GITLAB CI FOR DUMMIES

Dummy = Jon Mason
Dummy != Bruce Ashfield (debatable)

Dummies

Jon Mason

- Arm (very smart people)
- OpenEmbedded Board Member
- Co-maintainer of meta-arm layer

Bruce Ashfield

- Xilinx
- Maintainer of Yocto Project kernel, metavirtualization layer

Note:

You are not a dummy. I am a dummy. This took way longer that it should have, mostly because...I am a dummy. Learn from my mistakes to speed up your deployment and not be a dummy.



What is Gitlab?

Web based source code management (SCM), similar to github. As the name implies, it is used for git based source code.

Why Gitlab?

- Free
- Open Source
- Has robust features like wikipages and...CI/CD

What is this CI/CD?

Continuous integration (CI) is the practice of merging all developers' working copies to a shared mainline

Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time and, when releasing the software, without doing so manually

Put simply, it is a extremely frequent building and testing of software patches as part of the process of integrating them into a shared tree

Why is this useful for open source projects?

Maintainers of projects can get dozens (possibly hundreds) of patches daily. Identifying and removing faulty ones prior to releasing them to the public can avoid development issues with the wider developer community.

This process enables patches to be built and tested prior to integration, finding bugs BEFORE they break other developers of the shared code.

Drawbacks

 In a perfect world, we would have enough computing resources to compile and test every patch, weeding out the problematic ones as they arrive.

 Unfortunately, this isn't feasible, as each CI/CD run can take hours and dozens (perhaps hundreds) of patches arrive daily. So, they must be tested in batches. This increasing the difficulty of determining faulty patches and requiring more manual intervention.

For example, there were 97 patches pulled into poky between 05/10-05/17/21. If we compiled each one, for each qemu machine, and it only took an hour (which it takes way more), it would add up to 1067 (97 * 11 * 1). (note: 168 hours/week)

Gitlab CI

GitLab CI/CD is a tool built into GitLab for software development

Free (if you run everything on your own hardware, more about this later)

