ERP/1: Підприємство

CD/SRE Handbook

Настанови системного адміністратора для розгортання, спостереження і обслуговування кластерів ЦОД

Версія системи:	2026.7.10
Контекст:	kind (arm64)
Кластер:	synrc-control-plane (arm64)
Домен:	1 (erp.uno)
Неймспейси:	7 просторів (argocd, infra, telemetry, security, services, apps, ai)
Компоненти:	66 сервісів та 22 ІКС

Зміст

Зміст

1 Глосарій та умовні позначення **3**

2 Архітектура платформи ERP/1 **4**
2.1 Загальний огляд . . . 4
2.2 Архетипи розгортання та керування даними . . . 5
2.3 Простори та їх призначення . . . 6
2.4 Структура кластеру . . . 7

3 Структура CD репозиторію **8**
3.1 Огляд директорій . . . 8

4 Каталог компонентів ERP/1 **9**
4.1 Шаблон компонента . . . 9
4.2 Зведена таблиця маніфестів . . . 10
4.3 Параметри `deployment.yaml` . . . 11
4.4 Параметри `service.yaml` . . . 11
4.5 Параметри `hpa.yaml` . . . 12
4.6 Параметри `pvc.yaml` . . . 12

5 Синхронізація Helm-чарту **13**
5.1 Архітектура Helm-чарту . . . 13
5.2 Структура `helm/values.yaml` . . . 13
5.3 Механізм синхронізації `values.rb` . . . 14
5.4 Helm-шаблони . . . 14

6 Безпека та мережева ізоляція **15**
6.1 RBAC — рольова модель . . . 15
6.2 NetworkPolicy — Deny First . . . 15
6.3 Політика базових образів контейнерів (UA/Alpine) . . . 16

7 SRE: SLI, SLO, SLA та Error Budgets **17**
7.1 SLI (Service Level Indicator) — індикатори рівня сервісу . . . 17
7.2 SLO (Service Level Objective) — цілі рівня сервісу . . . 18
7.3 SLA (Service Level Agreement) — угоди про рівень сервісу . . . 18
7.4 Принцип Error Budget . . . 19

8 Архітектура спостереження **20**
8.1 Компоненти стеку . . . 20

9 Розгортання кластеру **21**
9.1 Послідовність розгортання неймспейсів . . . 22
9.2 GitOps-рекомендації для production . . . 22
9.3 Співвідношення Git-репозиторіїв та компонентів GitOps . . . 22
9.4 Формалізована послідовність «холодного» запуску (Cold Boot Sequence) . . . 23
9.5 Пререквізити prereq.sh . . . 25
9.6 Впровадження deploy.sh . . . 26
9.7 Генерація Helm моно-чарту values.rb . . . 28
9.8 Пакетування helm/deploy . . . 29
9.9 Деконструкція кластеру . . . 31

10 Локальний контур GitOps: Gitea та ArgoCD **32**
10.1 Концепція локального GitOps та ізоляція залежностей . . . 32
10.2 Локальний Git-сервер Gitea . . . 32
10.3 Розгортання та інтеграція ArgoCD . . . 33

11 Механіка оркестрації та життєвий цикл розгортання **36**
11.1 Формальний контур ініціалізації: від DNS та хостів до реєстру . . . 36
11.2 Кубернетес за лаштунками (Behind the Scenes) . . . 36
11.3 Генерація values.yaml за допомогою values.rb: зворотний GitOps . . . 37

12 Incident Management / On-Call **38**
12.1 Severity-рівні . . . 38
12.2 Playbooks . . . 38
12.3 Аналіз реальних відмов та постмортеми . . . 39

13 Патерни відмовостійкості (Resilience Patterns) **40**
13.1 Основні архітектурні патерни . . . 40

14 Chaos Engineering та Resilience Testing **41**
14.1 Стратегія ін'єкції відмов . . . 41
14.2 RTO / RPO цілі . . . 41

15 Інженерна культура та метрики доставки (DORA) **42**
15.1 DORA Метрики . . . 42
15.2 Pipeline-Driven Organization та Культура . . . 42

16 Операційні метрики та ресурсний бюджет **43**
16.1 Ключові SRE-метрики . . . 43
16.2 Ресурсне навантаження кластеру . . . 43

17 Roadmap надійності ERP/1 **44**

18 Висновок **45**

Анотація

Цей **CD/SRE Handbook** є офіційним технічним довідником з Continuous Deployment та Site Reliability Engineering для платформи **ERP/1: Підприємство** компанії "Критпографічні Телесистеми". Він охоплює повну архітектуру всіх просторів Kubernetes кластеру, деталізовану структуру Helm чартів, специфікацію всіх маніфестів компонентів, а також практики SRE: SLI/SLO, бюджети на помилки, чергування і операційні вікна, інженерія хаосу і GitOps/ArgoCD. Документ адаптує класичні принципи Site Reliability Engineering (Google SRE Book) до реалій державних ERP систем із вимогами КСЗІ, мультитенантності та неперервного впровадження.

1. Глосарій та умовні позначення

Термін	Визначення
SRE	Site Reliability Engineering — дисципліна, що поєднує розробку ПЗ із системним адмініструванням для досягнення надійних, масштабованих сервісів
SLO	Service Level Objective — цільовий показник рівня сервісу (напр. 99.9% availability за 30 днів)
SLI	Service Level Indicator — фактичний вимірюваний індикатор (напр. відсоток успішних HTTP-запитів)
SLA	Service Level Agreement — зовнішній договір з клієнтом щодо рівня сервісу
TOIL	Ручна операційна робота без довгострокової цінності; мета SRE — мінімізація TOIL <50%
Error Budget	Дозволена кількість «збоїв»: Error Budget $= 1 -$ SLO. Витрачений бюджет блокує нові релізи
MTTR	Mean Time To Recovery — середній час відновлення після інциденту
MTTD	Mean Time To Detect — середній час виявлення інциденту
RTO	Recovery Time Objective — цільовий час відновлення
RPO	Recovery Point Objective — допустима точка відновлення даних
K8s	Kubernetes — оркестратор контейнерів
Helm	Package manager для Kubernetes; чарт = пакет маніфестів + values
GitOps	Декларативний підхід: Git є єдиним джерелом істини для інфраструктури
HPA	HorizontalPodAutoscaler — автомасштабування подів за CPU/Memory
PVC	PersistentVolumeClaim — запит на постійне сховище
RBAC	Role-Based Access Control — керування доступом на основі ролей
StatefulSet	Kubernetes workload для stateful-сервісів (Prometheus, docker-registry)
NetworkPolicy	Kubernetes API для мережевої ізоляції між namespace'ами
ERP/1	Електронна система планування ресурсів підприємства версії 1 (erpuno)
КСЗІ	Комплексна система захисту інформації — державний стандарт

2. Архітектура платформи ERP/1

2.1. Загальний огляд

ERP/1 розгорнуто на **Kubernetes-кластері** (kind: desktop-control-plane, arch: arm64, 10 CPU, 8 ГБ RAM) під керуванням **Docker Desktop**. Уся конфігурація зберігається декларативно у репозиторії `erpuno/cd` та синхронізується через Helm-чарти і shell-скрипти.

Платформа складається з **7 просторів** та **88 сервісів**, організованих за принципом **Separation of Concerns**:

Порядок розгортання залежностей:

`infra` → `telemetry` → `security` → `ai` → `services` → `apps` → `argocd`

Нижні рівні не залежать від верхніх. Порядок відображає топологічне сортування залежностей між сервісами.

Мікросервіси. Термін "мікросервіси" (після того, як Director of Operations Engineering, який пофіксав Netflix, виступив з доповіддю на QCon в Сан Франціско "Mastering Chaos - A Netflix Guide to Microservices"[a]) отримав негативний відтінок, оскільки для контрольованого масштабування доводиться на практиці масштабувати моноліти (приклад ECO3). Тому в нашому документі ми будемо називати "мікросервісами" поди, що будуються на основі образів Alpine Linux (мають навіть вищі показники по безпеці ніж UA Linux із-за своєї мінімалістичності), а просто "сервісами" образи, що будуються на образах Linux з власними initd інфрастктурами, сервісами і розширеними пакетними менеджерами, такі як Ubuntu, UA Linux, тощо.

[a] `https://www.youtube.com/watch?v=CZ3wIuvmHeM`

2.2. Архетипи розгортання та керування даними

Вибір архітектури розгортання критично впливає на доступність (High Availability), затримку (Latency) та вартість інфраструктури. Відповідно до наукової класифікації Anna Berenberg та Brad Calder [1], архітектура ERP/1 проєктується з урахуванням **Zonal** / **Regional** архетипів:

- Всі компоненти розгортаються з огляду на *Fault Domains* (домени відмови) — логічні та фізичні межі, які ізолюють каскадні збої (Single Point of Failure).
- Для досягнення високої доступності застосовується шардування (Sharding) навантаження та інкрементальне оновлення сервісів (Incremental Rollout) з можливістю швидкого відкату.

Робота з даними (наприклад, `kvs-storage`, `bpe-processes`) є найскладнішим аспектом надійності і базується на трьох стовпах:

- **Data Durability**: Гарантія того, що дані не будуть пошкоджені або втрачені (забезпечується надійними PVC та механізмами реплікації).
- **Data Availability**: Забезпечення безперервного доступу до даних навіть при часткових збоях вузлів кластера (реалізується через StatefulSets та eventual/strong consistency).
- **Data Backup**: Регулярні знімки стану (snapshots) для захисту від людських помилок та пошкоджень на рівні програмного забезпечення (Disaster Recovery).

