхом балансування між latency та throughput. У реальних конфігураціях Alloy рекомендовано тестувати batch-розміри поступово: наприклад, починати з 512–1024 елементів для логів і понад 200–300 spans для трасування, коригуючи їх залежно від навантаження.

Ще один важливий аспект оптимізації пов'язаний із зменшенням обсягу телеметрії перед її передачею. Alloy дозволяє робити sampling трейсів, фільтрацію логів, дроп debug-рівня, редукцію метрик, обрізання навантаження або перетворення структур. Наприклад, у великих мікросервісних архітектурах debug-логи можуть генерувати сотні гігабайт на добу, і їх бездумна передача в Loki не має

сенсу. Використання шаблонів і фільтрів на рівні Alloy дозволяє зберігати інформативність телеметрії, але при цьому зменшувати її в десятки разів. У випадку з трасуванням `sampling` є не просто оптимізацією, а необхідністю, оскільки Тетро або будь-яке інше сховище трейсів може бути перевантажене при повному зборі `spans`. Alloy може застосовувати `tail-based sampling`, що дозволяє залишати тільки ті трейси, які містять помилки, латентність, виключення або інші аномалії, зберігаючи найважливіші дані при мінімальному обсязі.

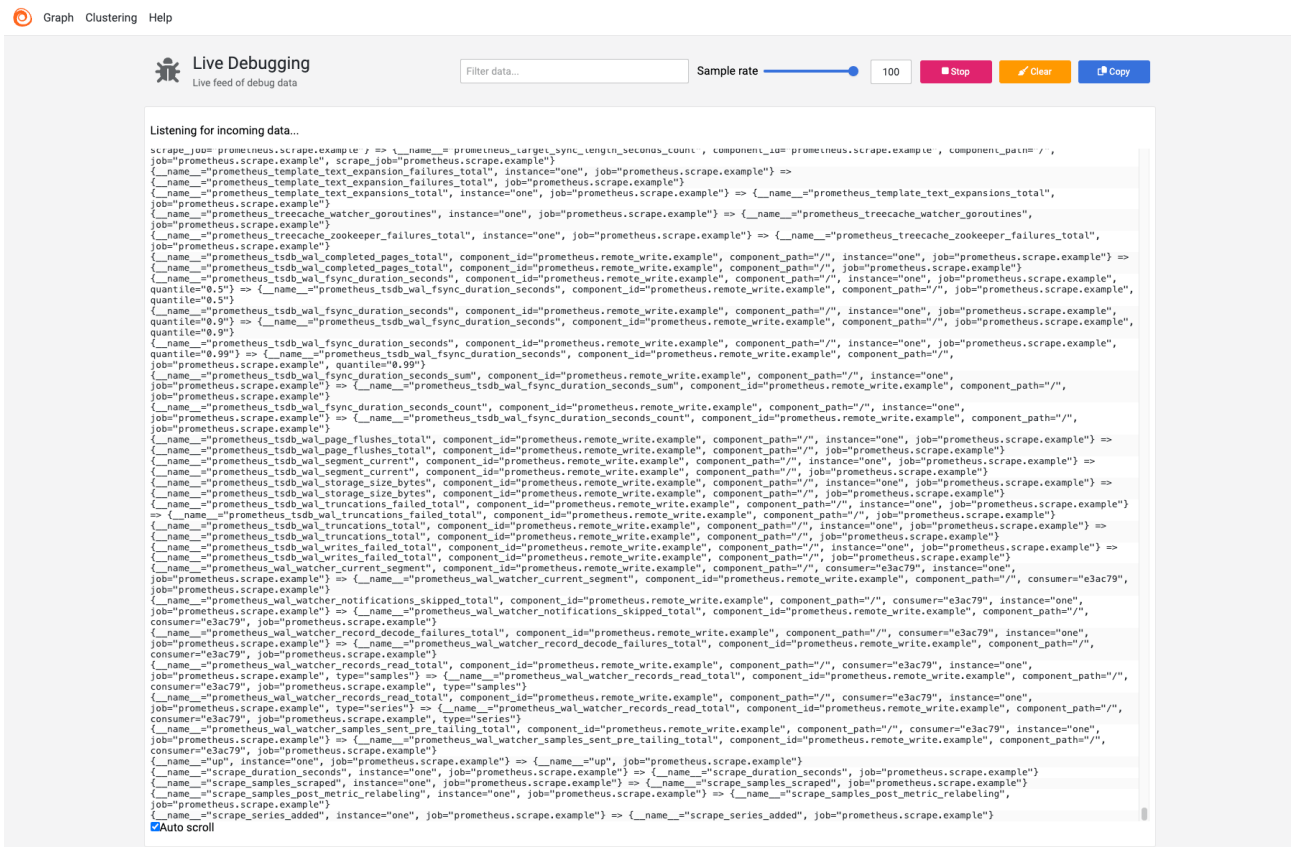


Рис 3.3. Приклад debug-логів

Нарешті, важливо враховувати оптимізацію продуктивності на рівні мережевих протоколів. OTLP/gRPC, як правило, забезпечує вищу пропускну здатність та менше навантаження у порівнянні з OTLP/HTTP або застарілими протоколами типу Jaeger Thrift. Перехід на OTLP глибоко зменшує навантаження при високому обсязі телеметрії, але вимагає коректного налаштування `timeouts` та `backoff`. Alloy дозволяє застосовувати передові політики `retry`, що допомагає уникати втрати даних у випадках короткочасної недоступності Loki або Тетро, але при цьому не допускати перевантаження цих систем.

