PowMio LAB
Lennart Ruck, 20.11.2025

Migrating microservice oriented application to Kubernetes

With focus on best industry practises in CI/CD

PowMio

Software is much more than code.

Agenda

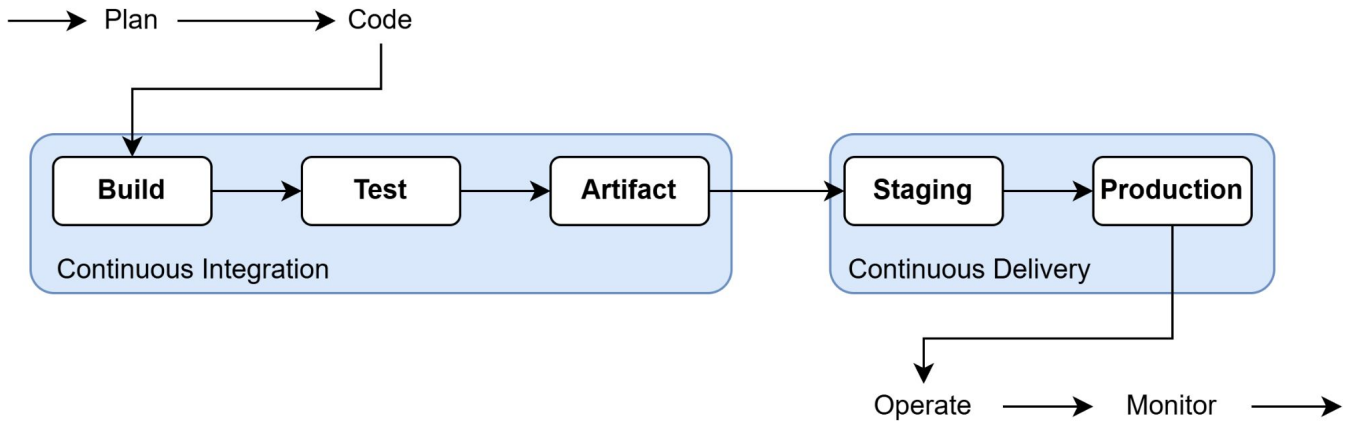
- Introduction
  - example project
  - current CI/CD process
- Improving current CI/CD
- Migrating app to new CI/CD environment
- Demo with improved CI/CD

NGINX

Improve software development

- “CI/CD [is] a set of practices and tools designed to improve the software development process.” [1]

software development lifecycle:

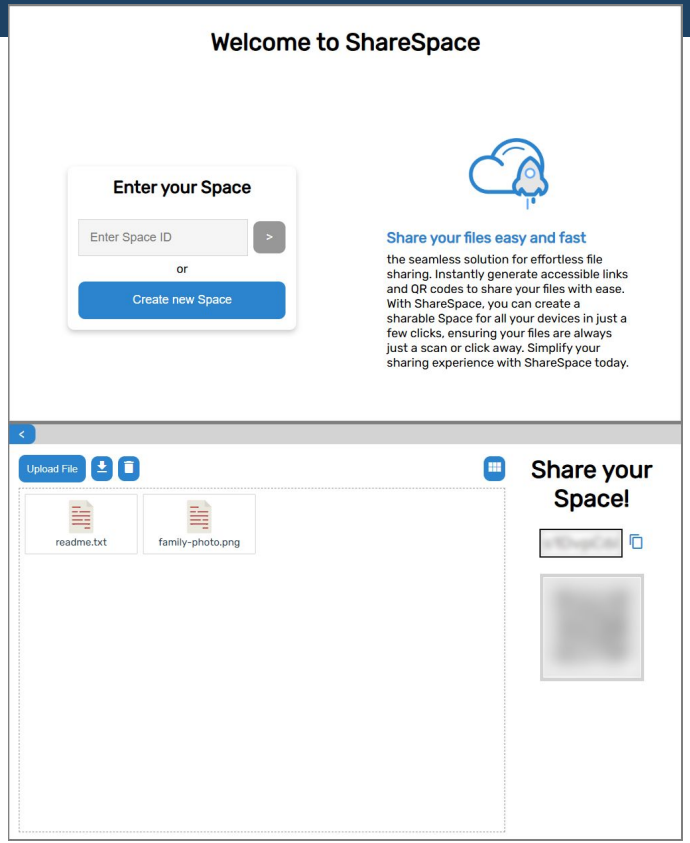


[1] https://github.com/resources/articles/ci-cd, accessed Nov 2025
Diagram inspired by GitHub's CI/CD documentation [1]

Example Project: Fileshare

"Cloud-based application for easy file sharing"

- create and share spaces
- upload and manage files
- download files on any device
- easily share your space via QR code



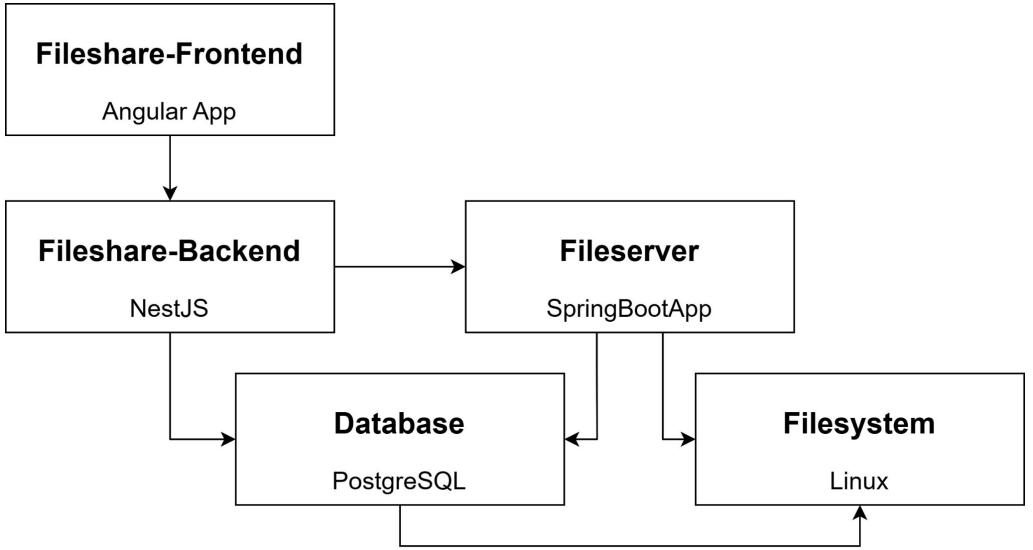
Fileshare app architecture

3 microservices

Angular frontend
- end-user application

NestJS backend
- manages spaces and what files are in which spaces

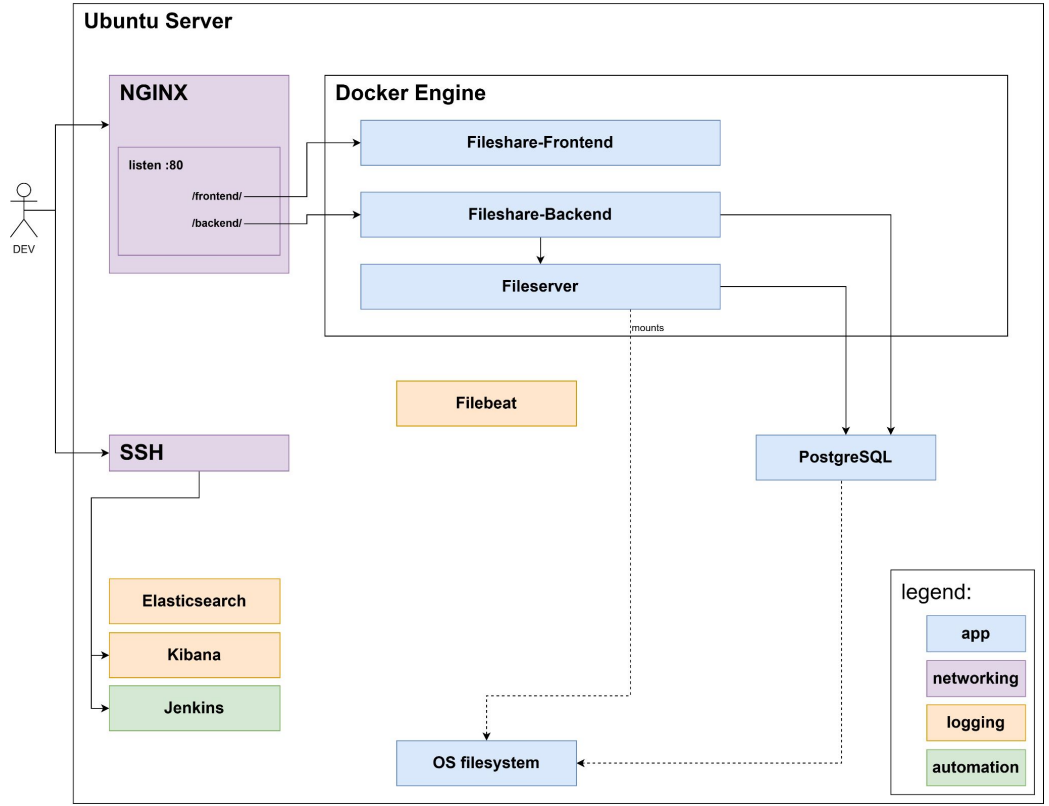
Spring Boot fileserver
- serves as repository for files



Complete Fileshare architecture

Architecture →

- microservices are containerized
- 1 git repository and 1 Jenkins pipeline for every microservice
- Jenkins deploys microservices in docker
- NGINX, PostgreSQL, Elastic Stack run natively



Fileshare CI/CD process

1. new app feature is planned
2. write code
3. test changes locally
4. check into git and create pull request
5. merging pull request starts CI/CD pipeline
    - build docker image for microservice
    - stop running docker container
    - create and run new docker container
    - run docker networks for server-local communication
6. manually use and test deployed app

Challenges in CI/CD

Addressing pain points and introducing industry solutions

Rollout strategy problem

Pain point scenario:

- “We release a new feature. In Prod we notice it’s breaking our service. We need to roll back.”

