From Jenkins under your desk to resilient service

Continuous Kernel Integration is growing up

Iñaki Malerba

Michael Hofmann

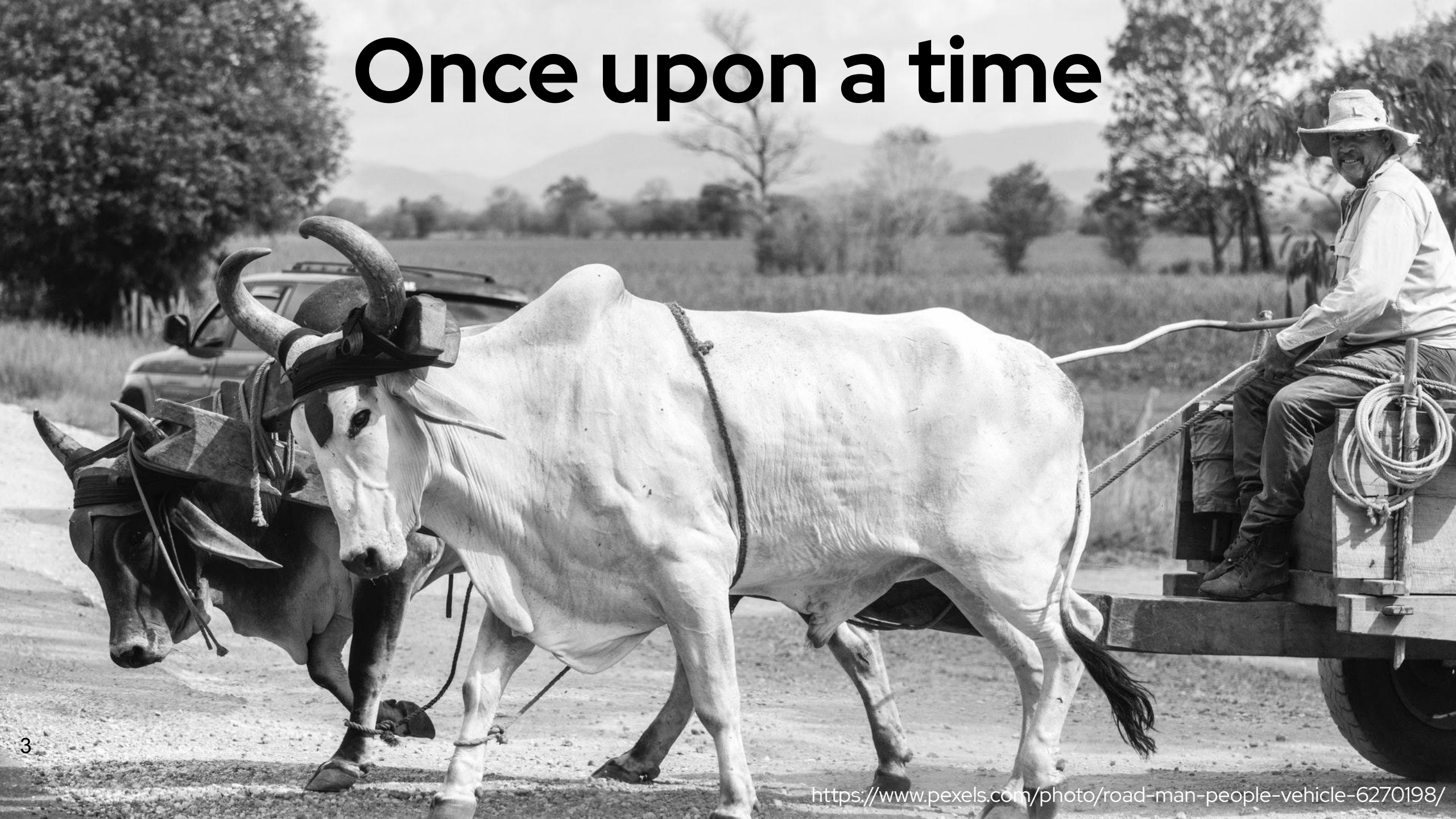
Continuous Kernel Integration: problem statement

Avoiding grumpy kernel developers

CI for the Linux Kernel:

- For each commit under test, run a build+test pipeline to completion
- Ideally that means:
  - detecting a commit
  - triggering a pipeline
  - reporting results
- How hard can it be...