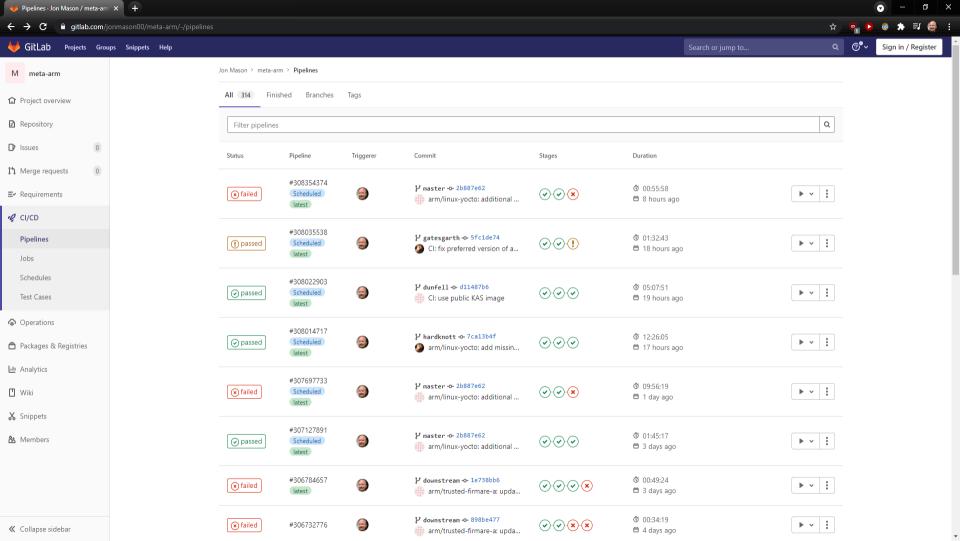
Alternatives to Gitlab CI

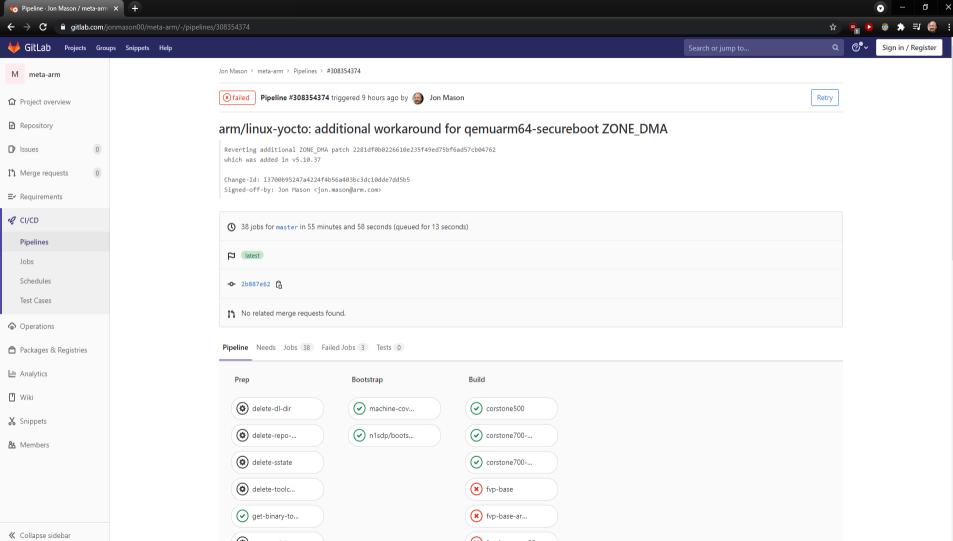
- Github workflow
- Jenkins and LAVA
- Many others