Root cause:

- faulty test / misconfiguration

The problem:

- downtimes in production
- not safe rollout strategy
- no rollback functionality

Rollout strategy solution

Solution:

- reliable rollout process
- automated rollout with easy rollback

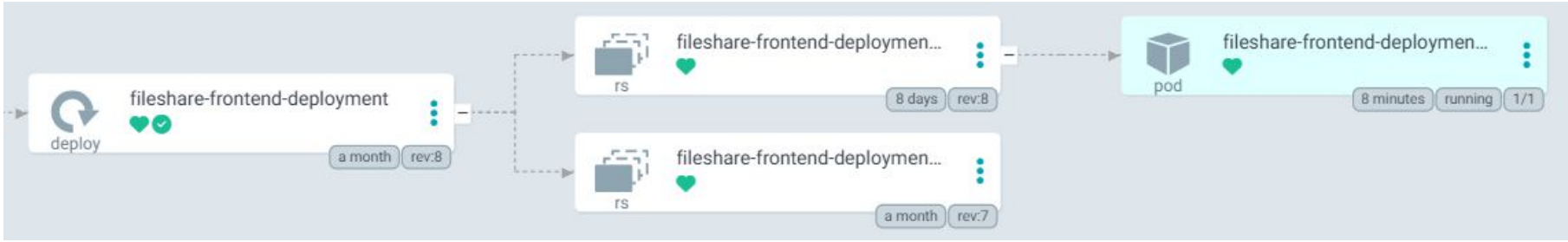
Technology:

- kubernetes
- “Kubernetes is a [...] extensible [...] platform for managing containerized workloads and services” [2]
- use kubernetes deployments for app microservices
- let kubernetes deploy them

[2] - https://kubernetes.io/docs/concepts/overview/, accessed Oct 2025

Rollout strategy solution

- kubernetes resource type deployment
- safe rollout strategy
  - only deletes old pods when new ones are running and healthy
- deployment keeps old replica sets for easy rollbacks



Spinning Up Environments problem

Pain point scenario:

- “For additional testing you need a new environment. A new deployment costs you one day of work.”

Root cause:

- deployment process is too complex
- difficult to see desired state and live state

The problem:

- the deployment process involves many manual steps
- troubleshooting infrastructure problems is difficult

Spinning Up Environments solution

Solution:

- declarative Configuration & orchestration
- Infrastructure as Code (IaC)

Technology:

- Kubernetes
- define deployment with kubernetes yaml files
- live state is held in the kubernetes yaml files
- “IaC is managing and provisioning [...] infrastructure through code instead of manual processes.” [3]

[3] - https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac#overview, accessed Oct 2025

Configuration drift problem

Pain point scenario:

- "The memory limit of a service is 2gb but in the configuration file it is 512mb. Is it supposed to be 2gb or 512mb? You don't know when or why it changed."

Root cause:

- manual, undocumented changes to deployment

The problem:

- no single source of truth for desired state
- difficult to keep track of changes

Configuration drift solution

Solution: GitOps

- git repository to store configuration files, serves as desired state
- changes with git commits, git history serves as audit log
- automatically update live state to reach git state

Technology:

- Argo CD
- “a declarative, GitOps continuous delivery tool for Kubernetes” [4]
- continuously compares live state to git state
  - marks resources out of sync
  - can automatically create, update, delete resources

[4] - https://argo-cd.readthedocs.io/en/stable/, accessed Oct 2025

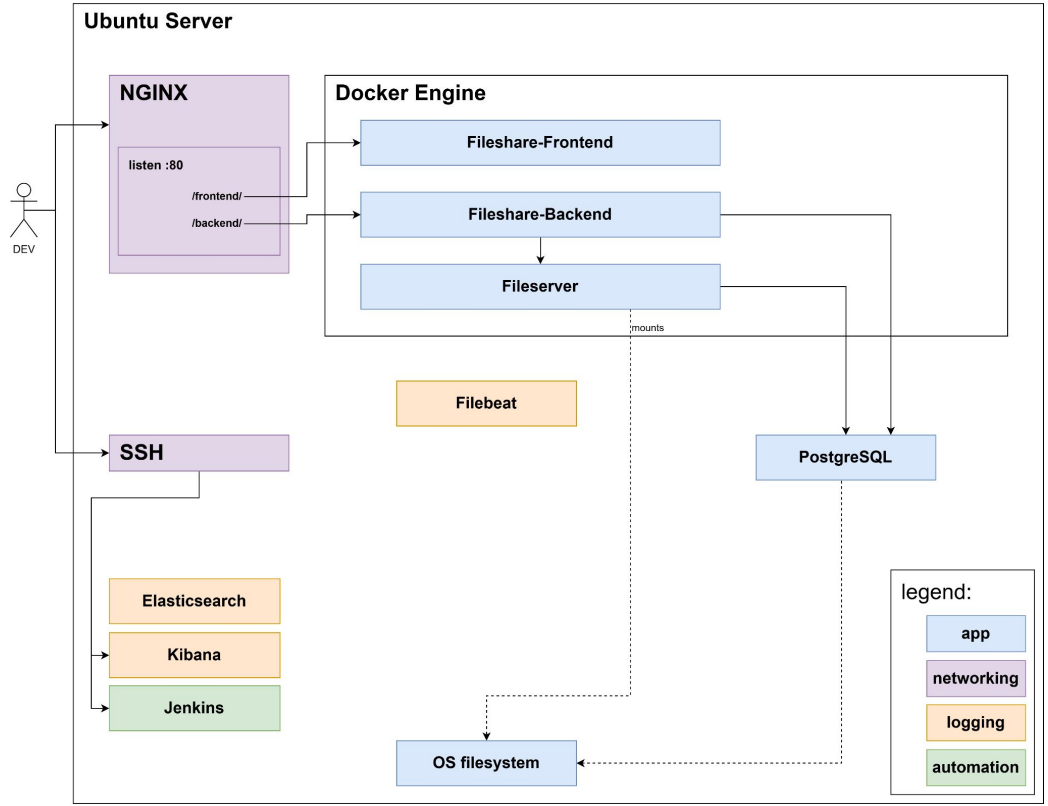
Migrating to an improved CI/CD

Preparing infrastructure and applying CI/CD industry practises

Deployment before

Architecture →

- microservices are containerized
- 1 git repository and 1 Jenkins pipeline for every microservice
- Jenkins deploys microservices in docker
- NGINX, PostgreSQL, Elastic Stack run natively



Migration plan

1. Deploy microservices in Kubernetes
   - ensures automated and reliable rollout
   - enables easy automated spin up of microservices
2. Implement GitOps with Argo CD
   - single source of truth
   - clear auditable change history
3. Add Secret Management with Sealed Secrets
   - add layer of security on our credentials
4. Adapt ELK to new CI/CD
   - restore logging functionality

