Imperial College London

RCS RESEARCH COMPUTING SERVICE
Gateway to Discovery

Research Software Engineering

Report: Nektar++ Phase I

Project details

Reference:	Nektar++/2019/1/1
Proposer:	Professor Spencer Sherwin, Department of Aeronautics (South Kensington)
Domain:	Computational Fluid Dynamics
Components:	Software engineering infrastructure

Introduction

Nektar++ is a sophisticated computational fluid dynamics codebase developed for multiple operating systems with an involved set of build dependencies. Current development relies on a Continuous Integration and Delivery (CI/CD) pipeline based on a Buildbot farm with test environments provided by virtual machines using on-premises compute infrastructure as execution hosts. Code hosting is provided by a GitLab instance that benefits from a degree of Buildbot customisation to improve workflow integration. Whilst the current system suffices to meet the needs of the Nektar++ it requires considerable investment of time and effort to maintain, particularly around the virtual machine infrastructure, and does not embrace modern “infrastructure-as-code” principles.

This report marks the completion of Phase I of this project. The goal was to undertake a comprehensive review of the current CI/CD State of the Art and propose a further body of work to improve the sustainability of Nektar++ development. Recent years have seen an explosion of mature and capable CI/CD products that provide a range of potentially viable alternatives. This review is specifically targeted at the requirements of Nektar++ as laid out fully in the assessment criteria below.

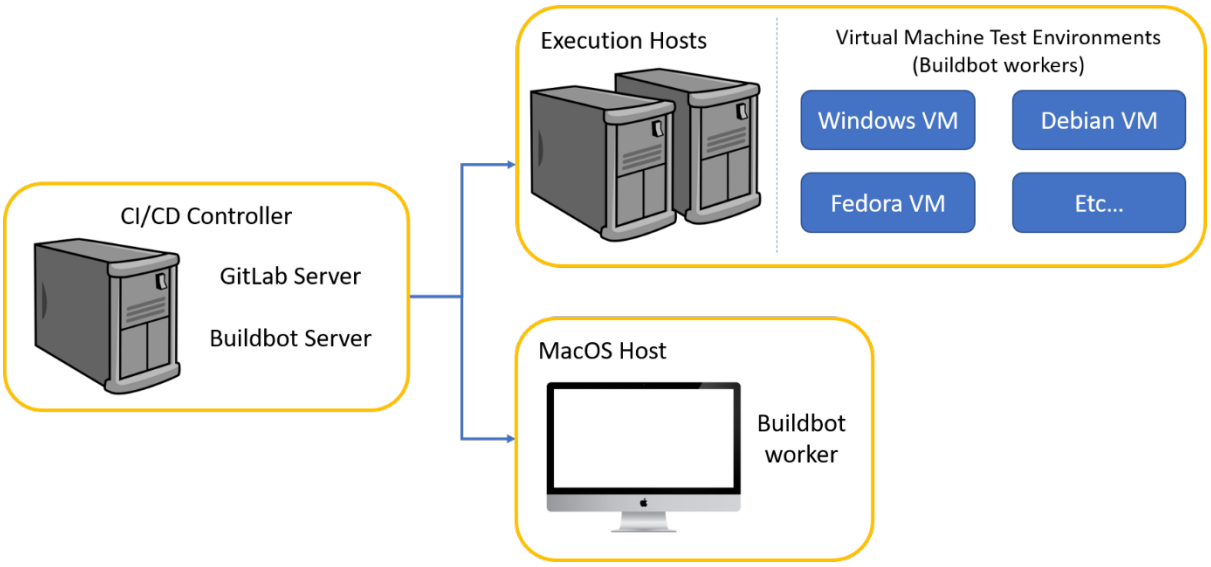

Figure 1. Diagrammatic representation of current Nektar++ CI/CD system

Objectives

The purpose of this review is to propose a CI/CD solution that meets the below objectives:

- Comprehensive
  - The proposed solution will completely recreate the required functionality of the existing implementation (it may be an extension of the existing implementation).
- Reliable and easily maintainable
  - Achieve a dramatic reduction in the time required to maintain project infrastructure allowing developers to focus on development.
  - Adopt an “infrastructure as code approach” that will facilitate automation and reproducibility. The relevant configuration of CI/CD can be stored under version control to track changes or empower debugging.
- Cross-platform
  - Support testing on Linux (multiple distributions), Windows and Mac.
  - Automate the generation and publication to the project web presence of any required binaries or other artefacts for distribution on the target platforms.
- Extensible
  - Support insertion of additional steps into the CI pipeline, e.g. benchmarking
  - Ability to work with multiple programming languages as required

