



# How the CKI team hacks on its service

Cyborg Infra Workshop 2021: Day 2

Iñaki Malerba

Michael Hofmann

# Stef's Open Source playbook to build SaaS Community

Picking some quotes

To enable efficient contributions both from the team and the world:

- ▶ Discover the documentation for contributing
- ▶ Understand what is running, the makeup of the service
- ▶ Identify the component
- ▶ Build the component locally
- ▶ MRs tested automatically
- ▶ After cross-check, MRs deployed in a testing environment
- ▶ ...

... in other words, documentation and DevOps

# Dimensions of hackability

I just made those up

- ▶ **Openness**
  - Public repos, documentation
  - MR workflow
- ▶ **Safety**
  - Continuous Integration: unit, integration, system tests
  - Can be run locally without interfering with production
- ▶ **Automated deployment**
  - Testing/Staging/Canary deployments
  - Continuous Delivery/Deployment

The good parts...

The good parts...

# Public repos (a lot)

From pillar to post

- ▶ Nearly everything public: <https://gitlab.com/cki-project> ~30 projects
  - Microservices, pipeline components, container images, ...
  - Issue tracker
- ▶ Internal: <https://gitlab.cee.redhat.com/cki-project>
  - Put another firewall in front of secret stuff
  - Credentials, internal docs, RHEL configuration, legacy projects
  - Deployment configuration including secrets
- ▶ ?? Secret management:
  - Split deployment into public configuration and internal secrets?

# Documentation

"A little inaccuracy saves tons of explanations" - H. H. Munro

- ▶ Public: <https://cki-project.org/docs/hacking/>
  - Plus individual README.md files per repo
- ▶ Internal: <https://documentation.internal.cki-project.org/>
- ▶ Inventory: <https://cki-project.org/docs/hacking/inventory/>
  - Components, dependencies, monitoring (WIP)
- ▶ RFC process: <https://cki-project.org/docs/hacking/rfcs/>
  - Asking for feedback
- ▶ ?? Integration of different pieces of documentation:
  - At the moment, inventory links to pieces





```
1 ---
2 kind: cronjob
3 summary: Sync the lookaside git repository to an S3 bucket
4 description: |
5     Some files need to be hosted somewhere to be accessible during tests without
6     SSL. For that purpose, the git-s3-sync module can sync the lookaside git
7     repository at https://gitlab.com/cki-project/lookaside/ to an S3 bucket.
8
9 infrastructure: ocp45
10
11 people:
12   - name: infra
13
14 docs:
15   - name: README
16     url: https://gitlab.com/cki-project/schedule-bot#git_s3_sync
17     repo: https://gitlab.com/cki-project/schedule-bot
18     path: README.md
19
20 sources:
21   - name: schedule-bot
22     repo: https://gitlab.com/cki-project/schedule-bot
23     path: git_s3_sync
24   - name: deployment-all
25     repo: https://gitlab.cee.redhat.com/cki-project/deployment-all
26     path: schedules/lookaside.yml
27
28 dependencies:
29   runtime:
30     - name: storage/s3-aws-arr
31       annotation: writes
32     - name: external/gitlab-com
33       annotation: clones
```

Documentation

Kernel developers

Test maintainers

Hacking

Contributing

RFCs

Operations

Inventory

Services

**Cron jobs**datawarehouse-  
backupdatawarehouse-  
report-crawlergit-cache-  
updaterlookaside-kernel-  
configs[Documentation](#) / [Hacking](#) / [Inventory](#) / [Cron jobs](#) / [lookaside-static](#)

# lookaside-static

Some files need to be hosted somewhere to be accessible during tests without SSL. For that purpose, the git-s3-sync module can sync the lookaside git repository at <https://gitlab.com/cki-project/lookaside/> to an S3 bucket.

- Kind: [cronjob](#)
- Infrastructure: [ocp45](#)
- People:
  - [infra](#)
- Documentation:
  - **README:** `README.md` at <https://gitlab.com/cki-project/schedule-bot>
- Sources:
  - schedule-bot: `git_s3_sync` at <https://gitlab.com/cki-project/schedule-bot>
  - deployment-all: `schedules/lookaside.yml` at <https://gitlab.cee.redhat.com/cki-project/deployment-all>
- Runtime dependencies:
  - writes: [storage/s3-aws-arr](#)
  - clones: `external/gitlab-com` (missing)



# CKI-001: CKI feedback mechanism

Description of the process behind the CKI feedback mechanism based on Requests for Comments (RFCs)

Michael Hofmann – [cki-project/documentation!49](#)

## 1 Abstract

This document specifies the process behind the feedback mechanism for the CKI project based on Request for Comments (RFCs). Each RFC documents a *need*, proposed *solutions* and links to the related *discussions*.

## 2 Motivation

The internal Red Hat `#kernelci` IRC channel is the place where nearly all CKI project communication happens. Unless a project member is online, logged into IRC and paying attention all the time, ad-hoc discussion of important topics might be missed. As a consequence, people might feel left out of the decision-making process, and proposed solutions might suffer from the lack of feedback.

## 3 Approach

A structured process for gathering feedback is introduced.

