Entwicklerhandbuch für Online-Projekte der dena

Stand: 15.07.2024

Hinweis:
Wordpress ist für das Energiesprong-Projekt nicht zulässig.

Inhalt

Willkommen **7**
Ansprechpersonen 7
1. Richtlinien **8**
1.1 Frontend-Guidelines **8**
Erlaubte Technologien 8
Nicht erlaubte Technologien 8
JavaScript 8
HTML & CSS 9
Video 9
Audio 9
Webfonts und Icons 9
Grafiken und Ornamente 9
Browsersupport 9
Mobile 10
Responsive Umsetzung 10
Minimierung und Kompilierung 10
Zugänglichkeit / Barrierefreiheit 10
1.2 Guidelines Extension-Programmierung **11**
Anforderungen an Extensions 11
1.3 Guidelines zur Suchmaschinenoptimierung **12**
1.4 Hinweise zum Aufbau von Dokumentationen **14**
Projekt-Dokumentation 14
Systemhandbuch 14
Systemhandbuch-Ergänzung 16
Extensions-Dokumentation 17
Redakteursleitfaden 18
Schnittstellen-Dokumentation 19

Datenimport via CSV oder TXT 19
Webservice-Schnittstellenbeschreibung 19
1.5 TYPO3 ELTS 20
Setup 20
1. Zugangsdaten 20
2. Projekt Bundle anpassen 20
2. Setup & Deployment 21
2.1 Workflow 21
Projektverlauf 21
Entwicklungszyklus 21
lokale Entwicklung 22
Abnahme auf STAGING 22
Ausliefern auf LIVE 22
Tickets 22
2.2 Projektstruktur 22
TYPO3 - Struktur 23
Dateien im Git-Repository 23
WordPress - Struktur 23
Dateien im Git-Repository 23
Struktur nach dem Bauen des Projektes 24
2.3 Konfiguration 25
Aufbau der Anwendung 25
TYPO3 26
2.4 Komponenten 26
Verzeichnisse 26
unerwünschte Komponenten 26
Updates 26
Patches 26
2.5 Abhängigkeiten 27
Bundle 27

Komponenten ... 27
2.6 Datenbank ... 27
bereinigte Auszüge ... 28
vollständige Datenbanken ... 28
2.6 Auslieferung ... 28
Übersicht ... 28
Zielsysteme ... 29
2.7 Auslieferung mit Capistrano ... 29
Konfiguration ... 29
`.deployment` - Ordner ... 29
.gitlab-ci.yml ... 30
Zugangsdaten in CI-System ... 30
Zugangsdaten in Zielsystem ... 30
2.9 Versionskontrolle ... 34
2.10 Infrastruktur ... 35
Infrastruktur SITS Deutschland ... 35
2.11 Hosting ... 36
2.12 Legacy Projekte ... 37
Komponenten konsolidieren ... 37
Mindestanforderungen ... 37
öffentliche Pakete ... 37
gepatchte Pakete ... 37
nicht weiter entwickelte Pakete ... 38
Vorgehen ... 38
Package Type ... 38
Installer ... 38
Optional: Installer Name ... 38
Bundle: Installer und Pfad ... 39
2.13 Beispiele ... 39
bundle `composer.json` ... 39

title: Beispiel composer.json im Projekt bundle 39
Hinweise 39
AdditionalConfiguration 41
LocalConfiguration 42
Configuration for Deployment via Helm 45
Variables 46
Includes 47
Overrides For Pre-Defined Jobs 47
Values 48
CI/CD 50
`.gitlab-ci.yaml` 50
`ignore_tables.txt` 53
Verwendung 53
`deploy.rb` 54

Willkommen

dena-Webprojekte werden nach einer einheitlichen Struktur aufgebaut, damit gemeinsame Lösungen für die Versionierung, Auslieferung und den Betrieb verwendet werden können.

Dieses Dokument informiert über die technischen und gestalterischen Vorgaben für die Umsetzung von Webprojekten für die dena. Die Anforderungen an den Aufbau der Software und Konfiguration für die automatische Auslieferung von dena-Webprojekten werden beschrieben

Die Vorgaben in diesem Dokument gelten bei Auftragserteilung als vereinbart und sind Vertragsbestandteil.

Abweichungen von diesen Vorgaben müssen schriftlich in den Auftragsunterlagen oder den zugehörigen Anlagen festgehalten werden.

Die in diesem Dokument beschriebenen Vorgaben zur Umsetzung von Webprojekten beschreiben das idealtypische Setup sowie die Systemumgebung am Beispiel des dena-Baukastensystems. Grundsätzlich gelten diese für jedwedes Webprojekt, wobei sie jedoch nicht sämtliche inhaltlich-technische Szenarien und Konzepte von Webprojekten abbilden können.

Für darüberhinausgehende Bestandteile von Webprojekten, wie z.B. die Einbindung eines Logins oder die Nutzung eines anderen Content-Management-Systems, bestehen unter Umständen weitere, individuelle Vorgaben. In jedem Fall ist es erforderlich, schon in den ersten Planungsschritten eines Webprojekts das Team Online in die Projektideen einzuweihen.

Ansprechpersonen

Name	E-Mail	Organisation	Funktion
Mezger, Stephan	Stephan.Mezger@dena.de	dena	Teamleiter Fachkommunikation & Netzwerke
Lamers, Michael	Michael.Lamers@dena.de	dena	Seniorexperte Fachkommunikation & Netzwerke
Frank, Ann-Katrin	Ann-Katrin.Frank@dena.de	dena	Teamleiterin Digital
Schießer, Johann	Johann.Schiesser@dena.de	dena	Seniorexperte Digital
Knochenhauer, Denise	Denise.Knochenhauer@dena.de	dena	Seniorexpertin Digital
Neumann, Dirk	Dirk.Neumann@dena.de	dena	IT-Verantwortlicher
Wenzel, Dirk	d.wenzel@familie-redlich.de	CPS GmbH	Entwickler

Haupenthal, Yannic	y.haupenthal@familie-redlich.de	CPS GmbH	WebOps/Systemadministration

1. Richtlinien

1.1 Frontend-Guidelines

Die von der Dena betriebenen Websites und Onlineapplikationen sollen möglichst gute Browserkompatibilität und hohe Zugänglichkeit erreichen. Websites sollen benutzbar sein, egal mit welchen Browser, Betriebssystem oder Gerät auf sie zugegriffen wird. Für Menschen mit Behinderungen – permanent oder zeitweise – sollen die Websites möglichst gut benutzbar sein.

Für die Umsetzung sollen Technologien so eingesetzt werden, dass sie ihre Vorzüge am besten einsetzen können:

- Semantisches valides HTML für die Dokumentenstruktur
- CSS für die Präsentation
- JavaScript für Funktionalität

Entwicklungen sollen im Sinne des "Progressive Enhancements" vorgenommen werden, d.h. bei Einsatz fortgeschrittener Technologien (die evtl. nicht jedem Besucher zur Verfügung stehen) soll zuerst eine einfache funktionsfähige Version entwickelt werden, die dann aufgewertet wird. Dabei gilt: Websites müssen nicht auf jedem System gleich aussehen, aber Kernfunktionen und müssen zugänglich sein.

Erlaubte Technologien

- HTML5
- CSS
- JavaScript

Nicht erlaubte Technologien

- Flash
- Java
- Plugins wie z.B. Unity (Ausnahmen nur in Einzelabsprache)

JavaScript

Komplexe interaktive Applikationen dürfen mit JavaScript entwickelt werden, aber die Anzeige statischer Inhalte und die Navigation auf der Site müssen auch ohne Javascript funktionieren.

Der JavaScript-Code muss fehlerfrei sein und darf keine Funktionen enthalten, die als veraltet (depreceated) klassifiziert sind.

Als JavaScript-Bibliothek ist jQuery zu verwenden. Andere JavaScript-Bibilotheken sind nicht zulässig (Ausnahmen können in Einzelabsprache vereinbart werden).

HTML & CSS

HTML-Dokumente sind in validem HTML5 zu erstellen. Maßgeblich ist der Validator des W3C (http://validator.w3.org/). CSS-Dokumente sind in validem CSS zu erstellen.

Video

- Video muss als HTML5-Video bereit gestellt werden im Format
- H.264 (MP4) und optional WebM (VP8 oder VP9)
- Optional YouTube (responsiv eingebunden)

Audio

Audio muss im Format MP3 verwendet werden.

Webfonts und Icons

Webfonts sind zulässig, sofern sie mit dem Performance-Budget vereinbar sind und bei Nicht-Anzeige nicht das Layout zerstören.

Icons sind in Form von SVG-Sprites umzusetzen.

Die Umsetzung von Text als Grafik ist generell nicht zulässig (einzige Ausnahme: Logos).

Grafiken und Ornamente

Der Einsatz von Bitmap-Grafiken ist zu minimieren. Wenn es möglich ist müssen CSS-Eigenschaften (z.B. für Verläufe, runde Ecken) oder SVG-Grafiken (Icons, Illustrationen) eingesetzt werden.

Browsersupport

Eine Website muss nicht in allen Browsern identisch aussehen. Sie soll aber möglichst weitgehend zugänglich und nutzbar sein.

Grundsätzlich müssen alle Kernfunktionen der Websites für die folgenden Browser funktionieren (Zeitangaben beziehen sich auf den Zeitpunkt der Liveschaltung - empfohlen ist, bei Auftragserteilung die konkreten Versionen zu benennen):

- Microsoft Edge nicht älter als ein Jahr
- Firefox ab der letzten ESR-Version
- Chrome nicht älter als ein Jahr
- Safari nicht älter als ein Jahr

Mobile

- iOS in den letzten beiden Versionen
- Android in den letzten beiden Versionen

Responsive Umsetzung

Unter einer responsiven Umsetzung verstehen wir ein fluides Layout mit zwei Breakpoints (also drei Ansichten). Das Design soll für die Betrachtung auf Smartphones, Tablets und Desktop-Rechnern gesondert angepasst werden. Dabei stehen die Bezeichnungen für Geräteklassen, nicht für einzelne Auflösungen. Maßgeblich für die Wahl der Breakpoints ist die optimale Anzeige der Inhalte. Je nach Art der Inhalte können auch Zwischenanpassungen (minor breakpoints) erforderlich sein. Bei der Vergrößerung des Layouts in der Desktop-Ansicht ist darauf zu achten, dass Zeilenlängen nicht beliebig lang werden - ggf. ist das Layout entsprechend zu begrenzen.

Minimierung und Kompilierung

Minimierte CSS und JS-Dateien sind zulässig und aus Performancegründen erwünscht. Es sind immer auch aktuelle nicht minimierte Dateien bereit zu stellen. Bevorzugt sind Lösungen, die das Minimieren und ggf. Kombinieren automatisch erledigen. Der Einsatz ist in der Systemdokumentation zu beschreiben.

Der Einsatz kompilierter CSS-Dateien ist zulässig, wenn auch die aktuellen Quelldaten bereitgestellt werden. Der Einsatz ist in der Systemdokumentation zu beschreiben.

Zugänglichkeit / Barrierefreiheit

Websites der dena sollen für Menschen mit Behinderungen so zugänglich wie möglich sein.

Als Orientierungsrahmen für Barrierefreiheit dienen die Prüfschritte des BITV-Tests: https://bitvtest.de/pruefverfahren/bitv-20-web. Die einzelnen Prüfschritte sollten mindestens „eher erfüllt“ sein. Zum Test werden drei Seiten geprüft. Kann ein Prüfschritt aus projektbezogenen oder dringenden technischen Gründen nicht mindestens „eher erfüllt“ werden, sollten die Projektverantwortlichen der dena dazu nachweislich vor Abschluss der Entwicklungsarbeiten informiert werden.

Bei der Gestaltung von Websites ist das Design-Manual der dena verbindlich zu beachten. In Einzelfällen kann das Design-Manual des jeweiligen Projektes zum Einsatz kommen. Dies muss mit der Unternehmenskommunikation der dena abgestimmt werden.