Methodology

Target Platforms

For this review four target platforms were considered:

- Buildbot - The existing platform
- GitLab CI - The GitLab native CI service
- Azure DevOps - Microsoft hosted CI service
- Anvil - STFC/EPSRC hosted CI service based on Jenkins

These platforms were considered to provide a range covering hosted online services (Azure DevOps), local open source solutions (Buildbot, GitLab CI) and a specialised system for research software (Anvil).

It was also found necessary to distinguish between what we shall term front and back end platforms. Here a front end is taken to be the system used to design and orchestrate a CI workflow. The above listed platforms constitute the considered front ends. The back end meanwhile is the infrastructure used to execute a CI workflow. The below back end options were examined:

- On-premise virtual machines - The current back end setup
- On-premise docker containers - A lighter weight replacement for virtual machines
- GitLab.com Shared Runners - The cloud CI back end for GitLab
- Azure DevOps - The cloud CI back end for Azure DevOps
- Anvil - The hosted back end provided by Anvil

GitLab CI, Azure DevOps and Anvil each provide their own native execution back ends, but GitLab and Azure are additionally capable of using on-premise facilities. Buildbot does not provide its own back end. The full compatibility matrix below applies between different front and back ends.

	Buildbot	GitLab CI	Azure DevOps	Anvil
VMs	✔	✔	✔	✘
Docker	✔	✔	✔	✘
GitLab.com Shared Runners	✘	✔	✘	✘
Azure DevOps	✘	✘	✔	✘
Anvil	✘	✘	✘	✔

Assessment Criteria

The assessment criteria below are designed to comprehensively express the CI/CD requirements for Nektar++ and were agreed in collaboration with the Nektar++ development team. Criteria were divided into those considered essential and others that are desirable. Broadly speaking, the essential criteria cover the functionality of the existing CI system whilst desirable criteria cover novel functionality.

Essential Criteria

- Agents must be capable of heterogeneous work loads
    - Build from source with minimal dependencies on all platforms
    - Build from source with support for MPI, FFTW, VTK, BLAS/LAPACK, ARPACK, Scotch, PETSC, HDF5, Meshgen, CCM and Python on Linux and MacOS
    - Build from source with support for BLAS/LAPACK on Windows
    - Build tutorial materials on at least one platform
    - Build documentation on at least one platform
    - Build developer documentation on at least one platform
- Multi-platform workers
    - Ubuntu (supported LTS versions), Debian (current stable and testing distributions), CentOS (latest version), Fedora (latest version)
    - 32-bit versions of one or two Linux platforms
    - Windows 10
    - MacOS 10.11 or later
- Gitlab Integration
    - Automatic triggering of builds by new commits to master, release branches and merge requests
    - Build status reported in Gitlab
    - Ignore CI for commits to merge requests whilst in draft state
- Sustainability (operational complexity and maintenance burden)
    - Minimise time and complexity of setting up new workers
    - Minimise maintenance of CI infrastructure (e.g. infrastructure upgrades)
- Ease of use
    - Observability i.e. clear indication of the causes of any build failure
    - Interactive debugging of failures for each platform, either via login to build server or (preferably) by access to an exact replica of the relevant environment
    - Simple presentation of build matrix status (e.g. dashboard view)
- Infrastructure
    - Strong preference for a self-hosted open source solution, making use of existing infrastructure
    - Resource footprint of workers must allow ~10 concurrent builds
- Cost
    - Strong preference to avoid significant monthly cost commitments

Desirable Criteria

- Infrastructure-as-code
    - All CI configuration as files checked into source repository
- Advanced Gitlab integration
    - Ability to force builds on all workers from Gitlab
    - Ability to trigger runs on only some workers
    - Improved interaction with running CI workloads (visibility, cancellation, etc)
- Continuous deployment for QA and to produce binaries (see “default” and “full”) and other artefacts (e.g. “userguide” and “doxygen”)
    - Build rpms, debs, tarballs and docker images
    - Specify criteria for these to be published (e.g. tagged releases)

Scoring System

Each CI system is scored between zero and three for compliance with each criterion. Three indicates that a criterion is fully satisfied whilst zero corresponds to a complete failure to be met. Where a criterion is applicable to both the front and back end the points are sub-divided between them.

