Der eigene Blog mit Hugo
- Ein Praxisbeispiel -

Chemnitzer Linux-Tage 2026

Tobias Heine | @toheine@social.tchncs.de | 29.03.2026

Ablauf

1. Motivation
2. Vorbereitung
3. Erste Blog-Posts
4. Bereitstellung/Deployment
5. Anpassungsmöglichkeiten
6. Fazit und weitere Infos

Motivation

Vorbemerkungen

- Ein Praxisbeispiel / ein Erfahrungsbericht wird vorgestellt
- Hugo Grundlagen:
  CLT Vortrag von 2024 (Gabriel Wustmann): Static Website am Beispiel Hugo
- **Ziel hier im Vortrag:** Einstiegshürden nehmen
- Keine Expertise in Webentwicklung
- Kein Vergleich zu anderen Static-Site-Generatoren
- Umsetzung ist natürlich verbesserungswürdig → feedback welcome

Warum überhaupt ein Blog?

Zitat aus meiner Bubble: Es gibt doch Social Media!!!

- Nee … ist nicht meine Welt!
- Social-Media-Instanzen …
  - sind IMHO **nicht** gut **durchsuchbar**
  - bieten zu **wenig Platz** für Text, Bilder, Codebeispiele
  - sind eher **für den Austausch** geeignet

Ich finde: **Bloggen lohnt sich!**

Einsatzzweck

- Eigene, sinnvolle **Notizen und Hinweise**
- **Empfehlungen** zu Linux, Open-Source-Anwendungen und Netzwerke (Toolbox)
- **Links** die ich immer wieder brauche und empfehle
- **Zielgruppe:**
  - Kolleginnen und Kollegen
  - Teilnehmende von Lehrgängen für Lehrkräfte in BW
  - Alle die es interessiert

Warum kein CMS?

- Bisherige eigene Erfahrungen mit Joomla und Wordpress
- CMS-Systeme bieten Möglichkeiten die ich nicht benötige
- Regelmäßige Updates → PFLICHT!
- Regelmäßiger Check ob Instanz arbeitet wie sie soll → PFLICHT!
- Hab ich Zeit für sowas → NOPE!
- Hab ich Bock auf sowas → NOPE!
- Merke: **CMS + Keine Zeit für Wartung/Updates = Shit happens**

Anforderungen

- System soll **zuverlässig** arbeiten und ich will damit nichts zu tun haben
- Ein **Blog** + ein paar wenige **statische Seiten**
- Eine **Suche** wird nicht benötigt (max. ein Beitrag pro Monat)
- Ausschließliche Konzentration auf Text mit wenigen Bildern
- Inhalte verfassen mit **Markdown**
- Versionierung mit **git** und **automatische Deployments**
- RSS- bzw. **Atom-Feed**

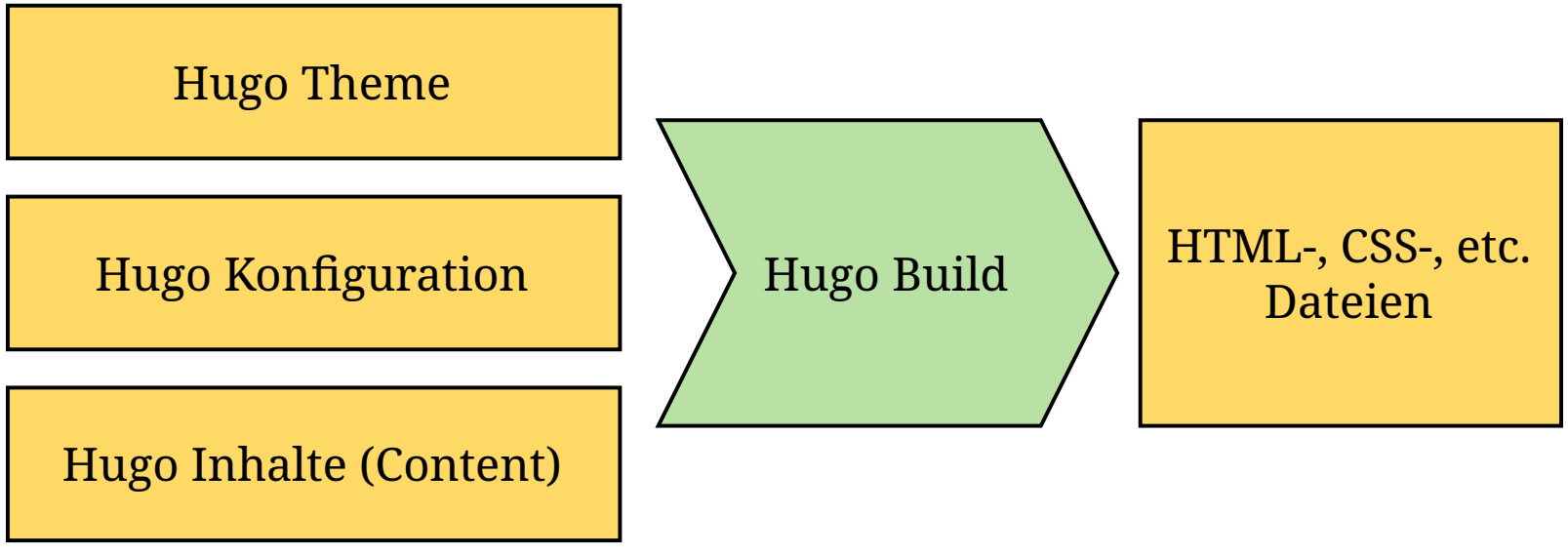
System das meine Anforderungen erfüllt → Hugo

