Common Pitfalls in GitOps Workflows

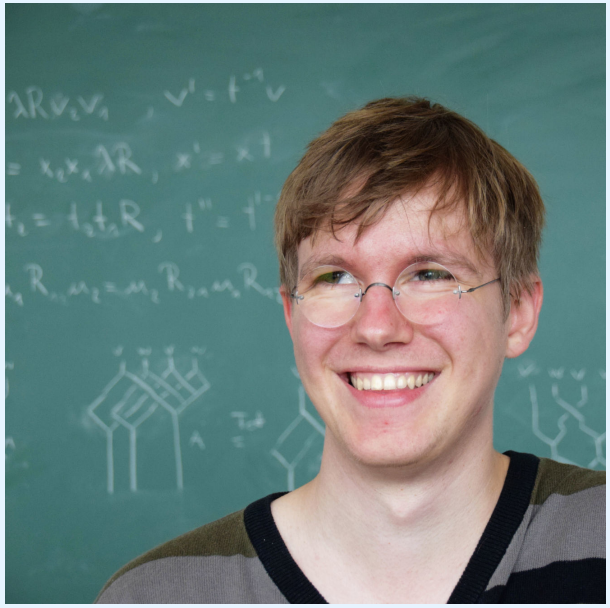
Pascal Fries

über mich

- in der IT seit 2021
- IT Consultant bei der ATIX
- Schwerpunkte:
  - Kubernetes
  - CI / CD
  - IaC
  - GitOps

davor

- theoretischer Physiker
- Schwerpunkte:
  - Quanteninformation
  - Quantenfeldtheorie
  - Quantengravitation
  - String Theorie



Kollaboration an LaTeX Projekten...

PHYSICAL REVIEW LETTERS **123,** 211603 (2019)

Entanglement Spectrum of Chiral Fermions on the Torus

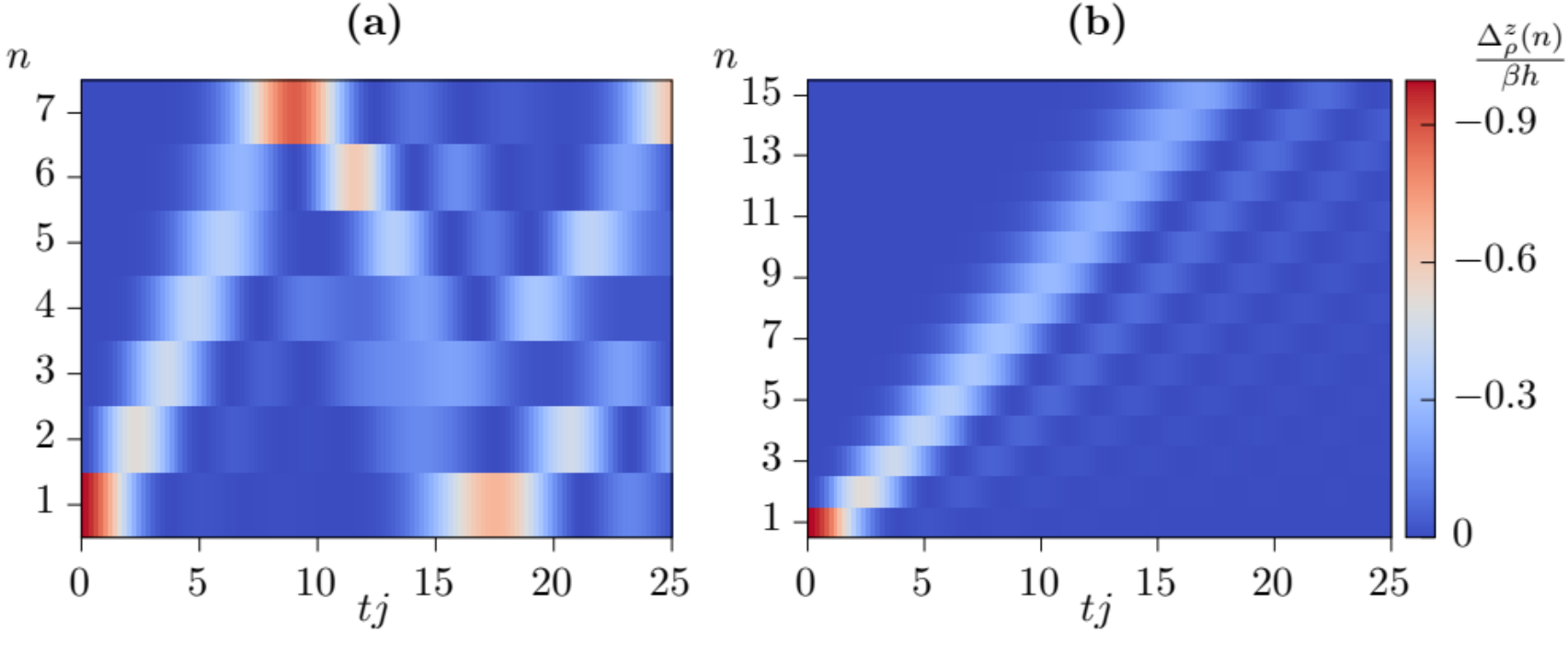
Pascal Fries[1,*] and Ignacio A. Reyes[2,†]
[1]*Fakultät für Physik und Astronomie, Julius-Maximilians Universität Würzburg, Am Hubland, 97074 Würzburg, Germany*
[2]*Max-Planck-Institut für Gravitationsphysik, Am Mühlenberg 1, 14476 Potsdam, Germany*

