Zpravodaj Československého sdružení uživatelů TeXu

Marei Peischl; Marcel Krüger; Oliver Kopp

Příprava (La)TeXových dokumentů v cloudu

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 34 (2024), No. 1-4, 59–72

Persistent URL: http://dml.cz/dmlcz/152651

Příprava (LA)TEXových dokumentů v cloudu

Marei Peischl, Marcel Krüger a Oliver Kopp

Používání webových služeb pro kolaborativní přípravu dokumentů v (LA)TEXu je dnes běžné, ale tyto nástroje se zaměřují na psaní dokumentů, nikoliv na tvorbu šablon nebo balíčků. Používání serverů pro průběžnou integraci a doručení je běžné ve vývoji softwaru, ale může být snadno přizpůsobeno pro TEX a přátele.

Tento článek ukazuje, jak začít s používáním automatizace na webových službách jako GitHub, Forgejo nebo GitLab, a to jak pro autory dokumentů, tak pro vývojáře (LA)TEXových balíčků.

Klíčová slova: průběžná integrace a doručení, GitHub, Forgejo, GitLab, docker

1 Úvod

Všechno se dnes přesouvá do cloudu nebo je tam již dostupné. (LA)TEX je v cloudu již přibližně 10 let a dnes je zcela běžné používat pro přípravu dokumentů webový editor (jako Overleaf [2; 3], pozn. red.), zatímco kompilace na vlastním počítači se stala „záležitostí geeků“. Existuje však také třetí varianta pro kompilaci (LA)TEXových dokumentů, kterou můžeme využít také ke zlepšení procesu vývoje balíčků a obecně ke zvýšení stability (LA)TEXu: osvojení metod DevOps, jako je průběžná integrace a doručení (tzv. CI/CD) s použitím automatizovaných postupů.

Pracovně můžeme definovat postup CI/CD (anglicky „workflow“, pozn. překl.) jako soubor kroků, které vedou ke kompilaci (LA)TEXového dokumentu do formátu PDF na nějaké online službě, která má přístup ke zdrojovým souborům.

2 Proč průběžná integrace?

Zaběhnuté pracovní postupy nás často odrazují od změn v našem způsobu práce. Aby tedy mělo smysl číst tento článek, natož pak integrovat tyto mechanismy do projektů, musí pro to existovat dobré důvody.

Raní uživatelé postupů CI/CD v ekosystému TEXu se snažili držet krok se současným stavem ve světě open source softwaru a otevřít dveře přispěvatelům do vývojového procesu. Například projekt LATEX interaguje s uživateli prostřednictvím projektů na webové službě GitHub [4]. Přijímá zde také příspěvky v podobě softwarových záplat, ze kterých může těžit celá (LA)TEXová komunita.

Výhody těchto metod však sahají mnohem dál. Zaměříme se na několik vybraných aspektů, protože vyčerpávající popis by vydal na samostatný článek.

Z anglického originálu [1] přeložil a rozšířil se svolením autorů a vydavatele Vít S. Novotný.

DOI: 10.5300/2024-1-4/59

2.1 U mě to funguje?!

Někdy se mi na mém počítači podaří úspěšně zkompilovat dokument, ale mému vedoucímu na jeho počítači už ne. Existuje mnoho důvodů, proč může kompilace TEXového dokumentu na jednom systému uspět a na jiném selhat. Spuštění externího průběžného integračního procesu nejenže ilustruje veškeré potřebné kroky pro přechod od zdrojových souborů k výstupu ve formátu PDF, ale také pomůže zjistit, zda jsou problémy specifické pro konkrétní počítač, nebo zda jsou obecné.

2.2 Zpětná kompatibilita a regresní testování

S použitím postupů CI/CD můžeme spustit kompilaci na různých distribucích TEXu nebo jejich verzích. Díky tomu můžeme otestovat, zda nějaký balíček způsobuje problémy ještě před tím, než např. nainstalujeme nejnovější verzi MiKTEXu na svém počítači. To nám může ušetřit mnoho času, protože návrat k předchozí verzi je často komplikovaný.[1]

Tuto metodu můžeme také využít ke kontrole zpětné kompatibility, například pokud jeden z přispěvatelů používá stabilní verzi Debianu, která obsahuje pouze zastaralou verzi TEXové distribuce. Jak již bylo řečeno, tým projektu LATEX tyto techniky již mnoho let používá a dokonce poskytuje funkce pro regresní testování v rámci jejich systému l3build [5; 6] pro sestavování (LA)TEXových balíčků.