Vorbereitung

Hugo installieren

- Infos: https://gohugo.io/installation/linux/
- Installation aus den Paketquellen
- Verschiedene Versionen verfügbar. Je nach Linux-Distribution wird eine andere Version paketiert:
  - Standard
  - Extended-Edition (z. B. Debian)
  - Extended-/Deploy-Edtion (z. B. Arch Linux)

Was macht Hugo eigentlich?



Arbeitsweise von Hugo

Ein geeignetes Hugo-Theme finden

- Reichlich Auswahl z. B. unter https://themes.gohugo.io/
- Gut zu filtern und meist mit einer Demo
- **Tipps** zur Auswahl:
  - Vorher überlegen, welche Features benötigt werden
    (z. B. Mehrsprachigkeit, Suche, Archiv-Seite, aktive Entwicklung, etc.)
  - Entscheidung für ein Theme beeinflusst Funktionsumfang
  - Entsprechende Git-Repos und Doku der Themes vorab sichten
  - Less is more (!!!)
  - Man kann hier echt viel Zeit verheizen (!!!)

Mein Theme der Wahl

- Hugo Geekblog: https://themes.gohugo.io/themes/hugo-geekblog/
- Demo = Doku: https://hugo-geekblog.geekdocs.de/
- Github: https://github.com/thegeeklab/hugo-geekblog
- **Danke Robert Kaussow**

Erste Hugo-Site

- Im gewünschten Arbeitsverzeichnis eine neue Seite erzeugen, Theme runter laden, Theme in Config hinterlegen:

```
hugo new project demo01
cd demo01
mkdir themes/hugo-geekblog
wget -c https://github.com/thegeeklab/hugo-geekblog/releases/download/v3.2.0/hugo-geekblog.tar.gz -O - | tar -xz -C themes/hugo-geekblog/
echo "theme = 'hugo-geekblog'" >> hugo.toml
```

- Hugo-Dateien bauen und unter localhost (Port 1313) inklusive Entwürfe (`-D` = draft) und Beiträge mit einem zukünftigen Veröffentlichungsdatum (`-F` = future) ausliefern:

```
hugo server -D -F
```

Mit Hugo arbeiten

Inhalt des Hugo-Projekts nach Erzeugung der Hugo-Instanz (siehe auch Hugo Docs):

```
demo01
├── archetypes # Vorlagen für neue Seiten
├── assets     # CSS, Javascirpt, Logos, etc.
├── content    # Blogposts und statsiche Seiten des Webauftritts
├── data       # Datenquellen die bei Hugo-Build werarbeitet werden
├── hugo.toml  # KONFIGURIATIONSDATEI (machmal auch config.toml, hugo.json, ...)
├── i18n       # Übersetzungsdateien
├── layouts    # definiert wie Seite aufgebaut / Inhalte gerendert werden
├── public     # Ort für gebaute Seiten nach Build
├── static     # Statischer Content der bei Build nach public kopiert wird
└── themes     # Ein oder mehrere Themes
```

