Building large-scale simulation pipelines using targets, Git and GitHub Actions

useR! 2024 Salzburg

Sergio Olmos

2024-07-11

About

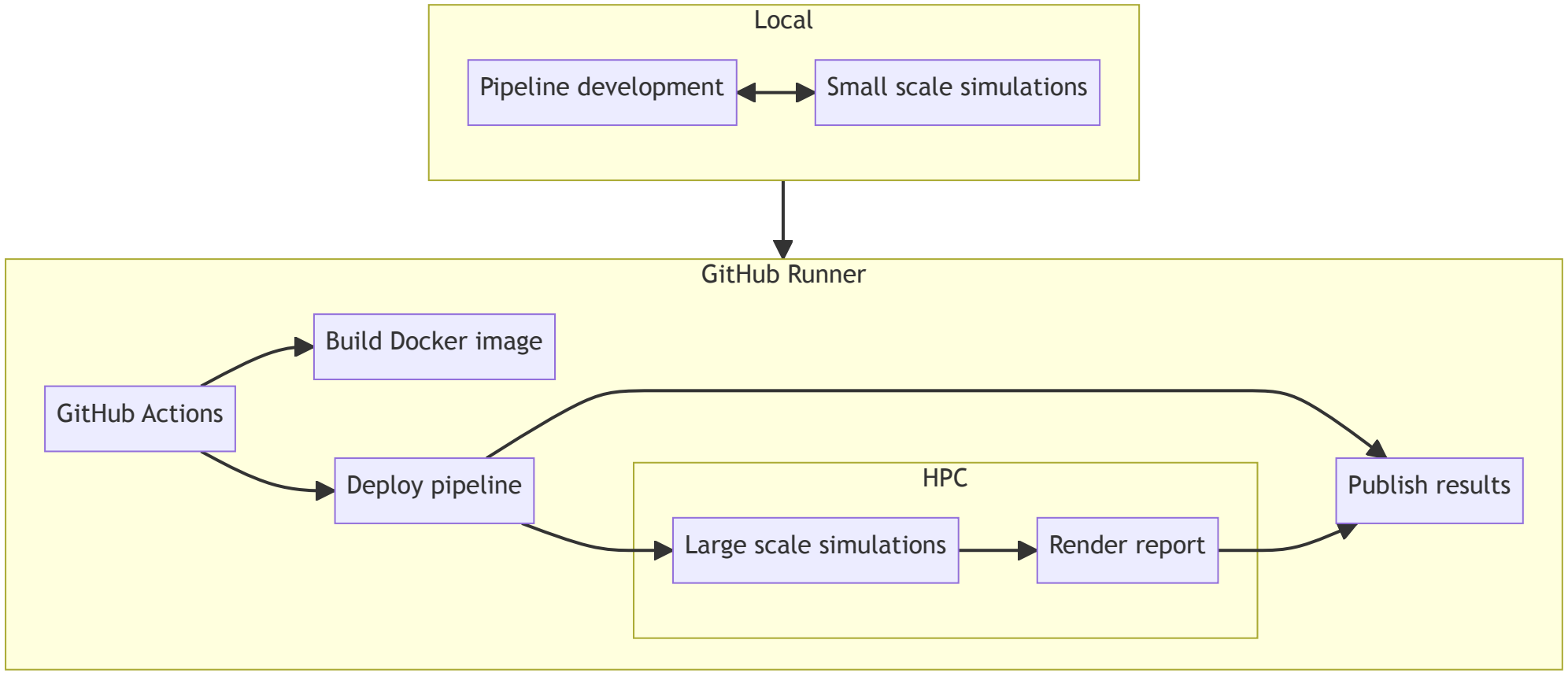
- Statistician at Sanofi
- Member of openstatsware
- Innovative clinical trial designs and analytical methods
- Large scale simulations
- Software engineering best practices

Overview

Tools

- {targets}
- Docker
- Git
- GitHub Actions

Pipeline


Local
Pipeline development
Small scale simulations
GitHub Runner
Build Docker image
GitHub Actions
Deploy pipeline
HPC
Large scale simulations
Render report
Publish results

{targets}

Pipeline management

R-based

```r
# _targets.R
library(targets)
tar_option_set(
  packages = c("dplyr", "ggplot2")
)

list(
  tar_target(
    name = data_sim,
    command = sim_data(reps = 100)
  ),
  tar_target(
    name = model_fit,
    command = fit_model(data_sim)
  )
)
```