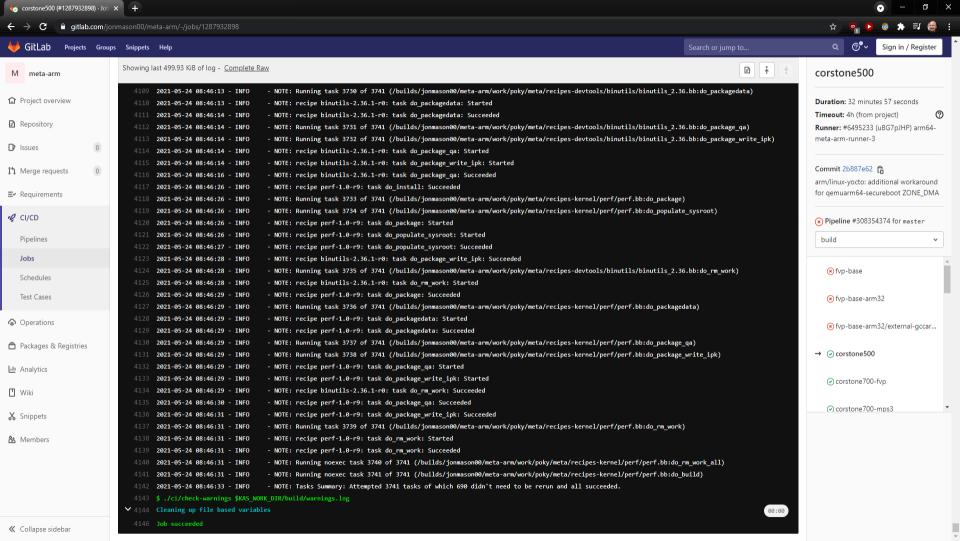
Current Users of Gitlab CI in YP/OE

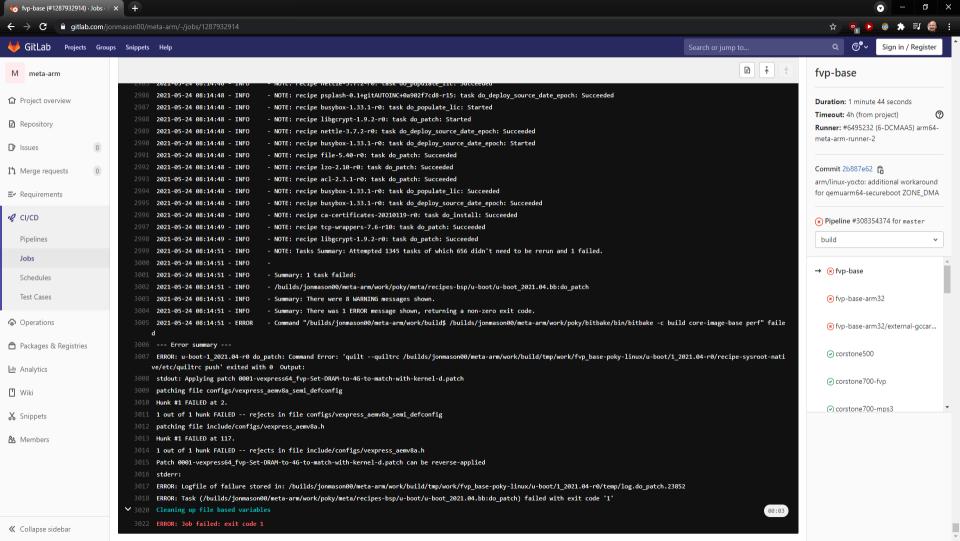
- Meta-arm
- Meta-security

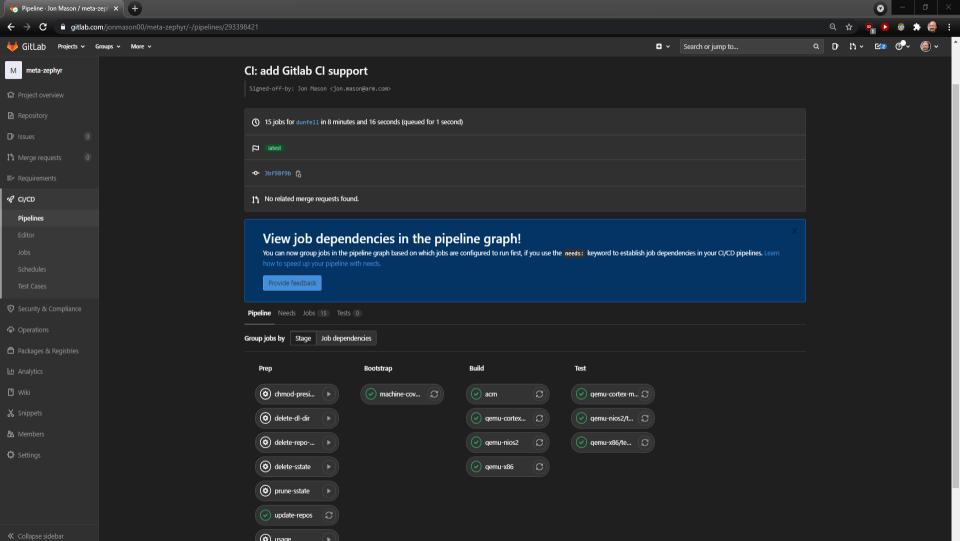
- WIP
 - Meta-zephyr (personally maintained)
 - Yocto Project kernel releases

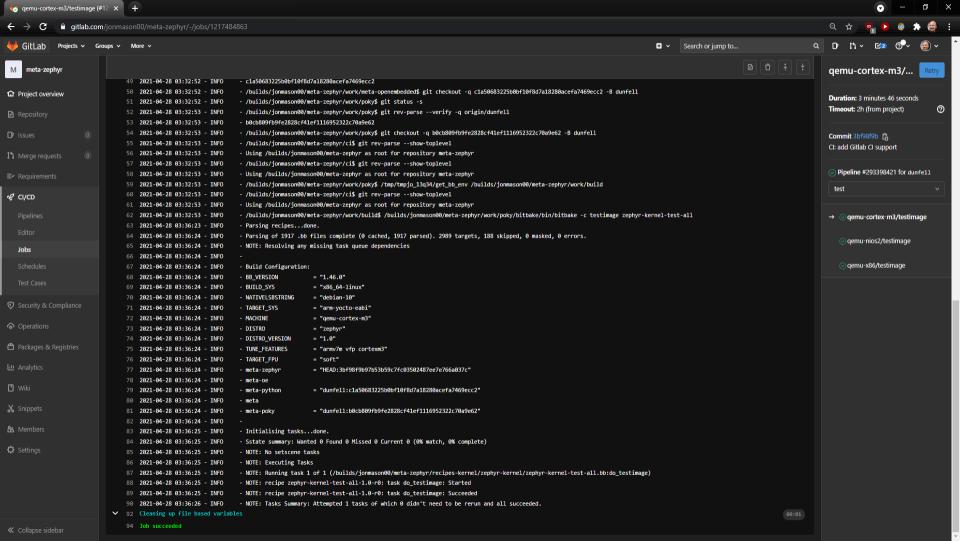












I'm sold, how do I set this up?