Once upon a time



https://www.pexels.com/photo/road-man-people-vehicle-6270198/

Simpler times: just running pipelines

"You can never go wrong by using Jenkins"

- Jenkins environment split into staging/production
- a lot of Python + Groovy
- an OpenShift project and a lot of clicking

... doesn't scale that well 😕

Continuous Kernel Integration: problem statement, revised

Prevent bugs from being merged into kernel trees by providing CI as a service

Kernel developers

- Onboard new kernel trees
- Run compile and testing pipelines for Linux kernels from Brew, Koji, git repos, GitLab MRs, Patchwork, ...
- RPM package gating
- Provide infrastructure for kernel workflow

Test Maintainers

- Onboard new kernel test (frameworks)
- Configuration of targeted testing
- Visualization and statistics for test failures

Where we want to be...

https://www.pexels.com/photo/timelapse-cityscape-photography-during-night-time-599982/

Buzzword bingo

Turns out this is slightly more complicated

Serverless	Microservices	Containers
Processes	Agile	DevOps
Cross-architecture	Cloud	Chatbot
Deploy before merge	Kubernetes	Default to Open
Open Source SaaS	Site Reliability Engineering	Continuous Delivery

Name or Service not known

https://www.pexels.com/photo/grey-kitten-on-silver-paper-bag-141496/

Part 1: Keeping it running (SRE)

General idea: reliable service on unreliable infrastructure

Murphy and It's Always DNS

- Lemma 1: Any component/dependency that can fail will fail
  - ... some will fail more than others
- Lemma 2: nearly all failures can be retried successfully
  - ... but we also have to detect the other ones
- So failures need to be...
  - Prevented: fewer/simpler components/dependencies
  - Detected: logging, monitoring, alerting
  - Recovered: retries at all levels, fallbacks

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, ...

Keeping it running: Prevention

Minimize the essentials

Less critical pieces means less critical failures

- Essential components
  - Needed for the service to run (SPOFs)
- Necessary components
  - Have to work at least *sometimes*
- Optional components
  - Only provide observability and increase reliability

Minimization example 1: Message Queues

Loose coupling for code

- Message queues instead of REST APIs
- Reliable and distributed
- Increase service portability
- Automatically reprocessed failed messages after some time
- Examples at CKI:
  - AWS-based RabbitMQ cluster with automatic retry queues
  - Reliable webhooks: webhook-bridge
  - Retries for UMB/fedmsg processing: amqp-bridge

- https://cki-project.org/docs/hacking/operations/messaging/

Minimization example 2: S3 buckets

Loose coupling for data

- S3 buckets instead of NFS, git hosting, etc
- Ubiquitous, fast and highly reliable: AWS, OCS, MinIO, ...
- Can be used as poor man's database
- Increase service portability
- Examples at CKI:
  - Ccache
  - Caching git repositories as tarballs
  - Pipeline artifacts
  - Static files and configurations

Minimization example 3: Container images

Commoditize all the things

- Serverless > containers > disposable VMs > pet VMs
- Infrastructure becomes somebody else's problem
- Everything wrapped up into container images to freeze time
- Examples at CKI:
  - AWS Lambda to host webhook bridge
  - Gitlab-runner to spawn jobs in docker, K8s, disposable VMs
  - OpenShift/K8s as workload API

Keeping it running: Detection

Detection

keeping track of many, many pieces

- Build and testing pipelines, micro services, cron jobs, FaaS
- AWS, OpenStack, K8s clusters, ...
- Logging, metrics, monitoring, alerting

