TYPO3 V8 => V10 :: Lessons learned

Ziele

Upgrade auf TYPO3 Core V10

Umbau Projekt-Setup

- Ziel: So wenig php Dateien über Browser aufrufbar wie mögich.
- möglich durch Nutzung `helhum/helhum/typo3-secure-web`
- Aufteilung statt einer DocumentRoot `web` nun 3 Verzeichnisse
- `public`
- `private`
- `packages`

Nutzung des Subtree-Splits

- `typo3/cms-core` + gewünschte Sys-Exts
- statt `typo3/cms`
- ist zwar schon lange möglich, aber manche Extensions aus dem TER / packagist erfordern das immer noch.

Nutzung von Umgebungsvariablen

- Umgebungsabhängige Einstellungen waren bisher in php Dateien gesetzt.
- Bei php kann man aber immer Code einbauen, der unerwünscht ist und nicht durch ein Deployment zurückgesetzt werden kann
- Besser Umgebungsvariablen nutzen
- möglich durch Nutzung von `helhum/dotenv-connector`
- Nutzbar in der Siteconfiguration:
  - `solr_host_read: '%env(SOLR_HOST)%'`
- Nutzbar in php

```
$GLOBALS['TYPO3_CONF_VARS']['DB']['Connections']['Default'] = [
    'charset' => 'utf8',
    'driver' => 'mysqli',
    'dbname' => getenv('MYSQL_DATABASE'),
    'host' => getenv('MYSQL_HOST'),
    'port' => getenv('MYSQL_PORT'),
    'user' => getenv('MYSQL_USER'),
    'password' => getenv('MYSQL_PASSWORD')
];
```

Ablauf

Strategie

- Focus auf den TYPO3 core und des Basis-Setups
- anschließend 3rd Party Extensions
- anschließend eigene Extensions

Umbau und Aufräumen

- **alle** Extensions (eigene und 3rd party) aus `typo3conf/ext` nach `packages/` verschieben
  - Sicherung des aktuellen Zustands
  - die eigenen werden wir dort auch weiter pflegen / entwicklen
  - 3rd Party Extensions
    - können wir nach und nach entfernen (falls TYPO3 V10 kompatibel)
    - können wir an der Stelle TYPO3 V10 kompatibel machen (falls notwendig)
- alle Non-Core Extensions aus der composer.json entfernen
  - ansonsten würde die Installation via composer immer brechen w\ version Requirements
- composer.json ergänzen

```
"repositories": [
   {
     "type": "path",
     "url": "./packages/*"
   },
```

- **ACHTUNG**: Extension-Settings können aus der LocalConfiguration entfernt werden
  - Referenz-Seite oder Versionsverwaltung notwendig, um alte Settings wieder herstellen zu können
  - Move `EXTCONF` (serialisiert) => `EXTENSIONS` (unserialisiert)

TYPO3 Core

- `composer.json` auf die neue Version anpassen
- `composer update` => `composer.lock` wird neu aufgebaut
- Database Compare
  - nicht destruktiv, da sonst z.B. die Tabellen von RealURL verloren gehen. Die werden beim Upgrade benötigt.
- alle Upgrade Wizards laufen lassen
  - `typo3cms upgrade:all`
  - oder Backend (man kann die auch > 10 Wizards auch durchklicken ;-), will man aber nicht)
- Basic - SiteConfiguration bauen
- danch sollte BE und FE wieder aufrufbar sein

TYPO3 V10 kompatible Extension aktivieren

- step by step wieder in die `composer.json` aufnehmen
- Testen, testen, testen
- auch wenn mache behaupten TYPO3 V10 kompatibel zu sein, gibt es doch immer wieder Bugs / Edgecases, die nicht getestet worden sind, Bsp: `EXT:ig_ldap_sso_auth`
- Extension - Konfiguration übernehmen

Eigene TYPO3-Extension TYPO3 V10 kompatibel machen

InstallTool => Extensionscanner

- gibt sehr gute Hinweise, was nicht mehr funktioniert
- Statische Code Analyse => auch "false positives" möglich
  - `@extensionScannerIgnoreLine@ schließt die folgende Zeile vom Scan aus
- Breaking Changes und Deprecations beheben
  - Deprecations machen die Extension safe für TYPO3 V11