Gitlab Runner Gitlab Executor

Gitlab Runner

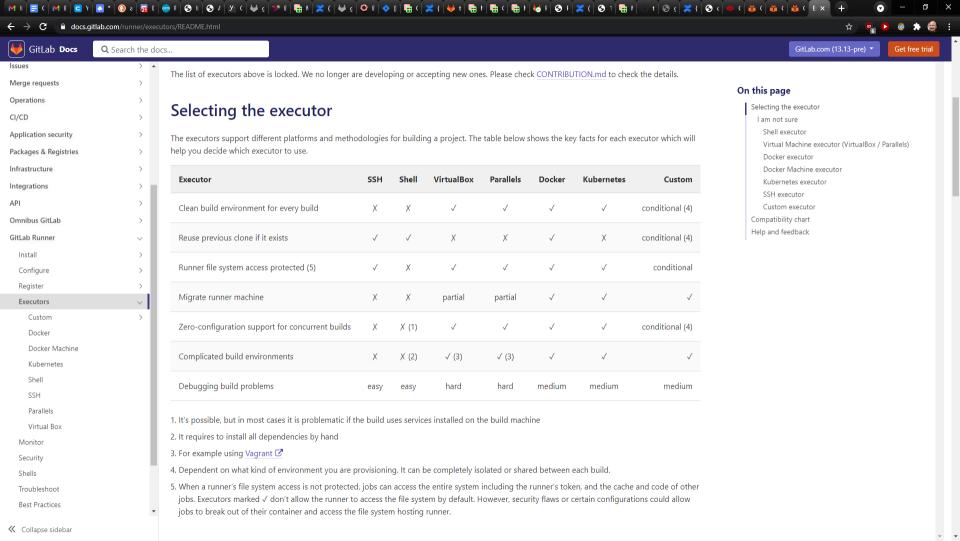
A runner is a lightweight, highly-scalable agent that picks up a CI job through the coordinator API of GitLab CI/CD, runs the job, and sends the result back to the GitLab instance.

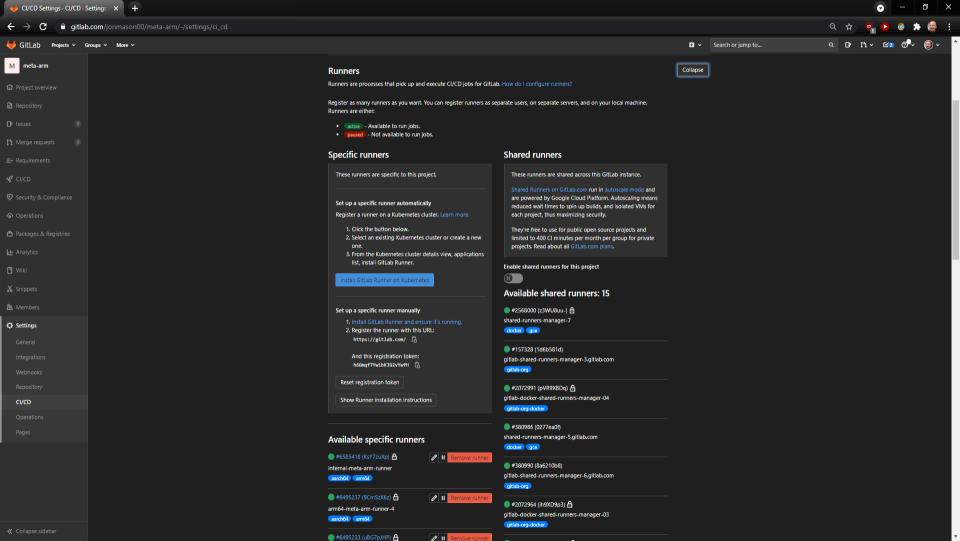
https://docs.gitlab.com/ee/ci/runners/README.html

Gitlab Executor

GitLab Runner implements a number of executors that can be used to run your builds
in different scenarios

- SSH
- Shell
- VirtualBox
- Parallels
- Docker
- Kubernetes
- Custom
- https://docs.gitlab.com/runner/executors/README.html





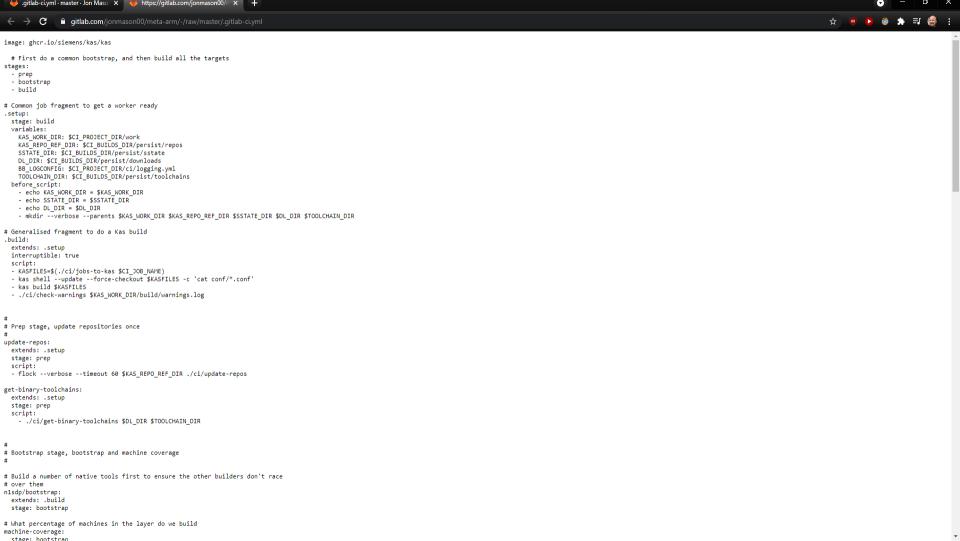
How to get Gitlab CI building and running Yocto/OE

