Pivotal®

12 Factor App Manifest

- 1. Die Codebasis
- 2. Abhängigkeiten
- 3. Configuration
- 4. Backing Services
- 5. Stages trennen
- 6. Prozesse
- 7. Port Binding
- 8. Concurrency
- 9. Verfügbarkeit
- 10. Development- und Produktions-Gleichheit
- 11. Logs
- 12. Admin Prozesse

Pivotal

1. Die Codebasis

- Die Grundlage des 12-Factor-Konzepts lautet, dass der Code zentral auf GitHub oder einer anderen Source Control Plattform verwaltet wird, also allen beteiligten Entwicklern jederzeit in der gleichen Fassung zur Verfügung steht. So wird vermieden, dass es mehrere verschiedene Codeversionen gibt.

Pivotal

2. Abhängigkeiten

- Implizite Dependencys sind immer eine schwierige Sache. Natürlich ist es notwendig, auf Datenbanken und Co zuzugreifen; wird die Anwendung in eine andere Umgebung übertragen, kann das aber problematisch werden. Ist eine Abhängigkeit nicht genau da zu finden, wo der Code sie sucht, läuft die Abfrage ins leere. Im Konzept der 12 Factor App ist darum vorgesehen, dass alle Abhängigkeiten explizit spezifiziert und kontrolliert werden. Dependencys werden als Libraries in den eigenen Code integriert und somit gemeinsam mit dem Code in andere Umgebungen portiert.

Pivotal

3. Configuration

- Alle Daten, die sich auf die Konfiguration beziehen, sollten außerhalb des Codes abgelegt werden. Der Code ändert sich nicht durch den Wechsel in eine andere Umgebung; die Konfiguration hängt im 12-Factor-Modell aber durchaus von den Umgebungsvariablen ab. Darum ist hier eine Trennung sinnvoll – besonders auch in Bezug auf die Sicherheit. Passwörter gehören auch zu den Umgebungs- und Konfigurationsdaten und sollten niemals mit dem Code gemeinsam in die Source Control geladen werden. Dadurch entstehen nämlich große Sicherheitslücken, weil dann jeder beteiligte Entwickler Zugang zu allen Zugangsdaten hat.

Pivotal

4. Backing Services

- Backing Services wie Datenbanken, Cache und Co sollten einfach austauschbar sein. Dependencys ändern sich nicht nur durch den Wechsel der Umgebung, sondern häufig auch im Laufe des Projekts. Wird der Cloud Storage Provider gewechselt, sollte diese Änderung nicht viel Arbeit am Projekt bedeuten. Darum sollten solche Services als angehängte Ressourcen behandelt werden, sodass für den Code schlussendlich kein Unterschied zwischen internen und externen Services besteht.

Pivotal

5. Stages trennen

- Änderungen am Code in der Runtime? Bloß nicht! Im 12-Factor-Konzept ist das tabu. Hier werden die Build-. Release- und Run-Zustände der Anwendung klar von einander unterschieden. Der Run-Zustand wird dadurch jederzeit stabil gehalten, kann ohne äußere Intervention laufen. So können sich die Entwickler ganz auf die vorherigen Stufen der Anwendung konzentrieren und in Ruhe testen und arbeiten, ohne dabei unter Umständen Probleme an der laufenden App zu verursachen.

Pivotal

6. Prozesse

- Zustandslose Prozesse machen 12 Factor Apps stabiler. Dadurch ist die Anwendung insgesamt nicht davon abhängig, ob ein bestimmter Prozess ausfällt – ist das der Fall, läuft die App weiter und greif auf eine andere Ressource zu. Das erhöht auch die Skalierbarkeit.

Pivotal

- Die Anwendung muss vollständig per URL erreichbar sein. Für Web-Anwendungen gehört das natürlich eh zum Standard; im 12-Factor-Konzept gilt das allerdings für alle Services inklusive APIs, die unter Umständen auf diesem Weg schneller aufrufbar sind.

Pivotal

8. Concurrency

- Sollten viele kleine Prozesse zusammengefasst werden oder jeder einzeln betrieben? In diesem Modell ist die Antwort klar: Alle Prozesse laufen einzeln und können somit einzeln gestartet, beendet oder skaliert werden. Das macht es möglich, beispielsweise neue Server problemlos zu integrieren und die CPU optimal auszunutzen.

Pivotal

9. Verfügbarkeit

- Dadurch, dass Prozesse einzeln betrieben werden, können sie schnell gestartet und gestoppt werden. Die App ist somit nach einem Update quasi sofort wieder verfügbar.
  Außerdem sollten 12 Factor Apps möglichst robust sein, wenn es um denn Neustart nach Abstürzen geht.

Pivotal

10. Development- und Produktions-Gleichheit

- Die Entwicklungs- und Produktionsumgebung sollten so gleich wie möglich sein. Dadurch wird das berühmte „aber bei mir läuft es doch!“-Problem vermieden. Wenn die Umgebungen sich gleichen, können Testszenarios einfacher in die Realität übertragen werden, sodass weniger Fehler auftreten.

Pivotal

11. Logs

- Logs geben Auskunft über Probleme mit dem Code. Das ist wichtig. Um sicherzustellen, dass diese Logs korrekt gespeichert werden, sollte darum ein gesonderter Service verwendet werden. Das ist nicht Aufgabe des laufenden Codes, der überwacht werden soll. Interne Fehler könnten nämlich in diesem Fall auch Logfiles beschädigen. Also sollten dazu externe Services genutzt werden.

Pivotal

12. Admin Prozesse

- One-Off-Admin-Tasks sammeln wichtige Daten über das reale Verhalten einer Anwendung. Darum ist es wichtig, sie in der Produktions-Umgebung laufen zu lassen, nicht in der Entwicklungs-Umgebung oder mit einer veralteten Version des Codes. Ansonsten sind die ausgegebenen Daten nur schwer auf die real beobachteten Anwendungs-Probleme übertragbar.

Pivotal

Referenz

Mit dem 12 Factor App Manifest zur guten Cloud-App

Pivotal

Pivotal®