Anpassungen vornehmen

- Konfiguration über `hugo.toml` Hinweise dazu aus:
  - Doku des verwendeten Themes (z. B. geekblog) ← erster Anlaufpunkt!
  - Hugo Docs (Abschnitt configuration)
- Weitere Dateien nach Vorgabe von Theme oder Recherche im Internet nach Bedarf abgelegt … z. B.:
  - `layouts` : Verhalten von Links angepasst
  - `static` : custom.css, favicon und includes
  - `data` : Hinweise zum Autor und Menu-Leiste

Inhalte bereitstellen

- Alles unterhalb von `content` beinhaltet dann die entsprechenden Inhalte
- Dieser Ordner wird weiter strukturiert. In meinem Fall:

```
content
├── about
├── bookmarks
├── linux
├── posts
│   ├── 2023
│   ├── 2024
│   ├── 2025
│   └── 2026
└── toolbox
```

Erste Blogposts

Vorlagen für neue Blogbeiträge

Unter `archetypes/default.md` gibt es bereits eine Stuktur:

```
+++
date = '{{ .Date }}'
draft = true
title = '{{ replace .File.ContentBaseName "-" " " | title }}'
+++
```

Diese kann bei Bedarf mit weiteren Default-Angaben ergänzt werden (z. B. Autor, Tags, etc.)

Inhalte lokale erstellen

Neuer Blogeintrag wird erzeugt `hugo new content` und einer Pfadangabe:

```
hugo new content posts/2026/clt26/index.md
```

Voreinstellungen aus `archetypes/default.md` bereits hinterlegt:

```
$ cat content/posts/2026/clt26/index.md
+++
date = '2026-03-27T06:10:45+01:00'
draft = true
title = 'Clt26'
+++
```

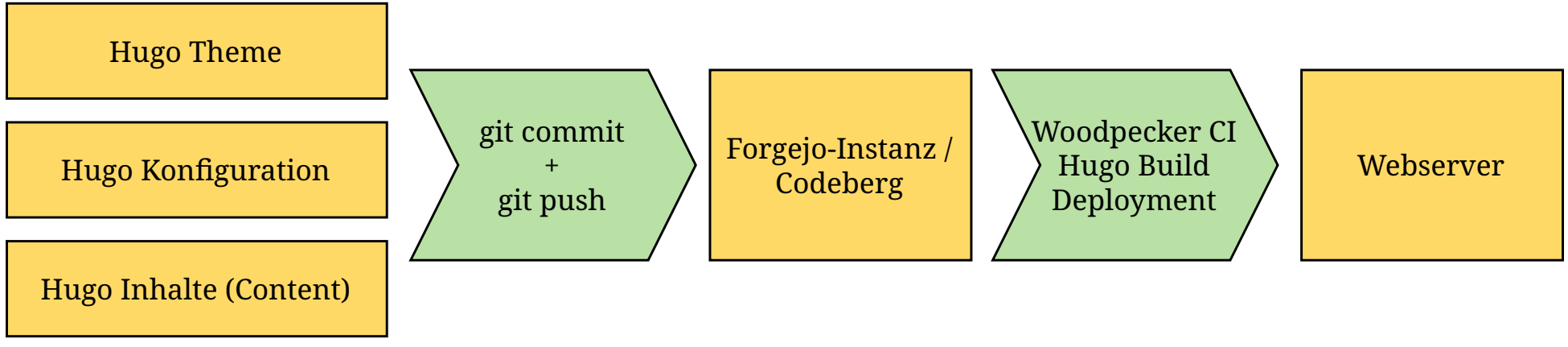
Alles weitere ist dann in Markdown verfasst.

Hinweise

- Seiten innerhalb eines Verzeichnisses aufzubauen hat sich für mich bewährt:
  - bevorzugt so: `hugo new content posts/2026/clt26/index.md`
  - statt so: `hugo new content posts/2026/clt26.md`
- Gezieltes Verwenden des Trenners `<!--more-->` für Strukturierung der Übersichtsseite (siehe gohugo.io)
- Verwendetes Theme bringt mit `{{< toc >}}` die Möglichkeit mit, schnell Inhaltsverzeichnisse anzulegen. (siehe Geekblog Demo-Seite).