Таким чином, оптимізація продуктивності – системний процес побудови правильної архітектури обробки телеметрії в Alloy, що включає оптимізацію пайплайнів, черг, паралельності, фільтрації, мережевих протоколів та ресурсообмежень. У результаті правильно оптимізований Alloy здатен обробляти набагато більше телеметрії, працюючи стабільно, передбачувано та з мінімальним навантаженням на інфраструктуру.

Інтеграція Grafana Alloy у традиційні Linux- або VM-орієнтовані інфраструктури зазвичай проходить найбільш передбачувано, адже саме ці середовища забезпечують найбільший рівень контролю над процесом збору й обробки телеметрії. На відміну від Kubernetes або хмарних платформ, де інструмент необхідно адаптувати під еластичність та динамічність контейнеризованих ворклоадів, у класичних Linux-вузлах Alloy працює стабільно та послідовно, часто виконуючи роль універсального агента телеметрії на кожній віртуальній машині. Проте навіть у таких умовах інтеграція потребує врахування великої кількості нюансів, пов'язаних із безпекою, файловою системою, лог-джерелами, системними сервісами та продуктивністю.

Передусім інтеграція Alloy у Linux-середовища передбачає встановлення самого агента як системного сервісу через `systemd`. Це дозволяє забезпечити автозапуск після перезавантаження, контроль стану, автоматичний рестарт при помилках та можливість централізованого логування. Встановлення, як правило, проходить через статичний двійковий файл або пакет (DEB/RPM), після чого адміністратори створюють `systemd-unit` із чітко визначеним користувачем, директоріями для конфігурації, логів і кешу. Саме такий варіант використання є найбільш зручним у VM-орієнтованих інфраструктурах, де агенти працюють довготривало і виконують функції `filelog`-читання, обробки метрик з `node_exporter` аналогів та передачі трейсів у Tempo.

Особливу увагу в Linux-інтеграції приділяють роботі з файловою системою. Alloy найчастіше отримує логи через `filelog-receiver`, тому необхідно забезпечити коректні доступи до `/var/log`, застосовуючи ACL-списки або групові права. Деякі системні сервіси, такі як `journald`, вимагають або прямого читання через `systemd-`

receiver, або експорту через journalctl. У випадку великих лог-файлів або швидкої ротації важливо налаштовувати позиційні файли, щоб Alloy не читав логи повторно після перезапуску. Для продуктивних Linux-вузлів, де генерується кілька сотень мегабайт логів на годину, часто додають механізми throttling або rate-limit фільтрацію, щоб уникнути перевантаження агента.

У контексті Docker інтеграція проходить дещо інакше. Alloy може працювати як окремий контейнер, використовуючи docker.sock або loki-драйвер для збору логів. У таких конфігураціях потрібно особливо обережно ставитися до прав доступу та ізоляції. Найчастіше контейнер Alloy запускають у режимі прив'язки до хоста через volume-mount для /var/log або journald, що дозволяє збирати логи всіх контейнерів без додаткових драйверів. Деякі використовують OpenTelemetry Logging Driver, але він менш універсальний. Також важливо враховувати, що контейнер Alloy не повинен перевантажувати Docker host своїм CPU-споживанням, тому необхідно застосовувати limits у docker-compose або Swarm, якщо ті використовуються.

У Kubernetes інтеграція проходить уже більш специфічно. Alloy можливий як DaemonSet, що дозволяє розгортати агент на кожному вузлі і забезпечувати збір логів із контейнерів через CRI-лог-файли або journald. У k8s середовищах Alloy часто замінює Promtail, Vector або Fluent Bit. Головна відмінність полягає у необхідності враховувати динамічність pod-ів, тому filelog-receiver повинен бути налаштований під високий churn: часті пересоздання контейнерів, зміни імен лог-файлів, етапи init-pod. Kubernetes також додає необхідність у правильному використанні metadata-enrichment, наприклад додавання namespace, pod_name, container_name, що робить телеметрію більш структурованою та придатною для пошуку у Loki та Grafana.

У підсумку інтеграція Grafana Alloy у Linux/VM, контейнерні та хмарні середовища створює єдину, уніфіковану платформу для збору телеметрії, яка працює передбачувано та стабільно в різних типах інфраструктурних ландшафтів. Завдяки своїй гнучкості Alloy може забезпечувати як класичний filelog-моніторинг на Linux-вузлах, так і масштабований збір телеметрії в Kubernetes, або ж працювати

в Docker для невеликих сервісів. Головне – враховувати особливості кожного середовища і адаптувати конфігурації для оптимальної продуктивності, стабільності та безпеки.