Ефективність Disaster Recovery жорстко контролюється двома цільовими метриками:

- **RPO (Recovery Point Objective)**: Максимально допустимий обсяг втрати даних у часовому вимірі (точці відновлення).
- **RTO (Recovery Time Objective)**: Максимально допустимий час, необхідний на відновлення повноцінної працездатності сервісу.

[1] `https://arxiv.org/abs/2105.00560` — Deployment Archetypes for Cloud Applications

2.3. Простори та їх призначення

Namespace	Tier	Призначення та сервіси
`erp-infra`	infra	Базова інфраструктура: `ns-dns` (DNS-резолвер, :8080), `docker-registry` (реєстр образів, :5000, PVC 10 Gi)
`erp-telemetry`	observability	Observability stack: `prometheus` (метрики, :9090, StatefulSet, PVC 10 Gi), `grafana` (дашборди, :3000, PVC 10 Gi), `otel-collector` (трейси, :8080)
`erp-security`	security	Безпека та PKI: `ca-pki` (центр сертифікації, :8080)
`erp-ai`	application	Штучний інтелект: `ai-generation` (генерація контенту, :8080)
`erp-services`	application	Ядро бізнес-логіки: `rest-bpe` (REST API, :8080), `bpe-process` (рушій бізнес-процесів), `n2o-connections` (No-code/Low-code), `kvs-storage` (KV-сховище), `chat-messenger` (месенджер)
`erp-apps`	application	Бізнес-додатки: `hl7-health` (HL7, :8502), `lms-education` (:8506), `wms-warehouse`, `acc-accounting`, `crm-documents`, `cart-registers`, `pm-projects`, `itsm-incidents`

2.4. Структура кластеру

Snapshot стану подів (04 липня 2026, uptime 99.97%, `kubectl get all -A`):

Namespace	Под	Ready	Статус	Uptime
erp-infra	ns-dns-ddbb77c5b-8s	1/1	Running	14d 6h
erp-infra	docker-registry-6f9c87cd	1/1	Running	14d 6h
erp-telemetry	prometheus-0	1/1	Running	14d 6h
erp-telemetry	grafana-68f7cf4799	1/1	Running	14d 6h
erp-telemetry	otel-collector-b5f2c	1/1	Running	14d 6h
erp-security	ca-pki-d598976d4-gx7fp	1/1	Running	14d 6h
erp-ai	ai-generation-55fffb8f64	1/1	Running	7d 3h
erp-services	rest-bpe-884475778-whqts	1/1	Running	14d 6h
erp-services	bpe-processes-9f4d2a-xk	1/1	Running	14d 6h
erp-services	kvs-storage-c8b3e1-pq	1/1	Running	14d 6h
erp-apps	hl7-health-6b9744588d-1	2/2	Running	14d 6h
erp-apps	lms-education-7c6f69c6d6-1	2/2	Running	14d 6h
erp-apps	wms-warehouse-77f55677f6	1/1	Running	7d 3h
erp-apps	acc-accounting-3a1b4c-rv	1/1	Running	7d 3h

Усі поди Running. Кластер ERP/1 працює у штатному режимі. Образи публікуються у приватному `docker-registry` (erp-infra) та кешуються локально (`imagePullPolicy: IfNotPresent`). Середній uptime системи — **14 днів** без неплановних перезапусків. Моніторинг доступний через Grafana на :3000.

3. Структура CD репозиторію

Як складна система, кластер ERP/1 управляється системою скриптів, яка була побудована природнім чином без використання сторонніх інструментів, окрім `kind`, `kubectl`, `helm`, `docker`, `argocd` і, звичайно `docker-desktop`. Цей набір bash скриптів можна було би оформити у вигляді Go command line utility зі своєю системою команд, можливо в наступних версіях `erpuno/cd` так і буде зроблено.

3.1. Огляд директорій

Шлях	Призначення
`argocd/`	Docker Compose леєр для Gitea (поза кластером): `install.yaml`, `uno.erp.argocd-portforward.plist`, `ingress.yaml`, `application.yaml`, `argocd-portforward.sh`
`compose/`	Docker Compose леєр для Gitea (поза кластером): `gitea.yaml`
`helm/`	Helm-чарт верхнього рівня: `Chart.yaml`, `values.yaml`, `deploy.sh`, `patch.sh`, `validate.sh`
`helm/templates/`	Шаблони для Helm чарту: `deployments.yaml`, `services.yaml`, `hpa.yaml`, `namespaces.yaml`
`lib/`	Сирі маніфести для служб та IKC: `lib/{namespace}/{service}/`
`lib/erp-*/`	По одній директорії на кожен з 6 просторів
`share/`	Кластерні ресурси: `namespaces.yaml`, `rbac.yaml`, `networkpolicy.yaml`, `storage-class.yaml`
`argocd.sh`	bash-скрипт розрогратння простору ArgoCD і його синхронізація з внутрішнім Gitea репозиторієм, що містить стан кластеру
`backup.sh`	bash-скрипт резервного копіювання PVC леєрів
`delete-pvc.sh`	Імітація втрати PVC ресурсів
`delete.sh`	Видалення всіх ERP-ресурсів з кластеру
`deploy.sh`	bash-скрипт функціонального розгортання кластера
`gitops.sh`	bash-скрипт розгортання Gitea і зовнішній міроринг
`images.sh`	bash-скрипт відображення Docker контейнерів зі службами та IKC
`kind.sh`	bash-скрипт управління K8S контекстами Kind
`liveness.sh`	bash-скрипт статусу ArgoCD простору
`manifest.rb`	bash-скрипт для `backup.sh`, що фільтрує маніфести
`monitor.sh`	termios bash-скрипт для безперервного одноекранного термінального моніторингу
`prereq.sh`	Перевірка та встановлення передумов (kind, kubectl, helm)
`rebuild.sh`	Збірка Docker контейнерів з Gitea і публікації в реєстрі контейнерів
`restore.sh`	bash-скрипт відновленя резервних копій PVC леєрів
`values.rb`	Ruby-скрипт генерує `helm/values.yaml` з `lib/`
`view.sh`	Переглядач та екстрактор файлів з резервних копій

4. Каталог компонентів ERP/1

4.1. Шаблон компонента

Кожен мікросервіс у `lib/{namespace}/{service}/` може містити:

Файл	Зміст та ключові параметри
`deployment.yaml`	Kubernetes `Deployment` або `StatefulSet`. Поля: `apiVersion: apps/v1`, `spec.replicas`, `containers[].image`, `resources.requests/limits`, `ports[].containerPort`, `livenessProbe`, `readinessProbe`, `serviceAccountName`, `affinity.podAntiAffinity`
`service.yaml`	Kubernetes `Service` (тип `ClusterIP`). Для StatefulSet — `clusterIP: None` (headless). Поля: `spec.ports[].port`, `.targetPort`, `.protocol`, `spec.selector.app`
`hpa.yaml`	`HorizontalPodAutoscaler` (`autoscaling/v2`). Поля: `spec.minReplicas`, `.maxReplicas`, `.metrics[].resource.name` (cpu/memory), `.target.averageUtilization`. Лише для сервісів з `hpa.enabled: true`
`pvc.yaml`	`PersistentVolumeClaim` для stateful-сервісів. Поля: `spec.accessModes: ReadWriteOnce`, `.storageClassName: local-path`, `.resources.requests.storage: 10Gi`
`values.yaml`	Локальні значення для конкретного сервісу. Читається `values.rb` при генерації `helm/values.yaml`. Дублює ключові параметри `deployment.yaml` у форматі Helm
`ingress.yaml`	`Ingress` Описа інгрес контролера. Поля: `spec.rules[].http.path[].backend.service`, `.ingressClassName: local-path`
`Dockerfile`	Файл для збірки контейнера служби чи ІКС з Gitea
`build.sh`	Файл для збірки імаджа або для `kind load docker-image` або для `docker push` (використовується `-registry` опція).

4.2. Зведена таблиця маніфестів

Легенда: D — deployment, S — service, P — pvc, V — values, H — hpa.

