tsi-cc Documentation

Release 1.0

David Yuan

Jan 13, 2022

Contents

This site is maintained by the Cloud Consultancy Team at EBI

CHAPTER 1

Table of contents

CHAPTER 2

ResOps Training

- Notice
  - Please check the course pre-requisite on the agenda page.
- Agenda for ResOps 2020
- Agenda for ResOps 2019

CHAPTER 3

Recommended Tools

- Ansible:
- Cloud Consultancy Team toolbox:
  - How to install tools
- Common Workflow Language
  - CWL self-contained training course
- Docker:
  - Docker best practice
- GitLab:
  - Overview
    - How to create docker image
    - How to deploy ECP instance
    - How to publish documentation
    - GitLab best practice
  - Useful links
    - GitLab CI with Maven
- Google Cloud Platform:
  - Accessing GCP node from CLI
- HPC in the cloud:
  - HPC with Slurm on GCP
  - Azure CycleCloud
    - HPC with CycleCloud on Azure
    - Useful links:

        - CycleCloud Overview
        - Setting up CycleCloud as a container
- Kubernetes:
  - Deployment of Kubernetes cluster on various clouds
  - Kubeflow for Machine Learning
  - Useful links:
- OneData:
  - Overview
  - Oneclient
  - User Guide
  - Onedata API
  - EBI setup details
  - EBI onedata portal
- Terraform:
  - Best Practice
- Supervisor:
  - Description: Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.
  - Reference documentation

CHAPTER 4

EBI Cloud Portal

CHAPTER 5

Architectural Design Patterns

- How to convert monolithic design to microservices

CHAPTER 6

About this documentation

This documentation is maintained at in a gitlab repository at EBI. Feel free to clone a local copy. You would need to run a local build to generate a local HTML site from the source files of **.md* and **.rst*. Install Sphinx with the instructions of Getting started with sphinx. Follow the instructions of Using markdown with sphinx and Local build with Sphinx to generate your local site.

The **master** branch will always be the latest, most up-to-date version. Other versions may be available through the selection drop-down on the bottom-left corner of this page.

If you find any mistakes, have suggestions for improvement, have questions, or wish to talk to us about your project, please contact us. Pull requests or issues filed against the git repository are also welcome.

CHAPTER 7

Contact us

You can contact us by email or come find us in pod 14 of the South Building.

CHAPTER 8

Site map

8.1 EMBL-EBI Cloud Portal documentation

- EMBL-EBI Cloud Portal documentation
  - Using the EMBL-EBI Cloud Portal
  - Packaging Applications for the EMBL-EBI Cloud Portal
  - Avoid security credentials on git public repository
  - API Endpoint documentation
- Presentations on UKRI 2019
  - EMBL-EBI Cloud Portal for Everyone

8.2 ResOps Training & Workshop - 2020

8.2.1 Introduction

ResOps training is mainly focus to research community who are willing to develop their cloud native skills. The ResOps course will not be delivered in-person for the foreseeable future, This course can be taken standalone, self-paced and for **FREE**.

8.2.2 Course objectives

This workshop will provide some background to cloud computing and practical experience in building, deploying and running applications in cloud platforms - OpenStack, Google, Amazon and Azure.

Using examples drawn from EMBL-EBI's experiences in the life-sciences, but using tools and technologies that are cross-platform and cross-domain, attendees will come away with a knowledge as to the user-centric hybrid cloud

strategy as well as practical aspects of deploying across different clouds and the architectural considerations around porting applications to a cloud.

8.2.3 Prerequisite

- An account should be created on Public GitLab CI/CD practical with GitLab.
- We are providing sandbox over OpenStack cloud to play exercises.
- Workshop participants can use Nano editor by default. Other CLI editors are also available in the sandbox.

8.2.4 Course videos

The course videos are all available for download for self-paced study. The slides are linked to the talks in the agenda.

8.2.5 Course guidelines

Date & Time (CEST)	Duration	Topic
Monday 05 Oct		
14:00 - 14:10	10 min	Introduction
14:10 - 14:40	30 min	Cloud 101
14:40 - 14:50	10 min	EBI Cloud Services (for courses at EBI only)
14:40 - 15:20	40 min	Porting apps into cloud [Part 1 | Part 2 | Part 3]
15:20 - 16:20	60 min	Creating Containers with Docker
16:20 - 16:30	10 min	Break
16:30 - 18:00	90 min	Docker Practicals
Tuesday 06 Oct		
14:00 - 15:00	60 min	Introduction to Gitlab
15:00 - 16:30	90 min	GitLab Practicals
16:30 - 16:45	15 min	GitLab Auto DevOps
16:45 - 17:05	20 min	Demo - GitLab Auto DevOps
17:05 - 18:00	55 min	Q&A, continue with practicals
Thursday 08 Oct		
14:00 - 15:00	60 min	Kubernetes 101
15:00 - 17:00	05 min	Overview of Kubernetes (Demo)
	115 min	Kubernetes (Demo)
17:00 - 17:45	05 min	Overview of K8S Practical
	40 min	Kubernetes Practical
17:45 - 17:50	05 min	Overview of Advanced K8S Practical
17:50 - 17:55	05 min	Summary
		Advanced Kubernetes
		Deployment and Deployment Strategies
		Service Mesh
		Advanced Kubernetes Practical [Homework]
		Advanced Kubernetes Reading [Homework]

8.3 ResOps Training & Workshop - 2019

8.3.1 Introduction

ResOps training is mainly focus to research community who are willing to develop their cloud native skills. The ResOps course will not be delivered in-person for the foreseeable future, This course can be taken standalone, self-paced and for **FREE**.

8.3.2 Course objectives

This workshop will provide some background to cloud computing and practical experience in building, deploying and running applications in cloud platforms - OpenStack, Google, Amazon and Azure.

Using examples drawn from EMBL-EBI's experiences in the life-sciences, but using tools and technologies that are cross-platform and cross-domain, attendees will come away with a knowledge as to the user-centric hybrid cloud strategy as well as practical aspects of deploying across different clouds and the architectural considerations around porting applications to a cloud.

8.3.3 Prerequisite

- Installation of Minikube and GIT, on your own laptop OR your VM* where you want to do your hand-on exercise.
- If you are choosing VM as your choice then please use at least 1vCPU, RAM 4GB and storage as per your choice.
- Understanding of Linux commands and GIT repository commands
- An account should be created on Public GitLab CI/CD practical with GitLab.
- We are providing sandbox over OpenStack cloud to play exercises.
- Workshop participants can use Nano editor by default. Other CLI editors are also available in the sandbox.

8.3.4 Course guidelines

Duration	Topic
30 min	Cloud 101
10 min	EBI Cloud Services
40 min	Porting apps into clouds
60 min	Creating Containers with Docker
90 min	Docker Practicals
60 min	Introduction to Gitlab
90 min	GitLab Practicals
60 min	Kubernetes 101
120 min	Overview / Kubernetes (Demo)
45 min	Overview / Kubernetes Practical
180 min	Overview / Advanced Kubernetes Practical
150 min	Overview / Advanced Kubernetes Reading

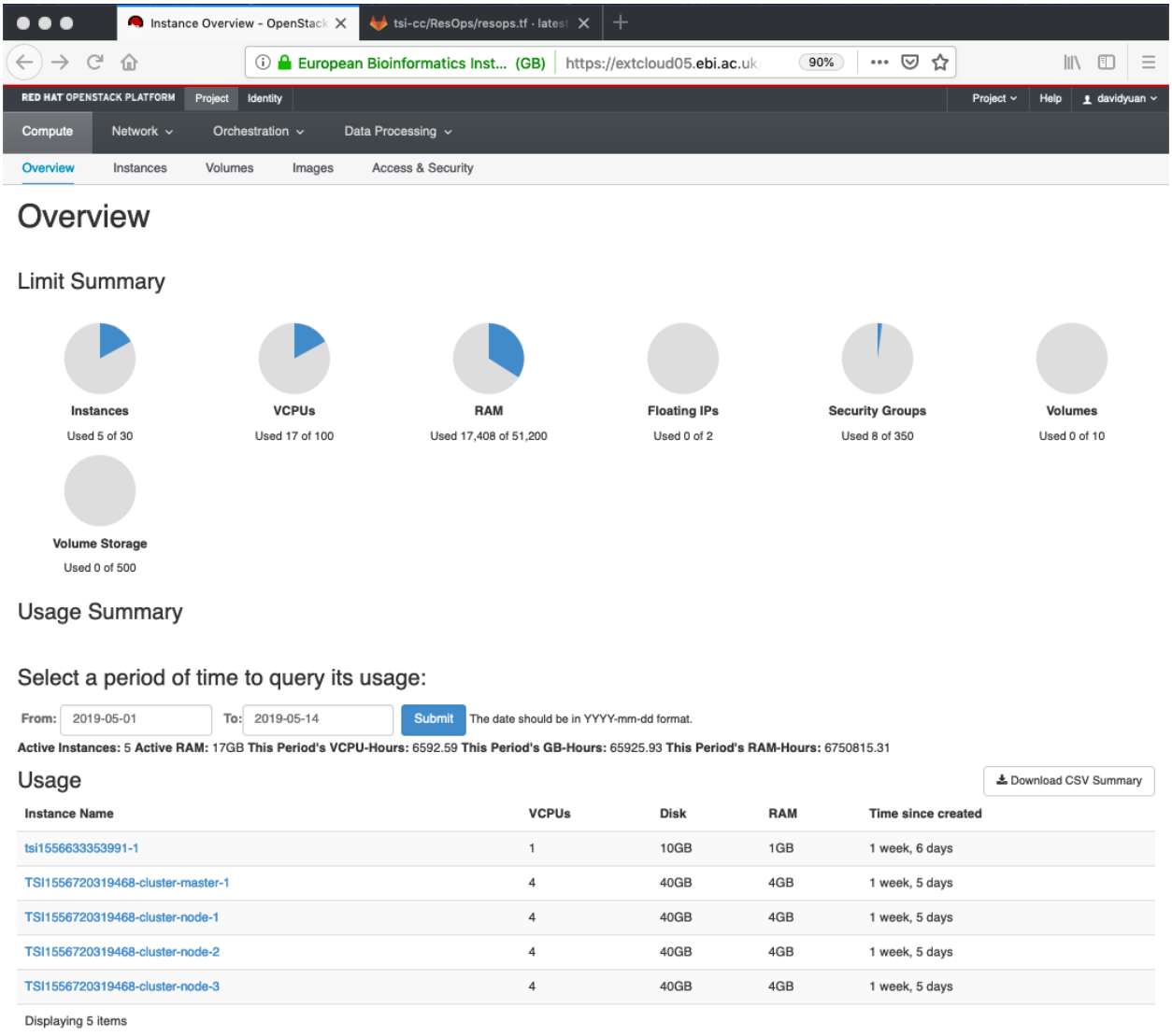
8.4 Creating VMs with Terraform on OSK for ResOps

8.4.1 Assumptions

1. Openstack CLI installed
2. Name and location of openrc file is hardcoded: *~/Downloads/ResOps-openrc.sh ~/Downloads/ResOps-openrc-V2.sh*

8.4.2 Process

- Log onto OpenStack Horizon. You can see the overview of the tenancy, where no resops cluster is created.

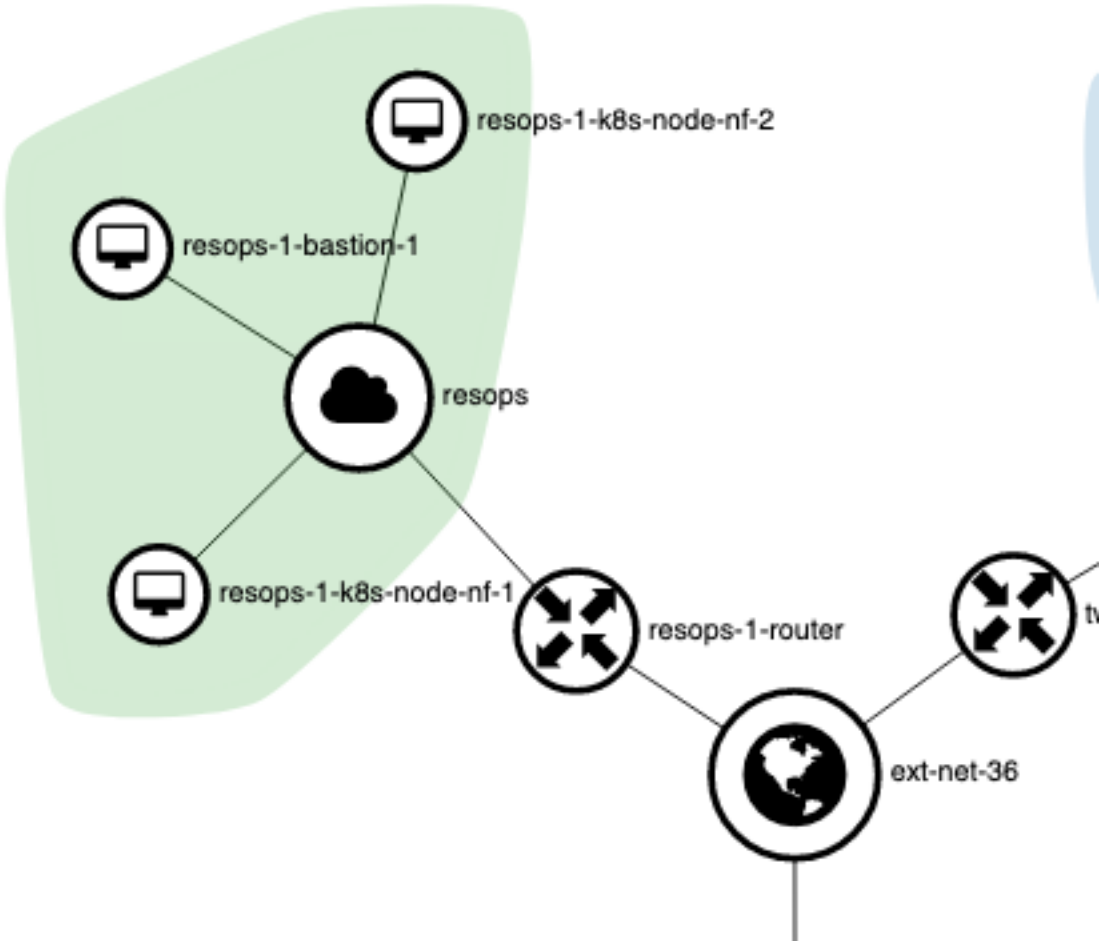


- Inspect variable values at https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/blob/master/tsi-cc/ResOps/scripts/kubespray/resops.tf. Make note of important variables such as *cluster_name*, *image*, *network_name*, *floatingip_pool*, *number_of_bastions*, *number_of_k8s_masters*, *number_of_k8s_nodes*, *number_of_k8s_nodes_no_floating_ip*, etc.. They describe the structure of the cluster.

- Inspect sample script https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/blob/master/tsi-cc/ResOps/scripts/kubespray/terraform.sh. Note that this script can not run other than on my laptop. It assumes my ResOps-openrc.sh at certain location. Replace the Bash script as needed.
- Inspect Terraform script for OpenStack under https://github.com/kubernetes-sigs/kubespray/tree/master/contrib/terraform/openstack. This script is well-written, following a lot of best practices as well as carefully modularized.
  - Default values are provided in the top level *variables.tf*.
  - The infrastructure are divided into three modules: network, ips and compute in the top level *kubespray.tf*.
  - Each module is organized with *main.tf* (as required for a Terraform module), *variables.tf* (input) and *output.tf* (output). The resource description is in *main.tf*. The APIs can be found at https://www.terraform.io/docs/providers/openstack/index.html.
- Run *~/IdeaProjects/tsi-ccdoc/tsi-cc/ResOps/scripts/kubespray/terraform.sh* in a terminal window. In the end, resources are created according to *~/IdeaProjects/tsi-ccdoc/tsi-cc/ResOps/scripts/kubespray/resops.tf*:

```
Apply complete! Resources: 20 added, 0 changed, 0 destroyed.

Outputs:

bastion_fips = [
    193.62.55.21
]
floating_network_id = e25c3173-bb5c-4bbc-83a7-f0551099c8cd
k8s_master_fips = []
k8s_node_fips = []
private_subnet_id = e457fd0b-c02d-4287-b049-4f143a32b2fb
router_id = 470c37d6-1ec6-49bd-9a9e-cc7712dfcf07
```

- Log back onto OpenStack Horizon to see VMs (a bastion, 2 nodes without floating IPs), private network / subnet (resops), router (resops-1-router) are created.



- Run *~/IdeaProjects/tsi-ccdoc/tsi-cc/ResOps/scripts/kubespray/ansipoor.sh* in a terminal window. It configures VMs ready for the parcticals. It takes 6 - 7 minutes for each VM. This script requires *openstack* CLI installed locally. Kubespray does not provide enough information to modify VMs remotely.
- Access the VMs via SSH directly if they have public IPs attached. Otherwise, use SSH tunnel via bastion server, for example *ssh -i ~/.ssh/id_rsa -o UserKnownHostsFile=/dev/null -o ProxyCommand="ssh -W %h:%p -i ~/.ssh/id_rsa ubuntu@193.62.55.21" ubuntu@10.0.0.5*
- After Docker practical and before Kubernetes practical, run *~/IdeaProjects/tsi-ccdoc/tsi-cc/ResOps/scripts/kubespray/startminikube.sh*. This is to avoid users to see overwhelming number of Docker processes for Minikube in the early practical. The script configures Minikube so that users do not have to do it manually.

NOTE:

- *-o UserKnownHostsFile=/dev/null* disables reading from and writing to ~/.ssh/known_hosts. This opens a security hole for man-in-the-middle attack. The option *-o StrictHostKeyChecking=No* would not work with *-o ProxyCommand* as the keys need to be exchanged. It is better practice for security to edit entries in ~/.ssh/known_hosts when deuplication happens but certainly less convenient.
- You can run *~/IdeaProjects/tsi-ccdoc/tsi-cc/ResOps/scripts/kubespray/terraform.destroy.sh* to remove the cluster completely.
- You can also run *~/IdeaProjects/tsi-ccdoc/tsi-cc/ResOps/scripts/kubespray/terraform.sh* with different parameters in *~/IdeaProjects/tsi-ccdoc/tsi-cc/ResOps/scripts/kubespray/resops.tf* to modify an existing cluster.

8.5 Docker exercises

There are several separate exercises, you can follow them in sequence or, if you prefer, jump straight to the exercise that interests you. For the most part, they're in the form of annotated command-line sessions. You can copy/paste the commands straight from the screen to your terminal, but you'll have to be aware of things that change - user names, process IDs etc.

The easy way to get access to the example Dockerfiles used in the exercises is to clone this repository from the command line:

```
# Clone this documentation if you haven't already done so
> git clone https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc.git
> cd tsi-ccdoc
```

The exercises are:

- Running Containers
- Running a Biobox Container
- Creating a Docker Image
- Optimising Docker Images
- Using the 'builder' pattern to build small images
- Exploring the security implications of running as root
- Passing metadata to your images and container runtime environments - optional
- Cleaning up - optional, you may need this eventually, but not when you start

For a practical look at real-world containers used for bioinformatics, check out http://bioboxes.org/

For more information, there are plenty of good tutorials out there. Two good ones are:

- https://prakhar.me/docker-curriculum/
- https://docs.docker.com/engine/tutorials/

8.6 Exercise 1: Running images

8.6.1 Objective

Learn how to run images, either one-off or as a running service

8.6.2 Terminology

In docker-speak, an **image** is a file (or set of files) that contain the application you want to run. You can copy images around, upload them, download them etc.

A **container** is a process that is started from an image. You can use the same image to run multiple containers, in the same way that you can use the same executable with many different arguments.

So, an image corresponds to files, a container corresponds to processes.

8.6.3 Running a simple command in a container

Docker images have a *name* and a *tag*. The default for the tag is 'latest', and can be omitted. If you ask docker to run an image that is not present on your system, it will download it from hub.docker.com first, then run it.

Most Linux distributions have pre-built images available on dockerhub, so you can readily find something to get you started. Let's start with the official Ubuntu linux image, and run a simple 'hello world'. The **docker run** command takes options first, then the image name, then the command and arguments to run follow it on the command line:

```
> docker run ubuntu /bin/echo 'hello world'
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
af49a5ceb2a5: Pull complete
8f9757b472e7: Pull complete
e931b117db38: Pull complete
47b5e16c0811: Pull complete
9332eaf1a55b: Pull complete
Digest: sha256:3b64c309deae7ab0f7dbdd42b6b326261ccd6261da5d88396439353162703fb5
Status: Downloaded newer image for ubuntu:latest
hello world
```

Note docker uses the 'ubuntu:latest' tag, since we didn't specify what version we want.

If we run the same command again, docker will find the image cached on our local disk, so it will run much faster:

```
> docker run ubuntu /bin/echo 'hello world'
hello world
```

8.6.4 Running an interactive command in an image

To do something more exciting, you might want an interactive shell, so you can poke around and do stuff. That's easy:

```
> docker run -t -i ubuntu /bin/bash
root@c69d6f8d89bd:/# id
uid=0(root) gid=0(root) groups=0(root)
root@c69d6f8d89bd:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
> exit # or hit control-D
```

The `-t -i` options make sure we can attach a terminal to the container, and we tell it to run our favourite shell as the application.

As you can see, you have root access in your container, and you are in what looks like a normal linux system. Now you can do whatever you like, e.g. install software and develop applications, all within the container of your choice.

This is a useful way of practicing and debugging the build process for an application while building a Dockerfile. Start with the base image, enter it with a shell, and learn what commands you need to run to build your package. Then capture those commands in the Dockerfile.

When the container environment gets messy, perhaps because you've installed stuff you didn't really want, you quit the container, build your image with what you've learned so far, and start again, from *that* image. This way, you progress step by step to a working image. Then, you optimise it!

8.6.5 Making changes to a container

In an interactive shell in a container, you can change the container contents. But the changes do not persist once you exit the container. If you re-run the image, you get a *new* container, not a re-run of the one you modified. Let's create a file in /tmp, then exit and restart the image, and look for the file:

```
> docker run -t -i ubuntu /bin/bash
root@1688f55c3418:/# touch /tmp/a-file.txt
root@1688f55c3418:/# ls -l /tmp/a-file.txt
-rw-r--r-- 1 root root 0 Nov 30 17:50 /tmp/a-file.txt
root@1688f55c3418:/# exit
exit
> docker run -t -i ubuntu /bin/bash
root@97b1e86df1f1:/# ls -l /tmp
total 0
root@97b1e86df1f1:/# exit
```

There are, of course, ways to make persistent changes to a container. More on that later.

So, can you get back to a container you modified before? Yes, actually, you can! Once a container exits docker keeps it cached for a while, so you can recover it. The `docker ps` command will tell you which containers are running now, and if you give it the `--all` flag, it tells you about containers that have exited, but still exist in the cache:

```
> docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
↪STATUS              PORTS               NAMES
> docker ps --all
CONTAINER ID        IMAGE               COMMAND                  CREATED
↪STATUS                      PORTS               NAMES
97b1e86df1f1        ubuntu              "/bin/bash"              5 minutes ago
↪Exited (0) 5 minutes ago                        backstabbing_brown
1688f55c3418        ubuntu              "/bin/bash"              5 minutes ago
↪Exited (0) 5 minutes ago                        ecstatic_hugle
c69d6f8d89bd        ubuntu              "/bin/bash"              9 minutes ago
↪Exited (0) 9 minutes ago                        sad_keller
960588723c36        ubuntu              "/bin/echo 'hello wor"   13 minutes ago
↪Exited (0) 13 minutes ago                       suspicious_mestorf
ffbb0f60bda6        ubuntu              "/bin/echo 'hello wor"   19 minutes ago
↪Exited (0) 19 minutes ago                       mad_booth
```

N.B. the container names, such as *sad_keller* above, are made up by docker at random. Use `--name xyz` to set the name explicitly yourself.

N.B. Since you're using the same virtual machine for the docker and kubernetes exercises, if someone has already started the kubernetes system there may be a lot of containers running. Filter them out with grep: `docker ps --all | grep -v k8s`

we know the container we modified was the one before last, with ID 1688f55c3418. We can **start** it again, then **attach** a terminal to it, and look for the file we created in /tmp:

```
> docker start 1688f55c3418
1688f55c3418
> docker attach 1688f55c3418
root@1688f55c3418:/# ls -l /tmp
total 0
-rw-r--r-- 1 root root 0 Nov 30 17:50 a-file.txt
```

In general, you don't want to do this much, it's messy if you run lots of containers. But it's useful to know, just in case you need it.

If you know you don't want to re-start a container afer you've run it, you can tell docker to clean it up automatically when it exits, with the `--rm` flag. E.g.:

```
> docker run --rm ubuntu echo 'hello world'
```

8.6.6 Starting a long-running service in a container

Containers are useful for running services, like web-servers etc. Many come packaged from the developers, so you can start one easily, but first you need to find the one you want to run. You can either search on hub.docker.com, or you can use the **docker search** command. Nginx is a popular web-server, let's look for that:

```
> docker search nginx
NAME                      DESCRIPTION                                     STARS
↪OFFICIAL   AUTOMATED
nginx                     Official build of Nginx.                        4719
↪[OK]
jwilder/nginx-proxy       Automated Nginx reverse proxy for docker c...   877
↪          [OK]
richarvey/nginx-php-fpm   Container running Nginx + PHP-FPM capable ...   311
↪          [OK]
million12/nginx-php       Nginx + PHP-FPM 5.5, 5.6, 7.0 (NG), CentOS...   76
↪          [OK]
webdevops/php-nginx       Nginx with PHP-FPM                              63
↪          [OK]
maxexcloo/nginx-php       Framework container with nginx and PHP-FPM...   58
↪          [OK]
bitnami/nginx             Bitnami nginx Docker Image                      20
↪          [OK]
gists/nginx               Nginx on Alpine                                 8
↪          [OK]
evild/alpine-nginx        Minimalistic Docker image with Nginx            8
↪          [OK]
million12/nginx           Nginx: extensible, nicely tuned for better...   8
↪          [OK]
maxexcloo/nginx           Framework container with nginx installed.       7
↪          [OK]
webdevops/nginx           Nginx container                                 7
↪          [OK]
1science/nginx            Nginx Docker images based on Alpine Linux       4
↪          [OK]
ixbox/nginx               Nginx on Alpine Linux.                          3
↪          [OK]
drupaldocker/nginx        NGINX for Drupal                                3
↪          [OK]
yfix/nginx                Yfix own build of the nginx-extras package      2
↪          [OK]
frekele/nginx             docker run --rm --name nginx -p 80:80 -p 4...   2
↪          [OK]
servivum/nginx            Nginx Docker Image with Useful Tools            2
↪          [OK]
dock0/nginx               Arch container running nginx                    2
↪          [OK]
blacklabelops/nginx       Dockerized Nginx Reverse Proxy Server.          2
↪          [OK]
xataz/nginx               Light nginx image                               2
↪          [OK]
```

(continues on next page)

(continued from previous page)

```
radial/nginx              Spoke container for Nginx, a high performa...   1
↪          [OK]
tozd/nginx                Dockerized nginx.                               1
↪          [OK]
c4tech/nginx              Several nginx images for web applications.      0
↪          [OK]
unblibraries/nginx        Baseline non-PHP nginx container                0
↪          [OK]
```

The official build of Nginx seems to be very popular, let's go with that:

```
> docker run -p 8080:80 nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
386a066cd84a: Pull complete
386dc9762af9: Pull complete
d685e39ac8a4: Pull complete
Digest: sha256:3861a20a81e4ba699859fe0724dc6afb2ce82d21cd1ddc27fff6ec76e4c2824e
Status: Downloaded newer image for nginx:latest
```

Note the `-p 8080:80` option. That tells docker to map port 80 in the container to port 8080 on the host, so you can communicate with it.

Note also that we didn't tell docker what program to run, that's baked into the container in this case. More on that later.

The next step depends on if you're running on your own laptop, or on a remote virtual machine.

- If you're running docker on your laptop, go to your browser and enter **localhost:8080** in the address bar, you should see a page with a **Welcome to nginx!** message.
- If you're running docker on a remote virtual machine, create another terminal session and SSH into the machine again, and run **curl -s http://localhost:8080 | grep Welcome**. You'll see the source-code, but also the **Welcome to nginx!** embedded in there

On your terminal where you ran the docker command, you'll see some log information:

```
172.17.0.1 - - [30/Nov/2016:18:07:59 +0000] "GET / HTTP/1.1" 200 612 "-" "Mozilla/5.0
↪(Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0" "-"
2016/11/30 18:07:59 [error] 7#7: *1 open() "/usr/share/nginx/html/favicon.ico" failed
↪(2: No such file or directory), client: 172.17.0.1, server: localhost, request:
↪"GET /favicon.ico HTTP/1.1", host: "localhost:8080"
172.17.0.1 - - [30/Nov/2016:18:07:59 +0000] "GET /favicon.ico HTTP/1.1" 404 169 "-"
↪"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0
↪" "-"
172.17.0.1 - - [30/Nov/2016:18:07:59 +0000] "GET /favicon.ico HTTP/1.1" 404 169 "-"
↪"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0
↪" "-"
2016/11/30 18:07:59 [error] 7#7: *1 open() "/usr/share/nginx/html/favicon.ico" failed
↪(2: No such file or directory), client: 172.17.0.1, server: localhost, request:
↪"GET /favicon.ico HTTP/1.1", host: "localhost:8080"
```

That's a good start, but you now have a terminal tied up with nginx, and if you hit CTRL-C in that terminal, your web-server dies. We can run it in the background instead:

```
> docker run --detach -p 8080:80 nginx
48a2dca14407484ca4e7f564d6e8c226d8fdd8441e5196577b2942383b251106
```

Go back to your browser or command line, reload **localhost:8080**, and you should get the page loaded again.

That nice long hexadecimal string is the *container-ID*, which we can use to get access to its logfiles with the **docker logs** command:

```
> docker logs --follow␣
↪48a2dca14407484ca4e7f564d6e8c226d8fdd8441e5196577b2942383b251106
172.17.0.1 - - [30/Nov/2016:18:18:40 +0000] "GET / HTTP/1.1" 304 0 "-" "Mozilla/5.0␣
↪(Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0" "-"
```

If you hit CTRL-C now, your container is still running, in the background. You can see this with the **docker ps** command:

```
> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
↪STATUS              PORTS                                NAMES
48a2dca14407        nginx               "nginx -g 'daemon off"   3 minutes ago
↪Up 3 minutes        443/tcp, 0.0.0.0:8080->80/tcp   pensive_booth
```

You can open a shell into the running container, if you wish. This can be handy for debugging. Let's take a look at the processes running in our nginx server. We have to install the procps package first, because the official nginx container is very lightweight, and doesn't have that in it by default:

```
> docker exec -t -i pensive_booth /bin/bash
root@48a2dca14407:/# apt-get update -y && apt-get install -y procps
root@48a2dca14407:/# ps auxww | grep ngin
root         1  0.0  0.0  31764  5164 ?        Ss   18:58   0:00 nginx: master␣
↪process nginx -g daemon off;
nginx        7  0.0  0.0  32156  2900 ?        S    18:58   0:00 nginx: worker process
root        15  0.0  0.0  11128  1036 ?        S+   18:59   0:00 grep ngin
```

Note that although you've installed the **procps** package in the container while it was running, we haven't installed it in the **image**, only into this running instance of the image. So if you kill this container and start it again, it won't have the **procps** package installed.

So now you've started nginx, how do you stop it? Use the **docker stop** command! You can give it either the container-ID or the name. You'll notice docker has made up a name for you, *pensive_booth* in this case. You may have to scroll the window below to the right so you can see it, depending on your browser.

```
> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
↪STATUS              PORTS                                NAMES
48a2dca14407        nginx               "nginx -g 'daemon off"   3 minutes ago
↪Up 3 minutes        443/tcp, 0.0.0.0:8080->80/tcp   pensive_booth

> docker stop pensive_booth
pensive_booth

> docker ps # check that it's gone
CONTAINER ID        IMAGE               COMMAND             CREATED
↪STATUS              PORTS               NAMES
```

8.6.7 Conclusion

You now know how to find a container that you want to run. You can start it, re-start it, run it as a detached service, attach a terminal to it for debugging, view the logs externally and even map ports between it and your host machine.

8.6.8 Best practices

- avoid making too many interactive changes to containers, see later exercises for how to modify and build your own containers
- prefer official images over those built by third-parties. Docker runs with privileges, so you have to be a bit careful what you run.

8.7 Exercise 2: Running a bioinfomatics application

8.7.1 Objective

Run a real-world bioinformatics application in a docker container

8.7.2 Running velvet from the bioboxes repository

This example is heavily abridged from the bioboxes.org tutorial, at http://bioboxes.org/docs/using-a-biobox/. Please refer to that tutorial for full details.

I've included the **biobox.yaml** file and the input reads directly in this repository, for convenience. You only need **cd** into the **bioboxes** directory, check you have everything, and invoke **docker run** appropriately.

```
# Clone this documentation if you haven't already done so
> # git clone https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc.git
> # cd tsi-ccdoc

# You need to be in the right working directory
> cd tsi-cc/ResOps/scripts/docker/bioboxes
> ls -l
total 0
drwxr-xr-x  4 wildish  wildish  136 Nov 30 11:37 input_data
drwxr-xr-x  5 wildish  wildish  170 Nov 30 11:45 output_data
> ls -l input_data/
total 64
-rw-r--r--  1 wildish  wildish    125 Nov 30 11:31 biobox.yaml
-rw-r--r--  1 wildish  wildish  27944 Nov 30 11:30 reads.fq.gz
```

Note the two `--volume` options, below. The first one maps the **input_data** directory from your machine to the **/bbx/input** directory on the container, where the application will look for it. The input directory is marked read-only (**:ro**). The second does the same for the output data, marking it read-write. This is a useful generic pattern for getting data into and out of containers. Be aware the behaviour is different if you use absolute or relative paths, we use absolute paths here.

