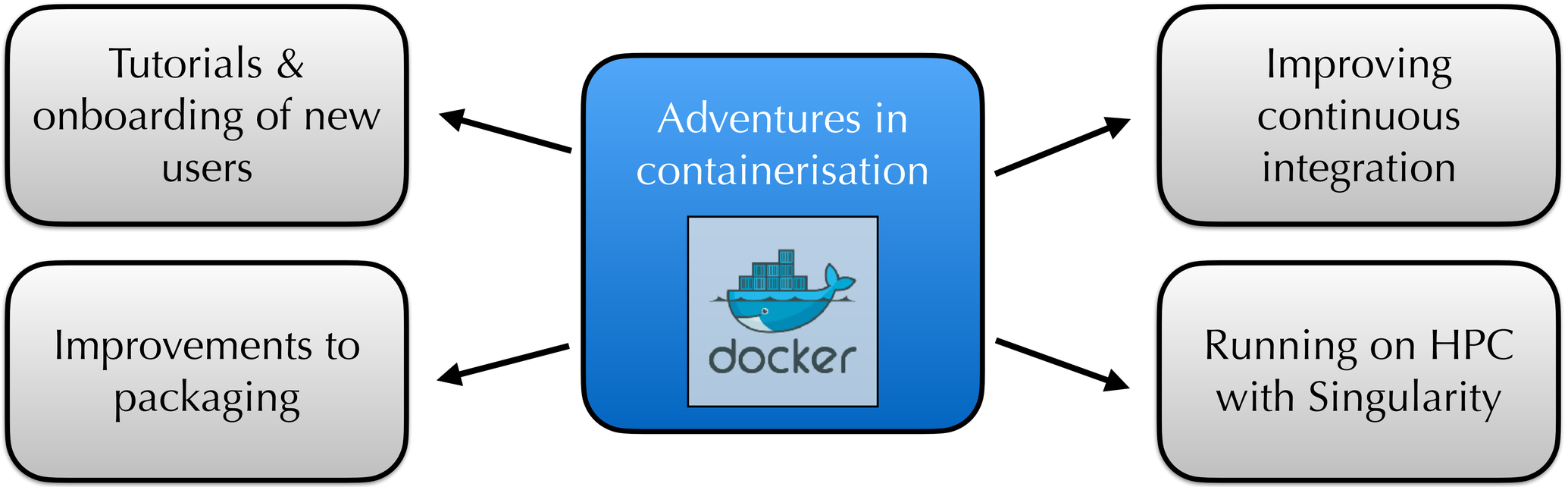
New and improved development practices in Nektar++

David Moxey
Department of Engineering, King's College London

Nektar++ Workshop 2021
13th December 2021

Overview


Tutorials & onboarding of new users
Improvements to packaging
Adventures in containerisation
docker
Improving continuous integration
Running on HPC with Singularity

What are containers?

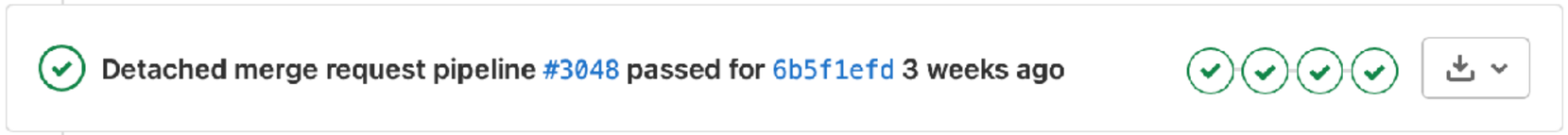
- Essentially a mini virtual machine.
    - **Images:** contains all of the software & its dependencies: a sort of snapshot or blueprint. These are immutable once built.
    - **Containers:** run on top of images without altering them, are where the program actually runs.
- Their main advantage is that they are easily distributable and have very low startup cost compared to a normal VM: you can pull an image and 'go'.