1.2 Guidelines Extension-Programmierung

Für die dena programmierte Extensions müssen nach aktuellem Stand der Technik entwickelt werden und den Coding Guidelines des TYPO3-Projekts folgen.

Bevor eine eigene Extension erstellt wird ist zu prüfen, ob bereits eine passende Extension im TYPO3-Extension-Repository verfügbar ist. Vorzugsweise sind solche Standardextensions zu verwenden.

Wenn eigene Extensions entwickelt werden müssen, sollen diese nach Möglichkeit nur Funktionen enthalten, die nicht bereits durch das Kernsystem oder vorhandene Standardextensions bereitgestellt werden (beispielsweise der TYPO3-Editor oder die Formularengine).

Neben der Erfüllung der vereinbarten Funktionalität gelten die folgenden Ziele für die Entwicklung eigener Extension:

- Hohe Sicherheit
- Wartbarkeit
- Erweiterbarkeit
- Transparenz

Die in den TYPO3 Coding Guidelines (https://docs.typo3.org/typo3cms/CoreApiReference/CodingGuidelines/) benannten Konventionen und Ansätze (best practices) sind anzuwenden. Abweichungen von den Guidelines gelten als Mängel sofern sie nicht von der dena schriftlich genehmigt sind.

Allgemeine Anforderungen und Hinweise zur TYPO3-Programmierung https://docs.typo3.org/typo3cms/CoreApiReference/

Anforderungen an Extensions

Extensions müssen autonom laufen. Die Installation oder die Entfernung der Extension aus System darf keinen System-Crash verursachen.

Aus anderen Extensions erforderliche Komponenten müssen als Bedingung implementiert werden. Falls Komponenten bzw. Extensions nicht gefunden werden darf kein System-Crash passieren. Ein Hinweis auf fehlende Komponente soll im debug-Modus erscheinen.

Extension-Struktur

- Alle in der Extension verwendete Komponenten, HTML-Templates, CSS- und JavaScript- Dateien so wie TypoScript, müssen in entsprechenden Ordner abgelegt werden.
- Anforderungen der Coding Guidelines beachten: https://docs.typo3.org/typo3cms/CoreApiReference/ExtensionArchitecture/FilesAndLocations/

Konflikte vermeiden

- Kompatible TYPO3-Versionen müssen ausgewiesen werden
- Nicht-kompatible Extensions von Drittanbietern müssen aufgelistet werden

Abhängigkeiten von anderen Extensions

- Falls erforderlich muss die Reihenfolge des Ladens der Extensions explizit angegeben werden

Verwendung von TYPO3 APIs

- Bei der Implementierung APIs verwenden (\TYPO3\CMS\Core\Utility\GeneralUtility) statt direkten Aufrufen

Verwendung von JavaScript-Bibliotheken

- Im System bereits eingesetzte Frameworks verwenden.
- Die Einbindung erfolgt ausschließlich im TypoScript und nicht im PHP-Code.
- Die Einbindung über Konstanten steuern – das soll die Deaktivierung der Einbindung ermöglichen falls die Bibliotheken bereits eingebunden sind.
- Inkludierung ohne Blockade des Renderings soweit möglich (am Seitenende oder asynchron).

Verwendung Caching Frameworks

- Keine

Konforme Lokalisierung

- Fallback auf default-Sprache

Realurl- Konforme Umsetzung

- Konfiguration für Realurl-Extension soll geliefert werden.
- Es soll möglich sein, Teile des Pfades bei Bedarf zu deaktivieren (über entsprechenden TypoScript-Parameter)

Dokumentation

- Zur Dokumentation siehe 9 Aufbau von Dokumentationen

1.3 Guidelines zur Suchmaschinenoptimierung

Die von der Dena betriebenen Websites und Onlineapplikationen sollen möglichst gut auffindbar sein.

Dafür sind mindestens die folgenden Maßnahmen umzusetzen:

- Verwendung von semantischen Markup und einer konsistenten Überschriftenhierarchie
- Verwendung von itemprop values
- Suchmaschinenoptimierter Seitentitel (auch für News und andere durch TYPO3 generierte Detailansichten)
- Suchmaschinenoptimierte Alternativtexte für Bilder
- Lesbare, suchmaschinenoptimierte URLs (flache Pfadstrukturen)
- (Möglichkeit zur Eingabe von) Metatags
- Anlegen eines Accounts bei den Google-Webmastertools
- Google-XML-Sitemap anlegen und bei Google anmelden
- Interne Suche (und andere nicht zur Indexierung geeignete Seiten) von Indexierung ausschließen

Bei Umbau/Relaunch eines bestehenden Projekts ist darauf zu achten, passende Weiterleitungen anzulegen, um vorhandene externe Links weiterhin zu unterstützen. Entsprechende Überlegungen sind im Feinkonzept darzulegen.

1.4 Sicherheitsrichtlinien und Datenschutz

Umgang mit Daten

Personenbezogene Daten, auf die während einer laufenden Entwicklung Zugriff gewährt wird, sind mit besonderer Sorgfalt und nach geltenden Datenschutzbestimmungen zu behandeln. Sie dürfen ohne Zustimmung der dena nicht an Dritte weitergegeben werden.

Datenbankauszüge von Webprojekten dürfen zu Entwicklungszwecken nur in bereinigter Form aus der Hosting-Umgebung ausgeführt werden. Der Hoster stellt für diesen Zweck bereinigte Kopien der Datenbank zur Verfügung. Der Dienstleister stellt dazu dem Hoster eine Liste der Datenbanktabellen zur Verfügung, welche persönliche Daten (Sessions, Anmeldenamen, E-Mail-Adressen, IP-Adressen etc.) enthalten und die bei der Bereinigung ausgeschlossen werden sollen.

In Ausnahmefällen dürfen nach Freigabe durch die dena vollständige Datenbankauszüge ausgeführt werden. Die Freigabe, der Zweck, der Verbleib und die Löschung der Auszüge müssen in geeigneter Form dokumentiert werden.

Werden Daten während der Entwicklung auf eine Entwicklungsumgebung des Dienstleisters transferiert, sind diese Daten nach Abschluss des Projekts zu löschen. Der Dienstleister hat insbesondere sicherzustellen, dass die Daten vor dem Zugriff durch Dritte geschützt werden.

Besonders sensible Daten (Bankverbindungen, Kreditkartendaten u. ä.) dürfen ausschließlich verschlüsselt über das Internet transferiert und in der Datenbank gespeichert werden.

Mit dem Dienstleister ist eine Vereinbarung zur Auftragsdatenverarbeitung gem. § 11 BDSG zu treffen.

Google Analytics

Der Einsatz von Google Analytics ist möglich und auch der Einsatz anderer Webanalyse-Anbieter ist erlaubt, wenn diese die Datenschutzrichtlinien der dena und die Vorgaben des deutschen Datenschutzrechts erfüllen. Die Auswertung muss in Google Analytics für jede Website separat angelegt werden. Die Datenschutzerklärung der entsprechenden Website ist anzupassen und die Speicherung der IP-Adressen muss anonymisiert werden (vgl. https://www.dena.de/datenschutz/). Zur Einrichtung einer Google Analytics Property wenden Sie sich an online@dena.de.

1.4 Hinweise zum Aufbau von Dokumentationen

Die im Folgenden beschriebenen Dokumentationen müssen zur Abnahme des Gesamtprojekts vorliegen.

Projekt-Dokumentation

Folgende Aspekte müssen im Wiki-Bereich des Projektrepositories nach Abschluss des Projektes festgehalten werden.

Projektname und dena-Projektnummer

Auftragsnummer

Besondere Anforderungen

- Vom dena-Standard abweichende Komponenten für TYPO3
- Sonderkonfiguration auf dem Server

Ansprechpartner

- Projektleiter
- Ggf. Technischer Ansprechpartner

Verweise auf Systemhandbuch und ggf. andere Dokumentationen

Systemhandbuch

Das Systemhandbuch enthält alle Informationen zum Aufbau des Webauftritts und soll vom erstellenden Dienstleister unabhängige Wartung und Weiterentwicklung des Systems ermöglichen. Die Dokumentation muss die folgend aufgelisteten Kapitel enthalten. Für projektspezifische Besonderheiten können weitere Kapitel erforderlich sein.

Bei der Konzeption des Projekts muss eine entsprechende Struktur für die Dokumentation erarbeitet werden. Die Sprache des Systemhandbuchs ist Deutsch.

TYPO3-Konfiguration

Struktur der Konfiguration (Aufbau des TypoScripts)

- Mehrsprachigkeit
- Multidomain-System
- Microsites
- Landing pages
- Separate mobile Version (falls existent)

Server

- Besondere Anforderungen an den Server (falls vorhanden)
    - PHP-, MySQL-Version, Memory_limit
    - Zusätzliche (PHP-)Bibliotheken
- Cron-Jobs (falls vorhanden)

Verwendete Domains / Domainweiterleitungen

- Domain-Provider (falls nicht I/P/B)
- Ansprechpartner

Frontend

- Verwendetes Layout-Framework
- JavaScript-Bibliotheken (zu beachten: Liste der erlaubten JS-Frameworks und –Bibliotheken (siehe Abschnitt 12.3 Javascript))

Backend

- Eingesetzte Techniken (FLUIDTEMPLATE, TemplaVoila, etc.)
- Konfiguration

Funktionen / Extensions

- Liste und Beschreibung aller Extensions, die nicht im Standardumfang von TYPO3 enthalten sind. Zu dokumentieren sind die Einsatzorte und besondere Konfigurationen
- Selbst entwickelte Extensions müssen zusätzlich separat dokumentiert werden (siehe Abschnitt 9.3 Extensions-Dokumentation).

Benutzerverwaltung

- Benutzergruppen

Geschützte Bereiche (falls vorhanden)

- Registrierung, Login
- Frontend-Benutzergruppen

SEO-Maßnahmen

- Besondere Maßnahmen zu SEO/SEM
- System- Konfiguration
- Google AdWords Kampagne (falls vorhanden)

Schnittstellen (falls vorhanden)

- Auflistung
- Beschreibung (Format, Struktur)
- Ansprechpartner

Systemhandbuch-Ergänzung

Erforderliche Informationen bei der Umsetzung eines Bereiches innerhalb eines bereits bestehenden Webauftrittes.

Allgemeines

- Ziel der Erweiterungen
- Neue Funktionen
- Verwendete Frameworks und Bibliotheken
- Inkl. Versionen

Konfiguration

- Einstellungen Extension Manager
- Repository für Templates / CSS / HTML
    - Extensions
    - Fileadmin
    - Includes via TS
    - TypoScript-Parameter
    - Standardeinstellungen überschreiben

Administration

- Vorhandene Cron-Jobs (inkl. Beschreibung ihrer Funktion)
- Spezielle Benutzerrollen und -rechte
- Registrierung (falls vorhanden)
- Regelmäßige Datenkontrolle
- Zu beachten
    - Transfer zu anderem System
    - Systemaktualisierungen
    - Vorhandene Schnittstellen
    - Erforderliche Authentifizierung (z. B. Verwendung von Google-Key)

Extensions-Dokumentation

Für alle im System eingesetzte Extensions müssen Handbücher vorhanden sein. Die Handbücher müssen in einem offenen Format (z.B. Markdown) im Quelltext der Extension geliefert werden.

Für Extensions aus dem TYPO3 Extension-Repository stammen, kann dort vorhandene Dokumentation verwendet werden, so sie entsprechende Informationen wie unten beschrieben enthält (<a href="https://docs.typo3.org" class="uri external-link">;).

Falls keine oder keine aussagekräftige Dokumentation vorhanden ist muss nach der Implementierung der Extension ein Leitfaden mit Screenshots und Hinweisen zur Konfiguration erstellt und im Systemhandbuch referenziert werden.