CKI RFCs (“Requests for Comments”) are markdown documents proposing to create or change something, and soliciting discussion and feedback. They live in the [documentation repository](#) and can be browsed at <https://cki-project.org/docs/hacking/rfcs/>. They are submitted and discussed via merge requests. Within the default time frame of one week, everyone is invited to give feedback on them.

# Continuous integration

## Who to blame for line length limits

- ▶ GitLab CI YAML file
  - Run container-based shell jobs for branches/tags/MRs
- ▶ Python linting: pylint, isort, flake8, pydocstyle, pytest wrapped in tox
  - Simple to run locally, in podman container, in CI
  - One common linting script to rule [all Python repositories]
  - Test coverage with regression check
  - Build and upload container images per MR
  - Convincing the team of using black is another matter
- ▶ Shell linting: shellcheck, check shell-in-yaml
- ▶ Documentation linting: link checking, review environments

## MR workflow/continuous deployment

Running main all the time: what do you mean with "you are scared"?

- ▶ **Merging an MR means deploying**
- ▶ At the end of a successful review, an MR is only approved
- ▶ For CKI team members, MR author merges themselves
  - Whoever does the merging has to handle any fallout
  - Not Done on a Friday or right before the end of the work day
- ▶ ?? Rollbacks:
  - Reverting a commit: 15..30 minutes until deployment
  - Temporary: `oc describe + oc tag image@sha256 is/name`

# Automated deployment

Move fast and break all things: deployment automation

- ▶ One deployment repo for Kubernetes YAMLs/Ansible for all projects
- ▶ Everything is deployed from there, no manual editing allowed
- ▶ Everything is redeployed on each change
  - 93 deployment jobs...

# The good parts: summary

## Patting yourself on the back

- ▶ Public repos: 8/10
  - Still wondering about the secrets split
- ▶ Documentation: 6/10
  - Inventory idea stolen from EXD seems promising
- ▶ Continuous integration: 8/10
  - No "prototypes", do it correctly from the beginning
- ▶ Continuous deployment: 8/10
  - Poor-man's rollback by reverting commits

... and the less good parts



# Automated deployments of unmerged code

## The two elephants in the room

- ▶ Local deployments
  - Somehow, run the code locally
- ▶ Staging deployments
  - Somehow, run the code somewhere else
  - Should be able to run production(-like) workloads
  - Must not interfere with production

## Some background: CKI service structure

Everything is better in layers

- ▶ Essential infrastructure outside the control of CKI
  - OpenShift, Beaker, AWS, gitlab.com, ...
- ▶ Communication fabric
  - AMQP cluster hosting work queues, webhook receiver, ...
- ▶ Internal microservices
  - Trigger pipelines, run them to completion, send email reports, ...
- ▶ Pipeline components
  - Gitlab-runner, test database, beaker provisioner, ...

# Testing deployments of the communication fabric

## High Stakes ~~Gambling~~ Development

- ▶ Very little code + Ansible deployment:
  - GitLab webhook receiver for sending AMQP: AWS Lambda
  - RabbitMQ cluster: AWS Route53/EC2
- ▶ Local deployments: not really possible atm
- ▶ Staging deployment:
  - Shell script to setup staging Lambda + AMQP cluster (WIP)
  - Manual updates of Lambda ZIP file
- ▶ ?? New AWS/Lambda container image support

# Testing deployments of internal microservices

Ask YouTube for "LAMBDA - A Serverless Musical"

- ▶ Everything packed into container images
- ▶ IS\_PRODUCTION=false env variable to prevent interference
- ▶ Local deployments:
  - Services have one-shot CLI interfaces (WIP)
  - Podman run, but there are a lot of env variables to get right
- ▶ Staging deployment:
  - shell script to deploy container images from MR
  - modifies configs to run them in non-production mode (WIP)
  - no bot support yet (planned)

# Testing deployments of pipeline components

Pip is reliable and stable right?

- ▶ Python CLI tools duct taped by Bash, pip and YAML
- ▶ Local deployments:
  - Standalone CLI tools: well supported
  - Pipeline YAML: not at all
- ▶ Staging deployments:
  - In the MR, **talk to the bot** which does the right thing for the repo
  - Rerun old successful pipelines with new code/YAML/config
  - Tagged so they do not cause user-visible effects
  - Same observability as production pipelines



**Veronika Kabátová** @veruu · 6 days ago  
Resolved by [Veronika Kabátová](#) 6 days ago

Owner



[@cki-ci-bot](#) please test [cki/620247] with [compiler=clang] again but a new container that has which:  
[builder\_image=quay.io/cki/builder-fedora] [builder\_image\_tag=mr-213]



▼ Collapse replies



**CKI CI Bot** @cki-ci-bot · 6 days ago

Maintainer



Group	Branch	ID	Status
cki	620247	<a href="#">620902</a>	! failed

Edited by CKI CI Bot 6 days ago



## The bad parts: summary

*After all, tomorrow is another day*

- ▶ Communication fabric: 2/10
  - Lambda container images seem promising
- ▶ Internal microservices: 6/10
  - Reference secrets for local deployment, bot support missing
- ▶ Pipeline components: 8/10
  - Production workloads in end-to-end tests for MR code
- ▶ ?? Improvements:
  - We fail mostly in making it easy to deploy stuff locally
  - Problematic observability of deployments for untrusted users