Vývojáři balíčků mohou postupy CI/CD používat jako obecné rozhraní pro regresní testování. To pomáhá předcházet některým chybám, které by jinak byly zveřejněny a nalezeny až samotnými uživateli. Dále je možné CI/CD postupy použít pro detekci nekonzistencí v kódu. Jeden z autorů např. zjistil, že značné množství balíčků a souborů v distribuci TEX Live neuvádí v kódu číslo verze.

3 Struktura článku

Cílem tohoto článku je seznámit čtenáře se základy nastavení postupů CI/CD na webových službách GitHub, Forgejo a GitLab. Článek je zaměřen především na dvě skupiny uživatelů:

1. autory, kteří se zaměřují na sazbu obsahu, a jejich spolupracovníky a
2. vývojáře balíčků a šablon, jejichž práce tvoří podloží pro práci autorů.

Vývojáři samozřejmě mohou využít postupy určené pro autory, například pro sazbu dokumentace balíčku.

Vzhledem k tomu, že všechny služby pokryté tímto článkem jsou založeny na systému správy verzí git, očekáváme, že projekt bude gitový repozitář. Pro čtenáře, kteří dosud git nepoužívají, odkazujeme doplňující informace z repozitáře tohoto článku [7]. S jejich využitím lze git používat, aniž by si to uživatel vůbec uvědomoval, protože je integrován do automatického ukládání v textovém editoru.

[1] To je další problém… Ale to už je jiný příběh, který by také vystačil na samostatný článek.

Tabulka 1: Ukázkové repozitáře zveřejněné spolu s tímto článkem. Názvy repozitářů mají strukturu ⟨*typ postupu*⟩`_`⟨*úkol*⟩ s přidáním koncovky `_minimal` v případě, že příklad nepoužívá předpřipravený dockerový obraz, ale zahrnuje metody instalace balíčků na základě souboru závislostí, jak popisujeme v sekci 6.

Všechny zde uvedené varianty jsou připraveny pro alespoň tři zde uvedené webové služby: GitHub, GitLab a Forgejo.

Vzhledem k tomu, že adresy repozitářů jsou poměrně dlouhé, zveřejnili jsme je v repozitáři tohoto článku [7] na adrese `https://tug.org/l/peischl-cicd2024`.

Název	Služby	Dokument	l3build	Testy	Docker
`latex`		✔			✔
`latex_minimal`		✔			
`latex_testing`		✔		✔	✔
`latex_testing_minimal`		✔		✔	
`l3build`			✔	✔	✔
`l3build_minimal`			✔	✔	

Protože projekt LaTeX používá službu GitHub, začneme podrobným vysvětlením postupů GitHub Actions a poté vytvoříme odpovídající postupy na dalších službách. Všechny postupy CI/CD jsou k dispozici v ukázkových repozitářích uvedených v Tabulce 1. Všechny ukázky kódu v tomto článku jsou označeny ikonami, aby nedošlo k záměně, protože služby budeme v článku střídat pro ilustraci rozdílů: Ikona označuje, že ukázka pochází z postupu GitHub Actions, zatímco ikona označuje postup GitLab CI.

4 Kompilace dokumentu pomocí průběžné integrace

4.1 První kroky s GitHub Actions

Nejprve založíme repozitář na službě GitHub. Repozitář nemusí být veřejný. Ke kompilaci (La)TeXového dokumentu v „čistém“ prostředí potřebujeme následující:

1. Připravit prostředí a nainstalovat TeXovou distribuci, např. TeX Live.
2. Spustit na dokumentu TeXový stroj s příslušným formátem, např. LaTeXem.

Kromě elementárních příkazů poskytuje služba GitHub předpřipravené „akce“, které provádí více elementárních příkazů v jednom kroku. Pro nás je zajímavá např. akce `latex-action` [8], která zkompiluje LaTeXový dokument pomocí příkazu `latexmk` uvnitř kontejneru s instalací TeX Live. Tato a další akce však obvykle omezují možnosti konfigurace, aby bylo rozhraní co nejjednodušší.

V článku se budeme zabývat dvěma přístupy: Použitím dockerového obrazu s plnou instalací TeX Live v této sekci a s minimální instalací TeX Live v Sekci 6.

Konfigurace postupu CI/CD se provádí vytvořením souboru YAML v adresáři `.github/workflows/`. Název souboru můžeme zvolit libovolný, např. `ahoj-světe.yml`. Následující ukázka kódu zobrazuje minimální konfiguraci:

```yaml
name: Ukázkový postup GitHub Actions
on: [push]
jobs:
  ukázková-úloha:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Ahoj světe! 🦁🦆"
```

Níže uvádíme význam jednotlivých sekcí souboru YAML:

name: Název postupu, který bude zobrazený ve webovém rozhraní GitHubu. Tento název je důležitý, pokud projekt obsahuje více postupů.

on: Podmínky, za kterých bude postup spuštěn. V naší ukázce se úlohy spustí při každém `push`i, tj. kdykoliv aktualizujeme obsah repozitáře. Kromě reakce na změny obsahu je možné spouštět úlohy pravidelně v určitý čas.
Pro všechny možnosti vizte dokumentaci [9]. Výchozí nastavení se liší pro každou platformu (jako Windows, Linux a Mac OS, pozn. překl.) a může být specifické pro konkrétní výpočetní uzel, vizte níže.

jobs: Seznam úloh, které se mají postupně spustit. Je například běžné mít jednu úlohu pro kompilaci dokumentu a druhou pro zveřejnění výsledného souboru PDF. V naší ukázce je jen jedna úloha nazvaná `ukázková-úloha`.

