Zip & Go: Samodejno nameščanje z zabojniki v produkcijo

Boštjan Praznik, Nejc Maleš

Novum-RGI Germany GmbH Podružnica Maribor, Maribor, Slovenija
bostjan.praznik@rgigroup.com, nejc.males@rgigroup.com

ZIP & GO predstavlja svež in praktičen pristop k dinamičnemu posodabljanju Node.js mikrostoritev v produkcijskem Docker okolju – brez potrebe po ustavitvi storitev ali ponovni gradnji zabojnikov. Namesto klasičnih, pogosto zapletenih CI/CD postopkov, omogoča ta rešitev preprost vnos nove funkcionalnosti z ZIP paketom, ki se samodejno razpakira znotraj zabojnika in sproži vnovični zagon aplikacije z novo vsebino. Z uporabo preizkušenih tehnologij, kot so Node.js, PM2 in Docker, rešitev ponuja visoko modularnost, zanesljivost in izjemno hitro implementacijo sprememb brez izpada storitev. Sistem je zasnovan za ekipe, ki potrebujejo učinkovito, robustno in dolgoročno vzdržno strategijo za upravljanje mikrostoritev. Članek skozi konkretne primere, konfiguracije in skripte pokaže, kako lahko z nekaj vrsticami kode in pravim arhitekturnim pristopom bistveno poenostavimo produkcijsko upravljanje aplikacij.

Ključne besede:
Docker
Node.js
PM2
Mikrostoritve
CI/CD

Univerzitetna založba Univerze v Mariboru

ISBN 978-961-299-037-4
DOI https://doi.org/10.18690/um.feri.7.2025.6

1 Uvod

V sodobnih informacijskih sistemih postaja potreba po hitrem, zanesljivem in nemotenem posodabljanju programske opreme vse bolj izrazita. Razvoj mikrostoritvene arhitekture, podprt z orodji za avtomatizacijo in upravljanje z zabojniki, kot so Docker, Kubernetes [4] in PM2, je močno spremenil pristop k razvoju in vzdrževanju aplikacij. Kljub tem napredkom ostaja izziv, kako v produkcijskem okolju zagotoviti dinamično in čim bolj nemoteno nadgradnjo vsebine mikrostoritev, ne da bi bilo potrebno ponovno ustvarjanje (angl. rebuild) zabojnikov.

Predstavljena rešitev z imenom "ZIP & Go"[1] omogoča prav to: hitro in varno posodabljanje z uporabo ZIP datotek, ki vsebujejo nove konfiguracije ali funkcionalne module. Ključna prednost tega pristopa je, da zabojnik teče neprekinjeno, posodobitve pa se izvajajo zgolj na nivoju vsebine preko samodejnega razpakiranja ZIP paketa in ponovnega zagona aplikacije z novo vsebino.

2 Pristop k rešitvi

Glavni cilj ZIP & Go pristopa je ločitev zabojniške infrastrukture od poslovne logike. S tem omogočimo, da se funkcionalni del sistema (npr. kalkulacijski moduli) posodablja neodvisno od zabojnika samega. Ta cilj dosežemo z naslednjimi koraki:

- ZIP datoteka vsebuje .mjs module (ESM format), ki predstavljajo poslovno logiko,
- ZIP se postavi v zunanji volumen, dostopen zabojniku,
- Skripta (watcher) nadzira ZIP datoteko in ob spremembi sproži razpakiranje,
- Aplikacija se ponovno zažene, tokrat z novo poslovno logiko.

Tako zagotovimo modularnost, hitrost in zanesljivost. Aplikacija se ne prekine dlje kot nekaj milisekund, vse spremembe pa ostanejo lokalizirane na nivoju poslovne logike.