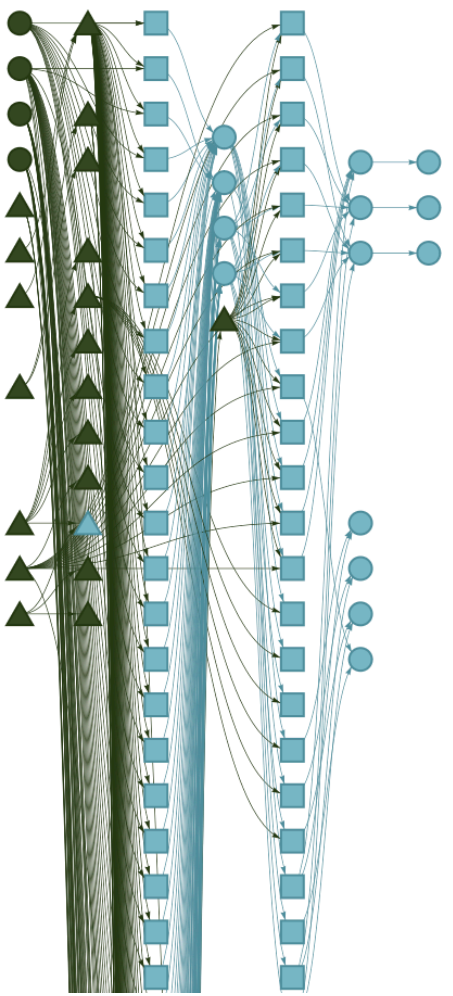
Run entire pipeline with:

```r
targets::tar_make()
```

Parallelization via {crew}

```
# _targets.R
library(crew)
tar_option_set(controller = crew_controller_local(workers = 30))
```

- Dynamic branching
  - {tarchetypes}
- Distributed computing:
  - Local
  - SGE
  - SLURM
  - AWS Batch



Seamless testing-deployment workflow via {config}

config.yaml

```yaml
default:
  sims: 100
  workers: 2
deployment:
  sims: 10000
  workers: 30
```

_targets.R

```r
n_sims <- config::get("sims")
n_workers <- config::get("workers")
tar_option_set(crew_controller_local(workers = n_workers))
list(
  tar_target(data_sim, sim_data(reps = n_sims))
)
```

Cloud computing

Cloud computing platform

- On-demand AWS EC2 instances
- Launch instance/cluster via API
- Custom Docker image
- Access to S3 buckets for persistent storage
- Choose HPC scheduler: SLURM, SGE, Kubernetes
- Live pipeline logs

Docker

Install all pipeline requirements

Production environment:

- Base images: Rocker Project
- System dependencies
- R version
- R packages

GitHub Actions

What are GitHub Actions?

- Workflow automation: CI/CD
- Runners:
  - GitHub-shared
  - Self-hosted
- Traceability
- Collaboration

Workflow

Before CI/CD

- Manually moving files
- Working in ineffient mounted folders
- Running costly cloud instance for long periods of time (mostly iddle)
- Interactive testing in the command line
- Non-traceability of results

After CI/CD

Local development:

- Free
- Offline
- Your development environment
  - Desktop IDE
  - Git client
- Interactive testing

Automated pipeline deployment:

- No manual file movement
- Traceable
- Efficient use of costly cloud computing

Build image

- Trigger workflow only when changes in:
  - `Dockerfile`
  - Dependencies (e.g., `renv.lock`)
- Build image in GitHub runners
- Push image to cloud computing registry

Deploy pipeline

- Custom trigger:
  - `git push <deploy-branch>`
  - `git tag`
- Custom GitHub Action:
  - Specify Docker image
  - Specify HPC specs
  - Automatated `git checkout`
  - Pipeline execution command

Deployment example

```yaml
name: Deploy pipeline
on:
  push:
    branches: deploy
jobs:
  deploy-pipeline:
    runs-on: self-hosted
    steps:
      - name: Run pipeline
        uses: org/custom-github-action/run-pipeline@main
        with:
          token: ${{ secrets.API_TOKEN }}
          docker-image: user/custom-docker-image:latest
          instance-type: r6i.32xlarge
          command: sh _deploy.sh # targets::tar_make()
```

Publish results

After deployment workflow:

- targets pipeline includes Quarto/RMarkdown documents reporting results

```
tarchetypes::tar_quarto(report, "report.qmd")
```

- Publish rendered HTML to GitHub Pages/Posit Connect
- See results
- Share with collaborators

Questions?

sergio.olmos.pardo@gmail.com

solmos

sergio-olmos

@solmos@fosstodon.org