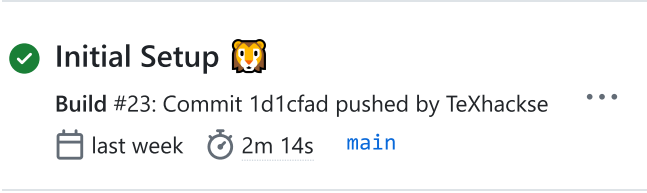
runs-on: Nastavení výpočetního uzlu (anglicky „runner“, pozn. překl.). Výpočetní uzly jsou servery, které vykonávají úlohy definované v sekci `jobs`. Nemusí jít o stejné servery, na kterém jsou hostovány gitové repozitáře. V naší ukázce označuje `ubuntu-latest` vestavěný výpočetní uzel poskytovaný službou GitHub, který používá linuxovou distribuci Ubuntu. Výpočetní uzel obsahuje program NodeJS a některé další nástroje pro usnadnění práce s předdefinovanými akcemi. Úplný seznam vestavěných uzlů najdeme v dokumentaci [10].

steps: Kroky, tj. elementární příkazy a akce, které má postup skutečně provést. V elementárních příkazech je možné přímo zadávat kód pro příkazovou řádku operačního systému a používat Unicode. Náš elementární příkaz vypíše text na standardní výstup a měl by proběhnout bez problémů.

Na GitHubu jsou postupy pro nové repozitáře automaticky povoleny. Při použití vestavěného výpočetního uzlu tak, jak to děláme zde, nám pro spuštění postupu stačí do repozitáře přidat soubor YAML. Následně se postup spustí při všech následujících změnách obsahu repozitáře.

Aktuální stav postupu, tj. který krok právě běží nebo zda již byl dokončen, lze vždy zkontrolovat na ⟨*adrese repozitáře*⟩`/actions`. Například stav postupu

pro první z našich ukázkových repozitářů najdete na adrese `https://github.com/islandoftex/tug2024-workflow-github/actions`. Vypadá nějak takto:



Postup proběhl úspěšně a vypsal text na příkazovou řádku operačního systému. Nyní přejdeme k dalšímu kroku: kompilaci LaTeXového dokumentu.

4.1.1 GitHub Actions s LaTeXem

GitHub Actions jsou postaveny na kontejnerizovaném softwaru, který je spouštěný pomocí dockeru. Naštěstí někteří z nás žijí na Ostrově TeXu (anglicky „Island of TeX“, zkráceně IoT, pozn. překl.) a udržují dockerové obrazy, které zde můžeme využít.[2] Příprava těchto obrazů byla popsána v předchozím článku [12].

V této sekci upravíme ukázkový postup z předchozí strany tak, aby zkompiloval LaTeXový dokument pomocí obrazu od IoT. První část postupu zůstane prozatím stejná, změny nastávají až po direktivě `runs-on`:

```
5     runs-on: ubuntu-latest
6     container:
7       image: texlive/texlive:latest
8     steps:
9       - name: Stáhni repozitář
10        uses: actions/checkout@v4
11      - name: Zkompiluj LaTeXové dokumenty
12        run: "latexmk --lualatex"
```

container: Dockerový obraz, uvnitř kterého se spouští veškeré kroky. V našem příkladu jsme zvolili obraz `texlive/texlive:latest` s plnou instalací TeX Live a některými doplňujícími nástroji [13].

steps: První krok vypadá odlišně od našeho předchozího příkladu. Tento má název „Stáhni repozitář“ a místo elementárního příkazu používá akci `actions/checkout@v4` z tagu `v4` gitového repozitáře na adrese `https://github.com/actions/checkout/`. Tato akce se stará o autentizaci a některé interní záležitosti, takže se nemusíme těmito detaily zabývat. Veškeré další kroky proběhnou už v kořenovém adresáři staženého repozitáře.

[2] Velké díky ostatním obyvatelům Ostrova!

