OpsDesk — Local-first Kubernetes DevOps/SRE Portfolio (Detailed Technical Report)

Enterprise-like local CI/CD • Hardened workloads • Metrics + Logs • Runbooks + Rollback

Document version: v1.0

Last updated: 2026-02-13

Table des matières

1 Scope (local-first on Kubernetes Docker Desktop)

In scope

- A complete local platform on Kubernetes (Docker Desktop) with:
- Frontend: React/Vite/Tailwind served by Nginx (non-root)
- API: FastAPI with /health, /ready, /metrics, JSON structured logs
- Worker: Celery + Redis broker, writes stats/KPIs to Postgres
- Data: Postgres (stateful), Redis (deployment)
- Edge: Ingress-NGINX + self-signed TLS
- Observability: kube-prometheus-stack + Loki/Promtail
- Ops: Makefile + scripts executed inside devops-toolbox, plus rollback + runbook

Out of scope (intentional)

- Cloud-managed services (EKS/GKE/AKS), multi-cluster, multi-region HA
- External DNS, public certificates, production identity providers (OIDC)
- Full supply-chain hardening (signing/attestations) — can be a next step
- Real SLO enforcement across months (this is a portfolio environment)

Assumptions

- You run everything from a reproducible toolbox container ("toolbox-only").
- You accept self-signed TLS for local domains.
- Docker Desktop is the target cluster runtime (not Minikube/kind).

1.1 Executive Summary

1.2 Project goal (what it demonstrates)

OpsDesk is a **local-first** platform deployed on **Kubernetes (Docker Desktop)** designed to simulate an "enterprise-style" internal Ops portal with **real operational constraints:**

- **Delivery:** build → test → scan → validate → deploy → smoke, driven by a local pipeline
- **Reliability:** clear health/readiness signals, safe rollouts, rollback path (N-1)
- **Security posture:** hardened workloads (non-root, no privilege escalation, read-only filesystem, seccomp), least-privilege access (RBAC + dedicated SAs)
- **Observability:** metrics + dashboards + alert rules + centralized logs (JSON parsing + labels)
- **Day-2 operations:** triage workflow using k9s + Grafana/Loki and a documented incident scenario

This report documents the architecture, **implementation** choices, and evidence that these requirements are met.

1.3 Final result (what runs, what is observable)

What's running (platform)

- **Frontend:** React/Vite/Tailwind built assets served by Nginx (non-root)
- **API:** FastAPI with:
  - /health liveness signal
  - /ready readiness signal (dependencies checked)
  - /metrics for Prometheus scraping
  - JSON structured logs (request context)
- **Worker: Celery** (Redis broker), asynchronous processing + stats/KPI-like writes
- **Data:** Postgres (stateful persistence) + Redis (deployment)

What's running (platform infrastructure)

- **Ingress-NGINX** as the entrypoint with **self-signed TLS**
- **Kustomize** packaging: base + overlays (dev and prod-like)
- **Observability stack**:
  - Prometheus + Grafana + Alertmanager (kube-prometheus-stack)
  - Loki + Promtail for log aggregation and JSON parsing

Observable signals you can demo

- **Golden signals** for the API (requests, errors, latency) via dashboards
- **Targets up** in Prometheus (API /metrics, cluster components)
- **Centralized logs** in Loki, queryable by labels (service/env/version)
- **Operational health** in Kubernetes: rollouts, restarts, resource usage, HPA/PDB (prod-like)

Local domains used for the demo :

opsdesk.local

api.opsdesk.local

grafana.opsdesk.local

1.4 Portfolio value (skills proven)

This project is intentionally built to be **auditable** (commands + expected outputs) and demoable). It proves capability across:

- **Kubernetes packaging:** clean separation between base and overlays, repeatable rendering/validation
- **Runtime securit**y: hardened pod security context applied consistently across services
- **DevSecOps workflow:** image scanning (Trivy) as a gate before deployment
- **Observability:** metrics + logs integrated into a single troubleshooting path
- **Operational discipline:** runbook, incident scenario, rollback N-1, smoke tests
- **Reproducibility:** toolbox-only execution model (same tools, same versions, same flow)

1.5 Local-first context (why Docker Desktop + toolbox)

OpsDesk is **local-first by design** to remove environment friction and maximize reproducibility:

- **Docker Desktop Kubernetes** provides a consistent local cluster experience (good for demos and iteration).
- A dedicated **devops-toolbox container** ensures the exact same CLI toolchain every time (kubectl/helm/kustomize/k9s/trivy/curl/jq/yq), avoiding "works on my machine" drift.
- **Self-signed TLS** simulates production-like ingress behavior (HTTPS routing, cert handling) without external dependencies.
- The architecture and practices are **cloud-transferable**: overlays, hardened pods, RBAC, observability patterns, CI/CD gates, rollback flow.

1.6 Section 1 — Evidence

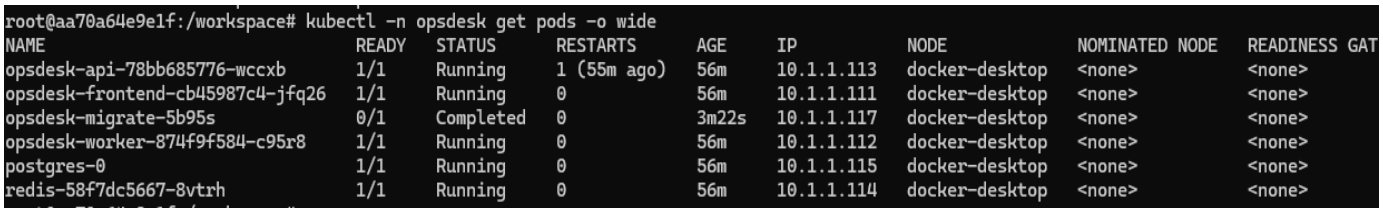
- **Evidence — local CI pipeline completes (dev)**

```
make ci ENV=dev
```

- **Expected**

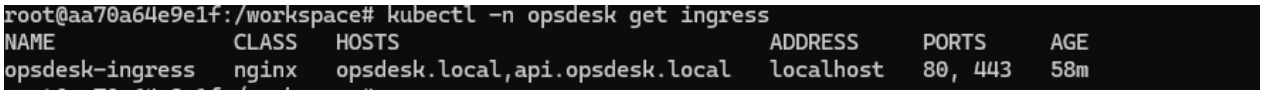
```
==> Build ...
==> Scan ...
==> Deploy ...
==> Smoke ...
Smoke OK
```

- **Evidence — all workloads are Ready**



```
root@aa70a64e9e1f:/workspace# kubectl -n opsdesk get pods -o wide
NAME                              READY   STATUS      RESTARTS      AGE     IP           NODE             NOMINATED NODE   READINESS GAT
opsdesk-api-78bb685776-wccxb      1/1     Running     1 (55m ago)   56m     10.1.1.113   docker-desktop   <none>           <none>
opsdesk-frontend-cb45987c4-jfq26  1/1     Running     0             56m     10.1.1.111   docker-desktop   <none>           <none>
opsdesk-migrate-5b95s             0/1     Completed   0             3m22s   10.1.1.117   docker-desktop   <none>           <none>
opsdesk-worker-874f9f584-c95r8    1/1     Running     0             56m     10.1.1.112   docker-desktop   <none>           <none>
postgres-0                        1/1     Running     0             56m     10.1.1.115   docker-desktop   <none>           <none>
redis-58f7dc5667-8vtrh            1/1     Running     0             56m     10.1.1.114   docker-desktop   <none>           <none>
```

- **Evidence — ingress exists**



```
root@aa70a64e9e1f:/workspace# kubectl -n opsdesk get ingress
NAME              CLASS   HOSTS                          ADDRESS     PORTS     AGE
opsdesk-ingress   nginx   opsdesk.local,api.opsdesk.local   localhost   80, 443   58m
```

- **Evidence — API health + readiness**

curl -sk https://api.opsdesk.local/health	root@aa70a64e9e1f:/workspace# curl -sk https://api.opsdesk.local/heal {"status":"ok"}root@aa70a64e9e1f:/workspace#
curl -sk https://api.opsdesk.local/ready	root@aa70a64e9e1f:/workspace# curl -sk https://api.opsdesk.local/ready {"ready":true}root@aa70a64e9e1f:/workspace#

2 Section 2 — Context & Requirements

2.1 Simulated "enterprise" problem (Ops portal)

OpsDesk is a **local-first internal operations portal** designed to mimic a real company scenario:

- Multiple services owned by different "teams" (UI, API, worker, data layer)
- A need for **consistent delivery** (repeatable deploys, controlled rollouts)
- A need for **operational confidence** (health/readiness, logs, metrics, alerting)
- A need for **safe changes** (security posture, resource limits, rollback plan)

What this simulates in practice

- A lightweight "change tracking / ops dashboard" where:
  - the **Frontend** exposes a single-entry point for operators,
  - the **API** offers CRUD + ops endpoints,
  - the **Worker** processes async tasks (stats/KPIs) decoupled from request latency,
  - **Postgres** persists the system of record,
  - **Redis** acts as broker for async processing.

2.2 Non-functional constraints (security, observability, rollback)

This project was intentionally driven by non-functional requirements (SRE/DevOps posture), not just features.

Security constraints (runtime + Kubernetes)

- Containers must run **non-root** and follow a “least privilege” runtime posture.
- Pods must enforce:
  - allowPrivilegeEscalation=false
  - capabilities.drop=["ALL"]
  - readOnlyRootFilesystem=true (with explicit writable paths only when needed)
  - seccompProfile=RuntimeDefault
- Service Accounts must be scoped and hardened:
  - dedicated service accounts per component
  - automountServiceAccountToken=false unless explicitly required
- Resources must be bounded:
  - CPU/RAM **requests + limits** to prevent noisy-neighbor effects and OOM loops.

