Laravel on k3s

Workload, container contract, rollout, CI with a migration Job, day-two ops

TL;DR

Start from a **small but correct** workload and a failure path before adding features. A healthy deployment is the first observable unit — **not** proof of disaster recovery or security. Migrations are a named release step, never a pod start.

1. Base workload (Deployment + Service + Ingress)

```
kind: Deployment            # namespace: storefront
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate: { maxUnavailable: 0, maxSurge: 1 }   # on 1 VPS the rollout waits rather than
killing a pod
  revisionHistoryLimit: 5
  template:
    spec:
      terminationGracePeriodSeconds: 30
      containers:
        - name: web
          image: registry.example.com/acme/store:3f18d7c   # tag = commit SHA
          ports: [{ containerPort: 8080, name: http }]
          envFrom: [{ secretRef: { name: storefront-runtime } }]  # a reference, not a secret
manifest
          resources:
            requests: { cpu: 100m, memory: 256Mi }
            limits:   { cpu: 500m, memory: 512Mi }
          readinessProbe: { httpGet: { path: /up, port: http }, periodSeconds: 10 }
          livenessProbe:  { httpGet: { path: /up, port: http }, initialDelaySeconds: 20 }
          lifecycle: { preStop: { exec: { command: ["/bin/sh","-c","sleep 5"] } } }
```

PROBES Readiness = the pod can take traffic. Liveness = the process isn't stuck. Do **not** query the database from liveness — a DB outage should drain traffic via readiness, not restart healthy web processes. k8s Secrets are only base64 — create values outside Git (external-secrets / a protected step).

2. The container is part of the contract

```
FROM php:8.5-cli-alpine AS application
WORKDIR /var/www/html
COPY --from=composer:2 /usr/bin/composer /usr/bin/composer
COPY composer.json composer.lock ./
RUN composer install --no-dev --prefer-dist --no-interaction --no-scripts
COPY . .
RUN composer dump-autoload --classmap-authoritative --no-dev \
```

```
    && addgroup -S laravel && adduser -S laravel -G laravel \
    && chown -R laravel:laravel storage bootstrap/cache
USER laravel
EXPOSE 8080
CMD ["php","artisan","octane:start","--server=frankenphp","--host=0.0.0.0","--port=8080"]
```

Assets + Composer dependencies in the image, config from the environment. One foreground process, a documented port, non-root, an image runnable locally under the same variable names. Octane → check singletons and static state before scaling (PHP-FPM behind nginx is equally valid).

3. CI/CD — build once, promote the same artifact

```
stages: [test, build, migrate, deploy, verify]
variables: { IMAGE_TAG: "$CI_COMMIT_SHA" }

build:
  script:
    - docker build --pull -t "$CI_REGISTRY_IMAGE:$IMAGE_TAG" .
    - docker push "$CI_REGISTRY_IMAGE:$IMAGE_TAG"
    - printf 'DEPLOY_IMAGE=%s\n' "$CI_REGISTRY_IMAGE:$IMAGE_TAG" > image.env
  artifacts: { reports: { dotenv: image.env } }   # later jobs won't pick a different value

deploy-production:
  when: manual                                    # a narrow approval, not a substitute for review
  script:
    - kubectl -n storefront set image deployment/web web="$DEPLOY_IMAGE"
    - kubectl -n storefront rollout status deployment/web --timeout=180s
```

Don't rebuild the same commit for production — the base image / package mirror / build time can change the bytes. CI gets an **identity** (a scoped ServiceAccount/RBAC), not admin credentials.

4. Migration = a named Job, before the web pods

```
kind: Job
metadata: { name: migrate-3f18d7c }              # SHA in the name — a finished Job isn't reused
spec:
  backoffLimit: 1
  ttlSecondsAfterFinished: 86400
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: migrate
          image: registry.example.com/acme/store:3f18d7c   # the same immutable image
          command: ["php","artisan","migrate","--force","--no-interaction"]
          envFrom: [{ secretRef: { name: storefront-runtime } }]
```

```
job="migrate-${CI_COMMIT_SHORT_SHA}"
kubectl -n storefront delete job "$job" --ignore-not-found
sed -e "s|IMAGE_PLACEHOLDER|$DEPLOY_IMAGE|g" -e "s|JOB_PLACEHOLDER|$job|g" k8s/migrate-job.yaml |
kubectl apply -f -
```

```
kubectl -n storefront wait --for=condition=complete "job/$job" --timeout=180s \
  || { kubectl -n storefront logs "job/$job" --all-containers=true; exit 1; }
```

> **EXPAND-CONTRACT** `migrate --force` confirms a non-interactive run, it does not make changes reversible. First a nullable column / new table → code that works with both shapes → async backfill → only later remove the old path. Don't blindly re-run a failed migration — check whether it wrote partial data.

5. Verify and roll back

```
kubectl -n storefront rollout status deployment/web --timeout=180s
kubectl -n storefront get pods -l app=storefront-web \
  -o custom-
columns=NAME:.metadata.name,READY:.status.containerStatuses[0].ready,IMAGE:.spec.containers[0].image
curl --fail --show-error --retry 3 https://shop.example.com/up      # HTTP 200 from an old pod ≠ a
successful deploy

# rolling back the pod template — does NOT undo the DB Job
kubectl -n storefront rollout undo deployment/web
```

Roll back the Deployment **only after** confirming the migration remains compatible with the previous image. A backward-incompatible schema → a deliberate recovery plan, not an optimistic k8s command.

6. Day-two operations

```
# signal, not a dashboard — tell a bad release from a node problem BEFORE deleting pods
kubectl -n storefront get events --sort-by=.lastTimestamp | tail -n 30
kubectl -n storefront top pods ; kubectl top nodes
kubectl -n storefront logs "$pod" --previous --all-containers=true --tail=200   # CrashLoopBackOff

# backup + PROOF of restore (a zero exit code doesn't prove a backup)
mysqldump --single-transaction --routines --hex-blob --host="$DB_HOST" "$DB_DATABASE" | gzip >
"storefront-$(date +%F-%H%M).sql.gz"
# regularly: restore a copy to an isolated database, run an integrity query, record the time
(RPO/RTO)
```

For queue workers, check the depth and errors of the *queue*, not the web pod's readiness. Pass `RELEASE_SHA` as a non-secret variable → structured logs connect "customers see 500" to the revision. Test uploads separately (object storage lifecycle, bucket versioning).

Supporting services (part 4) — briefly

MinIO / GlitchTip / Uptime Kuma / dashboards: each adds storage, upgrades, credentials, alert ownership. Add **only** for a concrete operational need, in its own namespace with requests/limits, with a tested restore, admin endpoints private (VPN / identity proxy — not an IP allowlist alone). DSNs/S3 keys never in a manifest in Git.

When this is NOT the right first deployment

A simple app that one container + a reverse proxy serves more cheaply and clearly. k3s earns its place when you genuinely need rollouts, workload isolation and declarative infrastructure — and the team can carry the platform (backups, upgrades, RBAC).