Für neu programmierte Extensions muss ein Handbuch mit folgenden Themen mitgeliefert werden. Die Struktur orientiert sich an der bei TYPO3 gebräuchlichen Struktur für die Dokumentation von Extensions. Die Dokumentation kann auf Deutsch oder Englisch verfasst werden.

EXT: Name der Extension (ext key)

Allgemeines

- Kurze Erklärung: Was macht die Extension?

Screenshots

Installation

- Hinweise: was ist bei der Installation der Extension zu beachten
- Abhängigkeiten
- Konflikte

Konfiguration

- Auflistung TypoScript-Parameter (Konstanten, Setup)
- Beispiele zum Auslagern von Vorlagen

Administration

- Hinweise zur Einstellungen im Extension-Manager
- Verwaltung vorhandener Cron-Jobs (falls zutreffend)
- Weitere regelmäßige Adminaufgaben

Benutzer-Leitfaden

- Beschreibung (in nicht-technischer Sprache) der Verwendung der Extension
    - Erstellen von Datensätzen (falls zutreffend)
        - Erfassungsmaske mit einzelnen Feldern
    - Anlegen von Plugins (falls zutreffend)
        - Einstellungen in Plugins (Flexform)
    - Daten- und Pluginslokalisierung

Redakteursleitfaden

Das Redakteurshandbuch soll in nicht-technischer Sprache das Arbeiten mit dem TYPO3-System des konkreten Projekts erklären. Der Anwender muss mit Hilfe dieses Handbuches in der Lage sein sich selbständig in das System einarbeiten zu können. Später kann das Dokument auch als Nachschlagwerk benutzt werden.

Es kann davon ausgegangen werden, dass die grundlegende Bedienung von TYPO3 bekannt ist. Das Layout des Backends sowie Basisfunktionen (Seiten anlegen, Standardinhalte anlegen) müssen nicht beschrieben werden. Stattdessen soll auf die Besonderheiten der konkreten TYPO3-Instanz (Konfiguration, eigenen Extensions, etc.) eingegangen werden.

Folgende Themen müssen im Handbuch erläutert werden:

Verwaltung von Datensätzen im Backend

- Erläuterung Felder in der Erfassungsmaske
- Speicherort der Daten

Verwendung von (Nicht-Standard-)Plugins auf der Seite

- Verwendung des Wizards (falls vorhanden)
- Optionen im Eingabeformular und ihre Verwendung

- Beispiele der Darstellung im FrontendListe, Details

Wichtiges zu beachten
Evtl. erforderliche weitere Einstellungen

Schnittstellen-Dokumentation

Falls im Rahmen des Projekts Schnittstellen entstehen, müssen Aufbau und Format der Daten und die Austauschprozeduren beschrieben werden.

Datenimport via CSV oder TXT

Allgemeine Anforderungen

- Format (csv, xml)
- ggf. Trennzeichen

Dateiaufbau

- Beschreibung der Inhalte (Zeilen)
- Zeilenumbruch
- Spalteninhalt

Datensatzaufbau

- Felder-Trennung

Dokumentation einer Beispieldatei

Wie erfolgt der Import/Export (ggf. Verweis auf Extension-Dokumentation)

Webservice-Schnittstellenbeschreibung

- Allgemeine Beschreibung der Funktion (wer fragt wen für welche Daten an?)
- Format (z.B. JSON, XML, etc.)
- Verfügbare Befehle und Reaktionen
- Datensatzaufbau der Übergabedaten

1.5 TYPO3 ELTS

TYPO3 bietet erweiterten Support für Versionen, bei denen der "Long Term Support" ausgelaufen ist (*ELTS - Extended Long Term Support*).

Es werden Sicherheits-Updates für diese Versionen veröffentlicht. Die Quellen sind nur für Abonnenten verfügbar.

Für die DENA-Projekte wurde ein entsprechender Supportvertrag abgeschlossen.

Setup

Die ELTS-Version werden als Pakete im Composer-Verzeichnis elts.typo3.com bereitgestellt. Der Zugang ist beschränkt (http basic auth).

Folgende Schritte sind notwendig, um das Projekt für die Verwendung der ELTS-Version einzurichten:

1. Zugangsdaten

Benutzername und token werden @fr.it-support bereitgestellt. Die Zugangsdaten sind vertraulich. Sie dürfen **nicht veröffentlicht** werden und **nicht ins Versionskontrollsystem (git)** gelangen!

2. Projekt Bundle anpassen

- `auth.json` zu `.gitignore` im bundle hinzufügen
- im Wurzelverzeichnis des bundles auth.json mit folgendem Inhalt erstellen:

```
json { "http-basic": { "elts.typo3.com": { "username": "<Benutzername>",
"password": "<token>" } } } ` * composer Verzeichnis in bundle-`composer.json
```

hinzufügen:

```
```json
  "repositories": [
    {
      "type": "composer",
      "url": "https://elts.typo3.com/"
    },
    {
      "type": "composer",
      "url": "https://composer.typo3.org/"
```
```

```
    }
  ]
```

Die Reihenfolge ist wichtig. Pakete, die in einem frühen Repo gefunden werden, werden in späteren nicht mehr gesucht.

- `build job in .gitlab-ci.yml` erweitern

```
build:
  stage: build
  script:
    - composer self-update
    - composer config --auth $COMPOSER_ELTS_AUTH
    - composer install $COMPOSER_COMMAND_PARAMETERS
```

Die Zugangsdaten für die Pipelines sind als Variable COMPOSER_ELTS_AUTH für alle Projekte in git.dena.de hinterlegt.

2. Setup & Deployment

2.1 Workflow

Projektverlauf

Neu- und Weiterentwicklung sollen dem Modell der kontinuierlichen Integration und Auslieferung folgen.

Dafür **müssen** neue Versionen der Software in kleinen Iteration ausgeliefert werden. Das Zielsystem (LIVE) muss während eines Entwicklungsprojektes kontinuierlich aktualisiert werden, um Fehler bei der Integration oder Konfiguration frühzeitig zu bemerken.

Für neue Projekte wird das LIVE-System bereits zu Projektbeginn eingerichtet und erst beim Live-Gang öffentlich erreichbar gemacht.

Entwicklungszyklus

Die Arbeit an Webprojekten der dena soll folgendem allgemeinen Schema folgen:

- 1. in lokalem System entwickeln und integrieren
- 2. auf STAGING abnehmen
- 3. auf LIVE ausliefern

lokale Entwicklung

Alle Änderungen **müssen** in lokalen Entwicklungssystemen umgesetzt werden.

Manuelle Änderungen am Softwarestand und der Konfiguration des LIVE-Systems **dürfen nicht** vorgenommen werden.

Sämtliche Konfiguration **muss** lokal stattfinden und im Versionskontrollsystem eingecheckt werden. Einstellungen, die umgebungsabhängig sind, müssten programmatisch auf die Umgebung reagieren (siehe Konfiguration).

Abnahme auf STAGING

Änderungen an der Software **müssen** vor dem Ausliefern auf LIVE auf dem STAGING-System zur Qualitätssicherung und Abnahme bereitgestellt werden. Nach erfolgter Qualitätssicherung kann in Absprache mit den Projektansprechpartnern die Abnahme intern beim Dienstleister erfolgen.

Ausliefern auf LIVE

Auf LIVE darf in der Regel nur nach Absprache mit den Projektverantwortlichen ausgeliefert werden. Bei Bedarf sollen Wartungsfenster eingehalten werden. Während vorher vereinbarter Wartungsfenster könnten die Zugänge zum Redaktionssystem deaktiviert werden.

Tickets

Aufgaben und Fehler werden als Tickets in Gitlab erfasst und verwaltet. Bei Änderungen werden Nachrichten an die Beteiligten versendet. Tickets können auch per E-Mail erstellt und in Gitlab bearbeitet werden.

2.2 Projektstruktur

Die Struktur aller Projekte ist einheitlich vorgegeben. Dadurch werden die Pflege und Wartung erleichtert und die automatische Auslieferung ermöglicht.

Alle Pakete werden entweder im vendor- Verzeichnis oder im Verzeichnis für Erweiterungen / Plugins installiert. Das vendor-Verzeichnis befindet sich außerhalb des Web-Verzeichnisses (web root). Sensible Konfigurationsdaten befinden sich ebenfalls außerhalb des Web-Verzeichnisses.

Im Bundle des Projektes werden nur die Dateien zur Steuerung des Bauprozesses, der Auslieferung sowie die Projektdokumentation gespeichert.

Der vendor-Ordner **darf nicht** im Repository eingecheckt werden.

Konfigurationsdateien **müssen** vollständig sein. Sie **dürfen keine** Geheimnisse (Token, Passwörter, Salt für Verschlüsselung etc.) enthalten.

TYPO3 - Struktur

Dateien im Git-Repository

```
├── .ddev                  # optional Konfiguration für lokale Entwicklungsumgebung
├── .deployment
│   ├── config             # obligatorisch
│   │   └── deploy
│   │       ├── live.rb    # obligatorisch spezifische Konfiguration für Deployment zu LIVE-Umgebung
│   │       └── staging.rb # obligatorisch spezifische Konfiguration für Deployment zu STAGING-Umgebung
│   ├── deploy.rb          # obligatorisch Steuerdatei für deployment-Tasks
│   └── ignore_tables.txt  # obligatorisch Liste der Tabellen, die nicht in bereinigte Datenbank-Dumps aufgenommen werden
│   ├── Capfile            # obligatorisch
│   ├── Gemfile            # obligatorisch
│   └── Gemfile.lock       # obligatorisch
├── .gitlab                # optional Gitlab Issue Templates
│   └── issue_templates
├── Documentation          # Beispiel
│   └── [...]
├── app
│   ├── config
│   │   └── sites           # Config für jede Site
│   │       └── [...]
│   └── web
│       └── typo3conf      # obligatorisch
│           ├── AdditionalConfiguration.php
│           └── LocalConfiguration.php
├── .gitignore             # obligatorisch
├── .gitlab-ci.yml         # obligatorisch
├── composer.json          # obligatorisch
├── composer.lock          # obligatorisch
├── ChangeLog              # optional
└── README.md              # obligatorisch
```

WordPress - Struktur

Dateien im Git-Repository

```
├── app
│   └── config
│       ├── environments
│       │   ├── development.php
```

```
│       │   ├── live.php
│       │   └── staging.php
│       └── application.php
│   ├── vendor_static
│   └── web
│       ├── app
│       │   ├── mu-plugins
│       │   │   ├── 000-sunrise.php
│       │   │   ├── bedrock-autoloader.php
│       │   │   ├── disallow-indexing.php
│       │   │   ├── index.php
│       │   │   └── register-theme-directory.php
│       │   ├── plugins
│       │   ├── themes
│       │   └── uploads
│       ├── index.php
│       └── wp-config.php
├── composer.json
├── composer.lock
├── .deployment
│   ├── Capfile
│   └── config
│       ├── deploy
│       │   ├── integration.rb
│       │   ├── development.rb
│       │   └── staging.rb
│       └── deploy.rb
├── Documentation
│   └── ...
├── .env.example
├── .env
├── fixtures
│   └── development
│       └── table-to-fix.sql
├── .gitignore
├── .gitlab-ci.yml
├── README.md
└── wp-cli.yml
```

Struktur nach dem Bauen des Projektes

```
├── app
│   └── config
│       ├── environments
│       │   ├── development.php
│       │   ├── live.php
│       │   └── staging.php
│       └── application.php
│   ├── vendor
│   ├── vendor_static
│   └── web
│       ├── app
│       │   ├── mu-plugins
```

```
│       │   │   ├── 000-sunrise.php
│       │   │   ├── bedrock-autoloader.php
│       │   │   ├── disallow-indexing.php
│       │   │   ├── index.php
│       │   │   └── register-theme-directory.php
│       │   ├── plugins
│       │   ├── themes
│       │   └── uploads
│       ├── wp
│       ├── .htaccess
│       ├── index.php
│       └── wp-config.php
├── composer.json
├── composer.lock
├── .deployment
│   ├── Capfile
│   └── config
│       ├── deploy
│       │   ├── integration.rb
│       │   ├── development.rb
│       │   └── staging.rb
│       └── deploy.rb
├── Documentation
│   └── ...
├── .env.example
├── .env
├── fixtures
│   └── development
│       └── table-to-fix.sql
├── .gitignore
├── .gitlab-ci.yml
├── README.md
└── wp-cli.yml
```

Hinweis: Die env.yml und .env Dateien enthalten die Zugangsdaten und Einstellungen für die **lokale** Umgebung. Sie dürfen nicht in die Versionskontrolle gelangen und werden nicht automatisch beim Bauen erzeugt. Sie sind nur der Vollständigkeit halber abgebildet.