Сервіс	Namespace	Port	CPU req/lim	Легенда
Рівень інфраструктури				
docker-registry	erp-infra	5000	50m/200m	PDSV
traefik-ingress	erp-infra	5000	50m/200m	PDSV
ns-dns	erp-infra	8080	100m/500m	DSV
Рівень спостереження				
prometheus	erp-telemetry	9090	100m/500m	PDSV
grafana	erp-telemetry	3000	50m/200m	PDSV
otel-collector	erp-telemetry	8404	100m/500m	DSV
Рівень безпеки				
ca-pki	erp-security	8201	100m/500m	DSV
abac-clearance	erp-security	8201	100m/500m	DSV
ldap-directory	erp-security	8201	100m/500m	DSV
vpn-wireguard	erp-security	8201	100m/500m	DSV
ias-auth	erp-security	8201	100m/500m	DSV
chat-messenger	erp-security	8507	100m/500m	DSV
Рівень моделей				
ai-generation	erp-ai	8601	100m/500m	DSV
faiss-search	erp-ai	8601	100m/500m	DSV
Рівень служб і сервісів				
kvs-storage	erp-services	8301	100m/500m	DSV
bpe-processes	erp-services	8302	100m/500m	DSV
rest-api	erp-services	8303	100m/500m	DSV
mach-ivr	erp-services	8303	100m/500m	DSV
n2o-connections	erp-services	8511	100m/500m	DSV
Рівень ІКС				
hl7-health	erp-apps	8502	100m/500m	HDSV
lms-education	erp-apps	8506	100m/500m	HDSV
wms-warehouse	erp-apps	8505	100m/500m	DSV
acc-accounting	erp-apps	8504	100m/500m	DSV
crm-documents	erp-apps	8503	100m/500m	DSV
cart-registers	erp-apps	8501	100m/500m	DSV
pm-projects	erp-apps	8514	100m/500m	DSV
plm-products	erp-apps	8514	100m/500m	DSV
scm-supply	erp-apps	8514	100m/500m	DSV
olap-analytics	erp-apps	8514	100m/500m	DSV
itsm-incidents	erp-apps	8513	100m/500m	DSV

4.3. Параметри deployment.yaml

Поле YAML	Тип / Замовч.	Призначення
apiVersion	apps/v1	API-версія Kubernetes
kind	Deployment / StatefulSet	StatefulSet для сервісів із persistence
metadata.name	string	Ім'я workload = назва сервісу
metadata.namespace	erp-*	Цільовий namespace; підставляється через sed у deploy.sh
spec.replicas	1 або 2	HL7-health і LMS-education — 2 для High Availability
containers[].image	registry/n:tag	erpuno/{service}:2024.6.15
containers[].ports[] .containerPort	int	Порт, на якому прослуховує застосунок
resources.requests.cpu	50m–100m	Гарантований CPU (milliCPU)
resources.limits.cpu	200m–500m	Максимальний CPU
resources.requests.memor	128–256Mi	Гарантована пам'ять
resources.limits.memory	512Mi–1Gi	Максимальна пам'ять
livenessProbe	httpGet /health	Перезапускає под якщо застосунок «завис» (failureThreshold: 3)
readinessProbe	httpGet /health	Виключає под з балансування під час старту (failureThreshold: 2)
serviceAccountName	{ns}-sa	RBAC-ідентифікатор пода
affinity.podAntiAffinity	preferred	Розподіл реплік по різних вузлах
volumeMounts / PVC	/data	Тільки для stateful-сервісів

4.4. Параметри service.yaml

Поле YAML	Значення	Призначення
kind: Service	v1	ClusterIP-сервіс (внутрішній)
spec.type	ClusterIP	Доступний тільки всередині кластеру
spec.clusterIP: None	headless	StatefulSet: DNS-рекорд на кожен под окремо
spec.ports[].port	int	Зовнішній порт сервісу
spec.ports[].targetPort	int	Порт пода
spec.ports[].protocol	TCP	Завжди TCP
spec.selector.app	string	Мітка для вибору подів-бекендів

4.5. Параметри hpa.yaml

Поле YAML	Значення	Призначення
apiVersion	autoscaling/v2	Підтримує кілька метрик (CPU + Memory)
spec.scaleTargetRef	Deployment	Об'єкт масштабування
spec.minReplicas	2	Мінімум реплік (HL7-health, LMS-education)
spec.maxReplicas	6	Максимум реплік при навантаженні
metrics: cpu	75%	Цільове використання CPU
metrics: memory	80%	Цільове використання пам'яті

4.6. Параметри pvc.yaml

Поле YAML	Значення	Призначення
kind: PersistentVolumeClaim	v1	Запит на постійне сховище
spec.accessModes	ReadWriteOnce	Один вузол — запис
spec.storageClassName	local-path	Provisioner для kind/Docker Desktop
spec.resources.requests.	10Gi	Розмір тому

5. Синхронізація Helm-чарту

5.1. Архітектура Helm-чарту

Helm-чарт `erp-uno` (version 2024.6.15) є єдиним umbrella-чартом, що покриває всі namespace'и та сервіси:

```
helm upgrade --install erp-uno ./helm \
  --values "./helm/values.yaml" \
  --set global.domain=erp.uno \
  --set global.environment=production \
  --server-side=true \
  --force-conflicts
```

5.2. Структура `helm/values.yaml`

Секція	Зміст
`global:`	`domain: erp.uno`, `environment: production`, `registry: docker.io`, `imagePullPolicy: IfNotPresent`, `version: 2024.6.15`
`namespaces:`	6 namespace'ів з `enabled: true` та `tier:` (infrastructure / observability / security / application). Вимикання namespace = вимикання цілого рівня
`services:`	Дворівнева карта `services.{ns}.{service}`. Кожен запис: `enabled`, `replicas`, `resources`, `port`, `image`, `protocol`, `persistence`, `hpa`
`rbac:`	`create: true` — дозволяє Helm-шаблонам генерувати RBAC
`networkPolicy:`	`enabled: true` — активує NetworkPolicy між namespace'ами
`ingress:`	`className: nginx`, hosts, TLS, маршрут до `nitro-portal:8510`

5.3. Механізм синхронізації `values.rb`

Принцип роботи:

Запускається сценарій зворотної генерації для збору конфігурацій окремих мікро- сервісів та автоматичного формування зведеного файлу параметрів Helm. Каталог `lib/` — джерело істини. Будь-яка зміна в `lib/` вимагає перегенерації: `ruby values.rb -force`, що генерує `helm/values.yaml`, який, в свою чергу, є джерелом істини для ArgoCD.

```
ruby values.rb --force          # overwrite without confirmation
ruby values.rb --dry-run        # preview only, no file write
ruby values.rb --version 2024.7 # update all image versions
```

Алгоритм:

1. `Dir.glob("lib/erp-*/*/")` – iterate components
2. `YAML.safe_load deployment.yaml` – parse manifest
3. Extract: `replicas, image, port, resources, persistence, hpa`
4. Write into `services[namespace][service]`
5. Serialize to `helm/values.yaml`

5.4. Helm-шаблони

Шаблон	Що генерує
`deployments.yaml`	Ітерується по `.Values.services`. Якщо `persistence != nil` — StatefulSet, інакше Deployment. Автоматично формує image, ports, resources, volumeClaimTemplates, serviceAccountName
`services.yaml`	ClusterIP Service для кожного увімкненого сервісу. Headless (`clusterIP: None`) якщо `persistence != nil`
`hpa.yaml`	HorizontalPodAutoscaler тільки якщо `hpa.enabled: true`. Метрики: CPU (75%) та Memory (80%) через `autoscaling/v2`
`namespaces.yaml`	Namespace-об'єкти з мітками для NetworkPolicy

6. Безпека та мережева ізоляція

6.1. RBAC — рольова модель

Ресурс	Деталі
`ServiceAccount: {ns}-sa`	Ідентифікатор пода у Kubernetes API. Шість SA: `erp-infra-sa`, `erp-telemetry-sa`, `erp-security-sa`, `erp-ai-sa`, `erp-apps-sa`, `erp-services-sa`
`Role: {ns}-role`	Мінімально достатні права: `get`, `list`, `watch` на `pods`, `endpoints`, `configmaps`, `secrets`. Жодних привілеїв на запис
`RoleBinding: {ns}-rolebinding`	Прив'язує Role до ServiceAccount у відповідному namespace

6.2. NetworkPolicy — Deny First

Двошаровий підхід у `share/networkpolicy.yaml`:

1. **erp-deny-all**: заборона всього Ingress/Egress у кожному namespace
2. **erp-allow-internal**: дозвіл явно необхідних з'єднань

Namespace	Дозволений Egress	Дозволений Ingress
erp-infra	Всі ERP-ns, DNS (:53), HTTPS (:443)	erp-telemetry, erp-security, erp-ai, erp-apps
erp-telemetry	Всі ERP-ns, DNS, HTTPS	erp-infra, erp-security, erp-ai, erp-apps
erp-security	Всі ERP-ns, DNS, HTTPS	erp-infra, erp-telemetry, erp-ai, erp-apps
erp-ai	erp-infra (:5000, :8301), erp-telemetry (:9090), erp-security (:8204), DNS, HTTPS	Тільки власний ns
erp-apps	erp-infra (:5000, :8301), erp-telemetry (:9090), erp-security (:8204), DNS, HTTPS	Тільки власний ns
erp-services	erp-infra (:5000, :8301), erp-telemetry (:9090), erp-security (:8204), DNS, HTTPS	Тільки власний ns

6.3. Політика базових образів контейнерів (UA/Alpine)

З метою мінімізації вектора атак, економії дискового простору та підвищення швидкості холодного старту подів у Kubernetes, діє сувора політика використання легковагого базового образу **Alpine Linux** (версії `alpine:3.20`) для побудови контейнерів BEAM-додатків платформи:

- **Мінімальний обсяг образів**: Використання Alpine Linux замість класичних дистрибутивів (наприклад, Ubuntu/Debian) дозволяє зменшити розмір базового шару образу з ∼80–100 МБ до ∼5–8 МБ. Це суттєво зменшує навантаження на мережу при авто-масштабуванні (HPA) подів.
- **Зменшення поверхні атаки (Security hardening)**: Базовий образ містить лише мінімальний набір системних утиліт і бібліотек. У фінальних продуктових контейнерах заборонено залишати компілятори, інструменти збірки чи пакетні менеджери (такі як `apk`).
- **Статична лінковка та C-бібліотеки**: Усі скомпільовані C/Rust модулі (NIF-бібліотеки), які завантажуються віртуальною машиною Erlang, лінкуються з системною бібліотекою `musl` C, що є стандартом в Alpine Linux.

7. SRE: SLI, SLO, SLA та Error Budgets

Концепції забезпечення надійності (Site Reliability Engineering), впроваджені в ERP/1, ґрунтуються на фундаментальних індустріальних стандартах, передусім на методології **Google SRE Book**.

