ЄСІКС

Архітектура ЦОД

Настанови системного архітектора для проектування кластерів ЦОД

Версія системи:	2024.6.15
Кластер:	synrc-control-plane (arm64)
Домен:	1 (erp.uno)
Неймспейси:	6 просторів (infra, telemetry, security, services, apps, ai)
Компоненти:	20 сервісів
Дата випуску:	04 липня 2026

2026 © Інформаційні Судові Системи

Зміст

Зміст

1 Історія впроваджень в органах сектору безпеки **2**

2 Оркестрація Erlang-кластерів у Kubernetes **2**
2.1 Динамічний пошук нод (Peer Discovery) . . . 2
2.2 Порти та дистрибуція через TLS . . . 3
2.3 Helm та Service Mesh (Istio) . . . 3

3 Політики інфраструктурного розгортання **3**
3.1 Політика віртуалізації (Proxmox VE) . . . 3
3.2 Політика оркестрації кластера (Kubespray) . . . 3
3.3 Політика доставки додатків (ArgoCD) . . . 4
3.4 Політика керування Helm-діаграмами . . . 4
3.5 Політика керування DNS (synrc/ns) . . . 4
3.6 Політика базових образів контейнерів (Alpine Linux) . . . 4

4 Покомпонентне розгортання ІКС та Служб ERP/1 через Helm **5**
4.1 Пакети прикладних продуктів ІКС ERP/1 . . . 5
4.2 Пакети інфраструктурних служб ERP/1 (SLUB) . . . 5
4.3 Специфікація інфраструктури та сервісів . . . 6
4.4 Аналіз мета-схеми Kubernetes та мінімальний набір об'єктів . . . 6
4.4.1 Використання StatefulSet (STS) для Mnesia/KVS . . . 7
4.4.2 Конфігурація HorizontalPodAutoscaler (HPA) . . . 7
4.5 Ієрархія конфігурацій (values.yaml) . . . 8
4.6 Відповідність репозиторіїв GitHub, ArgoCD та Helm . . . 8

5 Модель асинхронного ETL-процесингу черг на базі KVS **9**
5.1 Компоненти черги: Воркери, Курсори та Контексти . . . 9
5.2 Специфікація структур даних (Erlang) . . . 10
5.3 Алгоритм ETL-циклу воркера . . . 10

6 Практичні кейси та аналіз відмов (Post-Mortem) **11**
6.1 Кейс 1: Відбиття масованої DDoS-атаки на веб-портал реєстрів . . . 11
6.2 Кейс 2: Розділення мережі (Split-Brain) в Mnesia кластері . . . 11

7 Додаток. Інструкція розгортання на Ubuntu 24.04 LTS **12**
7.1 Крок 1. Конфігурація та запуск DNS-сервера synrc/ns . . . 12
7.2 Крок 2. Налаштування віртуалізації у Proxmox VE . . . 12
7.3 Крок 3. Встановлення Kubernetes за допомогою Kubespray . . . 13
7.4 Крок 4. Створення та обслуговування Helm-репозиторію . . . 14
7.5 Крок 5. Налаштування власного Docker Registry . . . 14
7.6 Крок 5а. Збірка образів додатків на базі Alpine Linux (Dockerfile) . . . 15
7.7 Крок 6. Налаштування GitOps доставки через ArgoCD . . . 16

8 Тестові питання та завдання **17**

Анотація

Технічний блюпрінт ЦОД.

1. Історія впроваджень в органах сектору безпеки

Компоненти та архітектурні рішення платформи «ERP/1» успішно пройшли етапи проектування, розгортання та тривалої промислової апробації у державних інформаційних системах органів сектору безпеки та оборони України:

- **ЄРЗ** (Єдиний реєстр зброї Національної поліції України) — ведення дозвільної системи, облік цивільної зброї, інтеграція з кабінетами громадян.
- **СУСЗЦЗ** (Система управління силами та засобами цивільного захисту Державної служби України з надзвичайних ситуацій) — координація чергових змін, оперативне реагування, моніторинг ресурсів.
- **ЄІС МВС** (Єдина інформаційна система Міністерства внутрішніх справ) — інтеграційна шина та обмін даними між відомчими реєстрами.
- **ФП МТРЗ** (Функціональна підсистема матеріально-технічного та ресурсного забезпечення МВС) — автоматизація логістики, обліку ТМЦ та державного замовлення.
- **ГСЦ МВС** (Головний сервісний центр) — оптимізація черг, облік посвідчень водія та транспортних засобів.