Logging: Loki

at least not > /dev/null

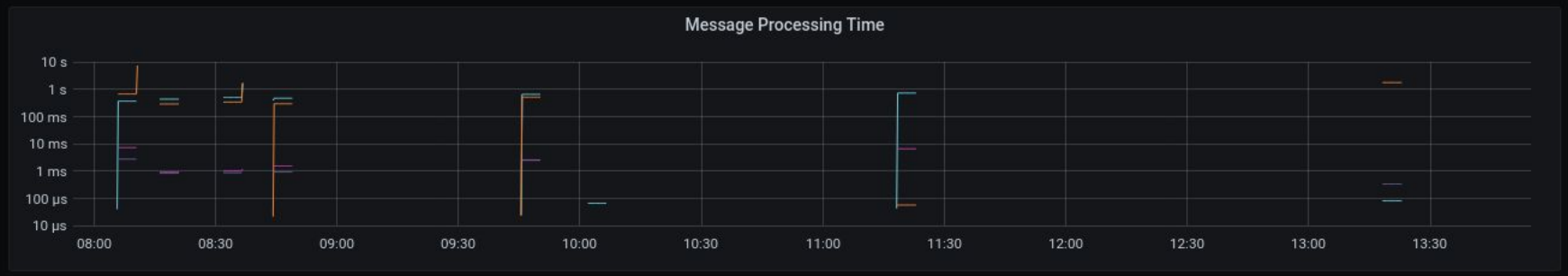
- Standardized Python logger names and levels
  - Easier to read and configure
- Putting all the logs on a common place
  - Shared volume within one K8s project
  - Human friendly, easily grepable
- Grafana Loki stack for processing
  - *'Like Prometheus, but for logs!'*
  - Indexed and easy retention policies

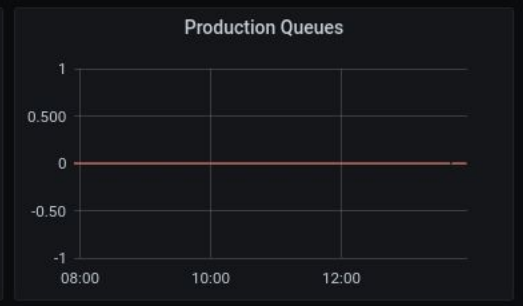
Metrics: Prometheus

everything deserves a /metrics endpoint

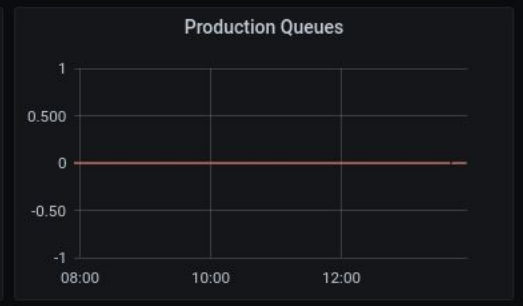
- Expose internal status of services
  - Monitor what a service is doing and how long it's taking
- Prometheus as an import-and-forget solution
  - Python's prometheus-client
  - Built in on many services
- K8s autodiscover and lay back
- Visualize via Grafana