The `default` argument at the end of the command line is fed to the application (velvet, in this case), docker doesn't pay attention to it.

```
> docker run --volume=`pwd`/input_data:/bbx/input:ro --volume=`pwd`/output_data:/bbx/
↪output:rw bioboxes/velvet default
Unable to find image 'bioboxes/velvet:latest' locally
latest: Pulling from bioboxes/velvet
e190868d63f8: Pull complete
909cd34c6fd7: Pull complete
0b9bfabab7c1: Pull complete
a3ed95caeb02: Pull complete
```

(continues on next page)

(continued from previous page)

```
c16026f9e2f2: Pull complete
d64cce756b2d: Pull complete
e1705543da3f: Pull complete
e003a99fece9: Pull complete
1ca78c008b50: Pull complete
1b41cafd4a53: Pull complete
e846c07b1d98: Pull complete
dc7515d258fb: Pull complete
6354b2058d01: Pull complete
497a947c4908: Pull complete
Digest: sha256:6611675a6d3755515592aa71932bd4ea4c26bccad34fae7a3ec1198ddcccddad
Status: Downloaded newer image for bioboxes/velvet:latest
[0.000002] Reading FastQ file /bbx/input/reads.fq.gz;
[0.001655] 228 sequences found
[0.001659] Done
[0.001697] Reading read set file /tmp/tmp.n2NrkikPB0/Sequences;
[0.001757] 228 sequences found
[0.001938] Done
[0.001941] 228 sequences in total.
[0.001970] Writing into roadmap file /tmp/tmp.n2NrkikPB0/Roadmaps...
[0.002214] Inputting sequences...
[0.002217] Inputting sequence 0 / 228
[0.019130]  === Sequences loaded in 0.016915 s
[0.019161] Done inputting sequences
[0.019164] Destroying splay table
[0.020453] Splay table destroyed
[0.000003] Reading roadmap file /tmp/tmp.n2NrkikPB0/Roadmaps
[0.000301] 228 roadmaps read
[0.000310] Creating insertion markers
[0.000338] Ordering insertion markers
[0.000430] Counting preNodes
[0.000459] 453 preNodes counted, creating them now
[0.001127] Adjusting marker info...
[0.001141] Connecting preNodes
[0.001255] Cleaning up memory
[0.001257] Done creating preGraph
[0.001260] Concatenation...
[0.001349] Renumbering preNodes
[0.001351] Initial preNode count 453
[0.001358] Destroyed 398 preNodes
[0.001360] Concatenation over!
[0.001362] Clipping short tips off preGraph
[0.001368] Concatenation...
[0.001381] Renumbering preNodes
[0.001383] Initial preNode count 55
[0.001387] Destroyed 18 preNodes
[0.001390] Concatenation over!
[0.001392] 9 tips cut off
[0.001396] 37 nodes left
[0.001435] Writing into pregraph file /tmp/tmp.n2NrkikPB0/PreGraph...
[0.001609] Reading read set file /tmp/tmp.n2NrkikPB0/Sequences;
[0.001657] 228 sequences found
[0.001834] Done
[0.002079] Reading pre-graph file /tmp/tmp.n2NrkikPB0/PreGraph
[0.002088] Graph has 37 nodes and 228 sequences
[0.002182] Scanning pre-graph file /tmp/tmp.n2NrkikPB0/PreGraph for k-mers
[0.002221] 3170 kmers found
```

(continues on next page)

(continued from previous page)

```
[0.002365] Sorting kmer occurence table ...
[0.002760] Sorting done.
[0.002763] Computing acceleration table...
[0.022207] Computing offsets...
[0.022236] Ghost Threading through reads 0 / 228
[0.022243]  === Ghost-Threaded in 0.000008 s
[0.022248] Threading through reads 0 / 228
[0.024005]  === Threaded in 0.001756 s
[0.027663] Correcting graph with cutoff 0.200000
[0.027683] Determining eligible starting points
[0.027725] Done listing starting nodes
[0.027729] Initializing todo lists
[0.027736] Done with initilization
[0.027738] Activating arc lookup table
[0.027741] Done activating arc lookup table
[0.027923] Concatenation...
[0.027927] Renumbering nodes
[0.027929] Initial node count 37
[0.027932] Removed 21 null nodes
[0.027939] Concatenation over!
[0.027965] Clipping short tips off graph, drastic
[0.027968] Concatenation...
[0.027971] Renumbering nodes
[0.027974] Initial node count 16
[0.027977] Removed 0 null nodes
[0.027980] Concatenation over!
[0.027983] 16 nodes left
[0.028051] Writing into graph file /tmp/tmp.n2NrkikPB0/Graph...
[0.028457] Measuring median coverage depth...
[0.028464] Median coverage depth = 7.238477
[0.028482] Removing contigs with coverage < 3.619238...
[0.028487] Concatenation...
[0.028504] Renumbering nodes
[0.028507] Initial node count 16
[0.028509] Removed 15 null nodes
[0.028511] Concatenation over!
[0.028514] Concatenation...
[0.028516] Renumbering nodes
[0.028518] Initial node count 1
[0.028521] Removed 0 null nodes
[0.028523] Concatenation over!
[0.028526] Clipping short tips off graph, drastic
[0.028528] Concatenation...
[0.028530] Renumbering nodes
[0.028533] Initial node count 1
[0.028535] Removed 0 null nodes
[0.028537] Concatenation over!
[0.028539] 1 nodes left
[0.028542] WARNING: NO EXPECTED COVERAGE PROVIDED
[0.028544] Velvet will be unable to resolve any repeats
[0.028546] See manual for instructions on how to set the expected coverage parameter
[0.028549] Concatenation...
[0.028551] Renumbering nodes
[0.028553] Initial node count 1
[0.028555] Removed 0 null nodes
[0.028558] Concatenation over!
[0.028560] Removing reference contigs with coverage < 3.619238...
```

(continues on next page)

(continued from previous page)

```
[0.028562] Concatenation...
[0.028684] Renumbering nodes
[0.028687] Initial node count 1
[0.028689] Removed 0 null nodes
[0.028692] Concatenation over!
[0.028730] Writing contigs into /tmp/tmp.n2NrkikPB0/contigs.fa...
[0.028981] Writing into stats file /tmp/tmp.n2NrkikPB0/stats.txt...
[0.029047] Writing into graph file /tmp/tmp.n2NrkikPB0/LastGraph...
[0.029271] Estimated Coverage cutoff = 3.619238
Final graph has 1 nodes and n50 of 2703, max 2703, total 2703, using 0/228 reads
```

And that's all there is to it! You can find the output data in the **output_data** directory (stunning, I know…)

```
> ls -l output_data/
total 16
-rw-r--r--  1 wildish  wildish   108 Nov 30 11:43 biobox.yaml
-rw-r--r--  1 wildish  wildish  2812 Nov 30 11:43 contigs.fa
cd ..
```

bioboxes.org is a really well-organized site, with excellent documentation, and is well worth exploring.

8.7.3 Conclusion

There are a lot of bioinformatics applications already wrapped up in container images. Bioboxes isn't the only site that provides them, but it's an extremely good one.

8.7.4 Best practices

- don't re-invent the wheel, it's worth looking to see who's done what you want already

8.8 Exercise 3: Creating your own docker image

8.8.1 Objective

Learn how to create a docker image that you can use later

8.8.2 Access to the tutorial material

You can create the dockerfiles in this exercise by cutting and pasting from the screen, or, if you've cloned the repository, you will find them already in the **tsi-cc/ResOps/scripts/docker/** subdirectory:

```
# Clone this documentation if you haven't already done so
> git clone https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc.git
> cd tsi-ccdoc/tsi-cc/ResOps/scripts/docker/
```

8.8.3 Creating an image, step by step

You can modify an image interactively, as we saw in the first exercise, but that's no sane way to build an image for re-use later. Much better is to use a **Dockerfile** to build it for you. Take a look at **Dockerfile.01**, which should have these contents:

```
#
# Comments start with an octothorpe, as you might expect
#
# Specify the 'base image'
FROM ubuntu:latest

#
# Naming the maintainer is good practice
LABEL Author="Your Name" Email="your@email.address"

#
# The 'LABEL' directive takes arbitrary key=value pairs
LABEL Description="This is my personal flavor of Ubuntu" Vendor="Your Name" Version=
↪"1.0"

#
# Now tell ubuntu to update itself
RUN apt-get update -y
```

You can have multiple **RUN** commands, though you should check out the **Best practices** for a comment about that.

You tell docker to build an image with that dockerfile by using the **docker build** command. We'll give it a **tag** with the `--tag` option, and we tell it which dockerfile to use with the `--file` option.

We also have to give it a *context* to build from, so we give it the current directory `.`. If we add files, they will be taken relative to that context. The context can also be a URL, see https://docs.docker.com/engine/reference/commandline/build/ for full details.

```
# N.B. This assumes you have the USER environment variable set in your environment
> docker build --tag $USER:ubuntu --file Dockerfile.01 .
Sending build context to Docker daemon 72.19 kB
Step 1 : FROM ubuntu:latest
 ---> 4ca3a192ff2a
Step 2 : MAINTAINER Your Name "your@email.address"
 ---> Running in 051314cdc3ec
 ---> ea59cb99c816
Removing intermediate container 051314cdc3ec
Step 3 : LABEL Description "This is my personal flavor of Ubuntu" Vendor "Your Name"␣
↪Version "1.0"
 ---> Running in 099b516c4bdf
 ---> 241e336f1ef1
Removing intermediate container 099b516c4bdf
Step 4 : RUN apt-get update -y
 ---> Running in 5ec72101d67b
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:4 http://archive.ubuntu.com/ubuntu xenial/main Sources [1103 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial/restricted Sources [5179 B]
Get:6 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
Get:7 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:8 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
```

(continues on next page)

(continued from previous page)

```
Get:9 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:10 http://archive.ubuntu.com/ubuntu xenial-updates/main Sources [261 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial-updates/restricted Sources [1872 B]
Get:12 http://archive.ubuntu.com/ubuntu xenial-updates/universe Sources [137 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [548 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial-updates/restricted amd64 Packages [11.
↪7 kB]
Get:15 http://archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [459␣
↪kB]
Get:16 http://archive.ubuntu.com/ubuntu xenial-security/main Sources [60.7 kB]
Get:17 http://archive.ubuntu.com/ubuntu xenial-security/restricted Sources [1872 B]
Get:18 http://archive.ubuntu.com/ubuntu xenial-security/universe Sources [15.8 kB]
Get:19 http://archive.ubuntu.com/ubuntu xenial-security/main amd64 Packages [225 kB]
Get:20 http://archive.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [11.
↪7 kB]
Get:21 http://archive.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [76.9␣
↪kB]
Fetched 24.6 MB in 14s (1721 kB/s)
Reading package lists...
 ---> 312bd6b10add
Removing intermediate container 5ec72101d67b
Successfully built 312bd6b10add
```

Now, you can see your image with the **docker images** command:

```
> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
wildish             ubuntu              312bd6b10add        About a minute ago  167.
↪6 MB
ubuntu              latest              4ca3a192ff2a        25 hours ago        128.
↪2 MB
```

Our new image is there, and it's about 40 MB bigger than the image we started from, because of the updates we applied.

We can now run that image and check that it really is updated by trying to apply the updates again, there should be nothing new to do:

```
> docker run -t -i $USER:ubuntu /bin/bash
root@4989d23e6e8b:/# apt-get update -y
Hit:1 http://archive.ubuntu.com/ubuntu xenial InRelease
Hit:2 http://archive.ubuntu.com/ubuntu xenial-updates InRelease
Hit:3 http://archive.ubuntu.com/ubuntu xenial-security InRelease
Reading package lists... Done
root@4989d23e6e8b:/# exit
```

As expected, there's nothing new to apply.

8.8.4 Inspecting an image to find out how it was built

A brief aside, if you want to find out how a container was built, you can use the **docker inspect** command. It gives full details as a JSON document, more than you'd normally want to know, but we can at least use it to get back the **MAINTAINER** and **LABELS** we added:

```
> docker inspect $USER:ubuntu | grep --after-context=6 Labels
            "Labels": {
                "Author": "Your Name",
                "Description": "This is my personal flavor of Ubuntu",
                "Email": "your@email.address",
                "Vendor": "Your Name",
                "Version": "1.0"
            }
--
            "Labels": {
                "Author": "Your Name",
                "Description": "This is my personal flavor of Ubuntu",
                "Email": "your@email.address",
                "Vendor": "Your Name",
                "Version": "1.0"
            }
```

Why do the **LABELS** we specified appear twice? I don't know...

8.8.5 Adding our own programs to the image

There's a sample Perl script, **hello-user.pl** in your working directory. Please take the time to make sure you understand how it works before proceeding.

Let's tell docker to add that script to the image, so we can run it as an application.

We'll use **Dockerfile.02**, which has the following content:

```
FROM ubuntu:latest
LABEL Author="Your Name" Email="your@email.address"

RUN apt-get update -y

#
# Set an environment variable in the container
ENV MY_NAME Tony

#
# Add our perl script
ADD hello-user.pl /app/hello.pl
```

You can see that we've set an environment variable in our image (**MY_NAME**) and we've added our script as **/app/hello.pl**. You can have as many **ENV** and **ADD** sections as you like, though as with the **RUN** section, it's worth learning about the best practices before adding too many.

Now build the image:

```
> docker build --tag $USER:ubuntu --file Dockerfile.02 .
Sending build context to Docker daemon  95.74kB
Step 1/6 : FROM ubuntu:latest
 ---> 7698f282e524
Step 2/6 : LABEL Author="Your Name" Email="your@email.address"
 ---> Using cache
 ---> 4da140dc87fa
Step 3/6 : LABEL Description="This is my personal flavor of Ubuntu" Vendor="Your Name
↪" Version="1.0"
 ---> Using cache
```

(continues on next page)

(continued from previous page)

```
 ---> a6f0cc9d1234
Step 4/6 : RUN apt-get update -y
 ---> Using cache
 ---> 2f162cdbcc1e
Step 5/6 : ENV MY_NAME Tony
 ---> Running in b166b73c2eb0
Removing intermediate container b166b73c2eb0
 ---> 4d2ba043c256
Step 6/6 : ADD hello-user.pl /app/hello.pl
 ---> d83241a70a07
Successfully built d83241a70a07
Successfully tagged wildish:ubuntu
```

Note steps 1 through 4, where the cache was used to save time building the image. I.e. we didn't have to build the entire image from scratch, and apply the updates again.

We've re-used the tag (`$USER:ubuntu`), so this version will replace the old one. That's not a good idea if the image is already in use in production, of course!

Now let's run the app in the image

```
> docker run -t -i --rm $USER:ubuntu /app/hello.pl
Hello Tony
```

What happens if we update our script, will docker be smart enough to pick up the changes? Yes, up to a point.

Let's start by copying a new version of the script in place, and re-build the image:

```
> cp hello-user-with-args.pl hello-user.pl
> docker build --tag $USER:ubuntu --file Dockerfile.02 .
Sending build context to Docker daemon 81.92 kB
Step 1 : FROM ubuntu:latest
 ---> 4ca3a192ff2a
Step 2 : MAINTAINER Your Name "your@email.address"
 ---> Using cache
 ---> ea59cb99c816
Step 3 : LABEL Description "This is my personal flavor of Ubuntu" Vendor "Your Name"␣
↪Version "1.0"
 ---> Using cache
 ---> 241e336f1ef1
Step 4 : RUN apt-get update -y
 ---> Using cache
 ---> 312bd6b10add
Step 5 : ENV MY_NAME Tony
 ---> Using cache
 ---> 0857feeb7bb0
Step 6 : ADD hello-user.pl /app/hello.pl
 ---> ae442bdee840
Removing intermediate container 5fe5c7d58e0d
Successfully built ae442bdee840
```

Step 6 didn't use the cache, because docker noticed the script had been updated. However, if the script itself hadn't changed, but modules or libraries that it uses have changed, docker wouldn't be able to pick that up on its own. Put differently, the build process can't 'see through' commands like **apt-get update -y** to know that there are changes since it was last run.

In case you want to, you can force a re-build from the start by telling docker not to use the cache:

```
> docker build --no-cache --tag $USER:ubuntu --file Dockerfile.02 .
[...]
```

8.8.6 Passing arguments to an application in an image

Can we change who it says hello to? Yes, we can! We can set environment variables in the container before the application runs by using the `--env` flag with **docker run**:

```
> docker run -t -i --rm --env MY_NAME=Whoever $USER:ubuntu /app/hello.pl
Hello Whoever
```

The new version uses the environment variable **MY_NAME** by default, as before, but also allows you to override that by giving command-line options. To do that, simply append the arguments to the end of the **docker run** command:

```
> docker run --rm -ti $USER:ubuntu /app/hello.pl someone
Hello someone
```

8.8.7 Running an application by default

Finally, let's try getting our application to run by default, so we don't have to remember the path to it whenever we want to run it. **Dockerfile.03** shows how to do that

```
FROM ubuntu:latest
LABEL Author="Your Name" Email="your@email.address"

#
# The 'LABEL' directive takes arbitrary key=value pairs
LABEL Description="This is my personal flavor of Ubuntu" Vendor="Your Name" Version=
↪"1.0"

#
# Now tell ubuntu to update itself
RUN apt-get update -y

#
# Set an environment variable in the container
ENV MY_NAME Tony
ADD hello-user.pl /app/hello.pl

#
# Specify the command to run!
CMD /app/hello.pl
```

So, build it, then run it:

```
> docker build --tag $USER:ubuntu --file Dockerfile.03 .
[...]
> docker run --rm -ti $USER:ubuntu
Hello Tony
```

8.8.8 Optimizing builds

We saw that `docker build --no-cache ...` solves the problem of docker not knowing if something was updated, but doing *everything* from scratch can be a bit expensive. The obvious solution is to build intermediate images, and move the more stable stuff into the earlier images. Take a look at **Dockerfile.04.base** and **Dockerfile.04.app**, they're just **Dockerfile.03** split into two parts:

```
> cat Dockerfile.04.base
FROM ubuntu:latest
LABEL Author="Your Name" Email="your@email.address"

RUN apt-get update -y

> cat Dockerfile.04.app
FROM wildish:ubuntu

ENV MY_NAME Tony
ADD hello-user.pl /app/hello.pl

CMD /app/hello.pl
```

Dockerfile.04.base builds an updated ubuntu image, while **Dockerfile.04.app** uses *that* image as its base. As long as we tag the base image as **$USER:ubuntu**, and refer to it correctly in the **FROM** statement for the app, the app will find it correctly. We can't use the environment variable in the **FROM** statement for the app, so we have to hard-code the user name there. Change it to your own user name before building the image.

Note also that **Dockerfile.04.app** doesn't have a **MAINTAINER** or **LABEL** section, which means it will inherit them from the base image.

Now we can build our app in two stages:

```
> docker build --tag $USER:ubuntu --file Dockerfile.04.base .
Sending build context to Docker daemon 86.53 kB
Step 1 : FROM ubuntu:latest
 ---> 4ca3a192ff2a
Step 2 : MAINTAINER Your Name "your@email.address"
 ---> Using cache
 ---> 223050aea37e
Step 3 : LABEL Description "This is my personal flavor of Ubuntu" Vendor "Your Name"␣
↪Version "1.0"
 ---> Using cache
 ---> c03ba3b7afd5
Step 4 : RUN apt-get update -y
 ---> Using cache
 ---> 00269c0edb02
Successfully built 00269c0edb02

> docker build --tag $USER:hello --file Dockerfile.04.app .
Sending build context to Docker daemon 86.53 kB
Step 1 : FROM wildish:ubuntu
 ---> 00269c0edb02
Step 2 : ENV MY_NAME Tony
 ---> Using cache
 ---> 0fa5ba428fe0
Step 3 : ADD hello-user.pl /app/hello.pl
 ---> Using cache
 ---> 704d3c5941c6
Step 4 : CMD /app/hello.pl
```

(continues on next page)

(continued from previous page)

```
 ---> Using cache
 ---> ce37fdc3bd4e
Successfully built ce37fdc3bd4e

> docker run --rm -ti $USER:hello
Hello Tony
```

If we force a rebuild of the app, it's very quick now, because it doesn't have to update the base ubuntu operating system.

Docker now supports a 'docker builder' pattern, which formalises this multi-step build approach. See https://docs.docker.com/develop/develop-images/multistage-build/ for more details, as well as exercise 5.

8.8.9 Conclusion

You can now build your own images, starting from a base image, updating it, adding files, specifying environment variables, and specifying the default executable to run.

You know how to tag your images, so they have a meaningful name, and you know how to specify useful metadata that you can retrieve programatically.

8.8.10 Best practices

- avoid building big images, start from the lightest base you can manage and only add what you really need
- move any stable, heavy parts of your build early in the Dockerfile, to maximize the benefit of the cache
- consider using intermediate builds, to further isolate stable parts from volatile parts if you need to force builds
- follow the official best-practices guide, at https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

8.9 Exercise 4: Optimising a docker image

8.9.1 Objective

Learn how to optimise a docker image

8.9.2 Introduction

There are a lot of best-practice guidelines out there, but they can be summarised in just two questions:

- If I run the build in a year from now, on a clean machine, will I get the same result? What might change that would give me a different result?
- What does each line in the Dockerfile do to the image, to the cache?

As for image size, how big is too big? It really does depend on what you're doing, and anything less than 200 MB or so is excellent. Beyond 1 GB, you're probably carrying a lot of stuff you don't need, and performance may suffer for it.

Software like R, Perl or Python (conda) tend to drag in a lot of packages depending on how they're installed, so watch out for them. Sometimes you can clean things up (e.g. **conda clean –all**), or you may need to install things manually, to keep control.

Also, many providers of base OS images have slimmed-down versions that you can use if you don't want everything. E.g. There's a **debian:jessie** (129 MB) and a **debian:jessie-slim** (81 MB).

8.9.3 Optimise a Dockerfile

Take a look at **Dockerfile.spades.no-opt** in the **tsi-cc/ResOps/scripts/docker** directory. You can build a container from this with `docker build -t spades:no-opt -f Dockerfile.spades.no-opt .`, but **don't do that!** This takes about 15 minutes to run, and the resulting container is about 6.3 GB in size. That's a huge image, and building it wastes a lot of your time!

Following the guidelines in this tutorial, you can get it down to 2 minutes and 760 MB or less. You should spend a few minutes thinking about how to do that yourself, using what you've already learned. Give it a go on your own, then compare your results with the steps shown here.

8.9.4 Are you installing stuff you don't need?

The docker file installs the Gnome graphical desktop, do you think you're likely to need that in a container? Probably not, so go ahead and remove it.

Also, since spades itself is being installed by **conda**, you probably don't need to install GCC - you're not compiling spades from source so why do you need a compiler? Remove that too. That gets the container down to 3.1 GB, half the size already.

Give that a go, and see if you can build the image and check its size. Tag the image with **spades:opt**, instead of **spades:no-opt**, so you can directly compare them later.

8.9.5 The centos image is rather large

How about starting from a smaller image, say **debian:jessie-slim**?

You'll need to replace the **yum** commands with **apt-get**, since CentOS and Debian use different package managers, but apart from that, that's all you'll need to do there.

That gets the image down to 2.6 GB, another 500 MB saved.

8.9.6 There's still more stuff you don't need in there

You may not know the **conda** installer (you should, but that's another story). Even so, you can see that it's being used to install a lot of things other than spades. Since we only want spades here, we don't need to install **metabat2**, **bwa**, **samtools** and **blast**. Remove them from the appropriate line of the Dockerfile.

Likewise, you can remove the lines that install **ncurses** and **openssl**, since we don't need to install them explicitly. If spades needs them, it will pull them in via its dependencies.

If you rebuild the image now you'll see it's only a bit over 900 MB in size, great progress!

8.9.7 Now to make better use of the cache

The **RUN** commands can be concatenated into a single RUN command. That alone will save about 200 MB more, so you now have an image that takes only 760 MB and less than 2 minutes to build!

Compare your Dockerfile with **Dockerfile.spades.opt** in the same directory and see how it looks.

You can check your image works with **docker run -it –rm spades:opt spades.py –test**

8.9.8 Conclusion

You now know several simple ways to optimise the size of a docker image:

- pick the smallest base image you can easily use
- only install what you need
- pay attention to the cache and the number of layers you're using

and from the previous exercise, you know you can optimise build-times by building your own base image for all your projects. However, there's one more technique worth knowing about, and that's the subject of the next exercise.

8.9.9 Best Practices

- check the size of the image you build, think if it's worth spending time making it smaller.
- check the base image you use, and the way you install software. Will this be reproducible in a year from now?

8.10 Exercise 5: Using the 'builder' pattern to build small images

8.10.1 Objective

Use the 'builder' pattern, otherwise known as multi-stage builds, to build even smaller docker images, with cleaner dockerfiles.

8.10.2 The Builder pattern

Optimising a build can be complex, it's hard to keep your image from getting cluttered. E.g. installing a package might bring in a pile of documentation which you really don't need. You can stay on top of that with chained RUN commands to remove the excess as it gets installed, but it gets messy eventually, and hard to follow.

Docker allows multi-step builds to get round this. The idea is that you first build a docker image that contains your software, then build a second image, copying across the installed software from the first one, taking across only the bits you need. You can leave behind all the junk that you don't need, without having to jump through hoops to clean it out explicitly.

Take a look at **Dockerfile.spades.builder** in the **tsi-cc/ResOps/scripts/docker** directory. You should recognise the first sections, they're identical to the optimised spades dockerfile from the previous exercise, with one exception. Instead of `FROM debian:jessie-slim`, the line now reads `FROM debian:jessie-slim AS builder`. This gives this particular image a name ('builder') which we can refer to later in the build process.

At the end of that first build, there are commands to build another image:

```
FROM debian:jessie-slim
COPY --from=builder /install /install
COPY --from=builder /root/.bashrc /root/.bashrc
ENV PATH /install/conda/bin:$PATH
```

Notice the **COPY** commands, which we use to copy from the image named 'builder' that we've just created! So we now have a **debian:jessie-slim** image with *only* the files we need to get spades running, and not much else.

In fact, we've done even better than that. Above that second **FROM** line, you'll see some cleanup code:

```
RUN conda clean --all -y && \
    rm -rf /install/conda/pkgs/*/info && \
    find /install -name '*.a' -exec rm {} \;
```

The first two lines are conda-specific, don't worry if you don't know what they do, suffice it to say that they remove stuff that's not strictly necessary for conda to run an application.

The third line removes all static libraries from the conda installation. You won't need these anymore, so they're useless.

Why do we only remove the static libraries from the conda installation, why not from the entire image? The reason is that the second image starts from debian:jessie-slim, and copies across only the stuff from the /install directory. So whatever cleanup we do in the first image outside that directory can have no effect in the second image.

Note also that if we didn't build the second image, but only built the first image, with those same cleanup commands in the same order that they are now, they wouldn't do anything useful. They won't reduce the size of the first image because of the layered image structure. The first **RUN** command that installs all the software creates one cache layer, the second one that removes the junk creates *another* layer, without that stuff. But the first layer, with it all in, is still part of the image! If we wanted to concatenate the two RUN commands together, that would make the first image smaller, but would also make the whole thing less readable.

Finally, in this example, although we've use **debian:jessie-slim** for both the image stages, we're not obliged to do that. Our 'builder' image can be something fat, with all the software we need to build our application already installed. E.g. it may have C and C++ compilers already in the image, to simplify the first steps. The second image can start from a much lighter base, even from a different distribution. As long as it has enough to *run* applications from the first image, that's good enough.

So, go ahead and build the image with **docker build -t spades:builder -f Dockerfile.spades.builder .**. It will take less than two minutes, and the final image will be a measly 330 MB in size. That's less than half the size of the previous optimised image, and is more than enough, there's little point in optimising beyond this point.

If you were to run **docker images** now on a clean system, you'd see something like this:

```
> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
spades              builder             194d8647f15d        3 seconds ago       274MB
<none>              <none>              d9552808c2f0        10 seconds ago      764MB
debian              jessie-slim         a0536420b3b7        8 days ago          81.4MB
```

You can see the base image, **debian:jessie-slim**, as well as the **spades:builder** image you just built. The other one, with no names for the repository or tag, is the first image built by the multi-stage build. You'll see it's the same size as the previously optimised spades image, because of course it's the same docker code that built it.

The last exercise will show you how to clean up unwanted images like that from your system. It's optional, you'll need it eventually if you're using your own laptop, but you may not need it on production systems.

8.10.3 Conclusion

Multi-stage builds can greatly reduce the size of your image, and make your dockerfile more readable. They're also very easy to use. Of course you can combine multi-stage builds with building your own base image for ultimate control of your images.

8.10.4 Best practices

- install your software as cleanly as possible, i.e. into a separate directory from the system software if possible
- see if you can use the builder pattern to reduce the amount of junk in your image, your savings could be significant

Congratulations, you're an expert! If you got this far, remember to update your CV to reflect your Docker skills.

8.11 Exercise 6: Running as root or a non-root user

8.11.1 Objective

Learn how to manage running a container as a given user.

8.11.2 Containers run as root by default

As a normal user, you should not have access to certain files on the system, e.g. the **/etc/sudoers** file:

```
> wc /etc/sudoers
wc: /etc/sudoers: Permission denied
```

However, running a container with a volume mount gives you complete access to the system, because the container runs as root:

```
> docker run --rm -it --volume /etc:/mnt alpine:3.5 wc /mnt/sudoers
       97       496      3174 /mnt/sudoers
```

This means you can read, and potentially change, files on the host system that you should not have access to!

To prove this point further, try creating a file on a volume-mounted directory, and see what user it shows as on the host system:

```
> mkdir tmp
> cd tmp
> touch a-file.txt
> touch another-file.txt
> ls -l
total 0
-rw-rw-r--. 1 centos centos 0 Aug  5 16:12 a-file.txt
-rw-rw-r--. 1 centos centos 0 Aug  5 16:12 another-file.txt
> docker run --rm --volume `pwd`:/mnt alpine:3.5 touch /mnt/alpine-file.txt
> ls -l
total 0
-rw-rw-r--. 1 centos centos 0 Aug  5 16:12 a-file.txt
-rw-r--r--. 1 root   root   0 Aug  5 16:12 alpine-file.txt
-rw-rw-r--. 1 centos centos 0 Aug  5 16:12 another-file.txt
```

Note that the file created within the alpine container is owned by root!

Using volume-mounts into a container allows you not only to read and write as root, but also to mount directories that you cannot even see otherwise. The volume-mount is executed by the docker daemon, which runs as root, so it can fulfill the mount request even if you wouldn't be able to see that directory as a normal user.

For example, in **/etc/selinux/final** there's a hierarchy of files that users should not be able to see. List the **/etc/selinux** directory and you'll see the **final** directory is protected. Attempt to list it's contents as a normal user, and you'll fail:

```
> ls -l /etc/selinux
total 8
-rw-r--r--. 1 root root  542 Jan 28  2019 config
drwx------. 3 root root   22 Aug  5 13:02 final
-rw-r--r--. 1 root root 2321 Oct 30  2018 semanage.conf
drwxr-xr-x. 8 root root  226 Aug  5 13:02 targeted
drwxr-xr-x. 2 root root    6 Oct 30  2018 tmp
> ls -l /etc/selinux/final
ls: cannot open directory /etc/selinux/final: Permission denied
```

However, if you *know* the path you want to examine under that tree, you can use a volume mount to access it in a container:

```
> docker run --rm -it --volume /etc/selinux/final/targeted/contexts:/mnt alpine:3.5␣
↪ls -l /mnt
total 0
drwx------    2 root     root             6 Aug  5 13:02 files
```

If you don't find that scary, please go back and read this section again, until you do.

This is why many HPC centres won't allow you to run docker images on their machines, which are shared among many users. There are alternatives, one of the best is **singularity**, which is essentially a drop-in replacement for docker which solves the security issues by only allowing you to run as the user you are on the host system.

If you're only running on your own machines, and nobody else has access to them, there's not much to worry about. If, however, you're running a service of some sort (a web server or portal) then you need to consider that someone could exploit bugs in your service to gain access to the host via the container. Yes, it does happen!

The solution, then, is to run your service as a non-root user in the container, and make sure that user cannot escalate their privilege within the container.

8.11.3 Adding a non-root user to an image

You can add users to containers in much the same way as you would with any linux distribution. How exactly you do this is specific to the particular linux flavour you're using. For the **alpine** distribution, you'll need to install the user-management packages. There's a dockerfile that does this, **Dockerfile.user**, in the **tsi-cc/ResOps/scripts/docker** directory of the tutorial repository, note that it creates both a *user* and a *group*, because if you don't specify a group then the *root* group is used by default:

```
> cat Dockerfile.user
FROM alpine:3.5

RUN apk update && \
    apk add shadow && \
    groupadd muggles && \
    useradd -ms /bin/sh -G muggles dudley

USER dudley:muggles
```

Build an image from this, tag it as **user**. Now try running a container with that image and see if you can see the hidden files:

```
docker run --rm -it --volume /etc/selinux/final/targeted/contexts:/mnt user
> ls -l /mnt
ls: can't open '/mnt': Permission denied
total 0
> ls -ld /mnt
drwx------    3 root     root            19 Aug  5 13:02 /mnt
> id
uid=1000(dudley) gid=1000(muggles)
```

You can see that the volume-mount still succeeds, but you can't see the files, because you don't have permission. The **dudley** user has no way to become root within the container, so has no way to cheat. So far so good!

But that's not the end of the story yet. Docker allows you to specify which user to run as in the container, from the command line. So a user can run the container as root by explicitly asking for it:

```
> docker run --rm -it --volume /etc/selinux/final/targeted/contexts:/mnt user ls -l /
↪mnt
ls: can't open '/mnt': Permission denied
total 0
> docker run --rm -it --volume /etc/selinux/final/targeted/contexts:/mnt --user root␣
↪user ls -l /mnt
total 0
drwx------    2 root     root             6 Aug  5 13:02 files
```

The bottom line is that if the user is allowed to run containers for themselves then they can access pretty much anything on your system as if they are the root user. If they only have access to a service that you are providing, such as a web server, then you can protect your system more by running the service as a non-root user in the container.

8.11.4 Conclusion

Docker has some significant security issues. There are measures you can take to mitigate them, but if the user can run containers on your infrastructure for themselves, there's no foolproof method to prevent possible abuse.

8.11.5 Best practices

- consider who has access to the host system when you install docker on a machine. Can you limit the set of users to reduce security exposure?
- if you're running a service, that users only access through the web or some other API, there are a few things you should do to protect it:
  - create an unprivileged user in the image, use that user to run the service
  - make sure your container only mounts volumes that you need from the host filesystem
    - be as restrictive as possible, don't mount more than you need, and never from the system directories
    - mount volumes read-only if you can, to prevent unwanted changes to files
    - if you allow users to upload data, provide separate input and output volumes, don't let them write in volumes where you store other files

8.12 Exercise 7: Using metadata

8.12.1 Objective

Learn how to pass information (metadata) from your build environment through to your running containers.

This is not often needed, but can be handy if you want to make professional-grade images with proper documentation built in. If you define an ontology for your labels and environment variables you can even use it for automated workflows which select the correct image to run based on their properties.

8.12.2 The four docker environments

There are four different environments that interact to create a running container

- the **development environment**, where the developer creates the dockerfile and builds the docker image
- the **docker build environment**, where the docker daemon executes the build on behalf of the developer
- the **docker image**, containing the baked-in software that will run when the container starts
- the **docker runtime environment**, which can be different from one invocation of an image to the next

You already know that you can create more than one image from a docker container by giving it different tags (`docker build -t ...`), but what if you want to create two images from the same base code but with different settings. For example, maybe you're building an annotation application that has experimental features that need to be compiled into the binary, and you want to build a DEV and a PROD version of the same application, with and without that feature.

One way to do this is to have two different dockerfiles, which differ only by the value you set for a flag or an environment variable. This is inconvenient if the dockerfiles are still under development, any changes you make to one must be propagated to the other and could easily be forgotten. What you need is a way to run the build twice, pass different settings in on the command line, and have them used in the container. You can do that with the **ARG** statment in your dockerfile.

8.12.3 Using ARG to control your image builds

Take the following dockerfile, which is a reduced version of **Dockerfile.metadata**, from the **tsi-cc/ResOps/scripts/docker** directory of the tutorial repository. It defines three **ARG** variables, one is **APP_VERSION**, which we set to either PRODUCTION or DEVELOPMENT, by hand. The other two are **BRANCH**, which we will set to the git branch we are working on, and COMMIT_ID, which we will set to the git commit hash. Normally it's enough to use git tags to identify a version, but this gives us more precision, in case someone moves a tag this will still be correct.

```
> cat Dockerfile.metadata
FROM alpine:3.5

ARG APP_VERSION=undefined
ARG GIT_BRANCH
ARG GIT_COMMIT_ID

LABEL uk.ac.ebi.BRANCH=$GIT_BRANCH \
      uk.ac.ebi.COMMIT_ID=$GIT_COMMIT_ID \
      uk.ac.ebi.APP_VERSION=$APP_VERSION

ENV VERSION=$APP_VERSION \
    BRANCH=$GIT_BRANCH \
    COMMIT_ID=$GIT_COMMIT_ID
```

A few things to note about this dockerfile:

- ARG lines cannot be chained in the same way that LABEL and ENV lines can. Why? I don't know...
- The recommended naming convention for LABELs like this is to use 'reverse DNS' notation. Java programmers will be familiar with this, it's just a scheme for making sure that labels are in separate namespaces. E.g, if the base image, alpine:3.5, had a VERSION label set in it, this will make sure yours doesn't collide with theirs.
  - why isn't this recommended for ENV variables too? For one thing, the period ('.') isn't valid in environment variable names. For another, people probably simply wouldn't use such complicated names in their applications.

So how do we set an ARG? With the **–build-args** argument to **docker build**. For example:

```
> app_version=DEVELOPMENT

# Use some git magic to get the branch name and commit-id
> git_branch=`git branch | egrep '^\*' | awk '{ print $NF }'`
> git_commit_id=`git rev-parse HEAD`

> docker build \
   --build-arg APP_VERSION=$app_version \
   --build-arg GIT_BRANCH=$git_branch \
   --build-arg GIT_COMMIT_ID=$git_commit_id \
   -t metadata:$app_version \
   -f Dockerfile.metadata .
```

Note how we use the value of **$app_version** both in the **–build-args** and in the tag that we apply to the image.