Results

Full details of the scores for all target platforms are available in Appendices 1 and 2 with brief explanations provided. Presented here is a high-level discussion comparing the different platforms followed by a recommended package of work to implement the preferred solution.

Front End Platform

The strongest discriminator for different front ends was found to be integration with GitLab for both the essential and desirable criteria. The code repository for Nektar++ is provided by a self-hosted GitLab instance so tight integration with the CI/CD system is essential to provide a smooth workflow for day-to-day activities. This favoured the GitLab CI front end with its native level of integration whilst penalising Azure DevOps most strongly.

Secondary discriminators where provided by the sustainability and infrastructure-as-code (IAC) criteria. Buildbot was penalised for providing additional self-hosted infrastructure to maintain. GitLab CI and Azure DevOps are the only systems that fully allow IAC configuration through the .gitlab-ci.yml file and azure-pipelines.yml file respectively.

Otherwise all of platforms were found capable of meeting the technical requirements for Nektar++ and provide similar user experiences. One feature not reflected in the scoring but worthy of an honourable mention is the strong support of Buildbot for continuous delivery of rpm and deb formats (provided by the RpmBuild and DebBuild classes respectively). Whilst not beyond the technical capabilities of the other platforms this support facilitates easy creation of these binary objects in Buildbot.

The below table summarises the scores of the different front ends against the assessment criteria. See Appendix I for full scoring information. Platforms are listed by highest to lowest scores, and hence most to least recommended.

	Essential Criteria	Desirable Criteria	Total
GitLab CI	9	6	15
Buildbot	8	4	12
Anvil	8	3	11
Azure DevOps	7	3	10

Back End

A variety of factors impact on the relative suitability of the various backends, see Appendix II for full details of scoring. We provide here a summary of the scores along some brief comments for each of the platforms.

	Essential Criteria	Desirable Criteria	Total
On-premise Virtual Machines	8	0	8
On-premise Docker	10	2	12
GitLab CI	8	2	10
Azure DevOps	9	2	11
Anvil	8	1	9

On-premise Virtual Machines

The back end used by the current CI/CD system. Whilst satisfying well the multi-platform, cost and infrastructure criteria this approach scores the lowest overall, primarily for the reasons discussed in the introduction. The need to setup and maintain the different test environments involves a large amount of manual work that impacts the sustainability of this approach and precludes the benefits associated with an infrastructure-as-code strategy.

On-premise Docker

As an alternative on-premise solution, Docker retains the strengths of using virtual machines but overcomes many of disadvantages allowing it to become the highest scoring backend. Compared to virtual machines:

- Less time and effort required to setup and maintain individual Linux environments.
- Increased job throughput on existing infrastructure by:
    - Reducing resource footprint required by each Linux environment
    - Memory and CPU resources not being consumed by idle virtual Machines
    - Removing virtualisation overheads
- Fully reproducible and portable environments for debugging – all Linux environments can be recreated from a Dockerfile on any system.

These improvements in sustainability and support for infrastructure-as-code make Docker the highest scoring back end considered.

It should be noted that use of Docker is only suitable for Linux environments. This approach would therefore need to be supplemented with a Windows virtual machine and a separate MacOS host. Combined with the need to maintain local compute infrastructure this approach falls short of level of sustainability that could be achieved by a cloud based back end however we believe it most closely meets the criteria as laid out for this project.

GitLab.com Shared Runners

This cloud hosted infrastructure provided by GitLab exclusively provides Linux hosts with support for Docker. In general, this platform would provide a similar experience to on-premise Docker. The use of the shared runners would need to be supplemented with local Windows and MacOS hosts impacting sustainability by retaining the need to maintain some local infrastructure. The main reason this platform scores joint last however is the recurrent cost structure required to obtain comparable computational power to the current CI/CD system.

Azure DevOps

This platform distinguishes itself by being the only cloud solution that provides native support for all required operating systems. For open source projects Azure DevOps provides free resources for up to 10 instances running in parallel. This rivals the level of compute currently provided by local infrastructure whilst completely removing any maintenance burden. This combination of factors provides Azure with the second highest score for any backend. The only notable disadvantage is the lack of interactive debugging for Windows and MacOS impacting ease of use. It should also be noted that there is no guarantee that the currently provided free resources will be maintained in the future potentially leading to unexpected costs further down the line.