Observability constraints (debuggable by design)

- Every component must produce:
  - **metrics** (Prometheus scraping) for service health / latency / errors
  - **structured logs** (JSON) so logs are queryable (service/env/version/request_id)
- A working workflow must exist:
  - **metrics → logs → mitigation** using Grafana + Loki (and k9s for cluster state)

Rollback constraints (safe delivery)

- A rollback must be:
  - **one command** (operator-friendly)
  - validated by rollout status + smoke checks
  - compatible with Kubernetes native rollout mechanisms

2.3 Architecture choices (services, async decoupling)

The architecture is intentionally modular and ops-friendly:

- **Split UI and API:**
  - front-end can evolve independently (static delivery via NGINX)
  - API remains the single source of truth for data + ops endpoints
- **Async worker with a queue:**
  - prevents long-running jobs from impacting API response times
  - enables retries and independent scaling behavior
- **Explicit data layer:**
  - Postgres stateful persistence
  - Redis for broker (and optional caching patterns if extended later)

Key design outcomes

- Predictable rollouts (independent deployments)
- Clear blast radius (worker failures don't break UI/API availability immediately)
- Observable behavior (each component emits logs/metrics for triage)

2.4 Success criteria (health/ready, SLO-like signals, pipeline OK)

OpsDesk is considered "done" when these criteria are met:

Platform availability (functional)

- Frontend reachable via Ingress and TLS
- API endpoints operational:
  - /health returns OK
  - /ready returns ready=true only when dependencies are available
- Minimal CRUD works end-to-end (create + list)

Operational readiness (SRE signals)

- Prometheus can scrape targets (API metrics + cluster components)
- Grafana dashboards show:
  - CPU/RAM trends
  - pod availability / restarts
  - logs volume and searchable log streams
- Loki queries can isolate:
  - a namespace (opsdesk)
  - a workload/pod
  - a request path or correlation field (when present)
  -

Delivery readiness (pipeline)

- Local CI/CD is reproducible via the toolbox:
  - lint + tests + image build + scan + deploy + smoke
- A rollback can be executed and validated:
  - rollout undo performed for the deployment
  - rollout status confirms stability

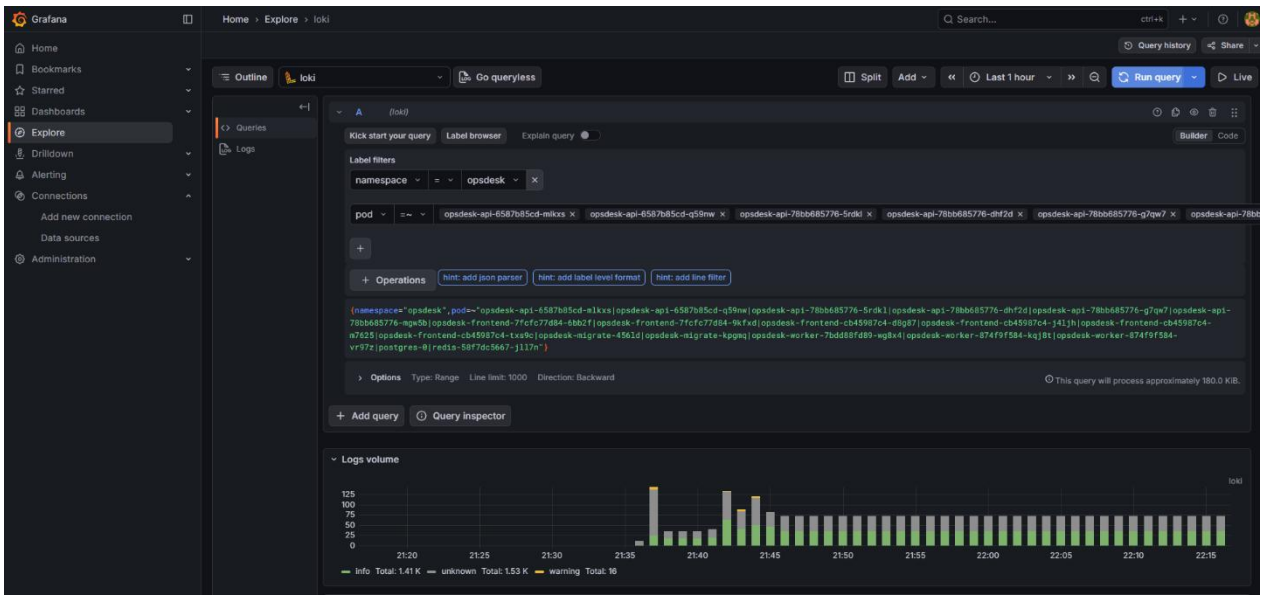
2.5 Section 2 — Evidence

2.5.1 Rollback is real

```
root@442b365d13fa:/workspace# make rollback

==> Rollback (rollout undo)
deployment.apps/opsdesk-api rolled back
deployment.apps/opsdesk-frontend rolled back
deployment.apps/opsdesk-worker rolled back

==> Wait after rollback
Waiting for deployment "opsdesk-api" rollout to finish: 0 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
deployment "opsdesk-api" successfully rolled out
deployment "opsdesk-frontend" successfully rolled out
deployment "opsdesk-worker" successfully rolled out
```

```
root@442b365d13fa:/workspace# kubectl -n opsdesk rollout history deploy/opsdesk-api
kubectl -n opsdesk rollout status deploy/opsdesk-api

deployment.apps/opsdesk-api
REVISION  CHANGE-CAUSE
2         <none>
3         <none>

deployment "opsdesk-api" successfully rolled out
```

2.5.2 Observability exists (logs queryable)



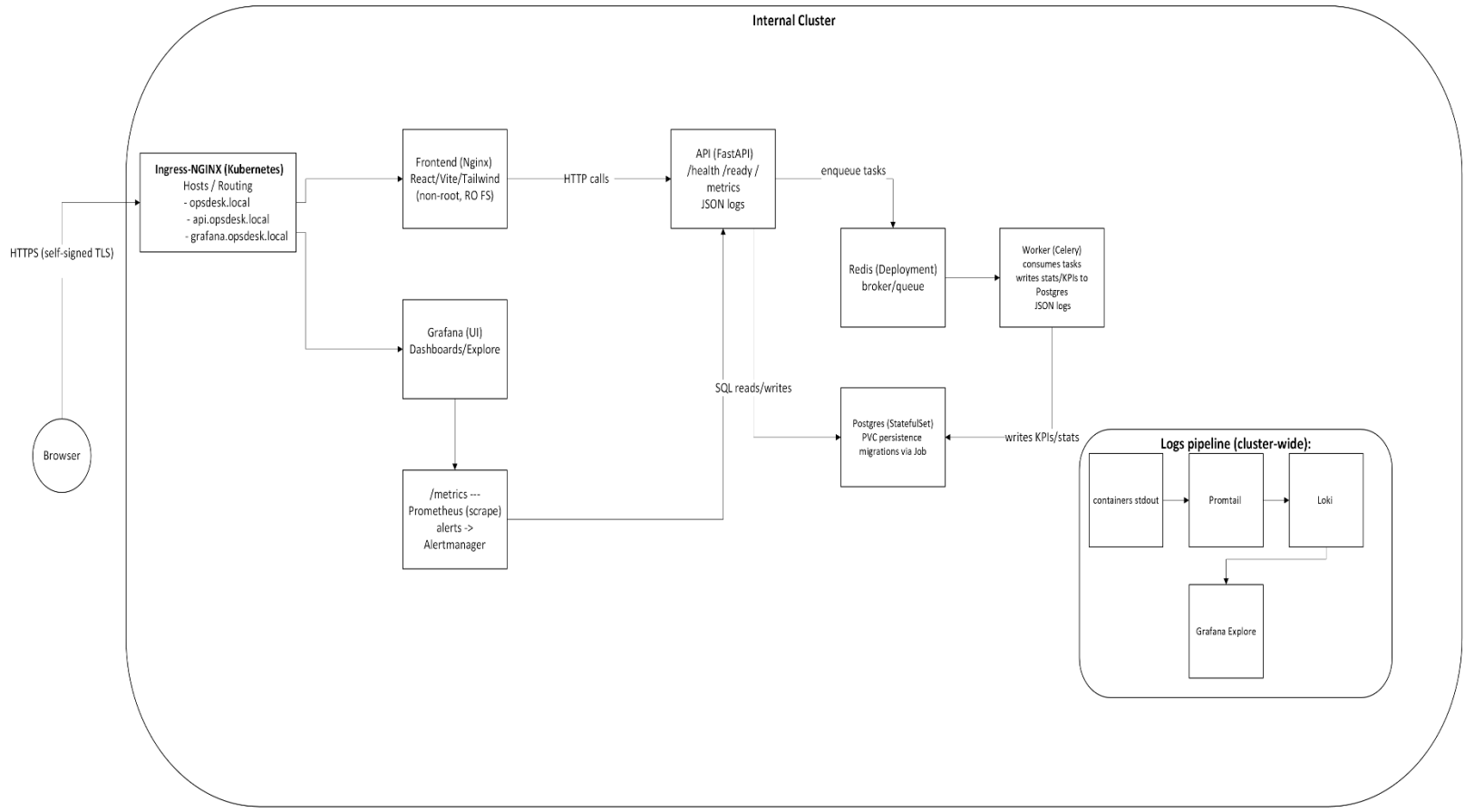
3 Section 3 — Global Architecture

3.1 Overview

OpsDesk is deployed local-first on Kubernetes (Docker Desktop) with a clear split between:

- Synchronous traffic (UI → API → DB) for user interactions
- Asynchronous processing (API → queue → worker → DB) for background jobs (stats/KPIs)
- Observability plane (metrics/logs/alerts) to support an SRE-style workflow

3.2 OpsDesk Global Architecture (local-first Kubernetes)



Global architecture (local-first on Kubernetes Docker Desktop). Ingress-NGINX terminates TLS and routes traffic to Frontend, API, and Grafana. The API persists data to Postgres and enqueues background tasks to Redis. Celery workers consume tasks and write computed KPIs back to Postgres. Prometheus scrapes /metrics and Grafana provides dashboards and Loki log exploration via Promtail.

3.3 Main flows (UI → API → DB / UI → API → queue → worker)

3.3.1 Flow A — Synchronous request path (user-facing)

1. User accesses the portal via the Frontend through Ingress.
2. The Frontend performs HTTP calls to the API (CRUD + ops endpoints).
3. The API reads/writes data in Postgres (stateful persistence).
4. The API returns a response to the UI.

Intent: keep end-user requests fast and predictable, with Postgres as the system of record.

3.3.2 Flow B — Asynchronous processing path (background jobs)

1. The UI triggers an API action that produces a job (e.g., compute stats/KPIs).
2. The API enqueues a task into Redis (broker/queue).
3. The Worker (Celery) consumes the task from Redis.
4. The Worker performs computation and persists results to Postgres.
5. Logs and metrics are emitted for traceability and triage.

Intent: decouple long-running or compute-heavy work from the API request lifecycle.

3.4 Observability flows (metrics / logs / alerts)

OpsDesk is designed to be **observable by default**, enabling a standard workflow:

symptom → metrics → logs → mitigation → verification.