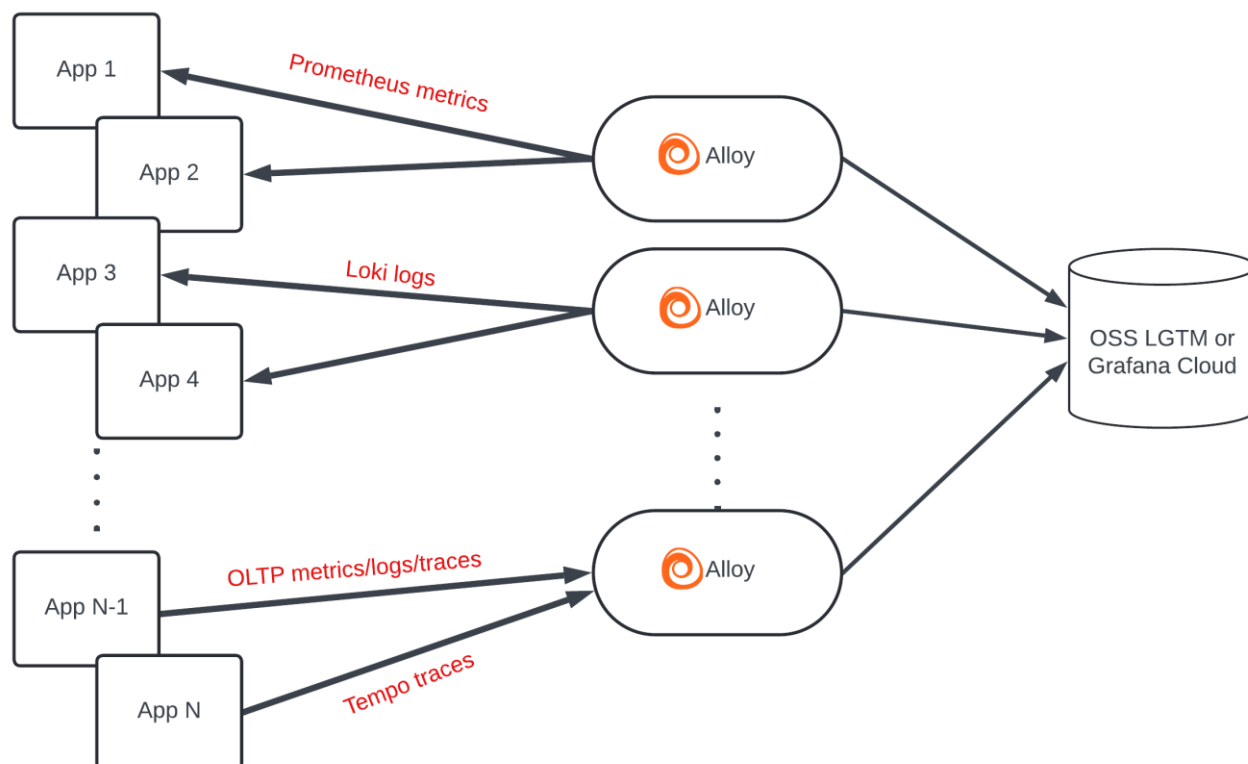


Рис. 3.4. Схема збору телеметрії з різних джерел

Важливо також приділяти увагу спрощенню конфігурацій та оптимізації продуктивності, оскільки саме ці аспекти визначають, наскільки ефективно система буде обробляти телеметричні потоки при високих навантаженнях. Alloy надає можливість зменшити загальну складність телеметрії, об'єднавши функції різних агентів у єдиному універсальному елементі, проте для досягнення стабільної роботи необхідно коректно налаштувати маршрутизацію, буферизацію, відмовостійкість і механізми збереження даних. Тестування інтегрованої системи дозволяє переконатися, що всі частини працюють узгоджено, а кінцеві користувачі отримують повноцінні, доступні та якісні дані у Grafana для аналізу та спостереження.

У підсумку, впровадження Grafana Alloy помітно підвищує рівень телеметрії в організації, забезпечуючи гнучкість, масштабованість і єдність підходів до збору

даних. Правильно виконана інтеграція дозволяє створити надійну базу для моніторингу, трасування та логування, значно спрощує операційні процеси й відкриває можливість подальшого розвитку інфраструктури спостереження без втрати сумісності й якості. Якщо всі кроки реалізовані послідовно й усвідомлено, Grafana Alloy стає не просто компонентом, а фундаментом сучасної, цілісної та ефективної системи Observability.

3.3. Розробка рекомендацій щодо практичної реалізації пайплайнів збору логів та метрик

Побудова ефективних пайплайнів збору логів і метрик у Grafana Alloy передбачає формування цілісної, модульної та стійкої архітектури, яка здатна витримувати високі навантаження, залишаючись при цьому простою у супроводі. Grafana Alloy – це консолідована платформа для обробки телеметрії, що поєднує в собі можливості збору, трансформації, маршрутизації й передачі логів, метрик та навіть трасувань. Тому проектування пайплайнів у Alloy базується на принципах потокової обробки даних, де кожен етап є окремим модулем із чітко визначеними вхідними та вихідними потоками.

Одним із ключових принципів є модульність, тобто кожен пайплайн складається з набору елементарних блоків: джерел, процесорів (обробників), маршрутизаторів і вихідних модулів. Ці блоки незалежні один від одного, що дозволяє гнучко перебудовувати структуру, змінювати логіку обробки або додавати нові модулі без впливу на решту системи. Модульний підхід також дає змогу чітко відокремлювати лог-потоки різного призначення (наприклад, системні, аплікаційні, мережеві) та метрик-потоки різних джерел (експортери, скрейпери Prometheus, бізнес-метрики). Це спрощує не лише архітектуру, але й подальшу експлуатацію.

Другий важливий принцип – потоковість. Alloy не працює з даними пакетно, а здійснює безперервну обробку кожного елемента телеметрії з моменту його появи. Завдяки цьому лог- та метрик-пайплайни легко розширюються, частково або

повністю паралелізуються, а обробка великих обсягів інформації відбувається без блокувань. Пайплайн, побудований за потоковою моделлю, дозволяє одночасно обробляти тисячі подій за секунду, забезпечуючи при цьому низьку затримку передачі даних у Loki чи Prometheus Remote Write.

Принцип декомпозиції та низької зв'язаності відіграє критичну роль у керуваності системи. Alloy надає можливість будувати окремі пайплайни для різних середовищ (prod, stage, dev), різних служб (backend, frontend, DB, queue), а також для різних типів даних. Наприклад, лог-потоки з NGINX потребують іншого парсингу та нормалізації, ніж JSON-логи мікросервісів, тоді як системні journald-логи можуть передаватися без додаткової обробки, лише з мінімальною маршрутизацією. Маючи окремі пайплайни, легше забезпечити контрольований розвиток конфігурацій та уникнути конфліктів у логіці обробки.