3 Uporabljena orodja

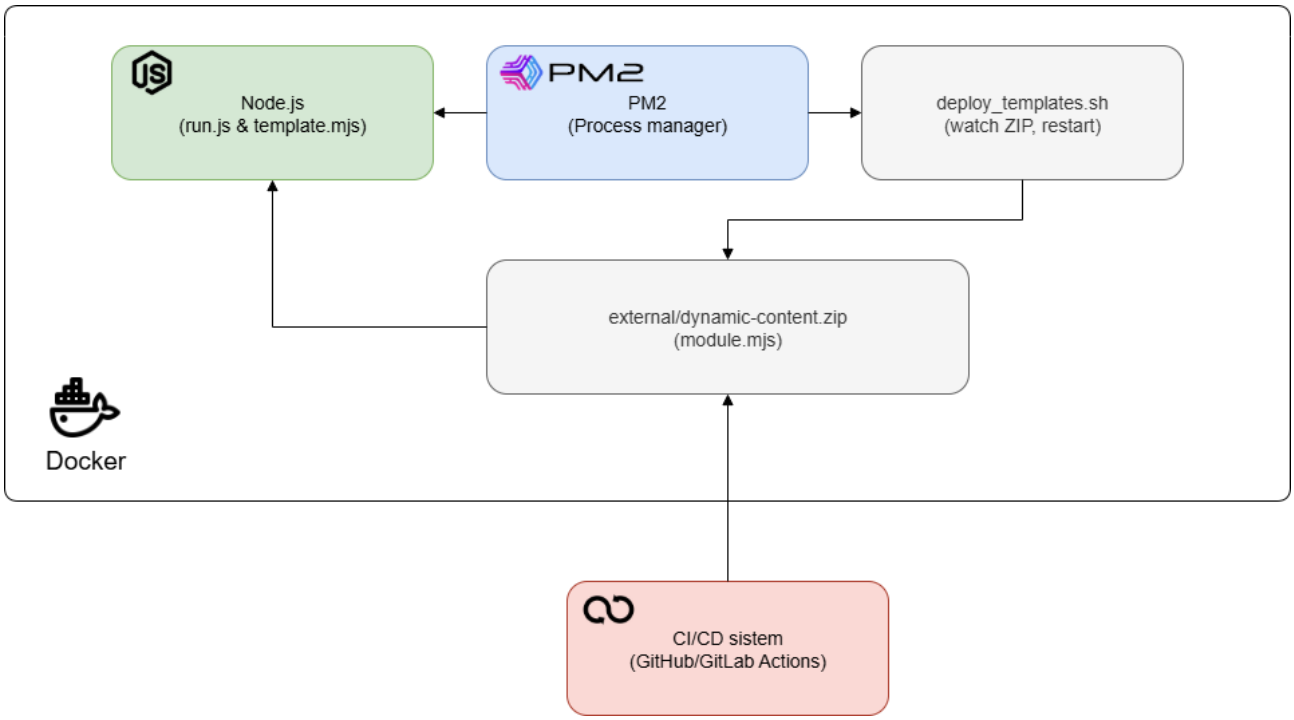
Orodja, uporabljena v opisani rešitvi:

- **Node.js:** Osnova mikrostoritve. Z uporabo ES modulov (.mjs) omogoča dinamično nalaganje logike. Poslovna pravila so definirana kot moduli, ki jih aplikacija ob vsakem zagonu uvozi.
- **PM2:** Upravitelj Node.js procesov. Omogoča nadzor, ponovni zagon in spremljanje delovanja mikrostoritve. Njegova podpora za konfiguracijo (.config.cjs) olajša upravljanje več procesov.
- **Docker:** Sistem za upravljanje z zabojniki poskrbi za izolirano okolje, v katerem teče aplikacija. ZIP datoteka je dostopna preko priklopljenega volumna, kar pomeni, da se vsebina zabojnika fizično ne spremeni.
- **Bash:** Preprosta, a učinkovita skriptna rešitev za razpakiranje vsebine ZIP in nadzor izvajanja PM2 procesov.

[1] Zip & Go je ime rešitve, kreirano in uporabljeno izključno za namene konference OTS.

4 Arhitektura

Arhitektura sistema je zasnovana modularno, z jasnim ločevanjem med infrastrukturo (Docker zabojnik, PM2, skripte) in poslovno logiko (ES moduli v ZIP datoteki). Namen je omogočiti vročo zamenjavo poslovne logike brez potrebe po rekonstrukciji zabojnika.