7.1. SLI (Service Level Indicator) — індикатори рівня сервісу

SLI (Service Level Indicator) — це ретельно визначена кількісна міра певного аспекту рівня наданого сервісу (наприклад, затримка запиту, частота помилок, пропускна здатність системи, доступність).

Для вибору оптимальних метрик SLI у мікросервісах ERP/1 застосовується **RED Method** (Rate, Errors, Duration), запропонований Tom Wilkie. Він фокусується на вимірюванні того, що безпосередньо впливає на користувацький досвід:

SLI	Тип	PromQL-вираз
Availability	Rate of Success	`sum(rate(http_requests_-total{status!~"5.."}[5m])) / sum(rate(http_requests_total[5m]))`
Latency p95	Duration	`histogram_quantile(0.95, sum(rate(http_request_duration_-seconds_bucket[5m])) by (le))`
Error Rate	Errors	`sum(rate(http_requests_-total{status~"5.."}[5m]))`
Saturation	Resource	`container_memory_usage_bytes / container_spec_memory_limit_bytes`
Pod Readiness	Availability	`kube_pod_status_-ready{condition="true"} == 1`

7.2. SLO (Service Level Objective) — цілі рівня сервісу

SLO (Service Level Objective) — це цільове значення або діапазон значень для рівня сервісу, який вимірюється за допомогою SLI. Природна структура для SLO — це SLI $\leq$ target або lower bound $\leq$ SLI $\leq$ upper bound. Публікація SLO для користувачів формує очікування щодо того, як працюватиме сервіс.

Нижче наведено розподіл цілей доступності для основних компонентів платформи:

Namespace	SLO	Бюджет (30 д.)	Обґрунтування
erp-telemetry	99.99%	4.3 хв	Без observability сліпий моніторинг платформи
erp-security	99.99%	4.3 хв	Відмова PKI/Auth блокує всі сервіси
erp-infra	99.95%	21.6 хв	Базова інфраструктура; можливі планові вікна
erp-services	99.95%	21.6 хв	Ядро бізнес-процесів; критичне
erp-apps	99.90%	43.2 хв	Бізнес-додатки з менш жорсткими SLA
erp-ai	99.50%	3.6 год	Допоміжний сервіс; деградація не блокує роботу

7.3. SLA (Service Level Agreement) — угоди про рівень сервісу

SLA (Service Level Agreement) — це явний або неявний контракт з вашими користувачами, який включає наслідки досягнення (або недосягнення) встановлених SLO. Основна відмінність від SLO полягає в наявності фінансових, юридичних або репутаційних наслідків: «що станеться, якщо SLO не буде досягнуто?».

В архітектурі ERP/1 SLA визначаються на рівні всієї платформи для кінцевих користувачів (наприклад, доступність модулів `erp-apps`), тоді як SLO використовуються внутрішніми командами розробки для контролю надійності окремих мікросервісів та їх залежностей.

7.4. Принцип Error Budget

Бюджет помилок (Error Budget) — це механізм контролю балансу між швидкістю інновацій та надійністю сервісу, що визначає допустимий час простою.

Error Budget $= 1 -$ SLO

При SLO 99.9% за місяць — допустимий downtime $= \mathbf{43.2}$ хв.
При SLO 99.95% — $\mathbf{21.6}$ хв.
При SLO 99.99% — $\mathbf{4.3}$ хв.

Якщо Error Budget вичерпано — всі нові релізи заморожуються до відновлення.

8. Архітектура спостереження

8.1. Компоненти стеку

Сервіс	Порт	Роль у стеку
`prometheus`	:9090	Збір і зберігання метрик (StatefulSet, PVC 10 Gi). SLI-обчислення та алертинг через Alertmanager. Health: `/-/healthy`, `/-/ready`
`grafana`	:3000	Дашборди та візуалізація (PVC 10 Gi). Підключення до Prometheus. Доступ: `kubectl port-forward -n erp-telemetry svc/grafana 3000:3000`
`otel-collector`	:8080	OpenTelemetry Collector — збір трейсів, пересилання до Jaeger/Tempo

```
# Access Prometheus UI
kubectl port-forward -n erp-telemetry svc/prometheus 9090:9090
# Access Grafana UI (admin/admin)
kubectl port-forward -n erp-telemetry svc/grafana 3000:3000
# Check stack status
kubectl get pods -n erp-telemetry
kubectl get pvc  -n erp-telemetry
```

9. Розгортання кластеру

Процедура розгортання та життєвого циклу оркестрації платформи ERP/1 є повністю детермінованим, автоматизованим та декларативно описаним процесом. Вона спроектована відповідно до методології Site Reliability Engineering (SRE) та архітектурних паттернів GitOps. Забезпечення високого ступеня повторюваності та зближення середовищ розробки й промислової експлуатації досягається за допомогою гібридної системи автоматизації. Ця система поєднує інструменти прямої імперативної ініціалізації (bash-сценарії, Ruby-генератори конфігурацій, Helm-пакетування) та декларативний GitOps-контролер безперервної синхронізації (ArgoCD), який використовує локальний сервер Git (Gitea) як внутрішнє джерело істини (Single Source of Truth).

Контур автоматизації розгортання включає такі фундаментальні компоненти:

- **Скрипт валідації пререквізитів (`prereq.sh`)**: Здійснює статичний та динамічний аналіз операційного середовища на відповідність вимогам (версії `kubectl`, `helm`, `kind`, стан вузлів, наявність StorageClass та конфігурація DNS).
- **Ruby-генератор конфігурацій (`values.rb`)**: Реалізує концепцію зворотного GitOps, парсячи сирі Kubernetes-маніфести мікросервісів та агрегуючи їх у єдиний структурований файл параметрів Helm (`helm/values.yaml`).
- **Скрипт топологічної оркестрації (`deploy.sh`)**: Керує послідовним розгортанням шарів інфраструктури та бізнес-додатків з урахуванням графів їхніх взаємних залежностей.
- **Моно-чарт Helm (`erp-uno`)**: Параметризований шаблон проектування Kubernetes-ресурсів для всіх 6 просторів імен і 21 сервісу платформи.
- **Локальний сервіс репозиторіїв Gitea**: Контейнеризоване сховище коду та конфігурацій, що мінімізує залежність від глобальних мереж і запобігає дрейфу залежностей (dependency drift).
- **Контролер безперервної доставки ArgoCD**: Декларативний рушій, що відстежує розбіжності між станом локальних репозиторіїв і поточним станом кластера, забезпечуючи автоматичне приведення (sync) та самовідновлення (self-heal) ресурсів.

Описана архітектура забезпечує мінімізацію рутинної операційної діяльності (TOIL), високу відтворюваність середовищ та прискорене відновлення після збоїв. Повне розгортання кластера на сучасному обладнанні займає менше 5 хвилин. Система є сумісною з вітчизняними державними вимогами до комплексних систем захисту інформації (КСЗІ) завдяки використанню захищеного базового дистрибутиву UA Linux, мережевих політик ізоляції `NetworkPolicy` за принципом *least-privilege* та закритого OCI-реєстру образів.

9.1. Послідовність розгортання неймспейсів

Крок	Дія	Що відбувається
1/8	Namespaces + RBAC	`kubectl apply` на `share/namespaces.yaml`, `rbac.yaml`, `storage-class.yaml`, `networkpolicy.yaml`
2/8	erp-infra	`ns-dns`, `docker-registry`
3/8	erp-telemetry	`prometheus`, `grafana`, `otel-collector`
4/8	erp-security	`ca-pki`, `vpn-wireguard`, `ias-auth`
5/8	erp-ai	`ai-generation`
6/8	erp-services	`kvs-database`, `bpe-engine`, `rest-bpe`, `n2o-server`, `nitro-portal`
7/8	erp-apps	`lms-education`, `hl7-health`, `crm-documents`, `acc-accounting`, `wms-warehouse`, `cart-registers`, `pm-projects`, `itsm-incidents`, `chat-messenger`
8/8	Summary	Вивід Running/Total подів по namespace'ах

9.2. GitOps-рекомендації для production

Інструмент	Роль у pipeline
ArgoCD	Безперервна синхронізація Git → кластер. Автоматичне застосування змін при push до `main`
`values.rb` (Ruby)	Pre-commit hook: регенерація `helm/values.yaml` при зміні файлів у `lib/`
Kyverno / OPA	Policy enforcement: перевірка образів, resource limits, securityContext
GitHub Actions / GitLab CI	`helm lint` → `ruby values.rb -dry-run` → `helm diff` → `helm upgrade`

9.3. Співвідношення Git-репозиторіїв та компонентів GitOps

Для автоматизації доставки в концепції GitOps за допомогою ArgoCD нижче наведено таблицю відповідності вихідних Git-репозиторіїв коду та конфігурацій до відповідних ArgoCD-додатків і Helm-компонентів:

GitHub / Git репозиторій	Додаток ArgoCD / Helm-компонент
synrc/ns	ns-dns (DNS-сервер)
synrc/ca	ca-pki (Центр сертифікації)
zencrypted/vpn	vpn-wireguard (VPN шлюз)
synrc/ldap	ldap-directory (ABAC-каталог)
zencrypted/ias	ias-auth (Служба автентифікації)
synrc/chat	chat-messenger (WebSocket-брокер)
erpuno/mail	mail-delivery (Поштовий транспорт)
synrc/rest	rest-api (REST API BPE шлюз)
zencrypted/clearance	abac-clearance (Аудит допусків)
synrc/mach	mach-ivr (Сценарії телефонії)
synrc/bpe	bpe-processes (BPMN workflow рушій)
synrc/kvs	kvs-storage (СУБД KVS)
synrc/nitro	nitro-io (Рендеринг веб-порталу)
synrc/n2o	n2o-connections (WebSocket-сервер)
synrc/faiss	faiss-search (Векторний пошук)
prom/prometheus	prometheus (База метрик)
grafana/grafana	grafana (Візуалізація та дашборди)
otel/otel-collector	otel-collector (OpenTelemetry шлюз)
erpuno/edu	lms-education (Освітній сервіс LMS)
erpuno/health	hl7-health (Медичний HL7 FHIR API)
erpuno/crm	crm-documents (Документообіг)
erpuno/acc	acc-accounting (Фінансовий облік)
erpuno/warehouse	wms-warehouse (Складський облік)
erpuno/cart	cart-registers (Low-code державні реєстри)
erpuno/ai	ai-generation (Локальний ШІ-інференс)
erpuno/olap	olap-analytics (DuckDB аналітика)
erpuno/pm	pm-projects (Управління задачами)
erpuno/itsm	itsm-incidents (Service Desk / SLA)

9.4. Формалізована послідовність «холодного» запуску (Cold Boot Sequence)

Повна ініціалізація кластера з нульового стану (після перезавантаження фізичного хоста або при первинному розгортанні локального середовища розробки) є строго послідовним процесом, де кожен крок є пререквізитом для наступного. Формалізований життєвий цикл «холодного» запуску складається з 7 етапів:

1. **Ініціалізація та дзеркалювання локального сховища залежностей** (`./gitops.sh all`): На першому етапі розгортається виділена служба Gitea у Docker-контейнері за допомогою Docker Compose. Сценарій ініціалізує адміністратора, створює організаційну структуру `synrc` та виконує міграцію (bare-clone) репозиторіїв `ca`, `ns`, `ldap` з публічного сховища GitHub. Це створює автономну локальну базу вихідного коду.
2. **Декларативне створення кластера** (`./kind.sh create synrc`): Створюється локальний Kubernetes-кластер за допомогою інструменту KinD (Kubernetes-

in-Docker). Скрипт створює файл конфігурації `kind-config.yaml`, динамічно адаптуючи локальні шляхи монтування сховищ (hostPath) під домашню директорію поточного користувача хост-системи, що гарантує працездатність механізмів персистентного зберігання даних (Persistent Volumes) на будь-якому робочому місці.

3. **Налаштування контексту доступу** (`./kind.sh kind`): Виконується перемикання контексту управління утиліти `kubectl` на створений кластер (`kind-synrc`) для автентифікації наступних команд розгортання.
4. **Автономне збирання та завантаження образів контейнерів** (`./rebuild.sh`): Здійснюється компіляція та пакування образів для кастомних сервісів платформи. При цьому Docker-демон під час складання звертається до вихідного коду, розміщеного виключно на локальному сервері Gitea через віртуальну мережеву адресу хоста `host.docker.internal:3000`. Отримані образи завантажуються безпосередньо в локальний реєстр образів кластера KinD, повністю виключаючи залежність від публічних Docker-реєстрів.
5. **Розгортання базового та інфраструктурного шарів** (`./deploy.sh`): Створюється структура просторів імен, ініціалізуються ролі доступу RBAC, застосовуються мережеві політики ізоляції (NetworkPolicy deny-by-default) та розгортаються системні служби: Ingress-контролер Traefik, підсистема динамічного виділення сховищ Local Path Storage та внутрішній OCI-реєстр.
6. **Синхронізація параметрів конфігурації** (`./values.rb -force`): Запускається сценарій зворотної генерації для збору конфігурацій окремих мікросервісів та автоматичного формування зведеного файлу параметрів Helm (`helm/values.yaml`).
7. **Запуск контуру автоматичної GitOps-доставки** (`./argocd.sh`): Розгортаються системні компоненти ArgoCD у просторі імен `argocd`. Локальний репозиторій конфігурацій (`cd`) публікується у Gitea. Застосовується маніфест Ingress та декларативно реєструється кореневий додаток ArgoCD Application (`erp-uno`), який бере на себе подальшу синхронізацію та підтримку цілісності стану кластера. Також ініціалізується фоновий macOS LaunchAgent для автоматичного тунелювання порту консолі ArgoCD на хост-машину.

Описана послідовність гарантує повну ізольованість локального середовища розробника від глобальної мережі Інтернет після завершення першого кроку. Це забезпечує захист від збоїв мережевої інфраструктури, виключає ризики перевищення лімітів завантаження та гарантує точну повторюваність всього стеку розгортання платформи.

9.5. Пререквізити prereq.sh

Ініціалізація:

```
+=========================================================+
|  ERP/1: Підприємство / Deployment Prerequisites Check  |
+=========================================================+

[1] kubectl CLI
    ✓ kubectl  installed

[2] Helm Charts
    ✓ Helm v4.2.2+gb05881c installed

[3] Kubernetes Cluster
    ✓ Cluster accessible
       Version:
       Nodes:        1
       Total CPU: 10m
       Total Memory: 7GB
       ⚠ Cluster resources may be tight for 25 services
         Recommended: >= 4 CPU cores, >= 16GB RAM

[4] Storage
    ✓ Default StorageClass: standard

[5] Ingress Controller
    ⚠ No Ingress controller detected

[6] Cluster DNS
    ⚠ CoreDNS not found
       Internal service discovery may fail

[7] Namespace Setup
    ✓ Namespace 'erp-uno' ready to create

[8] Image Registry
    Configured: registry.erp.uno
    ⚠ Ensure images are accessible from cluster
       For private registries: set ImagePullSecret

[9] Docker (for local testing with docker-compose)
    ✓ Docker 29.6.1, installed
```

Фіналізація:

```
+=========================================================+
|                      Next Steps                         |
+=========================================================+

If all checks pass, run K8S raw deploy: bash deploy.sh
Or deploy via helm: cd helm && ./deploy.sh
```

9.6. Впровадження deploy.sh

Ініціалізація:

```
+=========================================================+
|    ERP/1: Підприємство / Kubernetes Dev Deployment     |
+=========================================================+

[1/8] Setting up namespaces...
namespace/erp-security created
namespace/erp-ai created
namespace/erp-telemetry created
namespace/erp-infra created
namespace/erp-apps created
namespace/erp-services created
serviceaccount/erp-security-sa created
role.rbac.authorization.k8s.io/erp-security-role created
rolebinding.rbac.authorization.k8s.io/erp-security-rolebinding created
serviceaccount/erp-ai-sa created
role.rbac.authorization.k8s.io/erp-ai-role created
rolebinding.rbac.authorization.k8s.io/erp-ai-rolebinding created
serviceaccount/erp-telemetry-sa created
role.rbac.authorization.k8s.io/erp-telemetry-role created
rolebinding.rbac.authorization.k8s.io/erp-telemetry-rolebinding created
serviceaccount/erp-infra-sa created
role.rbac.authorization.k8s.io/erp-infra-role created
rolebinding.rbac.authorization.k8s.io/erp-infra-rolebinding created
serviceaccount/erp-apps-sa created
role.rbac.authorization.k8s.io/erp-apps-role created
rolebinding.rbac.authorization.k8s.io/erp-apps-rolebinding created
serviceaccount/erp-services-sa created
role.rbac.authorization.k8s.io/erp-services-role created
rolebinding.rbac.authorization.k8s.io/erp-services-rolebinding created
storageclass.storage.k8s.io/local-path unchanged
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-deny-all created
    ✓ Namespaces, RBAC, and network policies created

[2/8] Deploying erp-infra...
    Deploying ns-dns to erp-infra...
    Deploying docker-registry to erp-infra...
persistentvolumeclaim/registry-data unchanged
deployment.apps/docker-registry unchanged
service/docker-registry unchanged

[3/8] Deploying erp-telemetry...
    Deploying prometheus to erp-telemetry...
persistentvolumeclaim/prometheus-data unchanged
```

```
statefulset.apps/prometheus unchanged
service/prometheus unchanged
    Deploying grafana to erp-telemetry...
deployment.apps/grafana unchanged
service/grafana unchanged
    Deploying otel-collector to erp-telemetry...

[4/8] Deploying erp-security...
    Deploying ca-pki to erp-security...

[5/8] Deploying erp-ai...
    Deploying ai-generation to erp-ai...

[6/8] Deploying erp-services...
    Deploying kvs-database to erp-services...
    Deploying bpe-engine to erp-services...
    Deploying rest-bpe to erp-services...
    Deploying n2o-server to erp-services...

[7/8] Deploying erp-apps...
    Deploying lms-education to erp-apps...
deployment.apps/lms-education unchanged
service/lms-education unchanged
horizontalpodautoscaler.autoscaling/lms-education unchanged
    Deploying hl7-health to erp-apps...
deployment.apps/hl7-health unchanged
service/hl7-health unchanged
horizontalpodautoscaler.autoscaling/hl7-health unchanged
    Deploying crm-documents to erp-apps...
    Deploying acc-accounting to erp-apps...
    Deploying wms-warehouse to erp-apps...
    Deploying cart-registers to erp-apps...
    Deploying pm-projects to erp-apps...
    Deploying itsm-incidents to erp-apps...

[8/8] Deployment Summary
    ================================================
               erp-infra:        0/       0 pods running
           erp-telemetry:        1/       2 pods running
            erp-security:        0/       0 pods running
                  erp-ai:        0/       0 pods running
            erp-services:        0/       0 pods running
                erp-apps:        0/       0 pods running
```

