Dependency Management ist mehr als “composer update”

Nils Adermann
@naderman
Private Packagist
https://packagist.com

Was sind Dependencies / Abhängigkeiten?

- Services
  - APIs
  - Client-seitige Integrationen (OAuth / External JS / Analytics / …)
- Software
  - Libraries
  - Programme / Werkzeuge
- Externe Assets

Nils Adermann
@naderman

Was ist Dependency Management?

- Assembly / Zusammenfügen
- Dependency Change Management
- Risikoanalyse & -minimierung

Kann zu Build- oder Laufzeit stattfinden

Nils Adermann
@naderman

Dependency Assembly (Zusammenfügen)

- Installieren von Libraries, Tools, etc.
  - composer install
  - apt-get install foo
  - Ausführung von Configuration Management (Puppet, Chef, Ansible, Salt, …)
- Konfiguration zum Verbinden mit Services, externen APIs
  - Authentifizierung
  - Glue Code
- Verbindung zu Services (meist zur Laufzeit!)

Nils Adermann
@naderman

Dependency Assembly (Zusammenfügen)

Vergangenheit:

- Installationsanleitungen mit Einzelschritten
- Readmes, Pakete löschen und installieren

Heute:

- Beschreibung eines Systemzustands (z.B. composer.json, top.sls)
- Werkzeuge zum Herstellen des Zustands (z.B. composer, salt)

Nils Adermann
@naderman

Dependency Change Management

- Dependency Change
  - Hinzufügen, Löschen, Aktualisieren, Ersetzen von Bibliotheken
  - Austauschen von APIs
  - composer update
- Dependency Change Management
  - Risiken, Auswirkungen, Kosten, Vorteile abwägen
  - Architekturentscheidungen die “Change” ermöglichen:
    - Beispiel: Abstraktion für die Austauschbarkeit eines Services

Nils Adermann
@naderman

Risikoanalyse: Verfügbarkeit

Beeinträchtigt Assembly

Beispiele:

- Open Source Bibliothek gelöscht
- Payment Service nicht verfügbar
- EU UstId Service ausgefallen
- Jenkins nicht verfügbar

Nils Adermann
@naderman

Risikominimierung: Verfügbarkeit

- Software ist verfügbar wenn man eine Kopie hat
  - composer cache
  - Forks
  - Private Packagist oder Satis
- Serviceverfügbarkeit ist von externen Faktoren abhängig
  - Kann der Service asynchron ablaufen?
    - z.B. UStId Check bei Payment später durchführen
    - z.B. Private Packagist Paket in Worker initialisiert, Controller braucht GitHub nicht
  - Werden Fehler für Benutzer nachvollziehbar dargestellt?
    - z.B. niedrige Timeouts, Fehlermeldung externer Service X nicht verfügbar

Nils Adermann
@naderman

Risikoanalyse: Kompatibilität

Beeinträchtigt Change Management

Beispiele:

- BC Break in Library Update
- API Semantik ändert sich:
  - Payment API unterstützt keine Kreditkarten-Token mehr, nur noch Payment-Token die auch für Apple Pay, etc. funktionieren

Nils Adermann
@naderman

Risikominimierung: (Neue) Dependencies

Qualitätskriterien für Softwarebibliotheken (und Services)

- Anzahl Maintainer / Entwickler?
- Aktiv Entwickelt?
- Wieviele Benutzer?
  - Packagist zeigt Anzahl Installationen
- Woher bezieht man die Bibliothek?
  - GitHub, eigener svn server? -> Verfügbarkeit
- Alternativen / wie einfach austauschbar? Komplexität?

Nils Adermann
@naderman

Risikominimierung: Kompatibilität

Semantic Versioning (Semver) verspricht Kompatibilität

x.y.z

- Muss aber von Bibliotheken konsequent benutzt werden
- Nur wertvoll wenn BC/Kompatibilität klar definiert ist
  - Siehe http://symfony.com/doc/current/contributing/code/bc.html
- Sonst Version Constraints enger wählen, häufiger prüfen
  - z.B. ~1.2.3 statt ^1.2.3

Nils Adermann
@naderman

Risikominimierung: Kompatibilität

- Automatisiert
  - Tests
  - Statische Analyse
- Manuell
  - Changelogs lesen (und schreiben!)
  - Erfahrung welche Bibliotheken BC Versprechen nicht einhalten

Nils Adermann
@naderman

Risikominimierung: Kompatibilität

- `“composer update”`
  - keine Isolation von Problemen
- `“composer update <package...>”`
  - explizite bewusste Updates
- `“composer update --dry-run [<package...>]”`
  - Effekte von Updates verstehen & vorbereiten

Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
{   “name”: “zebra/zebra”,
    “require”: {
        “pferd/pferd”: “^1.0” }}