(Received 15 May 2019; published 19 November 2019)

We determine the modular Hamiltonian of chiral fermions on the torus, for an arbitrary set of disjoint intervals at generic temperature. We find that, in addition to a local Unruh-like term, each point is nonlocally coupled to an infinite but discrete set of other points, even for a single interval. These accumulate near the boundaries of the intervals, where the coupling becomes increasingly redshifted. Remarkably, in the presence of a zero mode, this set of points "condenses" within the interval at low temperatures, yielding continuous nonlocality.

DOI: 10.1103/PhysRevLett.123.211603

Entwicklung von HPC für Simulationen...



Arbeiten mit spezialisierter Software...

ideales System

- wiederherstellbar
- versionierbar
- nachvollziehbar

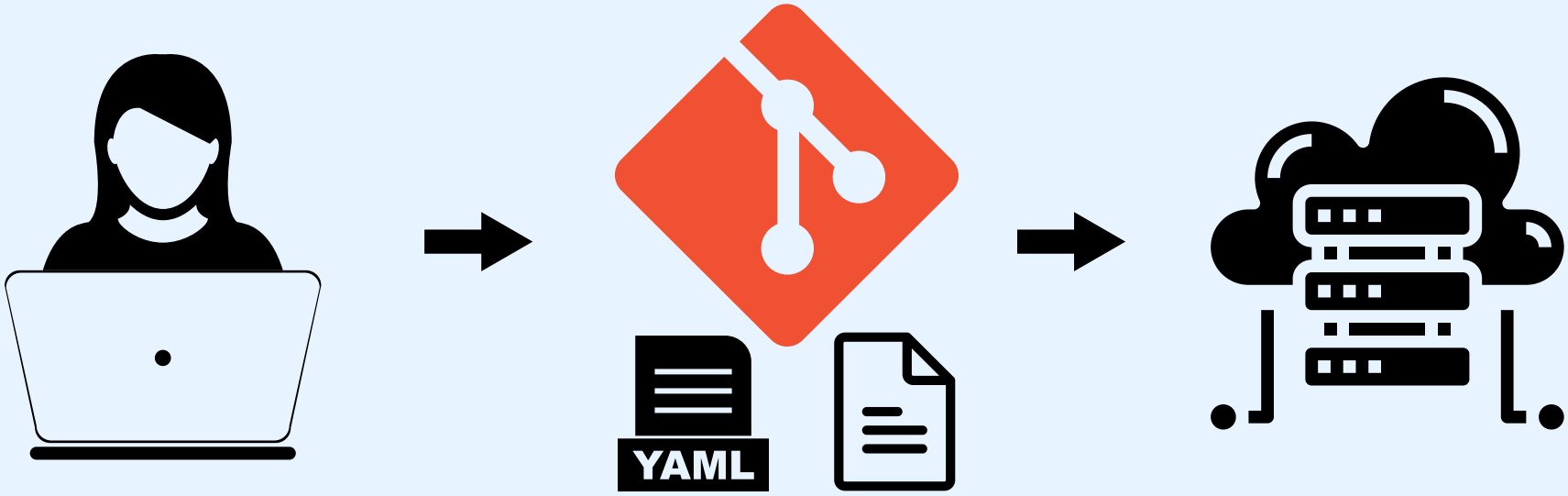
Die Lösung

deklarativ

IaC

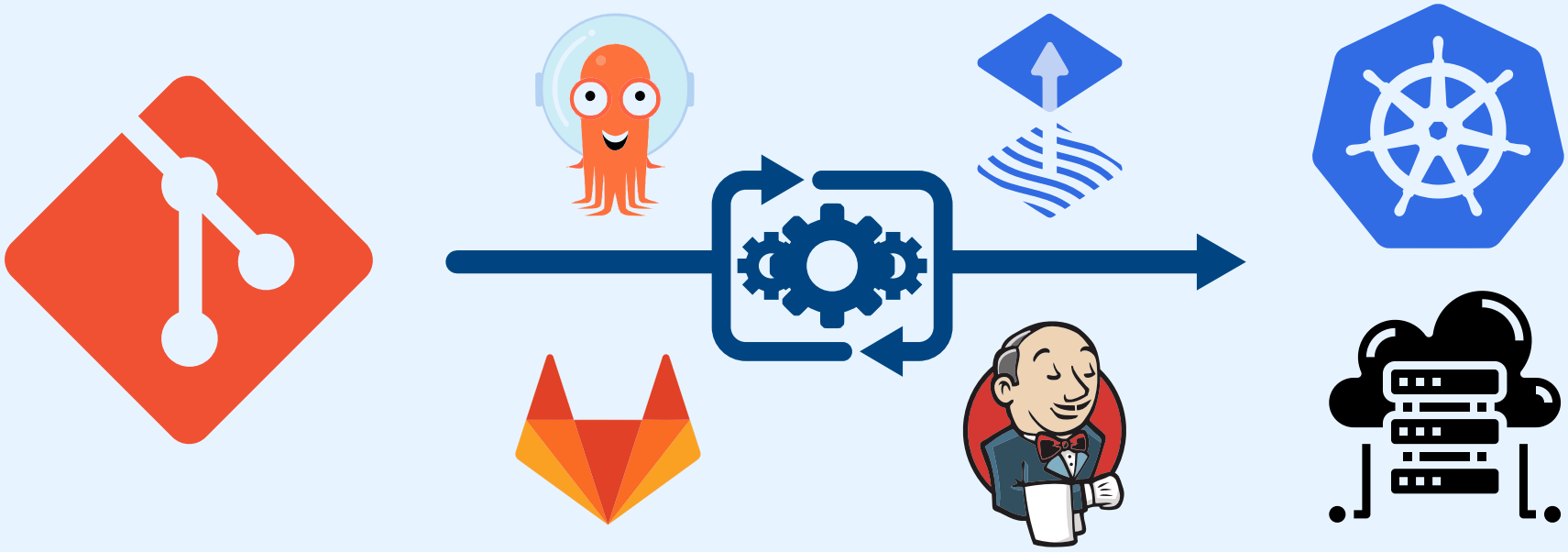
automatisiert

GitOps



"Git als die **Single Source of Truth**."

GitOps Operatoren



Aber wie...

- gehe ich mit empfindlichen Daten um?
- verwalte ich Berechtigungen?
- oragnisiere ich Release Cycles?

Empfindliche Daten

"Git vergisst nicht."

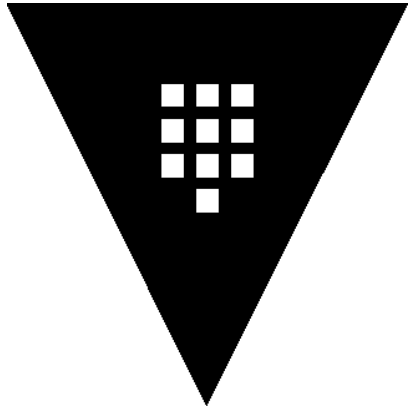
```
echo '{"root_password": "IAmGRoot"}' >> root.json
git add root.json
git commit -m "include root_password in root.json"