Slika 1: Arhitektura opisane rešitve.

4.1. CI/CD sistem

Zunanja komponenta, npr. GitHub Actions ali GitLab CI, samodejno generira ZIP datoteko, ki vsebuje .mjs module (poslovno logiko). Ko razvijalec potrdi spremembo v repozitoriju:

- se izvede testiranje,
- ustvari ZIP paket,
- prenese ZIP v zunanji volumen, dostopen zabojniku.

4.2. Docker zabojnik

Zabojnik vsebuje vnaprej pripravljeno osnovno okolje:

- mapo /opt/myapp/engine/ z glavno skripto run.js in nadzorno skripto deploy_templates.sh,
- mapo /opt/myapp/external/, kamor se namešča ZIP datoteka z novejšo poslovno logiko,
- referenčno datoteko template.mjs, ki se uporabi ob zagonu mikrostoritve.

Zabojnik je operativen ves čas – ne rekonstruira se, kar bistveno zmanjša čas uvajanja sprememb.

Drevesni prikaz strukture zabojnika:

```
/opt/myapp/
├── engine/
│   ├── run.js
│   └── deploy_templates.sh
├── external/
│   └── dynamic-content.zip
└── template.mjs
```

4.3. PM2 procesni nadzornik

V zabojniku se izvajata dva procesa:

- watcher: nadzira ZIP datoteko in ob spremembi sproži skripto za razpakiranje,
- myapp: Node.js paket, ki izvaja poslovno logiko iz .mjs modula.

PM2 omogoča avtomatiziran ponovni zagon aplikacije po razpakiranju nove logike in olajša spremljanje izpadov.

Primer konfiguracije:

```
module.exports = {
  apps: [
    {
      name: "watcher",
      script: "/opt/myapp/engine/deploy_templates.sh",
      watch: ["/opt/myapp/external/dynamic-content.zip"],
      watch_options: { followSymlinks: false, usePolling: true },
      autorestart: false,
      exec_mode: "fork"
    },
    {
      name: "myapp",
      script: "/opt/myapp/engine/run.js",
      watch: false,
      args: "8080 file:///opt/myapp/template.mjs",
    }
  ]
};
```

4.4. Bash skripta (deploy_templates.sh)

Skripta, ki jo sproži watcher, izvede naslednje:

- razpakira ZIP v ustrezno mapo (ponavadi /opt/myapp/),
- preveri vsebino (po potrebi z zgoščeno vrednostjo ali podpisom),
- ponovno zažene glavno aplikacijo z novo logiko.

Gre za preprosto rešitev, ki je hitro prenosljiva in enostavna za vzdrževanje.

Primer skripte:

```
#!/bin/bash

echo "Executing deploy_templates.sh..."
echo "ENV: $ENV"

ZIP_FILE="${BASE_DIR}/external/dynamic-content.zip"
EXTRACT_DIR="${BASE_DIR}"

# Extract ZIP, overwrite existing files
unzip -o "$ZIP_FILE" -d "$EXTRACT_DIR"

# Restart the app with PM2
pm2 restart myapp || pm2 start server.js --name myapp
```

4.5. Node.js mikrostoritev

Glavna aplikacija (run.js) prejme pot do .mjs modula kot parameter:

- node run.js 8080 file:///opt/myapp/template.mjs

V aplikaciji se .mjs modul dinamično uvozi, kar omogoča prilagoditev logike med izvajanjem brez potrebe po spremembi osnovnega sistema.

4.6. Zunanji volumen

ZIP datoteka se ne prenaša neposredno v zabojnik, temveč v zunanji volumen (npr. Kubernetes PersistentVolume), ki ga zabojnik le bere. To omogoča:

- delitev vsebine med več zabojnikih,
- zmanjšanje varnostnega obsega sprememb,
- ohranitev ločenosti med infrastrukturnim in logičnim delom sistema.