2.3 Konfiguration

Aufbau der Anwendung

Anwendungen müssen so aufgebaut sein, dass sie ohne Änderung an der Software und ohne manuelle Bearbeitung der Einstellungen in unterschiedlichen Umgebungen laufen. Die Umgebungen LIVE und STAGING müssen unterstützt werden. Die Unterstützung weitere Umgebungen wie z.B. eine lokale Entwicklungsumgebung werden empfohlen.

Die Konfiguration wird über Umgebungsdateien ins System injiziert. (TYPO3: env.yml, WordPress .env).

Diese Konfigurationsdateien werden vom Hoster bereitgestellt. Sie enthalten "Secrets" sowie Konfigurationen für Caches und andere Dienste.

Die Konfigurationsdateien müssen von der Anwendung geladen werden.

TYPO3

Für TYPO3-Instanzen müssen dazu im Ordner `<Project Root>/app/web/typo3conf/` die Dateien

- `AdditionalConfiguration.php` (siehe Beispiel)
- `LocalConfiguration.php` (siehe Beispiel)

gepflegt werden.

In der `AdditionalConfiguration` werden die Werte aus der Umgebungsdatei geladen.
Die Werte der aus der Umgebungsdatei überschreiben die Einstellungen der `LocalConfiguration`.

2.4 Komponenten

Verzeichnisse

Alle Komponenten müssen entweder in langlebigen, weit verbreiteten Verzeichnissen (TER, Packagist, GitHub) gehosted sein, oder als Repository im dena-Gitlab gepflegt werden. Komponenten in privaten Repositories müssen in ein Repository im dena-Gitlab gespiegelt werden.

unerwünschte Komponenten

Zum Auslieferungszeitpunkt dürfen keine Komponenten installiert sein, die zu Entwicklungszwecken benötigt werden. Dazu gehören z.B. Übersetzungs- und Datenbankwerkzeuge und Komponenten für automatisierte Test.

Für das Installieren solcher Komponenten in einer Testumgebung - z.B. in Pipelines - darf eine eigene Konfiguration verwendet werden (z.B. Pakete aus der Sektion `require-dev` in der `composer.json` des Bundles.

Updates

TYPO3 und WordPress Core werden stets in der jeweils aktuellsten Version (Bugfix-Level) ausgeliefert.

Patches

Core-Patches sind nur in Ausnahmefällen erlaubt. Sie müssen als einzelne Dateien im Ordner `patches` hinterlegt und in der Sektion `extras` der `composer.json` beschrieben werden:

```
{
  "extra": {
    "patches": {
      "typo3/cms-core": {
        "Fix undocumented variable `foo` in bar.php": "patches/3156260-
11.patch"
      }
    }
  }
}
```

Der Grund für Anwendung jedes Patches muss im Bundle dokumentiert werden. Es sollte ein Link zu einem Ticket in einem Issue-Tracker angegeben werden, in dem das Problem behandelt wird (z.B. TYPO3-Forge).

2.5 Abhängigkeiten

Bundle

Alle Abhängigkeiten eines Projektes werden in der `composer.json` des Projekt-Bundles nach Semantischer Versionierung beschrieben. Die aktuell installierten Versionen müssen in der `composer.lock` festgeschrieben sein. Beim Bauen des Projektes in der Pipeline für das Ausliefern wird `composer install` aufgerufen. Dabei müssen die Versionen aus der `composer.lock`-Datei verwendet werden.

In der Bundle-`composer.json` werden für TYPO3-Projekte auch alle benötigten Core-Pakete aufgeführt. Dadurch kann die Datei `typo3conf/PackageStates.php` beim Bauen des Projektes geschrieben werden und alle Extensions werden installiert und **aktiviert**.

Zusätzlich zu den Anforderungen des Projektes werden *immer* die folgenden Pakete benötigt:

```
roave/security-advisories:dev-master
helhum/console
symfony/yaml
helhum/config-loader
```

Komponenten

Alle Komponenten müssen eine eigene `composer.json` - Datei enthalten. Diese kann weitere Abhängigkeiten beschreiben.

2.6 Datenbank

Datenbanken der Webprojekte müssen geschützt werden, da sie persönliche Daten der Nutzer/-innen enthalten.

bereinigte Auszüge

Bereinigte Datenbankauszüge werden automatisch täglich vom Hoster erstellt und unter

- LIVE: dumps.live.dena.de
- STAGING: dumps.stage.dena.de
- zum Herunterladen bereitgestellt.

Der Zugang zu dieser Seite ist nur möglich, wenn sie mit einer der privilegierten IP-Adressen aufgerufen werden (z.B. Büro oder VPN der Agentur).

Zum Bereinigen werden bestimmte Tabellen ausgeschlossen. Die Anwendungsentwickler/-innen sind verpflichtet, für den Schutz persönlicher Daten zu sorgen. Dazu muss eine Liste der auszuschließenden Tabellen in `.deployment/config/ignore_tables.txt` hinterlegt werden.

Siehe: Beispiel `ignore_tables.txt`

vollständige Datenbanken

Vollständige Kopien der Datenbanken werden für jedes Webprojekt automatisch täglich vom Hoster erstellt. Diese dienen als Sicherheitskopien und dürfen **nicht** zu Entwicklungszwecken aus dem System ausgeführt werden.

In begründeten Ausnahmefällen kann eine vollständige Kopie der Datenbank vom Hoster angefordert werden.

Zweck, Freigabe und Dauer der Nutzung, sowie die Löschung der Kopie müssen z.B. im Ticketsystem dokumentiert werden.

Über die Freigabe entscheidet der Projektansprechpartner der dena.

2.6 Auslieferung

Übersicht

Die Auslieferung der Webanwendungen erfolgt automatisch. Alle Aufgaben im Zusammenhang mit der Auslieferung müssen skriptgesteuert erfolgen.

Vorlagen für diese Skripte werden von einem Dienstleister der dena bereitgestellt. Sie müssen von den Entwickler/-innen der Anwendung auf die Bedürfnisse des Projektes angepasst werden.

Derzeit ist die Auslieferung als Pipeline mithilfe von Capistrano-Tasks umgesetzt. Für Anwendungsentwickler/-innen besteht kein direkter Zugang (via SSH oder SFTP) zu den Systemen.

Zukünftige Projekte werden mithilfe von Helm in einen Kubernetes-Cluster ausgeliefert. (siehe Kubernetes-Deployment).

Für Anwendungsentwickler/-innen besteht kein direkter Zugang (via SSH oder SFTP) zu den Systemen.

Alle Zugangsdaten für die Zielsysteme werden vom Hoster in einer Umgebungsdatei (env.yml) oder in Kubernetes-Secrets bereitgestellt.

Die Anwendung muss diese auslesen und die Zugangsdaten und weitere umgebungsabhängige Variablen entsprechend der aktuellen Umgebung verwenden.

Zielsysteme

Das Hauptsystem jeder Anwendung wird *LIVE* genannt.

Für jede Anwendung *können* zusätzliche Instanzen bereitgestellt werden. Jede Instanz entspricht einem Zweig im Versionskontrollsystem. Der aktuelle Stand des jeweiligen Zweiges wird ausgeliefert, wenn alle Tests fehlerfrei durchlaufen wurden.

Bezeichnung	Zweig	Pflicht	Beschreibung
LIVE	main (master)	Ja	aktuelle Version der Anwendung
STAGING	release/*	Ja	Abnahmesystem für die nächste Version - für den Kunden
INTEGRATION	develop	Nein	Integrationssystem - für Entwickler, optional, wird nicht von dena bereitgestellt
DEVELOPMENT	develop	nein	lokales Entwicklungssystem - optional

2.7 Auslieferung mit Capistrano

Konfiguration

Erforderlich:

Das Projekt-Bundle * `.deployment`-Ordner mit `config` Ordner für Capistrano * Pipeline-Konfigurationsdatei `.gitlab-ci.yml` * Zugangsdaten in CI-System hinterlegt * Zugangsdaten auf Zielsystem ausgeliefert

`.deployment` - Ordner

Jedes Projekt enthält einen Ordner `.deployment`, in dem Skripte für die Auslieferung und Konfigurationen für jedes Zielsystem gepflegt werden.

```
.deployment
├── Capfile
├── Gemfile
```

```
├── Gemfile.lock
└── config
    ├── deploy
    │   ├── live.rb          # Konfiguration für Auslieferungsziel LIVE
    │   ├── development.rb   # Konfiguration für Auslieferungsziel INTEGRATION
(optional)
    │   └── staging.rb       # Konfiguration für Auslieferungsziel STAGING
    ├── deploy.rb            # Auslieferungsskript (Capistrano - von fr:digi-
tal)
    └── ignore_tables.txt    # Liste der Tabellen, die aus bereinigten Daten-
bank-Dumps ausgeschlossen werden müssen
```

.gitlab-ci.yml

Enthält die Konfiguration für die Pipeline des CI/CD-Servers. Beispiel: siehe Examples/GitlabCIYaml.md

Zugangsdaten in CI-System

Beim Bauen des Systems werden Zugangsdaten für private Versionskontrollsysteme benötigt. Beim Ausliefern wird ein spezieller Zugang zu den Zielsystemen verwendet. Die benötigten Zugangsdaten werden durch die Administratoren auf Gruppenebene im CI-System hinterlegt.

Zugangsdaten in Zielsystem

Auf dem Zielsystem werden folgende Zugangsdaten hinterlegt:

TYPO3

```
Backend:
  - installToolPassword
Datenbank:
  - Hostname
  - Benutzername
  - Passwort
  - Datenbankname

System:
  - encryptionKey
```

WordPress

```
Datenbank:
 - Datenbankname
 - Benutzername
 - Passwort
 - Hostname
```

```
  - Charset
  - Kollation (optional)

Authentifizierung:
   - AUTH_KEY
   - SECURE_AUTH_KEY
   - LOGGED_IN_KEY
   - NONCE_KEY
   - AUTH_SALT
   - SECURE_AUTH_SALT
   - LOGGED_IN_SALT
   - NONCE_SALT
```

2.8. Hosting und Auslieferung im Kubernetes-Cluster

Übersicht

Neue Projekte können in einem Kubernetes-Cluster gehostet werden. Dieser wird vom Hoster Swiss-IT-Security bereitgestellt.

Dadurch soll der Betrieb und die Einrichtung von Projekten vereinfacht werden. Die Architektur von Webprojekten wird flexibler, benötigte Dienste können leichter eingerichtet werden. Verschiedene Projekte können besser von einander isoliert werden und die Zuordnung von Ressourcen wird erleichtert.

Voraussetzungen

Anwendung kann gebaut werden

Beim Bauen der Anwendung werden alle benötigten Softwarepakete aus vertrauenswürdigen und stabilen Quellen installiert. Der Prozess läuft ohne manuellen Eingriff ab. Der Zustand der Anwendung nach dem Bauen ist determiniert und reproduzierbar.

Für PHP-Projekte der dena kommt Composer zum Einsatz.

Anwendung ist im Container lauffähig

Damit die Webanwendung im Cluster laufen kann, muss sie definierte Schnittstellen zu Diensten verwenden und in unterschiedlichen Umgebungen betrieben werden können (z.B. lokale Entwicklung, Test, Staging, Live).