2. Оркестрація Erlang-кластерів у Kubern

Для промислового розгортання та автоматичного масштабування (autoscaling) нод бекенду в хмарних та локальних середовищах застосовується контейнеризація на базі Docker та оркестрація в Kubernetes.

2.1. Динамічний пошук нод (Peer Discovery)

Оскільки Erlang-вузли вимагають постійної зв'язності для побудови розподіленого кластера Mnesia, використовується бібліотека `libcluster` з DNS-пошуком через headless сервіси Kubernetes:

```
config :libcluster,
  topologies: [
    k8s_erp: [
      strategy: Cluster.Strategy.Kubernetes.DNS,
      config: [
        service: "erp-backend-headless",
        application_name: "erp"
      ]
    ]
  ]
```

2.2. Порти та дистрибуція через TLS

Для безпеки розподілена взаємодія нод Erlang здійснюється через шифрований TLS-канал. Для цього налаштовується генерація сертифікатів для кожної ноди при старті контейнера та прокидаються наступні порти:

- **4369** — порт служби `epmd` (Erlang Port Mapper Daemon);
- **9100–9115** — діапазон портів для міжнодової дистрибуції.

2.3. Helm та Service Mesh (Istio)

Для спрощення розгортання розроблені Helm-діаграми (Helm Charts), які містять описи StatefulSet для баз даних (Mnesia) та Deployment для безстатусових серверів сесій. Istio Service Mesh інтегрується для тонкого налаштування трафіку (Traffic Splitting), моніторингу затримок (Telemetry) та взаємної mTLS авторизації між клієнтськими сервісами.

3. Політики інфраструктурного розгортання

Для забезпечення високої доступності, передбачуваності та автоматизації оновлень платформи «ERP/1» впроваджено суворі політики розгортання на базі концепції GitOps та підходу «Інфраструктура як код» (IaC). Технологічний стек спроектований за принципом мінімальної кількості рухомих частин та максимальної автономності.

3.1. Політика віртуалізації (Proxmox VE)

Усі обчислювальні ресурси платформи розгортаються на базі гіпервізора Proxmox VE з дотриманням таких вимог:

- **Мінімалістичний оверхед**: Віртуальні машини створюються виключно за технологією Thin Provisioning (LVM-Thin або ZFS) для економного споживання дискового простору та підтримки швидких знімків.
- **Декларативність та шаблони**: Впроваджено використання незмінних шаблонів ОС Ubuntu LTS з підтримкою Cloud-Init для автоматичного налаштування мережі, SSH-ключів та системних параметрів.
- **Резервування та мережі**: Ноди керування та ноди даних розподіляються по різних фізичних серверах Proxmox (анти-афініті політики). Мережева топологія ізолюється за допомогою віртуальних мостів (Linux Bridges) та VLAN.

3.2. Політика оркестрації кластера (Kubespray)

Створення, конфігурування та оновлення версій Kubernetes-кластерів виконується за допомогою Kubespray:

- **Мінімальний набір плагінів**: Виключаються сторонні хмарні інтеграції. Використовуються стандартні компоненти: `containerd` як контейнерне середовище виконання, інтеграція з `systemd` та локальні дискові сховища.

- **Контроль версій**: Конфігураційні інвентарі Ansible (hosts.yaml) та змінні кластера (group_vars) зберігаються у центральному Git-репозиторії інфраструктури.
- **Мережеві політики**: Використовується Calico CNI з активованим NetworkPolicies для ізоляції мікросервісів у межах просторів імен Kubernetes.

3.3. Політика доставки додатків (ArgoCD)

Впроваджено модель Pull-based GitOps для управління станом додатків:

- **Єдине джерело істини**: Жодна зміна конфігурації чи коду не вноситься в кластер вручну через `kubectl`. Усі ресурси Kubernetes декларуються в Git.
- **Дрейф конфігурацій (Configuration Drift)**: ArgoCD здійснює безперервний моніторинг стану кластера. У разі виявлення розбіжностей між Git та кластером автоматично ініціюється процес синхронізації (Self-Healing) або надсилається сповіщення.
- **Декларативне керування додатками**: Використовується структура ресурсів типу `Application` для опису зв'язку між Git-репозиторієм та цільовим простором імен.

