ERP/1: Підприємство

SRE Handbook

Настанови системного адміністратора для розгортання, спостереження і обслуговування 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

Зміст

Зміст

Анотація

Цей **SRE Handbook** є авторитетним технічним довідником з Site Reliability Engineering для платформи **ERP/1: Підприємство**. Він охоплює повну архітектуру multi-namespace Kubernetes-кластеру, деталізовану структуру Helm-чартів, специфікацію всіх маніфестів компонентів, а також практики SRE: SLI/SLO, error budgets, on-call, chaos engineering і GitOps. Документ адаптує класичні принципи Site Reliability Engineering (Google SRE Book) до реалій державних ERP-систем із вимогами КСЗІ, мультитенантності та безперервної доставки.

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 **6 namespace'ів** та **20 мікросервісів**, організованих за принципом *Separation of Concerns*:

Порядок розгортання (залежності):
`infra` → `telemetry` → `security` → `ai` → `services` → `apps`

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

2.2. 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), `loki` (логи, :8080), `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-engine` (рушій бізнес-процесів), `n2o-server` (No-code/Low-code), `kvs-database` (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`

2.3. Поточний стан кластеру

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

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	loki-77d4b9c6f9-q8r2t	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-engine-9f4d2a-xk	1/1	Running	14d 6h
erp-services	kvs-database-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 репозиторію

3.1. Огляд директорій

Шлях	Призначення
`helm/`	Helm-чарт верхнього рівня: `Chart.yaml`, `values.yaml`
`helm/templates/`	Go-шаблони: `deployments.yaml`, `services.yaml`, `hpa.yaml`, `namespaces.yaml`
`lib/`	Сирі маніфести на компонент: `lib/{namespace}/{service}/`
`lib/erp-*/`	По одній директорії на кожен з 6 namespace'ів
`share/`	Кластерні ресурси: `namespaces.yaml`, `rbac.yaml`, `networkpolicy.yaml`, `storage-class.yaml`
`values.rb`	Ruby-скрипт: читає `lib/` → генерує `helm/values.yaml`
`deploy.sh`	Bash-скрипт 8-крокового розгортання
`prereq.sh`	Перевірка та встановлення передумов (kind, kubectl, helm)
`delete.sh`	Видалення всіх ERP-ресурсів з кластеру

3.2. 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`

Файл	Зміст та ключові параметри
`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

4. Каталог компонентів ERP/1

4.1. Зведена таблиця маніфестів

Сервіс	Namespace	Kind	Port	CPU req/lim	Маніфести
erp-infra — Інфраструктурний рівень					
ns-dns	erp-infra	Deployment	8080	100m/500m	deployment, service, values
docker-registry	erp-infra	Deployment	5000	50m/200m	deployment, service, pvc, values
erp-telemetry — Observability рівень					
prometheus	erp-telemetry	StatefulSet	9090	100m/500m	deployment, service, pvc, values
grafana	erp-telemetry	Deployment	3000	50m/200m	deployment, service, pvc, values
loki	erp-telemetry	Deployment	8403	100m/500m	deployment, service, values
otel-collector	erp-telemetry	Deployment	8404	100m/500m	deployment, service, values
erp-security — Безпековий рівень					
ca-pki	erp-security	Deployment	8201	100m/500m	deployment, service, values
erp-ai — AI рівень					
ai-generation	erp-ai	Deployment	8601	100m/500m	deployment, service, values
erp-services — Сервісний рівень					
rest-bpe	erp-services	Deployment	8303	100m/500m	deployment, service, values
bpe-engine	erp-services	Deployment	8302	100m/500m	deployment, service, values
n2o-server	erp-services	Deployment	8511	100m/500m	deployment, service, values
kvs-database	erp-services	Deployment	8301	100m/500m	deployment, service, values
chat-messenger	erp-services	Deployment	8507	100m/500m	deployment, service, values
erp-apps — Бізнес рівень					
hl7-health	erp-apps	Deployment	8502	100m/500m	deployment, service, hpa, values
lms-education	erp-apps	Deployment	8506	100m/500m	deployment, service, hpa, values
wms-warehouse	erp-apps	Deployment	8505	100m/500m	deployment, service, values
acc-accounting	erp-apps	Deployment	8504	100m/500m	deployment, service, values
crm-documents	erp-apps	Deployment	8503	100m/500m	deployment, service, values
cart-registers	erp-apps	Deployment	8501	100m/500m	deployment, service, values
pm-projects	erp-apps	Deployment	8514	100m/500m	deployment, service, values
itsm-incidents	erp-apps	Deployment	8513	100m/500m	deployment, service, values

[HL] headless service (`clusterIP: None`) для StatefulSet

4.2. 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.memo	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

4.3. 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

4.4. 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

4.5. 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:

`lib/` $\xrightarrow{\text{values.rb}}$ `helm/values.yaml` $\xrightarrow{\text{Helm templates}}$ **Kubernetes manifests**

`lib/` — джерело істини. Будь-яка зміна в `lib/` вимагає перегенерації: `ruby values.rb -force`

```
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

# Algorithm:
# 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 та Error Budgets

7.1. Принцип Error Budget

Error Budget $= 1 - \text{SLO}$

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.

7.2. SLO по namespace'ах ERP/1

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. SLI — сигнали надійності (RED Method)

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))`

SLI	Тип	PromQL-вираз
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`

8. Observability Stack

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 та Loki. Доступ: `kubectl port-forward -n erp-telemetry svc/grafana 3000:3000`
`loki`	:8080	Log Aggregation. Кореляція логів з метриками через Grafana
`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. Автоматизація та GitOps

9.1. Процес розгортання: 8 кроків

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`, `loki`, `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 / Flux	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`
Renovate Bot	Автоматичне оновлення версій образів у `values.yaml` через PR

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-bpe (REST API BPE шлюз)
zencrypted/clearance	abac-clearance (Аудит допусків)
synrc/mach	mach-ivr (Сценарії телефонії)
synrc/bpe	bpe-engine (BPMN workflow рушій)
synrc/kvs	kvs-database (СУБД KVS)
synrc/nitro	nitro-portal (Рендеринг веб-порталу)
synrc/n2o	n2o-server (WebSocket-сервер)
synrc/faiss	faiss-search (Векторний пошук)
prom/prometheus	prometheus (База метрик)
grafana/grafana	grafana (Візуалізація та дашборди)
grafana/loki	loki (Збір системних логів)
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)

10. Механіка оркестрації та життєвий цикл розгортання

10.1. 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).

10.2. 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.

10.3. 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.

11. Incident Management / On-Call

11.1. 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

11.2. 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}`

11.3. 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.

12. Chaos Engineering та Resilience Testing

12.1. 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)

12.2. 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

13. Операційні метрики та ресурсний бюджет

13.1. 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

13.2. 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**

14. Розгортання кластеру

Розгортання платформи ERP/1: Підприємство є повністю автоматизованим, декларативним та відтворюваним процесом, побудованим за принципами GitOps та Site Reliability Engineering. Основна інфраструктура розгортання складається з набору спеціалізованих bash-скриптів, Ruby-генератора конфігурації та Helm umbrella-чарту, що забезпечує єдине джерело істини для всього кластеру. Система розгортання включає чотири ключові компоненти:

`prereq.sh` — скрипт попередньої перевірки середовища. Він перевіряє наявність та версії kubectl, helm, kind, стан Kubernetes-кластеру, StorageClass, Ingress-контролер, DNS та доступ до приватного реєстру образів. У разі відсутності критичних компонентів скрипт виводить чіткі рекомендації щодо їх встановлення.

`values.rb` — Ruby-скрипт зворотної генерації Helm-значень. Він сканує директорію lib/, парсить сирі Kubernetes-маніфести (deployment.yaml, service.yaml, hpa.yaml, pvc.yaml) і автоматично формує централізований файл helm/values.yaml. Це дозволяє розробникам редагувати прості маніфести для кожного сервісу, а Helm-чарт залишається єдиним точкою входу для розгортання.

`deploy.sh` — основний оркестратор розгортання. Скрипт виконує 8-крокове послідовне розгортання з урахуванням топологічних залежностей між namespace'ами (erp-infra → erp-telemetry → erp-security → erp-ai → erp-services → erp-apps). На кожному кроці відбувається застосування RBAC, NetworkPolicy, StatefulSet-ів з PVC, HPA та моніторинг статусу подів. Скрипт також генерує детальний звіт про стан кластеру після завершення.

Helm umbrella-чарт erp-uno — єдиний пакет, що охоплює всі 6 namespace'ів та 21 сервіс. Використовуючи Go-шаблони (templates/deployments.yaml, services.yaml тощо), чарт генерує всі Kubernetes-ресурси з урахуванням глобальних налаштувань (global.domain, imagePullPolicy, версія образів тощо).

Така архітектура забезпечує мінімізацію TOIL, високу відтворюваність між середовищами (development, staging, production) та швидке відновлення після збоїв. Повне розгортання кластеру займає менше 5 хвилин на сучасному hardware. Усі зміни проходять через Git, проходять dry-run перевірки та автоматично синхронізуються через ArgoCD у production-контурі. Система повністю сумісна з державними вимогами КСЗІ/АСБ завдяки використанню UA Linux, NetworkPolicy deny-by-default, RBAC least-privilege та приватного OCI-реєстру.

14.1. 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
```

14.2. 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 loki to erp-telemetry...
    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, loki, 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
```

14.3. 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
```

14.4. 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
```

14.5. 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
```

15. 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; Loki alerting rules; 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

16. 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