Фіналізація:

```
+=========================================================+
|           Multi-Namespace Deployment Done ✓             |
+=========================================================+

 Check Status:\n
   All pods: kubectl get pods -A
   All namespaces: kubectl get ns

 Access Services:\n
   UI (Nitro): kubectl port-forward -n erp-services svc/nitro-portal 8510:8510
   Prometheus: kubectl port-forward -n erp-telemetry svc/prometheus 9090:9090
   Grafana: kubectl port-forward -n erp-telemetry svc/grafana 3000:3000
   Registry: kubectl port-forward -n erp-infra svc/docker-registry 5000:5000

 Namespace Organization:\n
       erp-infra: ns-dns, docker-registry
   erp-telemetry: prometheus, grafana, otel-collector
    erp-security: ca-pki, vpn-wireguard, ias-auth
          erp-ai: ai-generation
    erp-services: kvs-database, bpe-engine, rest-bpe, n2o-server, nitro-portal
        erp-apps: lms-education, hl7-health, acc-accounting, wms-warehouse, cart-registers
        erp-apps: crm-documents, pm-projects, itsm-incidents, chat-messenger
```

9.7. Генерація Helm моно-чарту values.rb

Ініціалізація:

```
 Generating helm/values.yaml  (version: 2024.6.15)
Overwrite /Users/tonpa/depot/erpuno/cd/helm/values.yaml? (y/N): y
✓ Generated successfully!
   Namespaces : 6
   Services   : 21
```

9.8. Пакетування helm/deploy

Ініціалізація:

```
+=========================================================+
|    ERP/1: Підприємство Helm Deployment                  |
+=========================================================+

[1/5] Cleaning up immutable resources...
No resources found

[2/5] Ensuring namespaces...
namespace/erp-security created
namespace/erp-ai created
namespace/erp-telemetry created
namespace/erp-infra created
namespace/erp-apps created
namespace/erp-services created

[3/5] Deploying shared infrastructure...
serviceaccount/erp-security-sa created
role.rbac.authorization.k8s.io/erp-security-role created
rolebinding.rbac.authorization.k8s.io/erp-security-rolebinding created
serviceaccount/erp-ai-sa created
role.rbac.authorization.k8s.io/erp-ai-role created
rolebinding.rbac.authorization.k8s.io/erp-ai-rolebinding created
serviceaccount/erp-telemetry-sa created
role.rbac.authorization.k8s.io/erp-telemetry-role created
rolebinding.rbac.authorization.k8s.io/erp-telemetry-rolebinding created
serviceaccount/erp-infra-sa created
role.rbac.authorization.k8s.io/erp-infra-role created
rolebinding.rbac.authorization.k8s.io/erp-infra-rolebinding created
serviceaccount/erp-apps-sa created
role.rbac.authorization.k8s.io/erp-apps-role created
rolebinding.rbac.authorization.k8s.io/erp-apps-rolebinding created
serviceaccount/erp-services-sa created
role.rbac.authorization.k8s.io/erp-services-role created
rolebinding.rbac.authorization.k8s.io/erp-services-rolebinding created
storageclass.storage.k8s.io/local-path unchanged
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-deny-all created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-allow-internal created
networkpolicy.networking.k8s.io/erp-deny-all created
    ✓ Shared resources applied

[4/5] Patching Helm ownership metadata...
Applying Helm ownership metadata to all resources...
→ Processing namespace: erp-infra
namespace/erp-infra not labeled
namespace/erp-infra annotated
```

```
serviceaccount/erp-infra-sa labeled
serviceaccount/erp-infra-sa annotated
→ Processing namespace: erp-telemetry
namespace/erp-telemetry not labeled
namespace/erp-telemetry annotated
serviceaccount/erp-telemetry-sa labeled
serviceaccount/erp-telemetry-sa annotated
→ Processing namespace: erp-security
namespace/erp-security not labeled
namespace/erp-security annotated
serviceaccount/erp-security-sa labeled
serviceaccount/erp-security-sa annotated
→ Processing namespace: erp-ai
namespace/erp-ai not labeled
namespace/erp-ai annotated
serviceaccount/erp-ai-sa labeled
serviceaccount/erp-ai-sa annotated
→ Processing namespace: erp-services
namespace/erp-services not labeled
namespace/erp-services annotated
serviceaccount/erp-services-sa labeled
serviceaccount/erp-services-sa annotated
→ Processing namespace: erp-apps
namespace/erp-apps not labeled
namespace/erp-apps annotated
serviceaccount/erp-apps-sa labeled
serviceaccount/erp-apps-sa annotated
✓ All resources patched with Helm ownership metadata.
    ✓ Ownership metadata patched

[5/5] Deploying Helm chart...
Release "erp-uno" does not exist. Installing it now.
NAME: erp-uno
LAST DEPLOYED: Sat Jul  4 11:58:19 2026
NAMESPACE: default
STATUS: deployed
REVISION: 1
DESCRIPTION: Install complete
TEST SUITE: None
```

Фіналізація:

```
+=========================================================+
|           Helm Deployment Complete ✓                    |
+=========================================================+

📊 Quick Status:
erp-ai               Active   3s
erp-apps             Active   3s
erp-infra            Active   3s
erp-security         Active   3s
erp-services         Active   3s
erp-telemetry        Active   3s
Pods:
   4 Running
   1 ContainerCreating

🌍 Useful commands:
   kubectl get pods -A
   kubectl get hpa -A
   helm status erp-uno
   helm history erp-uno
```

9.9. Деконструкція кластеру

Деініціалізація:

```
release "erp-uno" uninstalled
namespace "erp-ai" deleted
namespace "erp-infra" deleted
namespace "erp-security" deleted
namespace "erp-services" deleted
namespace "erp-apps" deleted
namespace "erp-telemetry" deleted
```

10. Локальний контур GitOps: Gitea та ArgoCD

Впровадження концепції GitOps у локальному середовищі розробки дозволяє мінімізувати розходження конфігурацій (configuration drift) між локальними та продуктовими середовищами, забезпечити повну відтворюваність та усунути залежність від зовнішніх мережевих ресурсів під час розгортання.

Локальний контур GitOps платформи ERP/1 базується на двох ключових інструментах: локальному Git-сервері Gitea та контролері безперервної синхронізації ArgoCD.

10.1. Концепція локального GitOps та ізоляція залежностей

Класичний підхід до GitOps передбачає, що Git-репозиторій є єдиним джерелом істини (Single Source of Truth) для стану кластера. Проте, використання публічних сервісів, таких як GitHub, для розгортання локальних dev-середовищ має ряд недоліків:

- **Мережева залежність і ліміти**: Обмеження швидкості (rate limits) API GitHub та залежність від стабільного інтернет-з'єднання сповільнюють ініціалізацію.
- **Дрейф версій (Dependency Drift)**: Видалення або оновлення сторонніх репозиторіїв може призвести до неможливості відтворити складання кластера.
- **Конфіденційність**: Зберігання локальних конфігурацій та секретів у публічному хмарі несе ризики безпеки.

Для вирішення цих проблем локальний контур ERP/1 повністю ізолює вихідний код та маніфести у межах розробницької станції, використовуючи локальний сервер Gitea як проміжне сховище для ArgoCD та докер-білдера.

10.2. Локальний Git-сервер Gitea

Локальний Git-сервер Gitea розгортається у виділеному Docker-контейнері за допомогою Docker Compose, конфігурація якого описана у файлі `compose/gitea.yaml`:

```
services:
  gitea:
    image: gitea/gitea:latest
    container_name: gitea
    restart: always
    ports:
      - "3000:3000"    # Web UI
      - "2222:22"      # SSH
    volumes:
      - ./gitea/data:/data
      - ./gitea/ssh:/data/ssh
    environment:
      - USER_UID=1000
```

```
    - USER_GID=1000
    - GITEA__database__DB_TYPE=sqlite3
    - GITEA__database__PATH=/data/gitea.db
    - GITEA__security__INSTALL_LOCK=true
    - GITEA__server__DOMAIN=localhost
    - GITEA__server__SSH_DOMAIN=localhost
    - GITEA__server__SSH_PORT=2222
    - GITEA__server__ROOT_URL=http://localhost:3000/
  shm_size: '128m'
```

Ключові архітектурні рішення конфігурації Gitea:

1. **SQLite як СУБД**: Використання SQLite3 виключає необхідність запуску окремого сервера баз даних, забезпечуючи споживання оперативної пам'яті контейнером в межах 100 МБ та швидкий запуск (< 5 секунд).
2. **Блокування інсталятора (INSTALL_LOCK)**: Попереднє встановлення значення `true` дозволяє пропустити інтерактивний веб-інтерфейс первинного налаштування. Gitea відразу готова до роботи через API.
3. **Локальне монтування**: Усі дані репозиторіїв та налаштувань зберігаються у локальній директорії `./compose/gitea/data`, що дозволяє легко виконувати резервне копіювання.

Автоматизація управління Gitea здійснюється через скрипт `gitops.sh`. Команда `./gitops.sh setup` виконує запуск контейнера, створює адміністратора `root` (з паролем `ErpUnoGitea2026`), організацію `synrc` та репозиторії `ca`, `ns`, `ldap`. Наступна команда `./gitops.sh migrate` робить bare-clone репозиторіїв із GitHub (`github.com/synrc/*`) та дзеркалює їх у локальний Gitea.