3.4. Політика керування Helm-діаграмами

Усі компоненти платформи пакуються у Helm-діаграми та зберігаються у приватному репозиторії:

- **Календарне версіонування (за принципом N2O)**: Замість класичного SemVer використовується формат версій `X.Y.Z`, де `X` — кількість «нових років», які зустрів продукт з моменту створення, `Y` — номер місяця релізу, а `Z` — день релізу або порядковий номер збірки в межах цього місяця. Використання тегів `latest` заборонено.
- **Локальне сховище**: Helm-діаграми зберігаються у власному захищеному HTTP/OCI-репозиторії, що гарантує автономність доставки у закритих контурах без доступу до глобальної мережі.

3.5. Політика керування DNS (synrc/ns)

Навігація та резолвінг імен всередині інфраструктури виконується за допомогою власного легковагого DNS-сервера `synrc/ns`:

- **Автономність**: DNS-сервер написаний на Erlang/Elixir, інтегрується безпосередньо в мережевий контур та працює з мінімальним використанням RAM (до 20 МБ).
- **Декларативні зони**: Усі записи доменних зон зберігаються у вигляді JSON-файлів у Git-репозиторії та автоматично оновлюються при зміні IP-адрес чи додаванні нових сервісів.

3.6. Політика базових образів контейнерів (Alpine Linux)

З метою мінімізації вектора атак, економії дискового простору та підвищення швидкості масштабування (cold start) нод у Kubernetes, для побудови Docker-контейнерів BEAM-додатків діє політика використання легковагого базового образу **Alpine Linux** (версії `alpine:3.20`):

- **Мінімальний обсяг образів**: Використання Alpine Linux замість повноважних дистрибутивів на кшталт Ubuntu Server дозволяє зменшити розмір базового шару образу з ~80 МБ до ~5–8 МБ.

- **Безпека**: Базовий образ містить мінімальну кількість утиліт та бібліотек, що знижує вразливість системи перед CVE-загрозами. Заборонено встановлення компіляторів чи пакетних менеджерів (наприклад, `apk`) у фінальних продуктових образах.
- **Статична лінковка**: Усі NIF-бібліотеки на C/Rust, які завантажуються Erlang-додатками, компилюються під бібліотеку `musl` C, вбудовану в Alpine.

4. Покомпонентне розгортання IKC та Служб ERP/1 через Helm

Для забезпечення модульності та гнучкості керування інфраструктурою, кожен прикладний продукт екосистеми **IKC ERP/1** та кожна складова частина **Служб безпечного зв'язку (SLUB) ERP/1** поставляються як окремі незалежні Helm-діаграми. Це дозволяє розгортати лише необхідні модулі залежно від призначення конкретного контуру (наприклад, окремо освітній контур чи медичний сервіс).

4.1. Пакети прикладних продуктів IKC ERP/1

Усі 10 прикладних продуктів IKC мають власні Helm-діаграми:

- `lms-education` (Освіта) — автоматизоване розгортання LMS-сервісу, сервісів планування занять та інтерфейсів інтеграції з ЄДЕБО.
- `hl7-health` (Здоров'я) — MIC-сервіси, що підтримують HL7 FHIR API та адаптери синхронізації з eHealth.
- `crm-documents` (Документи) — служба електронного документообігу та сховищ файлів, що тісно взаємодіє з рушієм BPE.
- `acc-accounting` (Облік) — сервіси бухгалтерії, розрахунку зарплат та кадрового обліку з підтримкою Mnesia.
- `wms-warehouse` (Склад) — служба адресного зберігання ТМЦ та логістичних операцій.
- `cart-registers` (Реєстри) — low-code рушій реєстрів, що містить інтерфейси підключення до Трембіти.
- `ai-generation` (Істотність) — сервіс семантичного аналізу та інференсу локальних LLM з підтримкою прокидання GPU (NVIDIA CUDA).
- `olap-analytics` (Аналітика) — аналітичне DWH-сховище на базі DuckDB/MonetDB.
- `pm-projects` (Проєкти) — локальна система управління задачами (Jira-alternative) та Wiki-сервер.
- `itsm-incidents` (Service Desk) — диспетчер інцидентів та SLA-контролер.

4.2. Пакети інфраструктурних служб ERP/1 (SLUB)

Служби комунікацій та безпеки (SLUB) забезпечують роботу транспортного та криптографічного контуру платформи:

- `ns-dns` — легковагий локальний DNS-сервер для автономного резолвінгу імен.
- `ca-pki` — локальний центр сертифікації для випуску ключів X.509.