Why are we interested in containers?

- Nektar++ is a complex piece of software with quite a few dependencies; often a struggle to get compiled and running, particularly for new users.
  - Containers are obviously a neat way to package & ship all of those.
  - However containers offer also a way to provide a 'clean' environment which is really useful for testing & packaging.
- Pushing towards a development practice of not only testing commits but building releases alongside them: i.e. continuous delivery.

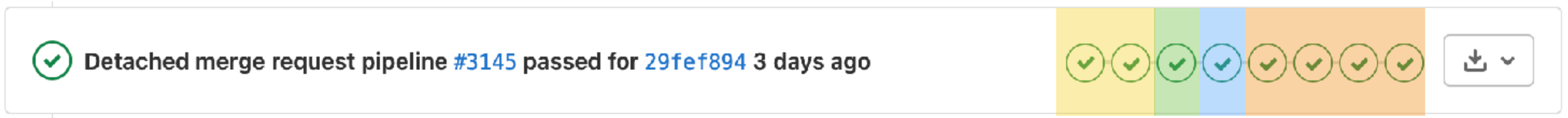
Containers and CI

- We use CI to build and test the code at every merge request to help keep the code stable and test for regressions.



- Previously used buildbot based on traditional VMs. Quite a few disadvantages to this that we discovered over time:
  - maintaining many VMs is a pain;
  - not easy to add new infrastructure to allow for faster builds;
  - when things went wrong, very hard to get into the environment to debug.

Switching to containers & gitlab CI



Build environments
i.e. Ubuntu, Debian etc + deps

Compile & test code
done in single step to avoid caching issues

Compiler warnings
based on output of previous stage

Build & test docker images

- `nektarpp/nektar`: just want to run Nektar++
- `nektarpp/nektar-dev`: development environment
- `nektarpp/nektar-workbook`: Jupyter notebook

You can grab either latest master with `latest`, or tagged versions (`v5.1.0`)

When pipelines go wrong

- gitlab is configured to build a new docker image when the pipeline fails.
  - Gives access to the entire build environment at the time of failure.
  - You can use this to debug compiler & test errors.
  - Images are pushed to tags called `pipeline_id_runner_name`

```
# log into the docker container registry
$ docker login gitlab.nektar.info:4567

# then pull a docker image and get a shell inside it
$ docker run -it gitlab.nektar.info:4567/nektar/nektar:pipeline_3097_bionic_full /bin/bash

# now you can run command to e.g. see a build error
nektar@beee5eb9f130:~/nektar$ cd build && make install
```

Running on HPC

- Our docker images are built with pretty much all the bells and whistles, including MPI for parallel execution.
- Increasingly HPC resources are looking into containerisation since compilation of codes is often an even larger problem in this environment.
- Singularity is an alternative containerisation that's increasingly used in HPC due to increased security, but can use docker images

```
$ module load apps/singularity                  # Load singularity in HPC environment
$ singularity pull docker://nektarpp/nektar     # Pull docker image
$ srun -n 4 -p test --pty /bin/bash             # Launch a job on 4 nodes (16 CPUs total)
$ singularity shell ./nektar_latest.sif         # Grab a shell on this image
$ mpirun -n 16 IncNavierStokesSolver s.xml      # Run Navier-Stokes solver
```

Packaging improvements

Now also using CI to automate build of binary packages when we tag new release

Pre-built packages for:

- Ubuntu/Debian (.deb)
- CentOS 7 (.rpm)

Details on website

MacPorts

```
# Vanilla serial version that
# has binary package
$ sudo port install nektarpp

# Spicy MPI + Python version,
# requires compilation
$ sudo port install nektarpp \
    +openmpi +python39
```

Homebrew

```
# Has most bells/whistles, but
# not yet in main trunk. Some
# binaries available for
# macOS 10.15, 11.0 using GitHub
$ brew tap mdave/nektar
$ brew install nektar
```