Це дозволяє локальному Docker-демону під час збирання образів звертатися безпосередньо до хост-машини (`host.docker.internal:3000`) для отримання вихідного коду сервісів, забезпечуючи 100% автономність складання.

10.3. Розгортання та інтеграція ArgoCD

ArgoCD виступає в ролі GitOps-контролера всередині KinD-кластера, забезпечуючи декларативне розгортання та автоматичну синхронізацію стану ресурсів.

Компоненти розгортання ArgoCD (каталог `./argocd/`):

- `install.yaml` — офіційні маніфести ArgoCD версії v2.11+, які розгортаються у виділеному просторі імен `argocd`.
- `ingress.yaml` — конфігурація Traefik Ingress для надання доступу до веб-консолі за доменним ім'ям `argocd.erp-uno.local` через стандартний HTTP-порт:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: argocd-server-ingress
  namespace: argocd
  annotations:
    traefik.ingress.kubernetes.io/router.entrypoints: web
spec:
  ingressClassName: traefik
```

```
  rules:
  - host: argocd.erp-uno.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: argocd-server
            port:
              number: 80
```

- `application.yaml` — маніфест кореневого додатку ArgoCD Application (`erp-uno`), який реалізує патерн «App-of-Apps»:

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: erp-uno
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: 'http://host.docker.internal:3000/synrc/cd.git'
    path: helm
    targetRevision: HEAD
    helm:
      valueFiles:
        - values.yaml
      parameters:
        - name: global.domain
          value: erp.uno
        - name: global.environment
          value: production
  destination:
    server: 'https://kubernetes.default.svc'
    namespace: default
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
      - ServerSideApply=true
```

Особливості інтеграції ArgoCD в ERP/1:

1. **Зв'язок із хостом**: Параметр `repoURL` вказує на `http://host.docker.internal:3000/synrc/cd.git`. Оскільки ArgoCD працює всередині контейнерів KinD-кластера, адреса `host.docker.internal` дозволяє виходити на Git-сервер Gitea, який працює на хост-системі розробника.
2. **Автоматична синхронізація**: Політика `automated.prune: true` та `automated.selfHeal: true` забезпечує миттєве видалення нерелевантних ресурсів та автоматичне відновлення стану при виявленні відхилень (out of sync).

3. **ServerSideApply**: Використання опції `ServerSideApply=true` вирішує проблеми з перевищенням ліміту розміру анотацій у маніфестах при роботі з великими конфігураційними картами.
4. **Постійний Port-Forward**: Скрипт `argocd.sh` автоматично створює та завантажує LaunchAgent для macOS (`uno.erp.argocd-portforward.plist`), що підтримує постійний тунель `kubectl port-forward svc/argocd-server 8080:80` у фоновому режимі розробника.

11. Механіка оркестрації та життєвий цикл розгортання

11.1. Формальний контур ініціалізації: від DNS та хостів до реєстру

Повне розгортання та підготовка локальної ШІ-інфраструктури виконується у суворій послідовності кроків, починаючи від налаштування хост-мережі та закінчуючи запуском OCI-реєстру:

1. **Налаштування хост-резолвінгу (/etc/hosts)**: На кожному фізичному вузлі та керуючій машині прописуються статичні записи домену `erp.uno` та локального OCI-реєстру `registry.erp.uno` для забезпечення роботи транспортного контуру без зовнішнього DNS-сервера.
2. **Налаштування сертифікатів безпеки OCI-реєстру**: Для виключення помилок небезпечного з'єднання (insecure registry) на всіх робочих вузлах Kubernetes прописується кореневий CA-сертифікат у каталозі `/etc/docker/certs.d/registry.erp.uno/ca.crt` та `/etc/containerd/certs.d/`.
3. **Запуск OCI Docker Registry**: На інфраструктурному хості запускається OCI-реєстр як системна служба `containerd` або Docker-контейнер з монтуванням локального сховища для образів компонентів.
4. **Складання та публікація образів (CI/CD)**: Образи мікросервісів збираються на базі **Alpine Linux**, тегуються версією релізу та завантажуються (`docker push`) до `registry.erp.uno`.
5. **Ініціалізація інфраструктури Kubernetes**: За допомогою `deploy.sh` або Helm застосовуються глобальні ресурси (Namespaces, StorageClasses, Calico NetworkPolicies та RBAC ролі).

11.2. Кубернетес за лаштунками (Behind the Scenes)

При виконанні команди `kubectl apply -f manifest.yaml` або запуску Helm-інсталяції ядро та базові служби Kubernetes виконують наступні внутрішні операції:

- **kube-apiserver** (Точка входу API): Отримує HTTPS-запит, автентифікує клієнта, авторизує дію на основі RBAC-правил, перевіряє схему об'єкта та зберігає декларативний стан у розподіленому сховищі **etcd**.
- **kube-controller-manager** (Контролери стану):
  - *Deployment controller* створює відповідний `ReplicaSet`.
  - *ReplicaSet controller* генерує специфікації для створення конкретних подів від-

повідно до ліміту реплік.

  - *StatefulSet controller* гарантує запуск подів з унікальними індексами (`prometheus-0`, `kvs-database-0`) та прив'язку стабільних сховищ PVC.

- **kube-scheduler** (Планувальник вузлів): Відстежує нерозподілені поди, аналізує вимоги до ресурсів (`requests.cpu`, `requests.memory`) та призначає поди на вузли з достатнім вільним обсягом ресурсів.
- **kubelet** (Робочий агент на вузлі): Отримує оновлення специфікації подів для свого вузла від API-сервера та викликає контейнерне середовище виконання **containerd** через інтерфейс CRI (Container Runtime Interface).
- **containerd** (Контейнерне середовище):
  - Здійснює запит та завантаження відповідного образу з OCI-реєстру `registry.erp.uno`.
  - Викликає мережевий плагін CNI (Calico) для виділення внутрішньої IP-адреси та створення віртуальних інтерфейсів.
  - Налаштовує обмеження ресурсів на рівні ядра ОС Linux через механізми `cgroups` (відповідно до `limits.cpu` та `limits.memory`).
  - Запускає ізольовані процеси контейнера та відслідковує їхній стан для передачі звітів про liveness/readiness проби назад до kubelet.

11.3. Генерація values.yaml за допомогою values.rb: зворотний GitOps

За класичним підходом Helm розроблявся як інструмент шаблонізації Kubernetes маніфестів на основі наперед визначених статичних змінних (`values.yaml`). Проте, при великій кількості мікросервісів це призводить до дублювання та розходження конфігурацій.

У платформі ERP/1 застосовується концепція **зворотного GitOps** за допомогою скрипта `values.rb`:

- **Маніфест як джерело істини**: Розробники редагують стандартні декларативні файли `deployment.yaml`, `service.yaml` та `hpa.yaml` у директоріях `lib/`.
- **Статична компіляція**: Скрипт `values.rb` обходить усі директорії компонентів, парсить YAML-файли та вилучає критичні параметри (кількість реплік, порти, ліміти ресурсів CPU/RAM, наявність сховищ та HPA).
- **Генерація конфігурації Helm**: Зібрані дані серіалізуються у єдиний файл `helm/values.yaml`. Це дозволяє зберегти простоту індивідуального супроводження сервісів у `lib/` та одночасно використовувати переваги Helm (реліз-менеджмент, відкати версій, декларативне оновлення) у продуктивному контурі.

12. Incident Management / On-Call

12.1. Severity-рівні

Рівень	Критерій	Відповідь
Sev1	Платформа недоступна (erp-telemetry або erp-security Down)	Негайно. MTTR < 30 хв. Обов'язковий постмортем
Sev2	Один або більше erp-apps / erp-services недоступні	До 1 год. Постмортем обов'язковий
Sev3	Деградація: висока latency, часткова відмова	Протягом робочого дня
Sev4	Незначні проблеми, cosmetic warnings	Наступний спринт

12.2. Playbooks

Інцидент	Кроки діагностики
Pod CrashLoopBackOff	`kubectl describe pod -n {ns} {pod}` → `kubectl logs -previous` → перевірити livenessProbe → resource limits
ImagePullBackOff	Перевірити авторизацію у приватному `docker-registry` → перевірити шлях до образу у `values.yaml` → pull secret
Prometheus PV Full	Перевірити вільне місце на томах (`df -h`) → переглянути retention policy → провести tsdb clean/compaction
NetworkPolicy violation	`kubectl get networkpolicy -A` → тест: `kubectl exec -it ... -- curl http://{svc}:{port}`
High CPU / Memory	`kubectl top pods -n {ns}` → `kubectl get hpa -A` → `kubectl scale deployment {name} -replicas=3 -n {ns}`

12.3. Аналіз реальних відмов та постмортеми

Промислова експлуатація платформи в державних органах сектору безпеки сформувала базу знань щодо запобігання критичним аваріям. Нижче наведено два ключових сценарії з реальної практики експлуатації та заходи щодо підвищення надійності:

1. **Кейс 1: Амортизація навантаження під час пікової HTTP-активності**
   - *Опис інциденту:* Під час масового звернення громадян до кабінетів електронних послуг навантаження зросло до 150,000 HTTP-запитів на секунду, що забило пули з'єднань веб-акцепторів Erlang.
   - *Коренева причина:* Ліміт TCP-акцепторів за замовчуванням (25k) та низька швидкість парсингу JSON-пакетів привели до вичерпання CPU.
   - *Рішення та стабілізація:* Впроваджено лімітування на рівні Ingress Gateway. Пул акцепторів збільшено до 200,000, а парсинг JSON переведено на оптимізовані C99 NIF-бібліотеки. Це знизило утилізацію CPU з 95% до 12% при аналогічному трафіку.
2. **Кейс 2: Запобігання розділенню мережі (Split-Brain) в кластері СУБД**
   - *Опис інциденту:* Внаслідок короткочасного збою мережевого комутатора відбулося розділення зв'язку між нодами розподіленої Mnesia/KVS. Ноди утворили два сегменти (Split-Brain) та продовжували приймати транзакції на запис.
   - *Коренева причина:* Автоматична реконсиляція Mnesia була вимкнена, що призвело до розходження стану реплік.
   - *Рішення та стабілізація:* Впроваджено обробники події `mnesia_down` та сценарії автоматичного злиття (merge/reconciliation) на основі вектора часових міток транзакцій. Налаштовано автоматичне повторне об'єднання сегментів та синхронізацію без втрати цілісності даних.

