Engineer

Blå-grön driftsättning

◐ att flytta trafiken mellan två likadana produktionsmiljöer

Vad det är

Blå-grön driftsättning är ett sätt att släppa programvara med två produktionsmiljöer som är byggda så lika som möjligt. Den ena, här kallad den blå, betjänar alla anrop. Den nya versionen installeras i den andra, den gröna, och testas där medan inga kunder når den. När den är klar ändras det som står framför de två miljöerna, så att alla nya anrop går till den gröna. Den blå kör vidare orörd, och om den gröna beter sig illa sätter samma ändring gjord baklänges den blå i tjänst igen. På engelska heter det blue-green deployment, och i svenska team säger man ofta bara blue-green.

Färgerna betyder ingenting, och det är hela poängen. Namn som ”gammal” och ”ny” eller ”primär” och ”reserv” byter innebörd efter varje release, medan två färger bara turas om: efter nästa växling är den gröna skarp och den blå den lediga. Martin Fowler beskrev tekniken på sin bliki 2010 och tillskriver namnet Daniel Terhorst-North och Jez Humble, även om han medger att han minns ursprunget något dimmigt. Verktygen stavar det på sitt eget sätt: Argo Rollouts, en Kubernetes-controller som lägger till driftsättningsstrategier av det här slaget, kallar det `blueGreen` i sin konfiguration.

Växeln kan vara vilken enskild punkt som helst som avgör vart anropen tar vägen. I Fowlers version är det en router. Det kan lika gärna vara målet för en lastbalanserare, upstream i en omvänd proxy eller, i Kubernetes, en Service vars label-selektor pekar ut vilka poddar som får trafiken. De två miljöerna behöver inte heller vara två datacenter: Fowler nämner separat hårdvara, separata virtuella maskiner eller två zoner på samma system med egna IP-adresser som lika goda lösningar.

Två grannstrategier gör skillnaden tydlig. En rullande driftsättning byter ut instanser några i taget, så att gammal och ny version svarar sida vid sida tills den sista gamla är borta, och att backa innebär ännu en rullande driftsättning åt andra hållet. En canary release skickar först en liten andel av trafiken till den nya versionen och vidgar den steg för steg. Blå-grön driftsättning flyttar allt på en gång men behåller en komplett, körande kopia av den föregående versionen, så vägen tillbaka är ett steg, lika stort som vägen framåt.

Använd begreppet i ett beslut

Använd ”Vad är blå-grön driftsättning?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Testade, släppta och möjliga att backa, så att ändringar når kunderna en tisdagseftermiddag i stället för sent en fredagskväll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Automatiserade GIS-driftsättning i GitHub Actions — driftsättning, data och rapportering, helt utan manuellt arbete

Automatiserade GIS-driftsättning i GitHub Actions Hör av er →

Varför det spelar roll

En release blir en kort händelse som går att ångra i stället för en lång enkelresa. Det som tar tid, som att kopiera filer, starta processer och värma upp cachar, sker i den gröna miljön medan den blå fortfarande betjänar kunderna, så ingen väntar på det. Själva växlingen går snabbt, och det gör ångrandet också: inget behöver byggas eller installeras om, eftersom den föregående versionen aldrig slutade köra. Det ändrar hur ett team känner inför att släppa en helt vanlig eftermiddag, och hur snabbt det kan agera när något ser fel ut.

Den gröna miljön kan dessutom prövas under förhållanden som ligger mycket nära produktion innan någon kund ser den: samma nätverk, samma konfiguration, samma databas. Ett röktest mot den gröna bevisar betydligt mer än samma test i en stagingmiljö som skiljer sig på ett dussin små punkter. Argo Rollouts formaliserar det med en preview-Service som skickar trafik till den nya versionen före promoteringen, alltså steget där den blir skarp, så att den kan testas utan att betjäna produktionstrafik.

Den lediga miljön har ett jobb till. När en release har satt sig blir den sida som nyss blev tyst platsen där nästa release förbereds, och fram till dess är den en varm reserv. Fowler påpekar att samma mekanism låter en organisation pröva sin plan för katastrofåterställning vid varje release, eftersom varje växling är en failover som någon håller ögonen på.

Priset är lätt att se. Två produktionsmiljöer kräver kapacitet för två, åtminstone medan en release pågår. Och även om applikationslagret är lätt att duplicera är data det inte: den blå och den gröna delar nästan alltid en databas, så varje release måste gå att läsa för båda versionerna. Ett team som inte kan lova det får en snabb växling framåt och en växling tillbaka som inte fungerar.