git push
```

Keine Secrets

unverschlüsselt im Git!

Externes Secret Management

- keine Secrets im Git
- Abrufen On Demand
- automatische Key Rotation
- eigene Konfiguration und Wartung
- Umdenken bei bestehenden Deployments



Secrets verschlüsselt im Git

Pre-Commit: **ver**schlüsseln mit öff. Schlüssel

Deployment: **ent**schlüsseln mit priv. Schlüssel

Entschlüsselung

- durch Operator (z.B. *SOPS*) oder
- im Zielsystem (z.B. *SealedSecrets*)

Berechtigungen

"Vor Git sind wir alle gleich."

```
git checkout deploy/prod
git rm -rf config/common
git commit -m "what could go wrong?"
git push
```

Nutze Rechteverwaltung

der Git Remote

Monorepo

```
atix-gitops.git
├── kube-prometheus-stack
│   └── ...
├── gitlab-runners
│   └── ...
├── wordpress-atix
│   └── ...
└── wordpress-orcharhino
    └── ...
```

Branches:

`master`, `deploy/testing`, `deploy/prod`

Sicherheitsmechanismen

- Änderungen an `deploy/*`-Branches nur über PRs
- Merge nach `deploy/prod` nur durch Admins

Anwendungsbereich

- Empfehlenswert für "kleine" Projekte.
- Admins sind für **alles** verantwortlich

Multirepo

```
atix-gitops-infra.git
├── kube-prometheus-stack
│   └── ...
└── gitlab-runners
    └── ...
```

```
atix-gitops-websites.git
├── wordpress-atix
│   └── ...
└── wordpress-orcharhino
    └── ...
```

