

mimacom

Bis zu Kubernetes und noch viel weiter

GitOps außerhalb der Komfortzone

Java Forum Stuttgart 2024

mimacom.com



Who am I?



Baris Cubukcuoglu
Senior Software Engineer

Loves challenging projects and heavy weights

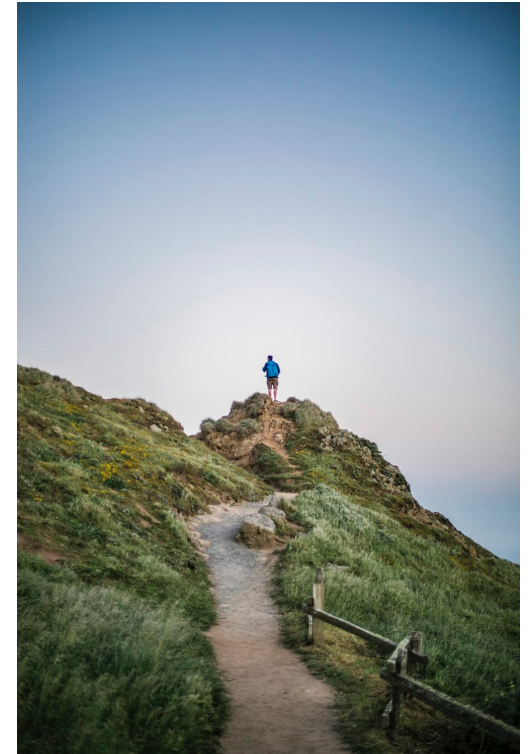
baris.cubukcuoglu@mimacom.com

GitOps outside of Kubernetes

So much potential: There are still way too many situations where GitOps is not yet used.

- Legacy/Custom and Niche Environments
- Security and Compliance Concerns
- Fear of Automation and Loss of Control
- Cultural Resistance

Let's review some real-life GitOps examples that do not use Flux or Argo CD!



Contents

1. The Freedom in The 4 Principles
2. Story 1: Ansible Agent for Docker Swarm
3. Story 2: Helmfile GitOps Agent
4. Lessons Learned

1) The 4 Principles - Declarative

- **Declarative Descriptions:** The entire desired state of the system is described using declarative configuration files.
- **Infrastructure as Code (IaC):** Utilizes IaC to define infrastructure and application configurations in a version-controlled repository.
- **Clarity and Consistency:** Ensures that the system's state is clear and consistent, reducing the potential for configuration drift.
- **Easier Audits and Compliance:** Provides a single source of truth, making audits and compliance easier to manage.



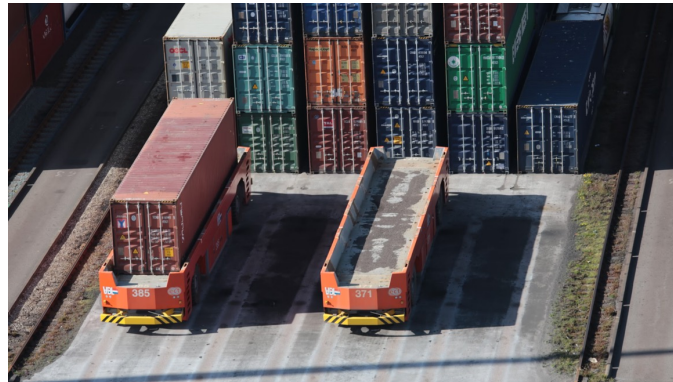
1) The 4 Principles - Versioned and Immutable

- **Version Control:** All changes to the system are versioned in a Git repository.
- **Immutable History:** The Git history is immutable, providing a clear audit trail of all changes.
- **Rollback Capabilities:** Allows for easy rollback to previous states if an issue occurs.
- **Collaborative:** Enables collaboration among team members with clear visibility of changes and their history.