Metrics (Prometheus)

- The API exposes:
  - /metrics for Prometheus scraping
  - /health and /ready for availability and dependency readiness
- Prometheus scrapes:
  - OpsDesk API metrics
  - Kubernetes and Ingress metrics (cluster health)
- Grafana dashboards visualize:
  - CPU/RAM usage (pods and workloads)
  - restarts / availability
  - service-level signals (errors/latency where applicable)

Logs (Loki via Promtail)

- Services log to stdout; Promtail ships logs to Loki.
- Logs are queryable in Grafana Explore using consistent labels (examples):
  - namespace="opsdesk"
  - pod="<pod-name>"
  - service, env, version (when present)
- JSON-structured logs allow filtering by fields such as:
  - request_id
  - status_code
  - path
  - latency_ms

Alerts (Alertmanager)

Alert rules focus on operational signals:

- service availability (target down)
- elevated error rates / restarts
- resource pressure (CPU/RAM saturation, OOM patterns)

3.5 CI/CD flow (build → scan → deploy → smoke → rollback)

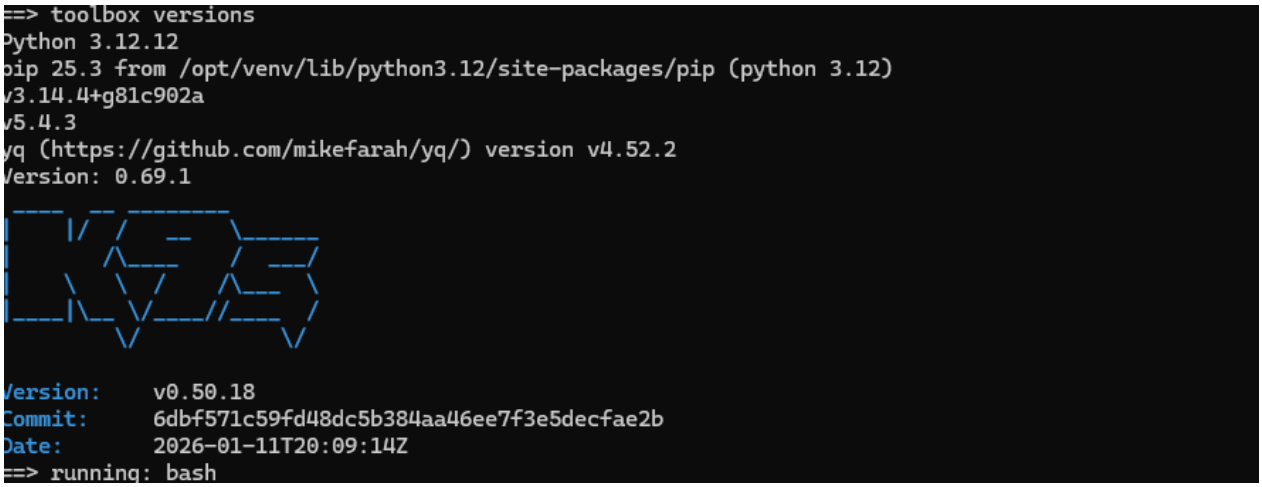
OpsDesk uses a **local-first, enterprise-style pipeline** executed from a controlled environment (**devops-toolbox**). The goal is reproducibility and safe delivery with clear gates.

Pipeline stages

1. **Lint** (Python + frontend)
2. **Tests** (API tests)
3. **Build** (versioned images)
4. **Scan** (Trivy gate)
5. **Render & validate** (Kustomize + client-side validation)
6. **Deploy & wait** (rollout gates + migration job)
7. **Smoke tests** (API checks + minimal CRUD)
8. **Rollback N-1** (native rollout undo + verification)

Why this matters

- Repeatable execution environment (no “works on my machine”)
- Clear quality gates before deployment
- Operator-friendly rollback under pressure



3.6 Naming & local domains (no-links policy)

OpsDesk uses stable local domains for operator usability and deterministic routing.

- opsdesk.local
- api.opsdesk.local
- grafana.opsdesk.local

Routing convention

- opsdesk.local → Frontend (UI)
- api.opsdesk.local → API (including /health, /ready, /metrics)
- grafana.opsdesk.local → Grafana UI (dashboards + Loki Explore)

Policy note

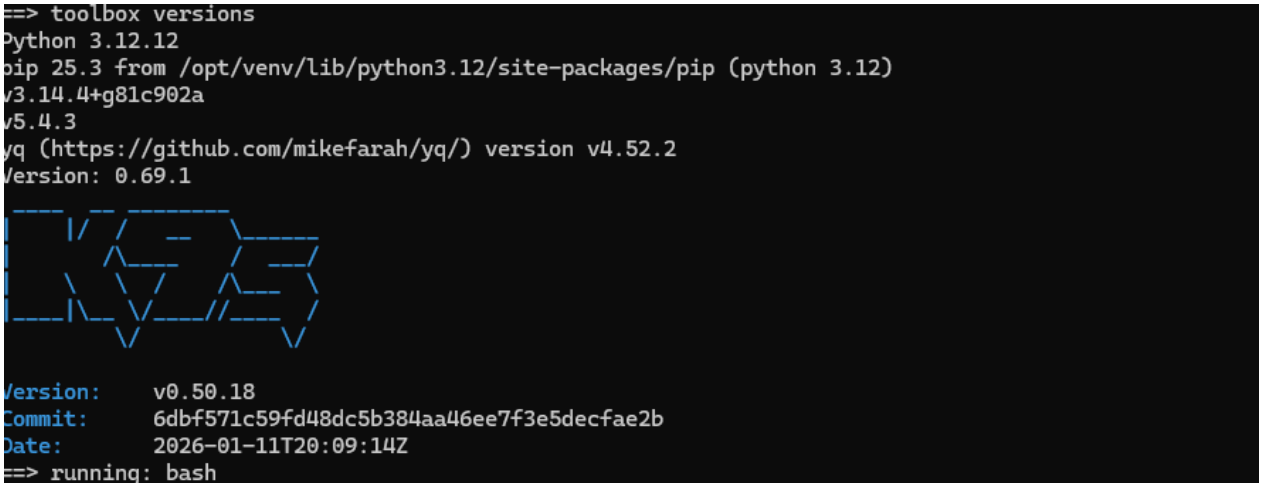
- Hostnames are shown in code blocks (no clickable links).
- Evidence commands are kept short with short expected outputs.

3.7 Section 3 — Evidence

3.7.1 Evidence A — Ingress routing exists

```
root@aa70a64e9e1f:/workspace# kubectl -n opsdesk get ingress
NAME              CLASS   HOSTS                          ADDRESS     PORTS     AGE
opsdesk-ingress   nginx   opsdesk.local,api.opsdesk.local   localhost   80, 443   58m
```

3.7.2 Evidence B — Reproducible toolbox environment



3.7.3 Evidence C — Rollout stability after deploy

```
root@d107a04355e3:/workspace# make rollback

==> Rollback (rollout undo)
deployment.apps/opsdesk-api rolled back
deployment.apps/opsdesk-frontend rolled back
deployment.apps/opsdesk-worker rolled back

==> Wait after rollback
Waiting for deployment "opsdesk-api" rollout to finish: 0 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 out of 2 new replicas have been updated...
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
deployment "opsdesk-api" successfully rolled out
deployment "opsdesk-frontend" successfully rolled out
deployment "opsdesk-worker" successfully rolled out
root@d107a04355e3:/workspace#
```

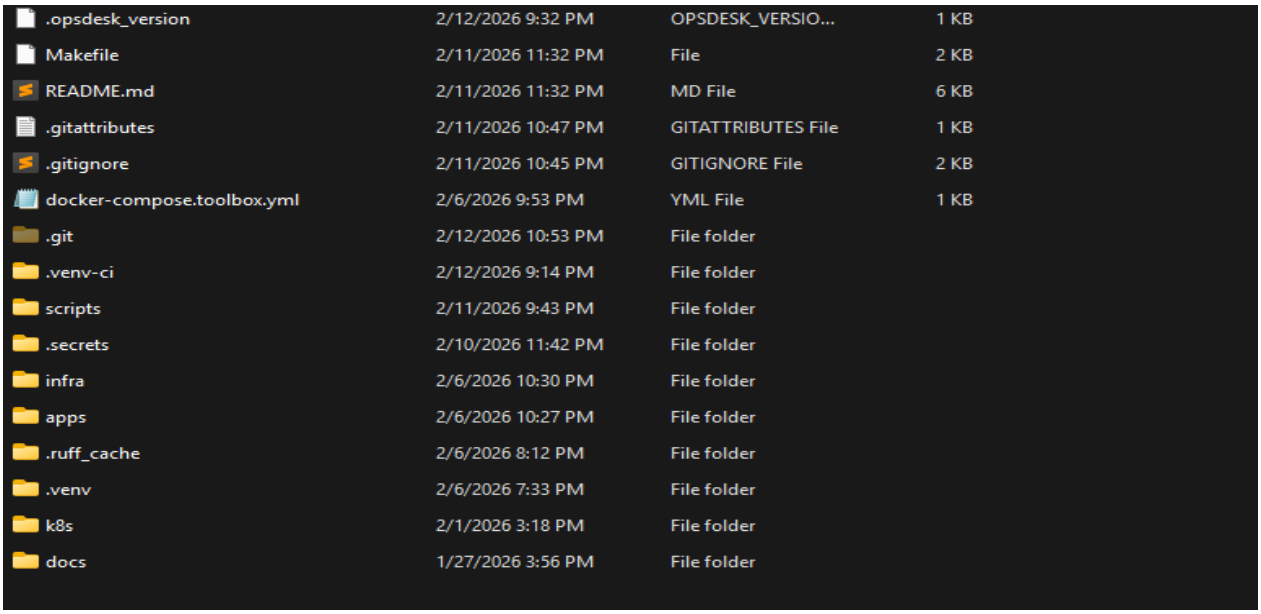
```
root@442b365d13fa:/workspace# kubectl -n opsdesk rollout history deploy/opsdesk-api
kubectl -n opsdesk rollout status deploy/opsdesk-api

deployment.apps/opsdesk-api
REVISION  CHANGE-CAUSE
2         <none>
3         <none>

deployment "opsdesk-api" successfully rolled out
```

4 Section 4 — Repository & Standards

4.1 Repository structure (root layout + responsibilities)

OpsDesk is structured to separate application code, Kubernetes delivery artifacts, and operational tooling. The layout supports a “real team” workflow: build/test/scan/deploy/rollback from the repo root, using a consistent toolbox environment.

- .opsdesk_version # single source of truth for build/deploy version tag
- Makefile # local enterprise-like pipeline entrypoint (ci/cd/smoke/rollback)
- README.md # project overview + quickstart + evidence
- .gitattributes # repo hygiene (line endings, etc.)
- .gitignore # ignore secrets, caches, local envs, build outputs
- docker-compose.toolbox.yml# toolbox runner (reproducible CLI env)
- scripts/ # CI/CD + ops scripts (lint/test/build/scan/deploy/smoke/rollback/secrets)
- infra/ # platform tooling (devops-toolbox) + helm values (observability)
- apps/ # application services (api/worker/frontend)
- k8s/ # Kubernetes manifests (base + overlays dev/prod-like)
- docs/ # runbook
- .secrets/ # local-only secret material (never committed)
- .venv/ # optional local venv (not required when using toolbox)
- .venv-ci/ # optional venv for CI-like local runs
- .ruff_cache/ # lint cache (ignored)
- .git/ # git history (not relevant for the report)



Key intent behind this layout

- apps/ = feature development surface (what you ship)
- k8s/ = platform declaration (how it runs on Kubernetes)
- scripts/ + Makefile = operations & delivery (how you run and prove it)
- infra/ = reproducible tooling + observability configuration
- .secrets/ = strictly local and excluded from version control

4.2 Versioning convention (git → .opsdesk_version → deploy)

OpsDesk uses a **single version string** (stored in .opsdesk_version) to drive:

- image tags for opsdesk/api, opsdesk/worker, opsdesk/frontend
- deployment output visibility (e.g., "Deploy ENV=prod-like VERSION=...")
- observability correlation (when version is present in logs/labels)

Why this matters

- deterministic rollbacks (you always know what version is running)
- consistent evidence capture (screenshots tie to a specific build/deploy)
- reduces "what did I deploy?" ambiguity during incident drills

4.3 Image naming convention (frontend/api/worker)

Images are named consistently so Kustomize overlays and the pipeline can inject tags reliably.