Deployment

Übersicht



Arbeitsweise von Hugo im Zusammenspiel mit git und Woodpecker CI

Prozess der Veröffentlichung

- Ein Blog-Post wird erstellt: `hugo new content posts/<JJJJ>/<Thema>/index.md`
- lokalen Webserver starten um eine Vorschau zu haben `hugo server -D`
- Blogpost verfassen
- via git commiten und auf git-Instanz pushen
- Push führt zur Ausführung eines Builds und Deployments via Woodpecker CI

Bereitstellung via Codeberg Pages

Verfahren bis 01.2025

- hier beschrieben:
  https://toheine.net/posts/2023/website-deployment/
- funktioniert zu Demozwecken heute noch
- Das Ergebnis dieses Builds und Deployments erreichbar via:
  https://toheine.codeberg.page/toheine/

Einrichtung und kurze Demo

- Hugo-Quellen liegen in einem git Repo auf Codeberg
- Codeberg CI ist beantragt und verfügbar
  https://codeberg.org/Codeberg-e.V./requests
- Ein Codeberg Pages Repo ist zusätzlich eingerichtet
- Codeberg CI ist mit dem Quellen-Repo verbunden und hat notwendige Secrets
- Eine `.woodpecker.yml` liegt im Quellen-Repo bereit. Vorlage:
  https://codeberg.org/Codeberg-CI/examples/src/branch/main/Hugo/.woodpecker.yaml
- Ein `git push` führt zur Ausführung der Kommandos aus der `.woodpecker.yml`

Bereitstellung auf eigenem Webserver

Verfahren ab 01.2025

- hier beschrieben:
  https://toheine.net/posts/2025/moved_blog/
- eigene Forgejo-Instanz
  https://forgejo.org/docs/latest/admin/installation/
- eigene Woodpecker-Instanz
  https://woodpecker-ci.org/docs/administration/installation/docker-compose
- eigener kleiner Webserver mit Zugriff via scp und rsync
- Zugriff via scp und rsync
- Einrichtung beinahe identisch mit veränderter `.woodpecker.yml` siehe:
  https://codeberg.org/toheine/toheine_sources/src/branch/main/.woodpecker.yml

Anpassungsmöglichkeiten

Schnelleres suchen und finden

- Nach knapp zwei Jahren Blog-Betrieb stand der Wunsch nach einem besseren Zugang zu alten Artikeln im Raum
- Denkbare Umsetzung:
  - Auflistung aller verwendeten Tags → hilfreich
  - Archivseite → hilfreich
  - Suche → eigentlich nicht notwendig
- Möglichkeiten:
  - Mächtigeres Theme → Keine Option weil viel Arbeit
  - Eigene Anpassung → Die Lösung

Ein Template erzeugen und verwenden

- Eine Datei `content/archive.md` erstellt:

```
---
title: "Posts Archive"
layout: archive
---
```