Ще один фундаментальний принцип – уніфікація. Навіть якщо дані походять із різних джерел, у продакшн-середовищі необхідно прагнути до єдиного формату повідомлень. Для логів це означає єдину структуру timestamp, однакові ключі JSON-об'єктів, спільний набір стандартних полів (service, environment, host). Для метрик – однакові назви, стандартизовані лейбли, узгоджені семантики типів метрик. Alloy надає широкі можливості для нормалізації даних ще до того, як вони потраплять у Loki або Prometheus, що дозволяє уникнути хаосу в системі моніторингу та підвищує якість аналітики.

Також важливим принципом є відмовостійкість. Пайплайни Alloy мають бути побудовані таким чином, щоб вони не руйнувалися в умовах тимчасових збоїв або пікових навантажень. Це означає використання внутрішніх буферів, механізмів backoff-retry, адаптивної черги, засобів дроселювання й можливості тимчасово зберігати дані локально (особливо актуально для логів). Alloy дозволяє будувати пайплайни так, щоб відмова зовнішнього сервісу (наприклад, Loki або Prometheus) не призводила до втрати телеметрії і не блокувала модулі збору.

Одним із найважливіших концепцій є прозорість і спостережуваність самого пайплайну. Alloy має власні вбудовані метрики, які дозволяють виявляти проблеми, такі як переповнені буфери, помилки парсингу, неправильно

налаштований routing або дроп логів. Побудова пайплайнів без спостережуваності вважається помилкою, оскільки у складних середовищах без моніторингу самого агента важко зрозуміти, де саме втрачаються дані або чому вони затримуються.

Ще один принцип – передбачуваність і стабільність. Пайплайни повинні працювати однаково для однакових даних, незалежно від умов, що дозволяє впевнено використовувати їх у промисловому середовищі. Тому конфігурації Alloy мають бути декларативними, версіонованими та максимально прогнозованими. Використання GitOps або CI/CD-підходу у керуванні конфігураціями додатково підсилює стабільність і полегшує відновлення в разі помилок.

Усі ці принципи – модульність, потоковість, уніфікація, стійкість до збоїв, прозорість, самоспостережуваність та передбачуваність – у комплексі формують правильний підхід до побудови лог- та метрик-пайплайнів у Grafana Alloy. Вони дозволяють побудувати архітектуру, яка не просто “працює”, а здатна витримувати промислові навантаження, легко розширюється, а головне – забезпечує чисту, надійну та структуровану телеметрію, необхідну для якісного моніторингу та спостережуваності сучасних розподілених систем.

Побудова пайплайнів для метрик у Grafana Alloy має багато спільного з підходом до логів, однак сама структура зазвичай виглядає простішою та більш лінійною. Це пов’язано з тим, що метрики, на відміну від логів, уже мають формалізований формат (Prometheus exposition format або OTLP), а їх обробка рідко включає складний парсинг. Основне завдання – правильно організувати збирання, нормалізацію та маршрутизацію потоків метрик так, щоб вони потрапляли у потрібні системи зберігання (Prometheus, Mimir, Tempo Metrics Generator або інший бекенд) у структурованому та однорідному вигляді.

Архітектура пайплайна метрик у Alloy майже завжди складається з трьох основних шарів: джерел даних, процесорів перетворення та механізмів експорту або відправлення. Джерела можуть бути різними – це може бути `prometheus.scrape`, який дозволяє збирати метрики з endpoint-ів застосунків, або `otelcol.receiver.otlp`, який приймає OTLP-метрики з інших агентів. Будь-яка така точка входу формує первинний потік метрик, який Alloy далі з’єднує з обробниками. Наприклад, у

типовому середовищі Linux/VM доволі часто проектують конфігурацію так, щоб сервер сам себе експортує: `node exporter` вбудовується прямо в Alloy.

Приклад такого базового джерела виглядає так:

```
prometheus.scrape "node" {
  targets = ["localhost:9100"]
  forward_to = [prometheus.relabel.node.receiver]
}
```

У цьому випадку всі метрики з `node-exporter` передаються в наступну секцію — зазвичай це `relabel`-блок, який відповідає за приведення метрик до єдиного стандарту. Це важливий етап, оскільки від того, наскільки узгоджені назви лейблів і структуровані ключові атрибути, залежить якість майбутніх запитів у Prometheus/Mimir. Наприклад, можна додати лейбл `host`, що відповідає реальному вузлу:

```
prometheus.relabel "node" {
  rule {
    target_label = "host"
    replacement = "${HOSTNAME}"
  }
  forward_to = [prometheus.remote_write.cluster.receiver]
}
```

Саме `relabel`-етап відрізняє якісний метрик-пайплайн від хаотичного: правильне маркування дозволяє легко уникнути колізій і спростити побудову дашбордів. Наприклад, якщо декілька сервісів експортують однакові метрики, є сенс додати `service`, `env`, `role` або `cluster`, що забезпечує передбачуваність усіх запитів у Grafana.

Наступним кроком є експорт або маршрутизація. Найчастіше метрики відправляються у Prometheus Remote Write або Grafana Mimir, і саме в Alloy ця процедура реалізується прозоро:

```
prometheus.remote_write "cluster" {
  endpoint {
    url = "http://mimir:9009/api/v1/push"
  }
}
```

Цей підхід легко масштабується: можна надсилати метрики у кілька бекендів одночасно або розділяти потоки за типами – наприклад, інфраструктурні метрики надсилати в Mimir, а метрики застосунку – у Prometheus, який працює ближче до сервісу.

Більш складні пайплайни виникають тоді, коли в інфраструктурі використовується OpenTelemetry. У такому випадку підхід стає трохи іншим: Alloy приймає OTLP-метрики, обробляє їх через OTel-процесори й потім конвертує у формат, потрібний бекенду.