You can now inspect the image to see those labels:

```
> docker inspect metadata:DEVELOPMENT | egrep uk.ac.ebi | head -3
                "uk.ac.ebi.APP_VERSION": "DEVELOPMENT",
                "uk.ac.ebi.BRANCH": "latest",
                "uk.ac.ebi.COMMIT_ID": "97eba80a9af544bdaba72d44f5e59f72e506c8d4"
```

and you can run the image to see the values, at runtime:

```
> docker run -it --rm metadata:DEVELOPMENT env | egrep 'VERSION|BRANCH|COMMIT_ID'
BRANCH=latest
COMMIT_ID=97eba80a9af544bdaba72d44f5e59f72e506c8d4
VERSION=DEVELOPMENT
```

To build a production version, simply define app_version=PRODUCTION and build the image again. Running **docker images** will then show you both images:

```
> app_version=PRODUCTION

> docker build \
   --build-arg APP_VERSION=$app_version \
   --build-arg GIT_BRANCH=$git_branch \
   --build-arg GIT_COMMIT_ID=$git_commit_id \
   -t metadata:$app_version \
   -f Dockerfile.metadata .

> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
metadata            PRODUCTION          73acfce2eac1        5 seconds ago       4MB
metadata            DEVELOPMENT         70f5c17bc2d8        12 minutes ago      4MB
alpine              3.5                 f80194ae2e0c        6 months ago        4MB
```

8.12.4 Conclusion

You can pass information from the development environment to the build environment with **–build-args**, which allows you to build multiple variants of an image from the same dockerfile. This information can then be baked into the image for inspection, or for use when the image is run in a container.

8.12.5 Best practices

- using ARGs is useful if you want to build multiple images from the same dockerfile, to avoid copy/paste errors
- choose your LABEL and ENV names carefully, make sure you don't overwrite anything in the base image that might cause confusion for users
- automated build environments (such as gitlab) often make a bunch of useful bit of information available during the build. You can incorporate that information into the image as a way of documenting it automatically.

8.13 Exercise 8: Removing images from your machine (optional)

8.13.1 Objective

Learn how to remove containers and images from your machine when you no longer need them

8.13.2 Cleaning up

Eventually, you may want to clean out the cache of images and the history of containers, to reclaim space or just keep things tidy. Cleaning up involves two steps, removing the containers that you've run first, then removing the images themselves.

To remove the containers, including those that have exited and are still in the cache, use **docker ps –all** to get the ids, then **docker rm** with the container ID or the container name:

```
> docker ps --all
CONTAINER ID        IMAGE               COMMAND                  CREATED
↪STATUS                      PORTS               NAMES
48a2dca14407        nginx               "nginx -g 'daemon off"   10 minutes ago
↪Exited (0) 4 minutes ago                        pensive_booth
1bf15ca4ba89        nginx               "nginx -g 'daemon off"   20 minutes ago
↪Exited (0) 16 minutes ago                       amazing_hugle
97b1e86df1f1        ubuntu              "/bin/bash"              37 minutes ago
↪Exited (0) 37 minutes ago                       backstabbing_brown
1688f55c3418        ubuntu              "/bin/bash"              37 minutes ago
↪Exited (127) 20 minutes ago                     ecstatic_hugle
c69d6f8d89bd        ubuntu              "/bin/bash"              41 minutes ago
↪Exited (0) 41 minutes ago                       sad_keller
960588723c36        ubuntu              "/bin/echo 'hello wor"   45 minutes ago
↪Exited (0) 45 minutes ago                       suspicious_mestorf
ffbb0f60bda6        ubuntu              "/bin/echo 'hello wor"   51 minutes ago
↪Exited (0) 51 minutes ago                       mad_booth
> docker rm ffbb0f60bda6 960588723c36 # cleaning by ID
ffbb0f60bda6
960588723c36
> docker rm sad_keller ecstatic_hugle # cleaning by name
sad_keller
```

(continues on next page)

(continued from previous page)

```
ecstatic_hugle
> docker ps --all
CONTAINER ID        IMAGE               COMMAND                  CREATED
↪STATUS                      PORTS               NAMES
48a2dca14407        nginx               "nginx -g 'daemon off"   23 minutes ago
↪Exited (0) 17 minutes ago                       pensive_booth
1bf15ca4ba89        nginx               "nginx -g 'daemon off"   33 minutes ago
↪Exited (0) 29 minutes ago                       amazing_hugle
97b1e86df1f1        ubuntu              "/bin/bash"              50 minutes ago
↪Exited (0) 50 minutes ago                       backstabbing_brown
```

You can concoct your own one-liner to clean up everything, if you wish. Docker will refuse to delete something that's actively in use, so you won't screw things up too badly this way

```
> docker rm `docker ps --all -q`
48a2dca14407
1bf15ca4ba89
97b1e86df1f1
```

Now that we've removed the containers, let's clean up the images they came from. This uses the **docker images** and **docker rmi** commands, in a similar manner:

```
> docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              4ca3a192ff2a        22 hours ago        128.2␣
↪MB
nginx               latest              abf312888d13        2 days ago          181.5␣
↪MB
> docker rmi ubuntu nginx
Untagged: ubuntu:latest
Untagged:␣
↪ubuntu@sha256:3b64c309deae7ab0f7dbdd42b6b326261ccd6261da5d883964393531627O3fb5
Deleted: sha256:4ca3a192ff2a5b7e225e81dc006b6379c10776ed3619757a65608cb72de0a7f6
Deleted: sha256:2c2e0ef08d4988122fddadfe7e84d7b0aae246e6fa805bb6380892a325bc5216
Deleted: sha256:918fe8ae141d41d7430eb887e57a21d76fb0181317ec4a68f7abbd17caab034a
Deleted: sha256:00f434fa2fa1a0558f8740af28aef3d6ee546a8758c0a37dddee3f65b5290e7a
Deleted: sha256:f9545ee77b4b95c887dbebc6d08325be354274423112b0b66f7288a2cf7905cb
Deleted: sha256:d29d52f94ad5aa750bd76d24effaf6aeec487d530e262597763e56065a06ee67
Untagged: nginx:latest
Untagged:␣
↪nginx@sha256:3861a20a81e4ba699859fe0724dc6afb2ce82d21cd1ddc27fff6ec76e4c2824e
Deleted: sha256:abf312888d132e461c61484457ee9fd0125d666672e22f972f3b8c9a0ed3f0a1
Deleted: sha256:21976f5d6f037350c076d6974b2ac97777c962869d5511855170dc5c926d5aac
Deleted: sha256:f9719c3716279b47727a51595d1482506be469c581383cb529f4005ba9bf2aeb
Deleted: sha256:fe4c16cbf7a4c70a5462654cf2c8f9f69778db280f235229bd98cf8784e878e4
```

8.13.3 Conclusion

Cleaning up containers and images is a two-step process. Now you should be able to keep your system tidy.

8.13.4 Best practices

- use the `--rm` flag when you know you won't want to re-start a container
- if you use containers heavily, clean up the images from time to time

8.14 Gitlab exercises

You can use either **gitlab.com** or **gitlab.ebi.ac.uk**, if you have a login there. Both run the Enterprise Edition of gitlab. The EBI instance requires two-factor authentication (2FA), which complicates things. You'll need an accessory app that can generate authentication tokens on your personal devices (phone, tablet, laptop). If you want to generate tokens on your laptop, Authy is a good choice.

To make life easier for these exercises, we're going to use **gitlab.com** in all the examples. Once you set up 2FA on **gitlab.ebi.ac.uk** you should be able to run there too, just as easily. You'll have to change a few hostnames, but that's all.

Also, you can run these exercises on your laptop, or on the virtual machine we provide for the course. We recommend using the virtual machine for consistency, but it's your choice.

There are several separate exercises, you can follow them in sequence or, if you prefer, jump straight to the exercise that interests you. For the most part, they're in the form of annotated command-line sessions. You can copy/paste the commands straight from the screen to your terminal, but you'll have to be aware of things that change - user names, process IDs etc.

The easy way to get access to the example files used in the exercises is to clone this repository from the command line:

```
# Clone this documentation if you haven't already done so
> cd
> git clone https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc.git
> cd tsi-ccdoc/tsi-cc/ResOps/scripts/gitlab
```

The exercises are:

- Log into gitlab, set up your account
- Create a new project and import the example code
- Download and run the docker image from your gitlab project
- Use git tags to create a named version of a docker image
- Extend the pipeline by adding further steps
- Change the order of the pipeline steps
- Pass secret information to the build pipeline
- Gitlab 101 Tool Certification.
- AutoDevOps Tutorial.

8.15 Exercise 1: Log into gitlab, set up your SSH key

8.15.1 Objective

Get your account ready for you to use it.

8.15.2 Introduction

Before you can make use of your gitlab account, there are a few things you should do to make it more useful:

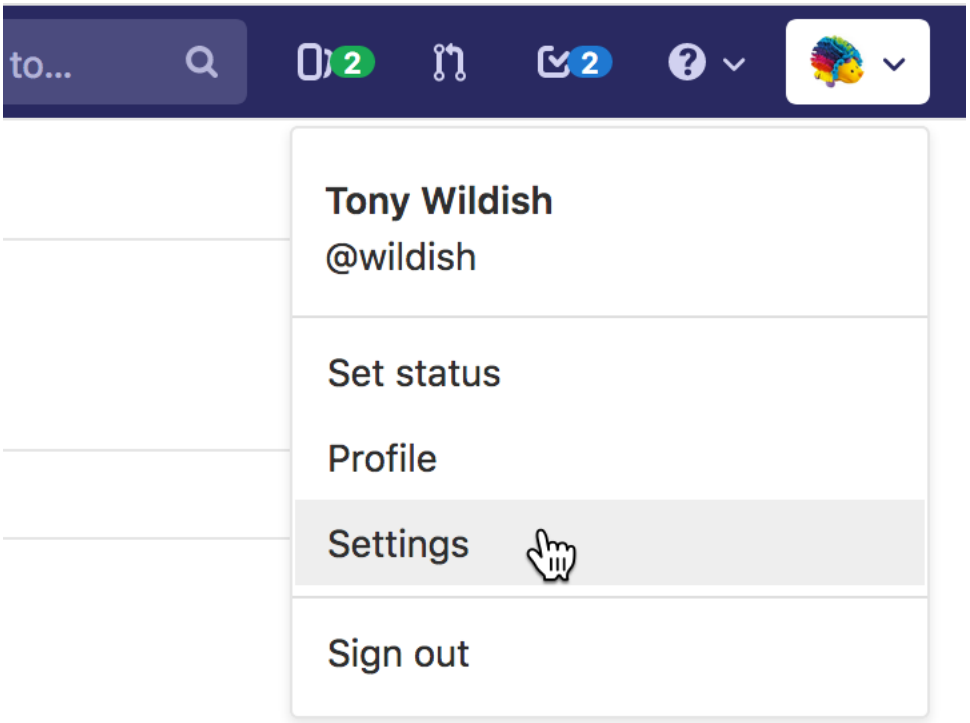
- create a Personal Access Token

  - this is mandatory on **gitlab.ebi.ac.uk** because of 2FA, but it also works on **gitlab.com**
- upload your SSH key (optional)
- configure email notifications (optional)

8.15.3 Create a Personal Access Token

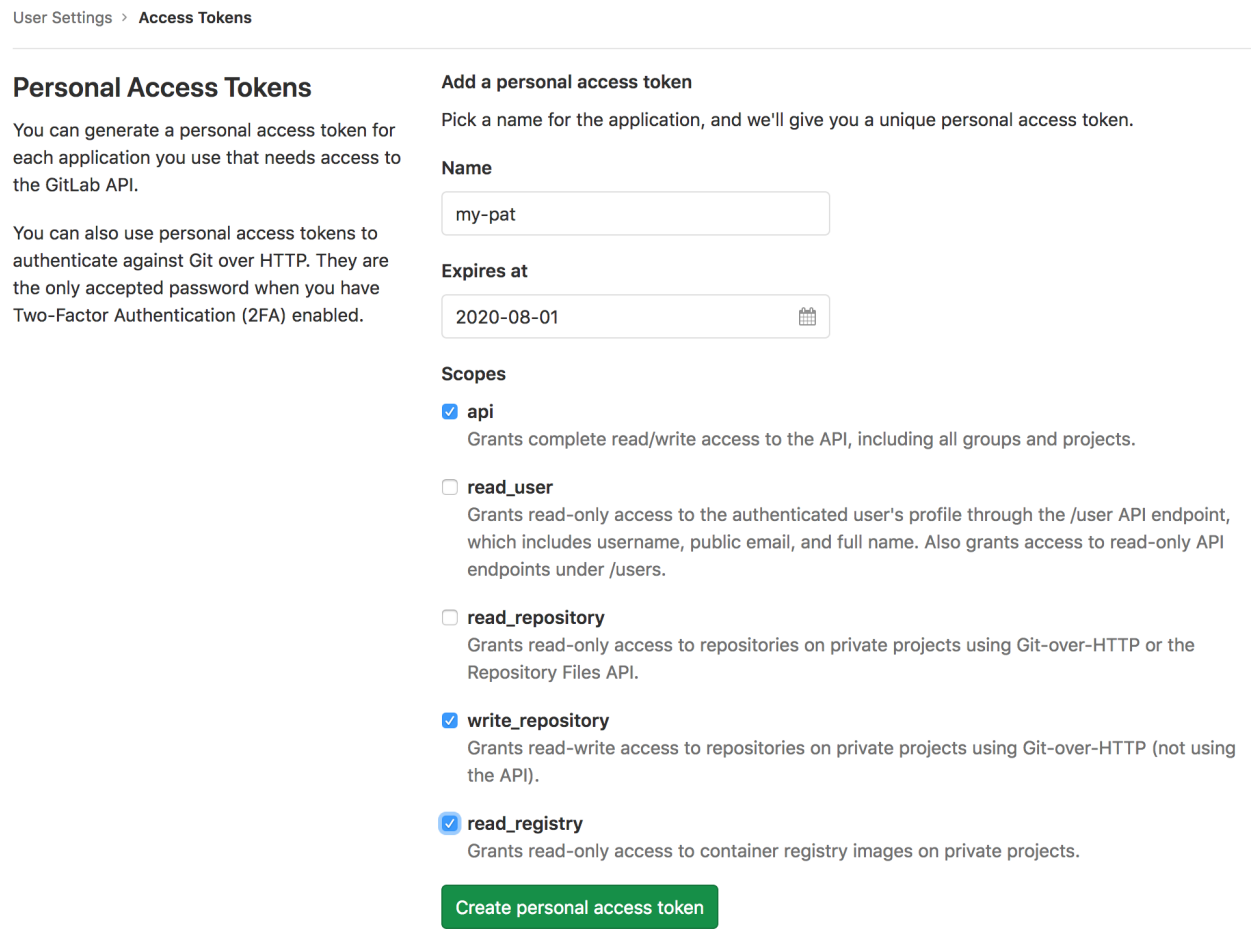
With 2FA, you can't use your normal password to authenticate a `git push`, instead you need to use a **Personal Access Token**, which you create in the gitlab website.

Click on your username or avatar in the top right-hand corner, and select **Settings**

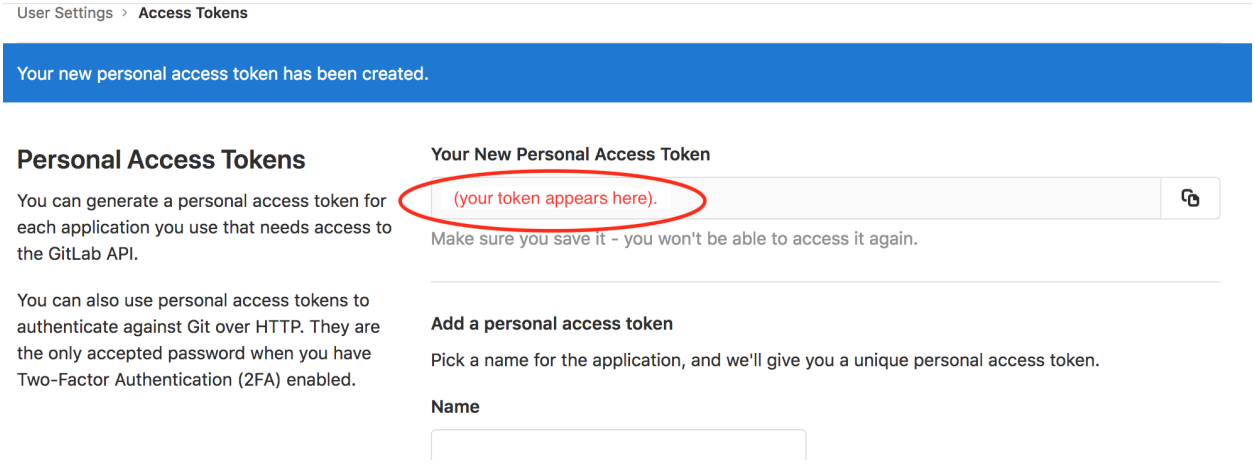


Click on **Access Tokens** on the navigation bar on the left, then fill in the form

- give your token a name, e.g. **my-pat**
- set an expiry date. One year in the future is a good value to use
- select **api**, **write_repository** and **read_registry** for the **scope**
- Then click the **Create personal access token** button



Make a note of the access token you see on the next screen, keep it somewhere safe. This is almost as powerful as your password, and should be treated with the same level of caution. If you lose it, you won't be able to recover it from the web interface, but you can create another one easily enough.



N.B. It's good hygiene to create separate tokens for separate purposes. That way you can revoke the access token for one use-case, if needed, without affecting others. Don't share tokens either, the token belongs to *you*, not to individual projects, so giving it to someone gives them access to all your projects, not just to any one project.

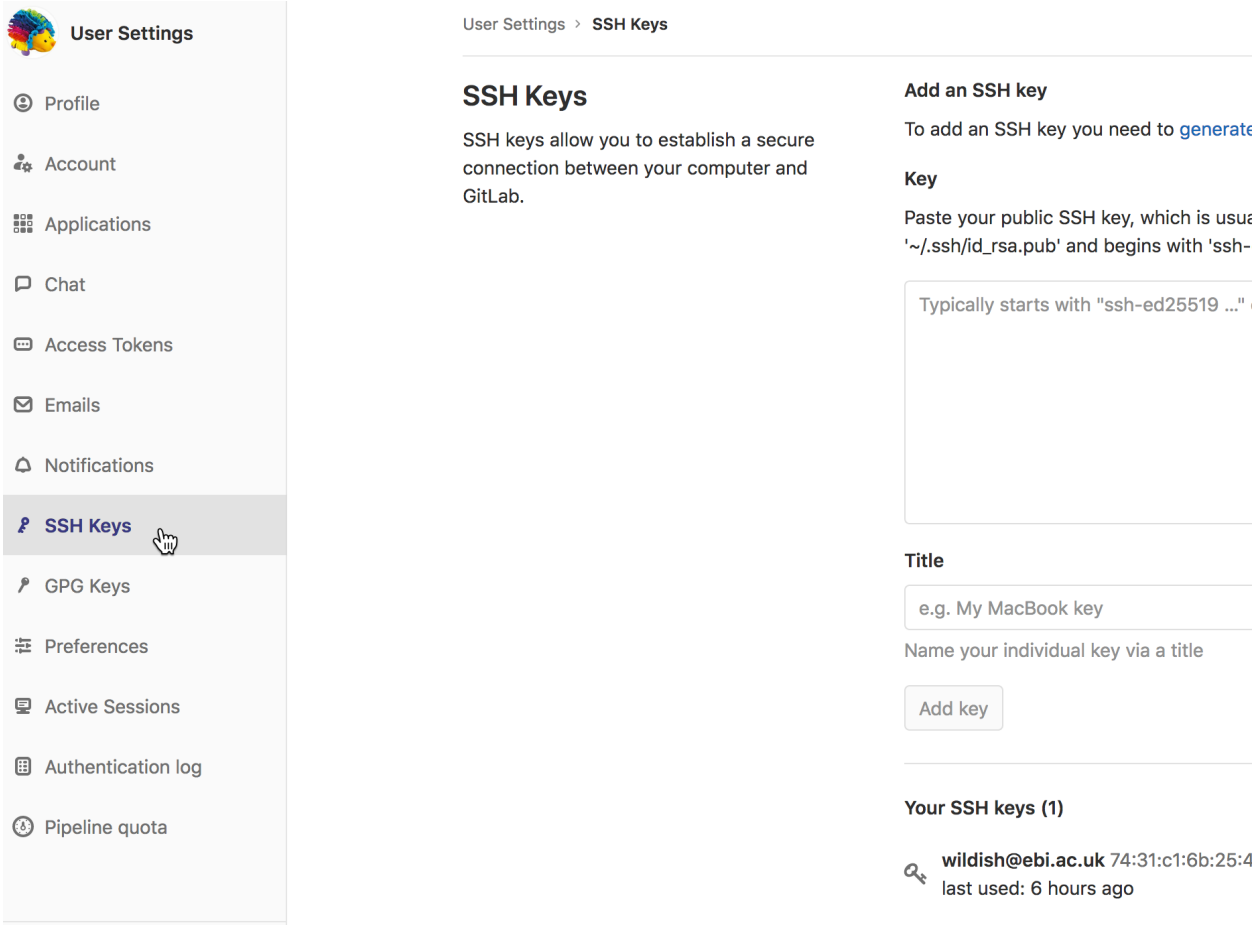
It's also important to keep track of your token expiry dates. If you're using one in a production environment for some purpose then you'll want to renew it before it expires.

8.15.4 Upload your SSH key (optional!)

This is another useful way to help you avoid having to type in your password every time you commit code to your projects. However, SSH access to gitlab.ebi.ac.uk is only available via the EBI network, not from outside, so its usefulness is a bit limited. Also, you will need to know how to use an SSH agent to cache your password. Google has a number of tutorials on that. We won't be using the SSH protocol in these exercises, to ensure that they work the same on gitlab.com and gitlab.ebi.ac.uk, so this step is optional.

Click on your username or avatar in the top right-hand corner, and select **Settings**

Click on **SSH Keys** on the navigation bar on the left



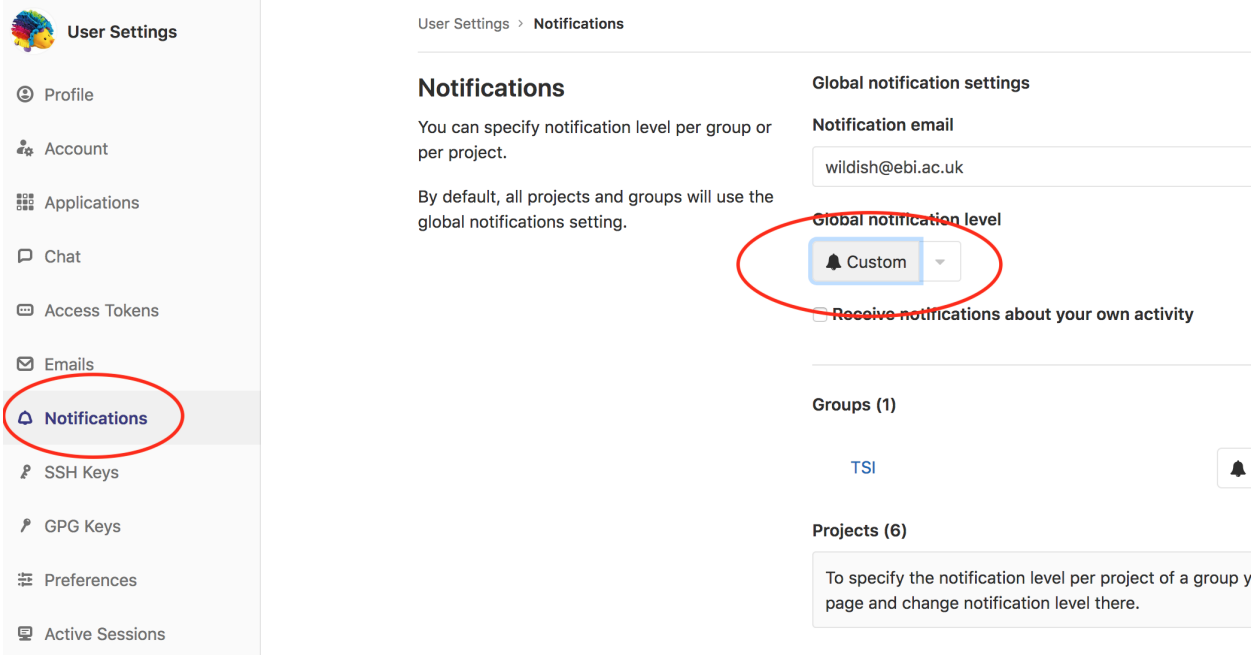
Follow the instructions on the right to upload your public SSH key. Not your private key, only the public one!

8.15.5 Configure email notifications (optional!)

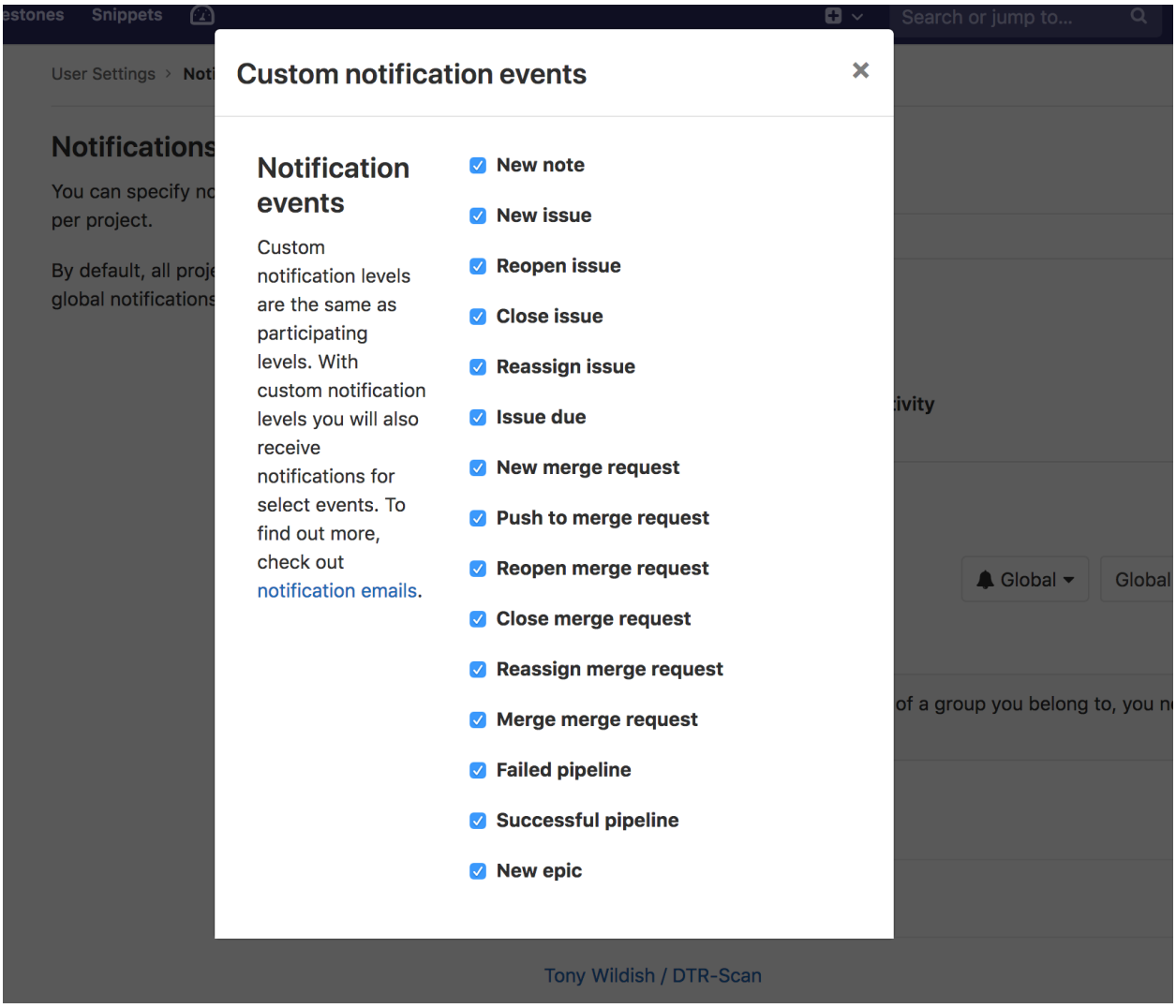
You can control the email notifications you get from gitlab in great detail. Here's how to customise them:

Click on your username or avatar in the top right-hand corner, and select **Settings**

Click on **Notifications** on the navigation bar on the left, then on the button under **Global notification level**



Check any/all boxes that might interest you. I've checked all mine, and will uncheck as/when I decide I don't want to get notifications anymore.



Elsewhere on the page you'll see that you can choose at the Group or Project level which set of notification presets you want for that group or project.

That's it, you don't need to save the options, they're saved automatically.

8.15.6 Conclusion

You should now be able to push to your gitlab repositories without having to log in with every commit, exactly as with github and other hosting services.

You will get notification by email for any interesting events in your repositories, and can tune that as needed to control the noise.

8.15.7 Best Practices

Ideally you should use a different SSH key for every service (gitlab, github, bitbucket, whatever) that requires you to register with a key. This limits the damage if ever one of those keys is compromised.

Likewise, use different access tokens for different purposes, with only the level of access required.

8.16 Exercise 2: Create a new project

8.16.1 Objective

Create a new project and import the sample code, see the build process it triggers.

8.16.2 Create the project and upload the code, watch the build fail

- In the browser, click on **Projects -> Your projects**, from the top-left of the window.
- Click 'New Project', on the top-right
  - if you don't have any projects yet, there will be a big 'Create a project' button on the left instead
- Give the project a name, e.g. 'gitlab-101' or 'my-test-project'
  - add a description if you like, e.g. a reference to this tutorial
  - set the project visibility to 'public'
  - **do not* click the 'Initialize repository with README' button
- Click 'Create Project'
  - that's all you need to do in this browser window, you don't need to follow the instructions you see on-screen next, instead do what it says below.

Create an empty directory, go into it, and create an empty git repository there:

```
mkdir ~/gitlab-test
cd ~/gitlab-test
git init
```

Add the code to your repository and commit it:

```
tar xvf ~/tsi-ccdoc/tsi-cc/ResOps/scripts/gitlab/tiny-test.tar

# replace gitlab.com with gitlab.ebi.ac.uk if you're using the EBI gitlab
git remote add origin https://gitlab.com/YOUR_GITLAB_USERNAME/YOUR_PROJECT_NAME.git

git add .
git commit -m "Initial commit"
git push -u origin master
```

You will be prompted for your usename and password.

Then go to **Projects -> Your Projects** and click on your project. Go to the **CI/CD -> Pipelines** tab and watch the progress of your pipeline. Unfortunately it will fail!

8.16.3 Fix the build and try again

The build fails because the code which is uploaded points to my personal project, which only I have access to.

To fix this, edit the **.gitlab-ci.yml** file, change the **APPLICATION** name to the name of your project, and the **REGISTRY_USER** to your gitlab username. Save the file, commit it to git, and push it to the server again. This time, you only need a `git push`, with no extra arguments.

Go to the **CI/CD -> Pipelines** tab again, this time it will succeed.

8.16.4 Conclusion

You now know how to create a project in gitlab, how to upload code to it from an existing area on your computer, and how to configure the basic features of the CI/CD pipeline to refer to this project correctly.

8.16.5 Best Practices

Keep your projects small, it's better to have many small projects than a few large ones with everything bundled in together. It keeps the CI/CD pipeline cleaner and faster, as well as being good coding hygiene in general.

If you have lots of projects, shared between many developers, you can look into using gitlab **Groups** to organise them.

8.17 Exercise 3: Download and run the docker image from your gitlab project

8.17.1 Objective

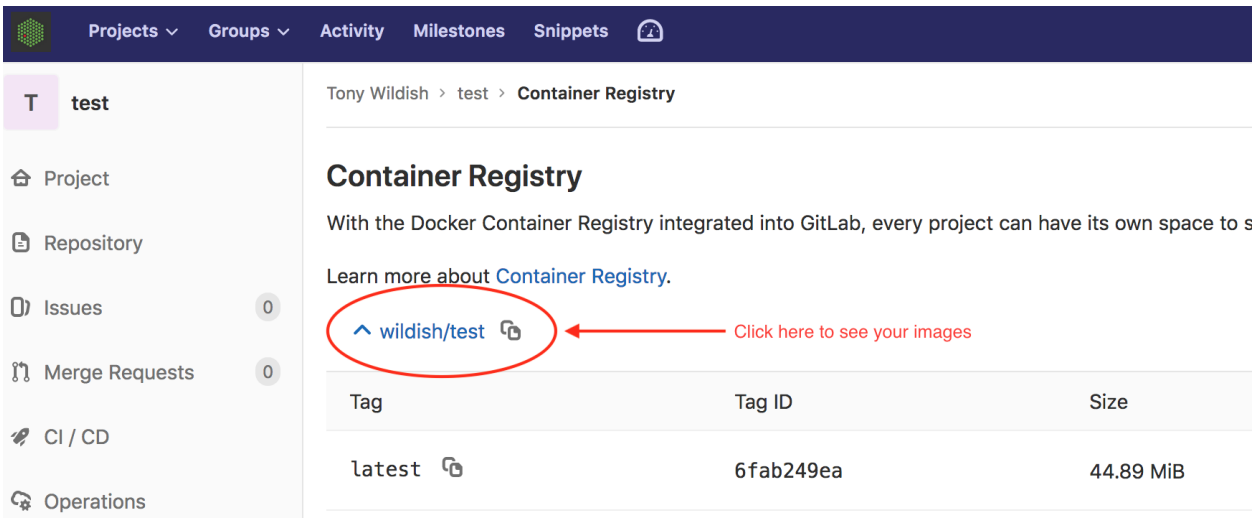
Learn how to download and run docker images built from your CI/CD pipelines.

8.17.2 Introduction

If your pipeline ran to completion in the previous exercise, you can now go to the **Packages** tab on your project page, then to **Container Registry**.

If you're on gitlab.ebi.ac.uk, the interface is different (as of 2019-07-07), and you can just go to the **Registry** tab on the project page directly.

Look near the top-left of the page, where you will see a link with your gitlab username and the name of the project in it. Click on that, and you'll see the list of images you've built so far. There will be only one image, tagged **latest**, with a size of about 45 MB:



How you run the image depends on wether or not you made your project public. If your project is private, you'll need to log into the docker registry first, giving your username and Personal Access Token when prompted:

```
docker login registry.gitlab.com
```

You only need to login once per machine, the credentials are cached.

If you're doing the exercises on the EBI gitlab instance, use **dockerhub.ebi.ac.uk** instead of **registry.gitlab.com**

Then simply run the image - note that you'll have to specify your own username and the name of your project, of course:

```
> docker run registry.gitlab.com/tonywildish/tiny-test
Unable to find image 'registry.gitlab.com/tonywildish/tiny-test:latest' locally
latest: Pulling from tonywildish/tiny-test
7413c47ba209: Pull complete
0fe7e7cbb2e8: Pull complete
1d425c982345: Pull complete
344da5c95cec: Pull complete
7e8d808bd962: Pull complete
1b5e91f7df4f: Pull complete
Digest: sha256:a1fc0a6209ac87cc8a009e8e2bee2cbfb528f246e2fd174d15a741efe7433bf6
Status: Downloaded newer image for registry.gitlab.com/tonywildish/tiny-test:latest
Hello World
Compiled on Tue Jul 30 09:54:56 UTC 2019
```

Docker supports multiple levels of namespacing within a repository, so it's possible to build more than one image from a given code base if you want to. Check the instructions on the Registry page for how to do that. This can be useful if you need it, but you should always consider if it's better to have multiple separate projects instead of a single project that's building many containers.

Now that you're logged into the registry you can even push images there by hand, just like on dockerhub.com. That, of course, rather defeats the point of having an automatic CI/CD pipeline to do it for you!

8.17.3 Deploy tokens

If you want to run this docker image in a batch script, or a web service, you won't want to have to log into the registry on every machine you use, or every time it gets rebooted. Instead, if you only want to *use* the images, but not to *upload* them from the command line, you want a way to have read-only access to the registry. You can make the registry public by making the project public, which works just fine, but there may be reasons you don't want to do that.

Gitlab provides **deploy-tokens** to address this problem. Deploy tokens are basically the same as the personal access token you've already created in the first exercise. For running batch jobs using containers from a private project registry, you should create a deploy token, giving it only **read_registry** access, nothing else. You can then safely publish this anywhere you like, the worst people can do with it is to download your images.

To create a deploy token, go to the dashboard for your project (deploy tokens are project-specific, unlike personal access tokens). Then go to **Settings** -> **Repository** and scroll down until you find **Deploy Tokens**. You can allow access to just the registry, just the code repository, or both, as you see fit.