Standard

- opsdesk/frontend:<version>
- opsdesk/api:<version>
- opsdesk/worker:<version>

Why this matters

- predictable kustomize image injection
- scanning (trivy) is consistent per service
- rollback is service-aligned (API/worker/frontend move together)

4.4 JSON logs convention (service/env/version/request_id)

Logs are structured to support Loki filtering and “SRE-style” triage. The convention is to emit JSON (or JSON-like structured logs) with stable fields.

Minimum fields (target standard)

- timestamp (or asctime)
- level / levelname
- service (api/worker/frontend)
- env (dev / prod-like)
- version (same as image tag)
- message
- API: request_id, path, method, status_code, latency_ms
- Worker: taskName/job_id, duration_ms, status

4.5 “Toolbox only” convention (reproducible environment)

All operations are meant to run from the devops-toolbox container, invoked via **docker-compose.toolbox.yml**. This guarantees:

- pinned tool versions (kubectl/helm/kustomize/trivy/k9s)
- consistent shell environment
- consistent kubeconfig usage
- reduced Windows/WSL drift

What lives where

- docker-compose.toolbox.yml defines the reproducible environment
- scripts/ contains all operational logic
- Makefile provides a clean UX (single entrypoints)

5 Section 5 — Runtime Environment (Local-first Platform)

5.1 Kubernetes on Docker Desktop: what changes vs "real cloud"

OpsDesk runs on a single-node Kubernetes cluster provided by Docker Desktop. This environment is intentionally chosen to be **local-first** and **reproducible**, while still practicing "production posture" patterns (probes, RBAC, resource limits, observability, rollbacks).

What is similar to a cloud cluster

- Same Kubernetes primitives: Deployment, StatefulSet, Service, Ingress, HPA, PDB, RBAC, Secrets/ConfigMaps.
- Same operational workflows: rollouts, rollbacks, debugging with kubectl and k9s.
- Same observability stack patterns: Prometheus scraping /metrics, Grafana dashboards, Loki log aggregation.

What is different

- Single node / limited HA: node failure is out of scope; HA is emulated via replicas and disruption budgets.
- Storage class behavior: Docker Desktop typically uses local/hostpath-style storage; PVC behavior is correct for learning but not identical to managed cloud storage.
- Ingress and LoadBalancer: external IPs are usually mapped to localhost; access is done through local hostnames + port mapping.
- TLS trust: self-signed certs are used; browsers require explicit trust acceptance.

5.2 devops-toolbox: role, tools, volumes, KUBECONFIG

OpsDesk operations are executed through a dedicated container: devops-toolbox. The goal is to ensure “works on my machine” becomes “works on the toolbox”.

5.2.1 Toolbox responsibilities

- Provide a pinned toolchain: kubectl, helm, kustomize, k9s, trivy, jq, yq, curl, openssl, bash.
- Run the local pipeline (make ci, make cd, make rollback) with consistent dependencies.
- Avoid OS drift (Windows/WSL PATH differences, inconsistent binary versions, missing tools).

5.2.2 How it works

- The repo is mounted into /workspace inside the container.
- The Kubernetes configuration is mounted read-only and exposed via KUBECONFIG.
- Docker socket access is provided so the toolbox can build and scan images.

5.3 Hostname mapping (Windows hosts) + browser access

To keep local access clean and “enterprise-like”, OpsDesk uses stable local hostnames routed through Ingress-NGINX.

Hostnames policy

- No clickable links in documentation.
- **Hostnames are always written in code blocks.**

Windows hosts entries

- 127.0.0.1 opsdesk.local
- 127.0.0.1 api.opsdesk.local
- 127.0.0.1 grafana.opsdesk.local

What each hostname serves

- opsdesk.local → Frontend (Nginx serving React build)
- api.opsdesk.local → API (FastAPI)
- grafana.opsdesk.local → Grafana UI (dashboards + Explore)

5.4 Self-signed TLS: secret, SANs, rotation (local scope)

OpsDesk terminates HTTPS at Ingress-NGINX with a self-signed TLS certificate. The objective is to validate the full path:

- HTTPS routing
- Ingress configuration
- application readiness behind TLS
- smoke tests that tolerate self-signed certs

5.4.1 TLS behavior

- The certificate is stored as a Kubernetes TLS secret (type kubernetes.io/tls).
- Hostnames are included as SANs (so the browser and curl match the expected DNS names).
- For command-line checks, curl is executed with -k because the certificate is not chained to a trusted CA.

Evidence

Rotation policy (local)

- Certificate rotation is intentionally simple: regenerate the secret and reapply manifests.
- In “next steps”, this can evolve toward a local certificate lifecycle (e.g., cert-manager) for a more cloud-like flow.

6 Section 6 — Kubernetes Foundation (Cluster Prerequisites)

6.1 Namespaces (separation of concerns)

OpsDesk is deployed with clear namespace boundaries so platform components, observability, and the application stack remain isolated.

Namespaces used

- opsdesk — application workloads (frontend, API, worker, Postgres, Redis, migrations job)
- ingress-nginx — Ingress controller components
- observability — monitoring + logging stack (Prometheus/Grafana/Alertmanager + Loki/Promtail)

Why it matters

- easier RBAC scoping
- cleaner troubleshooting (k9s :ns)
- avoids “everything in default” anti-pattern

6.2 Ingress-NGINX: role, routing, key behavior

Ingress-NGINX provides a single entry point for HTTP/HTTPS into the cluster, routing traffic by hostname and path to the right Kubernetes Services.

Responsibilities

- TLS termination (self-signed cert)
- host-based routing:
  - opsdesk.local → frontend service
  - api.opsdesk.local → api service
  - grafana.opsdesk.local → grafana service (from observability namespace)
- consistent “prod-like” ingress behavior locally

Typical routing model

- browser → Ingress controller → Service → Pod

Operational note

In Docker Desktop, the Ingress address often shows as localhost. That’s expected in a local-first setup.

6.3 metrics-server: role (HPA / kubectl top) + Docker Desktop considerations

OpsDesk uses Horizontal Pod Autoscaling (HPA) in the prod-like overlay, which requires metrics availability.

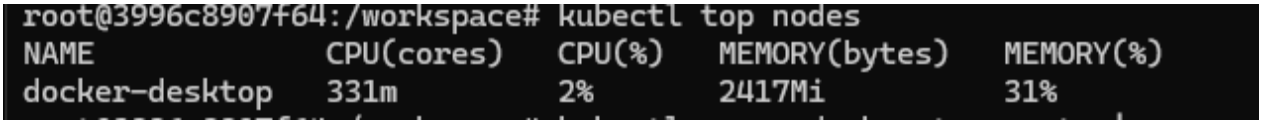
Why metrics-server matters

- Enables:
  - kubectl top pods/nodes
  - HPA target evaluation (CPU/memory utilization)
- Without it, HPAs remain stuck / do not scale as expected

Docker Desktop considerations

- metrics-server typically works, but local clusters sometimes need minor flags/patches (depending on environment) for TLS/metrics scraping.
- This project treats metrics-server as a required foundation component because HPA is part of the "prod-like posture".

Evidence



6.4 Secrets & ConfigMaps strategy (dotenv → secret)

OpsDesk uses Kubernetes-native configuration:

- ConfigMaps for non-sensitive runtime settings
- Secrets for credentials and connection strings (local scope)

Local-first approach

- .secrets/ stores local material only (never committed)
- A script generates/applies a Kubernetes secret (opsdesk-secrets) in the opsdesk namespace
- The pipeline ensures secrets are present before deployments roll out

Why it matters

- avoids hardcoding credentials into manifests
- reproducible "new machine bootstrap"
- supports a real delivery flow (secrets are applied as part of deployment gates)

6.5 Stateful storage (Postgres): PVC, persistence, limits

Postgres is deployed as a StatefulSet to guarantee:

- stable identity (postgres-0)
- persistent storage via PVC
- predictable data retention across pod restarts

Why this matters (even locally)

- demonstrates correct handling of stateful workloads
- aligns with real-world K8s patterns (PVC + stateful identity)
- supports migrations and application bootstrapping

Docker Desktop storage behavior

- PVCs are usually backed by a local storage provisioner; this is sufficient to demonstrate persistence and operational workflows.

6.6 Database migrations (Job-based execution)

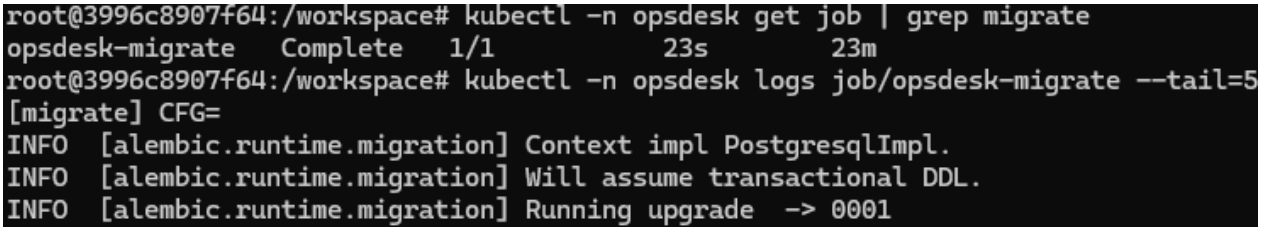
OpsDesk runs schema migrations as a dedicated Kubernetes Job (Alembic). This keeps migrations:

- explicit
- auditable
- separable from application runtime

Why it matters

- prevents “migrations on startup” surprises
- makes deployment order deterministic
- provides clean evidence in pipeline logs

6.6.1 Evidence



7 Section 7 — Application Components (Deep Dive)

This section describes each application component as a product-like service: responsibility, build/runtime model, Kubernetes objects, security posture, and short "evidence" commands with expected outputs.

7.1 Frontend — React + Vite + Tailwind served by Nginx

7.1.1 Responsibilities & functional scope

The frontend provides the OpsDesk UI:

- human-friendly entry point to the platform
- calls the API for data (CRUD "changes", stats, health indicators)
- acts as the "Ops portal" layer (navigation, dashboards links, operational UX patterns)

7.1.2 Build & packaging model (Vite build → static assets)

- **Build stage:** Vite compiles the SPA into static assets.
- **Runtime stage**: Nginx serves static content only (no Node.js in runtime).
- This split reduces runtime attack surface and keeps the container lean.