Umgebungsabhängige Werte wie Konfigurationen und "Secrets" werden außerhalb der Anwendung verwaltet und zur Auslieferungszeit an den Container übergeben.

Anwendung ist im Cluster bereitgestellt

Für jede Anwendung werden Bereiche und Ressourcen im Cluster bereitgestellt. Diese müssen vor der Einrichtung abgestimmt werden. Die Abstimmung der nötigen Daten erfolgt mit Swiss IT über Tickets im Ticketsystem (git.dena.de).

Die Anwendung erhält einen eindeutigen Namen und technische ID. Es werden mindestens Namensräume für die beiden obligatorischen Umgebungen STAGING und LIVE eingerichtet. "Secrets" für den Zugriff auf den Cluster und Dienst werden vom Hoster im Cluster eingerichtet.

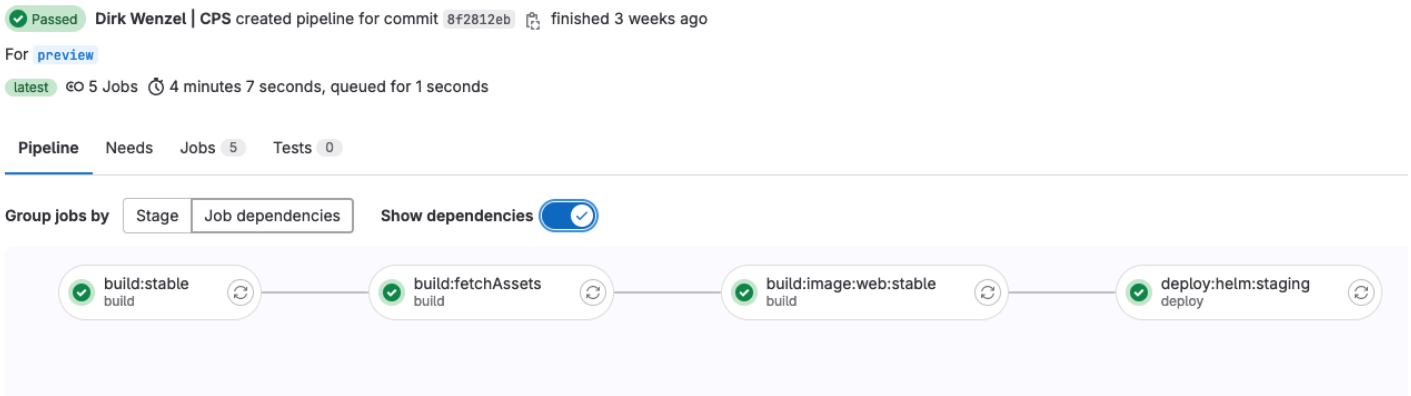
Komponenten

Komponente	Aufgabe
Kubernetes-Cluster	Stellt Ressourcen und Dienste für den Betrieb zur Verfügung, erlaubt deren Orchestrierung
Composer	erlaubt das Bauen einer PHP-Anwendung aus deren Abhängigkeiten
Gitlab-CI-Pipeline	orchestriert das Bauen, Testen und Ausliefern der Anwendung in mehreren Schritten (Jobs)
Helm	Paketmanager für Kubernetes, interagiert bei der Auslieferung mit Cluster
Helm-Chart	vorgefertigtes Paket aus Anwendungen und Diensten
Docker	isoliert Anwendungen in virtualisierte Container

CI-Pipeline

Für das Bauen und Ausliefern der Anwendung kommt eine Pipeline in Gitlab-CI zum Einsatz. Diese Pipeline wird von den Entwicklerinnen des Projektes eingerichtet.
Sie besteht aus mehreren Jobs. Es stehen vorgefertigte Jobs im Projekt common/deploy/gitlab-ci-config zur Verfügung.

Übersicht



Es werden mindestens die folgenden Jobs benötigt:

Job Name	Beschreibung
build:stable	Baut die Anwendung inklusive aller Abhängigkeiten.
build:image:web:stable	Baut ein neues Container-Image anhand eines Dockerfiles.
deploy:helm:live	Liefert die gebaute Anwendung in Kubernetes aus.

Jobs im Detail

Anwendung bauen - build:stable

Dieser Job baut die Anwendung mittels composer install. Diese müssen in der composer.json beschrieben und in der composer.lock festgelegt sein. Es gelten die Regeln für Abhängigkeiten.

Beispiel Ausgabe (gekürzt):

```
$ composer install $COMPOSER_PARAMS
Installing dependencies from lock file
Verifying lock file contents can be installed on current platform.
Package operations: 152 installs, 0 updates, 0 removals
  - Installing typo3/cms-composer-installers (v3.1.3): Extracting archive
  - Installing typo3/class-alias-loader (v1.1.4): Extracting archive
  - Installing cpsit/auditor (0.6.1): Extracting archive
  - Installing symfony/polyfill-mbstring (v1.29.0): Extracting archive
  - Installing typo3fluid/fluid (2.10.1): Extracting archive
  [...]
```

Image bauen - build:image:web:stable

Erstellt ein neues Container-Image anhand eines Dockerfiles. Als Basis wird ein vorgefertigtes Image (z.B. registry.dena.de:4567/common/deploy/container-images/typo3v11:0.2.0) verwendet. Dieses enthält einen vorkonfigurierten Webserver und Skripte für die Initialisierung der Anwendung sowie "post-deploy"-Skripte. (z.B. Datenbank-Update, Migrations-Wizards, Cache leeren …).

Kopiert die Quelldateien aus build:stable in dieses Image. Das Image wird mit einem Tag versehen und in der Container-Registry des Projektes gespeichert.

Ausliefern zu Kubernetes

Liefert die gebaute Anwendung und in Kubernetes aus. Ein vorgefertigtes Helm-Chart beschreibt die benötigten Dienste (z.B. Redis, Solr, Datenbank). Dabei werden spezifische Werte für die Zielumgebung angewendet. Es stehen Helm-Charts für verschiedene TYPO3-Versionen zur Verfügung. Sie beschreiben einen vollständigen Anwendungsstack und werden durch Konfigurationsdateien für die jeweilige Anwendung und Umgebung variiert (z.B. values.live.yaml).

Wenn die Anwendung im Cluster noch nicht existiert, wird sie erstellt. Wenn sie bereits existiert, wird sie aktualisiert. Dabei werden Änderungen an den values.*.yaml-Dateien berücksichtigt und das im vorherigen Schritt erzeugte Container-Image verwendet.

Das Ausliefern erfolgt atomar. D.h., wenn ein Fehler auftritt, wird das gesamte Deployment auf den vorherigen Zustand zurückgerollt.

Es findet ein Rolling Update statt. D.h. es wird jeweils ein "Pod" nach dem anderen aktualisiert. Per Default sind für LIVE-Umgebungen drei Pods parallel aktiv, für STAGING zwei.

Konfiguration

Von den Anwendungsentwicklern müssen die nötigen Konfigurationen in der Steuerdatei für Gitlab-CI (.gilab-ci.yaml) erstellt werden. Folgende Schritte sind notwendig:

1. Umgebungsvariablen für Pipeline in gitlab-ci.yaml eintragen.
2. vorgefertigte Jobs aus common/deploy/gitlab-ci-config inkludieren
3. Dockerfile aus Vorlage kopieren

For in-depth examples see Example/DeployHelm.

2.9 Versionskontrolle

Alle Anwendungen und Komponenten werden im Versionskontrollsystem GitLab erfasst. Es werden drei Typen unterschieden:

- Bundle
- Komponente
- Bibliothek

Für *Bundle* und *Komponenten* für Webanwendungen der DENA kommen git Repositories zum Einsatz. *Bibliotheken* können als beliebige installierbare Pakete geliefert werden, wenn sie automatisch aus öffentlich zugänglichen Verzeichnissen geladen werden können. *Bibliotheken* müssen aktiv gepflegt werden. D.h. es müssen zeitnah Fehler behoben werden. Verwaiste Bibliotheken sollten ersetzt werden oder müssen als Fork in die Versionskontrolle der DENA übernommen werden.

Gepatchte Versionen und *Forks* von Bibliotheken dürfen nur im Ausnahmefall und nach Absprache mit DENA und den technischen Projektleitern verwendet werden. Die Pflege dieser Varianten bedeutet zusätzlichen Aufwand in der Zukunft und erhöht das Risiko von Versionskonflikten.

Für die kontinuierliche Auslieferung wird die Anwendung eines bestimmten Branching-Musters, des Git-Flow Modells in der Versionskontrolle vorausgesetzt.

Das Tagging ist nach dem Standard Semantic Versioning 2.0.0 vorzunehmen.

Die Zuordnung wird im Abschnitt Auslieferung/Zielsysteme beschrieben.

2.10 Infrastruktur

Infrastruktur SITS Deutschland

Es gibt sowohl eine Infrastruktur auf VM Basis mit neuen Linux Systemen und vHosts für die jeweiligen Auftritte als auch eine Kubernetes-Infrastruktur für Container basierte Deployments.

Ein direkter Zugang zu den Servern (Web/DB) via SSH oder SFTP ist nicht vorgesehen.

Reguläre Wartungen sowie Security Updates werden durch den Hoster eingespielt.

Wiederkehrende Aufgaben wie das Anlegen und Löschen von virtuellen Hosts erfolgt ebenfalls skriptgesteuert.

Es werden zunächst zwei Umgebungen eingerichtet: STAGING und LIVE.

Kern-Infrastruktur für alle Umgebungen

Komponente	Bezeichnung	OS Version	App Version	Umgebung	Beschreibung
Firewall Cluster	ForcePoint NGFW	-		zentral	Managed Firewall
LoadBalancer Cluster	Virtual Traffic Manager	-		zentral	Managed LoadBalancer
dena-gitlab	Gitlab	Ubuntu 22.04	gitlab-ee 16.x	zentral	
git-runner01	Gitlab-Runner-1	Ubuntu 22.04	gitlab-runner 20.04	zentral	2 runner containers per VM
git-runner02	Gitlab-Runner-2	Ubuntu 22.04	gitlab-runner 20.04	zentral	2 runner containers per VM
prtg-probe	Monitoring-Probe	Windows 2022		zentral	Infrastruktur Monitoring

LIVE

Nach Möglichkeit sollen die neuen PHP8 Webserver und MySQL 8 Systeme genutzt werden. Für ältere Baukastenseiten steht noch eine PHP7 Umgebung mit MySQL 5.7 Datenbank und älterer Solr Version zur Verfügung.

Komponente	Bezeichnung	OS Version	App Version	Umgebung
web01-php8-prod.typo3	Apache/PHP	Ubuntu 22.04	apache2/php8.3	Prod
db01-prod.typo3	MySQL DB Master	Ubuntu 22.04	mysql-server-8.0	Prod
db02-prod.typo3	MySQL DB Slave	Ubuntu 22.04	mysql-server-8.0	Prod
web01-php7-prod.typo3	Apache/PHP	Ubuntu 22.04	apache2/php7.4-zend	Prod
solr01-prod	Solr	Ubuntu 22.04	ext-solr:9.0	Prod
db03-prod.typo3	MySQL DB Master	Ubuntu 22.04	mysql-server-5.7	Prod
db04-prod.typo3	MySQL DB Slave	Ubuntu 22.04	mysql-server-5.7	Prod

STAGING

Nach Möglichkeit sollen die neuen PHP8 Webserver und MySQL 8 Systeme genutzt werden. Für ältere Baukasten-seiten steht noch eine PHP7 Umgebung mit MySQL 5.7 Datenbank und älterer solr Version zur Verfügung.