You'll need to make a note of the expiry date of your deploy tokens, and set a reminder to create a new one well in advance if you're running in a production environment. Alternatively, if your containers are fit for public release, you can push them to a public service like dockerhub.com, either as well as or instead of using the gitlab registry.

8.17.4 Conclusion

You now know how to inspect the docker registry associated with your gitlab account, how to log into it from the command line, and how to download and run images from it. By using a deploy-token, you can now use even private docker images in your pipelines.

8.17.5 Best Practices

- Use the namespace hierarchy if you need to build multiple docker images from the same code base, use separate projects if the code bases are not related.
- Use deploy tokens when you only need read-only access to private images.

8.18 Exercise 4: Use git tags to create a named version of a docker image

8.18.1 Objective

Use git tags to create automatically labelled versions of docker images.

See how to use git tags to add extra steps to the CI/CD pipeline, for more detailed testing and verification of your code.

8.18.2 Using the git tag to tag the docker image

Gitlab directly supports specifying that certain stages will run only for git branches other than the 'master' branch, or only if the code has been tagged with 'git tag'. By using a combination of YAML directives and the environment variables that gitlab provides to the build, you can build a docker image which is tagged with the same name as your git tag or branch.

Once the **.gitlab-ci.yml** file has been correctly set up, creating a tagged docker image is as simple as this:

```
git tag v1.0
git push --tags
```

While you're waiting for the build to complete, inspect the **.gitlab-ci.yml** file to see how it uses the tag. There are a few places where the magic happens:

- the **variables** section defines two variables: **RELEASE_IMAGE** and **LATEST_IMAGE**, differing only in that one uses the **CI_BUILD_REF_NAME** environment variable, the other hardwires the name **latest**
- the **before_script** stanza sets **DOCKER_IMAGE** to one or the other of these variables, depending on the value of **CI_BUILD_REF_NAME**. For the **master** branch, **CI_BUILD_REF_NAME** will be **master**, this bit of code effectively changes that to **latest** instead.
- in the **install** step, the **DOCKER_IMAGE** variable is used to actually tag the image that's pushed to the registry

Most of the complication there comes from the fact that the default branch name in git is **master**, while the default tag name in docker is **latest**. We could remove the **before_script** part and just use the **RELEASE_IMAGE** variable everywhere, but then we'd end up with docker images tagged **master**, which isn't nice.

8.18.3 Using the git tag to control the build

If you look at the **CICD -> Pipelines** page after your tagged build has completed, you'll see it has three steps to it. There is a **run** step that executes the **test** stage, this isn't executed unless the code has been tagged in git. The **only:** directive in the **.gitlab-ci.yml** file controls that behaviour.

Since you are likely to commit code much more often than you are to tag it, you can see how this can be used to test tagged code in-depth, in ways that might be wasteful of resources if you tested every single commit to the same level.

8.18.4 Check your Registry to see the named docker image

Once the build completes, go to your Registry page again, and click on your project link at the top-left to see what images you have. You now have a **latest** and a **v1.0** version of your docker image. Check you can run that tagged version from the command line:

```
> docker run registry.gitlab.com/tonywildish/tiny-test:v1.0
Unable to find image 'registry.gitlab.com/tonywildish/tiny-test:v1.0' locally
v1.0: Pulling from tonywildish/tiny-test
7413c47ba209: Already exists
0fe7e7cbb2e8: Already exists
1d425c982345: Already exists
344da5c95cec: Already exists
69e5cbbf5881: Pull complete
737987b4e0ef: Pull complete
Digest: sha256:e1445aff15c845b2f795fc49f8b2bcb41f34cd4a6a3c3eef98bb61bea26a986b
Status: Downloaded newer image for registry.gitlab.com/tonywildish/tiny-test:v1.0
Hello World
Compiled on Tue Jul 30 10:58:09 UTC 2019
```

8.18.5 Bonus exercise: git branches

What happens if you use a git branch, instead of a tag? The CI/CD pipeline will still fire, and will know the branch name, so it uses that in the docker image tag. However, since your **.gitlab-ci.yml** file only runs the **test** stage for git *tags* and not for git *branches*, you will only get a two-stage pipeline.

Can you modify the YAML file so it runs for branches too? Check the documentation for help.

If you're on a branch, and you use a tag too, what do you think gitlab will do? Try it and find out.

8.18.6 Conclusion

You now know how to control the activities in a build when using git tags, and how to use those tags to automatically tag the docker image with the same name.

8.18.7 Best Practices

- Use git tags to define important versions of your code.
- Use the **only:** directive in your builds to make sure that tagged code is tested thoroughly.

8.19 Exercise 5: Extend the pipeline by adding further steps

8.19.1 Objective

Add another step to the **.gitlab-ci.yml** file to test the docker image itself, not just the executable.

8.19.2 Introduction

The **test** stage in the **.gitlab-ci.yml** file runs the **hello** executable directly, to check that it works. However, we then build and upload a docker image for it, but we don't test the image. We can do that, to make sure it works correctly.

- in the **stages** section, add a new stage, **test-docker**
- below, add a new stanza for the test-docker stage.
  - give this step a meaningful name, e.g. **run-docker**, configure it to run the **test-docker** stage.
  - this will need to use the latest docker image and the docker-in-docker service, so make sure they're declared.
  - you also need to set the dependencies correctly, so it has whichever artifacts it needs to download. (Hint: I'm misleading you here!)
  - then edit the script part, so that it does the docker login, and then just runs the container, like you did previously.
- commit your code to the repository, add a new git tag for it, and push it, with the tag, to the server:

```
git add .
git commit -m "Add new section to test the docker image"
git tag v2.0
git push --tags
```

If you're getting YAML errors, go to **CI/CD -> Pipelines** and look for the **CI Lint** button, top-right. There you can paste in the contents of your **.gitlab-ci.yml** file, click **Validate**, and see where it reports an error.

You can even edit in that window to try to correct the error, then when you've solved the problem, you can update your local file accordingly.

8.19.3 Conclusion

You now know how to add stages to your pipeline so you can implement complex builds and tests.

8.19.4 Best Practices

- Each stage in your pipeline should perform a well-specified action, with a clear definition of success and failure. Don't make your builds any more complex than necessary.
- If your build is unavoidably complex, consider coding the build script as an external script, which you debug separately, rather than hand-coding it in the **.gitlab-ci.yml** file. This keeps the YAML file cleaner, at the expense of making it harder to see the details of what it's actually doing.

8.19.5 Cheat-sheet

In case you get stuck, this is what your **.gitlab-ci.yml** file should now look like. Don't peek until you've had a go yourself though!

```
> cat .gitlab-ci.yml
variables:
  DOCKER_TLS_CERTDIR: ""
  GIT_STRATEGY: clone
  REGISTRY_USER: tonywildish
  APPLICATION: tiny-test
  LATEST_IMAGE: $CI_REGISTRY/$REGISTRY_USER/$APPLICATION:latest
  RELEASE_IMAGE: $CI_REGISTRY/$REGISTRY_USER/$APPLICATION:$CI_BUILD_REF_NAME
  DOCKER_DRIVER: overlay
```

(continues on next page)

(continued from previous page)

```
before_script:
  - echo "Starting..."
  - export DOCKER_IMAGE=$RELEASE_IMAGE
  - if [ "$CI_BUILD_REF_NAME" == "master" ]; then export DOCKER_IMAGE=$LATEST_IMAGE;␣
↪fi
  - echo "Build docker image $DOCKER_IMAGE"

stages:
  - build
  - test
  - deploy
  - test-docker

compile:
  stage: build
  image: gcc:6
  services:
    - docker:dind
  artifacts:
    name: "${CI_BUILD_NAME}_${CI_BUILD_REF_NAME}"
    untracked: true
    expire_in: 1 week
  script:
    - make

run:
  stage: test
  dependencies:
  - compile
  only:
    - tags
  script:
    - echo "Testing application. First, list the files here, to show we have the git␣
↪repo + the artifacts from the build step"
    - ls -l
    - echo 'Now try running it'
    - ./hello
    - echo "If that failed you won't see this because you'll have died already"

install:
  stage: deploy
  image: docker:latest
  services:
  - docker:dind
  dependencies:
  - compile
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER --password-stdin
↪$CI_REGISTRY
    - echo Building $DOCKER_IMAGE
    - docker build -t $DOCKER_IMAGE .
    - echo Deploying $DOCKER_IMAGE
    - docker push $DOCKER_IMAGE

run-docker:
  stage: test-docker
  image: docker:latest
```

(continues on next page)

(continued from previous page)

```
  services:
  - docker:dind
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER --password-stdin
↪$CI_REGISTRY
    - docker run $DOCKER_IMAGE

after_script:
  - echo "Congratulations, this step succeeded"
```

8.20 Exercise 6: Change the order of the pipeline steps

8.20.1 Objective

Learn how to change the order of pipeline stages, and how to build a parallel pipeline

8.20.2 Change the order of stages

Build stages have dependencies, which give them access to the artifacts from other stages, but the order is determined by the **stages:** declaration in the **.gitlab-ci.yml** file. After the last exercise, your **stages:** section will look something like this:

```
> cat .gitlab-ci.yml | grep --after-context=5 --before-context=1 stages

stages:
  - build
  - test
  - deploy
  - test-docker
```

If you simply swap the order of the **test** and **deploy** stages, commit, tag and push your code, you should see that the build pipeline runs in a different order, with the **deploy** (create and push the docker image) happening before the **test** (running the executable).

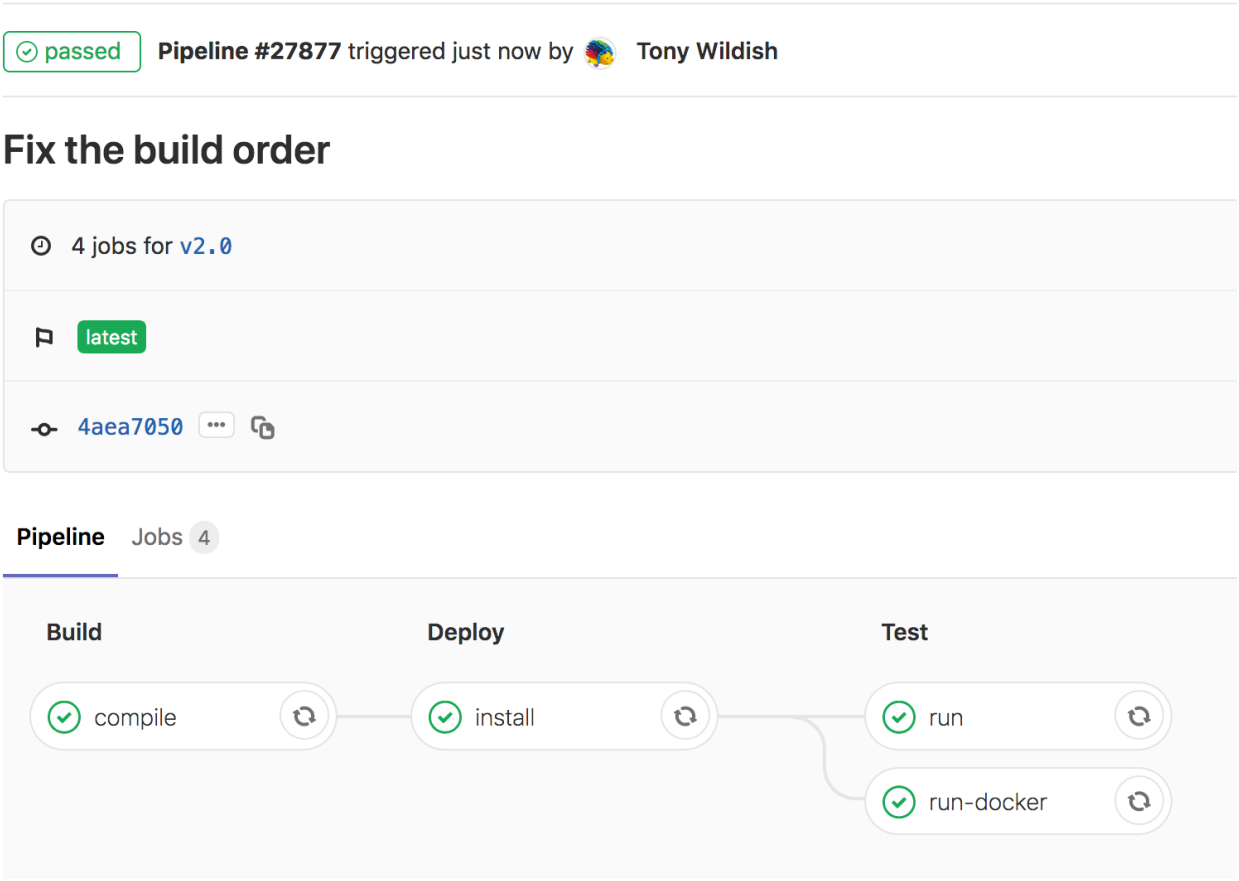
It's up to you to decide if it makes sense to 'deploy' before testing, or after, this simply illustrates how to do it.

8.20.3 Sequential and parallel builds

In our example, testing the executable and testing the docker image could, in principle, happen in parallel. If the tests were long-running, and we have enough resources in our gitlab runners, this can save time. We can do this very simply by removing the **test-docker** stage from the **stages:** piece and changing the **run-docker** step to execute the **test** stage, instead of having it execute the **test-docker** stage.

In other words, there will be two steps, **run** and **run-docker**, both of which run the **test** stage. They will happen in the correct order, after the **deploy** stage, but they are eligible to run in parallel if there are sufficient resources. If there are not sufficient resources for the steps to run in parallel, they will simply happen sequentially, in a random order.

Make this change to your YAML file, commit it, tag it, and push it to the server. Take a look at the pipeline page and see how it shows the parallel steps. It should look like this:



8.20.4 Conclusion

- You now know how to write pipelines which can run stages sequentially, or in parallel.
- Each **stage** of a pipeline can have many **steps** that run that stage.
- Each **step** in a stage must succeed, or the whole pipeline fails
- Each **step** in a stage must complete before the next stage is free to start
- Parallel stages can speed up testing, e.g. by running tests in many environments at the same time

8.20.5 Best Practices

- Think carefully about what you are trying to achieve in your pipeline. Keep the steps small, and parallelise where it makes sense.

8.20.6 Cheat-sheet

In case you get stuck, this is what your **.gitlab-ci.yml** file should now look like. Don't peek until you've had a go yourself though!

```
> cat .gitlab-i.yml
variables:
```

(continues on next page)

(continued from previous page)

```
  DOCKER_TLS_CERTDIR: ""
  GIT_STRATEGY: clone
  REGISTRY_USER: tonywildish
  APPLICATION: tiny-test
  LATEST_IMAGE: $CI_REGISTRY/$REGISTRY_USER/$APPLICATION:latest
  RELEASE_IMAGE: $CI_REGISTRY/$REGISTRY_USER/$APPLICATION:$CI_BUILD_REF_NAME
  DOCKER_DRIVER: overlay

before_script:
  - echo "Starting..."
  - export DOCKER_IMAGE=$RELEASE_IMAGE
  - if [ "$CI_BUILD_REF_NAME" == "master" ]; then export DOCKER_IMAGE=$LATEST_IMAGE;␣
↪fi
  - echo "Build docker image $DOCKER_IMAGE"

stages:
  - build
  - deploy
  - test

compile:
  stage: build
  image: gcc:6
  services:
    - docker:dind
  artifacts:
    name: "${CI_BUILD_NAME}_${CI_BUILD_REF_NAME}"
    untracked: true
    expire_in: 1 week
  script:
    - make

run:
  stage: test
  dependencies:
  - compile
  only:
    - tags
  script:
    - echo "Testing application. First, list the files here, to show we have the git␣
↪repo + the artifacts from the build step"
    - ls -l
    - echo 'Now try running it'
    - ./hello
    - echo "If that failed you won't see this because you'll have died already"

install:
  stage: deploy
  image: docker:latest
  services:
  - docker:dind
  dependencies:
  - compile
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER --password-stdin
↪$CI_REGISTRY
    - echo Building $DOCKER_IMAGE
```

(continues on next page)

(continued from previous page)

```
    - docker build -t $DOCKER_IMAGE .
    - echo Deploying $DOCKER_IMAGE
    - docker push $DOCKER_IMAGE

run-docker:
  stage: test
  image: docker:latest
  services:
  - docker:dind
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER --password-stdin
↪$CI_REGISTRY
    - docker run $DOCKER_IMAGE

after_script:
  - echo "Congratulations, this step succeeded"
```

8.21 Exercise 7: Pass secrets to the build pipeline

8.21.1 Objective

Learn how to pass confidential information to the build process

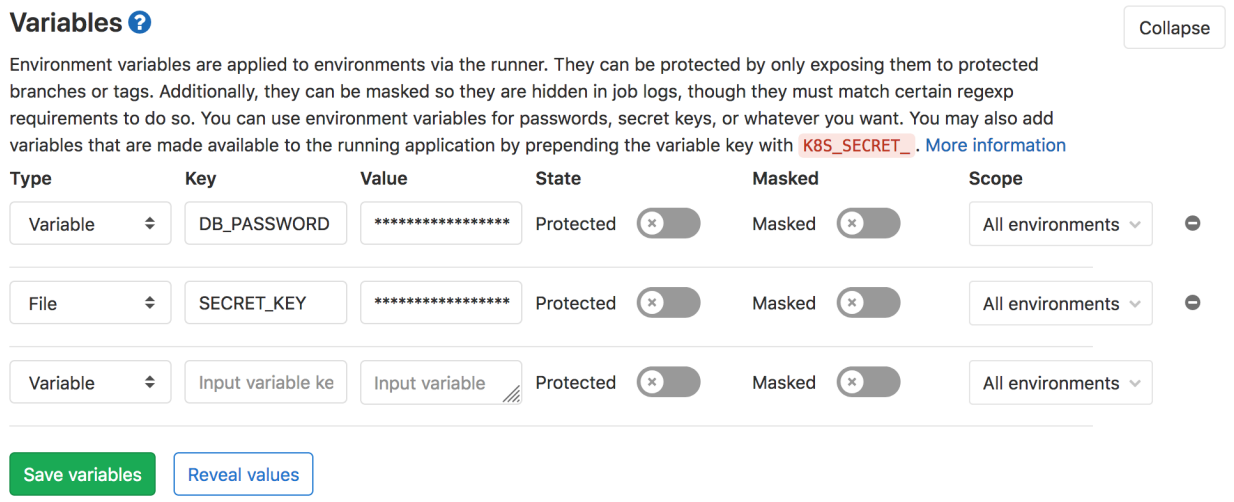
8.21.2 Gitlab Variables

Suppose you're building an application that needs access to a database, and you want to test that access from your CI/CD pipeline. You need to pass the name of the database host, the account, and password, to the running pipeline.

You could save them in a file in your git repository, but that makes them visible to anyone who can view the repository. Not a good idea, to say the least!

An alternative is to use **Gitlab Variables**. Go to your project page, **Settings** tab -> **CI/CD**, find **Variables** and click on the **Expand** button. Here you can define variable names and values, which will be automatically passed into the gitlab pipelines, and are available as environment variables there.

There are two types of variable you can define: **Variable** and **File**, as you can see in this example:



DB_PASSWORD has been declared as a **Variable**, and given a value. This means that you can use **$DB_PASSWORD** directly in your pipelines as it is.

SECRET_KEY has been defined as a **File**. This means that the value of **SECRET_KEY**, in your pipeline, will be a randomly-generated filename. The *contents* of that file will be the **value** that you supply for that key.

Go ahead and add two variables like this, one a variable and one a gile. Give them arbitrary values, but do *not* check the **Masked** button for either. Click **Save variables** at the bottom to save them.

Then edit your .gitlab-ci.yml file, and, in the **before_script** section, add some lines to print those variables to the screen (i.e. to the logfile). Something like this:

```
before_script:
  - echo "Starting..."
  - export DOCKER_IMAGE=$RELEASE_IMAGE
  - if [ "$CI_BUILD_REF_NAME" == "master" ]; then export DOCKER_IMAGE=$LATEST_IMAGE;␣
↪fi
  - echo "Build docker image $DOCKER_IMAGE"
  - echo DB_PASSWORD is $DB_PASSWORD
  - echo SECRET_KEY is $SECRET_KEY
  - echo Contents of the SECRET_KEY file are
  - cat $SECRET_KEY
```

Commit your file to git and push it to the server. Check the build log for the first stage and you should see how it all works, as below:

```
$ echo "Build docker image $DOCKER_IMAGE"

Build docker image registry.gitlab.com/tonywildish/tiny-test:latest
$ echo DB_PASSWORD is $DB_PASSWORD

DB_PASSWORD is pa55w0rd
$ echo SECRET_KEY is $SECRET_KEY

SECRET_KEY is /builds/TonyWildish/tiny-test.tmp/SECRET_KEY
$ echo Contents of the SECRET_KEY file are

Contents of the SECRET_KEY file are
$ cat $SECRET_KEY

th1s-iz-the-qi
$ make

echo "#define TODAY \"`date`\"" | tee hello.h
#define TODAY "Wed Aug  7 14:56:25 UTC 2019"
cc hello.c -static -o hello
Running after script...
$ echo "Congratulations, this step succeeded"
```

8.21.3 Masking the variable values

It would be nice if gitlab offered a way to prevent your variables from being printed to the logfile, in case that happens by accident. Well, it does! You can **mask** them.

Go back to the **Settings** -> **CI/CD** page and expand the **Variables** section again. Set both variables to be **Masked**, remembering to click **Save variables**. Go back to **CI/CD** -> **Pipelines**, click on the top pipeline (the one that just ran). Each stage has an icon on it to re-run that stage - two curved arrows pointing at each other in a circle. Re-run the compile stage, and examine the output.

You'll see that this time, despite explicitly trying to print the secrets to the screen, gitlab has masked them out, preventing them from being displayed:

```
$ echo "Build docker image $DOCKER_IMAGE"

Build docker image registry.gitlab.com/tonywildish/tiny-test:latest
$ echo DB_PASSWORD is $DB_PASSWORD

DB_PASSWORD is [MASKED]
$ echo SECRET_KEY is $SECRET_KEY

SECRET_KEY is /builds/TonyWildish/tiny-test.tmp/SECRET_KEY
$ echo Contents of the SECRET_KEY file are

Contents of the SECRET_KEY file are
$ cat $SECRET_KEY

[MASKED]
$ make

echo "#define TODAY \"`date`\"" | tee hello.h
#define TODAY "Wed Aug  7 15:10:55 UTC 2019"
cc hello.c -static -o hello
Running after script...
$ echo "Congratulations, this step succeeded"
```

8.21.4 Taking it to the next level

Although this is already very secure, anyone who has access to the project dashboard can see these variables and inspect or change their values. You may not want that, you may want to restrict the visibility even further, so the secrets can only be seen at the time they're used, and not by everyone who has access to the project.

One way to do that is to encrypt the secret variable value with OpenSSL, and store the encrypted value as the variable value. The decryption key is *not* stored in the project, instead it's made available in the runtime environment of the gitlab runner, or of the environment where the final product is deployed. Then, either in your pipeline or in your production environment, you decrypt the encrypted secret at the point of use, and nowhere else.

An attacker will then have to gain access to your dashboard *and* the configuration of the deployment environment before they can read the secrets. Because of this, it's safe to bake the encrypted variable into the docker image itself - it's useless without the decryption key!

We don't go into the details of this here, come and talk to us if you're interested in doing this.

8.21.5 Conclusion

You can define variables in the gitlab user interface which are then passed on to the build pipeline. You can tell gitlab to prevent these variables from being printed to the logfiles. This means you can use them to store sensitive information, like database passwords etc.

8.21.6 Best Practices

- Use variables to pass sensitive information, like passwords, account names, hostnames, license keys etc into your gitlab builds, do not under any circumstances commit their values into the code repository.
- Ask yourself if you really *need* to use variables at all!
  - e.g., if you need a database, why not create one and populate it with a schema for testing as part of the CI/CD pipeline? This means you don't need to store or pass any secrets at all.
  - Alternatively, can you store secrets in the deployment environment, so they're only visible at runtime?
- Make sure you mask variables unless you are *certain* they're not sensitive. Even seemingly innocuous information like hostnames or port numbers can be used to successfully attack an infrastructure.
- For extra security, encrypt the variable values, and provide the decryption key to the pipelines and deployments by other means. Kubernetes has techniques that can help here, see the sessions later today.
- Always separate your test and production environments, don't use the same infrastructure for both.
- If you bake variables and their values into your docker images, think *very carefully* about how you do it!

8.22 Error-Tracking

Contents

8.22.1 Why Error-Tracking is required

- Getting rid of risks to deploy code with bugs
- Helping QA with testing code
- Getting a quick notification about troubles
- Allowing a fast turnaround with bug fixes
- Receiving a convenient display errors in admin panel
- Sorting the errors by user / browser segments

8.22.2 Implementation

Implementation of Error-tracking,

https://gitlab.ebi.ac.uk/soumyadip/error-tracking-example

8.22.3 Gitlab Integration

Gitlab Integration information can be found at,

https://docs.gitlab.com/ee/user/project/operations/error_tracking.html

8.23 Gitlab to Github pipeline

Contents

8.23.1 Code Pipeline

There are teams/projects that need their public facing code to be on Github and their in-house code to be in Gitlab. The Gitlab to Github automated code pull/push can be set up and found at,

https://docs.gitlab.com/ee/user/project/import/github.html

8.23.2 Release Automation

There are cli tools available to automate release of code from gitlab pipelines to any gitlab or github project by the use of auth token.

Tools used:

- Github - https://github.com/github/hub
- Gitlab - https://pypi.org/project/gitlab-release/

8.23.3 Implementation

The below .gitlab-ci.yml demonstrates release automation of binary files generated on 'dist/' artifact from build(not shown) stage. This implementation has been taken from https://gitlab.ebi.ac.uk/TSI/RDSDS/DSDS-Client/blob/master/.gitlab-ci.yml.

```
release-gitlab:
  stage: release
  image: python
  script:
    - pip3 install gitlab-release
    - cd dist/
    - gitlab-release --server ${GITLAB_SERVER}/${IMAGE_GROUP}/${IMAGE_PROJECT} --
↪project_id ${IMAGE_PROJECT_ID} --description ${CI_REGISTRY}/${IMAGE_GROUP}/${IMAGE_
↪PROJECT}/${IMAGE_NAME}:${CI_BUILD_TAG} dsds-client
  artifacts:
```

(continues on next page)

(continued from previous page)

```
    paths:
      # Must include files passed to gitlab_release
      - dist/
  only:
    - tags

release-github:
  stage: release
  image: debian
  script:
    - apt-get update && apt-get install -y hub gettext-base
    - mkdir -p /root/.config
    - envsubst < hub.template > /root/.config/hub
    - hub clone --bare https://${GITHUB_USER}:${GITHUB_TOKEN}@github.com/${GITHUB_
↪REPO}.git cloned-project/
    - mkdir -p cloned-project/dist
    - cp -r dist cloned-project/
    - cd cloned-project
    - '[ -z "${RELEASE_MESSAGE}" ] && export RELEASE_MESSAGE="${CI_COMMIT_REF_NAME}
↪\n\nRelease-${CI_COMMIT_REF_NAME}"|| echo "Release message:\n${RELEASE_MESSAGE}"'
    - echo -e ${RELEASE_MESSAGE} > release_mssg.txt
    - cat release_mssg.txt
    - 'hub release create -F release_mssg.txt -a "dist/dsds-client#dsds-client-centos
↪" ${CI_COMMIT_REF_NAME}'
  artifacts:
    paths:
      # Must include files passed to gitlab_release
      - dist/
  only:
    - tags
```

8.24 Gitlab DevOps

This document is based on Gitlab instance deployed in EBI by web production team. The reference of the instance can be found here.

Contents

* *Example*

8.24.1 Pre-requisites

- Should have EBI Gitlab account
- Basic knowledge on Docker/Kubernetes

8.24.2 DevOps - Sounds familiar?

DevOps can be best explained as people working together to build, deliver, and run resilient software at the speed of their particular business. DevOps enables software development (**Dev**) and operations (**Ops**) teams to accelerate delivery through automation, collaboration, fast feedback, and iterative improvement.

For more, check https://about.gitlab.com/devops/

8.24.3 Why Gitlab for DevOps?

There are a lot of DevOps tools out there. As a single application for the entire DevOps life cycle, GitLab can remove the pain of having to choose, integrate, learn, and maintain the multitude of tools necessary for a successful DevOps tool chain.

For more, check https://about.gitlab.com/devops-tools/

8.24.4 Auto DevOps

Auto DevOps provides pre-defined CI/CD configuration which allows you to automatically detect, build, test, deploy, and monitor your applications. Leveraging CI/CD best practices and tools, Auto DevOps aims to simplify the setup and execution of a mature & modern software development lifecycle.

Pros

With Auto DevOps, the software development process becomes easier to set up as every project can have a complete workflow from verification to monitoring with minimal configuration. Just push your code and GitLab takes care of everything else. This makes it easier to start new projects and brings consistency to how applications are set up throughout a company.

Cons

As you can expect, it is not possible for Gitlab to detect and setup deployment for each and every possible application. There are certain expectation from application that Gitlab required it to be adhered to for Auto-DevOps to be working without any manual scripting, details on this later. Further, having Gitlab doing all the build and deployment might not be suitable use-case for everyone.

Pre-requisites

- GitLab Runner
- Base domain
- Kubernetes
- Prometheus

For more, https://docs.gitlab.com/ee/topics/autodevops/#requirements

Expectation

Although it is called *Auto* DevOps, there are some assumptions have been made for Auto Deployment to be working for the application. The assumptions(and experiences!) are as follows,

- Application should be listening on 5000 port
- Better to write your own Dockerfile, things generally don't work well with hiroku build packs

You can customize some of the parts, more details on https://docs.gitlab.com/ee/topics/autodevops/#customizing

Example

You can find the implementation of Auto-DevOps in,

https://gitlab.ebi.ac.uk/soumyadip/minimal-ruby-app

and sample pipeline in the same project,

https://gitlab.ebi.ac.uk/soumyadip/minimal-ruby-app/pipelines/22970

Security Dashboard, containing test result after running Auto-DevOps tests,

https://gitlab.ebi.ac.uk/soumyadip/minimal-ruby-app/security/dashboard/?project_id=1190&page=1&days=90

Metrics of the running service can be found at,

https://gitlab.ebi.ac.uk/soumyadip/minimal-ruby-app/environments/115/metrics

8.25 Tracing

Contents

8.25.1 Why Tracing is required

As on-the-ground microservice practitioners are quickly realizing, the majority of operational problems that arise when moving to a distributed architecture are ultimately grounded in two areas: networking and observability. It is simply an orders of magnitude larger problem to network and debug a set of intertwined distributed services versus a single monolithic application.

8.25.2 What Tracing addresses

- Distributed transaction monitoring
- Performance and latency optimization
- Root cause analysis
- Service dependency analysis
- Distributed context propagation

8.25.3 Implementation

Implementation of tracing (Opentracing - Jaegar) can be found in,

https://gitlab.ebi.ac.uk/soumyadip/opentracing-sample-python/

8.25.4 Gitlab Integration

Gitlab Integration information can be found at,

https://docs.gitlab.com/ee/user/project/operations/tracing.html

8.26 Auto DevOps tutorial

8.26.1 Objective

Learn how to setup Auto-DevOps.

8.26.2 Forking Repo

Fork the Auto-DevOps example provided by gitlab from below,

https://gitlab.com/auto-devops-examples/minimal-ruby-app

8.26.3 Adding Kubernetes Cluster

You can create cluster from **Operations > Kubernetes > Add Kubernetes cluster > Create cluster on GKE**

You need to create account on Google Cloud in able to use this feature. Once the cluster is created, you need to enable,

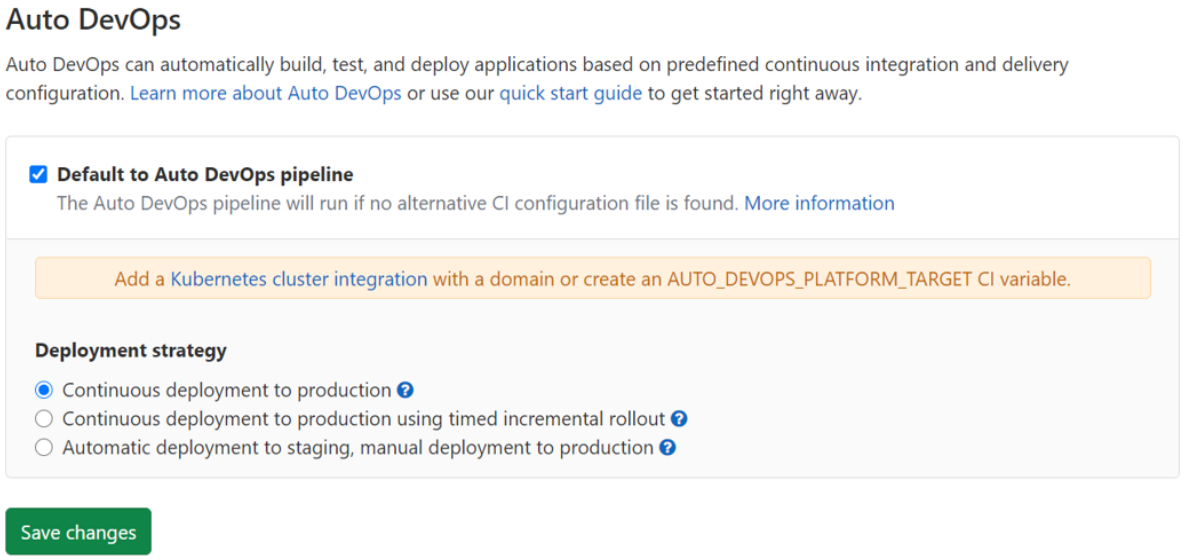
- **Ingress** (For load balancing)
- **cert-manager** (Optional - for SSL certificates)

- **Prometheus** (Optional - for monitoring cluster/application)

Once Ingress setup is done you need to set base domain as $public-ip$.nip.io. There should be suggestion below base domain input box for your reference.

8.26.4 Enabling Auto-DevOps

Enable Auto-DevOps from **Settings > CI/CD > Auto-DevOps**,



8.26.5 Checking Deployment

Once Auto-Devops is enabled, you can go to **CI/CD** to check for latest pipeline. Latest pipeline should show application build and deployment steps. Your application link should be displayed in **Production** stage.

8.26.6 Checking Security Dashoard

Once the pipeline is succeded, you can go to **Security & Complaince > Security Dashboard** to check for security scan reports.

8.26.7 Checking Monitoring Dashoard

If you have enabled **Prometheus** in Cluster Applications, you can check **Operations > Metrics** for application CPU/Memory/IO metrics.

8.26.8 Conclusion

This tutorial guides you through setting up auto-devops with a sample application. But there are some customization needed for your application to be enabled for auto-devops. You can find more about customization in,

https://docs.gitlab.com/ee/topics/autodevops/customize.html

8.26.9 Best Practices

1. Always have Dockerfile ready, don't let auto-devops build using build packs. Generally it does not result well.
2. Instead of enabling auto-devops through settings, you can create .gitlab-ci.yml and import auto-devops template. It does basically the same thing but the later gives you more flexibility in case you need to add more steps.
3. If you are trying out in cloud kubernetes services, be mindful this can incur heavy cost. Please destroy the cluster once you have tried out.
4. Adding your own cluster is the cost-efficient option. If your organization provides kubernetes cluster in private cloud, you can add the cluster following below guide, https://docs.gitlab.com/ee/user/project/clusters/add_remove_clusters.html

8.27 Additional considerations for research pipelines

This is continuation of Advanced Kubernetes Practical. There are additional considerations that we feel strongly about. It is a bit tricky to make practical exercises so we just share our opinions as reading materials for now.

- *Reading 1: Bringing compute close to data*
- *Reading 2: Resource consumption*
- *Reading 3: Docker builds*
- *Reading 4: CI/CD toolchain*

8.27.1 Reading 1: Bringing compute close to data

In the era of cloud computing, compute is not the center of the universe. Data is. Always try to bring compute to data, instead of data to compute for big data problems. The key is to make the compute highly portable.