7.1.3 Runtime configuration

Typical runtime concerns (implemented as config/env driven, not hardcoded):

- API base URL (targets api.opsdesk.local)
- headers / timeouts / error handling (client-side)
- health indicator UI behavior (optional)

7.1.4 Kubernetes objects

Frontend is deployed with:

- Deployment (replicas controlled via overlay)
- Service (ClusterIP)
- Ingress rule for host routing (opsdesk.local)

7.1.5 Runtime security posture

Frontend runtime is hardened:

- Nginx container runs non-root
- readOnlyRootFilesystem: true
- allowPrivilegeEscalation: false
- capabilities.drop: ["ALL"]
- seccompProfile: RuntimeDefault

This is intentionally “prod posture” even in local-first mode.

7.2 API — FastAPI + OpenAPI + /health /ready /metrics + JSON logs

7.2.1 Responsibilities & endpoints

The API is the operational core:

- CRUD endpoints (example: /api/changes)
- operational endpoints:
  - /health — process-level health (service up)
  - /ready — dependency readiness (Postgres/Redis reachable)
  - /metrics — Prometheus exposition format

7.2.2 readiness design (dependency checks)

Readiness is designed to catch “looks up but not usable” states:

- Postgres connectivity (DB ready)
- Redis connectivity (broker ready)
- optional: migrations completed / schema version acceptable (if implemented)

This supports:

- safe rollouts (traffic only when ready)
- incident diagnosis (ready flips when dependency fails)

7.2.3 Metrics (Prometheus)

The API exposes metrics for:

- request rate / status codes (2xx/4xx/5xx)
- latency (p95/p99 if histogram)
- dependency error counts (if instrumented)
- app metadata labels (service, env, version when available)

7.2.4 structured JSON logs

API logs are designed for Loki searchability:

- JSON format (machine-parsable)
- key fields (typical):
  - service, env, version
  - request_id
  - method, path
  - status_code
  - duration_ms (or equivalent)

This enables “investigate by request_id” workflows.

7.2.5 Kubernetes objects

API is deployed with:

- Deployment
- Service (ClusterIP)
- Ingress routing for host (api.opsdesk.local)
- probes configured (/health and /ready)

7.2.6 Config & secrets

Sensitive configuration is injected via Kubernetes Secret:

- Postgres connection (URL/user/pass)
- Redis URL
- any API secrets (JWT, signing keys if present)

Non-sensitive settings via ConfigMap:

- log level
- environment name (dev/prod-like)
- feature flags (if any)

7.3 Worker — Python Celery + Redis broker (async jobs / stats)

7.3.1 Responsibilities (jobs, stats, KPIs)

The worker executes background tasks asynchronously:

- consumes tasks from Redis queue
- computes stats / KPIs
- writes results to Postgres (or updates API-visible data)
- produces structured logs for traceability

This models real “enterprise” async decoupling:

- API stays responsive
- background workloads are isolated and scalable independently

7.3.2 Queue model (Redis broker)

- Redis acts as broker/queue backend for Celery.
- Tasks are enqueued by the API (or by scheduled jobs).
- Worker consumes from the configured queue(s)

7.3.3 Concurrency / pool tuning

Worker behavior is controlled via env vars:

- concurrency value
- prefetch / acks / retries (if implemented)
- resource limits ensure no “runaway worker” on local machine

7.3.4 Idempotency / retries (if implemented)

Production-style design goals (even locally):

- retries for transient failures (DB/Redis)
- idempotency to avoid duplicates (based on task id / change id)
- clear log fields per execution (job_id, duration, status)

7.3.5 JSON logs

Worker logs include:

- service identity
- task name
- task lifecycle events (start/end)
- errors with context

7.3.6 Kubernetes objects

Worker is deployed as:

- Deployment
- resource requests/limits (important for OOM control)
- optional probes (depending on implementation)

7.3.7 Evidence

```
[2026-02-13 16:07:09,190: INFO/MainProcess] Connected to redis://redis.opsdesk.svc.cluster.local:6379/0
[2026-02-13 16:07:09,190: WARNING/MainProcess] /usr/local/lib/python3.12/site-packages/celery/worker/consum
 no longer determine
whether broker connection retries are made during startup in Celery 6.0 and above.
If you wish to retain the existing behavior for retrying connections on startup,
you should set broker_connection_retry_on_startup to True.
  warnings.warn(

[2026-02-13 16:07:09,198: INFO/MainProcess] mingle: searching for neighbors
[2026-02-13 16:07:10,208: INFO/MainProcess] mingle: all alone
[2026-02-13 16:07:10,264: INFO/MainProcess] celery@opsdesk-worker-874f9f584-bph4d ready.
root@3996c8907f64:/workspace#
```

8 Section 8 — Data Layer (Postgres + Redis)

This section covers the stateful and queue layers that make OpsDesk behave like a real internal platform: persistence, migrations, and async execution.

8.1 Postgres (Stateful)

8.1.1 Why Stateful + persistence

Postgres is the system of record for OpsDesk:

- stores domain data (e.g., “changes”)
- stores computed stats/KPIs written by the worker
- must survive pod restarts and rollouts (even in local-first)

A StatefulSet is used to guarantee:

- stable network identity (postgres-0)
- stable volume binding (PVC) across restarts
- predictable operational behavior (upgrade/rollback scenarios)

8.1.2 PVC / storage on Docker Desktop

Because this runs on Kubernetes Docker Desktop:

- persistence relies on the default local storage provisioning (hostpath-like behavior)
- volumes are “good enough” for portfolio + reproducible demos
- constraints are documented (not a cloud CSI)

8.1.3 Migrations (Job-based)

Schema migrations are executed as a Kubernetes Job, so the platform can:

- apply DB changes in a repeatable, observable way
- keep API startup clean (no “migrate on boot” surprises)
- provide explicit evidence that schema is up-to-date

8.1.4 Backup/restore (optional)

Local-first approach: backup/restore is treated as an optional runbook item (export/import) rather than an HA strategy.

8.1.5 Evidence

StatefulSet + pod:

```
root@3996c8907f64:/workspace# kubectl -n opsdesk get sts postgres
NAME       READY   AGE
postgres   1/1     18m
root@3996c8907f64:/workspace# kubectl -n opsdesk get pods postgres-0
NAME         READY   STATUS    RESTARTS   AGE
postgres-0   1/1     Running   0          18m
```

PVC exists (persistence proof):

```
root@3996c8907f64:/workspace# kubectl -n opsdesk get pvc
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
data-postgres-0   Bound    pvc-1e7ee02d-129d-4e31-aec6-f22a5ed71d13   2Gi        RWO            hostpath       <unset>                 19m
```

Migration logs (tail):

```
root@3996c8907f64:/workspace# kubectl -n opsdesk logs job/opsdesk-migrate --tail=20
[migrate] CFG=
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 0001
```

8.2 Redis (Deployment)

8.2.1 Role

Redis is the async backbone:

- Celery broker/queue for background jobs
- (optional) cache if you ever extend the API

8.2.2 Resources & HA (why simple in local)

For a local-first portfolio:

- a single Deployment is intentional (simplicity + reproducibility)
- requests/limits prevent Redis from starving the node
- resilience is demonstrated through rollback + incident workflow, not multi-node HA

8.2.3 Evidence

```
root@3996c8907f64:/workspace# kubectl -n opsdesk get deploy redis
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
redis   1/1     1            1           27m
root@3996c8907f64:/workspace# kubectl -n opsdesk get svc redis
NAME    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
redis   ClusterIP   10.102.144.176   <none>        6379/TCP   27m
root@3996c8907f64:/workspace#
```

Connectivity is indirectly proven via:

- API readiness (/ready is true)
- Worker logs “Connected to redis…

```
[2026-02-13 16:07:09,190: INFO/MainProcess] Connected to redis://redis.opsdesk.svc.cluster.local:6379/0
[2026-02-13 16:07:09,190: WARNING/MainProcess] /usr/local/lib/python3.12/site-packages/celery/worker/co
 no longer determine
whether broker connection retries are made during startup in Celery 6.0 and above.
If you wish to retain the existing behavior for retrying connections on startup,
you should set broker_connection_retry_on_startup to True.
  warnings.warn(

[2026-02-13 16:07:09,198: INFO/MainProcess] mingle: searching for neighbors
[2026-02-13 16:07:10,208: INFO/MainProcess] mingle: all alone
[2026-02-13 16:07:10,264: INFO/MainProcess] celery@opsdesk-worker-874f9f584-bph4d ready.
root@3996c8907f64:/workspace#
```

9 Section 9 — Deployment: Kustomize Base + Overlays (dev vs prod-like)

This section explains how OpsDesk is deployed consistently, while still supporting two modes:

- dev = fast iteration
- prod-like = production posture (replicas/HPA/PDB/rollout discipline)

9.1 Base: shared settings (the "contract")

The base/ defines what is common everywhere:

- common labels/annotations (including env, version where applicable)
- service accounts + RBAC + automountServiceAccountToken: false
- hardened securityContext defaults (non-root, drop caps, RO filesystem, seccomp)
- resource requests/limits baseline
- Services + Ingress objects + ConfigMaps/Secrets wiring
- Postgres StatefulSet + Redis Deployment (core dependencies)

Think of the base as the platform contract: every overlay must respect it.

9.2 dev overlay: goals

dev focuses on speed:

- minimal replicas (often 1)
- lightweight resources
- short feedback loop for make ci ENV=dev

9.3 prod-like overlay: goals

prod-like focuses on operational realism:

- API/frontend replicas > 1 (rollout safe)
- HPA enabled (CPU-based scaling)
- PDB enabled (disruption control)
- rollout strategy tuned (rolling updates)
- stricter resource governance and labels (env/version)

9.4 Exact differences (what changes)

Typical “prod-like” deltas vs dev:

- replicas: api=2, frontend=2 (example)
- HPA: enabled for API (min/max + target CPU)
- PDB: created for api/frontend/worker
- rollout strategy: tuned maxUnavailable/maxSurge
- tighter resource limits (prevent OOM / noisy neighbor issues)

9.5 Image injection + validation

Images are injected from the pipeline tag (ex: .opsdesk_version):

- ensures the cluster runs exactly what was built/scanned
- allows rollback to the previous revision safely

Validation gates:

- Kustomize render step
- client-side apply validation (--dry-run=client)

9.6 Evidence

Overlay effect: replicas visible

```
root@3996c8907f64:/workspace# kubectl -n opsdesk get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
opsdesk-api         2/2     2            2           37m
opsdesk-frontend    2/2     2            2           37m
opsdesk-worker      1/1     1            1           37m
redis               1/1     1            1           37m
```

