Engineer

Blue-green deployment

◐ switching live traffic between two identical environments

What it is

Blue-green deployment is a way of releasing software with two production environments built to be as alike as possible. One of them, called blue here, serves every request. The new version is installed on the other, green, and tested there while no customer traffic reaches it. When it is ready, whatever sits in front of the two is changed so that every new request goes to green. Blue keeps running untouched, and if green misbehaves, the same change made in reverse puts blue back in service.

The colours carry no meaning, which is the point of them. Names such as “old” and “new” or “primary” and “standby” swap meaning after every release, while two colours simply take turns: after the next switch green is live and blue is the idle one. Martin Fowler described the technique on his bliki in 2010 and attributes the name, a little hazily, to Daniel Terhorst-North and Jez Humble. The usual spelling has a hyphen, though tools write it their own way: Argo Rollouts, a Kubernetes controller that adds deployment strategies of this kind, calls it `blueGreen` in its configuration.

The switch can be any single point that decides where requests go. Fowler’s version changes a router. It can equally be a load balancer’s target, a reverse proxy’s upstream, or in Kubernetes a Service whose label selector picks the Pods that receive its traffic. The two environments do not have to be two data centres either: Fowler lists separate hardware, separate virtual machines, or two zones on one system with their own IP addresses as all good enough.

Two neighbouring techniques make the contrast clear. A rolling update replaces instances a few at a time, so old and new versions answer requests side by side until the last old one is gone, and rolling back means rolling forward again in the other direction. A canary release sends a small share of traffic to the new version

first and widens it step by step. Blue-green moves everything at once, but keeps a complete, running copy of the previous version, so the way back is one step, the same size as the way forward.

Use the concept in a decision

Put “What is blue-green deployment?” 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

A release becomes a short, reversible event instead of a long, one-way one. The slow parts, copying files, starting processes, warming caches, happen on green while blue is still serving, so customers never wait on them. The switch itself is quick, and so is undoing it: nothing is rebuilt or reinstalled, because the previous version never stopped. That changes how a team feels about releasing on an ordinary afternoon, and how quickly it can act when something looks wrong.

Green can also be checked in conditions very close to production before any customer sees it: the same network, the same configuration, the same database. A smoke test against green proves far more than the same test in a staging environment that differs in a dozen small ways. Argo Rollouts formalises this with a preview Service, which routes to the new version before promotion so it can be tested without serving production traffic.

The idle environment has a second job. Once a release has settled, the side that just went quiet becomes the place to stage the next one, and in the meantime it is a warm standby. Fowler points out that the same mechanism “allows you to test your disaster-recovery procedure on every release”, because every switch is a failover that someone is watching.

The costs are plain. Two production environments need two environments’ worth of capacity, at least for the duration of a release. And while the application tier is easy to duplicate, the data is not: blue and green nearly always share one database, which means every release has to be readable by both versions. A team that cannot guarantee that gets a fast switch forward and a broken switch back.

What it looks like in practice

On Kubernetes, the smallest working version is two Deployments and one Service. The Deployments `web-blue` and `web-green` run the same application with the labels `app: web` and `slot: blue` or `slot: green`. The Service that customers reach selects one slot:

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector:
    app: web
    slot: blue
  ports:
    - port: 80
      targetPort: 8080
```

A release puts the new image on the idle side, waits for it, tests it through a second Service that selects `slot: green` alone, and then moves the live one:

```
kubectl set image deployment/web-green web=registry.example.com/web:1.5.0
kubectl rollout status deployment/web-green --timeout=5m
# smoke tests against the web-green preview Service go here
kubectl patch service web -p '{"spec":{"selector":{"slot":"green"}}}'
# the way back is the same command with the other colour
kubectl patch service web -p '{"spec":{"selector":{"slot":"blue"}}}'
```

`kubectl patch` defaults to a strategic merge patch, so only the `slot` key of the selector changes and `app: web` stays where it was. The Service controller keeps scanning for Pods that match the selector and updates the Service’s endpoints when the set changes, so new connections land on green while blue sits ready with the previous image.

Argo Rollouts automates the same dance with one `Rollout` object: an `activeService` that it points at the new pods on promotion, and an optional `previewService` that reaches them before. With `autoPromotionEnabled: false` the rollout pauses before the switch until someone runs `kubectl argo rollouts promote web`.

The DevOps and CI/CD service promises changes that are tested, released and reversible, and blue-green deployment is one way to make the last of those cheap. It rests on two habits that deployments of one platform to Azure show at work, without any second production slot: Bicep templates keep the target infrastructure identical from one environment to the next, and the same built artefact is promoted from development through staging to production, picking up each environment's settings where it lands. That route moves a release through environments in a line. Blue-green adds a twin at the end of it, and the same two habits, identical infrastructure and one unchanged artefact, are what let a team trust that the twin really is one.

Where it goes wrong

A schema change turns the way back into a dead end. Green starts, runs a migration that renames a column, and serves traffic happily. Ten minutes later a fault sends traffic back to blue, which still asks for the old column name and fails on every request. The rollback was planned for the application and not for the data. Fowler's advice from 2015 is to separate the two: ship a database change that both versions can work with, then the application, and remove what only the old version needed once the new one has settled.

State written on green is stranded by a switch back. Anything green keeps for itself, such as files uploaded to its local disk, an in-memory session store or its own cache of pending jobs, is not on blue when traffic returns there. Users find themselves logged out or missing what they just saved. Keep shared state in stores both colours use. Where green must have its own data, Fowler suggests feeding transactions to both environments, or starting green read-only and switching it to read-write only once it has proved itself.

The way back is scaled away before the fault shows. Argo Rollouts scales the old ReplicaSet down after the switch, and `scaleDownDelaySeconds` defaults to 30. A fault that appears after a quarter of an hour, a nightly job or a slow leak, finds no blue to return to. Set the delay to cover the time it usually takes for trouble to surface, and accept that the price is the capacity:

```
strategy:
  blueGreen:
    activeService: web
```

```
    previewService: web-preview
    autoPromotionEnabled: false
    scaleDownDelaySeconds: 3600
```

Promotion follows readiness, not correctness. `autoPromotionEnabled` defaults to true, which means Argo Rollouts switches as soon as the new pods are ready. A readiness probe tells you a process answers, not that checkout works. Pause before promotion as in the block above and test through the preview Service, or attach a `prePromotionAnalysis`, which aborts the rollout when it fails, and a `postPromotionAnalysis`, which switches back to the previous version when it fails after the switch.

The idle side drifts. Between releases, someone applies an urgent configuration change to the live environment only. The next release lands on the idle side, which still has the old setting, and the switch that was supposed to change one thing changes two. Build both slots from the same templates and the same configuration source, so a change to one is a change to both, and treat the idle side as the next release's staging area, as Fowler describes.

Further reading

- Blue Green Deployment — Martin Fowler's 2010 description, with his 2015 note on database changes.
- Argo Rollouts: BlueGreen — active and preview Services, automatic promotion, analysis before and after the switch, and the scale-down delay.
- Kubernetes Service — how a selector picks the Pods that receive a Service's traffic.
- kubectl patch — strategic merge, merge and JSON patches, the command that flips the selector in the example above.

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 blue-green deployment?” 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/blue-green-deployment/