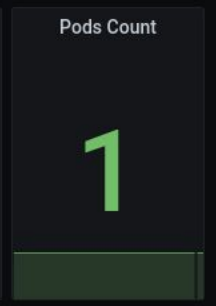
Triager



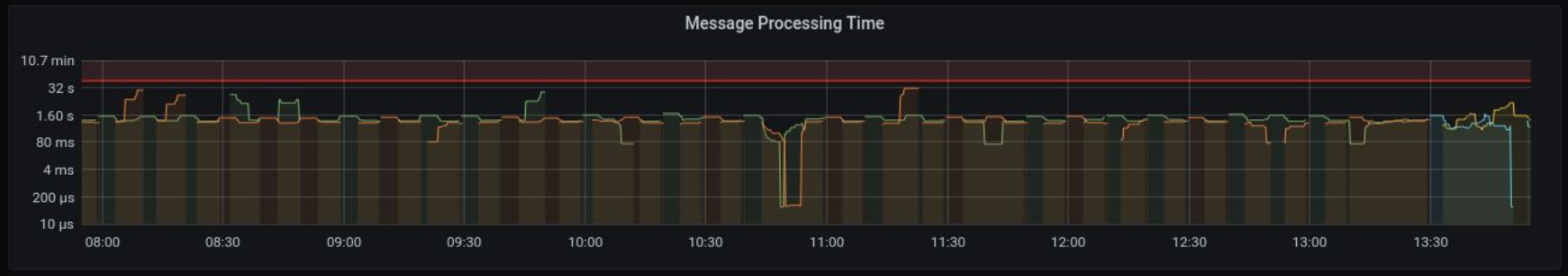


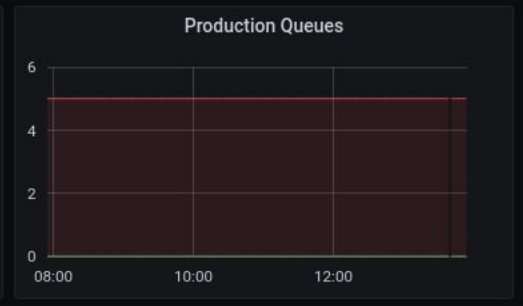
KCIDB Forwarder

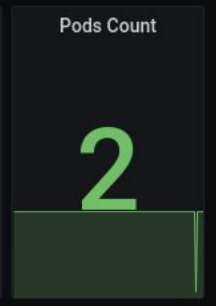




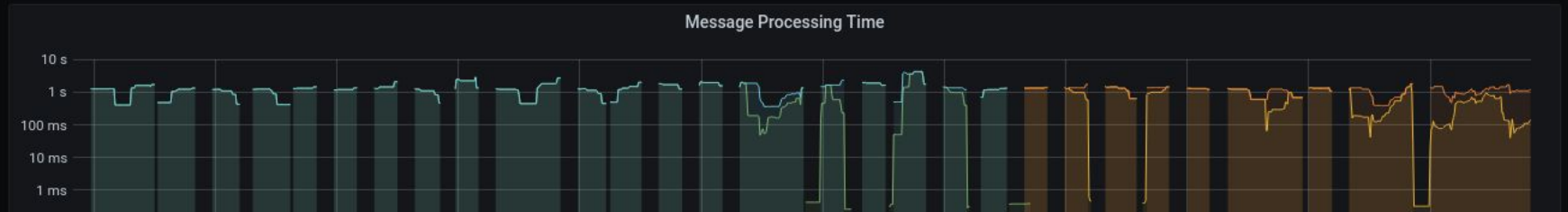
KCIDB Submitter

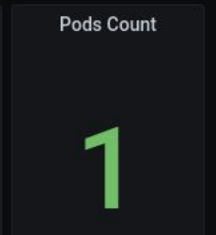






Pipeline Herder



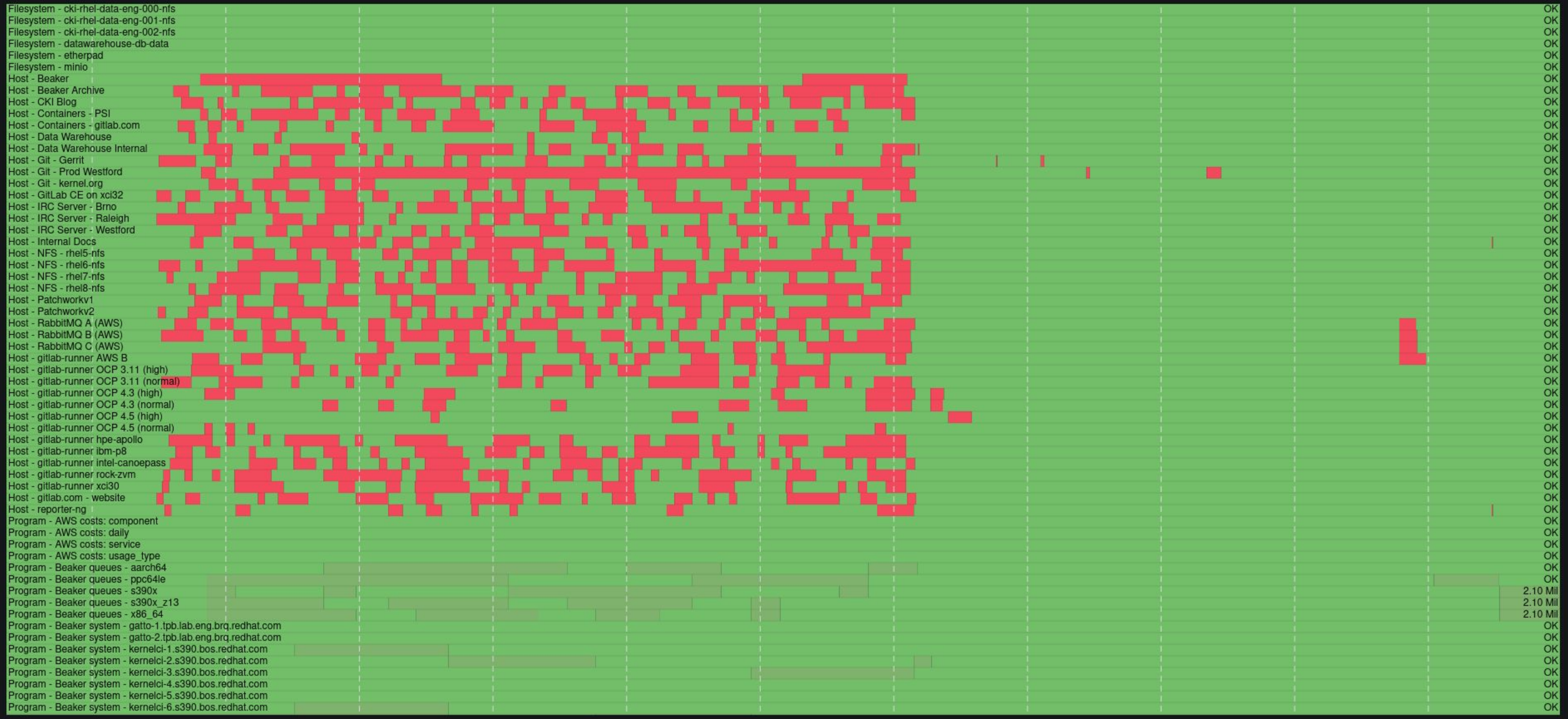


Monitoring: Monit

Just assume no one monitors their services

- Keep track of 3rd party resources that we depend on
- Monit as a simple solution for monitoring
  - Hosts uptime
  - NFS file systems uptime and size
  - Beaker hosts queues
  - S3 bucket sizes
  - RabbitMQ messages and queues
- Store instant statuses and **record** downtimes

Systems Status History



Monitoring: Sentry

How to be the first one to know when everything blows up

- Track errors in real time
- Allows to fix the long tail of unlikely errors
- sentry.engineering.redhat.com, sentry.io

Alerting

Where is my unsubscribe button

- Source: Monit, Sentry, Grafana
  - figuring out how to use Alertmanager is still on the TODO list
- Notify via dedicated mailing list
- Create real-time awareness via IRC
  - also notify about running pipelines, deployments, ...

Keeping it running: Recovery

Retries examples 1

What goes around comes around

- Retry **every** network access multiple times
  - looping helper for shell code
  - common Python code to setup a retrying requests session

Retries examples 2

Nothing as motivating to fix a bug as a full message queue

- Rescheduling messages with RabbitMQ
  - Endlessly circulate messages until successfully handled
  - Automatically reject messages on exception
  - Use DLX/TTL to requeue messages after some time **at the end**

Retries examples 3

"Ever tried. Ever failed. No matter. Try again. Fail again. Fail better." - Samuel Beckett

- Pipeline Herder:
  - Keeps track of failed GitLab jobs
  - Detects common transient errors
  - Retries jobs with increasing interval of time