Deploy prod-like (pipeline-driven):

```
==> Migrations OK

==> Wait for rollouts (api/frontend/worker)
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
deployment "opsdesk-api" successfully rolled out
deployment "opsdesk-frontend" successfully rolled out
deployment "opsdesk-worker" successfully rolled out

==> Deploy OK

==> Pods (opsdesk)
NAME                              READY   STATUS        RESTARTS      AGE   IP          NODE             NOMINATED NODE   READINESS GATES
opsdesk-api-6587b85cd-8jjfr       1/1     Running       0             37s   10.1.2.34   docker-desktop   <none>           <none>
opsdesk-api-6587b85cd-zrbsn       1/1     Running       0             77s   10.1.2.29   docker-desktop   <none>           <none>
opsdesk-api-78bb685776-45g9k      1/1     Terminating   1 (31m ago)   33m   10.1.2.22   docker-desktop   <none>           <none>
opsdesk-frontend-7fcfc77d84-67bzm 1/1     Running       0             76s   10.1.2.31   docker-desktop   <none>           <none>
opsdesk-frontend-7fcfc77d84-dg8kq 1/1     Running       0             37s   10.1.2.35   docker-desktop   <none>           <none>
opsdesk-frontend-cb45987c4-f4dq7  0/1     Completed     0             33m   10.1.2.23   docker-desktop   <none>           <none>
opsdesk-migrate-ljkpd             0/1     Completed     0             57s   10.1.2.33   docker-desktop   <none>           <none>
opsdesk-worker-7bdd88fd89-vshc4   1/1     Running       0             76s   10.1.2.32   docker-desktop   <none>           <none>
postgres-0                        1/1     Running       0             33m   10.1.2.26   docker-desktop   <none>           <none>
redis-58f7dc5667-tghz2            1/1     Running       0             33m   10.1.2.25   docker-desktop   <none>           <none>

==> Ingress
NAME             CLASS   HOSTS                          ADDRESS     PORTS     AGE
opsdesk-ingress  nginx   opsdesk.local,api.opsdesk.local  localhost   80, 443   33m

==> HTTP checks (TLS self-signed so -k)
{
  "status": "ok"
}
{
  "ready": true
}
[]

==> Smoke OK
make[1]: Leaving directory '/workspace'
```

HPA exists (prod-like posture):

```
root@3996c8907f64:/workspace# kubectl -n opsdesk get hpa
NAME          REFERENCE                 TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
opsdesk-api   Deployment/opsdesk-api    cpu: 2%/65%     2         5         2          39m
```

PDB exists:

```
root@3996c8907f64:/workspace# kubectl -n opsdesk get pdb
NAME                  MIN AVAILABLE   MAX UNAVAILABLE   ALLOWED DISRUPTIONS   AGE
pdb-opsdesk-api       1               N/A               1                     8m33s
pdb-opsdesk-frontend  1               N/A               1                     8m30s
pdb-opsdesk-worker    N/A             1                 1                     8m29s
```

Ingress hosts:

```
root@3996c8907f64:/workspace# kubectl -n opsdesk get ingress
NAME              CLASS   HOSTS                              ADDRESS     PORTS     AGE
opsdesk-ingress   nginx   opsdesk.local,api.opsdesk.local   localhost   80, 443   41m
```

10 Section 10 — Local CI/CD (Makefile + scripts)

This section is the “enterprise-like” delivery story: repeatable, gated, observable, rollbackable — all executed inside the devops-toolbox container.

10.1 Philosophy (enterprise-like, local-only)

Principles:

- **toolbox-only execution** = reproducibility across machines
- **gates** before deploy (lint/test/scan/validate)
- **immutable tags** (.opsdesk_version) for traceability
- **Kubernetes-first operations** (rollouts, jobs, health checks)
- **rollback N-1** as a first-class workflow

10.2 make lint

Runs code quality gates:

- Python: ruff (or equivalent)
- Frontend: ESLint (or equivalent

10.3 make test

Runs unit tests (pytest for API/worker).

10.4 make build

Builds container images for:

- opsdesk/frontend
- opsdesk/api
- opsdesk/worker

Tag is derived from .opsdesk_version, ensuring deploy references exact artifacts.

10.5 make scan (Trivy)

Security gate at image level:

- scans OS packages + language deps
- focuses on HIGH/CRITICAL
- supports “ignore unfixed” policy if configured

Evidence:

opsdesk/frontend:49d6b65-dirty (alpine 3.21.3)

Total: 17 (HIGH: 13, CRITICAL: 4)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
libcrypto3	CVE-2025-15467	CRITICAL	fixed	3.3.3-r0	3.3.6-r0	openssl: OpenSSL: Remote code execution or Denial of Service via oversized Initialization... https://avd.aquasec.com/nvd/cve-2025-15467
	CVE-2025-69419	HIGH				openssl: OpenSSL: Arbitrary code execution due to out-of-bounds write in PKCS#12 processing... https://avd.aquasec.com/nvd/cve-2025-69419
	CVE-2025-69421					openssl: OpenSSL: Denial of Service via malformed PKCS#12 file processing https://avd.aquasec.com/nvd/cve-2025-69421
libpng	CVE-2025-64720			1.6.47-r0	1.6.53-r0	libpng: LIBPNG buffer overflow https://avd.aquasec.com/nvd/cve-2025-64720
	CVE-2025-65018					libpng: LIBPNG heap buffer overflow https://avd.aquasec.com/nvd/cve-2025-65018
	CVE-2025-66293					libpng: LIBPNG out-of-bounds read in png_image_read_composite https://avd.aquasec.com/nvd/cve-2025-66293
	CVE-2026-22695				1.6.54-r0	libpng: libpng: Denial of service and information disclosure via heap buffer over-read... https://avd.aquasec.com/nvd/cve-2026-22695
	CVE-2026-22801					libpng: libpng: Information disclosure and denial of service via integer truncation in... https://avd.aquasec.com/nvd/cve-2026-22801
libssl3	CVE-2025-15467	CRITICAL		3.3.3-r0	3.3.6-r0	openssl: OpenSSL: Remote code execution or Denial of Service via oversized Initialization... https://avd.aquasec.com/nvd/cve-2025-15467
	CVE-2025-69419	HIGH				openssl: OpenSSL: Arbitrary code execution due to out-of-bounds write in PKCS#12 processing... https://avd.aquasec.com/nvd/cve-2025-69419
	CVE-2025-69421					openssl: OpenSSL: Denial of Service via malformed PKCS#12 file processing https://avd.aquasec.com/nvd/cve-2025-69421
libxml2	CVE-2025-49794	CRITICAL		2.13.4-r5	2.13.9-r0	libxml: Heap use after free (UAF) leads to Denial of service (DoS)... https://avd.aquasec.com/nvd/cve-2025-49794
	CVE-2025-49796					libxml: Type confusion leads to Denial of service (DoS) https://avd.aquasec.com/nvd/cve-2025-49796
	CVE-2025-32414	HIGH			2.13.4-r6	libxml2: Out-of-Bounds Read in libxml2 https://avd.aquasec.com/nvd/cve-2025-32414
	CVE-2025-32415					libxml2: Out-of-bounds Read in xmlSchemaIDCFillNodeTables https://avd.aquasec.com/nvd/cve-2025-32415
	CVE-2025-49795				2.13.9-r0	libxml: Null pointer dereference leads to Denial of service (DoS) https://avd.aquasec.com/nvd/cve-2025-49795
	CVE-2025-6021					libxml2: Integer Overflow in xmlBuildQName() Leads to Stack Buffer Overflow in libxml2... https://avd.aquasec.com/nvd/cve-2025-6021

```
==> Scan opsdesk/worker:49d6b65-dirty
2026-02-10T21:47:12Z    INFO    [vuln] Vulnerability scanning is enabled
2026-02-10T21:47:19Z    INFO    [python] Licenses acquired from one or more METADATA files may be subject to additional terms. Use `--debug` flag to see all affected packages.
2026-02-10T21:47:19Z    INFO    Detected OS     family="debian" version="13.3"
2026-02-10T21:47:19Z    INFO    [debian] Detecting vulnerabilities...   os_version="13" pkg_num=87
2026-02-10T21:47:19Z    INFO    Number of language-specific files       num=1
2026-02-10T21:47:19Z    INFO    [python-pkg] Detecting vulnerabilities...
2026-02-10T21:47:19Z    WARN    Using severities from other vendors for some vulnerabilities. Read https://trivy.dev/docs/v0.69/guide/scanner/vulnerability#severity-selection for details.

Report Summary
```

Target	Type	Vulnerabilities
opsdesk/worker:49d6b65-dirty (debian 13.3)	debian	0
app/opsdesk_worker.egg-info/PKG-INFO	python-pkg	0
usr/local/lib/python3.12/site-packages/SQLAlchemy-2.0.35.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/amqp-5.3.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/billiard-4.2.4.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/celery-5.4.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/click-8.3.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/click_didyoumean-0.3.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/click_plugins-1.1.1.2.dist-info/METADATA	python-pkg	0

usr/local/lib/python3.12/site-packages/amqp-5.3.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/billiard-4.2.4.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/celery-5.4.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/click-8.3.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/click_didyoumean-0.3.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/click_plugins-1.1.1.2.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/click_repl-0.3.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/greenlet-3.3.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/kombu-5.6.2.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/opsdesk_worker-0.1.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/packaging-26.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/pip-26.0.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/prompt_toolkit-3.0.52.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/psycopg-3.2.2.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/psycopg_binary-3.2.2.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/python_dateutil-2.9.0.post0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/python_json_logger-2.0.7.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/redis-5.1.1.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/six-1.17.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/typing_extensions-4.15.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/tzdata-2025.3.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/vine-5.1.0.dist-info/METADATA	python-pkg	0
usr/local/lib/python3.12/site-packages/wcwidth-0.6.0.dist-info/METADATA	python-pkg	0

```
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)
```

10.6 make deploy / make cd

This is the “delivery + platform orchestration” step:

- set images in the selected overlay
- render manifests (Kustomize)
- validate (client-side dry run)
- apply secrets/config
- apply manifests
- wait for dependencies (Postgres/Redis)
- run migrations via Job
- wait for rollouts

Evidence:

```
==> Migrations OK ✅

==> Wait for rollouts (api/frontend/worker)
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "opsdesk-api" rollout to finish: 1 old replicas are pending termination...
deployment "opsdesk-api" successfully rolled out
deployment "opsdesk-frontend" successfully rolled out
deployment "opsdesk-worker" successfully rolled out

==> Deploy OK

==> Pods (opsdesk)
NAME                              READY   STATUS        RESTARTS      AGE   IP          NODE             NOMINATED NODE   READINESS GATES
opsdesk-api-6587b85cd-8jjfr       1/1     Running       0             37s   10.1.2.34   docker-desktop   <none>           <none>
opsdesk-api-6587b85cd-zrbsn       1/1     Running       0             77s   10.1.2.29   docker-desktop   <none>           <none>
opsdesk-api-78bb685776-45g9k      1/1     Terminating   1 (31m ago)   33m   10.1.2.22   docker-desktop   <none>           <none>
opsdesk-frontend-7fcfc77d84-67bzm 1/1     Running       0             76s   10.1.2.31   docker-desktop   <none>           <none>
opsdesk-frontend-7fcfc77d84-dg8kq 1/1     Running       0             37s   10.1.2.35   docker-desktop   <none>           <none>
opsdesk-frontend-cb45987c4-f4dq7  0/1     Completed     0             33m   10.1.2.23   docker-desktop   <none>           <none>
opsdesk-migrate-ljkpd             0/1     Completed     0             57s   10.1.2.33   docker-desktop   <none>           <none>
opsdesk-worker-7bdd88fd89-vshc4   1/1     Running       0             76s   10.1.2.32   docker-desktop   <none>           <none>
postgres-0                        1/1     Running       0             33m   10.1.2.26   docker-desktop   <none>           <none>
redis-58f7dc5667-tghz2            1/1     Running       0             33m   10.1.2.25   docker-desktop   <none>           <none>

==> Ingress
NAME              CLASS   HOSTS                           ADDRESS     PORTS     AGE
opsdesk-ingress   nginx   opsdesk.local,api.opsdesk.local   localhost   80, 443   33m

==> HTTP checks (TLS self-signed so -k)
{
  "status": "ok"
}
{
  "ready": true
}
[]

==> Smoke OK
make[1]: Leaving directory '/workspace'
```