In this project, we have to investigate the possibility to run Freebayes / Samtools on GKE due to the computing resource allocated to worker nodes. This means that we no longer have "local" access to the data storage within EBI. There are many solutions to access data remotely, which also means that there is no single solution solving most of the problems without creating their own share of issues.

We are using a combination of *wget*, *gzip* and *samtools*. Different tools and strategies may be necessary to reduce the overall cost of network, cpu, memory, storage, and computing and development time. We are trying to find the best solution. Please do shed some light to help us if you have any thoughts.

8.27.2 Reading 2: Resource consumption

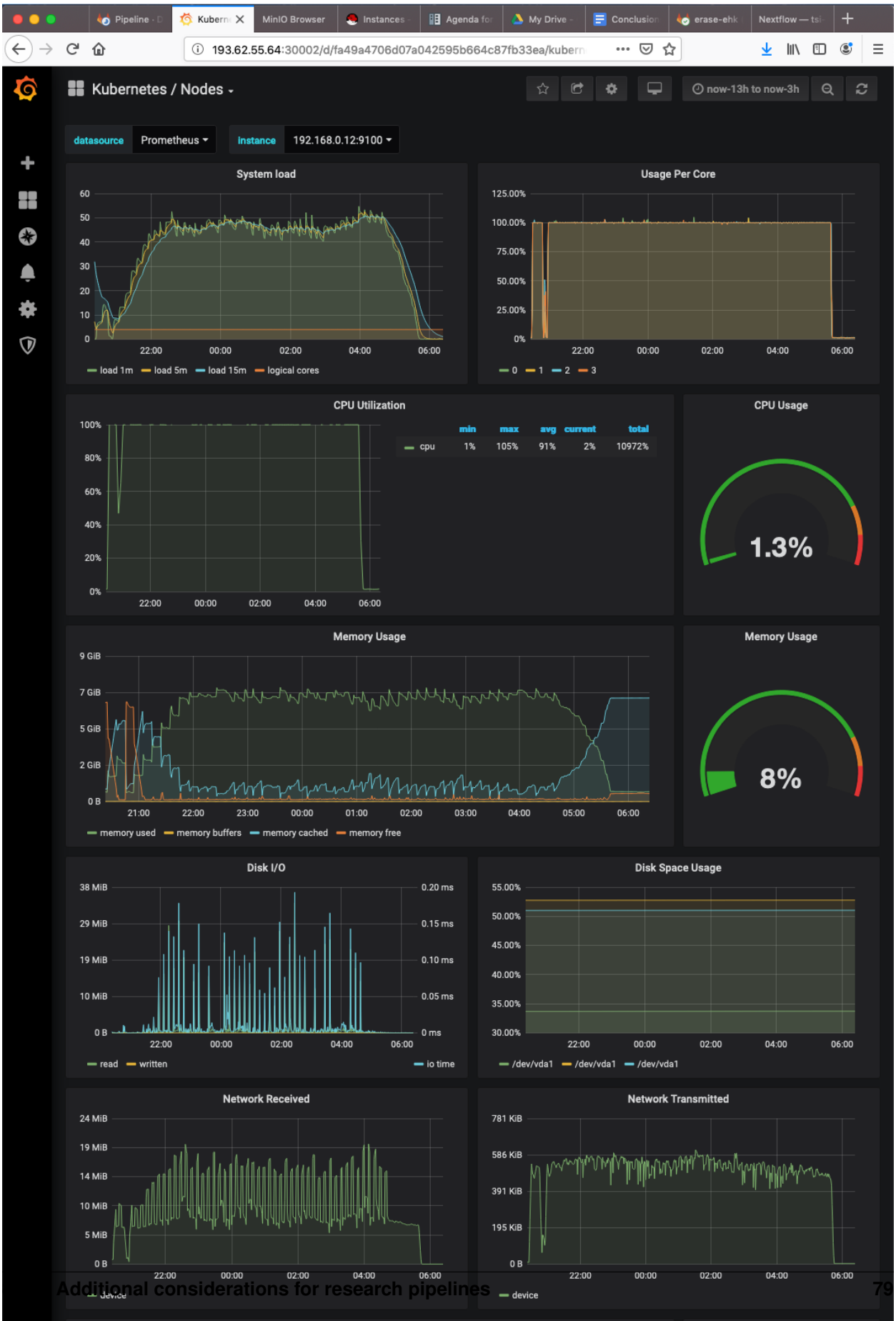
It is of paramount importance to monitor resource consumption to understand where the bottlenecks are. Prometheus & Grafana provide a simple monitoring tool but useful enough to view resource dynamics. The screenshot below shows an attempt to scale up Kubernetes pods to call variants in all 1000 genomes.

- CPU was maxed out almost immediately.
- Memory consumption kept increasing over about a hour and a half.
- Disk IO, network IO, disk space and Inode usage were all OK.
- The system failed after runing for about 8 hours.
- The dashboards for other worker nodes showed the same behaviour.

There were seemingly random SIGABRT errors on the SSH client after a while. There may not be enough CPU to handle scheduled jobs. There may be severe memory leaks by Freebayes, which eventually chocked up the pipeline. We can take the following actions:

1. Reducing the batch size so that there are less jobs waiting for CPU cycles
2. Adding more CPUs to the existing nodes and / or more nodes with larger number of CPUs
3. Using more sophisticated job scheduler instead of Bash script + Kubernetes

We would not know what to do otherwise. A resource monitor is useful for development, test and production.


Kubernetes / Nodes
now-13h to now-3h
datasource
Prometheus
instance
192.168.0.12:9100
System load
load 1m
load 5m
load 15m
logical cores
Usage Per Core
CPU Utilization
min
max
avg
current
total
cpu
CPU Usage
1.3%
Memory Usage
memory used
memory buffers
memory cached
memory free
Memory Usage
8%
Disk I/O
read
written
io time
Disk Space Usage
Network Received
Network Transmitted
device
Inodes Usage
Inodes Usage

8.27.3 Reading 3: Docker builds

There are two tools in the pipeline: Samtools and Freebayes. we have containerised the tools and wrapper / driver scripts. Here are some best practices to follow:

- Creating two separate containers to manage them separately at runtime.
- Starting with minimal base image, for example *debian:buster-slim* instead of *ubuntu:bionic* for example.
- Creating CI/CD toolchain to buiid Docker images to avoid doing repetitive work.
- Downloading binaries from trusted source only.
- Building from the source if only necessary.
- Including wrapper / driver scripts in the container for good encapsulation.
- Minimising the number of layers if possible.
- Ording layers according to the likelihood of changes.

Here an example of Docker files:

- https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/samtools/Dockerfile

The build jobs can be part of a larger toolchain to create Kubernetes cluster, to deploy and to run workload. This gives you better control and nicer integration. Docker hub can build images when integrated with a source control repository directly. But it provdes limited resources with less control. There is private repository for Docker images at EBI. My personal preference is to push the image built on GitLab to Docker Hub. The build process can take the following steps:

```
- git clone ${CI_GIT_URL}
- cd ${CI_BUILD_CONTEXT}
- docker login -u "${CI_REGISTRY_USER}" -p "${CI_REGISTRY_PASSWORD}" ${CI_REGISTRY}
- docker build -f ${CI_DOCKER_FILE} -t ${CI_SOURCE_IMAGE} ${CI_BUILD_CONTEXT} | tee $
↪{ARTIFACT_DIR}/build.log
- docker tag ${CI_SOURCE_IMAGE} ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}
- docker push ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG} | tee ${ARTIFACT_DIR}/push.
↪log
- docker logout
```

You can simply perform the seven steps to build an image on a standard GitLab runner with DIND enabled. Here is a link to the images created with this process https://hub.docker.com/?namespace=davidyuyuan.

8.27.4 Reading 4: CI/CD toolchain

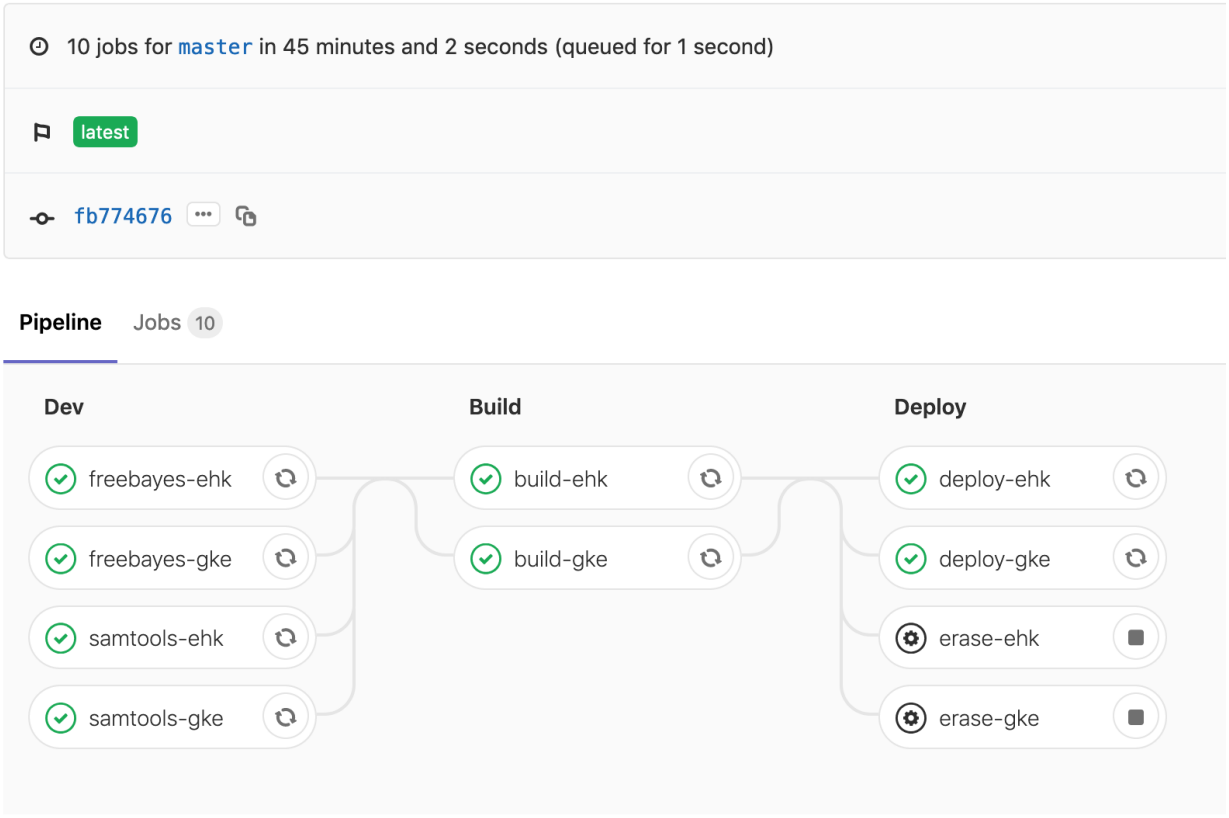
In addition to creating Docker images, many other jobs are needed in order to assemble the pipeline to produce VCFs for 1000g. Here are the major tasks:

1. Building Docker images for target clouds
2. Creating / configuring Kubernetes cluster on OpenStack (RKE) or Google (GKE)
3. Deploying pipeline into the environments and performing smoke-test
4. Clean up Kubernetes clusters when finished

The CI/CD toolchain below automates all the jobs. It improves the DevOps efficiency significantly. Always creating CI/CD toolchain. You will see immediate savings. Here are the links to the CI files:

- https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/.gitlab-ci.yml
- https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/gcp/.gitlab-ci.yml

- https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/.gitlab-ci.yml



8.28 Kubernetes on OpenStack

Kubernetes cluster can run on OpenStack. One of the many options is to use Rancher Kubernetes Engine. In this demo, we will explore capabilities of Kubernetes with an simple example slightly beyond "Hello World". We use Kubernetes to create a StatefulSet of NGINX pods. The system has fail-over protection and disaster recovery built in. The web content is stored in a S3 bucket as web servers in clouds would normally do.

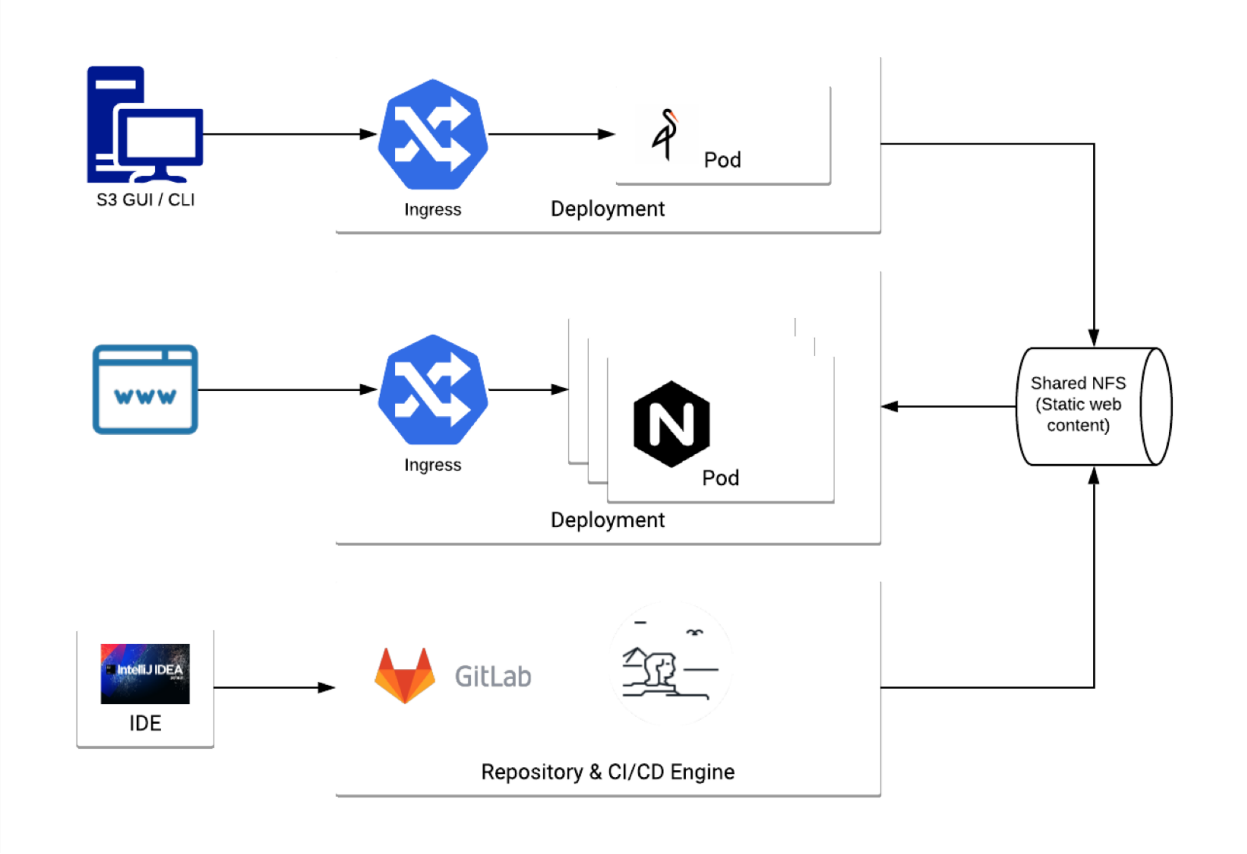


Figure: NGINX, NFS and Minio integration

In particular, we will discuss items 1 - 4 and leave items 5 - 7 as reading after the workshop:

1. *Credentials and Floating IP [FIP]* - Accessing a newly created K8S cluster.
2. *Kubectl* - Interacting with the cluster with CLI
3. *Kubernetes Dashboard* - Interacting with the cluster with GUI
4. *Prometheus and Grafana* - Monitoring the cluster with GUI
5. *Application for web (NGINX)* - Creating NGINX service via StatefulSet, including: Service, StatefulSet, Ingress & private or shared NFS volumes

a. *Persistent volume claim shared between Minio and NGINX*

b. *Stateful set and service for NGINX*

c. *Ingress to expose service externally*

6. *Data Loading via S3 (Minio)* - Loading data with Minio GUI into a new NFS volume

a. *Minio deployment and service*

b. *Deleting Minio after using it for better security*

7. *End-to-end: integrating NGINX, NFS and Minio* - How everything works together to form a basic system below:

8.28.1 Credentials and Floating IP [FIP]

If a Kubernetes cluster is created on Rancher Kubernetes Engine, it will generate the configuration of the new cluster as a *kube.yml* file. It is a good idea to rename and to move it to *~/.kube* following general convention. You may find that you have multiple configure files already with the default named *config*. I have moved my *./kube.yml* to *~/.kube/oskconfig*:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ cd ~/.kube
C02XD1G9JGH7:.kube davidyuan$ ls -l
total 160
drwxr-xr-x    3 davidyuan  384566875     96 Dec 20 10:42 cache
-rw-------    1 davidyuan  384566875  68200 Apr 29 10:46 config
-rw-r--r--    1 davidyuan  384566875   1872 Feb 23 17:12 eksconfig
drwxr-xr-x  667 davidyuan  384566875  21344 May 10 17:04 http-cache
-rw-r--r--    1 davidyuan  384566875   5387 May 10 15:38 oskconfig
C02XD1G9JGH7:.kube davidyuan$
```

Manipulate KUBECONFIG so that you do not have to use *–kubeconfig kube.yml* in every *kubectl* command. For example, update *~/.bash_profile* with the following:

```
export KUBECONFIG=~/.kube/oskconfig:~/.kube/config
```

After restarting bash session or reloading ~/.bash_profile, you should see the contexts are merged with the new configuration *local local kube-admin-local* as the current context:

```
C02XD1G9JGH7:~ davidyuan$ kubectl config get-contexts
CURRENT   NAME                                                CLUSTER
↪                                AUTHINFO                                                ␣
↪  NAMESPACE
          aks-ebi1                                            aks-ebi1
↪                                clusterUser_uksouth_aks-ebi1                            ␣
          arn:aws:eks:eu-north-1:871979166454:cluster/k8s-eks   arn:aws:eks:eu-north-
↪1:871979166454:cluster/k8s-eks   arn:aws:eks:eu-north-1:871979166454:cluster/k8s-eks
          arn:aws:eks:eu-north-1:871979166454:cluster/tsi       arn:aws:eks:eu-north-
↪1:871979166454:cluster/tsi       arn:aws:eks:eu-north-1:871979166454:cluster/tsi
          cyclecloud                                          cyclecloud
↪                                clusterUser_uksouth_cyclecloud                          ␣
          euwest1c.4ebi.uk                                    euwest1c.4ebi.uk
↪                                euwest1c.4ebi.uk                                        ␣
*         local                                               local
↪                                kube-admin-local                                        ␣
          ucp_hh-ucp.caas.ebi.ac.uk:6443_davidyuan            ucp_hh-ucp.caas.ebi.
↪ac.uk:6443_davidyuan               ucp_hh-ucp.caas.ebi.ac.uk:6443_davidyuan
          ucp_hx-ucp.caas.ebi.ac.uk:6443_davidyuan            ucp_hx-ucp.caas.ebi.
↪ac.uk:6443_davidyuan               ucp_hx-ucp.caas.ebi.ac.uk:6443_davidyuan
```

Note that you can use *kubectl config use-context local* to reset the current context. If you are working with multiple Kubernetes clusters in the same or different clouds, you always want to check and switch contexts with these two commands.

A floating IP is assigned to the new cluster for various endpoints. The value can be found in the config file or with the following command:

```
C02XD1G9JGH7:.kube davidyuan$ kubectl cluster-info
Kubernetes master is running at https://45.86.170.94:6443
CoreDNS is running at https://45.86.170.94:6443/api/v1/namespaces/kube-system/
↪services/kube-dns:dns/proxy
```

(continues on next page)

(continued from previous page)

```
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

8.28.2 Kubectl

Explore your new cluster to get familiar with it. With the following commands, you will have a fairly good idea what your cluster looks like. Here is my cluster when it was first created:

```
C02XD1G9JGH7:~ davidyuan$ kubectl get nodes
NAME           STATUS    ROLES               AGE       VERSION
192.168.0.12   Ready     worker              22h       v1.13.5
192.168.0.13   Ready     worker              22h       v1.13.5
192.168.0.14   Ready     worker              22h       v1.13.5
192.168.0.15   Ready     controlplane,etcd   22h       v1.13.5
192.168.0.16   Ready     controlplane,etcd   22h       v1.13.5
192.168.0.17   Ready     controlplane,etcd   22h       v1.13.5

C02XD1G9JGH7:~ davidyuan$ kubectl get svc --all-namespaces
NAMESPACE     NAME                                          TYPE        CLUSTER-IP
↪    EXTERNAL-IP   PORT(S)                      AGE
default       kubernetes                                    ClusterIP   10.43.0.1
↪    <none>        443/TCP                      22h
ingress       ingress-nginx-ingress-controller              NodePort    10.43.81.91
↪    <none>        80:30000/TCP,443:30001/TCP   22h
ingress       ingress-nginx-ingress-controller-metrics      ClusterIP   10.43.73.90
↪    <none>        9913/TCP                     22h
ingress       ingress-nginx-ingress-controller-stats        ClusterIP   10.43.142.79
↪    <none>        18080/TCP                    22h
ingress       ingress-nginx-ingress-default-backend         ClusterIP   10.43.106.89
↪    <none>        80/TCP                       22h
kube-system   kube-dns                                      ClusterIP   10.43.0.10
↪    <none>        53/UDP,53/TCP                22h
kube-system   metrics-server                                ClusterIP   10.43.200.
↪136   <none>        443/TCP                      22h
kube-system   prometheus-operator-coredns                   ClusterIP   None
↪    <none>        9153/TCP                     22h
kube-system   prometheus-operator-kube-controller-manager   ClusterIP   None
↪    <none>        10252/TCP                    22h
kube-system   prometheus-operator-kube-etcd                 ClusterIP   None
↪    <none>        2379/TCP                     22h
kube-system   prometheus-operator-kube-scheduler            ClusterIP   None
↪    <none>        10251/TCP                    22h
kube-system   prometheus-operator-kubelet                   ClusterIP   None
↪    <none>        10250/TCP                    22h
kube-system   tiller-deploy                                 ClusterIP   10.43.73.6
↪    <none>        44134/TCP                    22h
monitoring    alertmanager-operated                         ClusterIP   None
↪    <none>        9093/TCP,6783/TCP            22h
monitoring    prometheus-operated                           ClusterIP   None
↪    <none>        9090/TCP                     22h
monitoring    prometheus-operator-alertmanager              ClusterIP   10.43.172.51
↪    <none>        9093/TCP                     22h
monitoring    prometheus-operator-grafana                   ClusterIP   10.43.15.32
↪    <none>        80/TCP                       22h
monitoring    prometheus-operator-grafana-custom            NodePort    10.43.205.
↪187   <none>        3000:30002/TCP               22h
```

(continues on next page)

(continued from previous page)

```
monitoring    prometheus-operator-kube-state-metrics         ClusterIP   10.43.167.
↪241   <none>        8080/TCP                      22h
monitoring    prometheus-operator-operator                   ClusterIP   10.43.201.
↪240   <none>        8080/TCP                      22h
monitoring    prometheus-operator-prometheus                 ClusterIP   10.43.118.
↪118   <none>        9090/TCP                      22h
monitoring    prometheus-operator-prometheus-custom          NodePort    10.43.21.250␣
↪    <none>        9090:30986/TCP               22h
monitoring    prometheus-operator-prometheus-node-exporter   ClusterIP   10.43.179.
↪232   <none>        9100/TCP                      22h

C02XD1G9JGH7:~ davidyuan$ kubectl get pod --all-namespaces
NAMESPACE     NAME                                                    READY     ␣
↪STATUS      RESTARTS   AGE
default       listening-skunk-nfs-client-provisioner-79fb65dd79-86qgq   1/1       ␣
↪Running     0          22h
ingress       ingress-nginx-ingress-controller-f5cc4968f-nkb9d        1/1       ␣
↪Running     0          22h
ingress       ingress-nginx-ingress-default-backend-7965478b7-6jnwk   1/1       ␣
↪Running     0          22h
kube-system   canal-5frl6                                             2/2       ␣
↪Running     0          22h
kube-system   canal-7k8pp                                             2/2       ␣
↪Running     0          22h
kube-system   canal-dk44m                                             2/2       ␣
↪Running     0          22h
kube-system   canal-lk4sz                                             2/2       ␣
↪Running     0          22h
kube-system   canal-msmtf                                             2/2       ␣
↪Running     0          22h
kube-system   canal-xw6v4                                             2/2       ␣
↪Running     0          22h
kube-system   kube-dns-58bd5b8dd7-djlth                               3/3       ␣
↪Running     0          22h
kube-system   kube-dns-58bd5b8dd7-kx6ls                               3/3       ␣
↪Running     0          22h
kube-system   kube-dns-autoscaler-77bc5fd84-5qhb2                     1/1       ␣
↪Running     0          22h
kube-system   metrics-server-58bd5dd8d7-qjkn4                         1/1       ␣
↪Running     0          22h
kube-system   rke-kube-dns-addon-deploy-job-bxdht                     0/1       ␣
↪Completed   0          22h
kube-system   rke-metrics-addon-deploy-job-62kdt                      0/1       ␣
↪Completed   0          22h
kube-system   rke-network-plugin-deploy-job-5ql57                     0/1       ␣
↪Completed   0          22h
kube-system   tiller-deploy-5f4fc5bcc6-tpcps                          1/1       ␣
↪Running     0          22h
monitoring    alertmanager-prometheus-operator-alertmanager-0         2/2       ␣
↪Running     0          22h
monitoring    prometheus-operator-grafana-d6c4c5bb-vbg72              2/2       ␣
↪Running     0          22h
monitoring    prometheus-operator-kube-state-metrics-79f476bff6-hlk9p   1/1       ␣
↪Running     0          22h
monitoring    prometheus-operator-operator-55c67c5d64-hcx92           1/1       ␣
↪Running     0          22h
monitoring    prometheus-operator-prometheus-node-exporter-99mp6      1/1       ␣
↪Running     0          22h
```

(continues on next page)

(continued from previous page)

```
monitoring    prometheus-operator-prometheus-node-exporter-h6tkt         1/1
↪Running     0          22h
monitoring    prometheus-operator-prometheus-node-exporter-l4ggc         1/1
↪Running     0          22h
monitoring    prometheus-prometheus-operator-prometheus-0                3/3
↪Running     1          22h

C02XD1G9JGH7:~ davidyuan$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
↪STATUS    CLAIM
↪                                    STORAGECLASS   REASON    AGE
pvc-00183ff4-7318-11e9-9f21-fa163ebf1f8e   50Gi       RWO            Delete
↪Bound     monitoring/prometheus-prometheus-operator-prometheus-db-prometheus-
↪prometheus-operator-prometheus-0           nfs-client               22h
pvc-fc23cc6b-7317-11e9-9f21-fa163ebf1f8e   20Gi       RWO            Delete
↪Bound     monitoring/alertmanager-prometheus-operator-alertmanager-db-alertmanager-
↪prometheus-operator-alertmanager-0   nfs-client               22h

C02XD1G9JGH7:~ davidyuan$ kubectl get storageclass
NAME          PROVISIONER                                            AGE
nfs-client    cluster.local/listening-skunk-nfs-client-provisioner   22h

C02XD1G9JGH7:~ davidyuan$ kubectl get statefulset --all-namespaces
NAMESPACE    NAME                                          READY     AGE
monitoring   alertmanager-prometheus-operator-alertmanager   1/1       22h
monitoring   prometheus-prometheus-operator-prometheus      1/1       22h

C02XD1G9JGH7:~ davidyuan$ kubectl get replicaset --all-namespaces
NAMESPACE     NAME                                                DESIRED   CURRENT
↪READY     AGE
default       listening-skunk-nfs-client-provisioner-79fb65dd79   1         1
↪1         22h
ingress       ingress-nginx-ingress-controller-f5cc4968f          1         1
↪1         22h
ingress       ingress-nginx-ingress-default-backend-7965478b7     1         1
↪1         22h
kube-system   kube-dns-58bd5b8dd7                                 2         2
↪2         22h
kube-system   kube-dns-autoscaler-77bc5fd84                       1         1
↪1         22h
kube-system   metrics-server-58bd5dd8d7                           1         1
↪1         22h
kube-system   tiller-deploy-5f4fc5bcc6                            1         1
↪1         22h
monitoring    prometheus-operator-grafana-d6c4c5bb                1         1
↪1         22h
monitoring    prometheus-operator-kube-state-metrics-79f476bff6   1         1
↪1         22h
monitoring    prometheus-operator-operator-55c67c5d64             1         1
↪1         22h
```

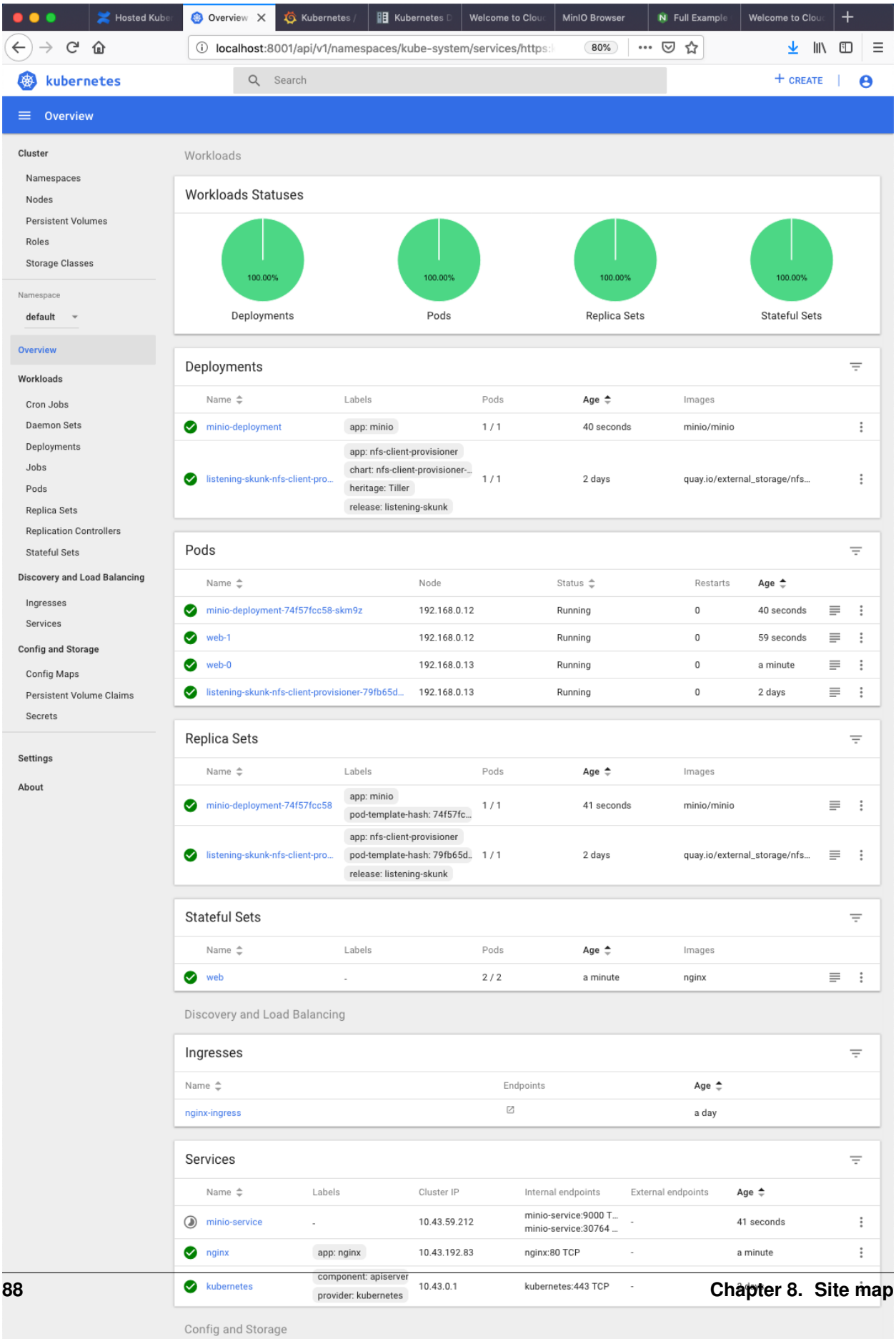
8.28.3 Kubernetes Dashboard

Kubernetes dashboard is very handy, especially for new users. The commands in the section of *Kubectl* can be replaced by single-button clicks in the dashboard. Following the instructions to create dashboard, admin-user, access token for

the dashboard, and finally enable proxy in background:

```
C02XD1G9JGH7:.kube davidyuan$ kubectl apply -f https://github.com/kubernetes/
↪dashboard/raw/master/aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created

C02XD1G9JGH7:.kube davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/
↪raw/external/tsi-cc/ResOps/scripts/k8s-dashboard/admin-user.yml
serviceaccount/admin-user created
clusterrolebinding.rbac.authorization.k8s.io/admin-user created

C02XD1G9JGH7:.kube davidyuan$ kubectl -n kubernetes-dashboard describe secret
↪$(kubectl -n kubernetes-dashboard get secret | grep admin-user | awk '{print $1}')
Name:         admin-user-token-fb742
Namespace:    kubernetes-dashboard
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: admin-user
              kubernetes.io/service-account.uid: 46d6fe53-53c9-11ea-8681-fa163ee0afdc

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1017 bytes
namespace:  20 bytes
token:      __DELETED__

C02XD1G9JGH7:.kube davidyuan$ kubectl proxy &
[1] 90298
C02XD1G9JGH7:.kube davidyuan$ Starting to serve on 127.0.0.1:8001
```

Now you can access the dashboard as if it is running on the local client http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/. It is more efficient to check the status of a lot of resource together on the dashboard sometimes.


