Задай въпрос на vevox.app
ID: 192-474-853

Димитър Захариев
Technical Trainer @ SoftUni

GitOps and Kubernetes

Agenda

- GitOps and Kubernetes
  - The Road to GitOps
  - What is GitOps
  - Benefits and Challenges
  - GitOps Toolkit
  - Best Practices
- Demo
- Q&A Session

GitOps and Kubernetes

A Modern Approach to Infrastructure and Application Management

Партньори:

DRAFT KINGS
THE CROWN IS YOURS

StorPool
DISTRIBUTED STORAGE

DataArt

Следете актуалните обяви за **DevOps**

DEV.BG

The Road to GitOps: DevOps

DevOps is a set of practices that combines software development and IT operations to deliver software solutions more **quickly**, **reliably**, and **stably**

Core Principles

- Culture
- Automation
- Continuous improvement
- Platform design

Key Practices

- Continuous Integration / Continuous Delivery
- Infrastructure as Code
- Microservices
- Platform engineering

What is GitOps

GitOps = **Git** + **Operations**

GitOps is a set of (DevOps) practices
for managing infrastructure and applications
using Git as the single source of truth

GitOps Principles

- **Declarative**

  A system managed by GitOps must have its **desired state** expressed declaratively

- **Versioned and Immutable**

  Desired state is stored in a way that enforces immutability, versioning and retains a complete version history

- **Pulled Automatically**

  Software agents automatically pull the **desired state** declarations from the source

- **Continuously Reconciled**

  Software agents continuously observe actual system state and attempt to apply the **desired state**

Why GitOps and Kubernetes

- Containers are the way to run applications and microservices
- They are usually managed by orchestrators like Kubernetes and Nomad
- Some of the strengths of Kubernetes include
  - Automation
  - Self-healing
  - Portability
  - Declarative management
- Kubernetes doesn't enforce a way to manage configurations
- GitOps adds an operational discipline on top of Kubernetes