Komponente	Bezeichnung	OS Version	App Version	Umgebung
web02-php8-stage.typo3	Apache/PHP	Ubuntu 22.04	apache2/php8.2	Stage
db01-stage.typo3	MySQL DB	Ubuntu 22.04	mysql-server-8.0	Stage
web01-php7-stage.typo3	Apache/PHP	Ubuntu 22.04	apache2/php7.4	Stage
solr02-stage	Solr	Ubuntu 22.04	ext-solr:9.0	Stage
db02-stage.typo3	MySQL DB	Ubuntu 22.04	mysql-server-5.7	Stage

Kubernetes Infrastruktur

Kubernetes Cluster werden durch den Hoster SITS Deutschland über ein SUSE Rancher System bereitgestellt und verwaltet. Es gibt hier jeweils ein staging sowie produktiv Kuberenetes Cluster bereit. Deployments werden über Gitlab ausgespielt, externe Zugriffe gehen über die zentrale Firewall und Load Balancer Infrastruktur.

Komponente	Bezeichnung	OS Version	App Version	Umgebung	Beschreibung
dena-k8s-mgmt	Kubernetes Management	Ubuntu 22.04		Kube Prod/Stage	SITS Management System
kube-150-dena-stage-a(X)	3 Kubernetes Nodes	Ubuntu 20.04	K8s 1.26	Kube Stage	
kube-150-dena-a(X)	3 Kubernetes Nodes	Ubuntu 20.04	K8s 1.26	Kube Prod	

2.11 Hosting

dena-Webprojekte werden in der Regel auf einem System des Hosting-Dienstleisters der dena gehostet.

In Ausnahmefällen können andere Hoster zum Einsatz kommen. Diese müssen sicherstellen, dass die Auslieferung mit den von der dena bereitgestellten Mechanismen erfolgt. Dazu ist mindestens ein SSH-Zugang für den Deployment-Benutzer notwendig.

Das Dateisystem muss den Vorgaben des Dena-Hostings entsprechen. Es müssen mindestens dieselben Dienste bereitgestellt werden. Die Konfiguration der Webprojekte muss entsprechend den Vorgaben in diesem Dokument erfolgen, sodass sie jederzeit mit minimalem Aufwand in die Hosting-Infrastruktur der dena portiert werden können.

Zusätzliche Aufwände, die durch abweichendes Hosting entstehen, müssen mit der dena abgestimmt und durch die dena freigegeben werden.

2.12 Legacy Projekte

Bestandsprojekte, die in die dena-Infrastruktur übernommen werden, müssen konsolidiert werden, damit sie den Anforderungen entsprechen.

Komponenten konsolidieren

Bei der Migration von Bestandsprojekten müssen vorhandene Komponenten konsolidiert werden, damit sie sicher betrieben und effizient gewartet werden können.

Mindestanforderungen

- `composer.json` vorhanden
- alle Abhängigkeiten in `composer.json` notiert (semantische Versionierung)
- Komponente kann *vollständig* über `composer install` installiert werden
- Informationen zu Autor/-innen, Firma und Lizenzmodell vorhanden
- Installationsschritte dokumentiert
- Verwendung dokumentiert

öffentliche Pakete

Bei Erweiterungen muss sichergestellt werden, dass eine aktuelle Version ohne bekannte Sicherheitsmängel aus öffentlichen Quellen (TER, Packagist, GitHub) installiert werden kann.

gepatchte Pakete

Bei Komponenten, die gepatcht wurden, ist im Einzelfall zu entscheiden, ob der Aufwand, gerechtfertigt ist, eine Abspaltung (*Fork*) zu pflegen.

nicht weiter entwickelte Pakete

Legacy-Komponenten (z.B. Konfigurationsdateien in Web-Root oder fileadmin-Ordner) werden entfernt oder in Versionskontrolle überführt.

Vorgehen

Legacy Code, der bisher z.B. im Ordner fileadmin gepflegt wurde, kann als Legacy-Package ausgeliefert werden.

Dazu sind folgende Schritte nötig:

Package Type

In der `composer.json` der Komponente wird ein Wert für den Typ gesetzt. Z.B.

```
"type": "legacy-fileadmin-library",
```

Installer

In der `composer.json` der Komponente wird der 'Composer Installer Extender' verlangt:

```
"require": {
    "oomphinc/composer-installers-extender": "^1.1"
},
```

Optional: Installer Name

In der `composer.json` der Komponente kann ein Name für die Installation des Paketes vergeben werden.

Im Beispiel wird das Paket `fr/biogasregister-fileadmin-lib` in den Ordner `biogasregister` (statt `biogasregister-fileadmin-lib`) installiert:

```
  "extra": {
    "installer-name": "biogasregister"
  },
```

Bundle: Installer und Pfad

In der `composer.json` des Projekt-Bundles werden Installer-Typen und Pfade verzeichnet:

```
  "extra": {
    "installer-types": [
      "legacy-fileadmin-library"
    ],
    "installer-paths": {
      "app/web/fileadmin/{$name}/": [
        "type:legacy-fileadmin-library"
      ]
    }
  }
```

2.13 Beispiele

bundle `composer.json`

title: Beispiel composer.json im Projekt bundle

Hinweise

- alle TYPO3-Pakete sind explizit aufgeführt
- `"repositories"` enthält zusätzliche öffentliche Verzeichnisse
- `"require-dev"` enthält `roave/security-advisories`. Dieses Paket verhindert, dass Komponenten mit bekannten Sicherheitslücken installiert werden
- `"require-dev"` enthält zwei Pakete in Entwicklungsständen
- Ordnerstruktur und Rechte werden durch Aufruf von `typo3cms install:fixfolderstructure` automatisch bereinigt

```
{
  "repositories": {
    "typo3": {
      "type": "composer",
      "url": "https://composer.typo3.org/"
```

```
    },
    "reporter": {
      "type": "vcs",
      "url": "git@github.com:dwenzel/reporter.git"
    },
    "reporter-api": {
      "type": "vcs",
      "url": "git@github.com:dwenzel/reporter-api.git"
    },
    "api-token": {
      "type": "vcs",
      "url": "git@github.com:CPS-IT/api-token.git"
    }
  },
  "type": "project",
  "authors": [
    {
      "name": "Dirk Wenzel",
      "email": "d.wenzel@familie-redlich.de"
    }
  ],
  "config": {
    "platform": {
      "php": "8.0"
    },
    "secure-http": true,
    "disable-tls": false,
    "vendor-dir": "app/vendor",
    "web-dir": "app/web"
  },
  "require": {
    "typo3/cms-adminpanel": "^11.5",
    "typo3/cms-backend": "^11.5",
    "typo3/cms-belog": "^11.5",
    "typo3/cms-beuser": "^11.5",
    "typo3/cms-core": "^11.5",
    "typo3/cms-extbase": "^11.5",
    "typo3/cms-filelist": "^11.5",
    "typo3/cms-filemetadata": "^11.5",
    "typo3/cms-fluid": "^11.5",
    "typo3/cms-fluid-styled-content": "^11.5",
    "typo3/cms-frontend": "^11.5",
    "typo3/cms-info": "^11.5",
    "typo3/cms-info-pagetsconfig": "^11.5",
    "typo3/cms-install": "^11.5",
    "typo3/cms-lowlevel": "^11.5",
    "typo3/cms-recordlist": "^11.5",
    "typo3/cms-reports": "^11.5",
    "typo3/cms-scheduler": "^11.5",
    "typo3/cms-setup": "^11.5",
    "typo3/cms-sys-note": "^11.5",
    "typo3/cms-tstemplate": "^11.5",
    "typo3/cms-viewpage": "^11.5",
```

```
    "helhum/typo3-console": "^7.1",
    "ext-pdo": "*",
    "dwenzel/reporter": "~0.4",
    "typo3/cms-introduction": "^4.4"
  },
  "require-dev": {
    "roave/security-advisories": "dev-latest",
    "dwenzel/reporter-api": "dev-develop as 0.4",
    "fr/api-token": "dev-feature/version11compatible as 1.8"
  },
  "extra": {
    "typo3/cms": {
      "cms-package-dir": "{$vendor-dir}/typo3/cms",
      "web-dir": "app/web"
    },
    "helhum/typo3-console": {
      "install-extension-dummy": "0"
    }
  },
  "scripts": {
    "package-states": [
    ],
    "folder-structure": [
      "php app/vendor/bin/typo3cms install:fixfolderstructure"
    ],
    "pre-deploy": [
      "# Scripts here will be executed after composer install"
    ],
    "post-autoload-dump": [
      "@package-states",
      "@folder-structure"
    ]
  },
  "name": "fr/typo3-10-test-bundle",
  "description": "Application bundle for typo3 11 test instance."
}
```

AdditionalConfiguration

??? warning "Hinweis" Das Beispiel stammt aus einer TYPO3-Installation der Version 9.5. Sie muss an die aktuelle Version angepasst werden.

```
<?php
$readers = [
    new \Helhum\ConfigLoader\Reader\EnvironmentReader('TYPO3', '_')
];
$absoluteConfigFilePath = getenv('ENV_FILE_PATH');
if (!empty($absoluteConfigFilePath)) {
    $fileReader = new \Helhum\ConfigLoader\Reader\YamlFileReader($absoluteCon-
figFilePath);
```

```
    $readers[] = $fileReader;
}

// We let the loader load context and environment specific configuration
// No other code must go in here!
$configLoader = new \Helhum\ConfigLoader\ConfigurationLoader(
    $readers,
    [
        new \Helhum\ConfigLoader\Processor\PlaceholderValue()
    ]
);
$config = $configLoader->load();

if(!getenv('ENV_FILE_PATH')) {
        $config['DB']['Connections']['Default']['dbname'] =
getenv('TYPO3_DB_NAME');
        $config['DB']['Connections']['Default']['host'] =
getenv('TYPO3_DB_HOST');
        $config['DB']['Connections']['Default']['password'] =
getenv('TYPO3_DB_PASS');
        $config['DB']['Connections']['Default']['user'] = getenv('TYPO3_DB_US-
ER');
        $config['BE']['installToolPassword'] = getenv('TYPO3_INSTALL');
        $config['SYS']['encryptionKey'] = getenv('TYPO3_ENCRYPTION_KEY');
}

$GLOBALS['TYPO3_CONF_VARS'] = array_replace_recursive(
    $GLOBALS['TYPO3_CONF_VARS'],
    $config
);
```