Så ser det ut i praktiken

I Kubernetes är den minsta fungerande versionen två Deployment-objekt och en Service. Objekten `web-blue` och `web-green` kör samma applikation med etiketterna `app: web` och `slot: blue` eller `slot: green`. Den Service som kunderna når väljer en av dem:

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

En release lägger den nya avbildningen på den lediga sidan, väntar in den, testar den genom en extra Service som bara väljer `slot: green` och flyttar sedan den skarpa:

```
kubectl set image deployment/web-green web=registry.example.com/web:1.5.0
kubectl rollout status deployment/web-green --timeout=5m
# röktester mot preview-Servicen för web-green här
kubectl patch service web -p '{"spec":{"selector":{"slot":"green"}}}'
# vägen tillbaka är samma kommando med den andra färgen
kubectl patch service web -p '{"spec":{"selector":{"slot":"blue"}}}'
```

`kubectl patch` använder som standard en strategic merge patch, så bara nyckeln `slot` i selektorn ändras medan `app: web` står kvar. Servicens controller letar hela tiden efter poddar som matchar selektorn och uppdaterar Servicens ändpunkter när mängden ändras, så att nya anslutningar hamnar i den gröna medan den blå står redo med den föregående avbildningen.

Argo Rollouts automatiserar samma dans med ett enda `Rollout`-objekt: en `activeService` som den pekar mot de nya poddarna vid promoteringen och en valfri `previewService` som når dem innan dess. Med `autoPromotionEnabled: false` pausar utrullningen före växlingen tills någon kör `kubectl argo rollouts promote web`.

Tjänsten DevOps och CI/CD-automatisering utlovar ändringar som är testade, släppta och möjliga att backa, och blå-grön driftsättning är ett sätt att göra det sista billigt. Tekniken vilar på två vanor som driftsättningarna av en plattform till Azure visar i bruk, utan någon andra produktionsmiljö: Bicep-mallarna håller målinfrastrukturen identisk från en miljö till nästa, och samma byggda artefakt förs vidare från utveckling via staging till produktion och plockar upp rätt inställningar där den hamnar. Den vägen för en release genom miljöerna i tur och ordning. Blå-grön driftsättning ställer en tvilling vid vägens slut, och samma två vanor, likadan infrastruktur och en oförändrad artefakt, är det som låter ett team lita på att tvillingen verkligen är likadan.

Var det går fel

En schemaändring gör vägen tillbaka till en återvändsgränd. Den gröna startar, kör en migrering som byter namn på en kolumn och betjänar trafik utan problem. Tio minuter senare skickar ett fel tillbaka trafiken till den blå, som fortfarande frågar efter det gamla kolumnnamnet och fallerar på varje anrop. Återgången var planerad för applikationen, inte för data. Fowlers råd från 2015 är att hålla isär de två: släpp först en databasändring som båda versionerna klarar av, sedan applikationen, och ta bort det som bara den gamla versionen behövde när den nya har satt sig.

Tillstånd som skrivits i den gröna blir strandat vid en växling tillbaka. Allt som den gröna håller för sig själv, till exempel filer uppladdade till den lokala disken, sessioner i minnet eller en egen cache med väntande jobb, finns inte i den blå när trafiken återvänder. Användarna blir utloggade eller saknar det de nyss sparade. Håll delat tillstånd i lagringsplatser som båda färgerna använder. Måste den gröna ha egna data föreslår Fowler att skicka transaktionerna till båda miljöerna, eller att starta den gröna skrivskyddad och öppna för skrivning först när den har visat sig hålla.

Vägen tillbaka skalas bort innan felet syns. Argo Rollouts skalar ner det gamla ReplicaSet-objektet efter växlingen, och `scaleDownDelaySeconds` är 30 som standard. Ett fel som dyker upp först efter en kvart, ett nattligt jobb eller en

långsam minnesläcka, hittar ingen blå att återvända till. Sätt fördröjningen så att den täcker den tid det brukar ta innan problem syns, och acceptera att priset är kapaciteten:

```
strategy:
  blueGreen:
    activeService: web
    previewService: web-preview
    autoPromotionEnabled: false
    scaleDownDelaySeconds: 3600
```

Promoteringen följer beredskap, inte korrekthet. `autoPromotionEnabled` är true som standard, så Argo Rollouts växlar så snart de nya poddarna är redo. En readiness probe säger att en process svarar, inte att kassan fungerar. Pausa före promoteringen som i blocket ovan och testa genom preview-Servicen, eller lägg till en `prePromotionAnalysis` som avbryter utrullningen om den misslyckas och en `postPromotionAnalysis` som växlar tillbaka till den föregående versionen om den misslyckas efter växlingen.

Den lediga sidan glider isär från den skarpa. Mellan två releaser lägger någon in en brådskande konfigurationsändring enbart i den skarpa miljön. Nästa release landar på den lediga sidan, som fortfarande har den gamla inställningen, och växlingen som skulle ändra en sak ändrar två. Bygg båda miljöerna från samma mallar och samma konfigurationskälla, så att en ändring i den ena är en ändring i båda, och använd den lediga sidan som staging för nästa release, så som Fowler beskriver.

Läs mer

- Blue Green Deployment — Martin Fowlers beskrivning från 2010, med hans tillägg från 2015 om databasändringar.
- Argo Rollouts: BlueGreen — aktiv och preview-Service, automatisk promotering, analys före och efter växlingen och fördröjningen av nedskalningen.
- Kubernetes Service — hur en selektor pekar ut vilka poddar som får en Services trafik.
- kubectl patch — strategic merge, merge och JSON-patchar, kommandot som vänder selektorn i exemplet ovan.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är blå-grön driftsättning?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/blue-green-deployment/