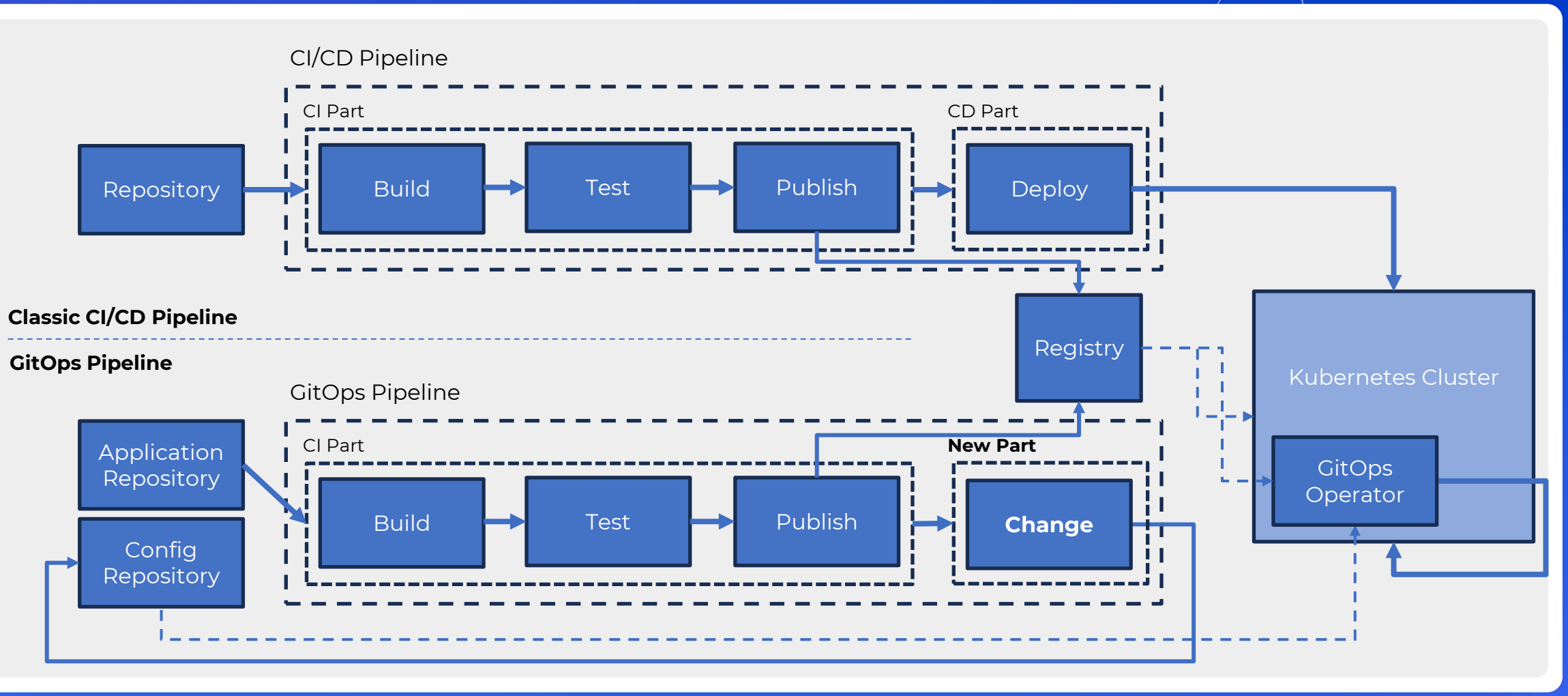
Key Benefits

- Faster releases through automation
- Safer operations thanks to versioned audits
- Easy rollbacks by reverting Git commits
- Consistent environments across dev, test, and production
- Improved security by minimizing direct access to systems

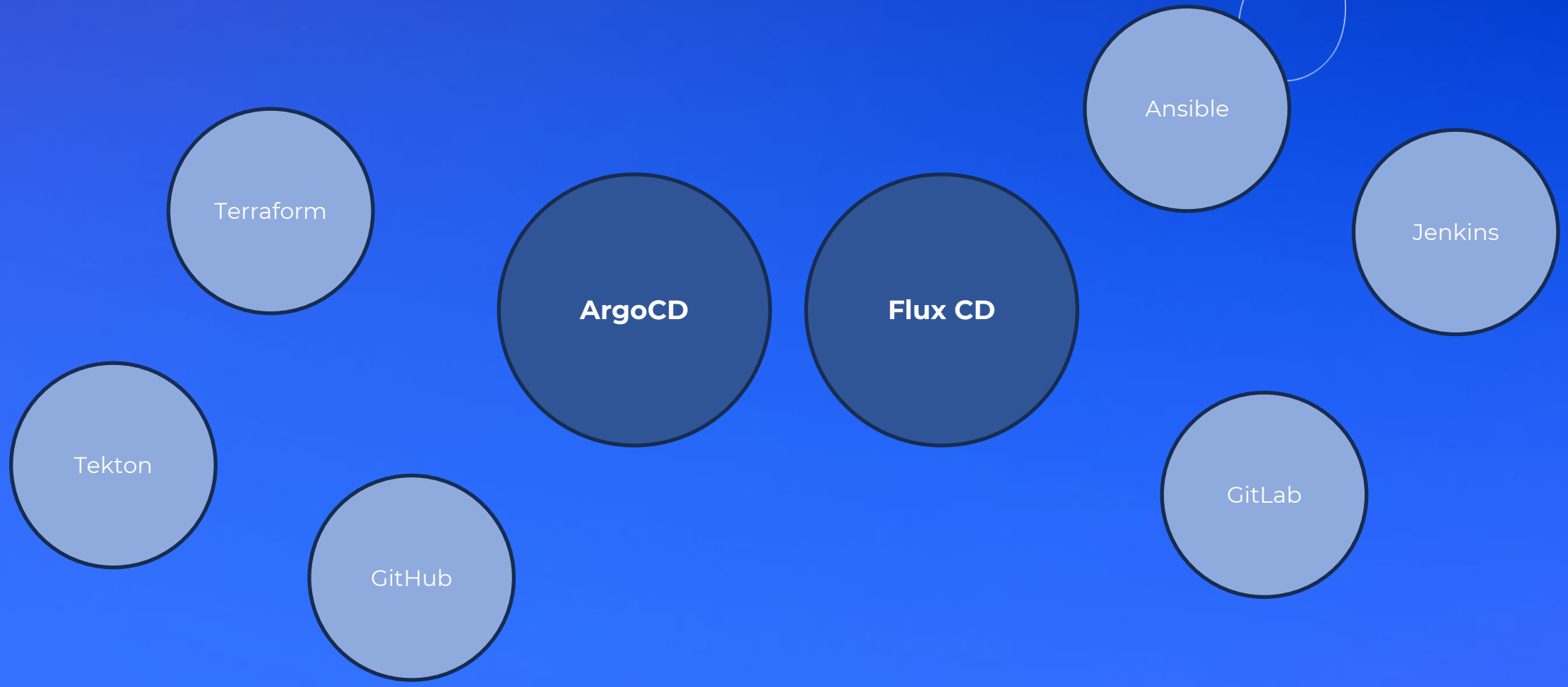
Typical Challenges

- Steep learning curve
- Tooling fragmentation
- Security concerns
- Cultural shift