- `vpn-wireguard` — VPN-шлюз для захисту міжнодових каналів зв'язку.
- `ldap-directory` — ієрархічний реєстр користувачів та організацій з підтримкою ABAC.
- `ias-auth` — служба ідентифікації, автентифікації та перевірки статусів сертифікатів (OCSP).
- `chat-messenger` — високопродуктивний WebSocket брокер реального часу.
- `mail-delivery` — поштовий сервер для асинхронного транспорту офіційної кореспонденції.
- `rest-bpe` — REST API шлюз для керування бізнес-процесами на базі рушія BPE.
- `abac-clearance` — Go-сервер для авторизації, перевірки допусків та збору аудит-логів доступу.
- `mach-ivr` — сценарний рушій станів телефонії та автоінформатора (IVR).
- `bpe-engine` — BPMN-рушій бізнес-процесів платформи.
- `kvs-database` — вбудоване сховище даних на базі СУБД KVS.
- `nitro-portal` — веб-портал рендерингу інтерфейсів.
- `n2o-server` — WebSocket-сервер додатків.
- `faiss-search` — векторний пошуковий рушій для інтелектуальних сервісів штучного інтелекту.
- `prometheus` — збір та збереження часових рядів метрик додатків.
- `grafana` — візуалізація метрик та побудова оперативних дашбордів моніторингу.
- `otel-collector` — OpenTelemetry шлюз збору уніфікованих метрик, логів та трасувань.

4.3. Специфікація інфраструктури та сервісів

Для координації мікросервісів приведемо повну специфікацію портів, протоколів та типу збереження стану (statefulness) компонентів платформи (табл. 1).

4.4. Аналіз мета-схеми Kubernetes та мінімальний набір об'єктів

Для забезпечення життєвого циклу платформи в Kubernetes використовується строго обмежений мінімальний набір об'єктів мета-схеми API. Цей мінімум є оптимальним для ізольованих контурів і складається з:

- **Namespace** — ізоляція ресурсів платформи.
- **ConfigMap та Secret** — декларативне конфігурування та збереження ключів/сертифікатів.
- **Service** — логічна маршрутизація трафіку та внутрішній DNS-резолвінг.
- **Ingress** — зовнішній доступ до вебинтерфейсів через Ingress-контролер.
- **Deployment** — розгортання Stateless-мікросервісів.
- **StatefulSet (STS)** — управління Stateful-нодами СУБД та систем моніторингу.
- **PersistentVolumeClaim (PVC)** — динамічне виділення дискового простору.
- **HorizontalPodAutoscaler (HPA)** — динамічне масштабування подів під навантаженням.

Компонент	Namespace	Призначення	Порт	Стан
Інфраструктурні служби				
ns-nameserver	core	DNS-сервер дозволу імен	53:8101	Є
vpn-wireguard	core	VPN-шлюз міжнодового зв'язку	51820:8102	Є
ca-authority	security	Локальний PKI та випуск ключів	8201	Є
ldap-directory	security	ABAC-директорія користувачів	389:8202	Є
abac-clearance	security	Мандати та аудит логів доступу	8203	Є
ias-authorization	security	Автентифікація користувачів	8204	Є
kvs-database	database	Розподілене сховище KVS	8301	Є
bpe-workflow	database	Рушій процесів BPMN	8302	Є
rest-openapi	web	BPE REST API шлюз	8303	Нема
mach-ivr	web	Телефонія та автоінформатор (IVR)	8509	Нема
nitro-portal	web	Веб-інтерфейс рендерингу (nitro)	8510	Нема
n2o-server	web	WebSocket-сервер додатків (n2o)	8511	Нема
prometheus	telemetry	Моніторинг та збір метрик	8401	Є
grafana	telemetry	Візуалізація та дашборди метрик	8402	Нема
open-telemetry	telemetry	OpenTelemetry шлюз збору OTLP	8404:8405:8406	Нема
Прикладні інформаційно-комунікаційні системи (ІКС)				
cart-registers	erp	Low-code рушій державних реєстрів	8501	Є
hl7-health	erp	MIC-сервіси та FHIR API (eHealth)	8502	Є
crm-documents	erp	Електронний документообіг	8503	Є
acc-accounting	erp	Бухгалтерський та кадровий облік	8504	Є
wms-warehouse	erp	Склад адресного зберігання ТМЦ	8505	Є
lms-education	erp	LMS-сервіси та інтеграція з ЄДЕБО	8506	Є
chat-messenger	erp	WebSocket брокер реального часу	8507	Є
mail-delivery	erp	Поштовий сервер транспорту	8508	Є
olap-analytics	erp	DuckDB/MonetDB сховище	8512	Є
itsm-incidents	erp	Диспетчер інцидентів та SLA	8513	Є
pm-projects	erp	Управління задачами та листами	8514	Є
Сервіси штучного інтелекту (AI & GPU)				
ai-generation	ai	Інференс LLM	8601	Нема
faiss-search	ai	Векторний пошук та ШІ-сервіси	8602	Нема