1) The 4 Principles - Pulled Automatically

- **Automated Deployment:** The system automatically pulls changes from the Git repository and applies them to the environment.
- **Reduced Human Error:** Minimizes manual interventions, reducing the likelihood of human error.
- **Faster Deployments:** Accelerates deployment cycles by automating the application of changes.



1) The 4 Principles - Continuously Reconciled

- **Continuous Reconciliation:** The system continuously monitors and reconciles the actual state with the desired state.
- **Self-Healing:** Automatically corrects any deviations from the desired state, ensuring system stability and reliability.
- **Real-Time Monitoring:** Provides real-time monitoring and alerts for any configuration drifts or issues.
- **Enhanced Reliability:** Ensures the system remains in the desired state, enhancing overall reliability and uptime.



1) Myths around The 4 Principles

1. **Declarative:** A system managed by GitOps must have its desired state expressed declaratively.
2. **Versioned and Immutable:** Desired state is stored in a way that enforces immutability, versioning and retains a complete version history.
3. **Pulled Automatically:** Software agents automatically pull the desired state declarations from the source.
4. **Continuously Reconciled:** Software agents continuously observe actual system state and attempt to apply the desired state.

✗ Kubernetes

✗ Containers

✗ Operator running in the target system

1) The P1 Market: Declarative IaC Formats

	Languages	Manage VMs	Orchestrate containers	Manage cloud infra / SaaS	GitOps operator for non-k8s
CFEngine	Custom DSL	✓			CFEngine
Chef Infra Cookbooks	Ruby	✓		(limited)	Chef Infra
OtterScript	PowerShell	✓ (Windows)			Otter
PowerShell DSC	Custom DSL	✓ (Windows)			Otter
Puppet	Custom DSL	✓	✓	(limited)	Puppet Enterprise
Docker Compose	YAML		✓		Portainer (BE)
Kubernetes	YAML	(via extensions)	✓	(via extensions)	
Pulumi	..., YAML	(limited)	✓	✓	
Terraform HCL	HCL	(limited)	✓	✓	

1) Most Wanted Features of a GitOps Operator

1. P3 „Pulled Automatically“
2. P4 „Continuously Reconciled“
3. 💣 Pruning (*delete by commit*)
4. 🔑 Secrets Management
5. 🚨 Reconciliation Alerting

1) The P3/4 Market: GitOps operators for not (only) K8s

	P3	P4	💣 Pruning	🔑 Secrets Mgmt via	🔔 Reconciliation Alerting
CFEngine	✓	✓		cf-secret	e-mail, scripts
Chef Infra	✓	✓		Data Bags	ServiceNow, Slack, Webhooks
Otter	✓	✓			
PipeCD	✓	✓	Only for k8s resources	PipeCD UI	Datadog, Prometheus
Portainer (BE)	✓	✓	Only for Docker Services		
Puppet Enterprise	✓	✓		hiera-eyaml (& some secret stores)	