Pipeline Evolution



The GitOps Universe *



ArgoCD

- Declarative application definitions
- Automated synchronization
- Real-time application status monitoring
- Role-based access control (RBAC)
- Multi-cluster support
- Web UI and CLI
- Created by Intuit, now part of CNCF

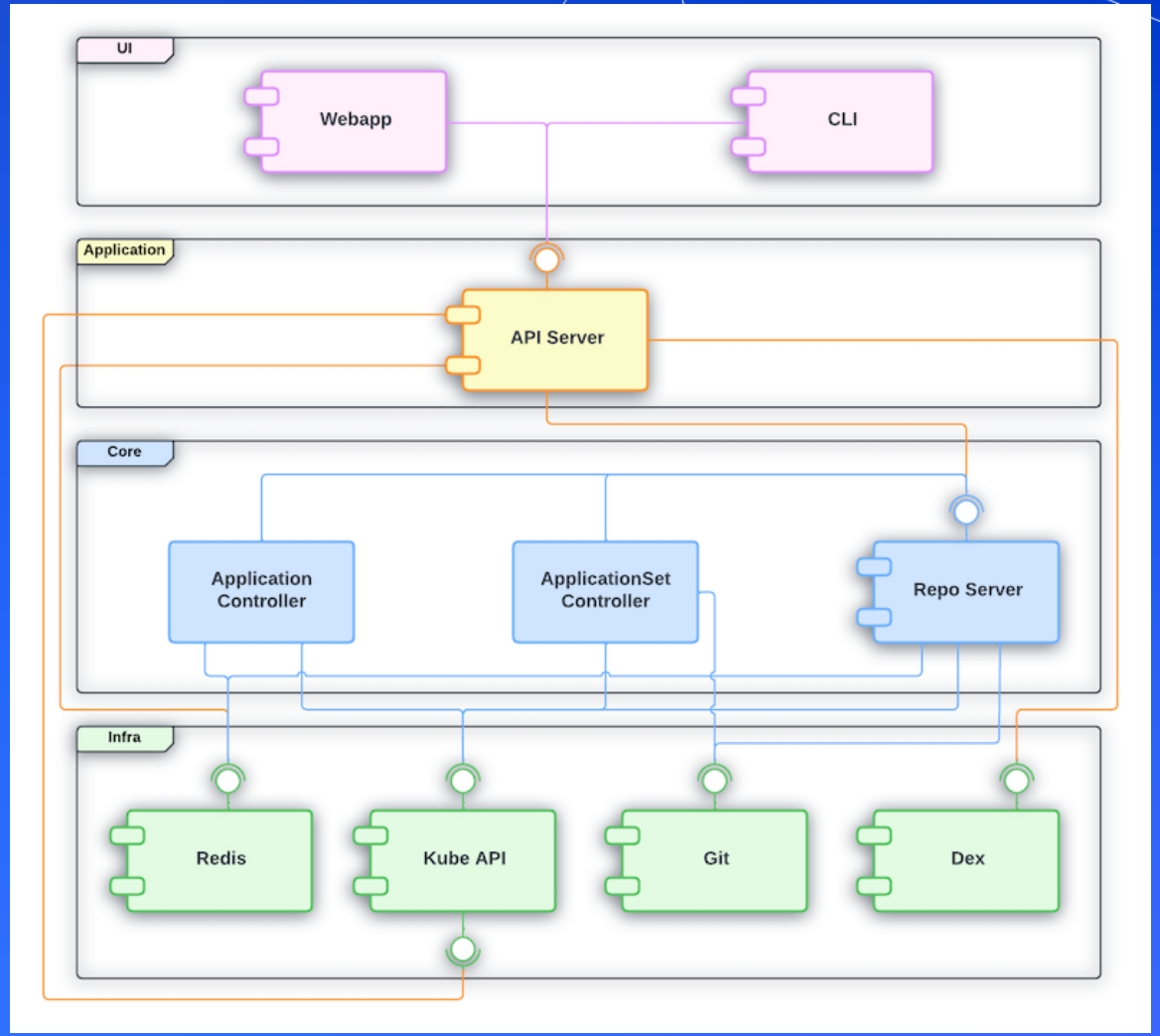
ArgoCD Architecture

- External components
  - Redis
  - Kubernetes API
  - Git
  - Dex

ArgoCD relies on those components

- Internal components
  - Application Controller
  - ApplicationSet Controller
  - Repo Server
  - API Server
  - Webapp and CLI