Табл. 1: Специфікація інфраструктури та типів розгортання компонентів

4.4.1. Використання StatefulSet (STS) для Mnesia/KVS

Для розподіленої СУБД Mnesia та KVS вкрай важливо зберігати стабільний мережевий ідентифікатор хоста при перезапуску та стабільний зв'язок з його дисковим сховищем. StatefulSet гарантує унікальні імена нод (`kvs-database-0`, `kvs-database-1`) та їх послідовний запуск і зупинку, що запобігає виникненню аварійних станів Split-Brain.

4.4.2. Конфігурація HorizontalPodAutoscaler (HPA)

Для амортизації різких стрибків користувацької активності на рівні вебпорталів застосовується HPA. Нижче наведено мінімальний приклад налаштування авто-масштабування для сервісу `crm-documents` на базі використання CPU:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: crm-documents-hpa
  namespace: erp
spec:
```

```
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: crm-documents
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 75
```

4.5. Ієрархія конфігурацій (values.yaml)

Для спрощення адміністрування використовується парадигма Umbrella-chart, який об'єднує всі підкомпоненти через залежності. Приклад налаштування файлу `values.yaml`:

```
global:
  domain: erp.uno
  environment: production
  tls:
    enabled: true
    secretName: erp-tls-cert

crm-documents:
  enabled: true
  replicaCount: 2
  resources:
    limits:
      cpu: 1000m
      memory: 2Gi
  bpe:
    engine: enabled

ca-pki:
  enabled: true
  storage:
    size: 10Gi
    storageClass: local-path
```

4.6. Відповідність репозиторіїв GitHub, ArgoCD та Helm

Для забезпечення прозорості автоматизованої доставки GitOps-процесом ArgoCD, нижче наведено таблицю відповідності вихідних GitHub-репозиторіїв коду додатків та назв відповідних ArgoCD-додатків / Helm-компонентів:

GitHub репозиторій	Додаток ArgoCD / Helm-компонент
synrc/ns	ns-dns
synrc/ca	ca-pki
zencrypted/vpn	vpn-wireguard
synrc/ldap	ldap-directory
zencrypted/ias	ias-auth
synrc/chat	chat-messenger
erpuno/mail	mail-delivery
synrc/rest	rest-bpe
zencrypted/clearance	abac-clearance
synrc/mach	mach-ivr
synrc/bpe	bpe-engine
synrc/kvs	kvs-database
synrc/nitro	nitro-portal
synrc/n2o	n2o-server
synrc/faiss	faiss-search
prom/prometheus	prometheus
grafana/grafana	grafana
otel/otel-collector	otel-collector
erpuno/edu	lms-education
erpuno/health	hl7-health
erpuno/crm	crm-documents
erpuno/acc	acc-accounting
erpuno/warehouse	wms-warehouse
erpuno/cart	cart-registers
erpuno/ai	ai-generation
erpuno/olap	olap-analytics
erpuno/pm	pm-projects
erpuno/itsm	itsm-incidents

Табл. 2: Таблиця відповідності репозиторіїв та компонентів GitOps

5. Модель асинхронного ETL-процесингу черг на базі KVS

У високонавантажених державних системах транзакційна обробка реального часу (OLTP) відокремлена від аналітичної обробки та побудови звітів (OLAP). Для цього в архітектурі ERP/1 реалізовано модель асинхронного ETL (Extract, Transform, Load) процесингу транзакційних логів на базі черг вбудованого сховища KVS.

5.1. Компоненти черги: Воркери, Курсори та Контексти

Процес асинхронної обробки базується на трьох сутностях:

- **Асинхронний воркер** — ізольований BEAM-процес, що виконує циклічну обробку повідомлень у черзі.
- **Курсор (Cursor)** — запис у сховищі KVS, який фіксує ідентифікатор останньої успішно обробленої транзакції для конкретного воркера. Це забезпечує семантику доставки «щонайменше один раз» (At-least-once).
- **Контекст (Context)** — структура даних воркера, що зберігає параметри з'єднання з цільовою БД,

сесійні ключі, ліміти пакетів (batch size) та поточний стан.

5.2. Специфікація структур даних (Erlang)

Модель черг та курсорів описується такими Erlang-записами:

```
-record(kvs_cursor, {
    id,
    last_read_id,
    updated_at
}).