Так створюється типова структура OTLP > Alloy > Prometheus/Mimir. Це дозволяє стандартизувати метрики між різними мовами програмування й різними SDK без потреби погоджувати формат на рівні кожного застосунку.

При побудові лог-пайплайнів у Grafana Alloy одним із ключових завдань є організація правильного збору логів із локальної системи. На відміну від класичного Promtail, де конфігурація жорстко прив'язана до scrape_configs, у Alloy аналог функціональності розділено на окремі модулі, які легше комбінувати, масштабувати та інтегрувати з іншими компонентами. Найпоширеніші механізми збору локальних логів – це loki.source.file, loki.source.journal та додаткові модулі, які працюють як універсальні точки входу в систему.

Основна концепція полягає в тому, що Alloy перетворює вхідні джерела логів на стандартизовані потоки, які можуть проходити через будь-які подальші блоки обробки – relabel, мультиплексування, перетворення, шифрування, агрегування або фільтрацію. Це дозволяє створювати пайплайни, які одночасно збирають лог-файли застосунків, системні журнали та нестандартні джерела, але при цьому залишаються однорідними та легко підтримуваними.

Найбільш поширений сценарій – це `scraping` локальних логів файлів. У Alloy цей функціонал реалізується блоком `loki.source.file`. Він працює подібно до Promtail: слідкує за файлами, лічить `offset`, реагує на ротацію, гарантує доставку рядків у правильному порядку. Простий приклад збору логів Nginx може виглядати приблизно так:

```
loki.source.file "nginx" {
  targets = [
    {
      __path__ = "/var/log/nginx/*.log"
      job      = "nginx"
    }
  ]

  forward_to = [loki.process.nginx.receiver]
}
```

У цьому прикладі Alloy використовує `internal target`-лейбли (`__path__`) так само, як і Promtail, тому міграція зі старих конфігурацій відбувається мінімально болісно. Такі пайплайни зазвичай містять подальший процесор, який додає лейбли, фільтрує небажані рядки або перетворює їх у структуровані записи, перш ніж передати їх у Loki.

Не менш важливим джерелом логів у Linux є `journald`. Для системних служб саме `journal` зазвичай є первинним джерелом, а не `plain`-файли. Alloy надає для цього модуль `loki.source.journal`, який може збирати як стандартний системний журнал (`/var/log/journal`), так і кастомні `unit`-логи окремих сервісів. Приклад:

```
loki.source.journal "systemd" {
  path      = "/var/log/journal"
  labels    = { job = "systemd-journal" }
  forward_to = [loki.process.system.receiver]
}
```

Цей підхід особливо корисний у середовищах, де лог-файли фізично не існують, або де важливо зберігати метадані `systemd` (`unit`, `priority`, `facility`), які

втрачаються при лог-експортуванні через `stdout`. Alloy дозволяє не лише витягувати ці метадані, а й використовувати їх для побудови логіки маршрутизації – наприклад, окремо обробляти повідомлення з рівнем `priority <= 3` або виділяти логи певного сервісу за назвою юніту.

Деякі архітектури використовують також модулі `Loki Source` як універсальні точки входу для логів, що приходять із віддалених агентів. Alloy дозволяє створювати вхідні канали, які приймають логи через OTLP або через власний `loki.source.vector`, використовуючи формат `Vector` або інші протоколи. Таким чином, замість того щоб окремо розгортати колекцію агентів у різних середовищах, Alloy може бути центральним вузлом агрегації, що приймає дані як з файлових джерел, так і зі сторонніх інжекторів логів.

Усі локальні джерела найчастіше об'єднуються через процесори, які відповідають за нормалізацію лейблів. Наприклад, універсальна практика – завжди додавати лейбли `host`, `env`, `service`, щоб уникнути конфліктів у `Loki` та забезпечити однаковість логів у масштабованому середовищі:

```
loki.process "common" {
  stage.static_labels {
    values = {
      host = "${HOSTNAME}"
      env  = "production"
    }
  }

  forward_to = [loki.write.main.receiver]
}
```

Завдяки цьому фінальний пайплайн виглядає дуже передбачувано: локальні файли, системний журнал та інші джерела надходять до єдиної точки, проходять однакову обробку й потрапляють до одного або декількох `Loki`-бекендів.

Перевага такого підходу полягає в тому, що Alloy дає можливість створювати багато маленьких, незалежних джерел логів, але зводити їх у стандартизовані архітектурні блоки через `process` або `mixin`-конфігурації. Це дозволяє поступово

нарощувати пайплайн без ризику поламати існуючу інфраструктуру. Крім того, кожне джерело може мати окрему логіку фільтрації – наприклад, відкидати healthcheck-рядки, debug-level події або access-логи з надто великим обсягом.

Таким чином, використання Loki Source, file scraping та journal scraping у Grafana Alloy формує універсальний фундамент для побудови надійних і керованих лог-пайплайнів. Вони дозволяють ефективно об'єднувати різні типи логів у спільну систему, забезпечуючи їх стандартизацію, узгодженість і можливість подальшої аналітики у Loki без значних витрат на налаштування або підтримку.

Парсинг логів є одним із ключових аспектів побудови ефективного лог-пайплайну в Grafana Alloy, адже саме він перетворює сирий текст у структуровані записи, придатні для пошуку, фільтрації, кореляції та побудови аналітики у Loki. На відміну від традиційного Promtail, де пайплайн складається з обмеженого набору pipeline_stages, Alloy дозволяє вибудовувати більш гнучкі й модульні конструкції через блоки loki.process, що відкриває шлях до створення адаптивних і складних парсерів, які реагують на реальний зміст логів.

