Engineer

Bereitstellung

eine Änderung dorthin bringen, wo sie genutzt wird

Was es ist

Eine Bereitstellung ist die Folge von Schritten, die eine getestete Anwendungsversion aus dem Buildsystem in den Betrieb überführt: die gebauten Dateien oder das Container-Image auf die vorgesehenen Rechner oder die Laufzeitplattform übertragen, die Konfiguration dieser Umgebung übergeben, die Anwendung starten und echte Anfragen auf sie umleiten. Das Wort bezeichnet sowohl den Vorgang als auch sein Ergebnis. Ein Team spricht deshalb von einer Bereitstellung, die beim Datenbankschritt hängen blieb, und von den aktuell in jeder Umgebung laufenden Bereitstellungen. Bereitgestellt werden kann eine Website, eine API, ein geplanter Job, eine Datenbankänderung oder das Backend einer mobilen App.

Bereitstellung und Freigabe werden oft getrennt. Die Bereitstellung bringt neuen Code an einen Ort, an dem er laufen kann; die Freigabe entscheidet, wer ihn sieht. Eine hinter einem Feature-Flag verborgene Funktion kann am Montag bereitgestellt und am Donnerstag für Kunden freigegeben werden, ohne die Server erneut anzufassen. Diese Trennung erlaubt kleine, häufige Auslieferungsschritte, während Produktentscheidungen über den Freigabezeitpunkt bei den dafür Verantwortlichen bleiben.

Viele Bereitstellungen liefern heute ein Container-Image aus. Die im Juni 2015 unter dem Dach der Linux Foundation gegründete Open Container Initiative veröffentlicht dazu drei Spezifikationen: wie ein Image aufgebaut ist, wie eine Laufzeitumgebung es nach dem Entpacken ausführt und wie Registries Images speichern und ausliefern. Ein Image wird durch seinen Digest identifiziert, einen Hash seines Inhalts. Derselbe Digest bedeutet deshalb immer dieselben Bytes. Ein Tag wie `1.4.2` ist ein lesbarer Verweis auf ein Image. Wird unter demselben Tag erneut gepusht, verweist es auf etwas Neues.

Die Twelve-Factor App, eine häufig zitierte Methodik für den Aufbau von Webdiensten, teilt den Weg vom Code zum laufenden Dienst in drei streng getrennte Phasen. Die Buildphase macht aus einem Commit ein ausführbares Paket. Die Releasephase verbindet dieses Paket mit der Konfiguration einer Umgebung. Die Runphase startet Prozesse aus einem ausgewählten Release. Jedes Release erhält eine eindeutige Kennung und wird nach seiner Erstellung nie verändert; jede Änderung erzeugt ein neues Release. Dadurch bedeutet ein Rollback nur noch, ein früheres Release auszuwählen.

Wie umgeschaltet wird, hängt davon ab, was der Dienst verträgt. Eine Recreate-Bereitstellung stoppt die alte Version, startet dann die neue und nimmt die Lücke dazwischen hin. Eine rollierende Bereitstellung ersetzt jeweils wenige Instanzen, sodass zeitweise beide Versionen Anfragen bedienen; sie ist der Standard für ein Kubernetes Deployment. Eine Blue-Green-Bereitstellung betreibt die neue Version neben der alten und schaltet den gesamten Verkehr auf einmal um. Die alte bleibt für eine Rückschaltung bereit. Ein Canary leitet zunächst einen kleinen Verkehrsanteil zur neuen Version und vergrößert ihn, solange Fehlerraten und Latenz unauffällig bleiben.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Bereitstellung?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Geprüft, veröffentlicht und rückgängig zu machen: Änderungen erreichen Ihre Kunden am Dienstagnachmittag statt spät am Freitagabend.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. GIS-SaaS-Deployments mit GitHub automatisiert — Bereitstellung, Daten und Berichte ohne Handarbeit

GIS-SaaS-Deployments mit GitHub automatisiert Kontakt aufnehmen →

Warum es wichtig ist

Bei der Bereitstellung beginnt eine Änderung, jemandem zu helfen. Zugleich ist dies der Punkt, an dem ein Fehler Nutzer erreicht. Teams mit seltenen Bereitstellungen sammeln meist viele Änderungen pro Release. Jede Bereitstellung trägt dadurch mehr Risiko und bietet im Fehlerfall mehr mögliche Ursachen. Kleine, häufige Bereitstellungen halten jeden Schritt leicht verständlich und leicht umkehrbar.

Wie einheitlich eine manuelle Bereitstellung abläuft, hängt von der Person ab, die sie ausführt. Schritte werden ausgelassen, in anderer Reihenfolge oder durch eine Urlaubsvertretung etwas anders ausgeführt. Eine Pipeline führt jedes Mal dieselben Schritte aus und hält Commit, Artefakt und Umgebung samt Zeitstempel fest. Mit dieser Aufzeichnung beginnt die Untersuchung eines Vorfalls. Oft beantwortet sie am schnellsten, was sich geändert hat.

Die wichtigste Praxis ist, einmal zu bauen und das Ergebnis weiterzureichen. Das in Staging getestete Artefakt sollte mit dem Artefakt, das die Produktionsumgebung erreicht, Byte für Byte übereinstimmen. Nur die Konfiguration unterscheidet sich, und die Umgebung liefert sie bei der Ankunft des Artefakts. Den Digest beim Bauen festzuhalten und nach Digest bereitzustellen macht dieses Versprechen überprüfbar:

```
podman build -t registry.example.com/web:1.4.2 .
podman push --digestfile web.digest registry.example.com/web:1.4.2
kubectl -n staging set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
# after staging checks pass, the same digest goes to production
kubectl -n production set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
```

Ein Rollback hängt von derselben Disziplin ab. Sind Releases unveränderlich und eindeutig gekennzeichnet, bedeutet die Rückkehr nur einen Verweis auf das vorherige Release. Das dauert Sekunden. Baut jede Umgebung erneut aus dem Quellcode, muss für die Rückkehr ein alter Commit neu gebaut werden, in der Hoffnung, dass seine Abhängigkeiten noch genauso aufgelöst werden.