LocalConfiguration

??? warning "Hinweis" Dieses Beispiel stammt aus einer TYPO3 Instanz der Version 9.5. Aufbau und Werte müssen an die tatsächliche Version angepasst werden.

```
<?php
return [
    'BE' => [
        'compressionLevel' => '0',
        'debug' => false,
        'explicitADmode' => 'explicitAllow',
        'fileDenyPattern' =>
'^(?!.*(jpeg|jpg|pdf|png|csv|xls|xlsx|svg|gif|doc|docx|ppt|ppsx|pptx|txt|zip|t
s|htaccess|YOUTUBE|VIMEO|TXT|TS|TYPO-
SCRIPT|HTML|HTM|CSS|TMPL|JS|SQL|XML|CSV|XLF))',
        'installToolPassword' => '%env(TYPO3_INSTALL)%',
        'loginSecurityLevel' => 'rsa',
        'versionNumberInFilename' => '0',
```

```
    ],
    'DB' => [
        'Connections' => [
            'Default' => [
                'charset' => 'utf8',
                'dbname' => '%env(TYPO3_DB_NAME)%',
                'driver' => 'mysqli',
                'host' => '%env(TYPO3_DB_HOST)%',
                'password' => '%env(TYPO3_DB_PASS)%',
                'tableoptions' => [
                    'charset' => 'utf8mb4',
                    'collate' => 'utf8mb4_unicode_ci',
                ],
                'user' => '%env(TYPO3_DB_USER)%',
            ],
        ],
    ],
    'EXTCONF' => [
        'helhum-typo3-console' => [
            'initialUpgradeDone' => '9.5',
        ],
        'lang' => [
            'availableLanguages' => [
                'de',
            ],
        ],
    ],
    'EXTENSIONS' => [
        'backend' => [
            'backendFavicon' => '',
            'backendLogo' => '',
            'loginBackgroundImage' => '',
            'loginFootnote' => '',
            'loginHighlightColor' => '',
            'loginLogo' => '',
        ],
        'extensionmanager' => [
            'automaticInstallation' => '1',
            'offlineMode' => '0',
        ],
        'solr' => [
            'allowSelfSignedCertificates' => '1',
            'useConfigurationFromClosestTemplate' => '0',
            'useConfigurationMonitorTables' => '',
            'useConfigurationTrackRecordsOutsideSiteroot' => '1',
        ],
    ],
    'FE' => [
        'debug' => false,
        'loginSecurityLevel' => 'normal',
        'pageNotFoundOnCHashError' => false,
        'pageNotFound_handling' => '/seite-nicht-gefunden/',
        'pageNotFound_handling_statheader' => 'HTTP/1.0 404 Not Found',
```

```
        'pageOverlayFields' => 'uid,doktype,title,subtitle,nav_title,me-
dia,keywords,description,abstract,author,author_email,url,url-
type,shortcut,shortcut_mode,rsm_page_header,rsm_header,rsm_page_header,rsm_ma-
jor_header,rsm_sub_header,rsm_intro_text,rsm_used_in_teaser,rsm_down-
loads,rsm_project_overview,rsm_event_link,rsm_event_address_ti-
tle,rsm_event_address_city,rsm_event_link,rsm_media_youtube,rsm_me-
dia_vimeo,rsm_project_partner_title',
    ],
    'GFX' => [    ],
    'HTTP' => [],
    'INSTALL' => [
        'wizardDone' => [
        ...
        ],
    ],
    'LOG' => [
        'TYPO3' => [
            'CMS' => [
                'deprecations' => [
                    'writerConfiguration' => [
                        5 => [
                            'TYPO3\CMS\Core\Log\Writer\FileWriter' => [
                                'disabled' => true,
                            ],
                        ],
                    ],
                ],
            ],
        ],
        'writerConfiguration' => [
            4 => [
                'TYPO3\CMS\Core\Log\Writer\FileWriter' => [
                    'disabled' => true,
                ],
            ],
        ],
    ],
    'MAIL' => [
        'transport' => 'mail',
        'transport_sendmail_command' => '/usr/sbin/sendmail -t -i ',
        'transport_smtp_encrypt' => '',
        'transport_smtp_password' => '',
        'transport_smtp_server' => 'localhost:25',
        'transport_smtp_username' => '',
    ],
    'SYS' => [
        'caching' => [
            'cacheConfigurations' => [
                'extbase_object' => [
                    'backend' => 'TYPO3\\CMS\\Core\\Cache\\Backend\\Typo3Data-
baseBackend',
                    'frontend' => 'TYPO3\\CMS\\Core\\Cache\\Frontend\\Varia-
bleFrontend',
```

```
                    'groups' => [
                        'system',
                    ],
                    'options' => [
                        'defaultLifetime' => 0,
                    ],
                ],
            ],
        ],
        'devIPmask' => '',
        'displayErrors' => 0,
        'encryptionKey' => '%env(TYPO3_ENCRYPTION_KEY)%',
        'exceptionalErrors' => 4096,
        'features' => [
            'TypoScript.strictSyntax' => false,
            'unifiedPageTranslationHandling' => true,
        ],
        'sitename' => 'Deutsche Energie-Agentur (dena)',
        'systemLocale' => 'de_DE.UTF-8',
        'systemLogLevel' => 4
    ],
];
```

Configuration for Deployment via Helm

Overview:

```
.
├── README.md
├── app
│   ├── vendor [...]
│   └── web
│       ├── fileadmin
│       ├── index.php
│       ├── typo3
│       │   ├── index.php
│       │   └── install.php
│       └── typo3temp [...]
├── composer.json
```

```
├── composer.lock
├── config
│   ├── sites [...]
│   └── system
│       ├── additional.php
│       └── settings.php
├── .deployment
│   ├── docker
│   │   ├── Dockerfile
│   ├── values.live.yaml
│   └── values.staging.yaml
├── .dockerignore
├── .gitignore
├── .gitlab-ci # overrides for pre-defined jobs
│   ├── build
│   │   ├── container
│   │   └── stable.yaml
│   └── deploy
│       └── helm
├── .gitlab-ci.yml
└── var [...]
```

.gitlab-ci.yml

Variables

```
# global variables
variables:
  PROJECT_NAME:
  RANCHER_PROJECT: '<project name>' # short name for deployment, must be
unique, e.g. 'dena'
  RANCHER_CONTEXT_ID: 'c-jjb46:p-<project id>' # project id, provided by the
hoster Swiss IT security via ticket
```

```
  URL_STAGING: 'https://<project name>.stage.dena.de/' # domain for STAGING
environment
  URL_LIVE: 'https://<project name>-live.dena.de/' # domain for LIVE environ-
ment
  HELM_CHART: 'dena/typo3-12' # helm chart for application
  CHART_VERSION: '0.4.0' # optional: version of helm chart, default is pro-
vided by pre-defined job
```

Includes

```
include:
  - project: 'common/deploy/gitlab-ci-config'
    file:
     - '/beforeScript.yaml'             # global `before_script`
     - '/stages.yaml'                   # pre-defined stages for jobs
     - '/variables.yaml'                # global variables
     - '/build/container/image.yaml'    # `build:web:image:*` jobs
     - '/build/stable.yaml'             # `build:stable` (PHP composer)
     - '/deploy/helm/template.yaml'     # templates for helm deployment
     - '/deploy/helm/staging.yaml'
     - '/deploy/helm/live.yaml'
    ref: '1.8.0'                        # version of deployment jobs
  - '.gitlab-ci/**.yaml'                # optional overrides, includes all
`*.yaml` files in folder `.gitlab-ci`
```

Overrides For Pre-Defined Jobs

```
.gitlab-ci/build/stable.yaml
# include `config` folder in build artifacts
build:stable:
  artifacts:
    paths:
      - app/
      - config/
```

```
.gitlab-ci/build/container/image.yaml
build:image:web:stable:
  # set name of container image
  variables:
    IMAGE_NAME: dena-web
```

```
.gitlab-ci/deploy/helm/live.yaml
deploy:helm:live:
  variables:
    KUBE_NAMESPACE: 'dena-live'  # namespace for LIVE environment
```

```
.gitlab-ci/deploy/helm/staging.yaml
deploy:helm:staging:
  variables:
    KUBE_NAMESPACE: 'dena-staging'
```

Values

Example for LIVE environment `(.deployment/values.live.yaml)`

```
replicaCount: 3
strategy:
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 2
    maxUnavailable: 1

image:
  repository: registry.dena.de:4567/ventures/dena.de/v12/dena-bundle
  name: dena-web
  pullPolicy: Always
  # Overrides the image tag whose default is the chart appVersion.
  tag: "0.1.0"

nameOverride: "dena"             # short application name (matches rancher
project name)
fullnameOverride: "dena-typo3"   # full name of application
environment: "live"              # name of target environment

podLabels:
  name: typo3
  venture: dena

service:
  type: LoadBalancer
  port: 80
  targetPort: dena-typo3-http
  protocol: TCP
  name: http
  annotations:
    metallb.universe.tf/loadBalancerIPs: <Loadbalancer IP> # IP-Address for
service in external Loadbalancer. Provided by hoster.
autoscaling:
  enabled: false

###
# Persistent volumes
persistentVolumes:
  fileadmin:
```

```
    resources:
      requests:
        storage: 3Gi    # value must be coordinated with hoster

# Additional volumes on the output Deployment definition.
volumes:
  - name: live-dena-fileadmin-storage
    persistentVolumeClaim:
      claimName: live-dena-fileadmin-claim
  - name: live-dena-uploads-storage
    persistentVolumeClaim:
      claimName: live-dena-uploads-claim
  - name: live-dena-var-storage
    persistentVolumeClaim:
      claimName: live-dena-var-claim
  - emptyDir:
      medium: Memory
      sizeLimit: 512Mi
    name: live-dena-temp

# Additional volumeMounts on the output Deployment definition.
volumeMounts:
  - name: live-dena-fileadmin-storage
    mountPath: /var/www/html/app/web/fileadmin
  - name: live-dena-uploads-storage
    mountPath: /var/www/html/app/web/uploads
  - name: live-dena-var-storage
    mountPath: /var/www/html/var
  - name: live-dena-temp
    mountPath: /var/www/html/app/web/typo3temp

cronJobSchedule: "*/1 * * * *"

# Sites configuration used in TYPO3 site config
sites:
  SITE_DENA_BASE: 'https://dena-v12-live.dena.de/'
  SITE_DENA_SOLR_CORE_READ_DE: 'core_de'
  SITE_DENA_SOLR_CORE_READ_EN: 'core_en'

config:
  db:
    port: '3306'
    user: '<database user name>'     # provided by hoster
    host: '<database IP>'            # provided by hoster. default for LIVE:
172.20.200.41
    database: '<database name>'      # provided by hoster
  applicationContext: 'Production'
  sys:
    sitename: 'dena LIVE'
    trustedHostsPattern: '(^dena-v12-live\.dena\.de$)|(^www\.dena\.de$)'

solrtypo3:
  image:
```

```
    tag: 12.0.1
  enabled: true
  environment: 'live'
  read:
    host: 'live-dena-solrtypo3'
    path: '/'
  volumes:
    - name: live-dena-solrtypo3-pv
      persistentVolumeClaim:
        claimName: live-dena-solrtypo3-pv-claim
  volumeMounts: []
redis:
  environment: 'live'
  host: 'live-dena-redis'
```

CI/CD

.gitlab-ci.yaml

```
image: webdevops/php:7.4


# global variables
variables:
  GIT_STRATEGY: clone
  COMPOSER_CACHE_DIR: /cache/composer
  COMPOSER_ALLOW_SUPERUSER: 1
  COMPOSER_NO_INTERACTION: 1
  COMPOSER_PARAMS: --no-progress --optimize-autoloader --prefer-dist --no-dev

# templates for environments
.environment_staging: &environment-staging-definition
  name: staging
  url: https://dena-de-9.stage.dena.de

.environment_live: &environment-live-definition
  name: "live-9"
  # working domain for LIVE environment, can be changed to "real" live domain
  url: https://www.dena.de

# template for all (!) jobs in stage deploy
.deploy-template: &deploy-definition
  stage: deploy
  image: cpsit/deploy-typo3-capistrano
  needs:
    - build:stable
  variables:
    GIT_STRATEGY: none
```

```
  script:
    - cd .deployment/
    - cap $CI_ENVIRONMENT_NAME deploy
  except:
    variables:
      # do not deploy after service tasks
      - $SERVICE

before_script:
  &before-script-definition # loading files from protected repositories is
currently broken
  - which ssh-agent || ( apt-get update -y && apt-get install openssh-client -
y )
  - mkdir -p ~/.ssh .build
  - chmod 700 ~/.ssh
  - echo "$KNOWN_HOSTS" > ~/.ssh/known_hosts
  - chmod 644 ~/.ssh/known_hosts
  - git config --global user.email "gitlab-ci@git.dena.de"
  - git config --global user.name "Dena Gitlab CI"
  - eval $(ssh-agent -s)
  - echo "$SSH_PRIVATE_KEY" | tr -d '\r' | ssh-add - > /dev/null

#template for all service jobs
.service-template: &service-definition
  stage: service
  image: cpsit/deploy-typo3-capistrano
  allow_failure: true
  only:
    variables:
      # only run in service pipeline
      - $SERVICE
  artifacts:
    paths:
      - .build/*

stages:
  - build
  - test
  - prepare service
  - service
  - deploy
  - post

build:stable:
  stage: build
  script:
    - cd app
    - composer self-update
    # enable the following line if ELTS auth is necessary
    #- composer config --auth $COMPOSER_ELTS_AUTH
    - composer install $COMPOSER_PARAMS
  artifacts:
    expire_in: 1h
```

```
    paths:
      - app/
      - .deployment/
      - .scripts/
  except:
    variables:
      # do not build before service tasks
      - $SERVICE

# provide a database dump for local development
service:provideDatabase:
  <<: *service-definition
  script:
    # a capistrano deployment configuration with the name <SOURCE_ENVIRON-
MENT>.rb
    # we rely on a cronjob providing the dump file
    - cd .deployment
    - cap $SOURCE_ENVIRONMENT utils:database_download_dump $ADDITIONAL_PARAMS
  only:
    variables:
      # only run in service pipeline when source environment is set and data-
base dump required
      - $SERVICE == 'provideDatabase' && $SOURCE_ENVIRONMENT

service:runCapistranoCommand:
  <<: *service-definition
  script:
    - cd .deployment
    - cap $SOURCE_ENVIRONMENT $COMMAND
  only:
    variables:
      # only run in service pipeline when source environment is set and data-
base dump required
      - $SERVICE == 'runCapistranoCommand' && $SOURCE_ENVIRONMENT && $COMMAND

  # enable logging for selected SOURCE_ENVIRONMENT
service:enable logging:
  <<: *service-definition
  stage: prepare service
  script:
    - cd .deployment
    - cap $SOURCE_ENVIRONMENT utils:enable_logging
  only:
    variables:
      - $SERVICE == 'logging'

# download error log from SOURCE_ENVIRONMENT
service:download error log:
  <<: *service-definition
  dependencies:
    - service:enable logging
  script:
    - cd .deployment
```

```
    - cap $SOURCE_ENVIRONMENT utils:load_error_log
  # start 2 minutes after previous job (cron job checks once per minute if
logging is enabled)
  # feature is available from gitlab 11.7 on
  #when: delayed
  #start_in: 2 minutes
  when: manual
  only:
    variables:
      - $SERVICE == 'logging' && $SOURCE_ENVIRONMENT

# download access log from SOURCE_ENVIRONMENT
service:download access log:
  <<: *service-definition
  dependencies:
    - service:enable logging
  script:
    - cd .deployment
    - cap $SOURCE_ENVIRONMENT utils:load_access_log
  when: manual
  only:
    variables:
      - $SERVICE == 'logging' && $SOURCE_ENVIRONMENT

# every commit in a release or hotfix branch will be deployed to STAGING
deploy:staging:
  <<: *deploy-definition
  environment:
    <<: *environment-staging-definition
  only:
    - /^release\/.*/
    - /^hotfix\/.*/

# commits in master will be deployed to LIVE
deploy:live:
  <<: *deploy-definition
  environment:
    <<: *environment-live-definition
  only:
    - tags
  except:
    - branches
```