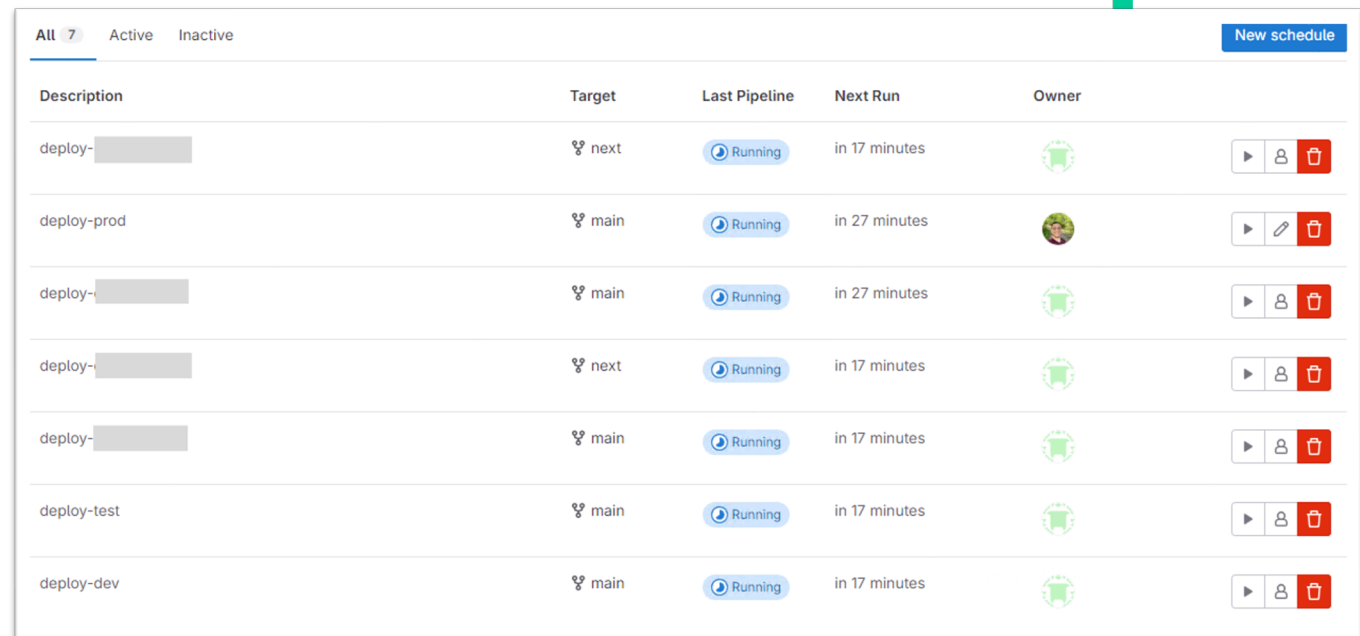
Requires k8s
Control
Plane

2) Ansible Agent for Docker Swarm: Project Context

1. Lots of change: first time containers, first time CI/CD
2. Docker Swarm for orchestration
3. Platform Engineering with on-prem infra
4. Multiple tenants
5. Secrets Management with sops/age

2) Ansible Agent for Docker Swarm: GitOps Setup

= an Ansible Playbook running on a Scheduled Pipeline in GitLab CI

A screenshot of the GitLab CI 'Scheduled Pipelines' page. At the top, there are tabs for 'All' (selected), 'Active', and 'Inactive', along with a 'New schedule' button. The main content is a table with columns: 'Description', 'Target', 'Last Pipeline', 'Next Run', and 'Owner'. There are seven rows of scheduled pipelines, all with a status of 'Running'. The 'Description' column contains names like 'deploy-[redacted]', 'deploy-prod', 'deploy-[redacted]', 'deploy-[redacted]', 'deploy-[redacted]', 'deploy-test', and 'deploy-dev'. The 'Target' column shows branches like 'next' and 'main'. The 'Next Run' column indicates times like 'in 17 minutes' and 'in 27 minutes'. The 'Owner' column shows various user avatars. To the right of each row are three icons: a play button, a user icon, and a trash can icon.

Description	Target	Last Pipeline	Next Run	Owner
deploy-[redacted]	next	Running	in 17 minutes	[Avatar]
deploy-prod	main	Running	in 27 minutes	[Avatar]
deploy-[redacted]	main	Running	in 27 minutes	[Avatar]
deploy-[redacted]	next	Running	in 17 minutes	[Avatar]
deploy-[redacted]	main	Running	in 17 minutes	[Avatar]
deploy-test	main	Running	in 17 minutes	[Avatar]
deploy-dev	main	Running	in 17 minutes	[Avatar]

Isn't this CIOps? 🤖

Not at all 😊

- Playbook only clones config repos & runs rollout
- Playbook is not triggered by single commits

2) Why Ansible?