Migration plan

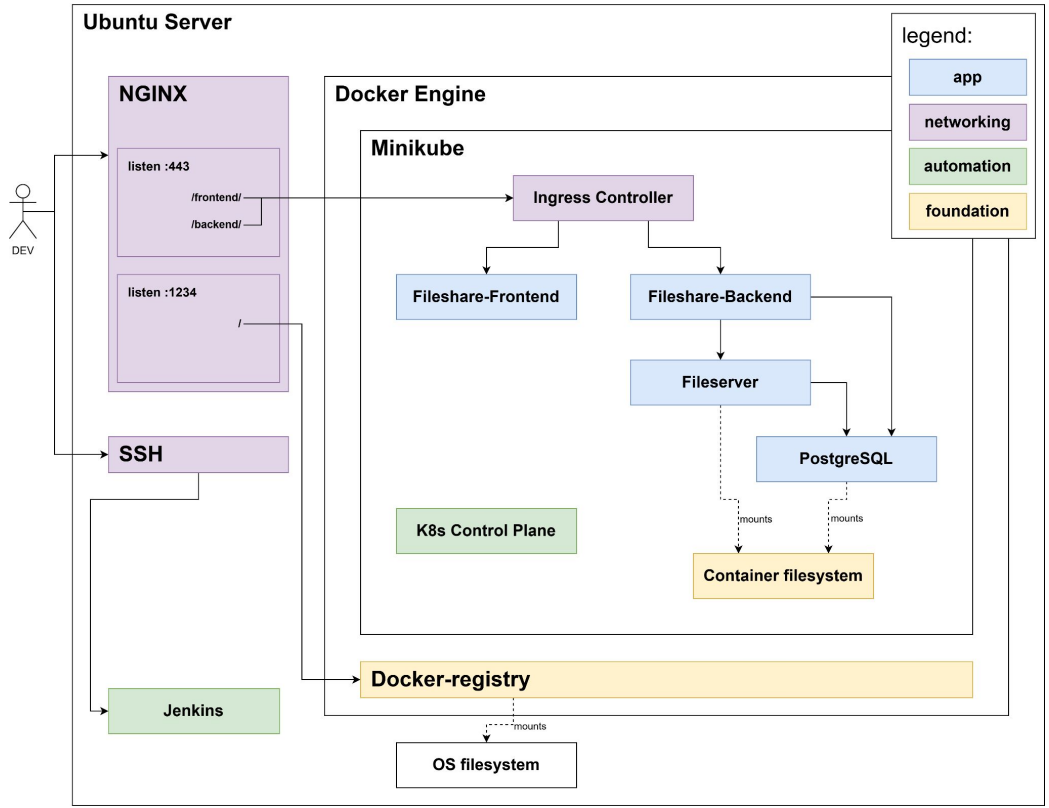
1. Deploy microservices in Kubernetes
   - ensures automated and reliable rollout
   - enables easy automated spin up of microservices
   - set up kubernetes environment
     - provision a kubernetes cluster
     - create kubernetes resources
   - adapt ci/cd process
     - install a private docker registry
     - modify jenkins pipelines

Deployment after step 1

Architecture →

differences:

- microservices deployed in kubernetes cluster
- traffic routed through ingress controller to microservices
- docker-registry providing app images



Migration plan

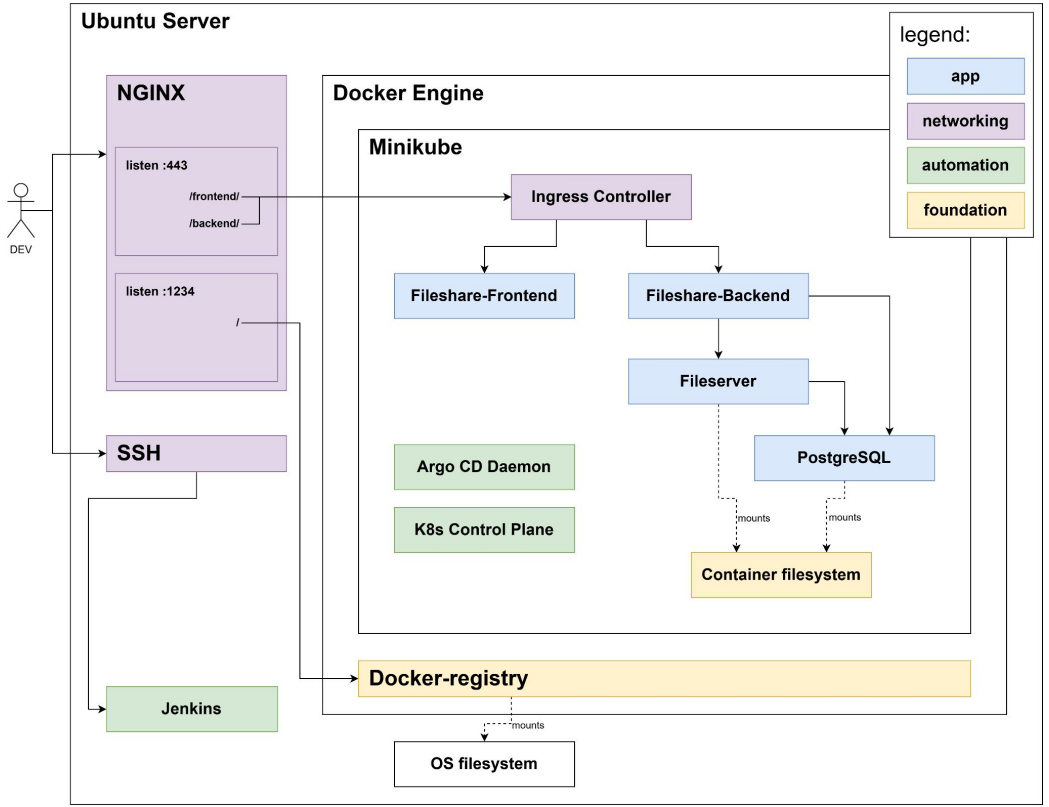
2. Implement GitOps with Argo CD
    - single source of truth
    - clear auditable change history
    - set up Argo CD
        - deploy Argo CD in kubernetes cluster
        - create Argo CD resources
            - hold sync policies for other resources

Deployment after step 2

Architecture →

differences:

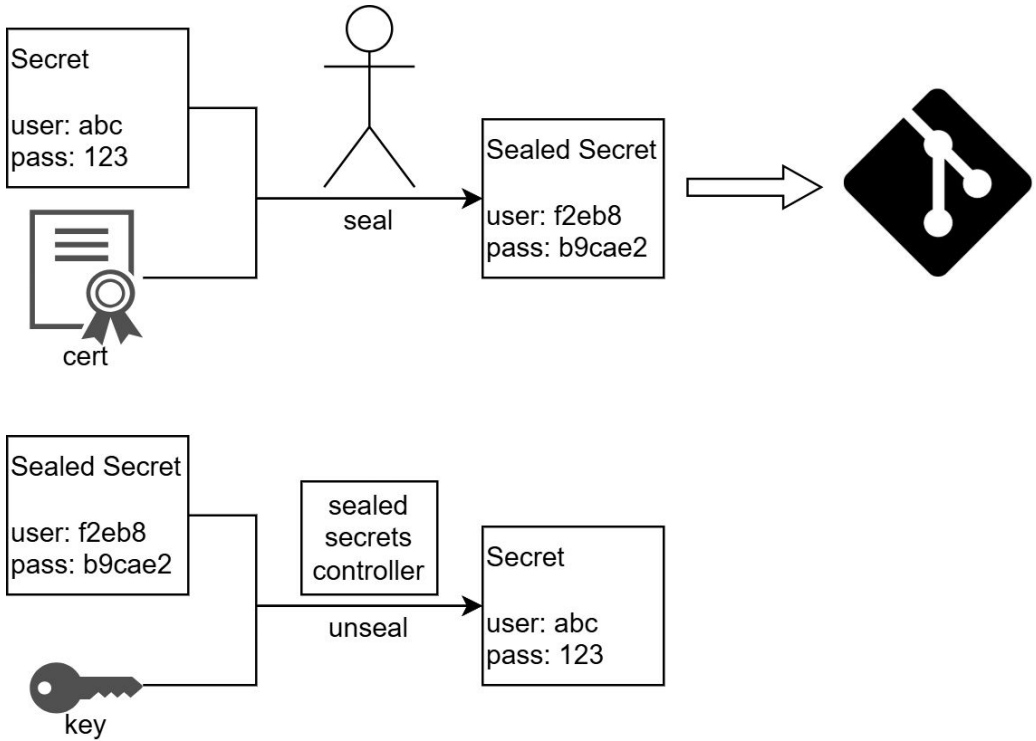
- Argo CD deployed in kubernetes cluster



Migration plan

3. Add Secret Management with Sealed Secrets
    - add layer of security on our credentials
    - core concept:
        - secrets are encrypted at rest
        - sealed secrets controller decrypts during runtime
    - seal secrets and put in git repository
    - deploy sealed secrets controller

Sealed Secrets



core concept:

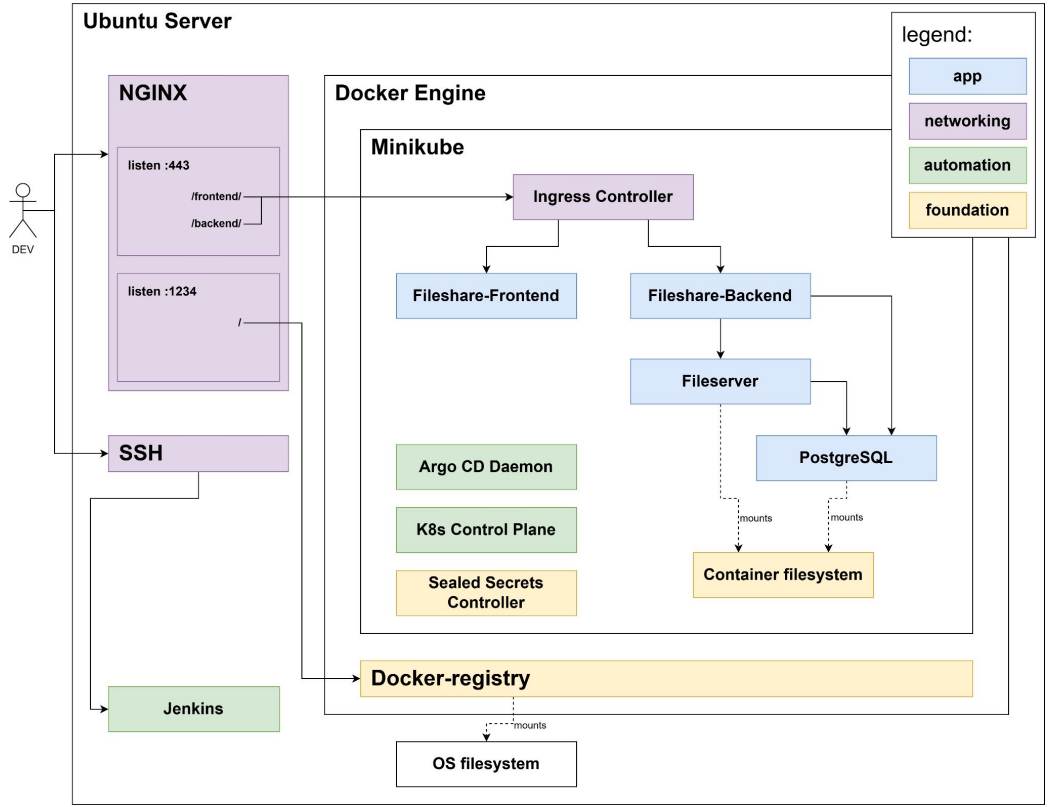
- secrets are encrypted at rest
- sealed secrets controller decrypts during runtime
- key needs to be handled manually

Deployment after step 3

Architecture →

differences:

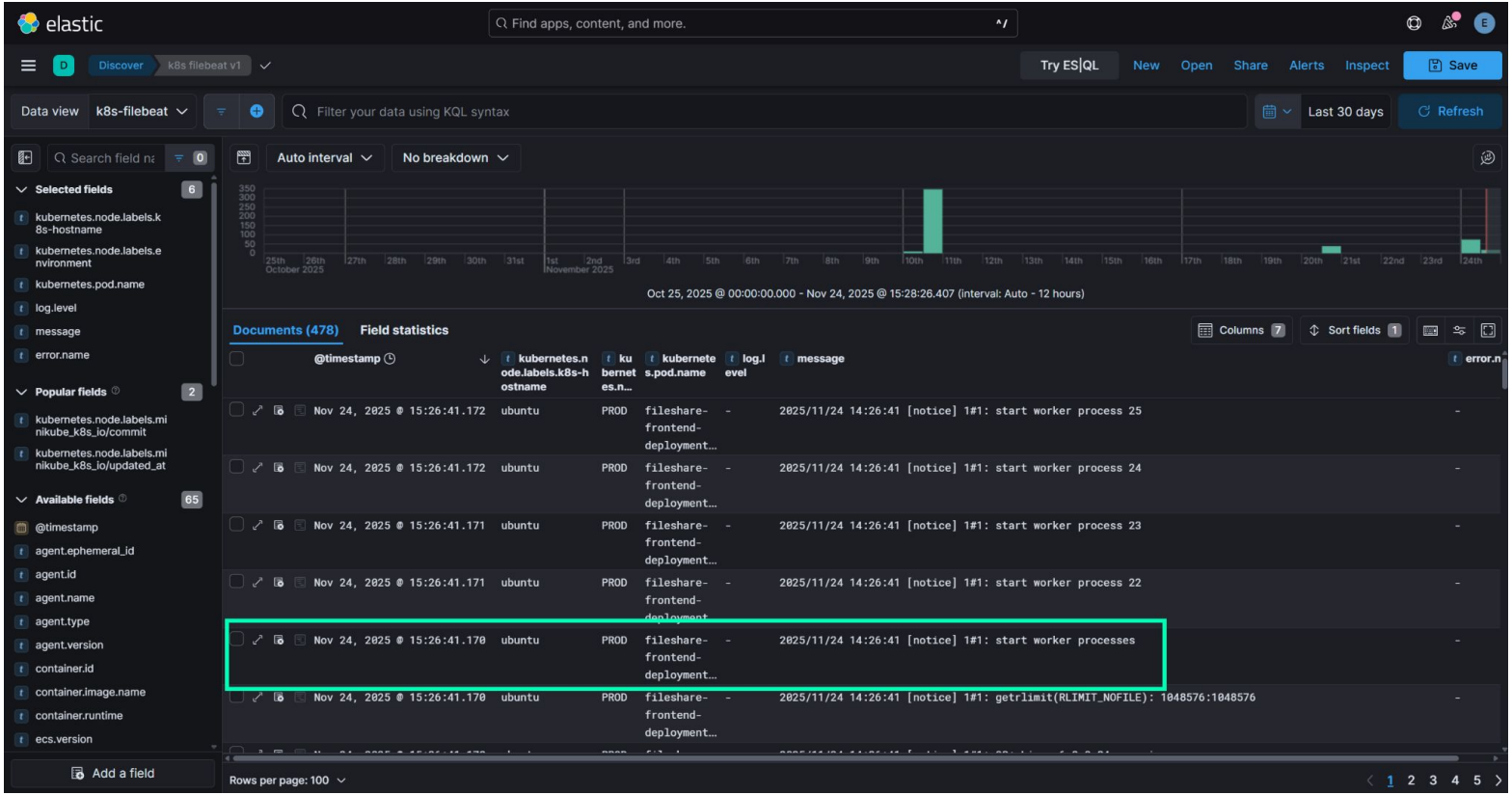
- Sealed Secrets Controller deployed in kubernetes cluster



Migration plan

4. Adapt ELK to new CI/CD
    - restore logging functionality
    - add kubernetes metadata
        - deploy Filebeat inside kubernetes
        - adapt configuration

Logging with ELK

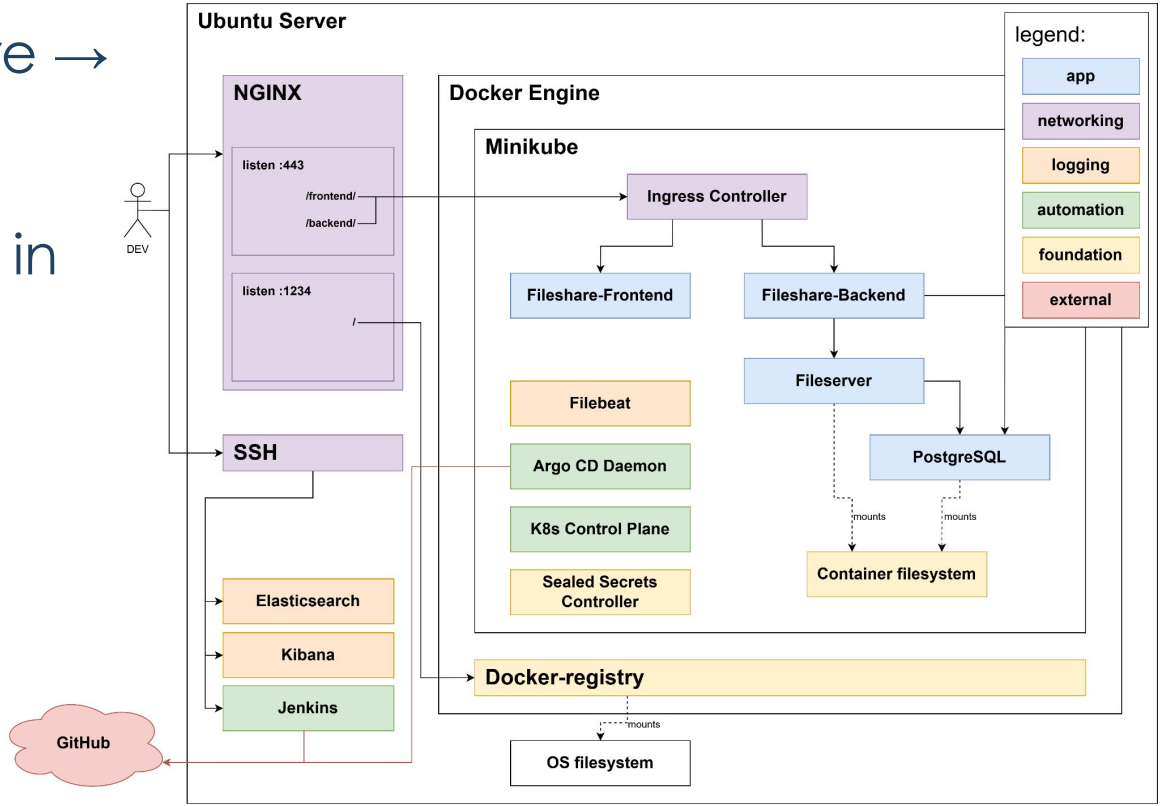


Deployment after

Architecture →

Main differences:

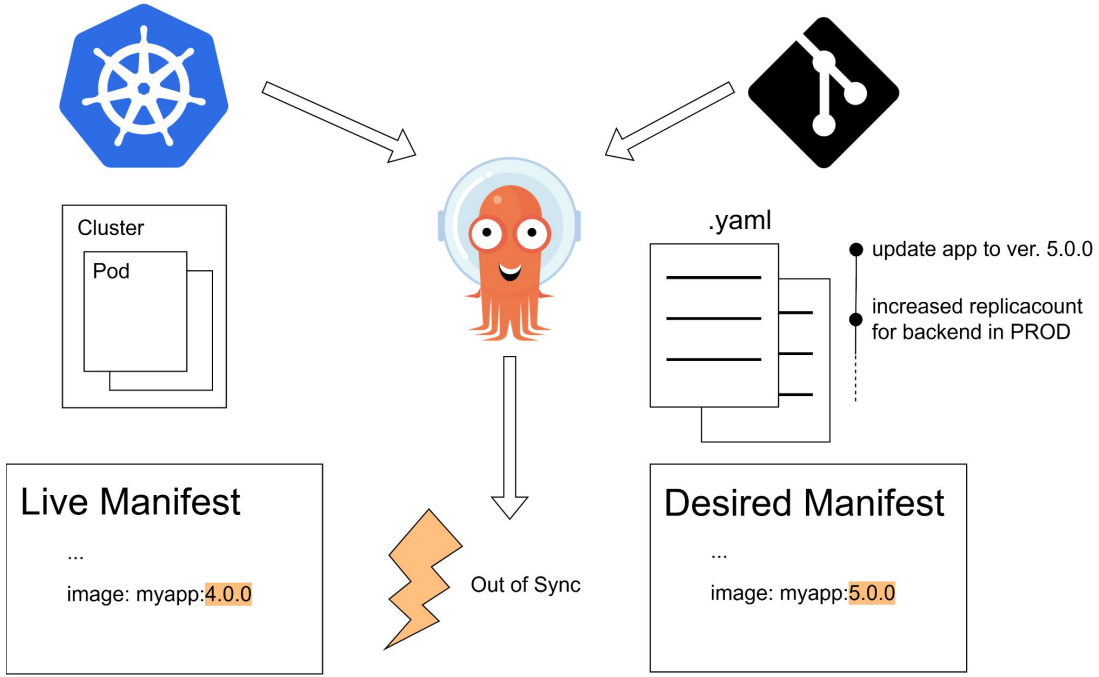
- microservices run in Kubernetes
- Argo CD deploys microservices
- PostgreSQL and Filebeat run in kubernetes



Argo CD

Sequence of events:

- cluster runs app image version 4.0.0
- commit with image tag update to version 5.0.0 pushed
- Argo CD compares git state to live state
- Argo CD marks resource as ‘Out of Sync’
- (configurable) Argo CD updates live manifest to desired state



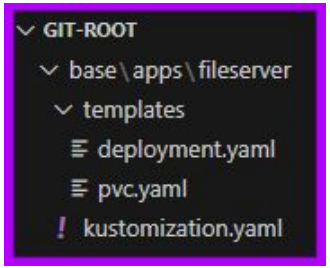
Argo CD

fileserver-application.yaml

```yaml
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: fileserver-argo-application
  namespace: argocd
spec:
  project: default

  source:
    repoURL: [redacted]
    targetRevision: HEAD
    path: base/apps/fileserver

  destination:
    server: https://kubernetes.default.svc
    namespace: fileshare

  syncPolicy:
    syncOptions:
      - CreateNamespace=true
    automated: null

```

Argo is not allowed to sync automatically

directory structure:

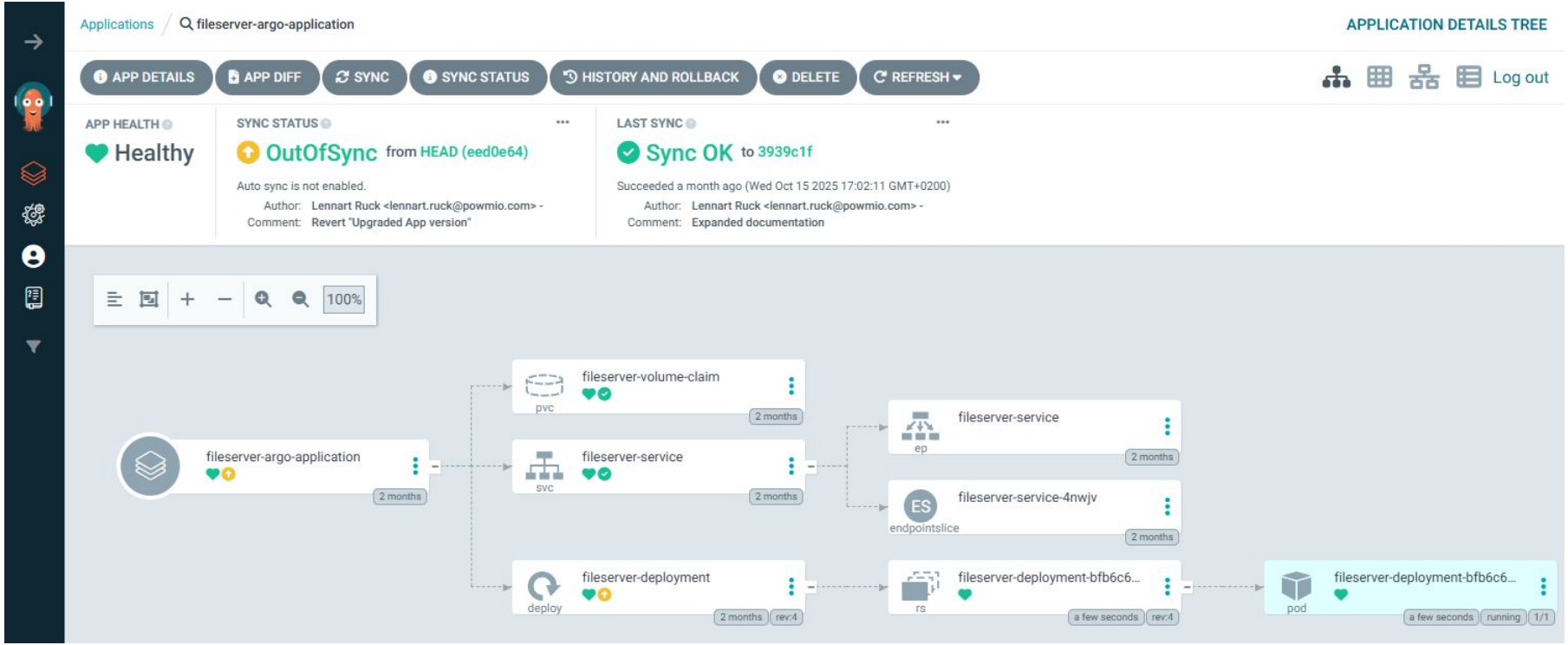


Argo application

- has a source, a collection of files containing kubernetes resources
- has a destination, a cluster and namespace

Argo CD

Dashboard:



Screenshot of Argo CD UI for demonstration purposes.

CI/CD process after

1. a new feature is planned
2. developer writes code
3. developer tests changes locally
4. check into source code repo and create pull request
5. merging pull request starts CI/CD pipeline
   - builds docker image for microservice
   - pushes docker image to docker registry
6. update image version in kubernetes manifest in GitOps repo
7. check changes into GitOps repo and create pull request
8. merging pull request makes Argo CD deploy updated version
9. developer verifies correct deployment and uses App

Hands-on with improved CI/CD

PowMio

Steps:

1. Change background color of Angular App
2. Prepare commit, push and merge to master
3. Write the new image tag into gitOps repository
4. Prepare commit, push and merge to master
5. Sync Angular deployment in Argo CD Web UI