13. Патерни відмовостійкості (Resilience Patterns)

Для забезпечення високої доступності (High Availability) мікросервісів та їх здатності витримувати часткові відмови інфраструктури, в ERP/1 застосовуються архітектурні патерни відмовостійкості. Основні принципи базуються на індустріальних стандартах, описаних у класичній праці Michael T. Nygard **«Release It!»**.

13.1. Основні архітектурні патерни

- **Retries з Exponential Backoff**: Стандартний підхід для клієнтських сервісів. У разі тимчасового збою мережі або недоступності сервісу-провайдера, запит повторюється. Для уникнення шторму повторних запитів (retry storm) використовується експоненційне збільшення затримки між спробами. Ефективно лише для ідемпотентних (idempotent) операцій та stateless-навантажень.
- **Deadlines (Таймаути)**: Замість локальних таймаутів між двома сервісами, визначається глобальний час очікування на стороні ініціатора запиту. Цей дедлайн передається по всьому ланцюжку викликів (через контекст), що запобігає марному використанню ресурсів, якщо клієнт вже розірвав з'єднання.
- **Rate Limiting**: Застосовується на рівні Ingress Gateway та API-шлюзів для обмеження кількості запитів від одного клієнта (наприклад, за алгоритмами Token Bucket або Leaky Bucket). Це захищає систему від перенавантаження (DDoS або помилок у клієнтському коді) і гарантує доступність для інших користувачів.
- **Circuit Breaker (Запобіжник)**: Якщо сервіс-провайдер починає масово повертати помилки, клієнтський сервіс тимчасово припиняє надсилати до нього запити («розмикає ланцюг»), щоб дати провайдеру час на відновлення. Через певний час клієнт пропускає тестовий запит («напіввідкритий стан») і, у разі успіху, відновлює нормальну роботу.
- **Fallback (Dummy / Rich Client)**: Запасний план (Plan B). Якщо основний сервіс недоступний, система може повернути кешовані дані, деградувати функціональність (наприклад, вимкнути персоналізовані рекомендації, залишивши базовий каталог) або обробити запит на стороні клієнта.

Ці патерни реалізуються як на рівні прикладного коду мікросервісів, так і на рівні інфраструктури (Traefik middlewares).

14. Chaos Engineering та Resilience Testing

14.1. Стратегія ін'єкції відмов

Тип відмови	Інструмент	Очікуваний результат
Pod Kill	Chaos Mesh	HPA/Deployment відновлює репліки; SLO зберігається
Node Drain	`kubectl drain`	Поди евакуюються завдяки podAntiAffinity
Network Partition	Chaos Mesh NetworkChaos	NetworkPolicy ізолює деградацію; telemetry продовжує роботу
Memory Pressure	LitmusChaos	Перевірка limits і OOMKill-поведінки
Resource Starvation	Chaos Mesh StressChaos	HPA масштабується (MTTD $<$ 2 хв)

14.2. RTO / RPO цілі

Сервіс	RTO	RPO	Механізм
prometheus (StatefulSet)	$<$ 5 хв	$<$ 1 хв	StatefulSet + PVC; авто-перезапуск
rest-bpe (Deployment)	$<$ 3 хв	N/A	Rolling update
hl7-health (HPA min 2)	$<$ 2 хв	N/A	HPA + podAntiAffinity
docker-registry	$<$ 5 хв	$<$ 5 хв	PVC 10 Gi; швидкий перезапуск

15. Інженерна культура та метрики доставки (DORA)

Для забезпечення стабільної якості (Quality Gate) та прогнозованості релізів, розробка ERP/1 спирається на метрики продуктивності доставки ПЗ, визначені дослідженням DORA (DevOps Research and Assessment) та описані у книзі «**Accelerate**».

15.1. DORA Метрики

Продуктивність команд (Software delivery performance tempo) та стабільність (Software stability) вимірюються за чотирма ключовими показниками:

- **Deployment Frequency**: Як часто код розгортається у production (в ERP/1 ціль — $\geq$ 1 разу на тиждень через ArgoCD).
- **Lead Time for Changes**: Час від коміту до розгортання в production (показник ефективності CI/CD пайплайну).
- **MTTR (Mean Time To Recover)**: Середній час відновлення сервісу після збою (ціль для критичних компонентів — $<$ 30 хв).
- **Change Failure Rate**: Відсоток релізів, які призводять до збоїв та потребують відкату (rollback) або екстрених виправлень (ціль — $<$ 5%).

15.2. Pipeline-Driven Organization та Культура

Автоматизація доставки (Continuous Delivery) є лише частиною успіху. Спираючись на концепцію **Pipeline-driven organization** (Roy Osherove), процес розробки будується навколо Internal Developer Platform (IDP). Це дозволяє розробникам самостійно керувати життєвим циклом своїх сервісів через декларативні пайплайни.

Крім того, відповідно до результатів **Google Project Aristotle**, критичним фактором ефективності є *психологічна безпека* в командах. Це означає, що інциденти розглядаються без пошуку винних (blameless postmortems), що дозволяє відкрито аналізувати кореневі причини збоїв.

При написанні коду розробники керуються принципами **Strategic Programming** (John Ousterhout): пріоритетом є створення надійної та зрозумілої архітектури (great design), а не лише працюючого коду, що дозволяє уникати накопичення технічного боргу.

16. Операційні метрики та ресурсний бюджет

16.1. Ключові SRE-метрики

Метрика	Ціль	Вимірювання
MTTR (Sev1)	< 30 хв	Grafana: час між першим алертом та відновленням
MTTD	< 5 хв	Prometheus alerting rules з `for: 5m`
Error Budget Burn Rate	< 1.0	`sum(rate(errors[1h])) / error_budget_rate`
TOIL (інженерний час)	$< 50\%$	Відслідковується у JIRA / itsm-incidents
Deployment Frequency	≥ 1/тижд.	Git commits to main; ArgoCD sync count
Change Failure Rate	$< 5\%$	Rollback / загальна кількість деплойментів

16.2. Ресурсне навантаження кластеру

Кластер: 1 вузол, 10 vCPU, 8 ГБ RAM (arm64, Docker Desktop).

Namespace	Сервісів	CPU req.	Mem req.	PVC
erp-infra	2	150m	384Mi	10 Gi
erp-telemetry	4	350m	896Mi	30 Gi
erp-security	1	100m	256Mi	—
erp-ai	1	100m	256Mi	—
erp-services	5	500m	1.25 Gi	—
erp-apps	8	900m	2.25 Gi	—
Разом	**21**	**2.1 cores**	**5.3 Gi**	**40 Gi**

17. Roadmap надійності ERP/1

Квартал	Напрям	Конкретні дії
Q3 2026	**Розширення**	Додавання нових модулів у erp-apps; розширення HPA на erp-services; Alertmanager routing
Q3 2026	**Observability**	Додаткові Grafana дашборди per-namespace; SLO burn-rate алерти
Q4 2026	**GitOps**	ArgoCD; авто-rollback на SLO-порушення; pre-commit hook для values.rb
Q4 2026	**Security**	ias-auth + vpn-wireguard; PKI-ланцюжок через ca-pki; mTLS між namespace'ами
Q1 2027	**Chaos**	Chaos Mesh pod-kill тести; quarterly fire drill
Q1 2027	**Scalability**	Додатковий worker-вузол; multi-node kind; HPA під навантаженням
Q2 2027	**Production**	Managed Kubernetes (GKE/EKS/AKS / bare-metal); TLS Ingress; DNS erp.uno

18. Висновок

SRE Handbook ERP/1 описує живу, еволюційну платформу. Ключові архітектурні рішення:

1. **Namespace-ізоляція** — 6 рівнів із NetworkPolicy deny-by-default забезпечують безпеку та незалежне масштабування.
2. **Helm-driven GitOps** — єдиний `values.yaml`, автоматично синхронізований із `lib/` через `values.rb`, є декларативним контрактом усієї платформи.
3. **SLO-орієнтований підхід** — диференційовані SLO (99.5%–99.99%) відображають реальну критичність сервісів.
4. **Observability-first** — erp-telemetry розгортається другим та має найвищий SLO 99.99%.
5. **Мінімальний TOIL** — deploy.sh + values.rb + майбутній ArgoCD зводять ручну роботу нанівець.

Кожна зміна в системі повинна:

- Проходити через Git (єдине джерело істини)
- Перевірятись через `helm lint` та `ruby values.rb -dry-run`
- Вимірюватись через Prometheus SLI
- Документуватись у постмортемі при Sev1/Sev2

«Reliability is the most important feature.»

— Google SRE Book, 2016