Anvil

As joint worst scoring backend, Anvil is hampered primarily by its lack of support for the required operating systems. Despite being one of only two platforms offering a native MacOS host, the limited range of Linux hosts and lack of support for Windows are severe problems. Unlike other platforms there is no suggestion that Anvil can be supplemented with local hosts creating an arguably fatal shortcoming. Whilst Anvil does provide some unique capabilities such as running on HPC infrastructure these did not come to the fore during this review.

Recommendation

Based on the preceding analysis we recommend transitioning the current combination of Buildbot with on-premise virtual machines to GitLab CI supported by on-premise Docker. Such a system fully recreates the functionality of the current system whilst bringing numerous advantages. Moving to the GitLab CI front end will provide the best possible integration with the Nektar++ code repository, remove the requirement to maintain a separate Buildbot installation and allow an infrastructure-as-code approach to the CI/CD workflow configuration. Using a Docker based backend will drastically improve the sustainability of the system by reducing the time required to setup and maintain different test environments, improve throughput on the available compute infrastructure and allow an infrastructure-as-code approach to the Linux testing environment.

As such this system will fully meet the objectives of this project as laid out in this document. Once implemented the above changes will free the Nektar++ development team to reduce time spent maintaining their software engineering infrastructure and focus on code development supporting core research activities.

Technical Proposal

We here layout the technical detail of the proposed system. What follows is designed as a starting point for further discussion with project partners.

Front End

- An implementation of the existing CI workflow in GitLab CI. Configuration of GitLab CI is handled via a .gitlab-ci.yml file stored in the root directory of the source code repository. This brings the CI workflow configuration, previously in the form of a separate Buildbot configuration file, under version control.
- Conditional test job execution based on code changes. This GitLab specific functionality would allow, for example, to skip testing for trivial edits to a changelog or readme file, or to only execute resource intensive jobs when relevant files are changed.

Back End

- Linux test environments provided by Docker images
    - In practice these are defined by a set of Dockerfiles checked into the Nektar++ code base.
- Local compute infrastructure configured for Docker test environments
    - Compute hosts configured for native Docker execution through deployment of GitLab Runner, the application that communicates with the GitLab server to deploy CI workloads.
    -
- Windows virtual machine and MacOS host configured for GitLab CI through deployment of GitLab Runner.