Fileshare before

PowMio

Welcome to ShareSpace

Enter your Space

Enter Space ID

>

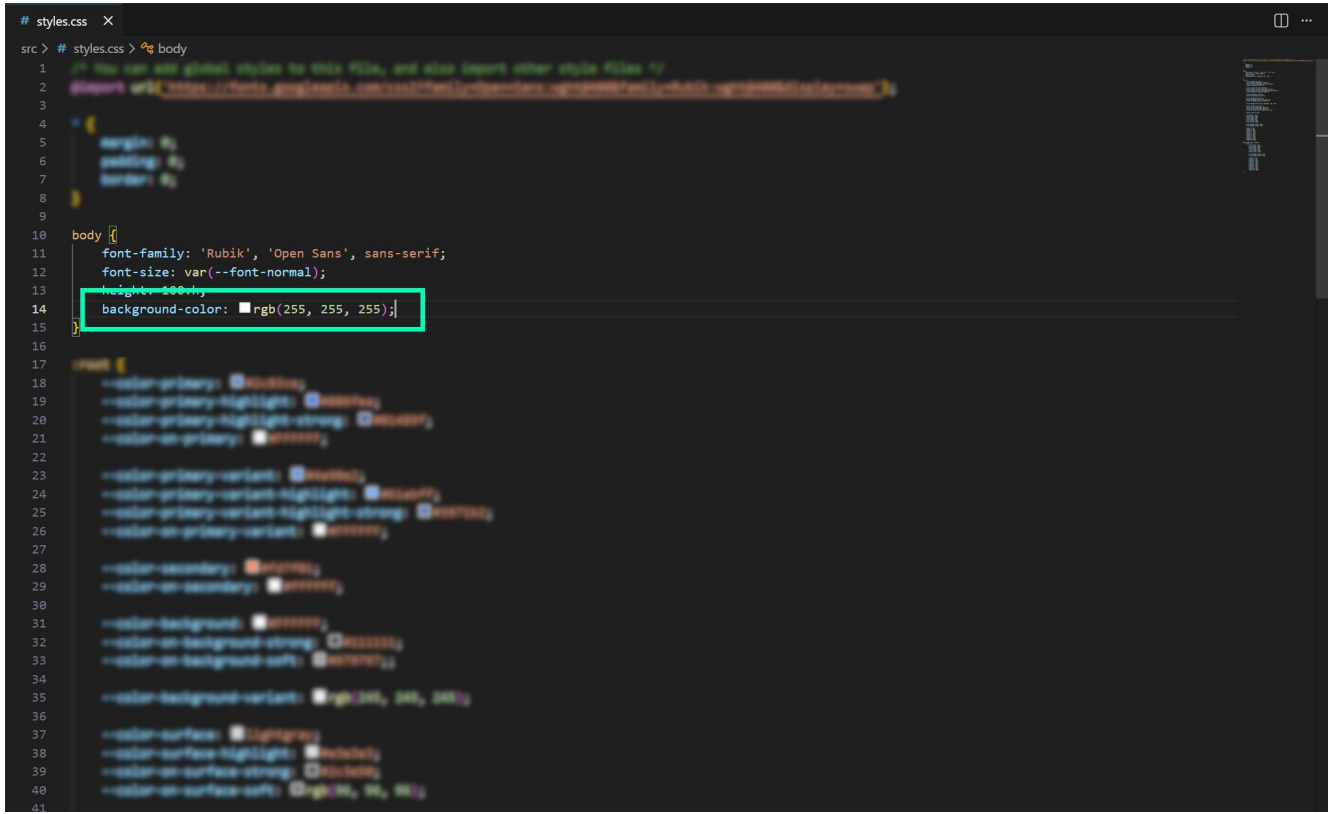
or

Create new Space

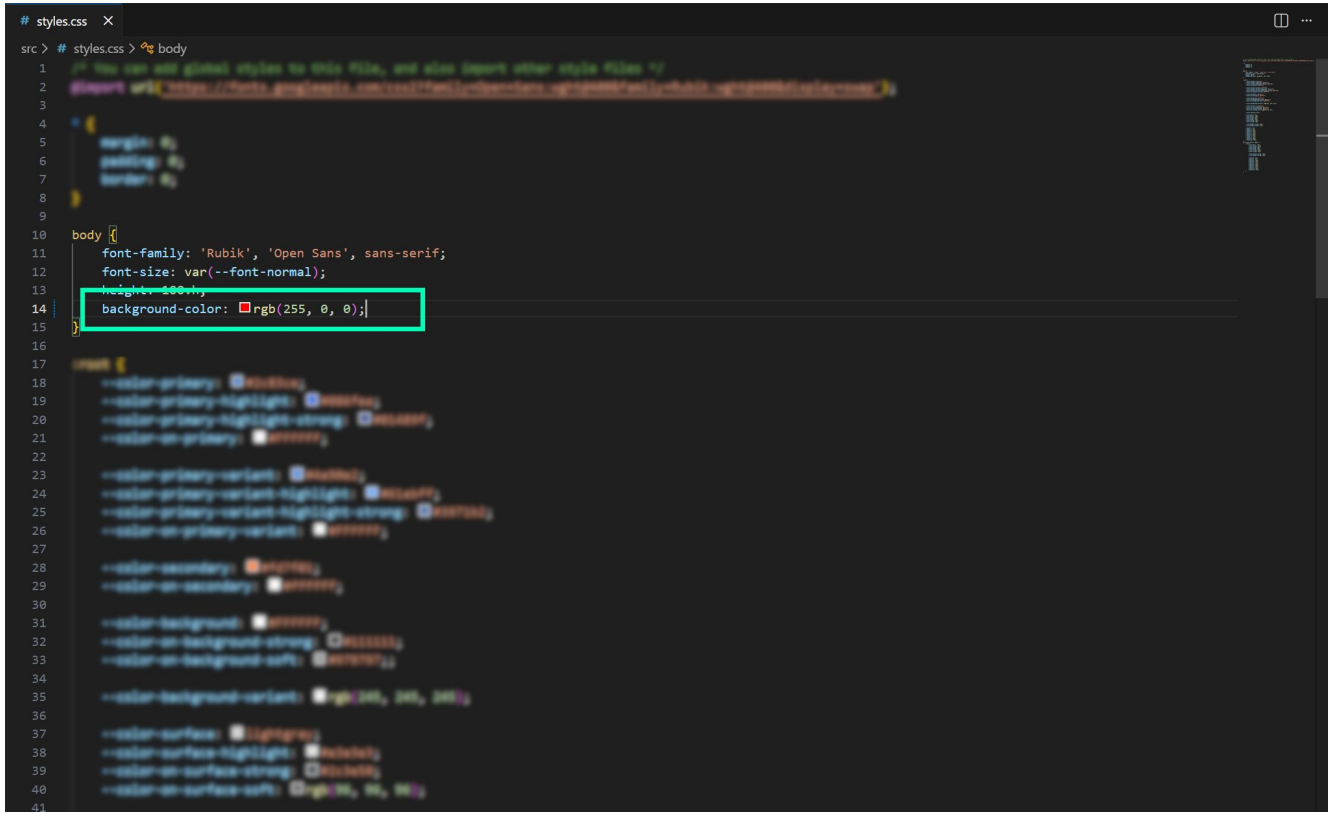
Share your files easy and fast

the seamless solution for effortless file sharing. Instantly generate accessible links and QR codes to share your files with ease. With ShareSpace, you can create a sharable Space for all your devices in just a few clicks, ensuring your files are always just a scan or click away. Simplify your sharing experience with ShareSpace today.