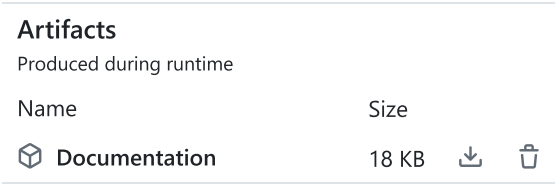
Druhý krok má název „Zkompiluj LaTeXové dokumenty“ a spouští elementární příkaz stejně jako naše předchozí ukázka. Tentokrát spustíme příkaz `latexmk` [15] s volbou `--lualatex`, která, jak možná tušíte, vynutí použití stroje LuaTeX s formátem LaTeX. Ve výchozím nastavení `latexmk` zpracuje všechny soubory s koncovkou `.tex` v kořenovém adresáři repozitáře. Díky tomu v direktivě `run:` nemusíme uvádět jména řídicích souborů našich LaTeXových dokumentů. Postačí, když veškeré soubory s koncovkou `.tex`, které nemají být zkompilovány, umístíme do podadresářů.

4.1.2 Kde jsou soubory PDF?

Pokud byl postup úspěšný, uvidíme sice zelenou ikonu, ale soubory PDF bychom hledali marně. Je to proto, že služba GitHub neví, které výstupní soubory chce uživatel skutečně vidět nebo stáhnout. Tyto kýžené soubory se nazývají „artefakty“. Nyní přidáme do předchozího postupu další krok s akcí `actions/upload-artifact@v4`, která soubory PDF označí jako artefakty:

```
13      - name: Ulož dokumentaci
14        uses: actions/upload-artifact@v4
15        with:
16          name: Documentation
17          path: "./*.pdf"
```

Po dalším úspěšném běhu postupu bude možné přistoupit k souborům PDF, které budou zabaleny do archivu ZIP a dostupné na stránce se stavem běhu. Kliknutím na běh na stránce s přehledem akcí se dostaneme k artefaktu:



Služba GitHub bohužel momentálně nenabízí možnost stáhnout jednotlivé nezabalené soubory a také neumožňuje vytvořit statický odkaz na nejnovější artefakty. (Jiné služby jako GitLab toto umožňují.) K vyřešení tohoto problému a pro snazší přístup k souborům PDF na GitHubu je třeba použít další akce. Nejčastější způsob je publikování PDF souborů do samostatné gitové větve. Tento mechanismus se obvykle používá k vytváření webových stránek a nazývá se „github-pages“.

Pro zápis do repozitáře je nejprve nutné upravit oprávnění v souboru YAML:

```
8     permissions:
9       contents: write
```

Většina akcí, které se používají k nahrávání artefaktů do gitových větví, navíc požaduje adresář namísto jednoho souboru. Proto je nutné všechny soubory přesunout do samostatného adresáře před tím, než tyto akce použijeme:

```
15      - name: Přesuň soubory PDF do adresáře build
16        run: mkdir -p build && mv *.pdf build/
17      - name: Publikuj soubory PDF do gitové větve pdf-output
18        uses: crazy-max/ghaction-github-pages@v4
19        with:
20          build_dir: build
21          target_branch: pdf-output
22        env:
23          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Token na posledním řádku je nutný k tomu, aby postup mohl měnit repozitář.

4.2 Rozdíly oproti Forgejo Actions

Forgejo [16] je další webová služba pro správu kódu, jejímž cílem je být otevřenější než GitHub. Lze ji provozovat na vlastním serveru a poskytuje mechanismy podobné GitHub Actions. Tyto mechanismy nikdy nebudou fungovat zcela stejně jako na GitHubu, ale jsou až na několik výjimek kompatibilní.

Forgejo nejprve hledá konfigurační soubory v adresáři `.forgejo/workflows/`. Pokud žádné nenajde, zkouší je hledat v adresáři `.github/`. Jediný problém, na který pravděpodobně narazíte při použití stejných postupů jako na GitHubu, je ten, že na většině instancí Forgejo buď nejsou dostupné výpočetní uzly, nebo jsou konfigurované odlišně od těch na GitHubu.

Pokud však dokážete nastavit svůj výpočetní uzel na vlastním serveru se stejnými štítky jako na GitHubu, můžete použít stejné konfigurační soubory pro postupy na obou službách [17]. V ukázkových repozitářích na serveru `codeberg.org`, což je instance platformy Forgejo, najdete také popis, jak jsou zde konfigurovány výpočetní uzly.

4.3 Kompilace dokumentu pomocí GitLab CI

Postupy GitLab CI se příliš neliší od GitHub Actions. Dokonce je zde konfigurace o něco jednodušší, protože některé kroky jako stažení repozitáře se provádí automaticky. Konfigurační soubor musí být umístěn v kořenovém adresáři repozitáře a pojmenován `.gitlab-ci.yml`. Zde je náš ukázkový postup:

```
1 ukázkový-postup-gitlab-ci:
2   image: registry.gitlab.com/islandoftex/images/texlive:latest
3   script:
4     - 'latexmk --lualatex'
```

```
  artifacts:
    paths:
    - "./*.pdf"
```

Veškeré kroky v postupu GitLab CI jsou elementární příkazy, které se vykonají jeden po druhém. Náš ukázkový postup obsahuje jen jeden příkaz pro spuštění příkazu `latexmk`. Na rozdíl od GitHubu poskytuje GitLab rozhraní, které umožňuje vytvořit statické odkazy na artefakty. Proto soubor `README.md` v ukázkových repozitářích na GitLabu obsahuje odkaz v následujícím tvaru:

⟨*adresa repozitáře*⟩`/-/jobs/artifacts/`⟨*gitová větev*⟩`/browse?job=`⟨*úloha*⟩

Takovýto odkaz zobrazí všechny artefakty připojené k dané úloze. Např. následující odkaz zobrazí všechny artefakty z ukázkového repozitáře na GitLabu [18]:

`https://gitlab.com/islandoftex/texmf/tug2024-workflow-document-gitlab/-/jobs/artifacts/main/browse?job=check:+[latest,+lualatex]`

5 Testování s více TEXovými distribucemi a s různými stroji

Jak jsme slíbili v Sekci 2.2, je možné postupy rozšířit pro testování kompilace s různými verzemi TEXových distribucí a s různými TEXovými stroji.

Další TEXové stroje můžeme přidat v samostatných krocích s odlišnými elementárními příkazy. Stejně tak můžeme další stroje použít v samostatných úlohách, nebo dokonce v samostatných konfiguračních souborech. Volba závisí pouze na tom, zda chceme spustit vše zároveň, nebo šetřit prostředky pomocí podmíněného nebo sekvenčního spuštění.

Pro použití různých verzí TEX Live poskytuje IoT obrazy `texlive/texlive` všech historických verzí s tagy `TL`⟨*verze*⟩`-historic` a té aktuální s tagem `latest`:

```
jobs:
  testování-s-IoT-obrazy-TeX-Live:
    runs-on: ubuntu-22.04
    strategy:
      matrix:
        tag: ["TL2022-historic", "TL2023-historic", "latest"]
    name: "Test s TeX Live ${{ matrix.tag }}"
    container:
      tag: texlive/texlive:${{ matrix.tag }}
    steps:
```

matrix: Seznam uživatelských parametrů a jejich hodnot, který úlohu přemění na šablonu. Při spuštění postupu vygeneruje šablona úlohy pro všechny kombinace hodnot z matice kartézského součinu parametrů. V našem ukáz-

kovém postupu máme jen jeden parametr `tag` s různými tagy obrazů `texlive/texlive`. K hodnotě parametru můžeme uvnitř šablony přistupovat pomocí syntaxe `${{ matrix.tag }}`. V našem ukázkovém postupu dojde k vygenerování tří úloh, které budou používat obrazy s TEX Live 2022, 2023 a s poslední dostupnou verzí TEX Live (v době psaní článku TEX Live 2024, pozn. red.). Úplný seznam dostupných obrazů najdeme na adrese `https://gitlab.com/islandoftex/images/texlive`.

Postupy GitLab CI také podporují šablony úloh. Následující ukázkový postup ilustruje spuštění několika TEXových strojů na několika různých verzích TEX Live:

```
1 check:
2   image: registry.gitlab.com/islandoftex/images/texlive:$TAG
[ … přeskočili jsme sekce script a artifacts … ]
6   parallel:
7     matrix:
8       - TAG: ['TL2022-historic', 'TL2023-historic', 'latest']
9         STROJ: ['pdflatex', 'xelatex', 'lualatex']
```

V tomto případě je syntax pro přístup k hodnotám parametrů podobná syntaxi proměnných příkazové řádky unixových operačních systémů, ale funguje stejně jako syntax v ukázkovém postupu pro GitHub Actions.

6 Použití malého dockerového obrazu

V předchozích příkladech jsme pro jednoduchost používali plnou instalaci TEX Live.[3] Obrazy od IoT dokonce obsahují všechny závislosti doplňujících nástrojů jako arara nebo minted, takže všechny tyto nástroje fungují bez další konfigurace.

Ne vždy ale potřebujeme plnou instalaci TEX Live a menší obrazy se stahují rychleji a šetří místo na disku výpočetních uzlů.[4] V takovém případě ale musíme vědět, které balíčky z TEX Live potřebujeme nainstalovat pro kompilaci dokumentů…

A zde přichází na scénu nástroj DEPP pro výpis závislostí v TEX Live [20] (anglicky „DEPendency Printer for TEX Live“, pozn. překl.), hrdě vytvořený na Ostrově TEXu.[5] Protože se tento článek zaměřuje na postupy CI/CD, nebudeme nástroj DEPP podrobněji popisovat. Místo toho ukážeme seznam všech balíčků z TEX Live potřebný pro kompilaci, který DEPP vygeneroval pro náš ukázkový projekt [21]:

```
# Proudly generated by the Island of TeX's DEPendency Printer…
blindtext …
```

[3] Nejde o zcela plnou instalaci: použité obrazy neobsahují dokumentaci ani zdrojové kódy.

[4] Pokud vás zajímá, jak snížit nikoliv velikost dockerového obrazu, ale dobu kompilace velkého množství dokumentů, pak doporučujeme vaší pozornosti loňský článek od IoT [13].

[5] Pokud mluvíte německy, možná vás překvapí, že tento nástroj je chytřejší, než by jeho název napovídal. („Depp“ je jihoněmecký výraz pro hlupáka, pozn. překl.)

6.1 GitHub Actions

Na GitHubu je k dispozici několik akcí pro instalaci TEX Live jako součást postupu [22; 23]. Díky tomu můžeme použít menší obraz, což zkrátí dobu kompilace:

```
9       - name: Instaluj TeX Live
10          uses: zauguin/install-texlive@v3
11          with:
12            package_file: .github/tl_packages
```

Tento úryvek umístíme do sekce `steps` před kompilaci dokumentů. Direktiva `package_file` odkazuje na výstup nástroje DEPP nebo na ručně vytvořený seznam balíčků. Direktiva `container` je nyní nadbytečná a můžeme ji odstranit.

6.2 GitLab CI

Na GitLabu je instalace TEX Live trochu složitější, protože nemáme k dispozici akce, ale jen elementární kroky. Repozitář nástroje DEPP proto poskytuje skript `minimal-portable-tl-setup.sh` [24] pro instalaci podle souboru s balíčky:

```
3   before_script:
4     - minimal-portable-tl-setup.sh tl_packages
```

Tento úryvek umístíme pod direktivu `image`, kterou můžeme následně změnit na menší dockerový obraz (jako např. `debian:latest`, pozn. red.).

Další možností je připravit dockerový obraz, který již potřebné balíčky obsahuje. Pokud používáme vlastní výpočetní uzel, obvykle je to nejlepší řešení, protože můžeme obraz udržovat v mezipaměti a aktualizovat ho podle potřeby.

7 Postupy CI/CD pro vývojáře balíčků

Jak jsme již zmínili, výhody používání postupů CI/CD jsou ještě větší, když je použijeme při vývoji balíčků nebo šablon. V takovém případě očekáváme, že repozitář bude obsahovat balíček a nějakou konfiguraci pro nástroj `l3build`.[6]

7.1 GitHub Actions

Příkaz `latexmk` z předchozích ukázek nahradíme příkazem `l3build`:

```
9       - name: Spusť l3build
10        run: l3build check --show-log-on-error -q -H
```

Tento krok spustí všechny testy balíčku podle konfigurace nástroje `l3build`.

[6] Podobné postupy fungují i pro jiné nástroje, ale to již přesahuje rámec tohoto článku.

Protože se jedná o spuštění testovací sady balíčku, artefakty jsou zcela odlišné. Zatímco při kompilaci dokumentu nás zajímá výstup ve formátu PDF, v tomto případě je důležitý výstup testů. Jeden z autorů tohoto článku proto vytvořil akci, která se o tento výstup postará [25]:

```
      - name: Archivuj výstupy neúspěšných testů
        if: ${{ always() }}
        uses: zauguin/l3build-failure-artifacts@v1
        with:
          name: testfiles
          retention-days: 3
```

Kromě nahrání artefaktů tento úryvek ilustruje další důležitý bod: Na rozdíl od PDF dokumentu, kde nás zajímá pouze úspěšný výstup, u testovacích sad nás zajímá především výstup z neúspěšných testů. Přidali jsme proto podmínku `if: ${{ always() }}`. Ta zajistí, že krok bude spuštěn, i když předchozí krok selže.

7.2 GitLab CI

Jak už jsme zmínili u předchozích ukázek, GitLab nativně podporuje nahrávání artefaktů bez potřeby načítání externích rozšíření:

```
  script:
    - l3build check --show-log-on-error -q -H
  artifacts:
    when: on_failure
    paths:
      - ./build/test/*.diff
```

Tím se sice zjednodušuje základní nastavení, ale zároveň se snižuje flexibilita. Pokud bychom například chtěli použít různé artefakty pro úspěšné a neúspěšné kroky, museli bychom pro to použít dvě samostatné úlohy.

8 Lokální spuštění

Během naší prezentace na konferenci TUG 2024 jsme si v rychlosti ukázali, jak můžeme postupy CI/CD spouštět lokálně. To může být užitečné při ladění, protože máme možnost ručně spouštět jednotlivé kroky, nebo když potřebujeme zajistit, aby lokální prostředí odpovídalo výpočetnímu uzlu.

Pro služby GitHub a Forgejo existuje nástroj `act` [26], který umožňuje spouštět akce z postupů GitHub Actions pomocí dockeru. Služba GitLab pak umožňuje jednoduše nainstalovat výpočetní uzel lokálně. Díky tomu můžeme spouštět postupy GitLab CI tak, že v adresáři s repozitářem spustíme příkaz `gitlab-runner exec`.

9 Toto je průběžný vývoj!

Všechny zde uvedené konfigurace jsou samozřejmě pouze příklady a můžeme je rozšířit podle požadavků projektu. Můžeme spouštět libovolné příkazy, což může být nezbytné zejména u složitějších projektů.

Například při vytváření časopisů je možné průběžně generovat tiskovou a online verzi, výňatky jednotlivých článků i výstupy ve formátu HTML/EPUB, přičemž redaktoři musí počkat pouze na kompilaci článku, na kterém právě pracují.

Totéž platí pro studijní materiály, kde můžeme mít vykreslené verze se zahrnutými řešeními i bez nich, nebo dokumenty, které obsahují vzájemně propojené odkazy, a vyžadují proto několik kompilací.

Integrace postupů CI/CD s pokročilejší prací s gitem, jako je smysluplný koncept větvení, může také zlepšit spolupráci nebo zjednodušit proces přispívání do open source projektů. To sníží pracovní zátěž správců projektů, protože některé problémy nebude třeba kontrolovat ručně.

10 Závěr, výzva k zapojení a žádost o zpětnou vazbu

V článku jsme si ukázali, jak můžeme používat služby GitHub, Forgejo a GitLab pro kompilaci dokumentů a vývoj balíčků v (LA)TEXu. Předvedli jsme si, jak můžeme využít různé verze distribuce TEX Live i různé TEXové stroje pro usnadnění testování napříč platformami. Také jsme se postarali o to, aby bylo možné přistoupit k výsledným souborům PDF nebo k výsledkům testování.

Doufáme, že tato řešení povedou ke zvýšení stability všech částí vývoje v (LA)TEXu tím, že usnadní testování balíčků a ušetří čas při kompilaci složitějších projektů.

Pokud spravujete nějaké balíčky, bylo by skvělé, kdybyste mohli zkusit nastavit testy, které by ověřovaly jejich funkčnost proti nejnovějšímu vydání TEX Live (v době psaní článku TEX Live 2024, pozn. red.), aktuální vývojové verzi TEX Live, nebo dokonce proti testovacím verzím TEX Live před dalším vydáním [27], pro které IoT od roku 2024 připravuje dockerový obraz `texlive/texlive` s tagem `pretest`. Pokud narazíte na problémy, rádi vám pomůžeme.

Rádi bychom tento tutoriál dále udržovali a zjednodušili použití automatizace pro uživatele TEXu a jeho přátel. Pokud objevíte nějaké nejasnosti, budeme rádi, když se nám ozvete, abychom mohli tento tutoriál vylepšit. Tabulka 1 na straně 61 shrnuje všechny související ukázkové repozitáře.

Vaše příspěvky do tohoto projektu jsou vítány!

Odkazy

1. PEISCHL, Marei; KRÜGER, Marcel; KOPP, Oliver. Creation of LATEX documents using a cloud-based pipeline. *TUGboat*. 2024, **45**(2), 227–233.
2. NOVOTNÝ, Vít. Overleaf: Kolaborativní webový editor LATEXu. *Zpravodaj CSTUGu*. 2021, **31**(1–4), 3–8. Dostupné z DOI: 10.5300/2021-1-4/3.

3. STANO, Michal; STARÝ NOVOTNÝ, Vít. Overleaf Community Edition: Postup inštalácie a rozdiely oproti Overleaf Professional [online]. [B.r.] [cit. 2024-09-03]. Dostupné z: http://www.overleaf.com/read/hmxhshbgnbzm.
4. *The LaTeX Project* [online]. [cit. 2024-08-29]. Dostupné z: https://github.com/latex3/.
5. THE LaTeX PROJECT. *l3build: A testing and building system for (La)TeX* [online]. [cit. 2024-08-29]. Dostupné z: https://ctan.org/pkg/l3build.