Найпоширеніший сценарій – це парсинг JSON-логів, які сьогодні використовуються в більшості сучасних застосунків. Alloy дає змогу легко перетворити поле з JSON-рядком у набір ключів і значень:

```
loki.process "json_parse" {
  stage.json {
    source = "message"
  }

  stage.static_labels {
    values = {
      format = "json"
    }
  }

  forward_to = [loki.process.common.receiver]
}
```

У цьому прикладі поле `message` розбирається як JSON, і всі вкладені ключі автоматично додаються як атрибути події. Далі лог потрапляє в загальний блок нормалізації. Такий підхід дозволяє легко зберігати структуровані логі в Loki без втрати інформації та без складного вручну написаного `regex`.

Для логів у форматі Nginx або Apache, де рядки мають чіткий патерн, набагато ефективнішим є парсинг через регулярні вирази.

Такий підхід дозволяє перетворити довгий рядок у структуровані елементи: метод запиту, код відповіді, шлях, IP-адресу клієнта, час обробки. Завдяки цьому в Loki з'являється можливість робити швидкі запити на кшталт:

```
{job="nginx"} | status >= 500
```

або знаходити уповільнені запити, фільтрувати методи, будувати графіки 4xx/5xx тощо.

Складнішим, але дуже важливим завданням є робота з багаторядковими логами — наприклад, `stacktrace` у Java, Python або Go. Alloy дає змогу використовувати `stage`, які поєднують декілька рядків в один лог-запис, якщо вони відповідають певному шаблону. Це важливо, оскільки Loki працює з логами построчно, і неправильна обробка `stacktrace` призведе до втрати контексту. Приклад:

```
loki.process "multiline" {
  stage.multiline {
    firstline = "^\\[.*\\] ERROR"
  }

  forward_to = [loki.process.common.receiver]
}
```

У цьому випадку кожен рядок, що не починається з шаблону `firstline`, буде автоматично приєднано до попереднього. Alloy дозволяє робити це не лише за регулярними виразами, а й на основі `indentation` (відступів), JSON-патернів і навіть евристик, що робить його універсальним інструментом для обробки складних застосунків.

Ще один важливий аспект – нормалізація лейблів. Парсинг дозволяє витягувати значення, але всі ключові поля, які повинні брати участь у фільтрації, варто підняти на рівень лейблів. Наприклад, у Nginx логах часто лейблом роблять status, але не path, оскільки шлях має надто велику кардинальність і буде дорого коштувати в Loki. Alloy дає змогу керувати такими правилами прямо в пайплайні.

Парсинг у Alloy – це також можливість комбінувати кілька джерел в один процесор. Наприклад, окремі модулі можуть розбирати syslog формати різних сервісів, але потім об’єднуватися в єдину структуру метаданих. Це важливо для систем, де застосунки ведуть логи у різних форматах або використовують сторонні бібліотеки логування.

Фінальним етапом пайплайна парсингу є передача структурованих логів у модуль запису loki.write. Alloy дозволяє направляти різні категорії логів у різні Loki кластери або окремі tenant'и. Наприклад, безпекові логи можуть іти в окремий бекенд, тоді як application logs – у стандартний кластер.

Маршрутизація логів у Grafana Alloy відіграє критичну роль під час побудови масштабованих і керованих лог-пайплайнів, оскільки саме від неї залежить, куди будуть передані структуровані події після парсингу, які правила будуть застосовані для відбору логів, та як буде забезпечено розмежування потоків між різними системами та середовищами. Концепція маршрутизації в Alloy значно ширша, ніж проста передача логів “куди треба”: вона дозволяє побудувати складну, ієрархічну логіку розподілу залежно від вмісту, лейблів, джерела або навіть типу події, що робить систему гнучкою та дуже адаптивною до будь-яких корпоративних вимог.

Основна ідея маршрутизації полягає в тому, що після зчитування й обробки логів вони можуть бути спрямовані в різні гілки пайплайна через механізми forward_to. Alloy дозволяє створювати окремі модулі, які відповідають за різні категорії логів: наприклад, технічні логи сервісів, системні події, безпекові audit-логи, або навіть окремі набори логів для тестових і продакшн-середовищ. Використання окремих маршрутизаторів дає змогу уникнути неконтрольованого змішування потоків і забезпечити детальний контроль над тим, які дані й куди зберігаються.

Наприклад, технічні логи застосунку можуть надходити з файлової системи через модуль `local.file` та проходити парсинг JSON, після чого потрапляють у маршрутизатор, який розподіляє їх залежно від рівня логуювання.

Інший поширений сценарій – маршрутизація за середовищем або кластером. Великі компанії часто мають `dev`, `stage`, `prod`, і в межах Kubernetes – декілька `namespace` чи `cluster`.

Це дає змогу передавати продакшн-логи у кластер Loki з підвищеним SLA, тоді як тестові логи можуть відправлятися у `lightweight`-інстанс з коротким `retention`. Маршрутизація також полегшує аудит, оскільки різні набори логів можна зберігати з різними політиками без необхідності дублювати пайплайни.

Не менш важливим є сценарій вибіркового маршрутування – коли певні категорії логів повинні зберігатися, а інші відкидатися або проходити мінімальну обробку. Це дозволяє суттєво зменшити навантаження на Loki та оптимізувати витрати. Наприклад, при роботі з шумнимилогами Nginx можна маршрутизувати тільки 4xx/5xx запити, а все інше – відкидати:

```
stage.match {
  selector = "{status=\"200\"}"
  action = "drop"
}
```

Цей простий прийом дозволяє зменшити обсяг логів у десятки разів, якщо застосунок генерує великий трафік зі статусом 200, але команда зацікавлена лише в аналізі помилок або проблемних ендпоінтів.

