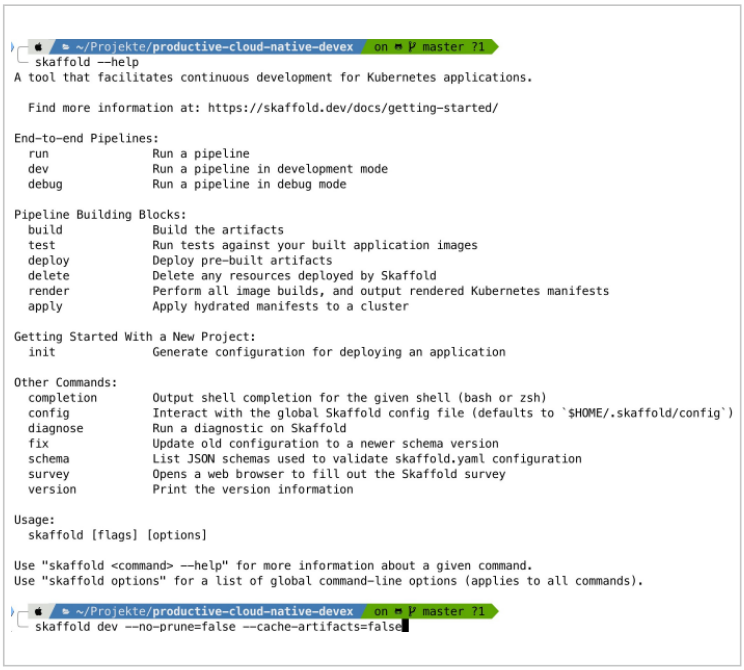

Kontinuierliche Anwendungsentwicklung mit Skaffold

Wie vom Fließband

Mario-Leander Reimer

Kubernetes-native Anwendungsentwicklung benötigt zahlreiche Schritte vom Quellcode zur lokal laufenden Anwendung. Skaffold automatisiert diesen Prozess und lässt sich in CI/CD-Pipelines integrieren.

Die effiziente Entwicklung Cloud-nativer Anwendungen und Microservices stellt Teams vor Herausforderungen. Denn vom Quellcode bis zum Deployment einer Anwendung bedarf es zahlreicher Schritte: Quellcode bei jeder Änderung übersetzen, ein aktualisiertes Docker-Image bauen, taggen und publizieren, das aktualisierte Image über YAML-Manifeste auf einem Kubernetes-Cluster deployen sowie die Ausgaben und Ports lokal zur Verfügung stellen.

Googles Kommandozeilenwerkzeug Skaffold automatisiert diese Continuous-Deployment-Prozesse. Der Befehl `skaffold dev` initiiert eine lokale Dev-Loop, die anschließend bei jeder Änderung am Quellcode automatisch im Hintergrund ausgeführt wird. Das Installieren auf dem Entwickler-PC ist schnell erledigt. Skaffold steht als einfaches Binary für alle gängigen Plattformen und Betriebssysteme auf der GitHub-Releaseseite des Projektes zum Download zur Verfügung (siehe ix.de/zkxb). Für macOS lassen sich alternativ Brew- oder MacPorts-Pakete für die Installation verwenden, unter Windows Scoop- und Chocolatey-Pakete.

Damit Skaffold seine Arbeit aufnehmen kann, braucht es die Datei skaffold.yaml im Root-Verzeichnis des Projektes, die die einzelnen Artefakte und Schritte der Skaffold-Pipeline definiert. Diese Beschreibungsdatei können Entwicklerinnen und Entwickler manuell oder mit dem Kommando `skaffold init` erstellen, das den Quellcode auf die verwendeten Tools analysiert und eine initiale Version dieser Datei generiert. Skaffold erkennt und unterstützt Dockerfile- und Buildpack-Projekte, Maven- und Gradle-Builds mit installiertem Jib-Plug-in sowie Go-, npm- und Python-Projekte. Die Qualität der generierten Datei ist wie so oft mit Vorsicht zu genießen: Sie ist ein guter Start, doch muss man nachträglich Hand anlegen, um die Skaffold-Pipeline exakt und vollständig zu definieren.

Die erste Phase stellt Docker-Artefakte bereit. Skaffold nutzt Tools wie Docker, Jib, Cloud Native Buildpacks, Bazel und ko (für Go); auch eigene Skripte sind verwendbar. Mit kaniko und Google Cloud Build kann man die Anwendung im Cluster oder in der Google Cloud erstellen. Auch für das Tagging der erzeugten Images liefert Skaffold zahlreiche Strategien (Commit-ID, SHA256-Hash, Date and Time).

Test-driven Development im Blick

Für die testgetriebene Entwicklung erlaubt Skaffold, externe Tools wie Container Structure Tests und Securityscanner wie etwa Snyk in die lokale Dev-Loop zu integrieren, um Inhalt und Integrität der erzeugten Images kontinuierlich auf Korrektheit zu prüfen. Die jeweiligen Tools müssen hierfür allerdings vorher lokal installiert sein. Für Container Structure Tests reicht es, die YAML-Testdateien im Skaffold-Manifest zu definieren. Alle anderen Tools müssen Entwickler über Custom-Test-Definitionen unter Angabe des entsprechenden Kommandos sowie ihrer Filesystemabhängigkeiten definieren.

Nach dem Bauen und Testen der Artefakte beginnt die Deployment-Phase. Auch hierfür hat Skaffold Integrationen von Tools im Gepäck. Per kubectl lassen sich Kubernetes-Manifest-Dateien direkt auf dem Cluster anwenden. Leider führt dieses Vorgehen bei komplexen Anwendungen schnell zu vielen redundanten Definitionen. Um die Codewartung gemäß dem Prinzip Don't Repeat Yourself (DRY) zu vereinfachen, implementiert Skaffold Tools wie Kustomize und Helm.

Nach dem Deployment steht die Anwendung lokal bereit. Die Logs der von Skaffold verwalteten Container werden automatisch gesammelt und auf der Konsole kontinuierlich ausgegeben. Für den einfachen Zugriff auf die Anwendung kann man lokale Port-Forwards auf Pods und Services definieren und verwenden.

Alle Phasen der Skaffold-Pipeline sind umgebungsspezifisch konfigurierbar. Umgebungsvariablen und aktueller Kube-Context aktivieren die Profile automatisch; alternativ gibt man das Profil auf der Kommandozeile mit an. Für den Einsatz in CI/CD-Pipelines lassen sich die Phasen zudem auch einzeln oder einmalig ausführen.

(nb@ix.de)

Quellen

Projekt- und Releaseseite: ix.de/zkxb

Mario-Leander Reimer

ist Principal Software Architect bei der QAware GmbH. Er beschäftigt sich mit Innovationen und Technologien rund um den Cloud-native-Stack.