Fallbacks

When retries are not enough

- Gitlab Runner's containerized jobs can run anywhere
- Runners set up on OSP, Beaker, different OCP clusters, AWS
- Fallbacks for multi-arch runners are hard to come by

Part 2: Making it hackable (DevOps)

Dimensions of hackability

I just made those up

- **Openness**
  - Public repos, documentation
  - MR workflow
- **Safety**
  - Continuous Integration: linting and testing
  - Understandable microservices
- **Easy deployment**
  - Local/Testing/Staging/Canary/Production deployments

Making it hackable: Openness

Open code

From pillar to post

- Nearly everything public: https://gitlab.com/cki-project ~30 projects
  - Microservices, pipeline components, container images, ...
- 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
- WIP:
  - split deployment into public YAML and private policy + secrets

Open 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
- Inventory: https://cki-project.org/docs/hacking/inventory/
  - Components, dependencies, monitoring (WIP)
- Internal: https://documentation.internal.cki-project.org/
- Documentation Friday
- WIP:
  - Integration of different pieces, internal -> public

lookaside-static.yml 906 Bytes

Edit | Web IDE | Lock | Replace | Delete

```yaml
---
kind: cronjob
summary: Sync the lookaside git repository to an S3 bucket
description: |
  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.

infrastructure: ocp45

people:
  - name: infra

docs:
  - name: README
    url: https://gitlab.com/cki-project/schedule-bot#git_s3_sync
    repo: https://gitlab.com/cki-project/schedule-bot
    path: README.md

sources:
  - name: schedule-bot
    repo: https://gitlab.com/cki-project/schedule-bot
    path: git_s3_sync
  - name: deployment-all
    repo: https://gitlab.cee.redhat.com/cki-project/deployment-all
    path: schedules/lookaside.yml

dependencies:
  runtime:
    - name: storage/s3-aws-arr
      annotation: writes
    - name: external/gitlab-com
      annotation: clones
```

Search this site...

Documentation
Kernel developers
Test maintainers
Hacking
Contributing
RFCs
Operations
Inventory
Services
Cron jobs
datawarehouse-backup
datawarehouse-report-crawler
git-cache-updater
lookaside-kernel-configs

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)

Open daily operations

Graphical strings also known as emojis

- Public issue tracker: gitlab.com, moved from internal Jira
- Internal #kernelci IRC channel: people and bots
  - Everybody and their emojis invited to join
  - Discussions, deployments, alerting, pipelines ...
- MR-based workflow
- RFC process: https://cki-project.org/docs/hacking/rfcs/
  - Asking for feedback

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

Edit this page
Create docum
1 Abstract
2 Motivation
3 Approach
3.1 Steps to subm
4 Benefits
5 Drawbacks
6 Alternatives

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.

Making it hackable: Safety

Continuous integration

Who to blame for line length limits

- As much linting and testing as possible
  - Shell, Shell-in-YAML
  - Python: pylint, isort, flake8, pydocstyle, pytest, coverage
  - Documentation: markdown, URLs, review environments
- One linting script to rule [all Python repositories] based on tox
  - Simple to run locally, in podman container, in CI
- WIP:
  - Convincing the team that using a formatter is a good thing
  - For libraries, testing all dependent projects

Understandable microservices

head <-> code size relation

- Code changes need to be predictable!
  - Simple mental models have to be good enough
  - If everybody is too scared to touch it, split it up
- Loose coupling means better interfaces
- Prioritize cleanups and fixes
  - Ignore management if they tell you otherwise

Making it hackable: Easy deployments

In general: continuous deployment/delivery

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

Reminder: 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, ...

Microservices: automated deployment

Move fast and break all things: deployment automation and no clicking allowed

- Everything is a container image
  - IS_PRODUCTION=false env variable to prevent interference
- Follow best practices, e.g. no configuration in the images
- 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 (~105 deployment jobs)
  - keeps everybody honest...