Gitlab CI config file

KAS

Gitlab CI config file

Basically, a way to run shell scripts



KAS

Tool provides an easy mechanism to setup bitbake based projects

- clone and checkout bitbake layers
- create default bitbake settings (machine, arch, ...)
- launch minimal build environment, reducing risk of host contamination
- initiate bitbake build process

Configuration via a yml file



KAS Docker

 Docker image created by the KAS project with all of the dependencies to build a Yocto image, as well as KAS installed

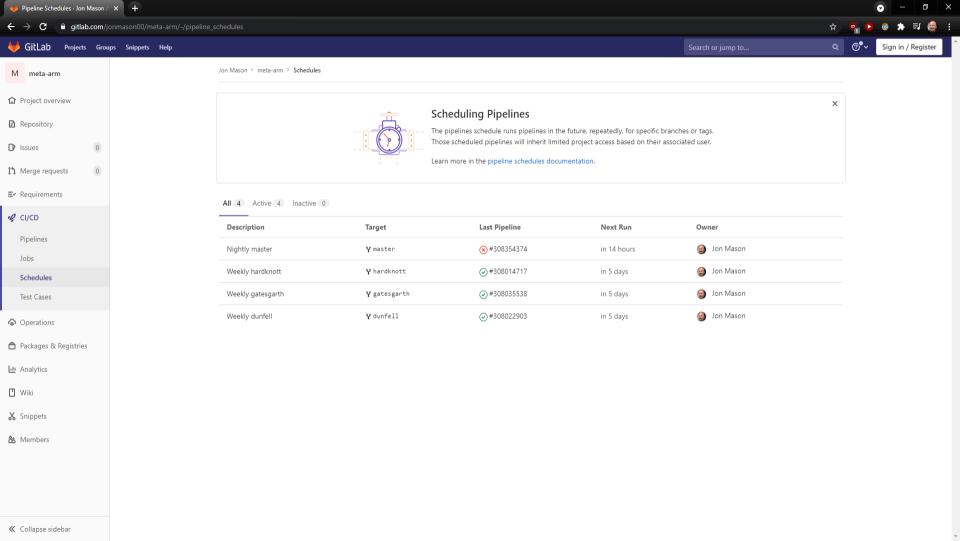
Benefits of Developing this way

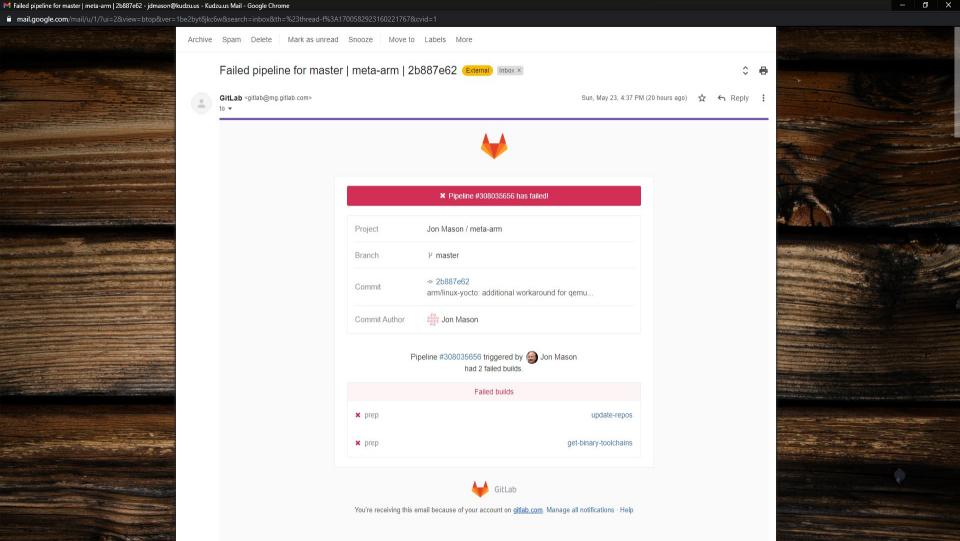
- Very reproducible builds
- Minimal environment and no dependency on installed packages
- Can use sstate and dl_dir