{   “name”: “giraffe/giraffe”,
    “require”: {
        “ente/ente”: “^1.0” }}
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
{   “name”: “pferd/pferd”,
    “require”: {
        “giraffe/giraffe”: “^1.0” }}

{   “name”: “ente/ente”,
    “require”: {
        “pferd/pferd”: “^1.0” }}
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
{
    “name”: “my-project”,
    “require”: {
        “zebra/zebra”: “^1.0”,
        “giraffe/giraffe”: “^1.0”
    }
}
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
Installiert: zebra:1.0 giraffe:1.0 ente:1.0 pferd:1.0
Alle veröffentlichen 1.1

$ composer update --dry-run zebra/zebra
    Updating zebra/zebra (1.0 -> 1.1)
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
Installiert: zebra:1.0 giraffe:1.0 ente:1.0 pferd:1.0
Alle veröffentlichen 1.1

$ composer update --dry-run zebra/zebra --with-dependencies
    Updating pferd/pferd (1.0 -> 1.1)
    Updating zebra/zebra (1.0 -> 1.1)
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
{   “name”: “pferd/pferd”,
    “require”: {
        “giraffe/giraffe”: “^1.0” }}

{   “name”: “ente/ente”,
    “require”: {
        “pferd/pferd”: “^1.0” }}
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
{
    “name”: “my-project”,
    “require”: {
        “zebra/zebra”: “^1.0”,
        “giraffe/giraffe”: “^1.0”
    }
}
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
Installiert: zebra:1.0 giraffe:1.0 ente:1.0 pferd:1.0
Alle veröffentlichen 1.1

$ composer update --dry-run zebra/zebra giraffe/giraffe
    Updating zebra/zebra (1.0 -> 1.1)
    Updating giraffe/giraffe (1.0 -> 1.1)
```


Nils Adermann
@naderman

Wie funktionieren partielle Updates?

```
Installiert: zebra:1.0 giraffe:1.0 ente:1.0 pferd:1.0
Alle veröffentlichen 1.1

$ composer update zebra/zebra giraffe/giraffe --with-dependenci
    Updating ente/ente (1.0 -> 1.1)
    Updating giraffe/giraffe (1.0 -> 1.1)
    Updating pferd/pferd (1.0 -> 1.1)
    Updating zebra/zebra (1.0 -> 1.1)
```


Nils Adermann
@naderman

Wie löst man lock Merge Konflikte?

- composer.lock kann nicht ohne Konflikt gemerged werden
  - Hash über relevante composer.json Werte

- `git checkout <refspec> -- composer.lock`
  - `git checkout master -- composer.lock`

- Wiederholen: `composer update <list of deps>`
  - Parameter in Commit message speichern
  - Separates Commit für lock file update

Nils Adermann
@naderman

Risikoanalyse: Compliance / Rechtlich

Beeinträchtigt Change Management

Beispiele:

- Virale Copy-Left Lizenz nicht mit proprietärem Produkt vereinbar
- Terms of Service
  - Darf ich eine API für meinen Zweck benutzen?
    Cloudflare / packagist.org
  - Mit welcher Frist kann mir gekündigt werden?
  - SLA die ausreichenden Support zusichert?

Nils Adermann
@naderman

Risikominimierung: Compliance / Rechtlich

- Lizenz von Softwareabhängigkeit muss zu Produktlizenz oder Kundenrichtlinien passen
  - `composer licenses`
  - Private Packagist License Review

- Terms of Service / SLA / Vertragsbedingungen
  - Kriterium für Auswahl
  - Verhandelbar
  - Starke Abhängigkeiten rechtfertigen finanziellen Aufwand um Sicherheit zu schaffen

Risikoabwägung

- Plan B formulieren
- Wahrscheinliche Probleme und Probleme mit großen Auswirkungen identifizieren

- Abhängigkeiten können viel Zeit und Geld sparen!
- Ressourcen nur soweit aufwenden bis das Risiko akzeptabel ist

Nils Adermann
@naderman

Zusammenfassung

- composer update [--dry-run] <package>
- git checkout <branch> -- composer.lock
- Formalisiert BC Versprechen gegenüber Nutzern eurer Bibliotheken
- SemVer: Habt keine Angst die erste Versionsnummer zu erhöhen
- Dokumentiert Änderungen an Abhängigkeiten
- Habt einen Plan B
- Verschwendet keine Ressourcen mit Problemen die mit geringer Wahrscheinlichkeit auftreten oder kaum Folgen haben

Entwickler müssen Dependency Management auch aus Geschäftssicht betrachten
Management sollte Risiko aus Softwareabhängigkeiten nicht ignorieren

Nils Adermann
@naderman

Vielen Dank!

Fragen / Anmerkungen?

E-Mail: n.adermann@packagist.com
Twitter: @naderman
Feedback: https://joind.in/talk/7841b