Hosted Kube
Overview
Kubernetes /
Kubernetes D
Welcome to Clou
MinIO Browser
Full Example
Welcome to Clou
localhost:8001/api/v1/namespaces/kube-system/services/https:
80%
kubernetes
Search
CREATE
Overview
Cluster
Namespaces
Nodes
Persistent Volumes
Roles
Storage Classes
Namespace
default
Overview
Workloads
Cron Jobs
Daemon Sets
Deployments
Jobs
Pods
Replica Sets
Replication Controllers
Stateful Sets
Discovery and Load Balancing
Ingresses
Services
Config and Storage
Config Maps
Persistent Volume Claims
Secrets
Settings
About
Workloads
Workloads Statuses
100.00%
Deployments
100.00%
Pods
100.00%
Replica Sets
100.00%
Stateful Sets
Deployments
Name
Labels
Pods
Age
Images
minio-deployment
app: minio
1 / 1
40 seconds
minio/minio
listening-skunk-nfs-client-pro...
app: nfs-client-provisioner
chart: nfs-client-provisioner-...
heritage: Tiller
release: listening-skunk
1 / 1
2 days
quay.io/external_storage/nfs...
Pods
Name
Node
Status
Restarts
Age
minio-deployment-74f57fcc58-skm9z
192.168.0.12
Running
0
40 seconds
web-1
192.168.0.12
Running
0
59 seconds
web-0
192.168.0.13
Running
0
a minute
listening-skunk-nfs-client-provisioner-79fb65d...
192.168.0.13
Running
0
2 days
Replica Sets
Name
Labels
Pods
Age
Images
minio-deployment-74f57fcc58
app: minio
pod-template-hash: 74f57fc...
1 / 1
41 seconds
minio/minio
listening-skunk-nfs-client-pro...
app: nfs-client-provisioner
pod-template-hash: 79fb65d...
release: listening-skunk
1 / 1
2 days
quay.io/external_storage/nfs...
Stateful Sets
Name
Labels
Pods
Age
Images
web
-
2 / 2
a minute
nginx
Discovery and Load Balancing
Ingresses
Name
Endpoints
Age
nginx-ingress
a day
Services
Name
Labels
Cluster IP
Internal endpoints
External endpoints
Age
minio-service
-
10.43.59.212
minio-service:9000 T...
minio-service:30764 ...
41 seconds
nginx
app: nginx
10.43.192.83
nginx:80 TCP
a minute
kubernetes
component: apiserver
provider: kubernetes
10.43.0.1
kubernetes:443 TCP
Config and Storage
Persistent Volume Claims

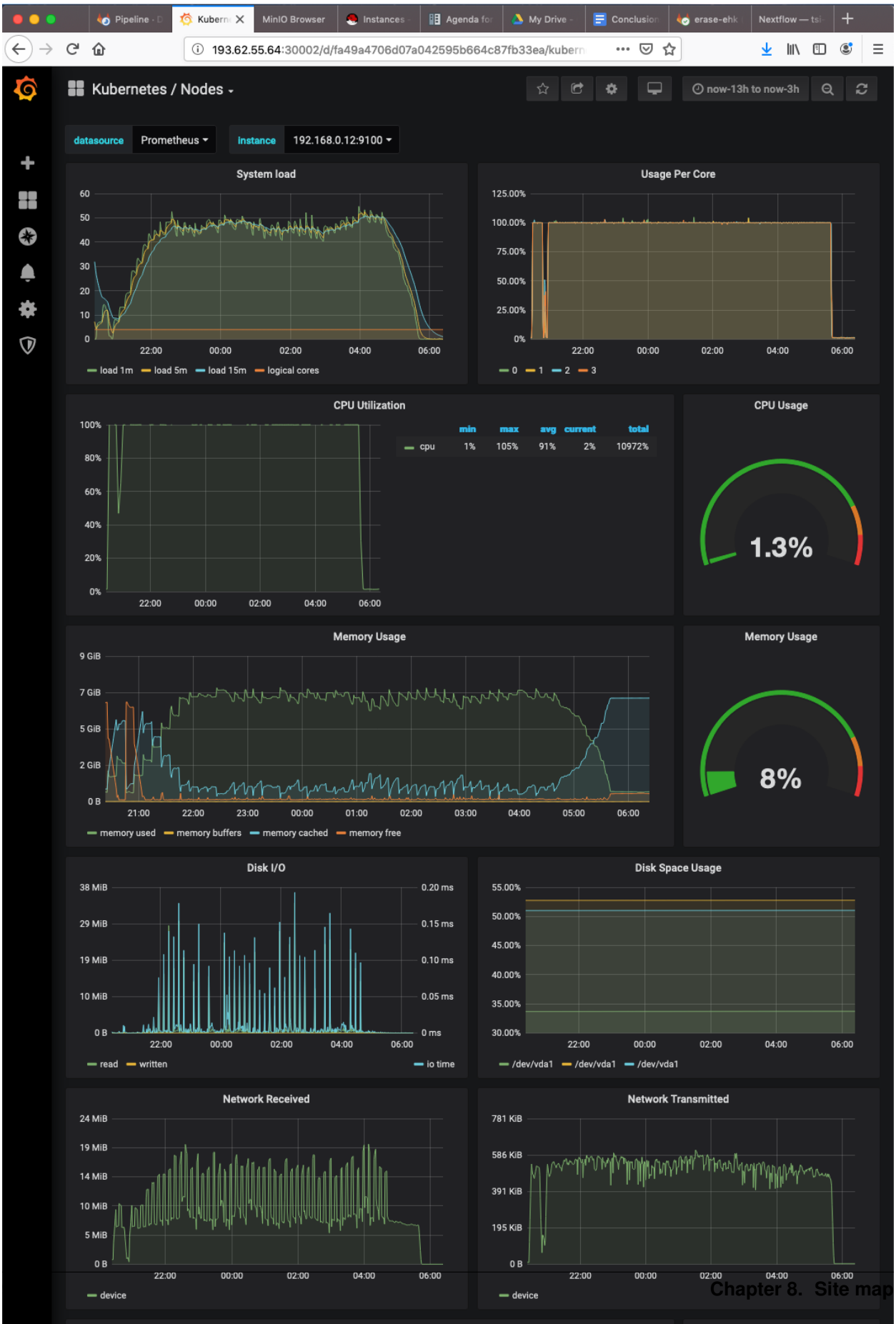
8.28.4 Prometheus and Grafana

A resource monitor is created specific to your cluster for you, where Prometheus provides data source and Grafana presents information in GUI. Run the following command to get the correct NodePort:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get svc --namespace monitoring
NAME                                          TYPE        CLUSTER-IP      EXTERNAL-
↪IP   PORT(S)              AGE
alertmanager-operated                         ClusterIP   None            <none>
↪    9093/TCP,6783/TCP   86d
prometheus-operated                           ClusterIP   None            <none>
↪    9090/TCP            86d
prometheus-operator-alertmanager              ClusterIP   10.43.172.51    <none>
↪    9093/TCP            86d
prometheus-operator-grafana                   ClusterIP   10.43.15.32     <none>
↪    80/TCP              86d
prometheus-operator-grafana-custom            NodePort    10.43.205.187   <none>
↪    3000:30002/TCP      86d
prometheus-operator-kube-state-metrics        ClusterIP   10.43.167.241   <none>
↪    8080/TCP            86d
prometheus-operator-operator                  ClusterIP   10.43.201.240   <none>
↪    8080/TCP            86d
prometheus-operator-prometheus                ClusterIP   10.43.118.118   <none>
↪    9090/TCP            86d
prometheus-operator-prometheus-custom         NodePort    10.43.21.250    <none>
↪    9090:30986/TCP      86d
prometheus-operator-prometheus-node-exporter  ClusterIP   10.43.179.232   <none>
↪    9100/TCP            86d
```

Combining with FIP from kube.yml, you can access the monitor at http://45.86.170.94:30002/login.

There are many useful dashboards built for you already. The most frequently used one is *Kubernetes / Nodes*. It provides very good overview on resource consumption, for example:


Kubernetes / Nodes
now-13h to now-3h
datasource
Prometheus
instance
192.168.0.12:9100
System load
load 1m
load 5m
load 15m
logical cores
Usage Per Core
CPU Utilization
min
max
avg
current
total
cpu
CPU Usage
1.3%
Memory Usage
memory used
memory buffers
memory cached
memory free
Memory Usage
8%
Disk I/O
read
written
io time
Disk Space Usage
Network Received
device
Network Transmitted
device
Inodes Usage
Inodes Usage

8.28.5 Application for web (NGINX)

NGINX stateful set demonstrates typical architecture of web frontend in Kubernetes. It consists of three components:

1. A stateful set of web server pods
2. A clusterIP service to distribute loads to the pods
3. An ingress to enable access from external network

This tutorial is modified from a Kubernetes tutorial https://kubernetes.io/docs/tutorials/stateful-application/basic-stateful-set/.

Persistent volume claim shared between Minio and NGINX

Create a persistent volume claim and check if the volume is created and bounded:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/nginx/pvc.yml
persistentvolumeclaim/minio-pv-claim created

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pvc
NAME             STATUS    VOLUME                                     CAPACITY
↪ACCESS MODES   STORAGECLASS   AGE
minio-pv-claim   Bound     pvc-4e377e97-74a1-11e9-b4f1-fa163eaf0769   25Gi       RWX
↪           nfs-client     10s
www-web-0        Bound     pvc-2ba8d5a1-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO
↪           nfs-client     13h
www-web-1        Bound     pvc-318d02b3-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO
↪           nfs-client     13h

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
↪STATUS    CLAIM
↪                                    STORAGECLASS   REASON    AGE
pvc-00183ff4-7318-11e9-9f21-fa163ebf1f8e   50Gi       RWO            Delete
↪Bound     monitoring/prometheus-prometheus-operator-prometheus-db-prometheus-
↪prometheus-operator-prometheus-0           nfs-client               46h
pvc-2ba8d5a1-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO            Delete
↪Bound     default/www-web-0
↪                                    nfs-client               13h
pvc-318d02b3-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO            Delete
↪Bound     default/www-web-1
↪                                    nfs-client               13h
pvc-4e377e97-74a1-11e9-b4f1-fa163eaf0769   25Gi       RWX            Delete
↪Bound     default/minio-pv-claim
↪                                    nfs-client               10s
pvc-fc23cc6b-7317-11e9-9f21-fa163ebf1f8e   20Gi       RWO            Delete
↪Bound     monitoring/alertmanager-prometheus-operator-alertmanager-db-alertmanager-
↪prometheus-operator-alertmanager-0   nfs-client               46h
```

Stateful set and service for NGINX

A stateful set always requires a service. They are typically included in the same manifest file. Create and explore the stateful set and its service in CLI or dashboard GUI:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/nginx/web.yml
service/nginx created
statefulset.apps/web created

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get service nginx
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
nginx     ClusterIP   10.43.32.1    <none>         80/TCP     100s

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get statefulset web
NAME      READY     AGE
web       2/2       2m35s

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pvc -l app=nginx
NAME         STATUS    VOLUME                                     CAPACITY    ACCESS␣
↪MODES    STORAGECLASS   AGE
www-web-0    Bound     pvc-2ba8d5a1-742f-11e9-9f21-fa163ebf1f8e   1Gi         RWO       ␣
↪      nfs-client     9m2s
www-web-1    Bound     pvc-318d02b3-742f-11e9-9f21-fa163ebf1f8e   1Gi         RWO       ␣
↪      nfs-client     8m52s

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pods -w -l app=nginx
NAME      READY     STATUS    RESTARTS   AGE
web-0     1/1       Running   0          3m
web-1     1/1       Running   0          2m
```

Ingress to expose service externally

Create an ingress so that the cluster IP gets exposed to the external network:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/nginx/ingress.yml
ingress.extensions/nginx-ingress created

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get ingress
NAME            HOSTS     ADDRESS   PORTS     AGE
nginx-ingress   *                   80        26s
```

Now the NGINX is accessible via the the same floating IP for other endpoints, which is provided in kube.yml. In my cluster, the URL is http://45.86.170.94/nginx/.

It is understandable that HTTP404 is returned. We did not define a homepage. There is nothing for the web servers to serve.

8.28.6 Data Loading via S3 (Minio)

Your pipeline should always consider using data in its currently location via a native protocol: S3, FTP, SFTP etc.. Check the official documentation by Kubernetes for storage classes for a specific file system. If you have to move it into new block storage with NFS facade, Minio can be considered. This utility adds S3 support to a POSIX file system, including NFS mount. For simplicity, a standalone Minio with a single volume is used. Check Minio documentation for distributed servers for production. There is another demo to elaborate on that.

Minio deployment and service

Create Minio deployment and service. Check if everything is started successfully:

```
C02XD1G9JGH7:minio davidyuan$ accesskey=__DELETED__
C02XD1G9JGH7:minio davidyuan$ secretkey=__DELETED__
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl create secret generic minio --from-
↪literal=accesskey=${accesskey} --from-literal=secretkey=${secretkey}
secret/minio created

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get secret
NAME                                                  TYPE                                  ⌐
↪     DATA   AGE
default-token-x66s8                                   kubernetes.io/service-account-
↪token   3      87d
listening-skunk-nfs-client-provisioner-token-nsggj    kubernetes.io/service-account-
↪token   3      87d
minio                                                 Opaque                                ⌐
↪     2      0s

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/minio/minio.yml
deployment.apps/minio-nginx created
service/minio-nginx created

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get deployment
NAME                                    READY     UP-TO-DATE   AVAILABLE   AGE
listening-skunk-nfs-client-provisioner   1/1       1            1           47h
minio-nginx                             1/1       1            1           63s

C02XD1G9JGH7:~ davidyuan$ kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes    ClusterIP   10.43.0.1       <none>        443/TCP          81d
minio-nginx   NodePort    10.43.151.136   <none>        9000:30968/TCP   52s
nginx         ClusterIP   10.43.173.206   <none>        80/TCP           30m
```

Note the NodePort. It is needed to access the web UI via the floating IP, for example http://45.86.170.94:30968/. Login with the access key and secret key specified in minio.yml. Upload files via GUI. Follow Minio documentation to use REST interface to load large number of files.

Deleting Minio after using it for better security

Normally, delete Minio after using it for better security. The persistent volume should be kept to be used by other applications such as NGINX. The Minio can be recreated with the same script and mounted to the same volume. The NodePort may be different:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl delete -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/minio/minio.yml
deployment.extensions "minio-nginx" deleted
service "minio-nginx" deleted

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get deployment
NAME                                      READY     UP-TO-DATE   AVAILABLE   AGE
listening-skunk-nfs-client-provisioner    1/1       1            1           47h

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.43.0.1     <none>        443/TCP   47h
nginx        ClusterIP   10.43.32.1    <none>        80/TCP    13h

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pvc
NAME             STATUS    VOLUME                                     CAPACITY
↪ACCESS MODES   STORAGECLASS   AGE
minio-pv-claim   Bound     pvc-4e377e97-74a1-11e9-b4f1-fa163eaf0769   25Gi       RWX
↪           nfs-client     20m
www-web-0        Bound     pvc-2ba8d5a1-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO
↪           nfs-client     13h
www-web-1        Bound     pvc-318d02b3-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO
↪           nfs-client     13h

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
↪STATUS    CLAIM
↪                                  STORAGECLASS   REASON    AGE
pvc-00183ff4-7318-11e9-9f21-fa163ebf1f8e   50Gi       RWO            Delete
↪Bound     monitoring/prometheus-prometheus-operator-prometheus-db-prometheus-
↪prometheus-operator-prometheus-0           nfs-client               47h
pvc-2ba8d5a1-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO            Delete
↪Bound     default/www-web-0
↪                                  nfs-client               13h
pvc-318d02b3-742f-11e9-9f21-fa163ebf1f8e   1Gi        RWO            Delete
↪Bound     default/www-web-1
↪                                  nfs-client               13h
pvc-4e377e97-74a1-11e9-b4f1-fa163eaf0769   25Gi       RWX            Delete
↪Bound     default/minio-pv-claim
↪                                  nfs-client               20m
pvc-fc23cc6b-7317-11e9-9f21-fa163ebf1f8e   20Gi       RWO            Delete
↪Bound     monitoring/alertmanager-prometheus-operator-alertmanager-db-alertmanager-
↪prometheus-operator-alertmanager-0   nfs-client               47h
```

In production, you may want to consider serving static content via REST interface instead of NFS file mount. NGINX would be used as reverse proxy in such web-native archicture. In this case, it is extremely important to choose S3 storage provider carefully for production.

8.28.7 End-to-end: integrating NGINX, NFS and Minio

It is very easy to clean up with Kubernete. Resources created are deleted if the same manifest is provided to the delete command. Note that it is important to follow the correct order to delete resources as one may have dependencies on another.:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl delete -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/minio/minio.yml
deployment.extensions "minio-nginx" deleted
service "minio-nginx" deleted

C02XD1G9JGH7:~ davidyuan$ kubectl delete -f https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/
↪raw/master/tsi-cc/ResOps/scripts/nginx/ingress.yml
ingress.extensions "nginx-ingress" deleted

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl delete -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/nginx/web.yml
service "nginx" deleted
statefulset.apps "web" deleted

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl delete -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/nginx/pvc.yml
persistentvolumeclaim "minio-pv-claim" deleted
```

However, there is an exception by design. Volumes created by stateful set are not deleted when the stateful set itself is deleted. It is designed to be mounted again if the stateful set is created again. Delete them explicitly if needed:

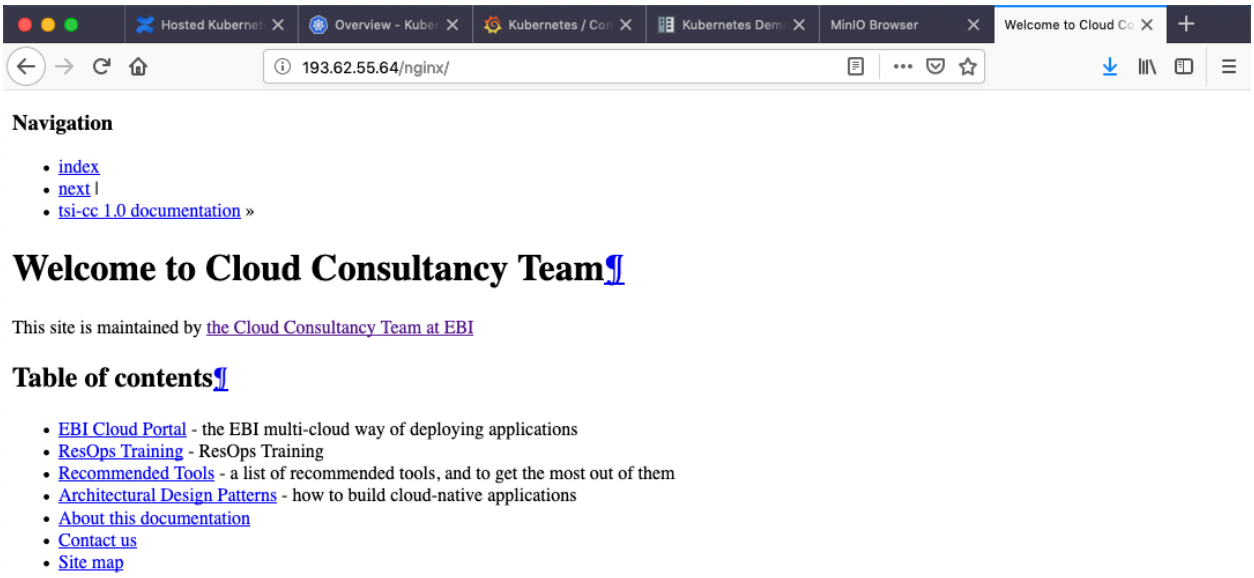
```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pvc
NAME              STATUS    VOLUME                                     CAPACITY  ␣
↪ACCESS MODES   STORAGECLASS   AGE
www-web-0         Bound     pvc-2ba8d5a1-742f-11e9-9f21-fa163ebf1f8e   1Gi       RWO␣
↪           nfs-client     15h
www-web-1         Bound     pvc-318d02b3-742f-11e9-9f21-fa163ebf1f8e   1Gi       RWO␣
↪           nfs-client     15h
```

Update pvc.yml, minio.yml and web.yml to make sure that the mount points are matched, which must be the parent of the default document root */usr/share/nginx/*. The volume is created in ReadWriteMany mode. The storage root for Minio server is changed to the mount point. Note that it is important to follow the correct order to create resources as one may have dependencies on another.:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/nginx/pvc.yml
persistentvolumeclaim/minio-pvc-nginx created

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/nginx/web.yml
service/nginx created
statefulset.apps/web created

C02XD1G9JGH7:~ davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/raw/
↪master/tsi-cc/ResOps/scripts/nginx/ingress.yml
ingress.extensions/nginx-ingress created

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/minio/minio.yml
deployment.extensions/minio-nginx created
service/minio-nginx created
```

Log onto Minio at http://45.86.170.94:30968/, where 30968 is the new NodePort show on GUI. Create a bucket *html* and place an index.html file in it.

Check NGINX http://45.86.170.94/nginx/. You should see an HTML page without styling instead of HTTP404 error.



If you are curious how the backends work, connect to either one of the three pods: *minio-deployment-74f57fcc58-skm9z*, *web-0* or *web-1*. You will see NFS mount of */usr/share/nginx*. In addition, a subdirectory of *html* is created by Minio. The *index.html* uploaded to the *html* bucket is stored in the subdirectory with the same name:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl get pod
NAME                                                     READY     STATUS    ␣
↪RESTARTS   AGE
listening-skunk-nfs-client-provisioner-79fb65dd79-86qgq   1/1       Running   0         ␣
↪    2d22h
minio-nginx-74f57fcc58-skm9z                             1/1       Running   0         ␣
↪    91m
web-0                                                    1/1       Running   0         ␣
↪    91m
web-1                                                    1/1       Running   0         ␣
↪    91m

C02XD1G9JGH7:tsi-ccdoc davidyuan$ kubectl exec -it web-0 -- /bin/bash

root@web-0:/# df -h
Filesystem                                                                                ␣
↪Size  Used Avail Use% Mounted on
overlay                                                                                   ␣
↪49G  3.3G   46G   7% /
tmpfs                                                                                     ␣
↪64M     0   64M   0% /dev
tmpfs                                                                                     3.
↪9G     0  3.9G   0% /sys/fs/cgroup
/dev/vda1                                                                                 ␣
↪49G  3.3G   46G   7% /etc/hosts
shm                                                                                       ␣
↪64M     0   64M   0% /dev/shm
192.168.0.19:/nfs/default-minio-pvc-nginx-pvc-40da7238-755a-11e9-9f21-fa163ebf1f8e        ␣
↪300G  868M  300G   1% /usr/share/nginx
192.168.0.19:/nfs/default-conf-web-0-pvc-b31d6419-74b3-11e9-9f21-fa163ebf1f8e             ␣
↪300G  868M  300G   1% /usr/share/nginx/conf
tmpfs                                                                                     3.
↪9G   12K  3.9G   1% /run/secrets/kubernetes.io/serviceaccount
```

(continues on next page)

(continued from previous page)

```
tmpfs                                                                                3.
↪9G     0  3.9G   0% /proc/acpi
tmpfs                                                                                3.
↪9G     0  3.9G   0% /proc/scsi
tmpfs                                                                                3.
↪9G     0  3.9G   0% /sys/firmware

root@web-0:/# ls -l /usr/share/nginx
total 0
drwxrwxrwx 2 root root  6 May 12 12:44 conf
drwxr-xr-x 2 root root 24 May 13 09:12 html

root@web-0:/# ls -l /usr/share/nginx/html/
total 32
-rw-r--r-- 1 root root 30802 May 13 09:12 index.html

root@web-0:/# head /usr/share/nginx/html/index.html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Welcome to Cloud Consultancy Team &#8212; tsi-cc 1.0 documentation</title>
    <link rel="stylesheet" href="_static/classic.css" type="text/css" />
```

By the way, the same *index.html* can also be accessed via S3. Here is what the link may look like:

```
http://45.86.170.94:30968/html/index.html?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
↪Credential=davidyuan%2F20190513%2F%2Fs3%2Faws4_request&X-Amz-Date=20190513T102759Z&
↪X-Amz-Expires=604800&X-Amz-SignedHeaders=host&X-Amz-
↪Signature=297542c0b696b6980acd9252e35da7604623006334ef9b20d028c7b736217ae8
```

The metadata for S3 protocol can be found under */usr/share/nginx/.minio.sys*:

```
root@web-0:/usr/share/nginx/.minio.sys# ls -la
total 4
drwxr-xr-x 6 root root 82 May 13 09:12 .
drwxrwxrwx 5 root root 48 May 13 09:11 ..
drwxr-xr-x 3 root root 18 May 13 09:12 buckets
drwxr-xr-x 2 root root 25 May 13 08:37 config
-rw------- 1 root root 94 May 13 08:37 format.json
drwxr-xr-x 2 root root  6 May 13 08:37 multipart
drwxr-xr-x 3 root root 50 May 13 08:37 tmp
```

8.29 Kubernetes Practical

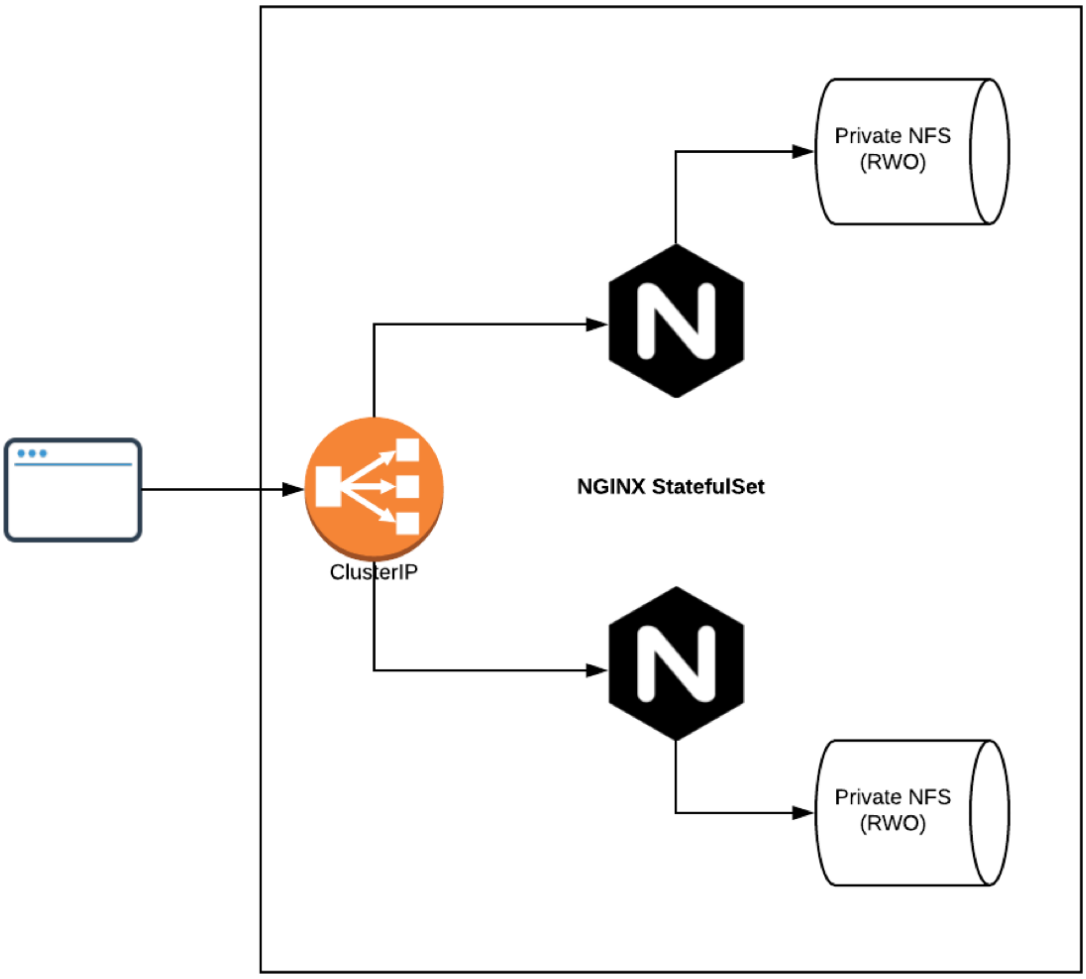
This practical provides step-by-step guide to create a Web front-end in Kubernetes. It is designed to help a developer new to Kubernetes to create a work environment from scratch. A virtual machine running Ubuntu 18.04 LTS is assumed. The following exercises are built on top of each other:

1. *Reading 0: Adding Minikube to the new VMs*

2. *Exercise 0.0: Starting Minikube*
3. *Exercise 0.1: (Optional) Enabling GUI for VNC*
4. *Exercise 1: Creating NGINX*
5. *Exercise 2: Adding HTML to pods*
6. *Exercise 3: Persistence and disaster recovery*
7. *Exercise 4: Problem determination step-by-step*
8. *Homework: Additional exercises after the workshop*

This practical intends to use a simple example to demonstrate the features and characteristics of Kubernetes as an orchestrator for container-based workloads. You should be able to create your own work environment and to start implementing your own project after finishing this practical.

The code for the practical is under https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/tree/master/tsi-cc/ResOps/scripts. The deployment would look like the following once it is completed:



8.29.1 Reading 0: Adding Minikube to the new VMs

This original exercise was designed to show user how to build a sandbox for an individual developer. This is automated to give users more time to focus on cloud-specific subjects. Read through this section so that you can build your own sandbox after the workshop.

Kubernetes has comprehensive documentation on if and how to use Minikube https://kubernetes.io/docs/setup/minikube/.

Access the VM assigned to you from your SSH client. The command is *ssh <user_id>@<IP_of_VM_assigned_to_you>*. Your user ID, password and IP address will be given to you in the workshop.

Install Docker as the runtime for Minikube:

```
sudo apt-get update
sudo apt install -y docker.io
```

Install kubectl as the client to access the Kubernetes cluster in Minikube:

```
sudo snap install kubectl --classic
```

Download, install and configure Minikube:

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-
↪linux-amd64 && chmod +x minikube
sudo cp minikube /usr/local/bin && rm minikube
```

Note that /home/ubuntu/.kube/config is owned by root. Run *chown* to avoid using *sudo*:

```
sudo chown -R $USER $HOME/.kube $HOME/.minikube
```

If a sudor other than ubuntu is used, move the following two directories:

```
sudo mv /home/ubuntu/.kube /home/ubuntu/.minikube $HOME
```

8.29.2 Exercise 0.0: Starting Minikube

Start Minikube with Docker:

```
minikube start --vm-driver='docker'
```

The following message should be displayed when Minikube is started successfully:

```
  minikube v1.17.1 on Ubuntu 18.04 (amd64)
  Using the docker driver based on user configuration
  Starting control plane node minikube in cluster minikube
  Pulling base image ...
  Downloading Kubernetes v1.20.2 preload ...
    > preloaded-images-k8s-v8-v1....: 491.22 MiB / 491.22 MiB  100.00% 18.07 Mi
  Creating docker container (CPUs=2, Memory=2200MB) ...
  This container is having trouble accessing https://k8s.gcr.io
  To pull new external images, you may need to configure a proxy: https://minikube.
↪sigs.k8s.io/docs/reference/networking/proxy/
  Preparing Kubernetes v1.20.2 on Docker 20.10.2 ...
     Generating certificates and keys ...
```

(continues on next page)

(continued from previous page)

```
    Booting up control plane ...
    Configuring RBAC rules ...
  Verifying Kubernetes components...
  Enabled addons: storage-provisioner, default-storageclass
  Done! kubectl is now configured to use "minikube" cluster and "default" namespace␣
↪by default
```

Enable addons such as metrics-server and ingress by running the following:

```
minikube addons enable metrics-server
minikube addons enable ingress
```

Verify the right addons are enabled:

```
minikube addons list

|-----------------------------|----------|--------------|
|         ADDON NAME          | PROFILE  |    STATUS    |
|-----------------------------|----------|--------------|
| ambassador                  | minikube | disabled     |
| csi-hostpath-driver         | minikube | disabled     |
| dashboard                   | minikube | disabled     |
| default-storageclass        | minikube | enabled    |
| efk                         | minikube | disabled     |
| freshpod                    | minikube | disabled     |
| gcp-auth                    | minikube | disabled     |
| gvisor                      | minikube | disabled     |
| helm-tiller                 | minikube | disabled     |
| ingress                     | minikube | enabled    |
| ingress-dns                 | minikube | disabled     |
| istio                       | minikube | disabled     |
| istio-provisioner           | minikube | disabled     |
| kubevirt                    | minikube | disabled     |
| logviewer                   | minikube | disabled     |
| metallb                     | minikube | disabled     |
| metrics-server              | minikube | enabled    |
| nvidia-driver-installer     | minikube | disabled     |
| nvidia-gpu-device-plugin    | minikube | disabled     |
| olm                         | minikube | disabled     |
| pod-security-policy         | minikube | disabled     |
| registry                    | minikube | disabled     |
| registry-aliases            | minikube | disabled     |
| registry-creds              | minikube | disabled     |
| storage-provisioner         | minikube | enabled    |
| storage-provisioner-gluster | minikube | disabled     |
| volumesnapshots             | minikube | disabled     |
|-----------------------------|----------|--------------|
```

Verify that Minicube is working and you can access it. You should see the following message:

```
kubectl get node

NAME       STATUS   ROLES                  AGE   VERSION
minikube   Ready    control-plane,master   12m   v1.20.2
```

Now, you have a Kubernetes environment to develop and to test your workload. Never use it for production though.

8.29.3 Exercise 0.1: (Optional) Enabling GUI for VNC

The xfce4 is a light weight GUI desktop. We use it together with VNC so that a web browser can be used on the VM. If this section is skipped, use *curl* for the rest of exercises.

Install xfce4 with the following command:

```
sudo apt install -y xfce4 xfce4-goodies
```

Answer *Yes* to the question "Restart services during package upgrades without asking?"

Start VNC server with the preferred geometry and color depth, for example:

```
vnc4server :1 -geometry 1600x900 -depth 24
```

Start another SSH session to enable SSH port forward with **your own** user ID and IP address, for example:

```
ssh resops6@45.86.170.124 -C -L 5901:127.0.0.1:5901
```

Start your favourite VNC client, for example *Screen Sharing* on Mac OS. Connect to the VNC server as `localhost:5901` and with your logon password. Make sure to select "Use default configuration" when the desk initialises for the first time.

8.29.4 Exercise 1: Creating NGINX

Create a Kubernetes manifest file with the following command:

```
nano ~/nginx.yml
```

You can copy and paste the manifest from https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/raw/master/tsi-cc/ResOps/scripts/minikube/nginx.yml to the editor. Review the file carefully to see what resources are to be created and how parts are connected with each other.

nginx.yml 831 Bytes

Edit Web IDE

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  type: ClusterIP
  ports:
    - port: 80
      name: web
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: "nginx"
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
              name: web
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
    - metadata:
        name: www
      spec:
        accessModes: [ "ReadWriteOnce" ]
        resources:
          requests:
            storage: 1Gi
        storageClassName: standard
```

This manifest creates a StatefulSet with a corresponding service, where NGINX will be listening on port 80 for HTTP. There are two pods to be created in the set for load balancing, minimising downtime and to certain extend disaster recovering. The container NGINX is pulled from Docker Hub. It will be started to serve HTML pages from persistent volumes private to each pod.

Apply the manifest to create a service and statefulset:

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl apply -f nginx.yml
service/nginx created
statefulset.apps/web created
```

This creates additional resources needed to for a cluster of NGINX servers(persistent volume claims, persistent vol-

umes, pods, and services):

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pvc
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS␣
↪MODES   STORAGECLASS   AGE
www-web-0   Bound    pvc-4cd5c31b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO␣
↪    standard       39s
www-web-1   Bound    pvc-5ac9090b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO␣
↪    standard       16s

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY␣
↪STATUS   CLAIM               STORAGECLASS   REASON   AGE
pvc-4cd5c31b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO            Delete␣
↪Bound    default/www-web-0   standard                41s
pvc-5ac9090b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO            Delete␣
↪Bound    default/www-web-1   standard                18s

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pod
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          52s
web-1   1/1     Running   0          29s

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   61m
nginx        ClusterIP   10.104.206.229   <none>        80/TCP    76s
```

You now have a cluster of two NGINX pods running in Minicube. They are listening on port 80 with one cluster IP. You have just built a web service infrastructure with redundancy managed by Kubernetes.

8.29.5 Exercise 2: Adding HTML to pods

Note: if you completed *Exercise 0.1: (Optional) Enabling GUI for VNC*, you can use *Firefox* on your desktop instead of *curl* for all the exercises.