Based on the installation profile, some or all of those are being installed

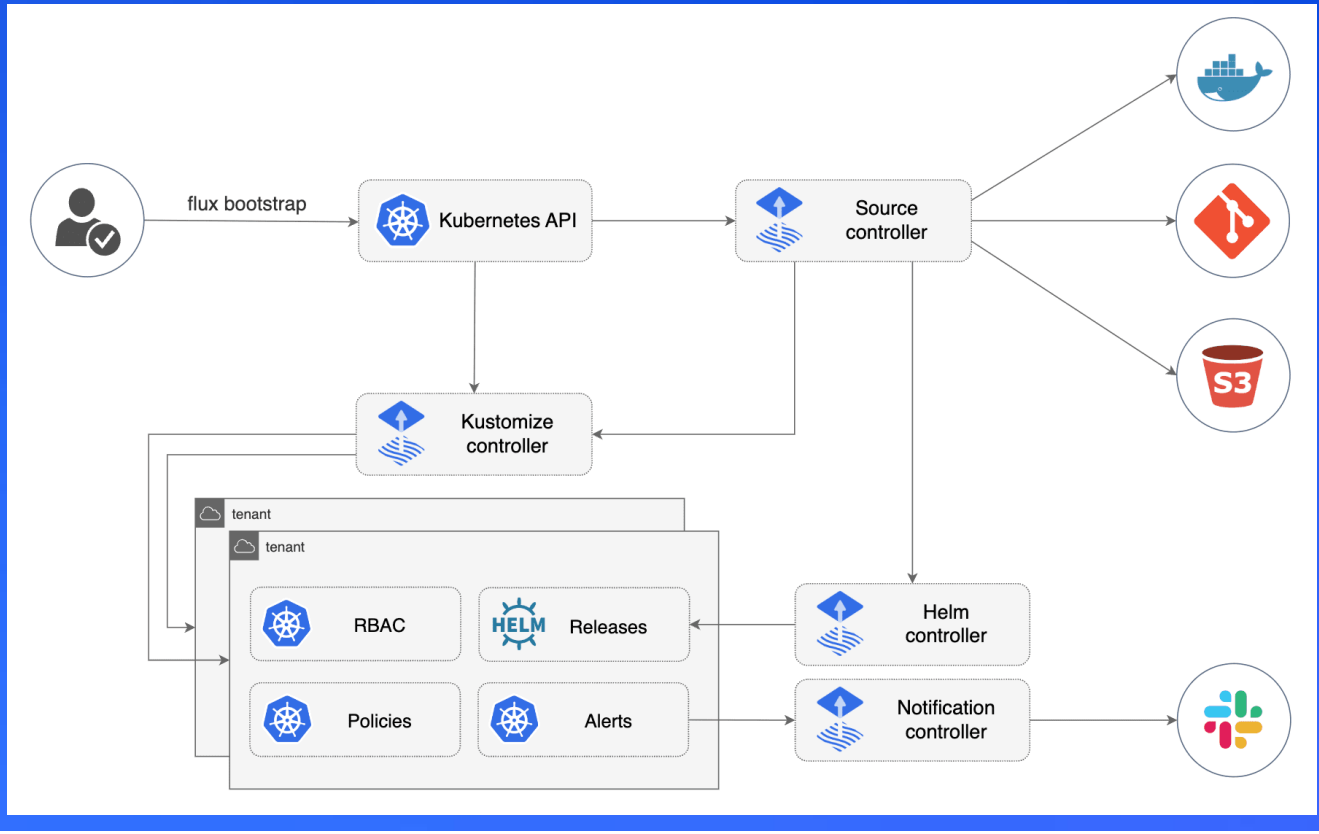


Flux CD

- Kubernetes RBAC support
- Supports image patches and updates automation
- A multi-tenant enabled architecture
- Compatible with Kubernetes Cluster API
- A rich ecosystem of extensions, interfaces, and platforms
- Created by Weaveworks, now part of CNCF

Flux CD Architecture

- Source Controller
- Kustomize Controller
- Helm Controller
- Notification Controller
- Image Automation Controllers



ArgoCD vs Flux CD

Feature	ArgoCD	Flux CD
CNCF Status	Graduated	Graduated
Design Architecture	Complete application	Modular and extensible toolkit
User Interaction	Built-in web UI, CLI available	CLI-first, optional lightweight web UI
Configuration Sources	Git, Kustomize, Helm	Git, Kustomize, Helm, OCI, S3
Reconciliation Process	Manual and automatic syncs	Manual and automatic syncs
Progressive Delivery	Argo Rollouts	Flagger
Access Management	Local users, SSO, built-in RBAC	Kubernetes identity and RBAC
Extensibility	Limited customization options	Good, third-party integrations
Ease of Use	Easy	Moderate