5 Težave in omejitve

Implementacija prinaša nekatere tehnične in varnostne izzive:

- **Polling obremenitev:** Neprestano preverjanje sprememb lahko obremeni sistem, a gre za najzanesljivejši način v zabojniških okoljih, kjer inotify pogosto ni na voljo.
- **ZIP struktura:** Zahteva strogo določeno strukturo (npr. imeniki, imena datotek). Napačno strukturiran ZIP lahko povzroči zlom aplikacije.
- **Dostopna dovoljenja:** Zabojnik potrebuje pravice za zapisovanje v ciljna mesta in izvajanje skript.
- **Napake ob zagonu:** Napačen .mjs modul lahko povzroči, da se aplikacija ne zažene. Potrebna je validacija pred zagonom (npr. z uporabo testnega okolja ali digitalnih podpisov).
- **Varnostni vektorji:** ZIP je lahko vektor za zlonamerno kodo. Priporočamo preverjanje vsebine pred razpakiranjem (npr. s SHA256 ali podpisanimi ZIP paketi).

6 Integracija v CI/CD

ZIP paket lahko postane artefakt v CI/CD verigi. Ob vsaki potrjeni spremembi repozitorija se izvede:

- testiranje funkcionalnosti modulov,
- generiranje ZIP paketa,
- prenos ZIP v zunanji volumen (npr. preko SFTP, SCP ali neposredno v omrežno mapo).

Primer CI/CD skripte:

```
zip_and_go:
  stage: deploy
  script:
    - npm run test
    - zip -r dynamic-content.zip ./modules
    - scp dynamic-content.zip user@host:/external/
```

7 Primer uporabe

V našem podjetju razvijamo programsko opremo za področje zavarovalništva. Zavarovalnice, s katerimi sodelujemo, imajo različne zahteve, ponujajo svoje zavarovalniške produkte in delujejo v različnih državah in geografskih okoljih. Produkte in pakete, ki jih ponujajo zavarovalnice, običajno pripravljajo zavarovalniški aktuarji, ti pa imajo omejeno tehnično znanje in dostope do produkcijskih sistemov, na katerih je nameščena programska oprema. V preteklosti smo morali vsako zahtevano spremembo podpreti v razvojnem oddelku. Da bi se temu izognili, smo kot podporo glavnemu delu programskega paketa razvili mikrostoritev za izračun zavarovalniških parametrov ter ga zapakirali v okolje, ki stranki omogoča, da ga pod posebnimi varnostnimi pogoji, lahko nadgradi sama. Tako ob spremembi tehnični ekipi ni več potrebno sestavljali nov Docker image, temveč uporabnik sam spremeni le vsebino in po potrebi tudi poslovno logiko, ogrodje in podporni deli pa ostanejo statični.

Prednosti takšnega pristopa so:

- hitra namestitev (v manj kot 1s),
- noben docker image se ne spreminja,
- razvijalci se lahko osredotočajo le na logiko,
- varen, ker se uporablja preverjanje vsebine.

8 Zaključek

ZIP & Go predstavlja sodoben in pragmatičen pristop za obvladovanje posodobitev mikrostoritev v produkcijskem okolju. Med njegove prednosti sodijo:

- rekonstrukcija zabojnika ni potrebna,
- hitro nalaganje novih vsebin,
- manjša kompleksnost pri razvoju,
- enostavna integracija v obstoječe DevOps tokove,
- prilagodljivost in razširljivost.

Čeprav obstajajo varnostna in tehnična tveganja, jih je z dobrimi praksami (npr. preverjanje vsebine ZIP, nadzor dostopov) mogoče učinkovito omiliti. Ta pristop je še posebej primeren za podjetja, ki si želijo večjo agilnost in boljšo odzivnost na spremembe poslovnih pravil brez žrtvovanja stabilnosti infrastrukture.

Literatura

[1] https://pm2.keymetrics.io/: Advanced, production process manager for Node.JS

[2] https://nodejs.org/en: Free, open-source, cross-platform JavaScript runtime environment

[3] https://www.docker.com/: Foundation for secure, intelligent development

[4] https://kubernetes.io/: Kubernetes, Production-Grade Container Orchestration

[5] https://www.flaticon.com/free-icons: slike, uporabljene v glavni sliki arhitekture (Docker created by orvipixel, Node.js created by Grand Iconic, CI/CD created by Ida Desi Mariana)