6. WRIGHT, Joseph. l3build: The beginner's guide. *TUGboat*. 2022, **43**(1), 40–43. Dostupné z DOI: 10.47397/tb/43-1/tb133wright-l3build.
7. PEISCHL, Marei; KRÜGER, Marcel; KOPP, Oliver. *TeXhackse/tugboat--tug2024-cicd* [online]. [cit. 2024-08-30]. Dostupné z: https://tug.org/l/peischl-cicd2024.
8. XU, Cheng. *xu-cheng/latex-action: GitHub Action to compile LaTeX documents* [online]. [cit. 2024-09-03]. Dostupné z: https://github.com/xu-cheng/latex-action/.
9. *Workflow syntax for GitHub Actions* [online]. [cit. 2024-09-03]. Dostupné z: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions.
10. *Choosing GitHub-hosted runners* [online]. [cit. 2024-09-03]. Dostupné z: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#choosing-github-hosted-runners.
11. *GitHub Workflow template for LaTeX packages* [online]. [cit. 2024-09-03]. Dostupné z: https://github.com/islandoftex/tug2024-workflow-github.
12. ISLAND OF TEX. Providing Docker images for TeX Live and ConTeXt. *TUGboat*. 2019, **40**(3), 231. Dostupné také z: https://tug.org/TUGboat/tb40-3/tb126island-docker.pdf.
13. ISLAND OF TEX. Living in containers: on TeX Live (and ConTeXt) in a Docker setting. *TUGboat*. 2023, **44**(2), 249–252. Dostupné z DOI: 10.47397/tb/44-2/tb137island-docker.
14. *Checkout action* [online]. [cit. 2024-09-03]. Dostupné z: https://github.com/actions/checkout/.
15. COLLINS, John. *latexmk: Fully automated LaTeX document generation* [online]. [cit. 2024-09-03]. Dostupné z: https://ctan.org/pkg/latexmk.
16. FORGEJO. *Forgejo Repository* [online]. [cit. 2024-07-27]. Dostupné z: https://codeberg.org/forgejo/forgejo.
17. *Using Forgejo Actions (Self-hosted)* [online]. [cit. 2024-09-10]. Dostupné z: https://docs.codeberg.org/ci/actions/.
18. *GitLab Workflow template for LaTeX documents* [online]. [cit. 2024-09-10]. Dostupné z: https://gitlab.com/islandoftex/texmf/tug2024-workflow-document-gitlab.

19. *TEX Live Docker image* [online]. [cit. 2024-07-20]. Dostupné z: https://gitlab.com/islandoftex/images/texlive.
20. *DEPP: Dependency Printer for TEX Live* [online]. [cit. 2024-09-15]. Dostupné z: https://gitlab.com/islandoftex/texmf/depp.
21. KOPP, Oliver. *File tl_packages* [online]. [cit. 2024-09-15]. Dostupné z: https://github.com/islandoftex/tug2024-workflow-document-github/blob/main/.github/tl_packages.
22. KRÜGER, Marcel. *teatimeguest/setup-texlive-action: A GitHub Action to set up TEX Live* [online]. [cit. 2024-09-15]. Dostupné z: https://github.com/zauguin/install-texlive.
23. *teatimeguest/setup-texlive-action repository* [online]. [cit. 2024-09-15]. Dostupné z: https://github.com/teatimeguest/setup-texlive-action.
24. PEISCHL, Marei. *File minimal-portable-tl-setup.sh* [online]. [cit. 2024-09-16]. Dostupné z: https://gitlab.com/islandoftex/texmf/depp/-/blob/main/setup-examples/texlive-minimal-portable/minimal-portable-tl-setup.sh.
25. KRÜGER, Marcel. *zauguin/l3build-failure-artifacts* [online]. [cit. 2024-09-16]. Dostupné z: https://github.com/zauguin/l3build-failure-artifacts.
26. *nektos/act: Run your GitHub Actions locally* [online]. [cit. 2024-09-16]. Dostupné z: https://github.com/nektos/act.
27. *Pretesting TEX Live 2024* [online]. [cit. 2024-09-16]. Dostupné z: https://www.tug.org/texlive/pretest.html.

Creation of LaTeX documents using a cloud-based pipeline

Using web-based platforms for collaborative editing of LaTeX documents is common these days. These tools focus on writing documents and not on creation of templates or packages. Using build servers with automated pipelines is common within software development but can easily be adapted for TeX & friends.

This article will show how to get started using automated workflows on platforms like GitHub, Forgejo, or GitLab for document authors as well as TeX developers.

Keywords: continuous integration & delivery, GitHub, Forgejo, GitLab, docker

Marei Peischl
Gneisenaustr. 18
202 53 Hamburk
Německo
marei@peitex.de
https://peitex.de

Marcel Krüger
Hamburk
Německo

Oliver Kopp
Sindelfingen
Německo