TypoScript

- neue `Include`-Syntax verwenden
- neue TypoScript-Conditions nutzen und ggf. eigene auf die SymfonyExpression Language umbauen
- Spracheinstellungen aus der `config` - Section löschen (=> SiteConfiguration)

Datenbank

- DB-Konfiguration "strict mode" kompatibel machen

Eigene TYPO3-Extension TYPO3 V10 kompatibel machen

Dependency Injection

- bisher nur für Extbase-Extensions verfügbar
- jetzt für alle Klassen des TYPO3 Cores und installierte Extensions
- Grund:
  - Der Symfony Service Contaier und damit die Dependency Injection wurde integriert
- Configuration über `EXT:extName/Configuration/Services.yaml`
- Üblicherweise reicht:

```yaml
# Configuration/Services.yaml
services:
  _defaults:
    autowire: true
    autoconfigure: true
    public: false

  In2code\Downloads\:
    resource: '../Classes/*'
```

- Alle Klassen der TYPO3 System Extensions stehen zur Verfügung
- Dependency Injection sollte über den Konstruktor (oder über inject Methoden) erfolgen
- `@inject` - Annotation funktioniert nicht mehr!!
- `ObjectManager->get` ist deprecated und wird in V11 / V12 entfernt
  - kann und sollte man jetzt schon machen

Eigene TYPO3-Extension TYPO3 V10 kompatibel machen

Annotations

- Wie gesagt `@inject` geht nicht mehr
- Extbase-ORM Annotations haben sich geändert
  - Bsp.: `@cascade` => `@TYPO3\CMS\Extbase\Annotation\ORM\Cascade("remove")`

Property Mapping

- Bisher in der `ext_tables.sql`
- Jetzt in `EXT:extName/Configuration/Extbase/Peristence/Classes.php`

TCA

- Feld für das Pfad-Segment des Datensatzes hinzufügen

```
'path_segment' => [
    'exclude' => true,
    'label' => 'LLL:EXT:users/Resources/Private/Language/backend.xlf:' .
        Appointment::TABLE_NAME . '.path_segment',
    'config' => [
        'type' => 'slug',
        'generatorOptions' => [
            'fields' => ['title'],
            'fieldSeparator' => '-',
        ],
        'fallbackCharacter' => '-',
        'eval' => 'uniqueInSite',
    ]
],
```

- Manche TCA Typen erfordern jetzt explizit einen `renderType

Eigene TYPO3-Extension TYPO3 V10 kompatibel machen

Upgrade Wizards (1/3)

- Anwendungsfall:
  - URL-Konfiguration via realURL
  - URls sollen in das Pfadsegment eines Datensatzes übernomme werden
- keine feste Abfolge der Upgrade-Wizards

1. Schritt: Upgrade - Wizard registrieren

```
$GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['ext/install']['update']['sliderUpdateWizard']
   = \In2code\In2template\Updates\SliderUpdateWizard::class;
```

Eigene TYPO3-Extension TYPO3 V10 kompatibel machen

Upgrade Wizards (2/3)

2. Schritt: Upgrade - Wizard implementieren

```
class SliderUpdateWizard implements UpgradeWizardInterface
{
    public function setOutput(OutputInterface $output): void
    {
        /** **/
    }

    public function getIdentifier(): string
    {
        /** **/
    }

    public function getTitle(): string
    {
        /****/
    }

    public function getDescription(): string
    {
        /****/
    }
```

Eigene TYPO3-Extension TYPO3 V10 kompatibel machen

Upgrade Wizards (3/3)

```
    public function executeUpdate(): bool
    {
      /** Das ist die eigentliche Update-Funktion **/
    }
    public function updateNecessary(): bool
    {
        /****/
    }
    public function getPrerequisites(): array
    {
        return [
            DatabaseUpdatedPrerequisite::class,
        ];
    }
}
```

Eigene TYPO3-Extension TYPO3 V10 kompatibel machen

php - Sprach - Features nutzen

- Viele Änderungen von php 7.0 => 7.4
- Upgrade auf 7.2 kein Problem bzw. ein Muß
- öffentliche Extension: man ist an php 7.2 gebunden
- private Extension: jede php-Version möglich, die in der Infrastruktur unterstützt wird

TYPO3 V8 => V10 :: Lessons learned

Fragen

Anmerkungen

Danke!