ignore_tables.txt

Verwendung

Je Zeile ein Ausdruck oder ein Tabellenname.

```
^cache_
^cf_
^cachingframework_
be_sessions
fe_session_data
fe_sessions
tx_solr_cache
tx_solr_cache_tags
tx_solr_indexqueue_indexing_property
tx_solr_indexqueue_item
tx_solr_last_searches
tx_solr_statistics
tx_realurl_pathcache
tx_realurl_uniqalias
tx_realurl_uniqalias_cache_map
tx_realurl_urlcache
tx_extensionmanager_domain_model_extension
tt_news_cache
sys_log
zzz_deleted
```

deploy.rb

```
set :application, 'dena-de'

append :linked_dirs, "app/web/fileadmin", "app/web/uploads", "app/web/intern",
"app/web/test"

# Default value for :scm is :git
set :scm,:copy
set :copy_source,   '../'
set :copy_exclude, ['env.yml', 'env.*.yml', '.env.php',\
 'Gemfile*', 'Capfile', 'Thumbs.db', 'composer.*', 'README.md', \
 'app/web/uploads', 'app/web/typo3temp', '.git', '.idea', '.gitignore', \
 '.DS_Store', '._.DS_Store', 'fixtures', '.ddev/']
set :use_sudo, false

set :console_bin_path, 'app/vendor/bin/typo3cms'

# Path to file with DB credentials and similar information which can not be
stored in the project
# The file is provided by the operator.
set :env_file_path, ->{ "#{fetch(:project_root)}/env.yml" }

# File path for log files. This file can be downloaded by invoking
set :error_log_file_path, ->{ "#{fetch(:deploy_to)}/error.log" }
set :access_log_file_path, ->{ "#{fetch(:deploy_to)}/access.log" }

# File path for database export (dump without caches and user data). This file
can be generated by invoking the task typo3:database_export and will
# be downloaded by invoking typo3:database_download_dump
set :db_dump_file_path, ->{ "#{fetch(:deploy_to)}/db-clean.sql.gz" }
```

```
# keep number of releases
set :keep_releases, 3
# temp directory on remote server
set :tmp_dir, '/tmp/deployment'

set :ssh_options, {
  user: 'dena.deploy'
}

namespace :check do
  desc "Check that we can access everything"
  task :test_write_permissions do
    on roles(:all) do |host|
      if test("[ -w #{fetch(:deploy_to)} ]")
        info "#{fetch(:deploy_to)} is writable on #{host}"
      else
        error "#{fetch(:deploy_to)} is not writable on #{host}"
      end
    end
  end
  task :test_env_file_exists do
    on roles (:all) do |host|
      if test("[ -f $(ENV_FILE_PATH) ]")
        info "$(ENV_FILE_PATH) does exist on #{host}"
      else
        error "ENV_FILE_PATH does not exist on #{host}"
      end
    end
  end
  task :error_log_file_exists do
    on roles (:all) do |host|
      if test("[ -f #{fetch(:error_log_file_path)} ]")
        info "#{fetch(:error_log_file_path)} does exist on #{host}"
      else
        error "#{fetch(:error_log_file_path)} does not exist on #{host}"
      end
    end
  end
  task :access_log_file_exists do
    on roles (:all) do |host|
      if test("[ -f #{fetch(:access_log_file_path)} ]")
        info "#{fetch(:access_log_file_path)} does exist on #{host}"
      else
        error "#{fetch(:access_log_file_path)} does not exist on #{host}"
      end
    end
  end

  task :db_dump_file_exists do
    on roles (:db) do |host|
      if test("[ -f #{fetch(:db_dump_file_path)} ]")
        info "#{fetch(:db_dump_file_path)} does exist on #{host}"
```

```
      else
        error "#{fetch(:db_dump_file_path)} does not exist on #{host}"
      end
    end
  end
end

namespace :utils do
  desc "Enable logging"
   # If a file 'enable_logging' exist, a cronjob will write logs to
   # the error_log_file_path and access_log_file_path. Those can be fetched
via load_error_log and load_access_log task.
   # after 1 hour the logging will be disabled automatically
  task :enable_logging do
    on roles(:all) do |host|
      execute "mkdir -p  #{fetch(:deploy_to)}/config; touch #{fetch(:de-
ploy_to)}/config/enable_logging"
    end
  end

  desc "Download error log file"
  task :load_error_log do
    on roles(:all) do |host|
      if test("[ -f #{fetch(:error_log_file_path)} ]")
        info "Error log found on #{host}"
        download! "#{fetch(:error_log_file_path)}", "../.build/er-
ror.#{host}.log"
      else
        error "Error log #{fetch(:error_log_file_path)} not found on #{host}"
      end
    end
  end

  desc "Download access log file"
  task :load_access_log do
    on roles(:all) do |host|
      if test("[ -f #{fetch(:access_log_file_path)} ]")
        info "Access log found on #{host}"
        download! "#{fetch(:access_log_file_path)}", "../.build/ac-
cess.#{host}.log"
      else
        error "Access log #{fetch(:access_log_file_path)} not found on
#{host}"
      end
    end
  end

  desc "Download database dump"
  task :database_download_dump do
    on roles(:db) do |host|
      if test("[ -f #{fetch(:db_dump_file_path)} ]")
        info "Database dump found on #{host}."
```

```
        download! "#{fetch(:db_dump_file_path)}",
"../.build/mysql.#{host}.sql.gz"
      else
        error "Database dump file #{fetch(:db_dump_file_path)} not found on
#{host}"
      end
    end
  end
end

namespace :typo3 do
  desc "Clear TYPO3 cache"
  task :cache_flush do
    on roles(:web) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)}; #{re-
lease_path}/#{fetch(:console_bin_path)} cache:flush"
      end
    end
  end

  desc "Update language"
  task :language_update do
    on roles(:web) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)}; #{re-
lease_path}/#{fetch(:console_bin_path)} language:update"
      end
    end
  end

  desc "Download database dump"
  task :database_download_dump do
    on roles(:db) do |host|
        if test("[ -f #{fetch(:db_dump_file_path)} ]")
          info "Database dump found on #{host}."
          download! "#{fetch(:db_dump_file_path)}",
"../.build/mysql.#{host}.sql.gz"
        else
          error "Database dump file #{fetch(:db_dump_file_path)} not found on
#{host}"
        end
    end
  end

  desc "Update database add schema"
  task :database_update_add do
    on roles(:db) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)}; #{re-
lease_path}/#{fetch(:console_bin_path)} database:updateschema '*.add'"
      end
    end
```

```
  end

  desc "Update database schema"
  task :database_update do
    on roles(:db) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)}; #{re-
lease_path}/#{fetch(:console_bin_path)} database:updateschema
'*.add,*.change'"
      end
    end
  end

  desc "Warm up cache"
  task :cache_warmup do
    on roles(:web) do
      within release_path do
        info "Task not implemented yet"
      end
    end
  end

  desc "Perform all upgrade wizards"
  task :upgrade_all do
    on roles(:web) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)}; #{re-
lease_path}/#{fetch(:console_bin_path)} upgrade:all"
      end
    end
  end

  desc "Fix folder structure"
  task :fix_folder_structure do
    on roles(:all) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)}; #{re-
lease_path}/#{fetch(:console_bin_path)} install:fixfolderstructure"
      end
    end
  end

  # overwrite dkdeploy task (which uses deploy_to path)
  desc "Enable install tool"
  task :enable_install_tool do
    on roles(:all) do
      within release_path do
        execute "touch #{fetch(:web_path)}/typo3conf/ENABLE_INSTALL_TOOL"
      end
    end
  end

  desc "Solr Initialize"
```

```
  task :solr_initialize do
    on roles(:web) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)} TYPO3_CON-
TEXT=#{fetch(:typo3_context)}; #{release_path}/#{fetch(:console_bin_path)}
solr:updateconnections"
      end
    end
  end

  # Note: This job should not be executed in LIVE environment
  # Make sure not to trigger it automatically
  desc "Solr initialize index queue"
  task :solr_initialize_queue do
    on roles(:serviceStaging) do
      within release_path do
        execute "export ENV_FILE_PATH=#{fetch(:env_file_path)} TYPO3_CON-
TEXT=#{fetch(:typo3_context)}; #{release_path}/.scripts/runSolrIndexQueueJobs
#{fetch(:project_root)}"
      end
    end
  end

  # Note: This job should not be executed in LIVE environment
  # Make sure not to trigger it automatically
  desc "Solr empty cores"
  task :solr_empty_cores do
    on roles(:serviceStaging) do
      within release_path do
        execute "#{release_path}/.scripts/emptySolrCores.sh
#{fetch(:solr_core_file_path)}"
      end
    end
  end

  desc "Advertise deployment"
  task :advertise_deployment do
    on roles(:app) do
      within release_path do
        execute "touch #{fetch(:deploy_to)}/current/.deployed"
      end
    end
  end

end

namespace :deploy do
  before :starting, "check:test_write_permissions"
  before :starting, "check:test_env_file_exists"
  before :cleanup, "typo3:database_update_add"
  before :cleanup, "typo3:upgrade_all"
  before :cleanup, "typo3:database_update"
  before :cleanup, "typo3:language_update"
```

```
  before :cleanup, "typo3:fix_folder_structure"
  before :cleanup, "typo3:enable_install_tool"
  before :cleanup, "typo3:cache_flush"
  before :cleanup, "typo3:advertise_deployment"
  before :cleanup, "typo3:solr_empty_cores"
  before :cleanup, "typo3:solr_initialize"
  before :cleanup, "typo3:solr_initialize_queue"
  before :cleanup, "typo3:cache_warmup"
end
```