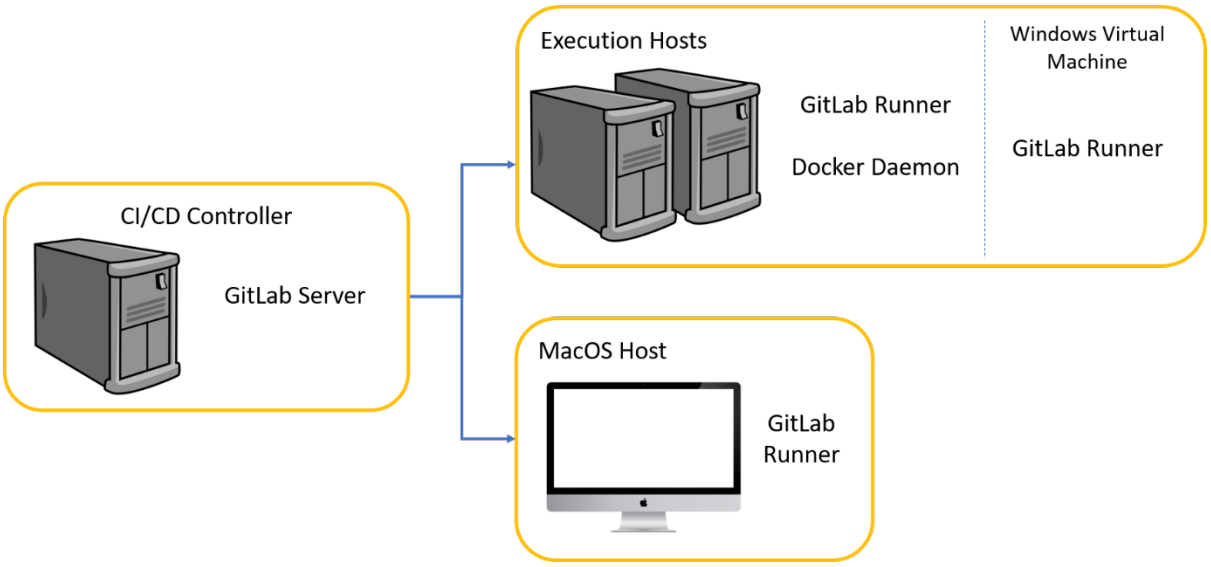

Figure 2. Diagrammatic representation of proposed Nektar++ CI/CD solution

Further Work

Whilst we believe it to be beyond the scope of this project, future work could be carried to examine the feasibility of configuring the local execution hosts as a Kubernetes cluster. Over and above basic Docker, this will allow dynamic placement and load balancing of CI workloads on available hardware, as well as providing simple options to add burst capacity to meet periods of expected high demand. The scheduling options of Kubernetes will also provide a simple mechanism to explore optimisation of workload throughput on existing hardware.

Despite the above benefits Kubernetes is a still a relatively new, fast developing and complex technology. For this reason we do feel able to recommend it at this time however with time it may become a more practical option worth exploring, particularly if the College offers a centrally hosted facility.

Work Plan

1. Deploy Proof of Principle Implementation – 1 week

A scaled down version of the final system designed to test key aspects for technical impediments. Greatest value will be obtained by deploying this directly on the relevant compute structure, but it should be able to sit alongside the existing system with minimal disruption. The system will comprise:

- A .gitlab-ci.yml file configured with a single job pipeline - compilation -> testing -> deployment
- GitLab Runner installed on one execution host
- A single Docker image for a suitable environment capable of building Nektar++ with minimal dependencies

This work will require administrative access to the relevant Nektar++ CI/CD infrastructure and may cause temporary disruption to the current system. Involvement of project partners will be needed to provide timely responses to technical queries about the infrastructure as required.

2. Develop Docker Images – 2 weeks

A set of images covering the full range of environments described in the assessment criteria. This work will benefit from ongoing efforts to containerise Nektar++, however to our knowledge no Docker image has yet been developed to build and test Nektar++ with a full set of dependencies.

This work can be carried without disruption to the existing system. Project partners will be asked to provide a set of build instructions for each platform to be translated into a Dockerfile but other required involvement should be minimal.

3. Prototype GitLab CI Configuration – 1 week

To avoid disruption to Nektar++ development the .gitlib-ci.yml file will be prototyped independently using a separate fork of the code base. This will allow the majority of the features of the CI/CD workflow to be developed but will need to be followed by further refinement during deployment.

This work can be carried out without disruption to the existing system. Access to the existing Buildbot configuration to act as a reference for implementation will be required. Involvement of project partners may be needed to provide timely responses to questions regarding desired behaviours of the system.

4. Deploy Final System – 1 week

This should simply be a full-scale version of the proof of principle deployment. Final configuration changes to the back end infrastructure and GitLab CI configuration will be made at this time. Installation of GitLab Runner on Windows and MacOS hosts.

At this point the existing CI/CD system will be effectively decommissioned without a functional replacement with consequences for Nektar++ development. In total CI/CD system may be unavailable for a period of up to two weeks.

Summary

In this report we have carried out a review of the state of currently available CI/CD platforms evaluating their suitability of use with Nektar++. Based on this we propose an overhaul of the current Nektar++ CI/CD system, replacing the current combination of Buildbot backed by virtual machines with the GitLab CI system with testing environments provided by Docker images.

This change will greatly improve the sustainability and reproducibility of the software engineering infrastructure of Nektar++, allowing the developers to concentrate their efforts on core research activities. The resulting system will offer improved throughput of CI/CD workloads and be able to intelligently determine appropriate testing from changes to the code base, providing a more responsive service overall.

We present a technical design for the proposed CI/CD solution and an implementation work plan for further discussion with the Nektar++ team. We estimate that implementing this system will require around 5 weeks FTE, with minimal support required from project partners, but only 2 weeks of possible disruption to Nektar++ development.