- Diese bezieht sich auf ein Hugo-Template unter `layouts/archive.html`

Alle Tags aufführen

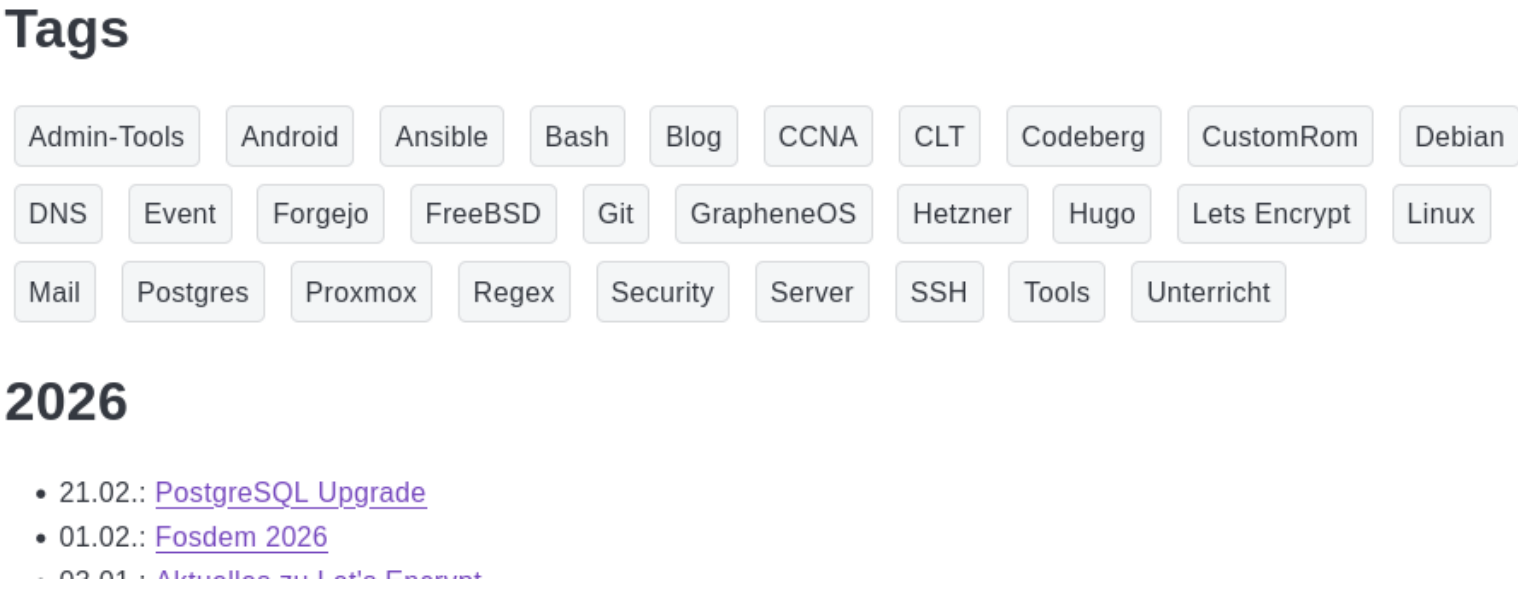
```
{{ range $name, $taxonomy := $tags }}
  {{ with $.Site.GetPage (printf "/tags/%s" $name) }}
    <span class="gblog-post__tag gblog-button gblog-button--regular" style="margin:
5px">
      <a
        class="gblog-button__link"
        href="{{ .RelPermalink }}" title="{{ .Title }}">
        {{ .Title }}
      </a>
    </span>
  {{ end }}
{{ end }}
```

Alle Posts nach Jahr gruppiert aufführen

```
{{ $type := "posts" }}
{{ $.Store.Set "count" 1 }}
{{ range (.Site.RegularPages.GroupByDate "2006") }}
  {{ if (gt (where .Pages "Type" $type) 0) }}
    {{ range (where .Pages "Type" $type) }}
      {{ if (eq ($.Store.Get "count") 1) }}
        <h1>{{ .Date.Format "2006" }}</h1>
        {{ $.Store.Set "count" 0 }}
      {{ end }}
    {{ end }}
    {{ $.Store.Set "count" 1 }}
    <ul>
    {{ range (where .Pages "Type" $type) }}
      <li>
          {{ .Date.Format "02.01." }}:
          <a href="{{ .RelPermalink }}">{{ .Title }} </a>
      </li>
      [...]
```

Ergebnis

- Archiv-Seite auf toheine.net https://toheine.net/archive/
- Blogpost zur Umsetzung mit vollständigem Code und weiterführenden Links: https://toheine.net/posts/2025/implement-archive/



Archivseite mit verwendeten Tags

Fazit und weitere Infos

Erkenntnis

- Bereitstellung / Webauftritt …
  - ist keine Raketentechnologie
  - ist bei Inbetriebnahme mit Arbeit verbunden (Zusammenspiel der Komponenten)
  - sorgt im Anschluss für einfache Veröffentlichung von Inhalten und
  - lässt mich nachts ruhig schlafen ;-)
- Geplante nächste Schritte:
  - Deployment via forgejo actions
  - Neue Artikel automatisch auf Mastodon tooten

Quellen und weitere Infos I

Hugo:

- Offizieller Webauftritt von Hugo: https://gohugo.io/
- Auswahl von Themes: https://themes.gohugo.io/
- Hugo-Forum: https://discourse.gohugo.io/
- Video Static Webseite am Beispiel Hugo (CLT 2024)
- Artikel-Serie Static-Site-Generators auf GNU/Linux.ch (Juni 2025)
- Eigene Erkenntnisse: https://toheine.net/tags/Hugo

Quellen und weitere Infos II

Deployment:

- Eingesetzte Git-Plattform: https://forgejo.org/
- Webauftritt von Codeberg e. V.: https://codeberg.org/
- Hinweise zur Nutzung von Codeberg Pages: https://codeberg.page/
- CI bei Codeberg beantragen: https://codeberg.org/Codeberg-e.V./requests
- Eingesetzte CI/CD-Plattform: https://woodpecker-ci.org/
- Umsetzungsbeispiel mit Codeberg Pages und Woodpecker CI auf toheine.net