Automated Deployment to Kubernetes with Go and GitHub Webhooks

Rui Wang[1]

rwa041@uib.no

[1]**Digital development group, education and research support section, University of Bergen Library, Bergen, Norway**

18.09.2024

Motivation and Benefits

- **Manual Deployment Challenges:**
  - Time-consuming and error-prone.
  - Requires manual intervention.
  - Hard to maintain consistency across environments.
- **Automated Deployment Benefits:**
  - Saves time and reduces human effort.
  - Ensures consistent deployments across environments.
  - Integrates with CI/CD pipelines for faster development.
  - Automates deployment via GitHub webhooks.
  - Boosts reliability and minimizes failures.

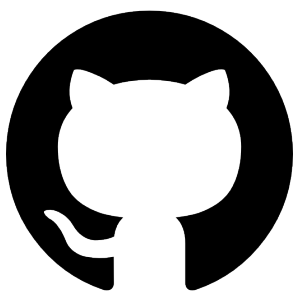
Application to be deployed

- **Hono API** on GitHub uib-ub organization's repository
- Containerizing the API in organization packages
- Deployment across environments: development, test, and production
- GitHub workflows for handling API secrets and environments

Automated Deployment Approach

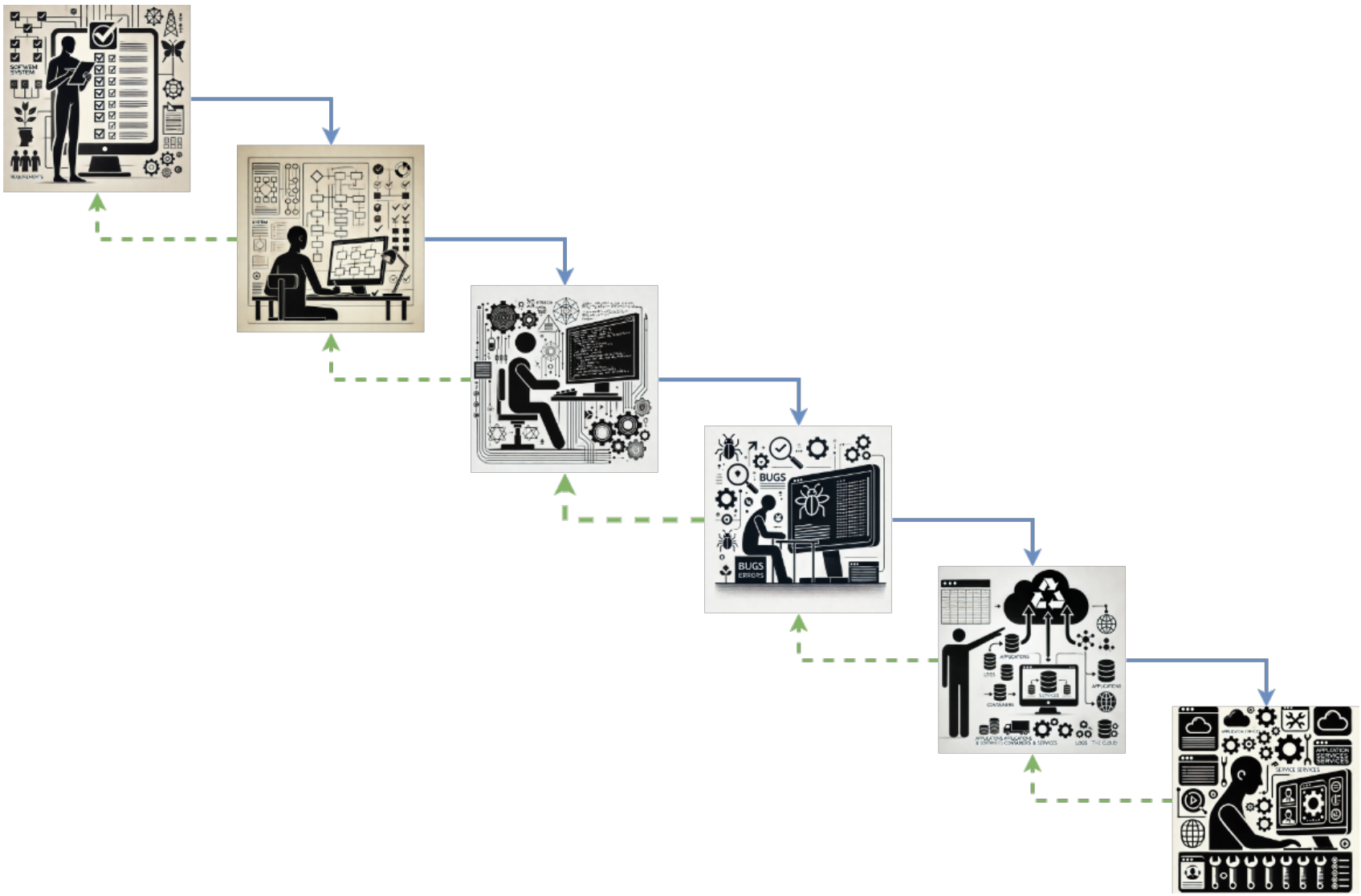
Approach: **Webhook-Kube-Auto-Deploy**

- Implemented in Go
- Containerized, deployed and runs in Kubernetes
- Utilizes GitHub webhooks events:
  - Issue Commits Event
  - Pull Requests Event
- Handles containerization
- Triggers GitHub workflows to handle Kubernetes secrets
- Automates deployment processes across environments




Iterative Waterfall Model

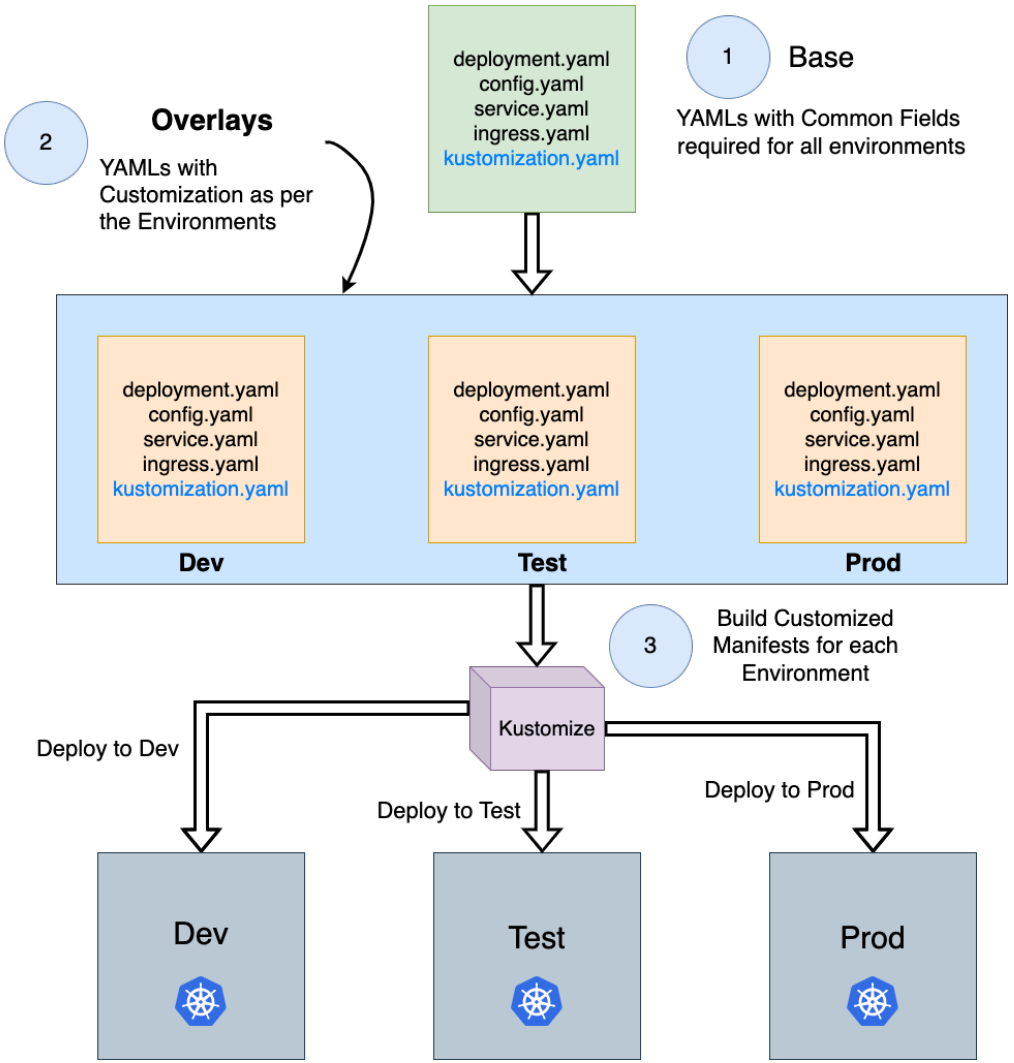
Development Process:



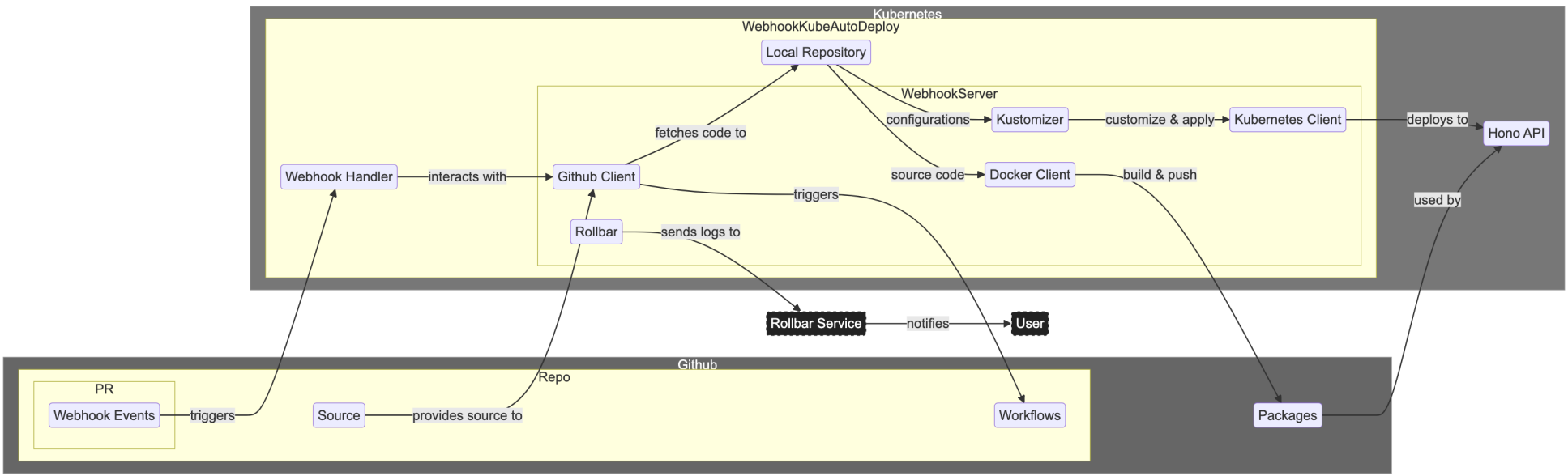
Requirements

- Automate containerization processes:
  - build docker images
  - push images to the container registry (GitHub packages)
  - delete images from both local and the registry
- Automate deployment processes across environments:
  - **"dev"** is deployed in pull requests as a "preview"
  - **"test"** is deployed when a pull request is merged to main
  - **"prod"** is deployed only on main branch

