Engineer

Deployment rollback

restoring an earlier deployment to replace a failing one

What it is

A deployment rollback restores an earlier deployed version or configuration after a change causes trouble. The exact operation depends on what the deployment system controls. A release manager may switch traffic back to a previous slot. An orchestrator may recreate application instances using an earlier revision. In Kubernetes, rolling back a Deployment restores the earlier Pod template from a saved revision. It applies to that Deployment's template and leaves dependencies on their own timelines.

Reverting a commit in the source repository creates new source history that undoes the effect of a change. It takes another deployment to put that source into a running system. Rolling back a deployment can restore service earlier using a previously built artifact. Keep a record of any emergency divergence, then reconcile source and release configuration so the next routine run can explain what is live and will do the intended thing.

Returning code to a previous version may leave recent database rows, events and stored files in their new forms. If the old code can read and use them, a rollback can keep the data people entered. A database restore is a different recovery operation with its own point in time, data consequences and validation. A safe plan states which parts it expects to restore and which parts need to stay as they are.

Use the concept in a decision

Put "What is a deployment rollback?" to work in a real decision. We can help relate the concept to the system you are building or buying. Tested, released and reversible, so changes reach customers on a Tuesday afternoon rather than late on a Friday night.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Automated GIS SaaS deployment with GitHub Actions — deployment, data and reporting, all off the hands

Automated GIS SaaS deployment with GitHub Actions Get in touch →

Why it matters

When an error appears after deployment, the immediate question is what change would help service recover. If the team has a usable earlier artifact and a tested traffic switch, rollback is one candidate. It may have a smaller blast radius than preparing an emergency patch while people wait. If an irreversible external action has already happened, changing the release may stop more occurrences while affected records still need repair. Write down which kind of recovery is being attempted.

Operators need to know what a good recovery looks like. A dashboard may turn green when processes restart even though a checkout still rejects valid payments or queue workers keep failing. Define a few checks for the service's real behavior and record who watches them. Verify the release identifier in the serving process, compare meaningful request results with the baseline, and confirm the error rate during a period long enough to reflect the affected traffic.

Retention also affects whether rollback is practical. An earlier revision can be listed in an orchestrator even when the referenced container image is no longer in the registry or a required secret has expired. Keep known good versions and their operational prerequisites long enough for your recovery window. Rehearse their use under the same permissions that an on-call person will have when they need them.

What it looks like in practice

The portfolio's case about repeatable Azure deployments describes a defined progression through development, staging and production. That predictable route is useful preparation for recovery because teams can reason about the versions and environments they change. The case describes the deployment path. The rollback example below is a separate illustration.

Suppose a team deploys version B of an order service. Errors appear for customers with saved addresses. Version A ran successfully shortly before the change, and the system retains the artifact and the configuration it used. First, the operator checks whether version B wrote any records that version A cannot

read. If both versions support the current schema and request data, the operator can return traffic to A and run a controlled order with a test account. Watch queue errors as well as HTTP status, then confirm that the earlier version is receiving traffic.

A different response may be needed if B removed a column A requires. Bringing back A immediately would then add new failures. A safe alternative might be to stop the affected writes and deploy a small compatibility fix that understands the current data. For a planned change, add new schema first, move readers and writers gradually, and postpone destructive cleanup until the previous versions no longer need the older shape. Test both directions at the intermediate step.

Where it goes wrong

The command succeeds and the team assumes the service is restored. A successful orchestrator action can tell you that a desired template was accepted. Application instances still have to start and meet readiness requirements, and traffic has to reach them. Observe what is actually serving requests and wait for the success conditions you agreed on during the rehearsal. If the old version does not recover the service, stop and reopen the diagnosis.

Exposure may be controlled independently of the deployed version. A feature flag, a traffic rule or a scheduled job can keep a new path active after the application returns to an earlier image. Map these controls before the release. Translate that map into an ordered checklist with one owner per step, including what to do about workers or clients that are still exercising the newer behavior.

A normal deployment loop can also reapply the faulty version after an emergency command. Controllers and GitOps systems often reconcile toward a declared version. Coordinate the change in the desired state with the traffic and artifact recovery. Record what happened, including relevant version IDs, timestamps and customer effects, and use a controlled redeploy when the corrected change is ready.

Further reading

- Kubernetes Deployments and rollback explains revision history and the scope of a restored Pod template.
- DORA release and deployment metrics distinguishes deployment recovery and failure measures from throughput measures.

- PostgreSQL table alteration documents schema changes and their effects on existing tables.
- Helm rollback describes how Helm selects a previous release revision.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a deployment rollback?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/deployment-rollback/