Microservices: Local deployments

You have to start small

- Build the image locally:

```
cki_build_image.sh irc-bot
```

- Or pull the image from the merge request:

```
podman pull registry.gitlab.com/cki-project/irc-bot:mr-123
```

- Use direnv and .envrc to keep configuration:

```
podman run -e var=value …
```

- Or just use the one-shot CLI interface to the service
- Summary: ok, but getting a working local configuration is painful

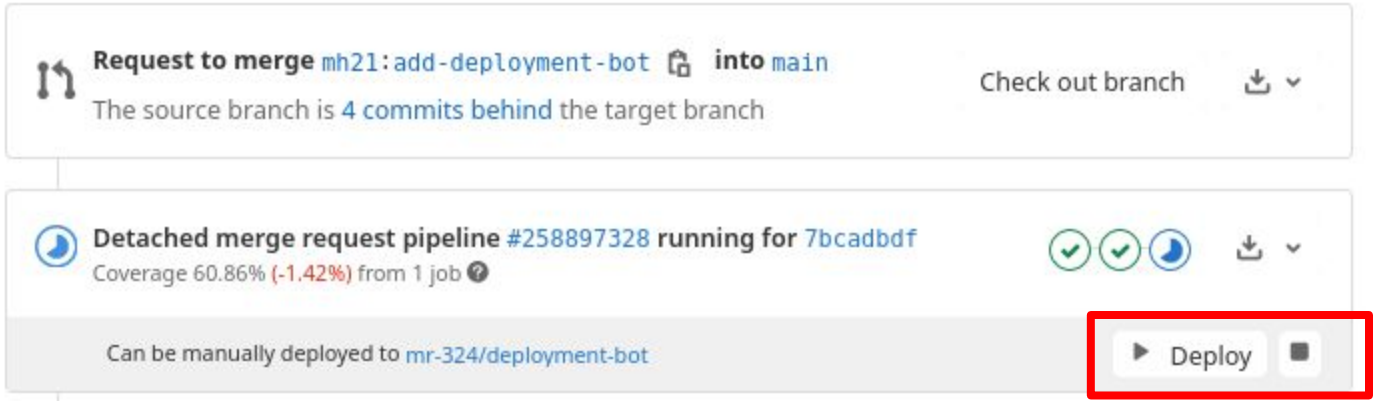
Microservices: testing deployments

Hopefully it doesn't bring down the cluster

- Deploys into K8s in **non-production mode** alongside production
- CLI on deployment repository checkout:

```
PROJECT_NAME=irc-bot PROJECT_CONTEXT=ocp4_prod \
 MERGE_REQUEST=mr-123 ./openshift_staging_{create,destroy}.sh
```

- From the GitLab UI:

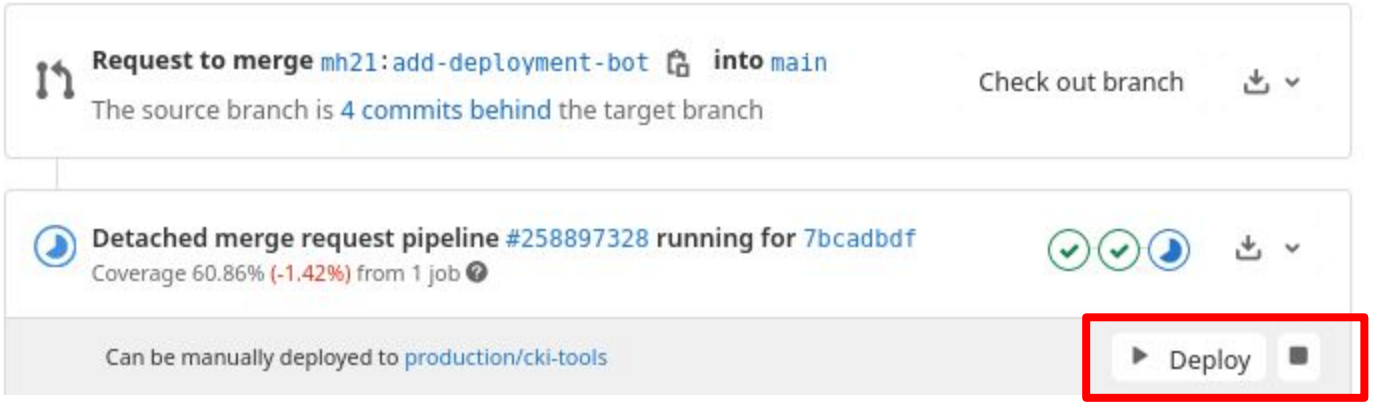


Red Hat

Microservices: production deployments

Did I mention "scared"?