В Alloy маршрутизація може відбуватися не тільки за лейблами, а й за довільними полями вмісту. Наприклад, можна проаналізувати JSON-поле `service` і спрямувати логи різних сервісів у різні `backends`, навіть якщо вони читаються з одного й того ж файлу:

```
stage.match {
  selector = "{service=\"auth\"}"
  forward_to = [loki.write.auth.receiver]
}
```

```
stage.match {
```

```
selector = "{service=\"payments\"}"
forward_to = [loki.write.payments.receiver]
}
```

На практиці це зручно під час моніторингу монолітних інфраструктур або сервісів, що використовують спільну інфраструктуру логування.

Важливо також те, що Grafana Alloy дозволяє створювати каскадну маршрутизацію – тобто розподіляти логи не в один, а у кілька наступних пайплайнів, де кожен буде обробляти свій фрагмент даних. Наприклад, той самий потік логів може одночасно відправлятися в основний Loki та у систему зберігання архівних даних із довгим retention.

Нормалізація логів є однією з ключових задач у побудові якісної телеметричної системи, оскільки саме вона забезпечує єдину структуру подій, сумісність між різними джерелами і зручність подальшого пошуку в Loki. У звичайних інфраструктурах логи надходять з різних сервісів, написаних на різних мовах програмування, з різною структурою, форматом часу, рівнями логування, набором ключів у JSON та навіть різними правилами escape-символів. Grafana Alloy дозволяє впорядкувати цей хаос на етапі обробки, щоб у Loki потрапляли уніфіковані та стандартизовані записи.

Оскільки Alloy працює як повноцінний процесор логів, він дозволяє трансформувати дані ще до того, як вони покинуть сервер-джерело. Один із найтиповіших випадків – різний формат timestamp. Наприклад, Python-сервіс може логувати час у ISO 8601, Node.js – у мілісекундах Unix time, а Nginx – у власному текстовому форматі. Нормалізація дозволяє привести всі значення до одного стандарту, наприклад, RFC 3339, щоб у Loki не виникало проблем при сортуванні або побудові часових графіків.

Ще одна важлива частина – уніфікація полів. Багато сервісів використовують різні назви одного й того ж параметра: десь це severity, десь level, десь loglevel. Для побудови зручних дашбордів і запитів у Loki потрібно, щоб рівень логування був завжди однаково представлений, наприклад: level="info". Alloy дозволяє

перейменувати, видаляти або створювати поля, формуючи стандартизований формат незалежно від вихідної структури.

Приклад базової нормалізації рівнів:

```
loki.process "normalize" {
  stage.regex {
    expression = "^(?P<level>(INFO|WARN|ERROR|DEBUG))"
  }
  stage.replace {
    source = "level"
    expression = "WARN"
    replace = "warning"
  }
}
```

Цей простий приклад демонструє, як можна привести нестандартний рівень WARN до уніфікованого warning. У реальних сценаріях нормалізація може включати значно складнішу логіку: мапінги JSON-полів, формування nested-структур, обробку ключів, що містять масиви, або навіть об'єднання декількох полів у нове агреговане значення.

Ще один типовий сценарій – очищення шуму. Наприклад, деякі сервіси логують величезну кількість службових полів, таких як stack traces у debug-режимі або тимчасові технічні ключі, які не мають практичної цінності для аналітики. Alloy дозволяє видаляти їх ще до відправки, зменшуючи навантаження на Loki.

У деяких інфраструктурах виникає проблема дублювання полів: наприклад, коли сервіс вже генерує JSON з полями timestamp і message, але сам лог-файл також містить префіксований час у вигляді plain-text. Alloy дозволяє або перезаписати значення, або вилучити зайву частину. Це не лише знижує обсяг логів, але й робить їх чистішими для подальшої роботи.

Особливо важливою є уніфікація для багатомовних мікросервісів, де частина логів може бути англійською, а частина українською або іншою мовою. Хоч значення полів залишаються мовними, назви ключів можна стандартизувати, щоб

запити у Loki не дублювалися. Наприклад: користувач_id, userID, userid – усе це може бути нормалізовано до одного поля user_id.

Окрім структурних трансформацій, нормалізація включає також контекстну обробку. Наприклад, лог може містити у полі path значення /api/v1/users/123, а ми хочемо зберігати нормалізоване значення /api/v1/users/:id, що дозволить групувати запити за шаблоном, а не за конкретним ідентифікатором. Alloy дозволяє виконувати такий pattern replacement за допомогою регулярних виразів:

```
stage.regex {
  expression = "/api/v1/users/[0-9]+"
  replace    = "/api/v1/users/:id"
}
```

Уніфікація також допомагає на рівні безпеки. Деякі додатки містять у логах персональні дані або іншу конфіденційну інформацію. Alloy може приховати або захешувати такі дані перед тим, як вони будуть передані до Loki [12]:

```
stage.regex {
  expression = "(email=)([A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+)"
  replace    = "${1}[redacted]"
}
```

Таким чином, нормалізація забезпечує не лише структурну цілісність, але й відповідність безпековим політикам.

Таким чином, нормалізація забезпечує не лише структурну цілісність, але й відповідність безпековим політикам.

У підсумку, процес нормалізації та уніфікації логів забезпечує єдиний формат, стандартизовані поля, контрольований обсяг і правильну структурованість перед відправкою до Loki. Це дозволяє будувати більш інформативні дашборди, запускати прогнозування, створювати коректні alerting-правила та значно полегшує роботу аналітиків і SRE-команд, які щоденно працюють із телеметрією.

ВИСНОВКИ

В роботі проведено комплексне дослідження проблеми інтеграції систем збору та обробки телеметрії в сучасних розподілених ІТ-системах, визначено мету роботи та виконано поставлені наукові завдання.