1. Team experience
2. Existing SSH access to VMs
3. Idempotent tasks
4. Huge ecosystem (Ansible Galaxy)
5. Existing projects were not complete enough
e.g. <https://gitlab.com/stavros/harbormaster>



2) Building the playbook: Shopping in Ansible Galaxy

1. P3 „Pulled Automatically“:
→ `ansible.builtin.git`
2. P4 „Continuously Reconciled“:
→ `community.docker.docker_stack`
3. Pruning:
 - For services: `docker stack up --prune`
 - For stacks: manual pipeline
4. Secrets Management:
→ `community.sops.sops filter`
5. Alerting:
→ CI alerts from failing pipelines

2) Building the playbook: Actual Playbook

```
- name: Clone all config repos
  ansible.builtin.git: # ...

- name: Find all manifests
  ansible.builtin.find: # ...

- name: Render manifests with secrets, save to register
  ansible.builtin.shell: # ...

- name: Create directories for manifests on host
  ansible.builtin.file: # ...

- name: Create manifest files on host from register
  ansible.builtin.copy: # ...

- name: Deploy Stacks from created manifests
  community.docker.docker_stack: # ...
```

- P3 „Pulled Automatically“

- P4: Rendering like Argo CD (incl. the sops filter)

- P4: Rollout (incl. pruning)

2) Ansible Agent for Docker Swarm: Results

1. Stability
2. Development & Deployment Speed
3. Better Scalability for deploying new stuff
4. Headaches from asynchronous behavior

Find an excerpt of the playbook at
[https://gitlab.com/gitops-book/
docker-swarm-gitops-ansible](https://gitlab.com/gitops-book/docker-swarm-gitops-ansible)

3) Helmfile GitOps Agent: Project Context

1. Shared Kubernetes cluster
2. Deploying Helm charts
3. No Flux or Argo CD installed
4. Tenant role (= no operator installation)



→ We found **Helmfile** as a declarative format for managing Helm releases.

```
helmDefaults:
  createNamespace: false
repositories:
  - name: metrics-server
    url: https://kubernetes-
      sigs.github.io/metrics-server
releases:
  - name: metrics-server
    chart: metrics-server/metrics-server
    namespace: kube-system
    version: 3.11.0
    values:
      - args:
        - --kubelet-insecure-tls
```

3) Helmfile GitOps Agent: GitOps Setup

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: helmfile-gitops-agent
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          initContainers:
            - name: git-clone
              image: alpine/git
              command:
                - /bin/sh
                - -c
                - git clone GIT_REPO_URL /cloned-repo
          containers:
            - name: helmfile
              image: ghcr.io/helmfile/helmfile
              command:
                - /bin/sh
                - -c
                - helmfile apply
```

= a Kubernetes CronJob

3) Helmfile GitOps Agent: Implementing the Features

1. P3 „Pulled Automatically“
→ Init Container
2. P4 „Continuously Reconciled“
→ Helmfile Container
3. Pruning:
 - For Helm releases: 2-step reconciliation
 - For full Helmfile: manual destroy
4. Secrets Management:
→ possible e.g. via sops (Helm plugin)
5. Alerting:
→ Prometheus alerts on failed k8s Jobs

3) Helmfile GitOps Agent: Results

🎉 We can do GitOps on Kubernetes — even without admin superpowers!

It's also useful for fast iterations with less Flux or Argo CD knowledge

Find the Helm chart at
<https://gitlab.com/gitops-book/helmfile-gitops-agent>

4) Lessons Learned

1. Whenever you have a declarative IaC format, you can do GitOps!
2. Writing and maintaining your own GitOps operator does not come for free, but it's almost always worth it.
3. Kubernetes is (by design) a perfect breeding ground for GitOps agents, even without using actual Operators and CRDs.



Thank you!

You might also enjoy our 300+ pages GitOps book which is available since May 2024 at your lovely bookstore ...

<https://dpunkt.de/produkt/gitops>

<https://gitops-book.dev/>