Kustomize for Kubernetes configuration and management

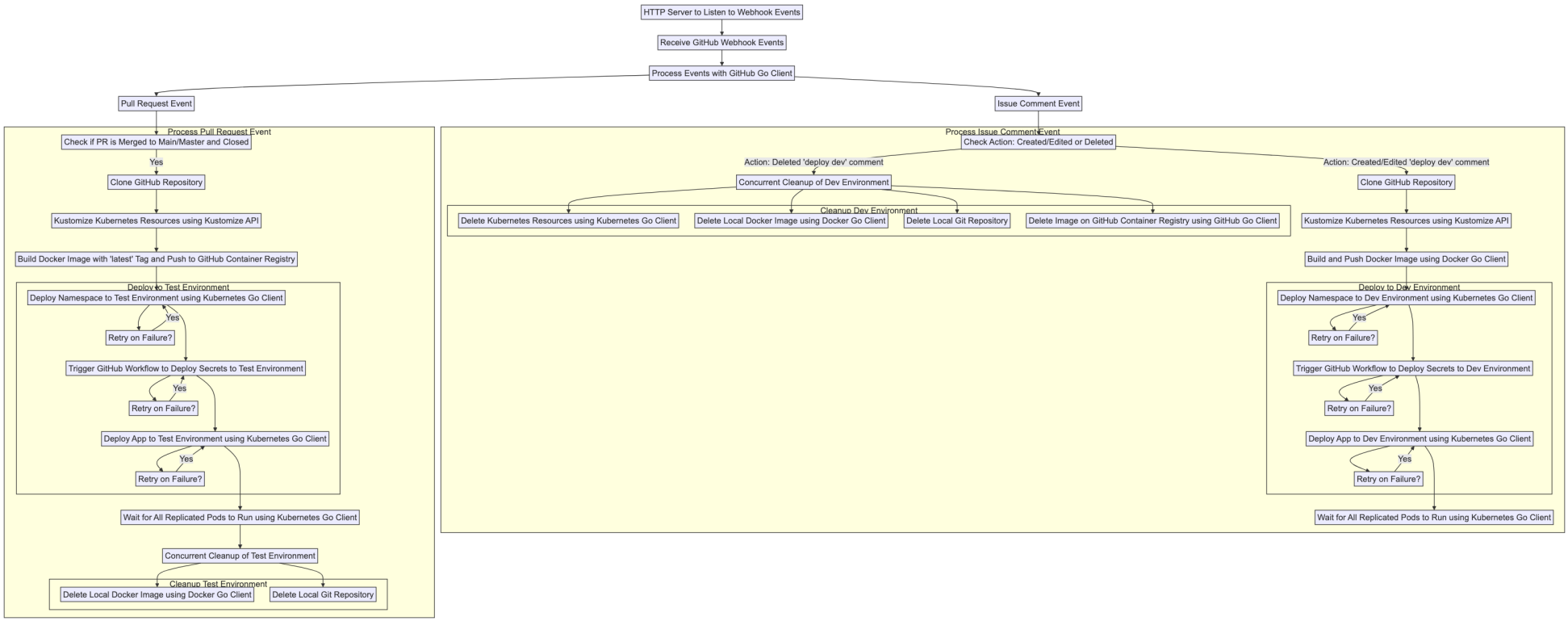


Architecture:



Design

Workflow:



Implementation

- Webhook event handler
- Webhook server integrates with multiple Go clients:
  - **GitHub**: Listens for specific webhook events and handles workflow actions, packages, and repository.
  - **Docker**: Builds, pushes, and deletes Docker images.
  - **Kustomize**: Builds Kubernetes configuration resources based on different environments.
  - **Kubernetes**: Deploys and manages Kubernetes resources.
- Integrates with **Rollbar** for error monitoring and logging.
- Health checks for liveness and readiness
- Follows **Google Go Code Style Guide**.

Testing and Coverage

Approaches

- **Table-driven Tests in Go:** Organizes test cases where a table of inputs and expected outputs is defined to systematically validate multiple scenarios.
- **Unit Testing:** Ensures individual functions and methods of all integrated Go clients work as expected.

Code Coverage

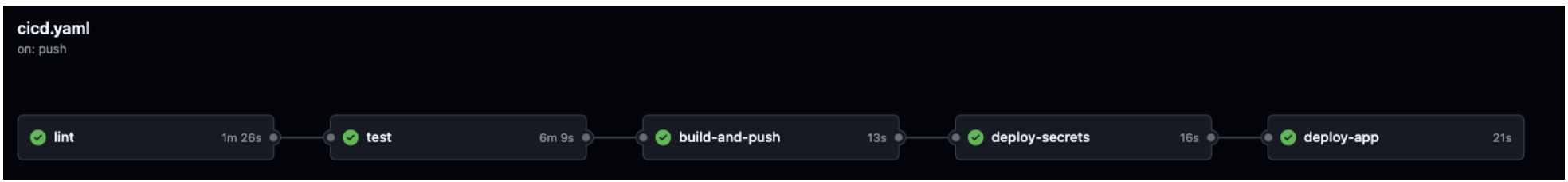
- **Local development:** Go's built-in testing framework is used to track code coverage.
- **CICD workflow:** Codecov is integrated via GitHub Actions to provide insights on coverage.
- 82% of the code coverage is achieved for integrated Go clients

Kubernetes YAML Configuration

- **Deployment:** Manages application pods.
- **Service:** Exposes app internally via `ClusterIP` on port 80, forwards to port 8080 (Service: `Webhook-Kube-Auto-Deploy`).
- **Ingress:** Routes external traffic (`api-git-deploy.testdu.uib.no`) and handles TLS via Let's Encrypt.
- **ServiceAccount:** Defines a service account for the deployment.
- **ClusterRole:** Grants access to namespaces and resources for `Webhook-Kube-Auto-Deploy`.
- **ClusterRoleBinding:** Binds ClusterRole to the ServiceAccount.

Deployment

GitHub Action Workflow:



Maintenance

- Documentation
- Kubernetes cluster on NREC
- Error tracking and reporting

Demo

Future work

- Extend it to be generic for other application development
- Optimize containerization for efficiency
- Currently only work with GitHub
- General improvements