Best Practices (1)

- **Separation of Concerns**
  - Separate repositories for application code and configuration
  - For complex systems, consider further separation for different components
  - Represent environments via directories in the configuration repositories
  - Separate the responsibilities of development, delivery, and deployment
- **Declarative Configuration**
  - Use manifest and leverage tools like Helm, Kustomize, and Terraform
  - Strive for idempotency and reproducibility

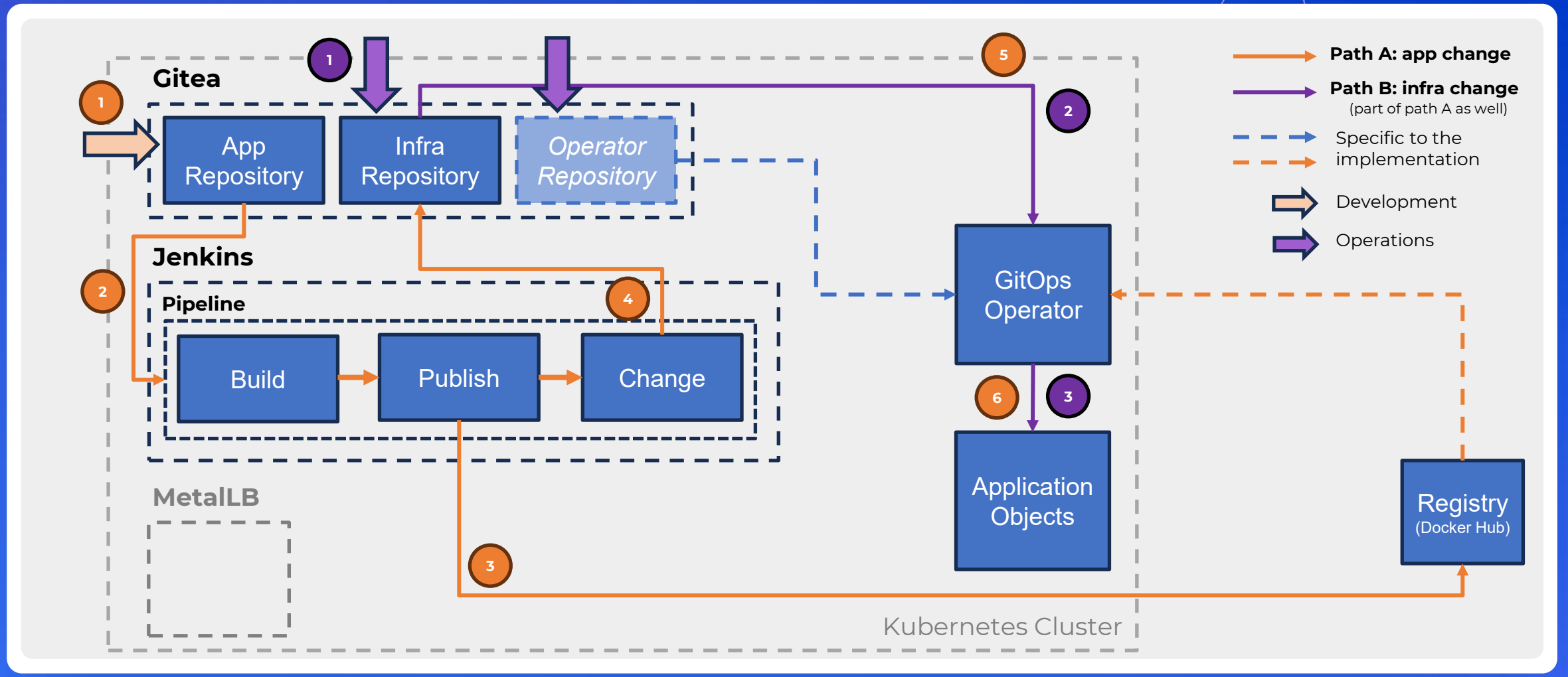
Best Practices (2)

- **Automation, Observability, and Reconciliation**
  - Automate the whole process, from building, to testing, and deployment
  - Use pull-based agent-driven deployments
  - Continuously monitor for drift between the desired and actual state
  - Implement automated reconciliation
- **Security and Governance**
  - Encrypt sensitive data stored in the repository
  - Even better, use a dedicated secrets management solution (like Vault)
  - Implement strict access controls

Demo

Let’s see it in action

Demo Setup



Q&A Session

You ask, I answer (if I can)

Q&A

Задай въпрос на vevox.app
ID: 192-474-853

Thank you!

Contacts:

https://www.linkedin.com/in/dzahariev/

https://github.com/shekeriev/