Finally: training and tutorials

Our training requirements

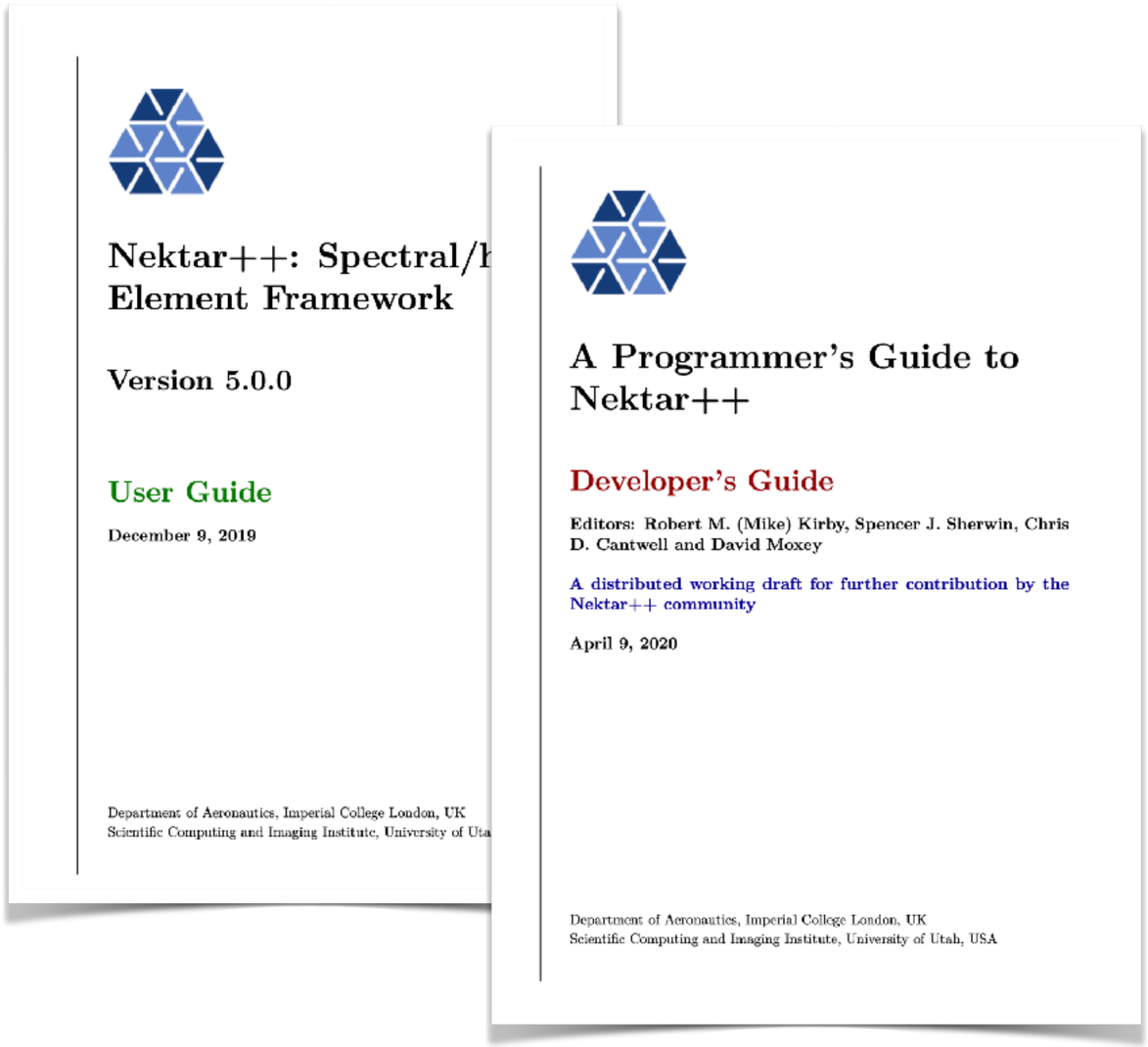
undergraduate, postgraduate or PDRA projects

conference workshops or training sessions

Some common challenges:

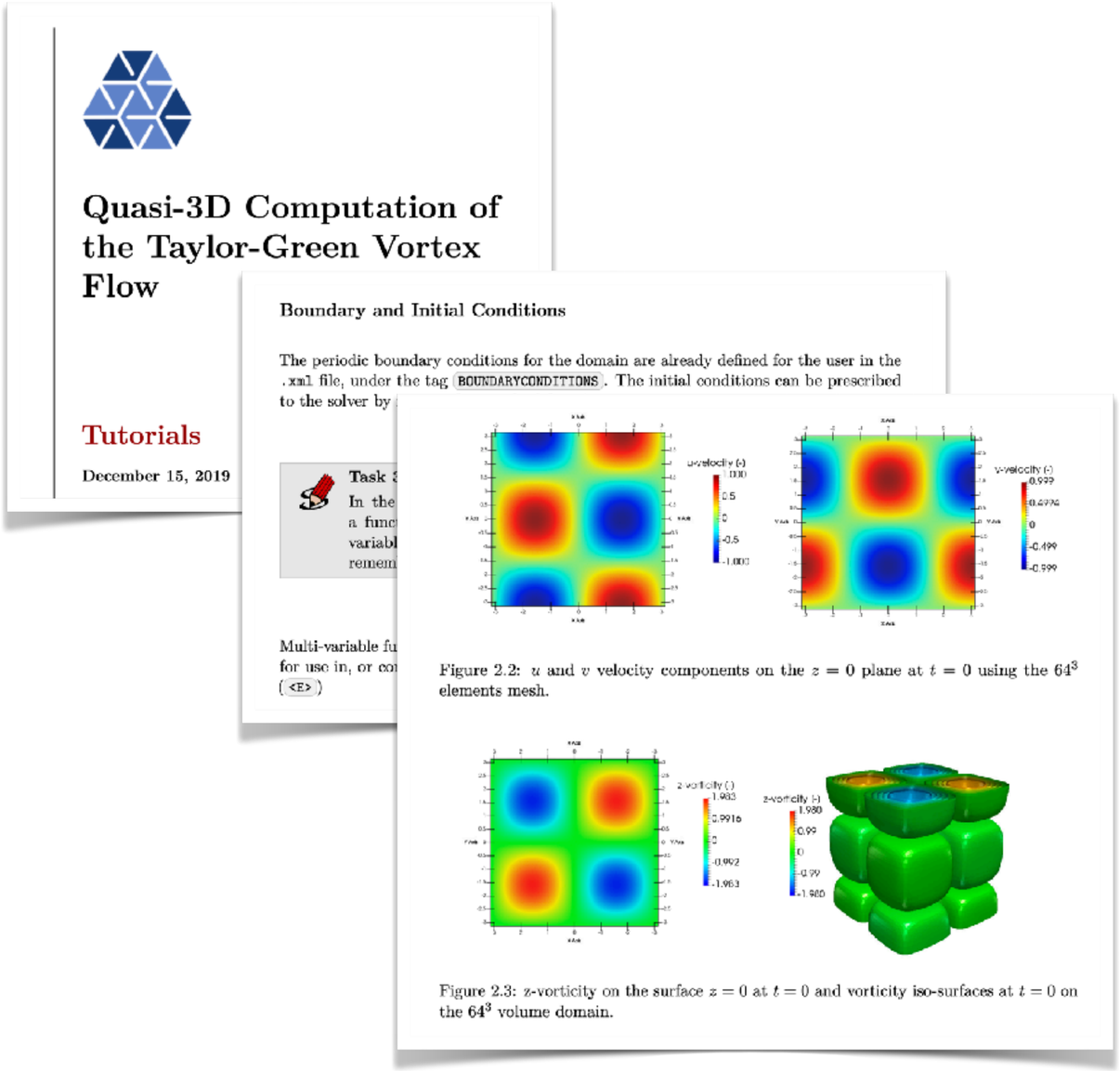
- Onboarding into the environment: a complex C++ framework with advanced numerics.
- Developing resources for a wide range of user abilities and experience.
- Taking into account requirements: some want to use the code, others want to develop
- Handling vastly heterogeneous computational environments (OS, compilers, etc).
- Doing all of this remotely!