10.7 make smoke

Minimal production-style smoke checks:

- /health and /ready via Ingress TLS
- minimal CRUD (create + list)

Evidence:

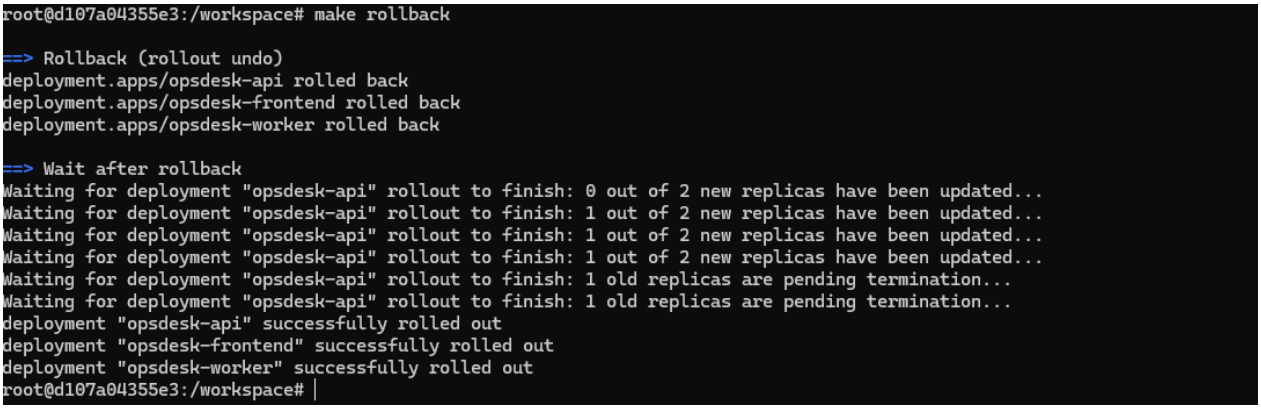
```
root@3996c8907f64:/workspace# make smoke

==> Pods (opsdesk)
NAME                              READY   STATUS      RESTARTS   AGE   IP          NODE             NOMINATED NODE   READINESS GATES
opsdesk-api-6587b85cd-8jjfr       1/1     Running     0          28m   10.1.2.34   docker-desktop   <none>           <none>
opsdesk-api-6587b85cd-zrbsn       1/1     Running     0          29m   10.1.2.29   docker-desktop   <none>           <none>
opsdesk-frontend-7fcfc77d84-67bzm 1/1     Running     0          29m   10.1.2.31   docker-desktop   <none>           <none>
opsdesk-frontend-7fcfc77d84-dg8kq 1/1     Running     0          28m   10.1.2.35   docker-desktop   <none>           <none>
opsdesk-migrate-ljkpd             0/1     Completed   0          28m   10.1.2.33   docker-desktop   <none>           <none>
opsdesk-worker-7bdd88fd89-vshc4   1/1     Running     0          29m   10.1.2.32   docker-desktop   <none>           <none>
postgres-0                        1/1     Running     0          61m   10.1.2.26   docker-desktop   <none>           <none>
redis-58f7dc5667-tghz2            1/1     Running     0          61m   10.1.2.25   docker-desktop   <none>           <none>

==> Ingress
NAME              CLASS   HOSTS                           ADDRESS     PORTS     AGE
opsdesk-ingress   nginx   opsdesk.local,api.opsdesk.local   localhost   80, 443   61m

==> HTTP checks (TLS self-signed so -k)
{
  "status": "ok"
}
{
  "ready": true
}
[]

==> Smoke OK
```

10.8 make rollback (N-1)

Rollback is treated as a standard operation:

- kubectl rollout undo for api/frontend/worker
- wait for rollouts to complete
- post-rollback verification via smoke (optional)

Evidence



11 Section 11 — Security (Detailed, Concrete)

This section documents the runtime security posture and least-privilege access model used across OpsDesk. The intent is not to claim “production-grade security”, but to demonstrate correct defaults, clear threat boundaries, and verifiable controls.

11.1 Local portfolio” threat model (covered / not covered)

Covered (explicit goals)

- Reduce container breakout risk via hardened runtime settings.
- Prevent accidental privilege escalation and reduce attack surface.
- Limit blast radius using least-privilege Kubernetes access (RBAC + SA hardening).
- Avoid “resource denial” incidents through requests/limits and operational guardrails.
- Ensure secrets are not committed and are delivered via Kubernetes Secrets.

Not covered (out of scope for local-first portfolio)

- External IAM/OIDC, SSO, centralized secrets manager (Vault/KMS).
- Multi-node HA, network policies across nodes (depends on cluster capabilities).
- Full supply-chain security (signing/attestations) beyond scanning gates.

11.2 Pod security posture (non-root, no priv esc, read-only, seccomp)

Baseline applied to workloads (frontend/api/worker):

- runAsNonRoot: true
- allowPrivilegeEscalation: false
- readOnlyRootFilesystem: true (with explicit writable mounts only where required)
- capabilities.drop: ["ALL"]
- seccompProfile: RuntimeDefault

Why it matters

- Removes common escalation paths.
- Makes containers behave like immutable artifacts.
- Aligns with Pod Security best practices.

11.3 Capabilities + filesystem immutability (+ /tmp handling)

Capabilities

- Dropping all Linux capabilities reduces what the process can do even if compromised.

Filesystem

- Read-only root filesystem forces explicit writable paths (e.g., /tmp).
- Prevents runtime tampering of binaries/config by default.

11.4 Minimal RBAC (roles, verbs, resources)

OpsDesk uses RBAC with the principle:

- workloads should not get broad cluster access by default
- service accounts are scoped to what the workload truly needs

RBAC goals

- avoid cluster-admin for app workloads
- keep permissions namespace-scoped
- document any required verbs/resources (auditability)

11.5 Dedicated service accounts + automountServiceAccountToken=false

Default posture:

- each component gets its own ServiceAccount
- tokens are not mounted unless required

This reduces token leakage risk and stops accidental in-cluster API access.

11.6 Network exposure (Ingress only, internal services)

Public surface (local)

- Ingress is the only entry point exposed to the host network.

Internal-only

- Postgres and Redis are ClusterIP services, not directly reachable from outside.

11.7 Secrets management local (generate → Secret)

Approach

- local secret material is stored under .secrets/ (never committed)
- scripts/secrets.sh (or equivalent) generates/applies Kubernetes Secrets
- workloads consume secrets via env vars or mounted files

11.8 Resource requests/limits (anti-OOM, predictability)

Requests/limits are defined to:

- reduce OOM risk
- make scaling signals meaningful (HPA)
- improve node stability during incident drills

11.9 Security evidence (what to screenshot)

Best screenshots to use inside Section 11:

- Trivy summary (0 HIGH/CRITICAL for API/worker)
- one screenshot showing frontend findings + “fixed versions available” (if you want to show improvement story)
- kubectl describe deploy snippet showing securityContext fields (optional)

12 Section 12 — Observability (Detailed)

OpsDesk includes a full observability stack to support a real troubleshooting workflow:

metrics → dashboards → alerts → logs.

12.1 kube-prometheus-stack (Prometheus / Grafana / Alertmanager)

The observability stack is installed via Helm values and provides:

- **Prometheus** for scraping and storing metrics
- **Grafana** for dashboards and ad-hoc investigations
- **Alertmanager** for alert routing/aggregation (local scope)

12.2 Prometheus targets (API /metrics, ingress, k8s)

Targets expected to be “UP”:

- API metrics endpoint (/metrics)
- cluster components (kube-state-metrics, node exporters, etc.)
- ingress controller metrics (if enabled)

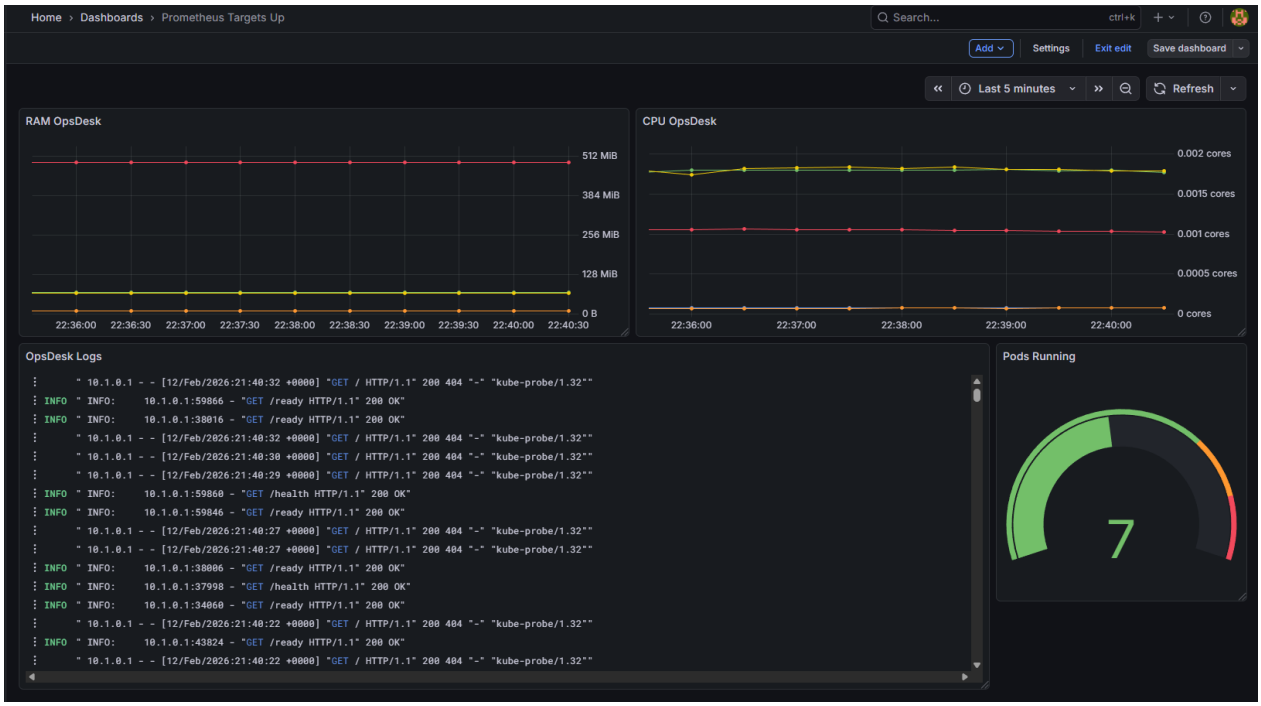
12.3 Dashboards (list + what you read)

Minimum dashboards used:

- OpsDesk / API overview: traffic, errors, latency (p95)
- Kubernetes / workloads: restarts, CPU/memory, pod health
- Ingress: request volume, 4xx/5xx, latency (if enabled)

What you should be able to answer in 60 seconds

- Is the API up? Are errors increasing?
- Are pods restarting / OOMing?
- Is latency spiking under load?
- Is the ingress returning 5xx?