Wie es in der Praxis aussieht

Die Bereitstellungen einer Plattform nach Azure wurden durchgängig verantwortet. Jede Änderung nahm denselben festen Weg von Entwicklung über Staging nach Produktion, und niemand pushte direkt in eine laufende Umgebung. CI/CD baut die Images und liefert sie aus. Bicep-Vorlagen halten die Zielinfrastruktur in jeder Umgebung gleich, sodass jedes weitergereichte Artefakt auf entsprechende Infrastruktur trifft. Umgebungsabhängige Einstellungen werden getrennt von Geheimnissen gehalten. Ein einzelner Build kann deshalb alle Phasen durchlaufen und bei der Ankunft die Einstellungen der jeweiligen Umgebung erhalten, ohne unterwegs neu gebaut zu werden.

Änderungen erreichen jede Umgebung auf diesem festgelegten Weg ohne spontane manuelle Bereitstellungen. Die Freigabe wurde zu einem kontrollierten Schritt, der jedes Mal gleich abläuft.

Eine frühere Tätigkeit zeigt den Wert dieser Einheitlichkeit in Stunden. Die Automatisierung von Build, Test und Bereitstellung sparte rund 4.000 Stunden, indem sie den Weg von einer Änderung in die Produktion vereinheitlichte und manuelle Schritte entfernte, die je nach ausführender Person variierten. Releases wurden schneller und zuverlässiger.

Auf der eigenen Plattform des Unternehmens bleiben Bereitstellung und Provisionierung getrennt. Bei einer Containerschicht auf Basis von Podman und Quadlet richtet root den Bereitstellungsweg einmal ein. Danach stellt ein unprivilegierter Nutzer erneut bereit, indem er ein einziges festgelegtes Skript über eine eng begrenzte Regel ausführt. Sie erlaubt genau diesen einen Befehl ohne Argumente. Das Skript baut das Image und startet den Dienst neu. Ein gewöhnliches Release benötigt deshalb weder eine Root-Shell noch beliebige Befehle.

Wo es schiefgeht

Die meisten Bereitstellungsprobleme entstehen aus einem Unterschied zwischen dem Geprüften und der tatsächlichen Änderung.

In Produktion läuft etwas, das Staging nie gesehen hat. Baut jede Umgebung eine eigene Kopie, werden Versionsbereiche für Abhängigkeiten auf neuere Versionen aufgelöst, ein Basis-Image ändert sich zwischen Builds oder ein Build-Flag unterscheidet sich. Das getestete Artefakt ist dann nicht dasjenige, das Kunden bedient. Bauen Sie einmal, halten Sie den Digest fest, reichen Sie diesen Digest weiter und behalten Sie alles Umgebungsspezifische in der Konfiguration statt im Artefakt.

Ein Tag wandert, und ein Rollback landet bei der falschen Version. Eine Bereitstellung mit `latest` oder einem nach einem Neubau wiederverwendeten Versionstag hängt davon ab, worauf das Tag an ihrem Ausführungstag zeigt. Die Kubernetes-Dokumentation rät in Produktion von `:latest` ab, weil sich die laufende Version schwerer erkennen und ein Rollback schwerer korrekt ausführen lässt. Nutzen Sie einen nie wiederverwendeten Versionstag und fixieren Sie den Digest, wo die Plattform dies erlaubt.

Die Pipeline meldet Erfolg, bevor die neue Version funktioniert. Viele Bereitstellungsbefehle kehren zurück, sobald die Plattform die Anfrage annimmt. Die neuen Instanzen starten dann noch und können eine Minute später abstürzen. Ein Schritt, der auf die Bereitschaftsmeldung der neuen Version wartet, einige echte Anfragen sendet und die Antworten prüft, macht einen stillen Fehler zu einem roten Build, solange das vorherige Release noch griffbereit ist.

Neuer Code trifft mitten im Rollout auf das alte Schema. Bei einer rollierenden oder Canary-Bereitstellung bedienen alte und neue Versionen nebeneinander Anfragen gegen eine Datenbank. Eine Migration, die eine Spalte umbenennt oder entfernt, beschädigt die noch laufenden alten Instanzen und macht einen Rollback unsicher, weil der alte Code das Entfernte erwartet. Verteilen Sie solche Änderungen auf mehrere Bereitstellungen: zuerst die neue Struktur hinzufügen, dann Code ausliefern, der mit beiden zurechtkommt, und die alte Struktur erst entfernen, wenn keine laufende Version sie mehr braucht.

Weiterführende Lektüre

- Build, Release, Run — das Plädoyer der Twelve-Factor App für drei getrennte Phasen und Releases, die nach ihrer Erstellung nie verändert werden.
- Open Container Initiative — das Projekt der Linux Foundation hinter den Image-, Runtime- und Distributionsspezifikationen, auf denen die meisten Containerbereitstellungen beruhen.
- OCI-Imageformatspezifikation — Manifeste, Schichten, Konfiguration und die Inhaltsdigests, die ein Image identifizieren.
- Kubernetes Deployments — rollierende Updates, die Recreate-Strategie, die Fortschrittsfrist und Rollbacks.
- Kubernetes-Images — Tags, Digests, Abrufrichtlinien und warum `:latest` Rollbacks erschwert.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Bereitstellung?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/deployment/