Appendix 1 - Front End Scoring

Essential Criteria:

	Heterogeneous work loads		Multi-platform	Gitlab Integration		Sustainability		Ease of Use		Infrastructure	Cost	Total
Buildbot	Defined through multiple Builders	**3**	N/A	Satisfied by current setup	**3**	Maintenance of Buildbot infrastructure needed.	**0**	Good observability. Informative dashboard view.	**2**	N/A	N/A	8
Gitlab CI	Achieved through definition of dependencies between jobs	**3**	N/A	Native Gitlab so naturally provides good integration. Ignoring WIP MR's possible through except: variables: - $CI_MR_TITLE =~ /^WIP/	**3**	Requires no additional maintenance on top of that already required for local GitLab installation.	**1**	As above	**2**	N/A	N/A	9
Azure Devops	Achieved through definition of dependencies between jobs	**3**	N/A	Limited. No trigger on MRs without third party tools.	**1**	Using hosted services completely removes maintenance burden.	**1**	As above	**2**	N/A	N/A	7
Anvil	Can be defined through multiple Pipelines	**3**	N/A	Possible through merge event hooks.	**2**	As above	**1**	Good observability. Informative dashboard view via Build monitor plugin	**2**	N/A	N/A	8

Desirable Criteria

	Infrastructure as Code		Advanced Gitlab Integration		Continuous Delivery		Total
Buildbot	Buildbot configuration does not readily allows this.	0	May be possible to improve integration through further customisation of GitLabStatusPush and ChangeHooks. Changes to GitLab interface may not be possible.	2	Strong native support for building and validating debs and rpms via RpmBuild and DebBuild. Docker and tarballs supported through custom shellCommand build steps. Conditional publication may be possible through use of doStepIf argument of a BuildStep.	2	4
Gitlab CI	Fully supported by .gitlab-ci.yml file	1	Native Gitlab so naturally provides the best possible	3	Conditional publication of artefacts supported by 'rules' parameter and CI_COMMIT_TAG environment variable. No native support for rpm/deb but can be implemented manually.	2	6
Azure Devops	Fully supported by azure-pipelines.yml file	1	Lack of even basic GitLab integration suggests this is unlikely to be possible	0	Conditional publication of artefacts supported by 'conditions' for a job step. No native support for rpm/deb but can be implemented manually.	2	3
Anvil	Not supported for high level Jenkins configuration.	0	A similar level of integration to BuildBot should be possible. Open source gitlab integration plugin is customisable however not clear if supported by Anvil.	1	Conditional publication of artefacts supported by when directive. No native support for rpm/deb but can be implemented manually.	2	3

Appendix 2 - Back End Scoring

Essential Criteria

	Heterogeneous work loads	Multi-platform		Gitlab Integration	Sustainability		Ease of Use		Infrastructure		Cost		Total
On-prem VMs		Windows and Linux (supplement with OSX host). 32-bit hosts supported.	2		Considerable time needed to create new workers. Maintenance of host machines.	0	Interactive debugging through direct access to VMs.	1	On-premises hardware.	2	Hardware capital expenditure.	3	8
On-prem Docker		Linux (supplement with Windows VM, OSX host). 32-bit containers possible.	2		Minimal setup time for new linux environments. Maintenance of host machines.	1	Interactive debugging through local docker containers.	1	On-premises hardware.	3	Hardware capital expenditure.	3	10
Gitlab CI		Linux. 32-bit containers possible. (supplement with Windows VM, OSX host)	2		Minimal setup time for all platforms. Hosted service, no maintenance required.	2	As above.		Externally hosted hardware.	2	Subscription payment service for required bandwidth.	1	8
Azure Devops		Windows, Linux and MacOS	3		Minimal setup time for all platforms. Hosted service, no maintenance required.	2	No interactive debugging for Windows or MacOS hosts	0	Externally hosted hardware.	2	Current resources for open source projects have unlimited minutes for 10 parallel builders.	2	9
Anvil		Limited Linux plaftorms and MacOS (windows may be possible but undocumented)	1		Minimal setup time for all platforms. Hosted service, no maintenance required.	2	No interactive debugging.	0	Externally hosted hardware.	2	Free for use.	3	8

Desirable Criteria

	Infrastructure as Code		Advanced Gitlab Integration	Continuous Delivery	Total
Buildbot	Precluded by manual VM setup.	0	N/A	N/A	0
Gitlab CI	Supported via Dockerfile	2	N/A	N/A	2
Azure Devops	As above	2	N/A	N/A	2
Anvil	Fully supported by .gitlab-ci.yml file	2	N/A	N/A	1