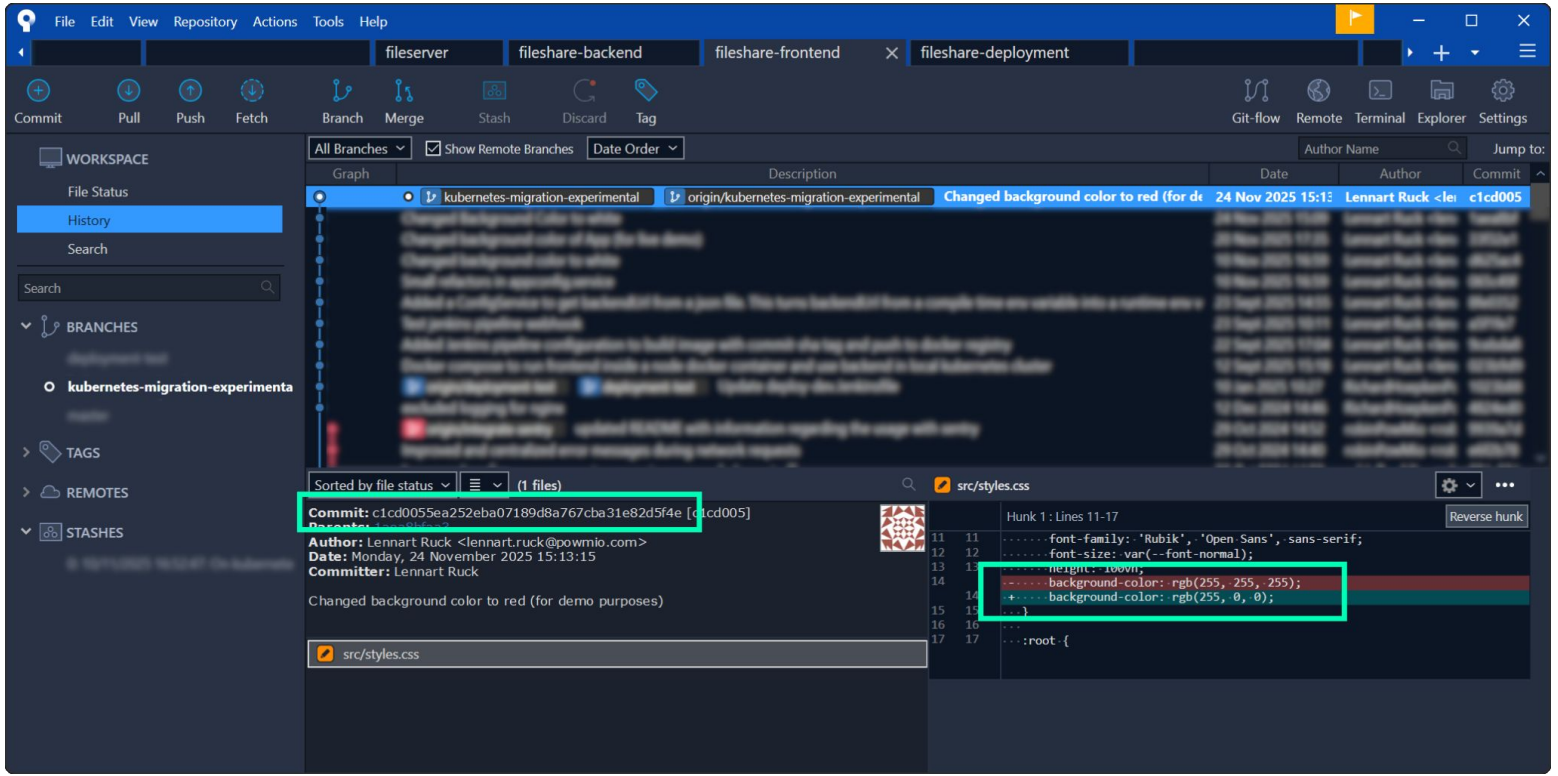
Fileshare Frontend before



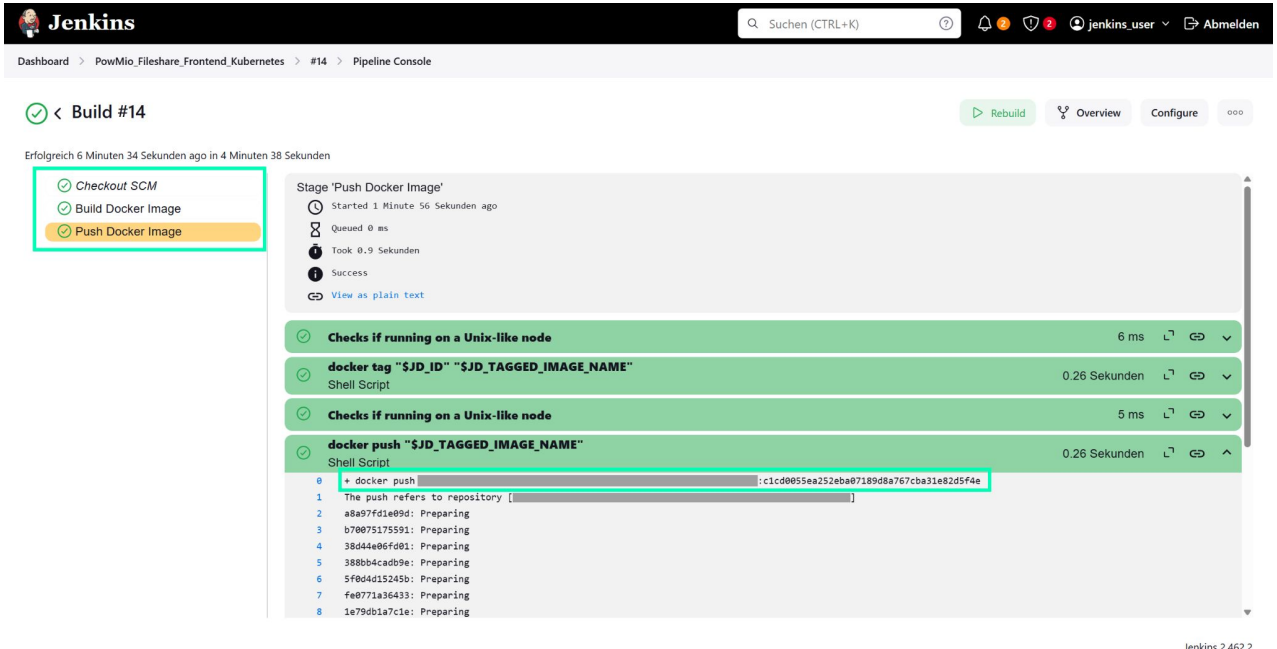
Fileshare Frontend after

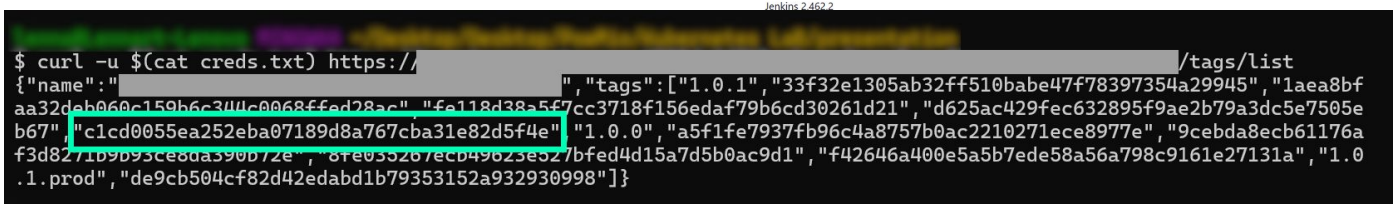


Committed Frontend changes



Jenkins running Frontend pipeline

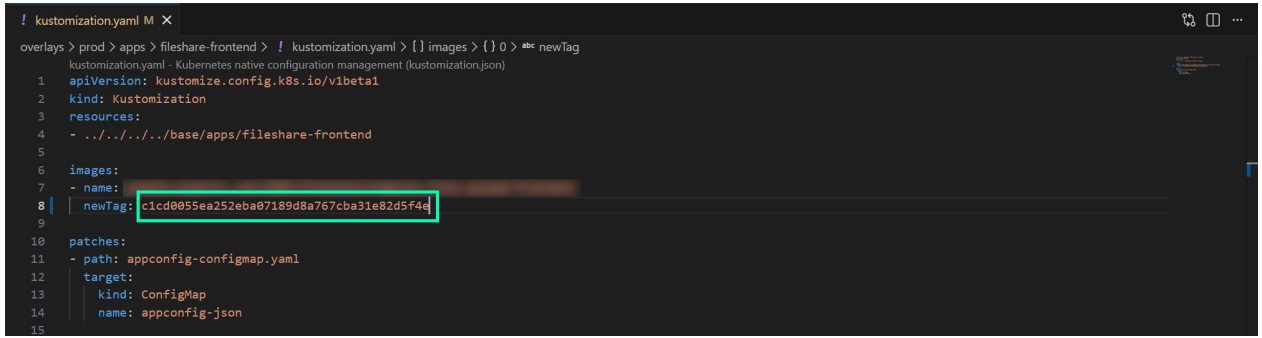




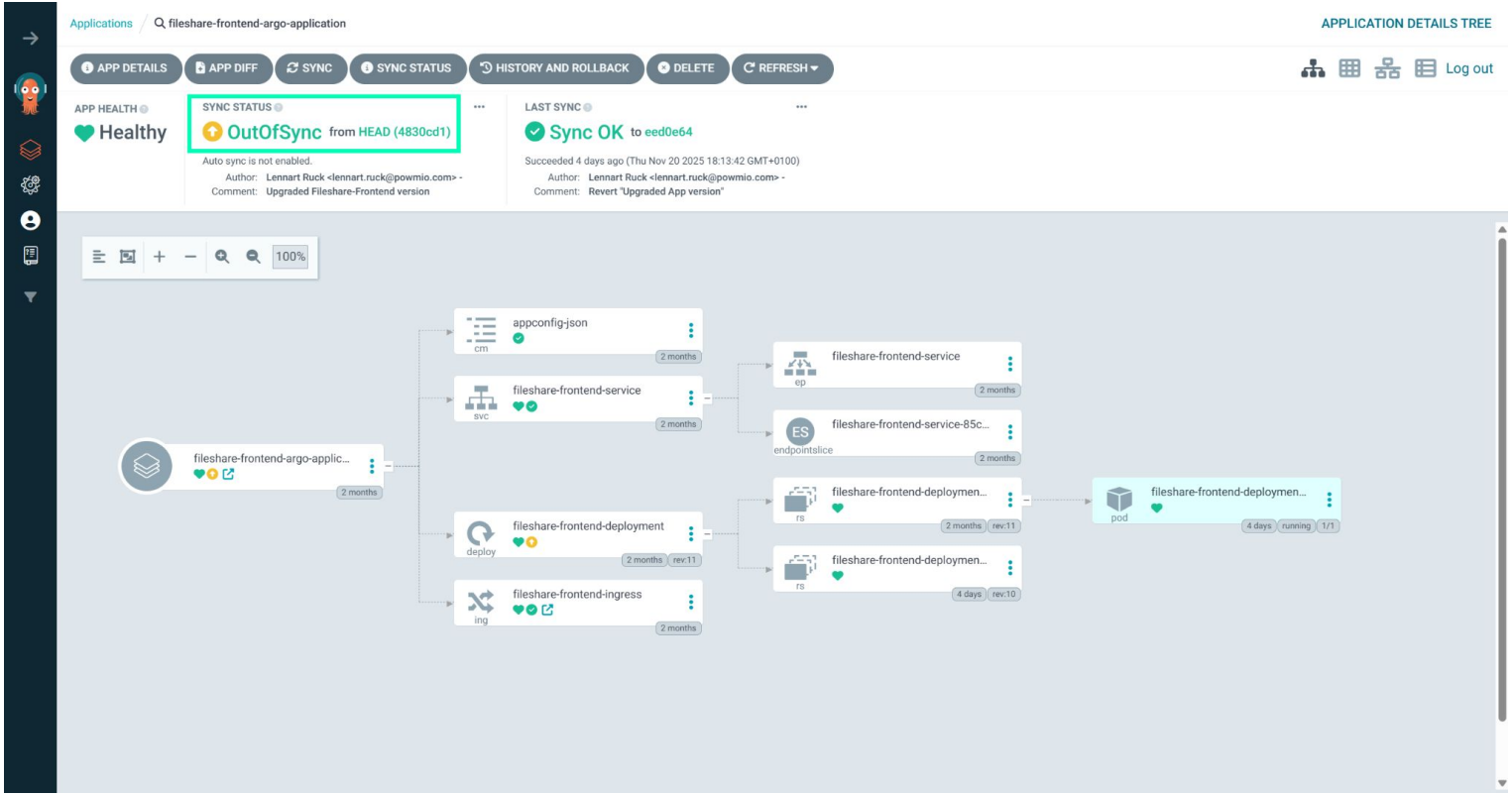
Updating App version in Deployment

Before:

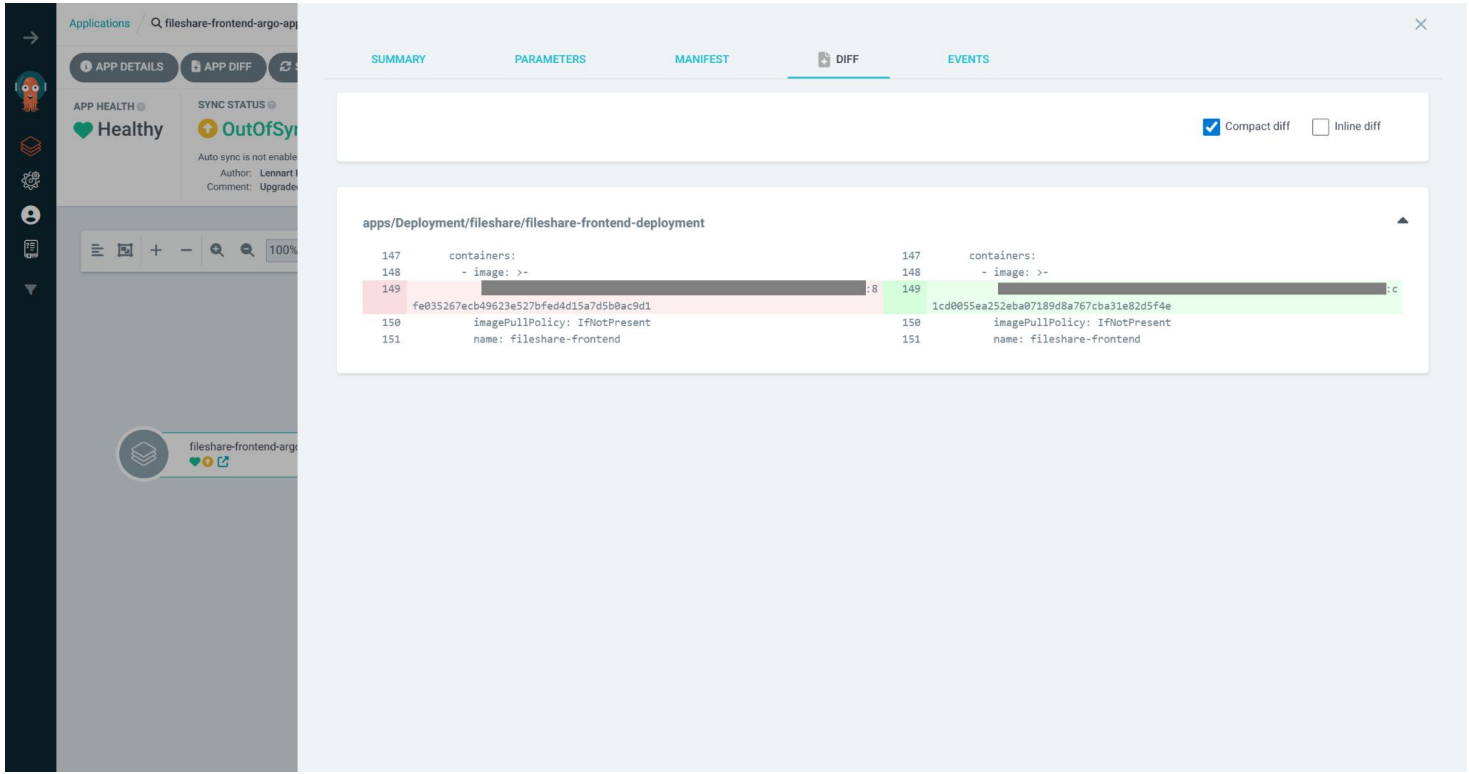
After:



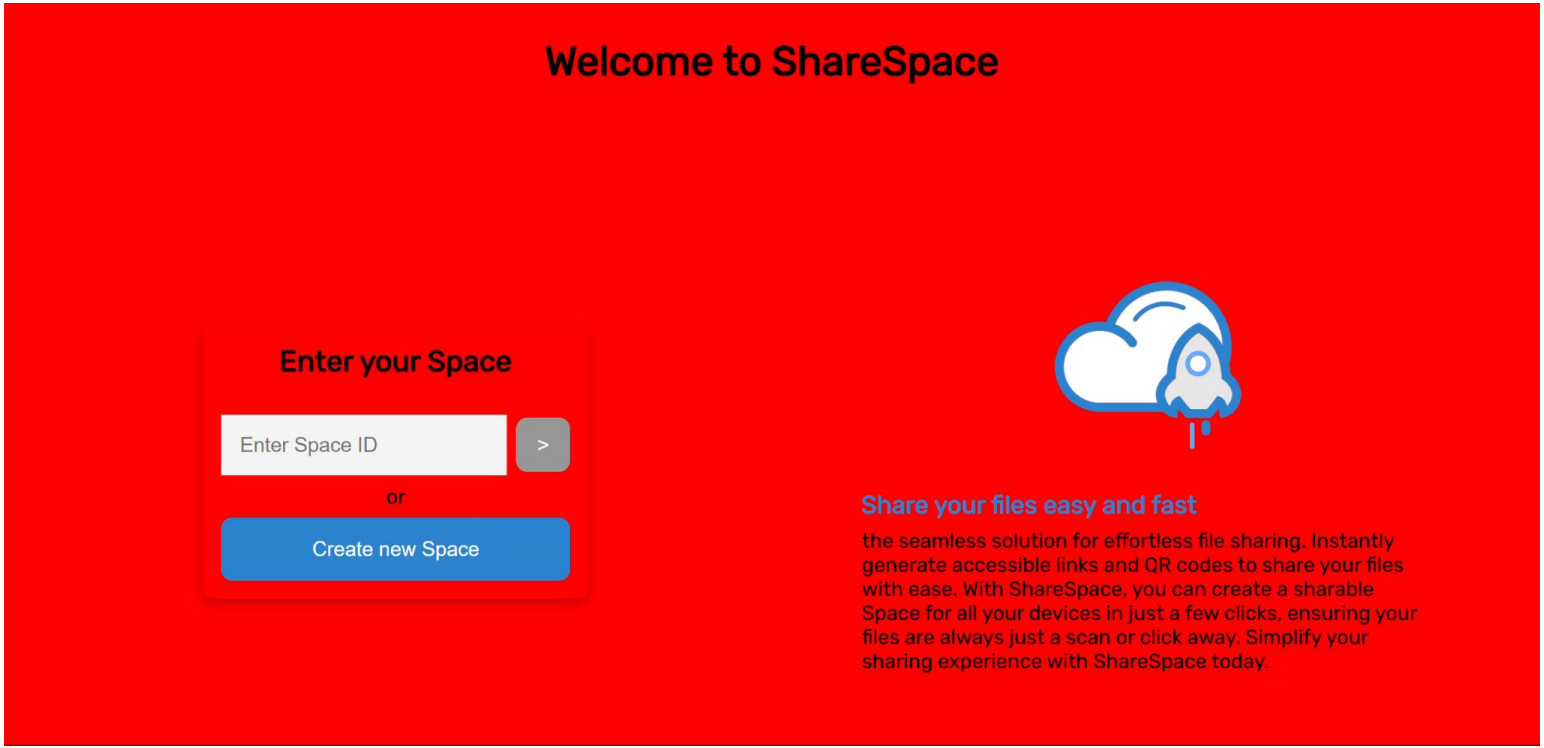
Argo CD shows out of sync resource



Argo CD ‘App Diff’



Fileshare after pressing Sync in Argo CD



Thank you!

PowMio

Alter Schlachthof 33
76131 Karlsruhe

www.powmio.com

contact@powmio.com