Zusätzlich

- Eigenes Repo pro Team (i.S.v. DevOps)
- Eigene Schlüssel für Secret-Management

Anwendungsbereich

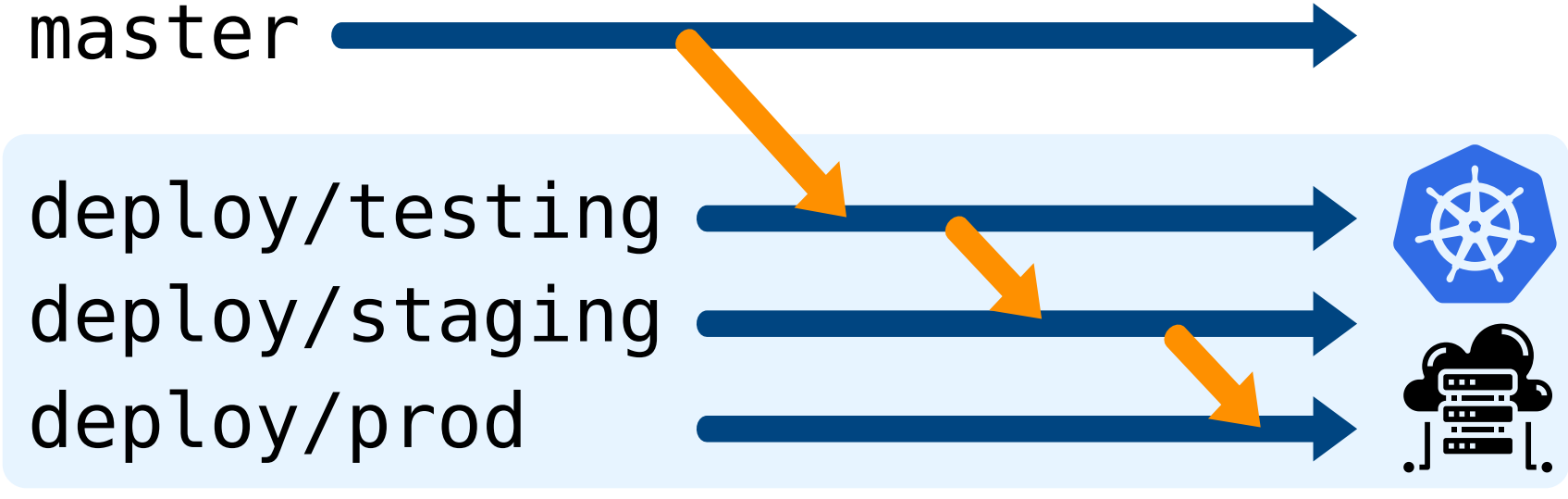
- Empfehlenswert für "große" Projekte.
- Jedes Team hat **eigene** Admins.

Release-Cycles

"Die drei Zweige bedeuten drei Tage." (1 Mose 40.12)

```
git checkout deploy/prod
git pull
git merge deploy/testing -m "rollout new runner config"
gt push
```

Benutzt Branches!


master
deploy/testing
deploy/staging
deploy/prod

aber Code-Anpassungen

sind umgebungsspezifisch

Bitte nicht...

```
atix-some-app.git?ref=deploy%2Ftest-a
├── base/...
└── config-test-a/...
```

```
atix-some-app.git?ref=deploy%2Ftest-b
├── base/...
└── config-test-b/...
```

```
atix-some-app.git?ref=deploy%2Fprod
├── base/...
└── config-prod/...
```

Benutzt Branches,

die man mergen kann.

Base und Config gemeinsam

```
atix-some-app.git
├── base/...
├── config-test-a/...
├── config-test-b/...
└── config-prod/...
```

	testing	prod
config-test-a	X	
config-test-b	X	
config-test-prod		X

- leichter umzusetzen
- gut wenn Base und Config sich oft gemeinsam ändern
- ggf. Merge-Konflikte bei Hotfixes

Base und Config getrennt

`master`, `deploy/testing`, `deploy/prod`:

```
atix-some-app-base.git
└── base/...
```

`master`, `deploy`:

```
atix-some-app-conf.git
├── config-test-a/...
├── config-test-b/...
└── config-prod/...
```

- schwieriger umzusetzen
- flexibler in der Struktur des Base-Repos
- gut wenn Config über mehrere Releases identisch bleibt
- ggf. Probleme wenn Base und Config gleichzeitig geändert werden müssen

Mittelweg

- "konstanter" Teil der Config im GitOps Operator konfiguriert
- restliche Config gemeinsam mit Base versioniert

Zusammenfassung

~~Git(Ops) löst alle Probleme in der IT.~~

- viele Fragen bei GitOps
- noch mehr mögliche Antworten
- idealer Workflow hängt vom Anwendungsfall ab

Danke fürs Zuhören!

Fragen?