- Merging to main triggers automatic redeployment
- For unmerged code, from the GitLab UI:



- Can be rolled back in the same way

Pipeline: local deployments

Pip is reliable and stable right?

- Pipeline: Python CLI tools duct taped by Bash, pip and YAML
  - Sadly no local testing of all the YAML
- For the python tools, install via pip:

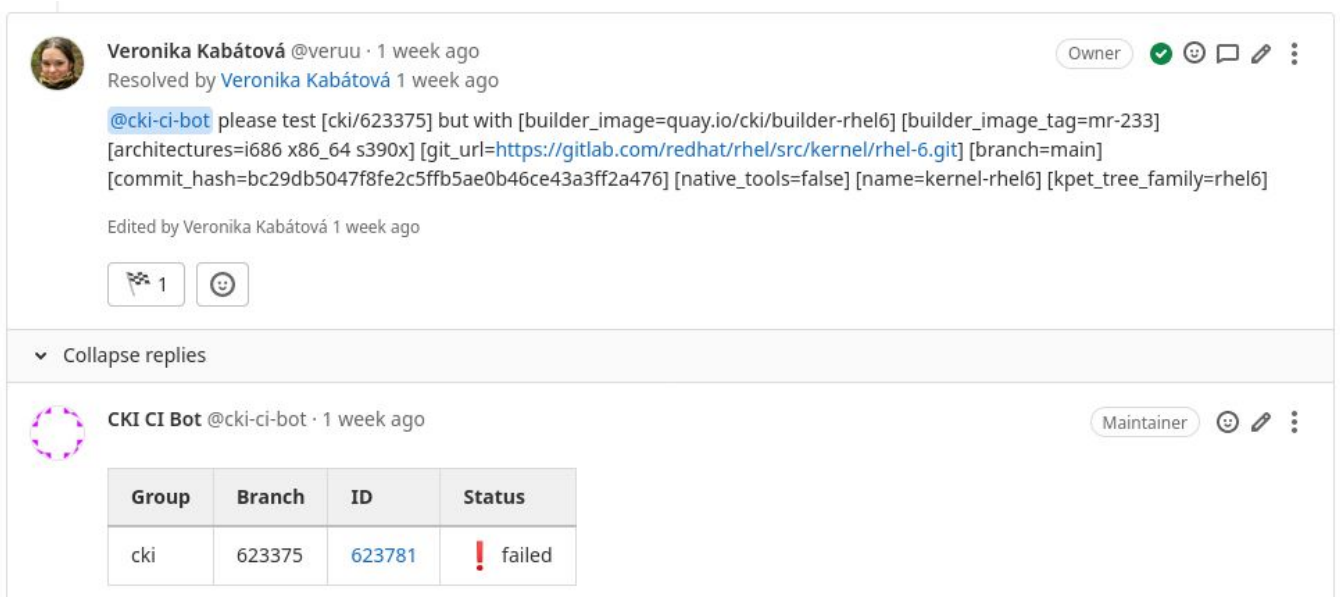
```
git clone https://gitlab.com/cki-project/kpet
python3 -m pip install -e .
kpet --help
```

- Tools are designed to work outside of the pipeline

Pipeline: canary deployments

Being nice and friendly to our future robot overlords

- 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



Red Hat

Pipeline: production deployments

Also scary

- New code gets automatically picked up after merging to main
- Rollback by git revert


Questions?
https://cki-project.org/
https://gitlab.com/cki-project/