-record(etl_context, {
    worker_id,
    stream_id,
    cursor,
    batch_size = 100,
    target_db,
    transform_fun
}).
```

5.3. Алгоритм ETL-циклу воркера

Асинхронний вокер працює за таким циклічним алгоритмом:

1. **Extract (Вилучення)** — воркер зчитує чергову порцію записів з KVS-стріму, починаючи з позиції `last_read_id`, збереженої в курсорі.
2. **Transform (Трансформація)** — отримані сирі бінарні дані розкодовуються з ASN.1 DER, перевіряються на цілісність (підпис КЕП) та приводяться до реляційного або стовпчикового вигляду.
3. **Load (Завантаження)** — трансформований пакет записується в аналітичне OLAP-сховище (DuckDB) через швидкі NIF-порти. Після успішного запису курсор воркера оновлюється в KVS в межах однієї транзакції.

Приклад мінімальної реалізації ETL-циклу на Erlang:

```
-module(etl_worker).
-export([process_loop/1]).

process_loop(Context) ->
    Cursor = Context#etl_context.cursor,
    LastId = Cursor#kvs_cursor.last_read_id,
    StreamId = Context#etl_context.stream_id,
    Limit = Context#etl_context.batch_size,
    case kvs:get_range(StreamId, LastId, Limit) of
        [] ->
            timer:sleep(1000),
            process_loop(Context);
        Records ->
            Transformed = [ (Context#etl_context.transform_fun)(R) || R <- Records ],
            case load_to_olap(Context#etl_context.target_db, Transformed) of
                ok ->
                    LastRecord = lists:last(Records),
```

```
                NewLastId = element(2, LastRecord),
                NewCursor = Cursor#kvs_cursor{
                    last_read_id = NewLastId,
                    updated_at = erlang:system_time(second)
                },
                kvs:put(NewCursor),
                process_loop(Context#etl_context{cursor = NewCursor});
            {error, Reason} ->
                logger:error("ETL Load failed: ~p", [Reason]),
                timer:sleep(5000),
                process_loop(Context)
        end
end.
```

6. Практичні кейси та аналіз відмов (Post-Mortem)

6.1. Кейс 1: Відбиття масованої DDoS-атаки на веб-портал реєстрів

Опис інциденту: Під час запуску оновленого кабінету власника зброї система зазнала HTTP-флуду обсягом до 150,000 запитів за секунду, що призвело до вичерпання пулу з'єднань на рівні веб-серверів.

- **Аналіз відмови**: Базовий ліміт TCP акцепторів був встановлений у 25,000. Процеси-обробники блокували пул Mnesia через повільні клієнтські запити.
- **Вирішення**: Обмеження швидкості (Rate Limiting) перенесено на рівень Istio Ingress Gateway. Пул акцепторів збільшено до 200,000, а парсинг вхідних JSON-пакетів перекладено з Elixir-інтерпретатора на C99 NIF парсери, що знизило навантаження на CPU з 95% до 12%.

6.2. Кейс 2: Розділення мережі (Split-Brain) в Mnesia кластері

Опис інциденту: Внаслідок аварії на комутаторі датацентру виникло розділення мережі між основними нодами кластера, що призвело до паралельного запису даних у різні гілки Mnesia.

- **Аналіз відмови**: Вимкнена автоматична реконсиляція Mnesia призвела до розходження стану таблиць транзакцій.
- **Вирішення**: Налаштовано обробник подій `mnesia_down`. Впроваджено скрипт автоматичного злиття гілок (reconciliation) на основі часових міток транзакцій та перепідключення розділених нод з перезапуском реплікації без втрати даних користувачів.

7. Додаток. Інструкція розгортання на Ubuntu 24.04 LTS

У цьому додатку наведено покрокову інструкцію для мінімалістичного розгортання платформи ERP/1 на Ubuntu 24.04 LTS з використанням затвердженого стеку.

7.1. Крок 1. Конфігурація та запуск DNS-сервера synrc/ns

DNS-сервер розгортається як інфраструктурний вузол (IP: `10.0.0.5`) для забезпечення локального розв'язання імен домену `erp.uno`.

1. Встановіть середовище виконання Erlang та Elixir:

```
sudo apt update
sudo apt install -y erlang-dev elixir git
```

2. Клонуйте репозиторій та підготуйте проект:

```
git clone https://github.com/synrc/ns.git /opt/synrc-ns
cd /opt/synrc-ns
mix deps.get
```

3. Створіть конфігураційний файл зони `priv/erp.zone.config`:

```
{ttl, 3600}.
{soa, "erp.uno", "ns1.erp.uno", "admin.erp.uno", 2026070101, 86400, 7200, 604800, 300}.
{ns, "erp.uno", "ns1.erp.uno"}.
{a, "ns1.erp.uno", "10.0.0.5"}.
{a, "k8s-control.erp.uno", "10.0.0.10"}.
{a, "k8s-worker1.erp.uno", "10.0.0.11"}.
{a, "charts.erp.uno", "10.0.0.5"}.
{a, "argocd.erp.uno", "10.0.0.10"}.
```

4. Відредагуйте `config/config.exs`, вказавши шлях до файлу зони та налаштувавши прослуховування на стандартному порті DNS:

```
import Config
config :ns,
  servers: [
    [{:name, :inet_dns}, {:address, ~c"0.0.0.0"}, {:port, 53}, {:family, :inet}]
  ],
  zones: ~c"priv/erp.zone.config"
```

5. Запустіть DNS-сервер в інтерактивній консолі:

```
sudo iex -S mix
```

7.2. Крок 2. Налаштування віртуалізації у Proxmox VE

1. Завантажте офіційний образ Ubuntu 24.04 Cloud-Image на хості Proxmox:

```
wget https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-amd64.img
```

2. Створіть шаблон віртуальної машини (VM ID: 9000):

```
qm create 9000 --memory 2048 --cores 2 --name u24-cloud --net0 virtio,bridge=vmbr0
qm importdisk 9000 noble-server-cloudimg-amd64.img local-lvm
qm set 9000 --scsihw virtio-scsi --scsi0 local-lvm:vm-9000-disk-0
qm set 9000 --ide2 local-lvm:cloudinit
qm set 9000 --boot c --bootdisk scsi0
qm set 9000 --serial0 socket --vga serial0
qm template 9000
```

3. Створіть дві ноди кластера шляхом клонування шаблону:

```
qm clone 9000 100 --name k8s-control
qm clone 9000 101 --name k8s-worker1
```

Через інтегляційне Cloud-Init меню вкажіть IP-адреси нод (`10.0.0.10/24` та `10.0.0.11/24`), а як DNS-сервер задайте IP нашого DNS `10.0.0.5`.

7.3. Крок 3. Встановлення Kubernetes за допомогою Kubespray

1. Встановіть залежності Ansible на машині управління:

```
sudo apt install -y python3-pip python3-venv git
python3 -m venv venv
source venv/bin/activate
pip install ansible
```

2. Клонуйте Kubespray та встановіть вимоги:

```
git clone https://github.com/kubernetes-sigs/kubespray.git
cd kubespray
pip install -r requirements.txt
```

3. Створіть файл інвентарю `inventory/mycluster/hosts.yaml`:

```
all:
  hosts:
    k8s-control:
      ansible_host: 10.0.0.10
      ip: 10.0.0.10
      access_ip: 10.0.0.10
    k8s-worker1:
      ansible_host: 10.0.0.11
      ip: 10.0.0.11
      access_ip: 10.0.0.11
  children:
    kube_control_plane:
      hosts:
        k8s-control:
    kube_node:
      hosts:
        k8s-control:
```

```
        k8s-worker1:
    etcd:
      hosts:
        k8s-control:
    k8s_cluster:
      children:
        kube_control_plane:
        kube_node:
```

4. Запустіть встановлення кластера за допомогою Ansible Playbook:

```
ansible-playbook -i inventory/mycluster/hosts.yaml --become --become-user=root cluster.yml
```

7.4. Крок 4. Створення та обслуговування Helm-репозиторію

Для збереження автономності розгортаємо мінімалістичний Helm-сервер на базі Nginx на інфраструктурному хості (10.0.0.5):

1. Встановіть Nginx та створіть каталог для діаграм:

```
sudo apt install -y nginx
sudo mkdir -p /var/www/helm
```

2. Налаштуйте віртуальний хост Nginx для домену charts.erp.uno та перезапустіть службу.

3. Увійдіть до каталогу з вашим Helm-чартом додатку ERP/1 та упакуйте його:

```
helm package ./erp-backend --destination /var/www/helm/
```

4. Згенеруйте індекс репозиторію:

```
helm repo index /var/www/helm/ --url http://charts.erp.uno/
```

7.5. Крок 5. Налаштування власного Docker Registry

Для збереження образів контейнерів на інфраструктурному хості (10.0.0.5) розгортається локальний реєстр образів registry.erp.uno.

1. Встановіть інструментарій docker.io та запустіть сервіс реєстру:

```
sudo apt update
sudo apt install -y docker.io
sudo docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

2. Налаштуйте Nginx як реверс-проксі для домену registry.erp.uno з SSL-сертифікатами, згенерованими локальним ca-pki:

```
server {
    listen 443 ssl;
    server_name registry.erp.uno;

    ssl_certificate /etc/nginx/certs/registry.crt;
    ssl_certificate_key /etc/nginx/certs/registry.key;

    location / {
```

```
        proxy_pass http://localhost:5000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

3. Зберіть локальний образ додатку на базі Alpine Linux, додайте тег та відправте його до власного реєстру:

```
docker build -t registry.erp.uno/erpuno/crm-documents:1.0.0 .
docker push registry.erp.uno/erpuno/crm-documents:1.0.0
```

7.6. Крок 5a. Збірка образів додатків на базі Alpine Linux (Dockerfile)

Для забезпечення стабільної роботи BEAM-релізів у мінімалістичному оточенні Alpine Linux використовується такий багатоетапний (multi-stage) `Dockerfile`:

```
FROM alpine:3.20 AS builder

RUN apk add --no-cache erlang-dev elixir git build-base

WORKDIR /opt/app

RUN mix local.hex --force && \
    mix local.rebar --force

ENV MIX_ENV=prod

COPY mix.exs mix.lock ./
COPY config ./config
RUN mix deps.get --only prod && mix deps.compile

COPY lib ./lib
RUN mix release

FROM alpine:3.20

RUN apk add --no-cache openssl ncurses-libs libstdc++

WORKDIR /opt/app

COPY --from=builder /opt/app/_build/prod/rel/erp_system ./

ENV PORT=8080

CMD ["./bin/erp_system", "start"]
```

7.7. Крок 6. Налаштування GitOps доставки через ArgoCD

1. Встановіть клієнтський інструментарій `kubectl` на керуючій ноді та підключіть кластер:

```
mkdir -p ~/.kube
sudo cp /etc/kubernetes/admin.conf ~/.kube/config
sudo chown $(id -u):$(id -g) ~/.kube/config
```

2. Встановіть ArgoCD:

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manife
```

3. Створіть файл конфігурації додатку GitOps `erp-app.yaml` для відстеження інфраструктурного репозиторію `helm/`:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: erp-uno
  namespace: argocd
spec:
  project: default
  source:
    repoURL: 'http://git.erp.uno/erpuno/erp.uno.git'
    path: helm
    targetRevision: HEAD
    helm:
      valueFiles:
        - values.yaml
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: erp
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    createNamespace: true
```

4. Налаштуйте Ingress-маршрутизацію для доступу до панелі управління ArgoCD за адресою `argocd.erp.uno` (файл `argocd-ingress.yaml`):

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: argocd-server-ingress
  namespace: argocd
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  ingressClassName: nginx
  rules:
  - host: argocd.erp.uno
```

```
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: argocd-server
            port:
              number: 443
  tls:
  - hosts:
    - argocd.erp.uno
    secretName: argocd-server-tls
```

5. Розгорніть додаток та Ingress в ArgoCD:

```
kubectl apply -f erp-app.yaml
kubectl apply -f argocd-ingress.yaml
```

ArgoCD автоматично виявить зміни в Git-репозиторії, оновить ресурси у кластері та розгорне платформу «ERP/1» у просторі імен `erp`.

8. Тестові питання та завдання

1. Які державні інформаційні системи України використовують компоненти ERP/1?
2. Опишіть механізм побудови Erlang-кластера всередині Kubernetes без фіксованих IP-адрес нод.
3. Які переваги дає використання легковагого DNS-сервера `synrc/ns` у порівнянні з класичним BIND/CoreDNS в закритих контурах?
4. Які переваги та структуру мають Helm-діаграми для прикладних продуктів (IKC) та інфраструктурних служб (SLUB) платформи ERP/1?
5. Опишіть модель асинхронного ETL-процесингу черг з використанням курсорів та контекстів на базі сховища KVS.
6. Які ризики несе явище Split-Brain для розподіленої СУБД Mnesia та як їх мінімізувати?
7. **Практичне завдання**: Напишіть конфігураційний файл зони `synrc/ns` у нативному форматі Erlang Terms для обслуговування трьох нод бази даних Mnesia та однієї ноди ArgoCD.
8. **Практичне завдання**: Напишіть на Erlang функцію ініціалізації контексту воркера `#etl_context{}` з первинним зчитуванням курсора з KVS.