12.4 Alert rules (list + thresholds)

Example alert rules (portfolio-level):

- High 5xx rate (API or ingress)
- High latency p95 (API)
- Pod restart / CrashLoop detection
- Target down (Prometheus can’t scrape)

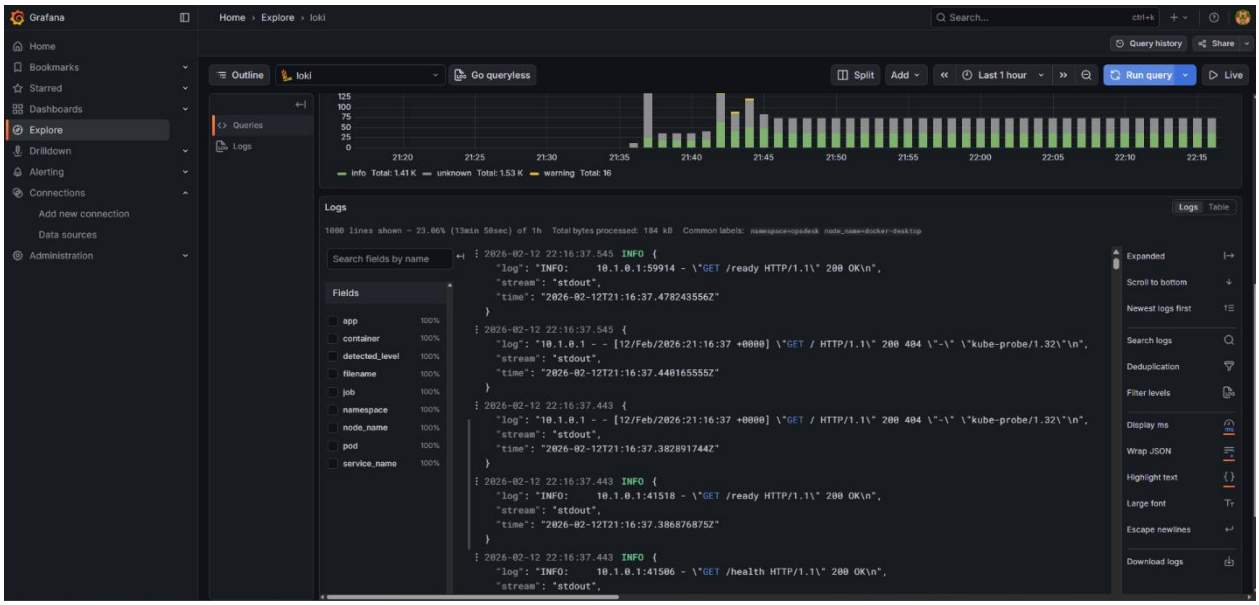
12.5 Loki + Promtail (JSON parsing pipeline + labels)

Logs are centralized using:

- Promtail to ship container logs
- Loki to store and index logs
- Grafana Explore to query logs

Labeling strategy

- Kubernetes labels: namespace, pod, container
- app labels (when present): service, env, version



12.6 Log query patterns (service/env/version/request_id)

Examples of “investigation patterns”:

- start broad: namespace="opsdesk"
- narrow to service: service="api"
- narrow to pod/container
- isolate a failure window by time range
- search by:
  - status_code=500
  - endpoint path (e.g., /api/changes)
  - request_id for end-to-end correlation

12.7 Incident workflow (metrics → logs → fix → verify)

Standard drill:

1. Detect symptom (dashboard shows errors/restarts/latency)
2. Confirm scope (which service/pods affected)
3. Query logs in Loki for the same time window
4. Validate dependencies (DB/Redis readiness, migration job status)
5. Apply mitigation (restart, rollback, config fix)
6. Verify: dashboards stabilize + smoke tests pass

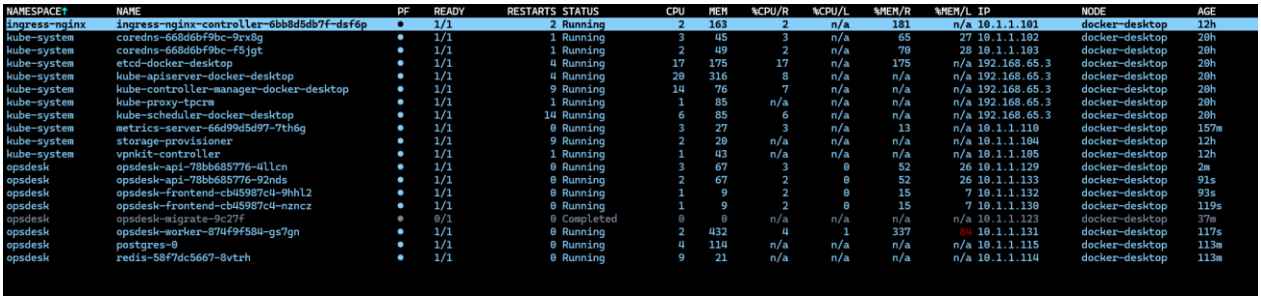
13 Section 13 — Runbooks & Operations (SRE Workflow)

This section proves OpsDesk is not only deployable, but operable: you can diagnose incidents, mitigate safely, and document learnings.

13.1 Debug with k9s (pods/events/logs/rollout)

k9s is used as the primary live-debug cockpit:

- pods health (Ready, restarts, OOMKilled, CrashLoopBackOff)
- events (image pull errors, readiness failures)
- logs (API/worker)
- rollout status for deployments



NAMESPACE↑	NAME	PF	READY	RESTARTS	STATUS	CPU	MEM	%CPU/R	%CPU/L	%MEM/R	%MEM/L	IP	NODE	AGE
ingress-nginx	ingress-nginx-controller-6bb8d5db7f-dsf6p	•	1/1	2	Running	2	163	2	n/a	181	n/a	10.1.1.101	docker-desktop	12h
kube-system	coredns-668d6bf9bc-9rx8g	•	1/1	1	Running	3	45	3	n/a	65	27	10.1.1.102	docker-desktop	20h
kube-system	coredns-668d6bf9bc-f5jgt	•	1/1	1	Running	2	49	2	n/a	70	28	10.1.1.103	docker-desktop	20h
kube-system	etcd-docker-desktop	•	1/1	4	Running	17	175	17	n/a	175	n/a	192.168.65.3	docker-desktop	20h
kube-system	kube-apiserver-docker-desktop	•	1/1	4	Running	20	316	8	n/a	n/a	n/a	192.168.65.3	docker-desktop	20h
kube-system	kube-controller-manager-docker-desktop	•	1/1	9	Running	14	76	7	n/a	n/a	n/a	192.168.65.3	docker-desktop	20h
kube-system	kube-proxy-tpcrm	•	1/1	1	Running	1	85	n/a	n/a	n/a	n/a	192.168.65.3	docker-desktop	20h
kube-system	kube-scheduler-docker-desktop	•	1/1	14	Running	6	85	6	n/a	n/a	n/a	192.168.65.3	docker-desktop	20h
kube-system	metrics-server-66d99d5d97-7th6g	•	1/1	0	Running	3	27	3	n/a	13	n/a	10.1.1.110	docker-desktop	157m
kube-system	storage-provisioner	•	1/1	9	Running	2	20	n/a	n/a	n/a	n/a	10.1.1.104	docker-desktop	12h
kube-system	vpnkit-controller	•	1/1	1	Running	1	43	n/a	n/a	n/a	n/a	10.1.1.105	docker-desktop	12h
opsdesk	opsdesk-api-78bb685776-4llcn	•	1/1	0	Running	3	67	3	0	52	26	10.1.1.129	docker-desktop	2m
opsdesk	opsdesk-api-78bb685776-92nds	•	1/1	0	Running	2	67	2	0	52	26	10.1.1.133	docker-desktop	91s
opsdesk	opsdesk-frontend-cb45987c4-9hhl2	•	1/1	0	Running	1	9	2	0	15	7	10.1.1.132	docker-desktop	93s
opsdesk	opsdesk-frontend-cb45987c4-nzncz	•	1/1	0	Running	1	9	2	0	15	7	10.1.1.130	docker-desktop	119s
opsdesk	opsdesk-migrate-9c27f	•	0/1	0	Completed	0	0	n/a	n/a	n/a	n/a	10.1.1.123	docker-desktop	37m
opsdesk	opsdesk-worker-874f9f584-gs7gn	•	1/1	0	Running	2	432	4	1	337	84	10.1.1.131	docker-desktop	117s
opsdesk	postgres-0	•	1/1	0	Running	4	114	n/a	n/a	n/a	n/a	10.1.1.115	docker-desktop	113m
opsdesk	redis-58f7dc5667-8vtrh	•	1/1	0	Running	9	21	n/a	n/a	n/a	n/a	10.1.1.114	docker-desktop	113m

13.2 Postmortem template (timeline, root cause, actions)

Postmortems include:

- timeline of events (detection → mitigation → recovery)
- root cause analysis
- contributing factors
- corrective actions:
  - immediate fix
  - preventive controls (alerts, limits, probes)
  - documentation updates

13.3 Rollback N-1 (conditions + command + validation)

Rollback is a first-class operation:

- executed via a single Makefile entrypoint
- uses Kubernetes native rollout undo
- validated via rollout status + smoke tests

13.4 Section 15 — What I Learned

- **Kustomize packaging matters**: a strong base/ plus overlays makes environments predictable and reviewable.
- **Readiness is a contract:** /ready must reflect dependency health (DB/broker), not just “process is up”.
- **Hardened defaults reduce risk:** non-root, no privilege escalation, read-only FS, dropped caps, seccomp are high-impact, low-cost controls.
- **Observability is a workflow, not a tool:** dashboards + logs + labels enable fast narrowing from symptom to root cause.
- **Delivery gates prevent chaos:** scan + manifest validation + rollout waits + smoke tests catch problems early.
- **Rollback muscle memory is real SRE value:** practicing rollback + verification makes incident response calm and repeatable.