Cool features

Scheduling runs

Emails from failed/newly working runs





Debugging in Gitlab CI

Running docker locally to reproduce issue

jdm@athena:~\$ sudo docker run -it --name kas-test --volume /yocto/downloads:/builds/persist/downloads --volume /yocto/sstate-cache:/builds/persist/sstate --volume /yocto/repos:/builds/persist/repos --volume /yocto/toolchains:/builds/ persist/toolchains --privileged ghcr.io/siemens/kas/kas/bin/bash builder@ea0827034b68:~\$ git clone https://gitlab.com/jonmason00/meta-arm.git -b downstream

Cloning into 'meta-arm'...

remote: Enumerating objects: 579, done.

remote: Counting objects: 100% (579/579), done.

remote: Compressing objects: 100% (244/244), done.

remote: Total 7374 (delta 344), reused 529 (delta 310), pack-reused 6795

Receiving objects: 100% (7374/7374), 1.49 MiB | 7.77 MiB/s, done.

Resolving deltas: 100% (4204/4204), done. builder@ea0827034b68:~\$ cd meta-arm/

builder@ea0827034b68:~/meta-arm\$ SSTATE_DIR=/builds/persist/sstate DL_DIR=/builds/persist/downloads kas build

ci/gemuarm64-secureboot.vml:ci/testimage.vml

Potential Problems

Can't share sstate with non-container users using KAS

Difficult/impossible to access failing docker image

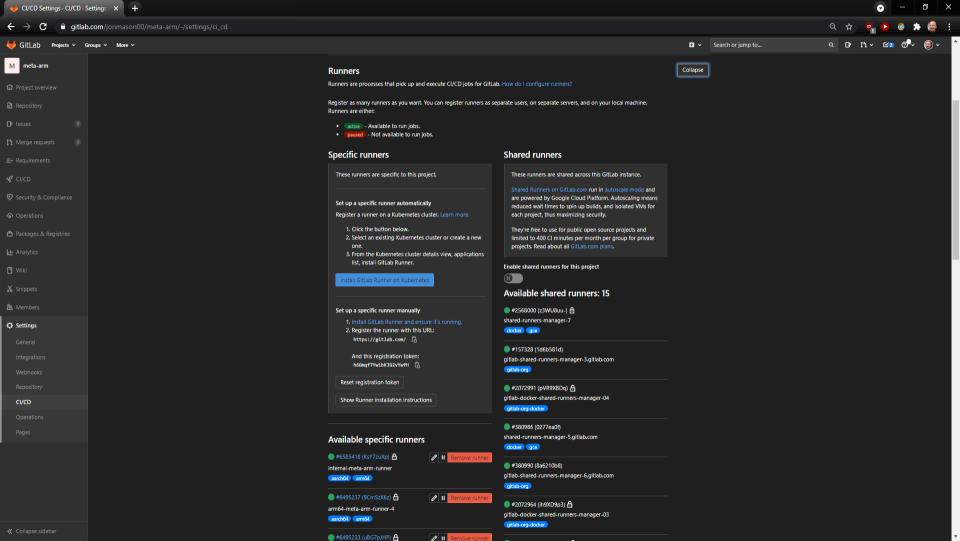
Improvements?

Running in tmpfs Runqemu working in non-privlidged container Optimal number of runners on a single host

THANKS

https://gitlab.com/jonmason00/meta-arm/-/pipelines

To see more info on Gitlab CI running on meta-arm



Gitlab Runner and executor docker from shell

- \$ sudo docker run -d --name gitlab-runner --restart always -v /var/run/docker.sock:/var/run/docker.sock gitlab/gitlab-runner:latest
- \$ sudo docker exec -it gitlab-runner gitlab-runner register --non-interactive --tag-list "aarch64,arm64" --run-untagged --url "https://gitlab.com/" --registration-token "h6Gmqf7YwibKJG2vYwfH" --executor "docker" --docker-image "ghcr.io/siemens/kas/kas" --description internal-meta-arm-runner