Проведено аналіз проблеми розрізненості даних, що виникає внаслідок експоненційного зростання складності інфраструктур і переходу до мікросервісних архітектур. Визначено, що феномен інформаційних силосів перешкоджає отриманню цілісної картини стану системи та призводить до збільшення показників MTTD і MTTR. Досліджено концепцію спостережуваності, яка ґрунтується на трьох фундаментальних компонентах: логах, метриках і трасуванні. Встановлено, що ключовою особливістю спостережуваності є не просто наявність цих даних, а можливість їхньої кореляції та інтеграції для швидкого переходу від виявлення симптомів до розуміння першопричин інцидентів.

Досліджено методи та засоби стандартизації збору телеметрії. Визначено, що найбільш перспективним підходом є використання відкритого стандарту OpenTelemetry, який уніфікує формати та протоколи передачі даних. На основі цього аналізу розглянуто технологію інтеграції на базі Grafana Alloy – сучасної платформи, що об'єднує функції збору, обробки та маршрутизації всіх типів телеметрії в єдиній системі.

Аналіз архітектури Grafana Alloy показав, що рішення базується на трирівневій моделі обробки:

1. Рівень приймання: забезпечує отримання даних з різноманітних джерел (systemd, Prometheus endpoints, OTLP-агенти тощо).
2. Рівень обробки : реалізує логіку фільтрації, нормалізації, збагачення метаданими та агрегації телеметрії.
3. Рівень експорту: відповідає за передачу оброблених даних у цільові системи зберігання, такі як Grafana Loki, Mimir та Tempo.

Особливістю Grafana Alloy є використання повністю декларативної мови конфігурації у форматі YAML, що дозволяє описувати потоки даних як код (IaC)

та будувати складні розгалужені пайплайни (fan-out). Порівняльний аналіз з альтернативними рішеннями (Prometheus, Fluent Bit, Vector) підтвердив, що Alloy є найбільш універсальним агентом, який усуває необхідність використання великої кількості розрізнених інструментів на одному вузлі.

На основі досліджень, проведених у роботі, запропоновано стратегію розгортання та масштабування Grafana Alloy для побудови високонадійної системи спостережуваності. Рекомендована архітектура ґрунтується на принципах безстановості (stateless), що дозволяє здійснювати горизонтальне масштабування та забезпечувати високу доступність (HA) за балансувальниками навантаження.

В роботі розроблено практичні рекомендації щодо реалізації пайплайнів збору логів та метрик. Визначено ключові етапи обробки даних, такі як парсинг та структурування, нормалізація та уніфікація, умовна маршрутизація та оптимізація продуктивності.

Окрему увагу приділено безпеці, що включає обов'язкове TLS-шифрування каналів зв'язку та приховування конфіденційної інформації в логах на етапі обробки. Розроблено порядок тестування інтегрованої системи, що включає перевірку під навантаженням та контроль цілісності даних при відмовах бекенд-сервісів.

Таким чином, впровадження Grafana Alloy як уніфікованого шару інтеграції телеметрії дозволяє організаціям подолати проблему розрізненості даних, спростити управління інфраструктурою спостережуваності та суттєво підвищити ефективність захисту й підтримки працездатності сучасних ІТ-ресурсів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Incident Management - MTBF, MTTR, MTTA, and MTTF [Електронний ресурс] // Atlassian. — URL: <https://www.atlassian.com/incident-management/kpis/common-metrics> (дата звернення: 10.10.2025).
2. Що таке спостережуваність і навіщо потрібні ІТ для управління гібридним хмарним середовищем? [Електронний ресурс] // Oberig IT. — URL: <https://oberig-it.com/statti/shho-take-sposterezhuvanist-i-navishho-potribni-it-dlya-upravlinnya-gibrydnym-hmarnym-seredovyshhem/> (дата звернення: 10.10.2025).
3. Observability vs. Monitoring: What Really Matters for System Reliability [Електронний ресурс] // EPAM Careers. — URL: <https://careers.epam.ua/blog/observability-vs-monitoring-what-really-matters-for-system-reliability> (дата звернення: 21.10.2025).
4. Metric types [Електронний ресурс] // Prometheus Documentation. — URL: https://prometheus.io/docs/concepts/metric_types/ (дата звернення: 25.11.2025).
5. What Is Observability? Key Components & Best Practices [Електронний ресурс] // Honeycomb. — URL: <https://www.honeycomb.io/blog/what-is-observability-key-components-best-practices> (дата звернення: 25.11.2025).
6. What is OpenTelemetry? [Електронний ресурс] // OpenTelemetry Documentation. — URL: <https://opentelemetry.io/docs/what-is-opentelemetry/> (дата звернення: 21.12.2025).
7. OTLP Specification [Електронний ресурс] // OpenTelemetry Documentation. — URL: <https://opentelemetry.io/docs/specs/otlp/> (дата звернення: 27.11.2025).
8. Elastic Common Schema (ECS) Reference [Електронний ресурс] // Elastic Documentation. — URL: <https://www.elastic.co/docs/reference/ecs> (дата звернення: 15.12.2025).
9. Components [Електронний ресурс] // Grafana Alloy documentation. — URL: <https://grafana.com/docs/alloy/latest/get-started/components/> (дата звернення: 15.12.2025).
10. Configuration language reference [Електронний ресурс] // Grafana Alloy documentation. — URL: <https://grafana.com/docs/alloy/latest/reference/> (дата звернення: 17.12.2025).
11. Monitor an app with Kubernetes Monitoring [Електронний ресурс] // Grafana Labs Tutorials. — URL: <https://grafana.com/tutorials/k8s-monitoring-app/> (дата звернення: 21.12.2025).
12. Redaction processor [Електронний ресурс] // GitHub. open-telemetry/opentelemetry-collector-contrib. — URL: <https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/processor/redactionprocessor> (дата звернення: 21.12.2025).