First approach: user/developer guides



- First attempt at getting good documentation was to develop a user guide (developer guide also available but still a WIP!)
- A good way to give reference for the capabilities of the framework...
- ...but maybe too unwieldy to give new users a way to access the code.

Tutorials



- Next, developed a series of tutorials:
  - ➡ documents key solvers;
  - ➡ numerical concepts;
  - ➡ pre- and post-processing.
- Come with a set of incomplete files to work through, and a set of solution files to show final configuration.
- Available as a PDF or on the website.

More recent developments

Jupyter notebook, contains Python interface as well as core solvers, utilities for pre- and post-processing

```
docker pull nektarpp/nektar-workbook
```

- Can be combined with cloud resources/Kubernetes to deliver flexible resources for workshops.
- We'll be demoing this in morning session on Tues/Weds

What we are working towards

increasing complexity

application area →

Onboarding

Increased interactivity
(Jupyter notebooks)

Theory
Intro to SEM
Numerical integration

Navier-Stokes
basics: incompressible
basics: compressible
3DH1D simulations

Pre-post processing
NekMesh basics
FieldConvert basics

Terminal/HPC usage

More traditional tutorials
(command line)

Theory
Higher dimensions

Navier-Stokes
turbulence simulations
stability
bioflows
finite strip modelling

Pre-post processing
Mesh generation from CAD
Advanced postprocessing

Developer

How to develop codes
based around Nektar++

Compiling
How to compile
Using docker dev. env.

Basics
Creating a forcing term
Creating a solver

Post-processing
Creating a filter/forcing
Using the Python API

Conclusions

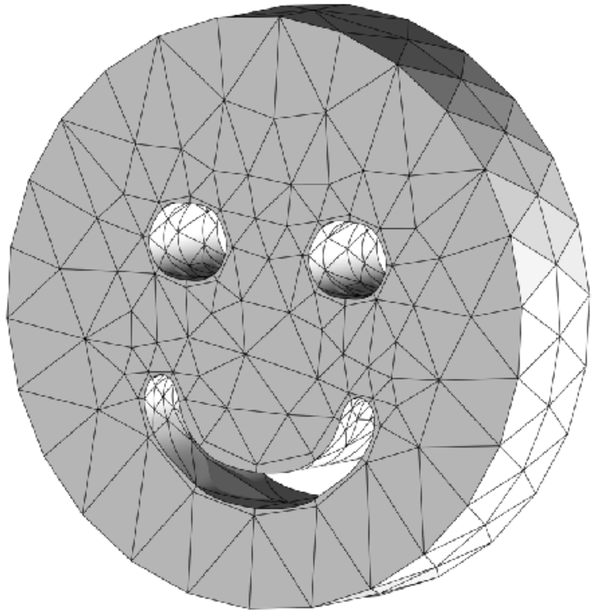
- Lots of new developments in this area with some quite exciting work!
- I hope some of these efforts might help in challenges you or your groups face using/developing Nektar++ in a more practical manner!
  - Chris Cave-Ayland (ICL RSE team) for efforts on modernising our CI
  - Chris Cantwell for packaging developments
  - Mohsen Lahooti for leading tutorial development efforts, with Mohammad Hossain & Ganlin Lyu.

Thanks for listening!

https://davidmoxey.uk/

david.moxey@kcl.ac.uk

www.nektar.info



EPSRC
Pioneering research and skills