Each of the two NGINX server has its own storage, running in its own pod but share the same cluster IP, for example *10.104.206.229*. The cluster IP is listed with *kubectl get svc* in the previous exercise. If you try to access the home page, you will get HTTP403:

```
ubuntu@resops-1-k8s-node-nf-1:~$ curl http://10.104.206.229
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>
```

Let's fix this on one of the two NGINX servers. Recall that the volume *www* is mounted on */usr/share/nginx/html/*, which is the default document root for NGINX server. Connect to pod web-0 and you can see that there is no file to be served by NGINX. That's why the HTTP403 is sent back:

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl exec -it web-0 -- /bin/bash
root@web-0:/# cd /usr/share/nginx/html/
root@web-0:/usr/share/nginx/html# ls -la
```

(continues on next page)

(continued from previous page)

```
total 8
drwxrwxrwx 2 root root 4096 May 15 13:49 .
drwxr-xr-x 3 root root 4096 May  8 03:01 ..
```

Create a simple */usr/share/nginx/html/index.html* with a title and heading "Hello World from web-0":

```
cat <<EOF > /usr/share/nginx/html/index.html
<html>
<head><title>Hello World from web-0</title></head>
<body>
<center><h1>Hello World from web-0</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>
EOF
```

Exit out of the pod web-0:

```
root@web-0:/usr/share/nginx/html# exit
exit
```

Repeat the process in pod web-1. It is important to use a title and heading "Hello World from web-1" so that the two pods have two different index.html files:

```
kubectl exec -it web-1 -- /bin/bash

cat <<EOF > /usr/share/nginx/html/index.html
<html>
<head><title>Hello World from web-1</title></head>
<body>
<center><h1>Hello World from web-1</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>
EOF
```

Exit out of the pod web-1. Send HTTP request to the cluster IP again. You will find that the two NGINX servers take turns to serve the home page. The HTTP 403 error is gone:

```
ubuntu@resops-1-k8s-node-nf-1:~$ curl http://10.104.206.229
<html>
<head><title>Hello World from web-1</title></head>
<body>
<center><h1>Hello World from web-1</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>

ubuntu@resops-1-k8s-node-nf-1:~$ curl http://10.104.206.229
<html>
<head><title>Hello World from web-0</title></head>
<body>
<center><h1>Hello World from web-0</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>
```

Note that Kubernetes tend to route the requests to the same pod for better performance. You may keep seeing your HTML page served from the same pod, for example web-0. If this is happening, rename the index.html in web-0. Then you will see the page gets served from the other pod web-1:

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl exec -it web-0 -- /bin/bash
root@web-0:/# mv /usr/share/nginx/html/index.html /usr/share/nginx/html/index.html.bak
root@web-0:/# exit
exit

ubuntu@resops-1-k8s-node-nf-1:~$ curl http://10.104.206.229
<html>
<head><title>Hello World from web-1</title></head>
<body>
<center><h1>Hello World from web-1</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>
```

Change the index.html page in web-0 back:

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl exec -it web-0 -- /bin/bash
root@web-0:/# mv /usr/share/nginx/html/index.html.bak /usr/share/nginx/html/index.html
root@web-0:/# exit
exit
```

From exercises 2 & 3, we understand that the NGINX serves independent copies from */usr/share/nginx/html* persisted on two separate volumes via two pods web-0 and web-1. See output by *kubectl get* commands in exercise 2 and *curl http://10.104.206.229* command in exercise 3 again.

8.29.6 Exercise 3: Persistence and disaster recovery

If a stateful set is removed, the persistent volumes are preserved by design. Run *kubectl delete -f nginx.yml* to simulate scheduled outage. The service and pods are deleted but persistent volumes are saved:

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl delete -f nginx.yml
service "nginx" deleted
statefulset.apps "web" deleted

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pvc
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS␣
↪MODES   STORAGECLASS   AGE
www-web-0   Bound    pvc-4cd5c31b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO␣
↪    standard       110m
www-web-1   Bound    pvc-5ac9090b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO␣
↪    standard       110m

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY␣
↪STATUS   CLAIM               STORAGECLASS   REASON   AGE
pvc-4cd5c31b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO            Delete␣
↪Bound    default/www-web-0   standard                110m
pvc-5ac9090b-7718-11e9-8b0b-fa163ede6c1a   1Gi        RWO            Delete␣
↪Bound    default/www-web-1   standard                110m

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
```

(continues on next page)

(continued from previous page)

```
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   173m

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pod
No resources found.
```

Run *kubectl apply -f nginx.yml* to recreate the stateful set and service. The each new pod recreated will mount its original volume as if it was never deleted. The pod web-0 still has the index.html with the message of "Hello World from web-0":

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl apply -f nginx.yml
service/nginx created
statefulset.apps/web created

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pod
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          14s
web-1   1/1     Running   0          7s

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl exec -it web-0 -- /bin/bash
root@web-0:/# cat /usr/share/nginx/html/index.html
<html>
<head><title>Hello World from web-0</title></head>
<body>
<center><h1>Hello World from web-0</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>

root@web-0:/# exit
exit
```

Run *kubectl delete pod web-1* to simulate unscheduled outage. The recovery happens really fast. We need to chain two kubectl command to see what is happening:

```
kubectl delete pod web-1 && kubectl get pod
```

Kubernete tries to restart pod web-1 immediately. After a little while web-1 will be running again as if noting happened. It will be mounted to its original volume. The index.html in used by pod web-1 still has the same message of "Hello World from web-1":

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl delete pod web-1 && kubectl get pod
pod "web-1" deleted
NAME    READY   STATUS              RESTARTS   AGE
web-0   1/1     Running             0          6m50s
web-1   0/1     ContainerCreating   0          0s

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl get pod
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          11m
web-1   1/1     Running   0          4m27s

ubuntu@resops-1-k8s-node-nf-1:~$ kubectl exec -it web-1 -- /bin/bash
root@web-1:/# cat /usr/share/nginx/html/index.html
<html>
<head><title>Hello World from web-1</title></head>
<body>
```

(continues on next page)

(continued from previous page)

```
<center><h1>Hello World from web-1</h1></center>
<hr><center>nginx/1.15.12</center>
</body>
</html>
```

As you can see, Kubernetes restarts web-1 immediately. The newly started pod still mount to the same persistent volume as if the pod was never killed.

8.29.7 Exercise 4: Problem determination step-by-step

Frequently, a manifest file that we have created does not behave as we expected or contains bugs. Here is a micky-mouse example how we can investigate what is going on.

First, repeat the first step in the previous exercise to remove the deployment:

```
ubuntu@resops-1-k8s-node-nf-1:~$ kubectl delete -f nginx.yml
service "nginx" deleted
statefulset.apps "web" deleted
```

Run *kubernetes get* commands to confirm if the resources are deleted. Check the instructions in the previous exercise if you do not remember the details.

Second, inspect https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc/blob/master/tsi-cc/ResOps/scripts/minikube/mickymouse.yml to see if you can spot any problems in the manifest.

Third, apply mickymouse.yml:

```
resops49@resops-k8s-node-17:~$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/minikube/mickymouse.yml
service/nginx created
statefulset.apps/web created
```

Cool! Or, is it? Let's repeat the *kubernetes get* commands to see if there is anything unusual. Here is what you would likely see:

```
resops49@resops-k8s-node-17:~$ kubectl get pvc
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS␣
↪MODES   STORAGECLASS   AGE
www-web-0   Bound    pvc-4290e18c-d2b1-4714-b960-2f892121fd5b   1Gi        RWO␣
↪    standard       6m
www-web-1   Bound    pvc-dcbd9627-9b02-4d5a-8be1-c40045c0670c   1Gi        RWO␣
↪    standard       5m48s

resops49@resops-k8s-node-17:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY␣
↪STATUS   CLAIM               STORAGECLASS   REASON   AGE
pvc-4290e18c-d2b1-4714-b960-2f892121fd5b   1Gi        RWO            Delete␣
↪Bound    default/www-web-0   standard                6m6s
pvc-dcbd9627-9b02-4d5a-8be1-c40045c0670c   1Gi        RWO            Delete␣
↪Bound    default/www-web-1   standard                5m54s

resops49@resops-k8s-node-17:~$ kubectl get pod
No resources found.

resops49@resops-k8s-node-17:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
```

(continues on next page)

(continued from previous page)

```
kubernetes    ClusterIP    10.96.0.1        <none>        443/TCP         6d22h
nginx         NodePort     10.107.235.150   <none>        80:31541/TCP    3m54s
```

We are expecting two pods running but nothing is created. PVs and PVCs are bounded. However, you would notice the mismatch of the timestamp between Service and PersistentVolumes. This is because, again, StatefulSet always leaves PersistentVolumes untouched by design even if the reclaim policy is *delete*. We need to delete these volumes explicitly to fully understand how naughty mickymouse.yml is:

```
resops49@resops-k8s-node-17:~$ kubectl delete -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/minikube/mickymouse.yml
service "nginx" deleted
statefulset.apps "web" deleted

resops49@resops-k8s-node-17:~$ kubectl delete pvc www-web-0
persistentvolumeclaim "www-web-0" deleted

resops49@resops-k8s-node-17:~$ kubectl delete pvc www-web-1
persistentvolumeclaim "www-web-1" deleted

resops49@resops-k8s-node-17:~$ kubectl get pv
No resources found.

resops49@resops-k8s-node-17:~$ kubectl get svc
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes    ClusterIP   10.96.0.1    <none>        443/TCP   6d23h
```

Because of the reclaim policy, deleting the PVCs cleans out PVs as well. Now the environment is clean. Let's apply mickymouse.yml, again:

```
resops49@resops-k8s-node-17:~$ kubectl apply -f https://gitlab.ebi.ac.uk/TSI/tsi-
↪ccdoc/raw/master/tsi-cc/ResOps/scripts/minikube/mickymouse.yml
service/nginx created
statefulset.apps/web created

resops49@resops-k8s-node-17:~$ kubectl get pvc
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS␣
↪MODES   STORAGECLASS   AGE
www-web-0   Bound    pvc-9d3be2c4-3605-4f86-95d2-0e1f7328ded4   1Gi        RWO␣
↪    standard       45s

resops49@resops-k8s-node-17:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY␣
↪STATUS   CLAIM               STORAGECLASS   REASON   AGE
pvc-9d3be2c4-3605-4f86-95d2-0e1f7328ded4   1Gi        RWO            Delete␣
↪Bound    default/www-web-0   standard                54s

resops49@resops-k8s-node-17:~$ kubectl get pod
No resources found.

resops49@resops-k8s-node-17:~$ kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        6d23h
nginx         NodePort    10.105.177.74   <none>        80:30482/TCP   74s
```

The Service and StatefulSet seem created successfully again. The timestamps of Service, PV and PVC are close enough now. We still do not have any pods. In addition, we only see one pair of PV and PVC instead of two pairs. Things are going from bad to worse.

Fourth, check the events recorded by Kubernetes to see if there have been any failures. The event log can be long and messy. Always remember to sort the events by last seen or to apply various filters. See *kubectl get –help* to see how to apply filters:

```
resops49@resops-k8s-node-17:~$ kubectl get event --sort-by='{.metadata.
↪creationTimestamp}'
LAST SEEN   TYPE      REASON                  OBJECT                                  ␣
↪MESSAGE
60m         Normal    Starting                node/minikube                           ␣
↪Starting kube-proxy.
60m         Normal    RegisteredNode          node/minikube                            Node␣
↪minikube event: Registered Node minikube in Controller
46m         Warning   FailedScheduling        pod/web-0                                pod␣
↪has unbound immediate PersistentVolumeClaims
46m         Normal    ProvisioningSucceeded   persistentvolumeclaim/www-web-0         ␣
↪Successfully provisioned volume pvc-4290e18c-d2b1-4714-b960-2f892121fd5b
46m         Normal    Provisioning            persistentvolumeclaim/www-web-0         ␣
↪External provisioner is provisioning volume for claim "default/www-web-0"
46m         Normal    ExternalProvisioning    persistentvolumeclaim/www-web-0         ␣
↪waiting for a volume to be created, either by external provisioner "k8s.io/minikube-
↪hostpath" or manually created by system administrator
46m         Normal    SuccessfulCreate        statefulset/web                         ␣
↪create Pod web-0 in StatefulSet web successful
46m         Normal    SuccessfulCreate        statefulset/web                         ␣
↪create Claim www-web-0 Pod web-0 in StatefulSet web success
46m         Normal    Scheduled               pod/web-0                               ␣
↪Successfully assigned default/web-0 to minikube
46m         Normal    Pulling                 pod/web-0                               ␣
↪Pulling image "nginx"
46m         Normal    Pulled                  pod/web-0                               ␣
↪Successfully pulled image "nginx"
46m         Normal    SuccessfulCreate        statefulset/web                         ␣
↪create Claim www-web-1 Pod web-1 in StatefulSet web success
46m         Normal    ExternalProvisioning    persistentvolumeclaim/www-web-1         ␣
↪waiting for a volume to be created, either by external provisioner "k8s.io/minikube-
↪hostpath" or manually created by system administrator
46m         Normal    Provisioning            persistentvolumeclaim/www-web-1         ␣
↪External provisioner is provisioning volume for claim "default/www-web-1"
46m         Normal    ProvisioningSucceeded   persistentvolumeclaim/www-web-1         ␣
↪Successfully provisioned volume pvc-dcbd9627-9b02-4d5a-8be1-c40045c0670c
46m         Normal    SuccessfulCreate        statefulset/web                         ␣
↪create Pod web-1 in StatefulSet web successful
46m         Warning   FailedScheduling        pod/web-1                                pod␣
↪has unbound immediate PersistentVolumeClaims
46m         Normal    Created                 pod/web-0                               ␣
↪Created container nginx
46m         Normal    Started                 pod/web-0                               ␣
↪Started container nginx
46m         Normal    Scheduled               pod/web-1                               ␣
↪Successfully assigned default/web-1 to minikube
46m         Normal    Pulling                 pod/web-1                               ␣
↪Pulling image "nginx"
46m         Normal    Pulled                  pod/web-1                               ␣
↪Successfully pulled image "nginx"
46m         Normal    Started                 pod/web-1                               ␣
↪Started container nginx
46m         Normal    Created                 pod/web-1                               ␣
↪Created container nginx
```

(continues on next page)

(continued from previous page)

```
45m         Normal    Killing                 pod/web-1                                  ␣
↪Stopping container nginx
45m         Normal    Killing                 pod/web-0                                  ␣
↪Stopping container nginx
32m         Warning   FailedCreate            statefulset/web                            ␣
↪create Pod web-0 in StatefulSet web failed error: Pod "web-0" is invalid: spec.
↪containers[0].volumeMounts[0].name: Not found: "html"
18m         Normal    ExternalProvisioning    persistentvolumeclaim/www-web-0            ␣
↪waiting for a volume to be created, either by external provisioner "k8s.io/minikube-
↪hostpath" or manually created by system administrator
18m         Normal    Provisioning            persistentvolumeclaim/www-web-0            ␣
↪External provisioner is provisioning volume for claim "default/www-web-0"
18m         Normal    ProvisioningSucceeded   persistentvolumeclaim/www-web-0            ␣
↪Successfully provisioned volume pvc-9d3be2c4-3605-4f86-95d2-0e1f7328ded4
18m         Normal    SuccessfulCreate        statefulset/web                            ␣
↪create Claim www-web-0 Pod web-0 in StatefulSet web success
7m5s        Warning   FailedCreate            statefulset/web                            ␣
↪create Pod web-0 in StatefulSet web failed error: Pod "web-0" is invalid: spec.
↪containers[0].volumeMounts[0].name: Not found: "html"
```

From the event log, we can see an error message *create Pod web-0 in StatefulSet web failed error: Pod "web-0" is invalid: spec.containers[0].volumeMounts[0].name: Not found: "html"*. Checking the *mickymouse.yml* by following the path *spec.containers[0].volumeMounts[0].name*, we see a mismatch of names between *volumeMounts.name=html* and *volumeClaimTemplates.metadata.name=www*. This needs to be fixed.

Delete everything created by mickymouse.yml. Remember to delete the PVC as well. By now, you should be able to find relevant commands in this practical to complete the task.

Fifth, clone the repository and start modifying mickymouse.yml locally:

```
resops49@resops-k8s-node-17:~$ git clone https://gitlab.ebi.ac.uk/TSI/tsi-ccdoc.git
Cloning into 'tsi-ccdoc'...
remote: Enumerating objects: 388, done.
remote: Counting objects: 100% (388/388), done.
remote: Compressing objects: 100% (196/196), done.
remote: Total 4488 (delta 200), reused 302 (delta 145)
Receiving objects: 100% (4488/4488), 102.69 MiB | 30.49 MiB/s, done.
Resolving deltas: 100% (1904/1904), done.

resops49@resops-k8s-node-17:~$ cd tsi-ccdoc/tsi-cc/ResOps/scripts/minikube/

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ nano ./
↪mickymouse.yml
```

Change name from *html* to *www*. Apply mickymouse.yml to see if the problem is fixed. Things certainly look much better. Everything seems created successfully. Is everything working? If not, why? How are you going to find out? How are you going to fix the problems?

```
resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl apply␣
↪-f ./mickymouse.yml
service/nginx created
statefulset.apps/web created

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY  ␣
↪STATUS   CLAIM               STORAGECLASS   REASON   AGE
pvc-8b5f0d99-f4dd-461d-8f46-2514d23e784c   1Gi        RWO            Delete          ␣
↪Bound    default/www-web-0   standard                8s
```

(continues on next page)

(continued from previous page)

```
pvc-ad55fa2d-d194-4b58-8b6c-815da976d038   1Gi        RWO            Delete
↪Bound    default/www-web-1   standard                1s

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl get
↪pvc
NAME         STATUS   VOLUME                                     CAPACITY   ACCESS
↪MODES   STORAGECLASS   AGE
www-web-0    Bound    pvc-8b5f0d99-f4dd-461d-8f46-2514d23e784c   1Gi        RWO
↪    standard       18s
www-web-1    Bound    pvc-ad55fa2d-d194-4b58-8b6c-815da976d038   1Gi        RWO
↪    standard       11s

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl get
↪pod
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          27s
web-1   1/1     Running   0          20s

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl get
↪svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        7d
nginx        NodePort    10.106.111.118   <none>        80:31542/TCP   55s
```

8.29.8 Homework: Additional exercises after the workshop

There are two more problems and one enhancement for your exercise after the workshop. If you get stuck, compare *mickymouse.yml* against *nginx.yml*. It might be easier if you know what you are looking for.

Here are some general guidance:

1. Check the event log again to confirm that deployment was successful. Use *kubectl get event –help* to learn how to use it.
2. Check the runtime logs to see if there are errors. Use *kubectl logs –help* to learn how to use it.
3. Explore on *Kubernetes Dashboard* to see if you can spot anything unusual. Note that Kubernetes Dashboard would not work in the sandbox as the VM does not have GUI enabled.
4. Connect to pods and containers in the pods to see if everything is configured properly.
5. Test runtime behaviour to see if the system acts as designed. In this case, run *curl http://* to see what happens.

Here are some steps to look for issues. You may wan to continue working on this after the workshop:

```
resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl get
↪event --sort-by='{.metadata.creationTimestamp}'
LAST SEEN   TYPE      REASON                  OBJECT
↪MESSAGE
34m         Warning   FailedCreate            statefulset/web
↪create Pod web-0 in StatefulSet web failed error: Pod "web-0" is invalid: spec.
↪containers[0].volumeMounts[0].name: Not found: "html"
10m         Normal    SuccessfulCreate        statefulset/web
↪create Claim www-web-0 Pod web-0 in StatefulSet web success
10m         Normal    ExternalProvisioning    persistentvolumeclaim/www-web-0
↪waiting for a volume to be created, either by external provisioner "k8s.io/minikube-
↪hostpath" or manually created by system administrator
```

(continues on next page)

(continued from previous page)

```
10m         Normal    ProvisioningSucceeded   persistentvolumeclaim/www-web-0   ␣
↪Successfully provisioned volume pvc-8b5f0d99-f4dd-461d-8f46-2514d23e784c
10m         Normal    Provisioning            persistentvolumeclaim/www-web-0   ␣
↪External provisioner is provisioning volume for claim "default/www-web-0"
10m         Warning   FailedScheduling        pod/web-0                          pod␣
↪has unbound immediate PersistentVolumeClaims
10m         Normal    SuccessfulCreate        statefulset/web                    ␣
↪create Pod web-0 in StatefulSet web successful
10m         Normal    Scheduled               pod/web-0                          ␣
↪Successfully assigned default/web-0 to minikube
10m         Normal    Pulling                 pod/web-0                          ␣
↪Pulling image "nginx"
10m         Normal    Pulled                  pod/web-0                          ␣
↪Successfully pulled image "nginx"
10m         Normal    Created                 pod/web-0                          ␣
↪Created container nginx
10m         Normal    Started                 pod/web-0                          ␣
↪Started container nginx
10m         Normal    ExternalProvisioning    persistentvolumeclaim/www-web-1   ␣
↪waiting for a volume to be created, either by external provisioner "k8s.io/minikube-
↪hostpath" or manually created by system administrator
10m         Normal    Provisioning            persistentvolumeclaim/www-web-1   ␣
↪External provisioner is provisioning volume for claim "default/www-web-1"
10m         Warning   FailedScheduling        pod/web-1                          pod␣
↪has unbound immediate PersistentVolumeClaims
10m         Normal    SuccessfulCreate        statefulset/web                    ␣
↪create Claim www-web-1 Pod web-1 in StatefulSet web success
10m         Normal    SuccessfulCreate        statefulset/web                    ␣
↪create Pod web-1 in StatefulSet web successful
10m         Normal    ProvisioningSucceeded   persistentvolumeclaim/www-web-1   ␣
↪Successfully provisioned volume pvc-ad55fa2d-d194-4b58-8b6c-815da976d038
10m         Normal    Scheduled               pod/web-1                          ␣
↪Successfully assigned default/web-1 to minikube
10m         Normal    Pulling                 pod/web-1                          ␣
↪Pulling image "nginx"
10m         Normal    Created                 pod/web-1                          ␣
↪Created container nginx
10m         Normal    Pulled                  pod/web-1                          ␣
↪Successfully pulled image "nginx"
10m         Normal    Started                 pod/web-1                          ␣
↪Started container nginx

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl logs␣
↪web-0 --all-containers=true

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl exec -
↪it web-0 -- bash
root@web-0:/# ls -l /usr/share/nginx/html
total 0

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ kubectl get␣
↪svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        7d
nginx        NodePort    10.106.111.118   <none>        80:31542/TCP   15m

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ curl http://
↪10.106.111.118
```

(continues on next page)

(continued from previous page)

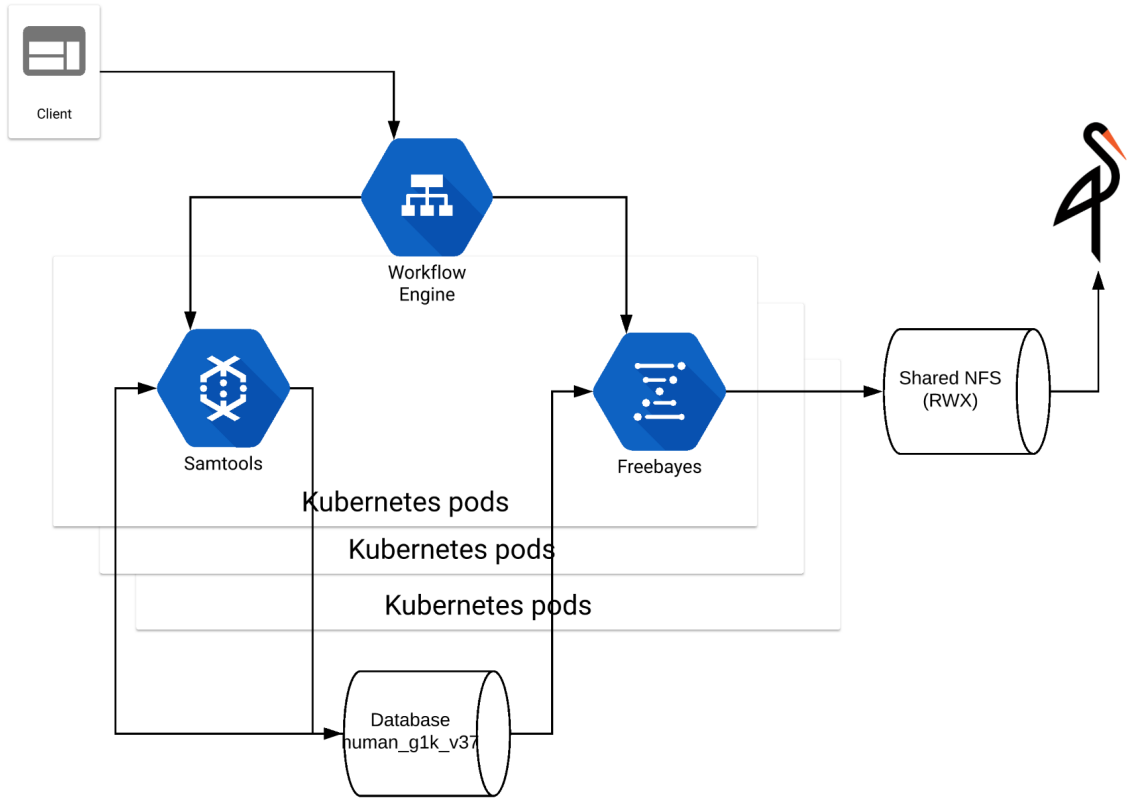
```
curl: (7) Failed to connect to 10.106.111.118 port 80: Connection refused

resops49@resops-k8s-node-17:~/tsi-ccdoc/tsi-cc/ResOps/scripts/minikube$ curl http://
↪10.106.111.118:31542
curl: (7) Failed to connect to 10.106.111.118 port 31542: Connection timed out
```

8.30 Scaling up Kubernetes for research pipelines

This practical is trying to assist you to apply basic skills in Kubernetes. Hopefully, you would be able to use the sample scripts and general ideas in your projects directly. For research pipelines, there are some major considerations when moving to the clouds:

- *Reading 0: Preparing Minikube VM to support NFS volumes*
- Accessing large amount of data from the source:
  - *Exercise 1: ReadWriteMany for shared output*
  - *Exercise 2: ReadOnlyMany for data source*
  - *Exercise 3: ReadWriteOnce for private workspace*
  - *Exercise 4: Initialising persistent volumes*
  - *Exercise 5: Kubernetes secret & S3 interface*
- Scaling up:
  - *Exercise 6: Horizontal scaling*
  - *Reading 1: Vertical scaling*
  - *Exercise 7: Autoscaling*

This practical is based on a project at EBI. We are creating the pipeline on both RKE and GKE. This practical is only focusing on RKE. The objective is to create a pipeline for variant calling on Kubernetes. Kubernetes need to be scaled up to schedule a large number of jobs. The containers such as Samtools and Freebayes read and write on persistent volumes serving as database and S3 buckets.



8.30.1 Reading 0: Preparing Minikube VM to support NFS volumes

This original exercise was designed to show user how to build a sandbox for an individual developer. This is automated to give users more time to focus on cloud-specific subjects. Read through this section so that you can build your own sandbox after the workshop. This is an continuation built on top of Reading 0: Adding Minikube to the new VMs.

Access the VMs via SSH directly if they have public IPs attached. Otherwise, use SSH tunnel via bastion server, for example *ssh -i ~/.ssh/id_rsa -o UserKnownHostsFile=/dev/null -o ProxyCommand="ssh -W %h:%p -i ~/.ssh/id_rsa ubuntu@193.62.54.185" ubuntu@10.0.0.5*

Helm needs port-forward enabled by *socat*. NFS mount needs *nfs-common* on the work nodes. Install the following packages:

```
sudo apt-get install -y socat nfs-common
```

8.30.2 Exercise 1: ReadWriteMany for shared output

Git clone the project https://gitlab.ebi.ac.uk/davidyuan/adv-k8s before starting the exercises and readings. If you have an Kubernetes cluster of your own, you can try all the code in readings. Otherwise, only try the exercises on Minikube:

```
cd ~
git clone https://gitlab.ebi.ac.uk/davidyuan/adv-k8s.git
```

Exercises assume that the git repository is cloned to *~/adv-k8s/* from now on.

The output should be sent out directly from the pipeline if possible. However, most of the pipelines assume local POSIX file systems for output. It is necessary to define a shared storage for multiple pods. Unfortunately, Azure is the only cloud with native storage supporting ReadWriteMany. Kubernetes has a detailed list of kinds of volumes and access modes supported. Use it creatively, you should be able to avoid copying data. NFS is necessary evil. Use it only if it is unavoidable.

However, Minikube does not have the storage class nfs-client:

```
resops25@resops-k8s-node-4:~/adv-k8s$ kubectl get storageclass
NAME                 PROVISIONER                AGE
standard (default)   k8s.io/minikube-hostpath   47h
```

We are to create a toy NFS server providing such storage class on Minikube by running *~/adv-k8s/osk/nfs-server.sh*. Provide your password when prompted. After a little while, you should see messages ending with the following:

```
Waiting for 1 pods to be ready...
partitioned roll out complete: 1 new pods have been updated...
NAME                 PROVISIONER                                     AGE
nfs-client           cluster.local/nfs-nfs-server-provisioner   22s
standard (default)   k8s.io/minikube-hostpath                   47h
```

The Helm chart that we are using here is a sandbox for development purposes. Never try to use it for production.

Open https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/pvc-workspace.yml to see a PersistentVolumeClaim of 50 GB is made to the storage class:

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: shared-workspace
spec:
  storageClassName: nfs-client
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 50Gi
```

Apply the *PersistentVolumeClaim* to get the storage class to allocate an NFS volume. Note that a new PVC is with access mode *RWX* is created and bounded to a new PV with access mode *RWX*:

```
ubuntu@resops-k8s-node-nf-2:~/adv-k8s$ kubectl apply -f ~/adv-k8s/osk/dpm/pvc-
↪workspace.yml
persistentvolumeclaim/shared-workspace created

ubuntu@resops-k8s-node-nf-2:~/adv-k8s$ kubectl get pvc
NAME               STATUS   VOLUME                                     CAPACITY
↪ACCESS MODES   STORAGECLASS   AGE
shared-workspace   Bound    pvc-5ae9a98b-4669-47fb-8a5b-8e5b95a74936   50Gi
↪           nfs-client     28s

ubuntu@resops-k8s-node-nf-2:~/adv-k8s$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
↪STATUS   CLAIM                       STORAGECLASS   REASON   AGE
pvc-5ae9a98b-4669-47fb-8a5b-8e5b95a74936   50Gi       RWX            Delete
↪Bound    default/shared-workspace   nfs-client              35s
```

In the pod template in https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/freebayes.yml, refer to this shared volume claim:

```
volumes:
  - name: shared-workspace
    persistentVolumeClaim:
      claimName: shared-workspace
```

In the container of freebayes in the same file, define the mount point to be used for output, where "/workspace/" is an arbitrary path name as a mount point. It does not have to exist in your container:

```
volumeMounts:
  - name: shared-workspace
    mountPath: "/workspace/"
```

Do not apply the *Deployment* for now. Let's get all the persistent volumes created first. Otherwise, the deployment will be waiting for the PVs and PVCs and may fail due to the timeout.

8.30.3 Exercise 2: ReadOnlyMany for data source

Here is a sample of PV https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/pv-1000g.yml:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1000g
spec:
  capacity:
    storage: 100Ti
  accessModes:
    - ReadOnlyMany
  nfs:
    server: "<host name or IP>"
    path: "<mount path>"
```

Here is a sample of PVC https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/pvc-1000g.yml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv1000g
spec:
  storageClassName: ""
  accessModes:
    - ReadOnlyMany
  resources:
    requests:
      storage: 100Ti
```

The read-only volume bound in the beginning can be mounted by containers in a pod. There are two steps.

Apply the *PersistentVolume* and *PersistentVolumeClaim* get gain access to the data source as *ReadOnlyMany* or *ROX*:

```
ubuntu@resops-k8s-node-nf-2:~$ kubectl apply -f ~/adv-k8s/osk/dpm/pv-1000g.yml
persistentvolume/pv1000g created

ubuntu@resops-k8s-node-nf-2:~$ kubectl apply -f ~/adv-k8s/osk/dpm/pvc-1000g.yml
persistentvolumeclaim/pv1000g created
```

(continues on next page)

(continued from previous page)

```
ubuntu@resops-k8s-node-nf-2:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY␣
↪STATUS   CLAIM                      STORAGECLASS   REASON   AGE
pv1000g                                    100Ti      ROX            Retain␣
↪Bound    default/pv1000g                                  37s
pvc-5ae9a98b-4669-47fb-8a5b-8e5b95a74936   50Gi       RWX            Delete␣
↪Bound    default/shared-workspace   nfs-client              72m

ubuntu@resops-k8s-node-nf-2:~$ kubectl get pvc
NAME               STATUS   VOLUME                                     CAPACITY␣
↪ACCESS MODES   STORAGECLASS   AGE
pv1000g            Bound    pv1000g                                    100Ti      ROX␣
↪                           28s
shared-workspace   Bound    pvc-5ae9a98b-4669-47fb-8a5b-8e5b95a74936   50Gi       RWX␣
↪           nfs-client     72m
```

In the pod template, refer to the PersistentVolumeClaim pv1000g:

```
volumes:
  - name: pv1000g
    persistentVolumeClaim:
      claimName: pv1000g
```

In the containers of samtools and freebayes, defines the logical mount point that everything running in them would see:

```
volumeMounts:
  - name: pv1000g
    mountPath: "/datasource/"
```

The Samtools and Freebayes, running in their containers can access the human reference genome and assemblies from the 1000 Genome Project as if local files. Again, do not apply the *Deployment* in the pod template in https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/freebayes.yml yet. We will do that in the future.

You may have noticed that volume and volumeMount related to pv1000g are commented out in the pod template. This is because the actual mount would fail due to the network differences between the environment for our project and the Minikube.

8.30.4 Exercise 3: ReadWriteOnce for private workspace

All clouds provide cloud-specific volumes for ReadWriteOnce. Check the API reference for details how to use them. The syntax in Kubernetes manifest is the same as above, except for the access mode of "ReadWriteOnce".

In many cases, ReadWriteOnce volumes are intended as a temporary directory that shares a pod's lifetime. It is handier to use *emptyDir* instead of ReadWriteOnce storage volume. If *Memory* is used as the medium, IO can be much faster given additional memory consumption.

In the pod template in https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/freebayes.yml, uncomment the volumes below:

```
volumes:
  - name: private-samtools
    emptyDir:
      medium: ""
  - name: private-freebayes
```

(continues on next page)

(continued from previous page)

```
    emptyDir:
      medium: Memory
```

In the container of samtools and freebayes in the same file, uncomment the mount points to be used for temporary output, where "/private-samtools/" and "/private-freebayes/" are arbitrary path names as mount points. They not have to exist in your containers:

```
volumeMounts:
  - name: private-samtools
    mountPath: "/private-samtools/"

volumeMounts:
  - name: private-freebayes
    mountPath: "/private-freebayes/"
```

Apply the *Deployment* in the pod template in https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/freebayes.yml, which may take a while:

```
ubuntu@resops-k8s-node-nf-2:~$ kubectl apply -f ~/adv-k8s/osk/dpm/freebayes.yml
deployment.apps/freebayes-dpm created

ubuntu@resops-k8s-node-nf-2:~$ kubectl rollout status deployment.v1.apps/freebayes-
↪dpm --request-timeout=60m
Waiting for deployment "freebayes-dpm" rollout to finish: 0 of 3 updated replicas are␣
↪available...
Waiting for deployment "freebayes-dpm" rollout to finish: 1 of 3 updated replicas are␣
↪available...
Waiting for deployment "freebayes-dpm" rollout to finish: 2 of 3 updated replicas are␣
↪available...
deployment "freebayes-dpm" successfully rolled out
```

Note that emptyDir is not a persistent volume. It uses the local storage of memory where a pod is running on. Thus, *kubectl get pv* or *kubectl get pvc* does not know if and how emptyDir is mounted. You would need to connect to the pods to see the mounted volume, for example "/private-freebayes/":

```
C02XD1G9JGH7:adv-k8s davidyuan$ kubectl get pod
NAME                                                READY     STATUS    ␣
↪RESTARTS   AGE
freebayes-dpm-c69456659-c9x9d                       2/2       Running   0         ␣
↪    2m38s
freebayes-dpm-c69456659-lmmcj                       2/2       Running   0         ␣
↪    2m39s
freebayes-dpm-c69456659-xj2qh                       2/2       Running   0         ␣
↪    2m38s
listening-skunk-nfs-client-provisioner-79fb65dd79-86qgq   1/1       Running   3         ␣
↪    65d
minio-freebayes-8dd7db8f4-5jvbd                     1/1       Running   0         ␣
↪    2m39s
nfs-in-a-pod                                        1/1       Running   7         ␣
↪    40d

C02XD1G9JGH7:adv-k8s davidyuan$ kubectl exec -it freebayes-dpm-c69456659-c9x9d -c␣
↪freebayes -- bash

root@freebayes-dpm-c69456659-c9x9d:/# ls -l /
total 75
```

(continues on next page)

(continued from previous page)

```
drwxr-xr-x   1 root root 4096 Jul 10 02:26 bin
drwxr-xr-x   2 root root 4096 Jun 14  2018 boot
drwxrwxrwx   5 1000 1000   86 May 31 16:00 datasource
drwxr-xr-x   5 root root  380 Jul 15 10:41 dev
drwxr-xr-x   1 root root 4096 Jul 15 10:41 etc
drwxr-xr-x  15 root root 4096 Jul 12 15:50 freebayes
drwxr-xr-x   2 root root 4096 Jun 14  2018 home
drwxr-xr-x   1 root root 4096 Jul 10 02:27 lib
drwxr-xr-x   2 root root 4096 Jul  8 03:30 lib64
drwxr-xr-x   2 root root 4096 Jul  8 03:30 media
drwxr-xr-x   2 root root 4096 Jul  8 03:30 mnt
drwxr-xr-x   2 root root 4096 Jul  8 03:30 opt
drwxrwxrwx   2 root root 4096 Jul 15 10:40 private-freebayes
dr-xr-xr-x 209 root root    0 Jul 15 10:41 proc
drwx------   2 root root 4096 Jul  8 03:30 root
drwxr-xr-x   1 root root 4096 Jul 15 10:41 run
drwxr-xr-x   2 root root 4096 Jul  8 03:30 sbin
drwxr-xr-x   2 root root 4096 Jul  8 03:30 srv
dr-xr-xr-x  13 root root    0 Jul 15 10:46 sys
drwxrwxrwt   1 root root 4096 Jul 12 15:53 tmp
drwxr-xr-x   1 root root 4096 Jul  8 03:30 usr
drwxr-xr-x   1 root root 4096 Jul  8 03:30 var
drwxrwxrwx   4 root root   38 Jul 15 10:41 workspace
```

Exit out of the container.

8.30.5 Exercise 4: Initialising persistent volumes

There is no life-cycle management for persistent volumes in Kubernetes. The closest thing is command array in initContainers. Here is an example to create a subdirectory on a mounted volume */workspace/*, in the pod template in https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/freebayes.yml:

```
initContainers:
  - name: init
    image: busybox
    command: ["/bin/sh"]
    args: ["-c", "mkdir -p /workspace/result/"]
    volumeMounts:
      - name: shared-workspace
        mountPath: "/workspace/"
```

Be careful when using multiple initContainers instead of one to configure the pod. The behaviour can be puzzling.

Connect to a pod to see a subdirectory is created on the mounted volume:

```
ubuntu@resops-k8s-node-nf-2:~$ kubectl exec -it freebayes-dpm-7ff686fdcf-b7w95 -c␣
↪freebayes -- bash

root@freebayes-dpm-7ff686fdcf-b7w95:/# ls -l /workspace/
total 4
drwxr-sr-x 2 nobody 4294967294 4096 Jul 16 14:17 result

root@freebayes-dpm-7ff686fdcf-b7w95:/# ls -l /workspace/result/
total 0
```

Exit out of the container.

8.30.6 Exercise 5: Kubernetes secret & S3 interface

To integrate Freebayes with other pipelines requiring S3 bucket, or to review output easily via a browser, Minio can be mounted to the storage for shared output. The manifest in the pod template and the container as the same as in *Exercise 1: ReadWriteMany for shared output*.

Kubernetes can store secrets and use them in manifest referring to them. Check the online help *kubectl create secret –help* for more details. We need access key and secret key for Minio deployment:

```
kubectl create secret generic minio --from-literal=accesskey=YOUR_ACCESS_KEY --from-
↪literal=secretkey=YOUR_SECRET_KEY
secret/minio created

resops49@resops-k8s-node-17:~/adv-k8s/osk$ kubectl get secret
NAME                                  TYPE                                  DATA␣
↪AGE
default-token-4wmbq                   kubernetes.io/service-account-token   3␣
↪7d18h
minio                                 Opaque                                2␣
↪52s
nfs-nfs-server-provisioner-token-7mk47   kubernetes.io/service-account-token   3␣
↪6m41s
```

The Minio container will get the access key and secret key securely from the environment when it is initialised. See container arguments in https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/minio.yml:

```
env:
  # MinIO access key and secret key
  - name: MINIO_ACCESS_KEY
    valueFrom:
      secretKeyRef:
        name: minio
        key: accesskey
  - name: MINIO_SECRET_KEY
    valueFrom:
      secretKeyRef:
        name: minio
        key: secretkey
```

Make sure that the arguments to initialize the container must refer to the same mount point */workspace/* as in the previous exercise. Then, the subdirectories will be treated as S3 buckets by Minio:

```
containers:
  - name: minio
    image: minio/minio
    args:
      - server
      - /workspace/
```

Apply the *Deployment* for Minio to turn the shared persistent volume in *ReadWriteMany* mode into a S3 storage:

```
ubuntu@resops-k8s-node-nf-2:~$ kubectl apply -f ~/adv-k8s/osk/dpm/minio.yml
deployment.apps/minio-freebayes created
service/minio-freebayes created

ubuntu@resops-k8s-node-nf-2:~$ kubectl get svc
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)␣
↪                             AGE
```

(continues on next page)

(continued from previous page)

```
kubernetes                    ClusterIP   10.96.0.1        <none>        443/TCP
↪                             5h35m
minio-freebayes               NodePort    10.100.108.50    <none>        9001:30037/
↪TCP                             53s
nfs-nfs-server-provisioner    ClusterIP   10.109.187.100   <none>        2049/TCP,
↪20048/TCP,51413/TCP,51413/UDP   3h43m

ubuntu@resops-k8s-node-nf-2:~$ kubectl get deployment
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
freebayes-dpm      3/3     3            3           28m
minio-freebayes    1/1     1            1           4m42s
```

If you have had VNC enabled in Exercise 0.1: (Optional) Enabling GUI for VNC, you should be able to access MinIO via Firefox at http://10.100.108.50:9001. Note that the IP address is different in every deployment. You should be able to log on with YOUR_ACCESS_KEY and YOUR_SECRET_KEY defined earlier in this exercise.

8.30.7 Exercise 6: Horizontal scaling

Kubernetes is an orchestration engine. It is understandable that it provides limited capability for workflow management. Kubernetes and some simple shell-scripting can scale pods and schedule jobs to be run in parallel:

```
kubectl get pod
dpmname=$(kubectl get deployment -o name | grep -m 1 freebayes | cut -d '/' -f2)
kubectl scale deployment ${dpmname} --replicas=4
kubectl get pod
```

If there were three pods before the scaling, you should see one more pod initialized to reach the total number of replicas to 4:

```
NAME                                                READY     STATUS
↪RESTARTS   AGE
freebayes-dpm-c69456659-c9x9d                       2/2       Running     0
↪     167m
freebayes-dpm-c69456659-fckgx                       0/2       Init:0/1    0
↪     3s
freebayes-dpm-c69456659-lmmcj                       2/2       Running     0
↪     167m
freebayes-dpm-c69456659-xj2qh                       2/2       Running     0
↪     167m
listening-skunk-nfs-client-provisioner-79fb65dd79-86qgq   1/1       Running     3
↪     66d
minio-freebayes-8dd7db8f4-5jvbd                     1/1       Running     0
↪     167m
nfs-in-a-pod                                        1/1       Running     7
↪     40d
```

If you run *kubectl get pod* in about 1 minute, you should see the new pod is running and ready.

In Bash script, you can then send jobs into each container in round-robin. This is the most simple-minded job scheduling. This shotgun approach can overwhelm pods easily. Jobs scheduled can fail due to lack of resources or timeouts. It is work-in-progress to integrate Kubenetes cluster with a workflow engine.

8.30.8 Reading 1: Vertical scaling

Kubernetes performs vertical scaling automatically to allocate addintional CPU, memory to a pod as needed. You can set minimum and maximum limits to resources as shown in https://gitlab.ebi.ac.uk/davidyuan/adv-k8s/blob/master/osk/dpm/freebayes.yml:

```
resources:
  requests:
    cpu: 1
    memory: 2Gi
  limits:
    cpu: 4
    memory: 8Gi
```

If there are many pods to manage, allocating resources in manifest can go out of control quickly. You may want to leave the resource allocation and pod scheduling to Kubernetes in most cases.

There is a limitation in Minikube. You would not be able to run *kubectl top* due to missing heapster service. Otherwise, you would see something like the following:

```
C02XD1G9JGH7:~ davidyuan$ kubectl top pod freebayes-dpm-7c87bcf4c6-rkfz2 --containers
POD                              NAME         CPU(cores)   MEMORY(bytes)
freebayes-dpm-7c87bcf4c6-rkfz2   samtools     0m           0Mi
freebayes-dpm-7c87bcf4c6-rkfz2   freebayes    0m           2Mi
```

8.30.9 Exercise 7: Autoscaling

Kubernetes has certain capability of autoscaling. The following script creates horizontal pod autoscaler to manage autoscaling policies:

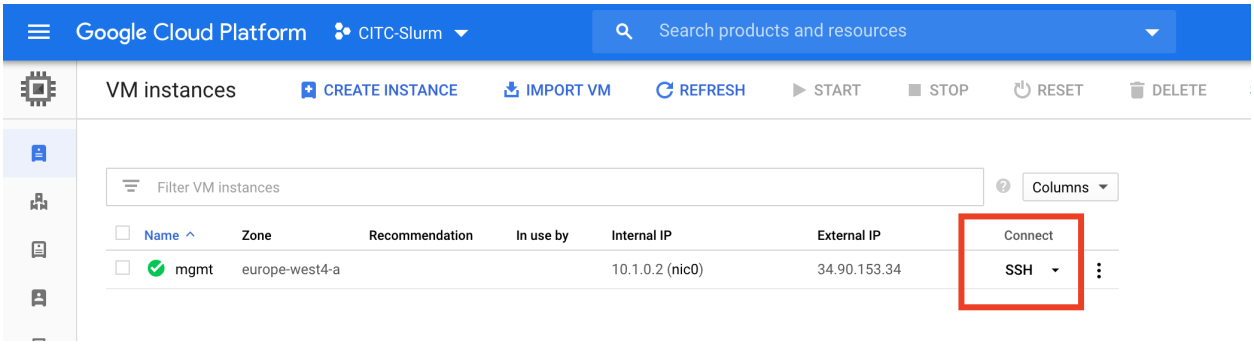
```
max_pods=5
dpmname=$(kubectl get deployment -o name | grep -m 1 freebayes | cut -d '/' -f2)
kubectl autoscale deployment ${dpmname} --cpu-percent=50 --min=1 --max=${max_pods}
kubectl get hpa

NAME             REFERENCE                   TARGETS          MINPODS   MAXPODS
↪REPLICAS   AGE
freebayes-dpm    Deployment/freebayes-dpm    <unknown>/50%    1         5         0
↪     4s
```

8.31 Accessing GCP node from CLI

The following focuses on accessing a GCP node without public IP. If a public IP is available, simply drop the optional flag *–tunnel-through-iap* from a command.

8.31.1 SSH or SCP to a node without public IP

With GCP Identity-Aware Proxy (IAP), you can SSH to any node without public IP with your G Suite ID at EBI. There are many ways to access nodes. They require little or no extra effort.



SSH via a web browser window

Pros: Simplest option. No installation or configuration required.

Cons: Only a dumb terminal in a web browser.

Click an SSH button for the node in GCP Cloud Console. This pops up a browser window emulating an SSH terminal.

SSH via Cloud Shell

Pros: Fully functioning VM for development to keep your laptop clean.

Cons: One time configuration required. Files can be copied between your laptop / EBI cluster and Cloud Shell via GUI or SCP.

Click a drop-down button next to SSH in GCP Cloud Console. Click "View gcloud command". Click "RUN IN CLOUD SHELL". This starts your private Cloud Shell VM for you to SSH to a node. You may be asked to generate keys for the first time. This option keeps your laptop clean.

SSH from your laptop

Pros: Direct connection from your laptop. Convenient if it is your default development environment already.

Cons: Installation of GCP SDK required. One time configuration and authentication required.

Start a terminal window. Copy and paste the gcloud command (e.g. *gcloud compute ssh –zone $ZONE $LOGIN_HOST –tunnel-through-iap –project $PROJECT*) into it. You may have to install Cloud SDK first by following the instructions on Installing Google Cloud SDK. You may also need to authenticate with your G Suite ID by following instructions in the terminal window.

SSH from EBI cluster

Pros: Direct connection from EBI cluster. Convenient if you need to transfer files between GCP and EBI cluster.

Cons: Most complicated installation, configuration and authentication. Ancient Python version in EBI cluster making life harder.

Log into EBI cluster (e.g. *ssh ebi-cli* or *ssh ebi-login*) from a terminal window. Run the following commands to install Miniconda3 as instructed by https://docs.conda.io/en/latest/miniconda.html. Answer *yes* to all the questions:

```
[davidyuan@noah-login-03 ~]$ cd "${HOME}" && curl -O https://repo.anaconda.com/
↪miniconda/Miniconda3-latest-Linux-x86_64.sh
[davidyuan@noah-login-03 ~]$ chmod +x ./Miniconda3-latest-Linux-x86_64.sh
[davidyuan@noah-login-03 ~]$ ./Miniconda3-latest-Linux-x86_64.sh
[davidyuan@noah-login-03 ~]$ . "${HOME}"/.bashrc
```

This should create a base conda environment with Python 3.8 installed. You can confirm that with the following commands:

```
(base) [davidyuan@noah-login-03 ~]$ python --version
Python 3.8.3
(base) [davidyuan@noah-login-03 ~]$ which python
~/miniconda3/bin/python
```

Now you can install Google Cloud SDK with Miniconda3 as instructed by https://anaconda.org/conda-forge/google-cloud-sdk. Again, answer *yes* when asked:

```
(base) [davidyuan@noah-login-03 ~]$ conda install -c conda-forge google-cloud-sdk
```

It is always a good idea to double-check what you have done:

```
(base) [davidyuan@noah-login-03 ~]$ which gcloud
~/miniconda3/bin/gcloud
(base) [davidyuan@noah-login-03 ~]$ gcloud --version
Google Cloud SDK 310.0.0
```

You can SSH from EBI cluster to any node on GCP (e.g. *gcloud compute ssh –zone $ZONE $LOGIN_HOST –tunnel-through-iap –project $PROJECT*).

Notes:

1. You can use IAP with SCP in a similar fashion (e.g. *gcloud compute scp –zone $ZONE –tunnel-through-iap –project $PROJECT <normal_scp_parameters>*). It can be handy to push or pull files between EBI cluster and GCP nodes via SCP.
2. You can also use gsutil to upload files to or to download objects from the storage buckets (e.g. *gsutil ls gs://<bucket_name>*).

8.32 Arvados on Kubernetes

8.32.1 Arvados installation options

Arvados on Kubernetes is under development. Here is a link to the installation options.

The official documentation on how to install Arvados on Kubernetes can be found here.

8.32.2 Kubenetes cluster

There are three options to host a Kubernete cluster:

1. CaaS at EBI. The Helm chart requires the namespace of *kube-system*, which TSI has no access to.
2. GKE on GCP. This is a good option for production. This requires funding.
3. Kubernetes CPA on ECP backed by OSK.

It is also possible to run it on Kubernetes clusters in AWS or MSA. The best option is GKE. The vendor only tested their Helm chart in GKE.

8.32.3 Kubernetes cluster on GKE

Run the following command in Cloud Shell:

```
gcloud beta container --project "extreme-lore-114513" clusters create "hdr-uk-1" --
↪region "europe-north1" --no-enable-basic-auth --cluster-version "1.11.6-gke.2" --
↪machine-type "n1-standard-2" --image-type "COS" --disk-type "pd-standard" --disk-
↪size "100" --scopes "https://www.googleapis.com/auth/cloud-platform" --num-nodes "3
↪" --enable-cloud-logging --enable-cloud-monitoring --no-enable-ip-alias --network
↪"projects/extreme-lore-114513/global/networks/default" --subnetwork "projects/
↪extreme-lore-114513/regions/europe-north1/subnetworks/default" --enable-autoscaling␣
↪--min-nodes "3" --max-nodes "20" --addons HorizontalPodAutoscaling,
↪HttpLoadBalancing --enable-autoupgrade --enable-autorepair

davidyuan@cloudshell:~ (extreme-lore-114513)$ gcloud beta container --project
↪"extreme-lore-114513" clusters create "hdr-uk-1" --region "europe-north1" --no-
↪enable-basic-auth --cluster-version "1.11.6-gke.2" --machine-type "n1-standard-2" --
↪image-type "COS" --disk-type "pd-standard" --disk-size "100" --scopes "https://www.
↪googleapis.com/auth/cloud-platform" --num-nodes "3" --enable-cloud-logging --enable-
↪cloud-monitoring --no-enable-ip-alias --network "projects/extreme-lore-114513/
↪global/networks/default" --subnetwork "projects/extreme-lore-114513/regions/europe-
↪north1/subnetworks/default" --enable-autoscaling --min-nodes "3" --max-nodes "20" --
↪addons HorizontalPodAutoscaling,HttpLoadBalancing --enable-autoupgrade --enable-
↪autorepair
WARNING: Starting in 1.12, new clusters will not have a client certificate issued.␣
↪You can manually enable (or disable) the issuance of the client certificate using␣
↪the `--[no-]issue-client-certificate` flag.
WARNING: Starting in 1.12, default node pools in new clusters will have their legacy␣
↪Compute Engine instance metadata endpoints disabled by default. To create a cluster␣
↪with legacy instance metadata endpoints disabled in the default node pool, run␣
↪`clusters create` with the flag `--metadata disable-legacy-endpoints=true`.
This will enable the autorepair feature for nodes. Please see https://cloud.google.
↪com/kubernetes-engine/docs/node-auto-repair for more information on node␣
↪autorepairs.
This will enable the autoupgrade feature for nodes. Please see https://cloud.google.
↪com/kubernetes-engine/docs/node-management for more information on node␣
↪autoupgrades.
Creating cluster hdr-uk-1 in europe-north1... Cluster is being health-checked (master␣
↪is healthy)...done.
Created [https://container.googleapis.com/v1beta1/projects/extreme-lore-114513/zones/
↪europe-north1/clusters/hdr-uk-1].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/
↪kubernetes/workload_/gcloud/europe-north1/hdr-uk-1?project=extreme-lore-114513
kubeconfig entry generated for hdr-uk-1.
NAME      LOCATION       MASTER_VERSION  MASTER_IP       MACHINE_TYPE   NODE_VERSION ␣
↪NUM_NODES  STATUS
hdr-uk-1  europe-north1  1.11.6-gke.2    35.228.159.196  n1-standard-2  1.11.6-gke.2 ␣
↪9          RUNNING
```

Configure kubectl command-line access:

```
gcloud beta container clusters get-credentials hdr-uk-1 --region europe-north1 --
↪project extreme-lore-114513
```

Note that the default namespace is not used. Pods are under kube-system namespace.

8.32.4 Helm installation

Install Helm and Tiller in the namespace of kube-system on Kubernetes master:

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get | bash
helm init --upgrade
kubectl create serviceaccount --namespace kube-system tiller
kubectl create clusterrolebinding tiller-cluster-rule --clusterrole=cluster-admin --
↪serviceaccount=kube-system:tiller
kubectl patch deploy --namespace kube-system tiller-deploy -p '{"spec":{"template":{
↪"spec":{"serviceAccount":"tiller"}}}}'
```

8.32.5 Clone Git repository

Install Git client before cloning the repository:

```
sudo yum install git
git clone https://github.com/curoverse/arvados-kubernetes.git
cd arvados-kubernetes/charts/arvados
```

8.32.6 Start a cluster

Reserve a static IP at VPC network. Here are the detailed instructions:

```
./cert-gen.sh 35.228.126.19
Generating a RSA private key
..................+++++
...................+++++
writing new private key to './config/ssl/key'
-----
```

Update values.yml as needed, for example:

```
adminUserEmail
adminUserPassword
superUserSecret
anonymousUserSecret
nodes
```

Start the Arvados cluster:

```
helm install --name arvados . --set externalIP=35.228.126.19
```

Check the status with *kubectl*:

```
kubectl get pods
kubectl get svc
```

Test the web access at:

```
https://35.228.126.19
```

8.32.7 Final note

There seems significant mismatch what is deployed by the Helm chart and what features are available via GUI (e.g. https://35.228.126.19) according to the documentation. Perhaps, Some work is needed considering the chart was not really updated alone with the runtime.

8.33 Cloud Consulting Team toolbox

The instructions below are either for Mac OS X or RedHat / CentOS. You can easily figure out instructions for other OS.

8.33.1 Git client

Install Homebrew with Ruby, then install Git with Homebrew.

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
↪install)"
brew doctor
brew install git
```

Reference

Install Git

8.33.2 SFTP client

1. Download and install FileZilla client for Mac at https://filezilla-project.org
2. Define a new site under Site Manager.
3. Once connected, accept the site public key to make it become a known host.

8.33.3 SSH client

Termius is available in Mac App Store. Once installed, perform some configuration to make it really handy.

Adding key

1. Select Preferences > Keychain
2. Click "+ ADD KEY"
3. Copy and paste a private key

Adding identity

1. Select Preferences > Keychain
2. Click "+ ADD IDENTITY"
3. Type user ID and select a key in the keychain

New group

1. Select Hosts
2. Click "+ NEW GROUP"
3. Select optional values such as Username or Key to be shared by hosts in the group. Groups can also be nested.

New host

1. Select Hosts
2. Click "+ NEW HOST"
3. Type in address, select ID, key or group as desired.

8.33.4 IntelliJ IDEA

1. Download IntelliJ IDEA Community Edition.
2. Open the disk image with the default DiskImageMounter. Drag and drop to Application.
3. Disable plugins not in use. Keep plugins such as Git enabled. Install plugins such as Terraform, Ansible, Bash, MarkDown, etc. as needed.
4. Configure and check out from version control right on the welcome window.
5. Import settings from previous version if upgrade.

Note that it is possible to use other IDEs such as Eclipse, Android Studio, Visual Studio etc.. Each IDE is designed to please a particular sub-population. Choose one most handy for your coding needs. IntelliJ seems fit well with cloud implementation.

8.33.5 Terraform client

1. There is no installation required. The binary can simply be downloaded from their site. The binary can be run directly.
2. Make sure that the binary is executable `chmod +x terraform`.
3. Move it to a right location `mv ./terraform /usr/local/bin/terraform`.

8.33.6 Ansible

The recommended way to install Ansible on Mac OS X is to use pip 9.0.3 or later.

```
sudo -H pip install ansible
```

8.33.7 Linux screen sharing - TMUX

Many times it comes with default, In case it not available than

```
sudo yum install tmux|sudo apt-get install tmux
```

Some tips/tricks with cheatsheet

8.33.8 Yasha

Yasha is a code generator based on Jinja2 template engine. This is required to generate YAML files from Jinja2 template for CPA.

```
sudo pip install yasha
```

8.33.9 Missing xcrun after upgrading to Mojave

After upgrading to Mac OS Mojave, `make html` fails with `xcrun: error: invalid active developer path (/Library/Developer/CommandLineTools), missing xcrun at: /Library/Developer/CommandLineTools/usr/bin/xcrun`.

Update Xcode Command line Tools to fix it.

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ sudo xcode-select --install
Password:
xcode-select: note: install requested for command line developer tools
```

8.33.10 Shell utility for json

Almost all modern languages provide libraries for json data. Sometime, it is handy to deal with it in a Unix shell directly. The `jq` command can be installed on Mac OS.

```
brew install jq
```

8.33.11 Virtual environment

It is generally a good habit and sometimes useful to use virtual environment for each project.

```
pip3 install virtualenv
cd my-project/
virtualenv venv
source venv/bin/activate
```

Note that `virtualenv venv --system-site-packages` inherits globally installed packages.

8.33.12 Find command for command line search

```
find . -type f -print | xargs grep "key_words"
```

8.33.13 Slack and GoogleDocs

Copy and paste a link to a document on GoogleDocs. Answer yes to OAuth questions. Note that you do not have to repeat this in additional workspaces.

8.34 DevOps toolchain from GitLab to Docker Hub for Container Build

The default builder on Docker Hub has short timeout when building an image. This can be a problem when an image is large or network is slow. Build can be moved to a GitLab runner to solve the problem. As the CI/CD pipeline is relative simple, there is no need for a dashboard or library. The toolchain for Docker build pipelines consists of the following tools:

1. IntelliJ IDEA CE
2. GitLab
3. GitHub
4. Docker Hub

The integration of tools are the same as for documentation and ECP above. The overall workflow is show as the diagram below:

The toolchain can pull Dockerfile from a GitHub repository. It builds the Docker image on a GitLab runner. It pushes the image to Docker Hub when a build is done. This is useful when an image is too big for Docker Hub to build it.

The easiest way to integrate with an external repository for CI/CD is to click *CI/CD for external repo* when creating a new project. Alternatively, mirroring repositories at *Settings* > *Repository*.

8.34.1 Building Docker image

There is only one file *.gitlab-ci.yml* needed to create a toolchain to build a docker image with GitLab at EBI, instead of DockerHub. It runs the build engine on the GitLab runners for EBI. It is a production-grade Kubernetes cluster managed by the cloud team. There is no extra configuration needed.

The *.gitlab-ci.yml* file can be edited with online editor in GitLab or any text editor. Developers typically use an IDE such as IntelliJ integrated with GitLab. The file contains the following major sections.

Add official Docker image to a runner with a Kubernetes cluster and require docker-in-docker service:

```
# Official docker image.
image: docker:latest

services:
  - docker:dind
```

Log into Docker Hub and install git. Leave CI_REGISTRY blank for Docker Hub:

```
before_script:
  - docker login -u "${CI_REGISTRY_USER}" -p "${CI_REGISTRY_PASSWORD}" ${CI_REGISTRY}
  - apk add git
  - mkdir -p ${ARTIFACT_DIR}
```

Clone the repository for Docker build onto the runner. Build, tag and push a new Docker image:

```
docker.image:
  stage: build
  variables:
    CI_GIT_REPO_NAME: Metagenomics-Assembly
    CI_GIT_URL: https://github.com/Medalibi/${CI_GIT_REPO_NAME}.git
    CI_REGISTRY_IMAGE: davidyuyuan/metagenomics
    CI_COMMIT_REF_SLUG: assembly
    CI_SOURCE_IMAGE: metagenomics
    CI_DOCKER_FILE: ${CI_PROJECT_DIR}/${CI_GIT_REPO_NAME}/Dockerfile
  script:
    - git clone ${CI_GIT_URL}
    - cd ${CI_GIT_REPO_NAME}
    - docker build -f ${CI_DOCKER_FILE} -t ${CI_SOURCE_IMAGE} . | tee ${ARTIFACT_DIR}/
↪build.log
    - docker tag ${CI_SOURCE_IMAGE} ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}
    - docker push ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG} | tee ${ARTIFACT_DIR}/
↪push.log
  only:
    - master
  artifacts:
    name: "$CI_COMMIT_REF_NAME-$CI_JOB_STAGE-$CI_JOB_NAME"
    paths:
      - ${ARTIFACT_DIR}
```

It is a good practise to tee build log and push log onto the *ARTIFACT_DIR*. They can then be accessed and reviewed under the pipeline or job generating thme.

8.34.2 Appendix: Command line instructions

Git global setup:

```
git config --global user.name "User Name"
git config --global user.email "username@ebi.ac.uk"
```

Create a new repository:

```
git clone git@gitlab.ebi.ac.uk:davidyuan/adv-k8s.git
cd adv-k8s
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Existing folder:

```
cd existing_folder
git init
git remote add origin git@gitlab.ebi.ac.uk:davidyuan/adv-k8s.git
git add .
```

(continues on next page)

(continued from previous page)

```
git commit -m "Initial commit"
git push -u origin master
```

Existing Git repository:

```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@gitlab.ebi.ac.uk:davidyuan/adv-k8s.git
git push -u origin --all
git push -u origin --tags
```

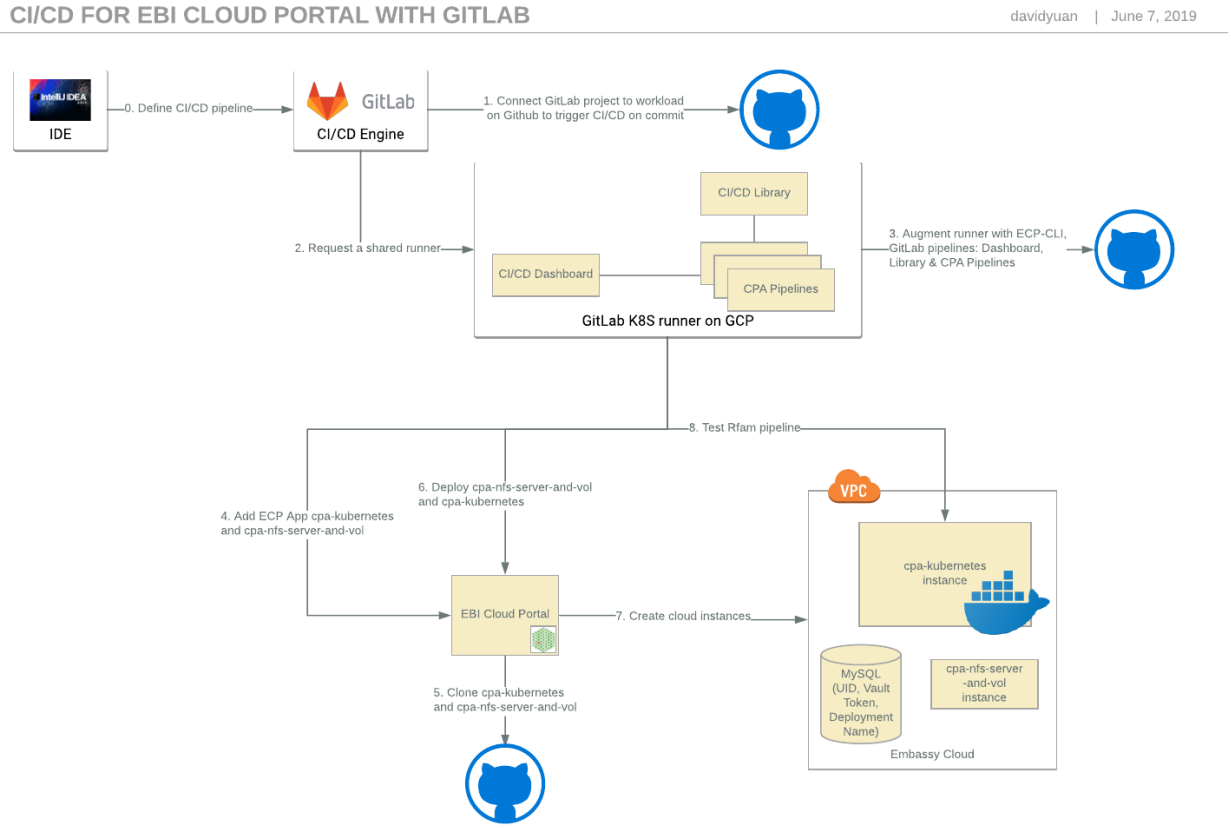
8.35 DevOps toolchain from Gitlab to OpenStack for pipelines on ECP

The toolchain for DevOps pipelines consists of the following tools:

1. IntelliJ IDEA CE
2. GitLab
3. GitHub
4. Docker Hub
5. EMBL-EBI Cloud Portal
6. A cloud provider (e.g. openstack)

All ECP applications are stored in Github. The projects are named as cpa_*. The commandline interface is ecp_cli.

The overall workflow is show as the diagram below:



The complex logic to interact with ECP with its CLI and to interact with a cloud provider such as OSK, GCP, AWS or Azure via ECP is coded in the CI/CD Library. To create a new pipeline invoked on CI/CD Dashboard, perform the following tasks:

8.35.1 Adding test case onto the Dashboard

Create a job to be managed by the Dashboard in *.gitlab0ci.yml*. The token value and pipeline URL are unique to each project, for example:

```
cpa-instance:
stage: test
script:
  - curl -X POST -F token=81ee2ce150624134ec0462845f4dda -F ref=master -F␣
↪variables[os_username]="${os_username}" -F variables[os_password]="${os_password}" -
↪F variables[ecp_user]="${ecp_user}" -F variables[ecp_password]="${ecp_password}" -F␣
↪variables[app_account_name]="${app_account_name}" -F variables[config_account_name]=
↪"${config_account_name}" https://gitlab.ebi.ac.uk/api/v4/projects/731/trigger/
↪pipeline
when: manual
```

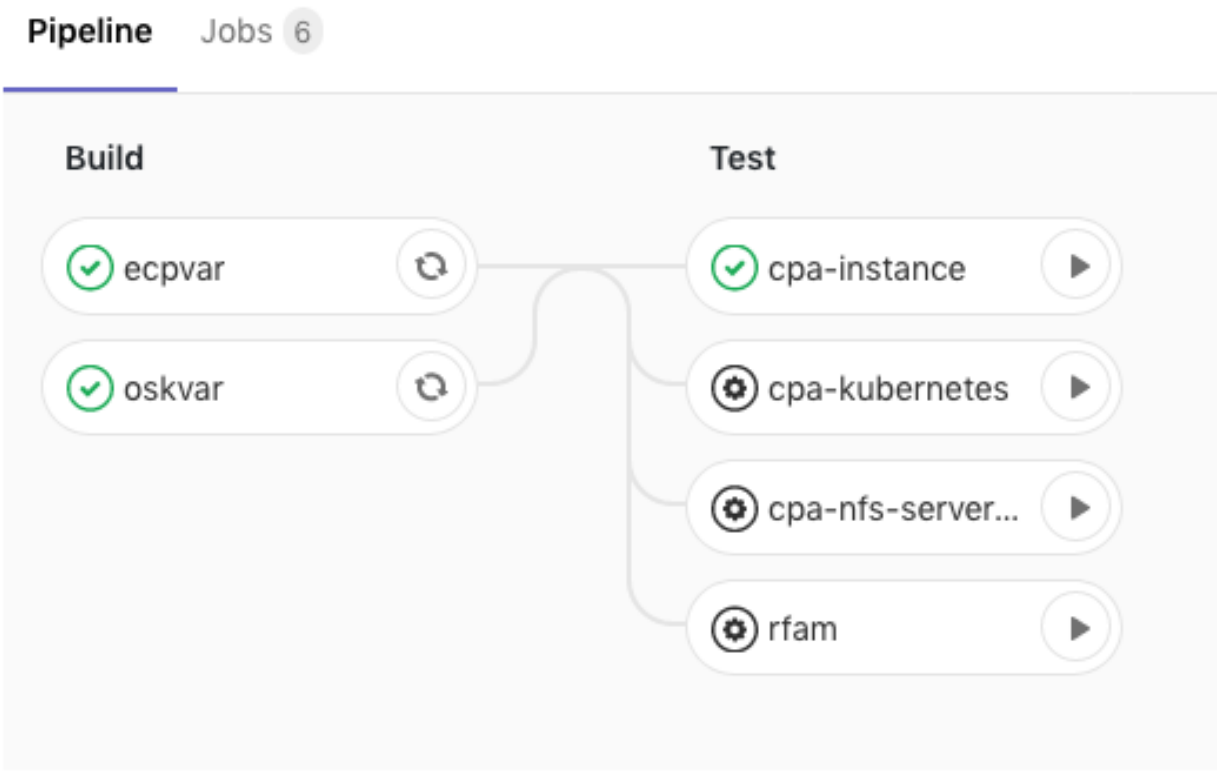
8.35.2 Creating pipeline

There can be at most one pipeline for each project in GitLab. A CI/CD pipeline can be triggered by an external events such as a commit on Github or an update on Docker Hub, Cron job. It can also be triggered manually.

- To create a new cluster on GKE with a Google account on GitLab.com, click *Add Kubernetes cluster*. This provides a dedicated running in GCP. Otherwise, a shared runner is used by default.
- To trigger CI/CD by a commit on Github, create a new CI/CD project with an option of *CI/CD for external repo*. Click *git Repo by URL*.
- To trigger CI/CD manually, create a black project and click *Set up CI/CD* to start editing *.gitlab-ci.yml*.

Expand Pipeline triggers. Add a brief description and click *Add trigger*. The token and the RESTful API can be used in three different ways:

1. Third party utility such as cURL to trigger the pipeline by calling the RESTful API.
2. Pipeline in another project to trigger the pipeline by calling the RESTful API. This is how the Dashboard above works.
3. Push or tag push events in another project to trigger the pipeline by defining a webhook in that project. This can be very handy for changes in multiple projects to trigger the same pipeline so that the integration of them are tested. The owner of these projects would have to agree to add the webhooks to notifying push or tag push events. Also, these projects need to be public or protected.



8.35.3 Implementing ECP test cases

Include the test framework for ECP CI/CD:

```
include:
  - 'https://gitlab.ebi.ac.uk/davidyuan/cicd-lib/raw/master/lib/ecp-cicd.yaml'
```

Define variables at the pipeline scope:

```
variables:
  app_name: 'Generic server instance'
  ssh_key: ssh-rsa␣
↪AAAAB3NzaC1yc2EAAAADAQABAAABAQDbVJdgtiojWuTmIpa6GzIVeaapJKwMb3zG7Y8iWgkhFRziLK+SfRoeg4VigFOMhpDRuM
↪5Ui61XJ/mDGNoKFX8Zr4CZ8f+e5nzZWY/
↪w58p5s2g2cbJcpJV249qKmlnNKQJi+qONAoIczQN4Hc7J4rqlxXwv+lH9uRZE15+O6Ughp9SZp3EY+ZhuGw1lrnXz933OygL3q
↪/xfA/
↪WVAS5pnKIYRtaSwVzMZTzmhT70DymfXsNWX72d+N8BDlt4wOLE9EgUXog7z8akdNoXwkcF5qYDRhsjEc5KIgScBN8l6O/
↪9uB4v
```

Note that app_name and ssh_key are required by ECP. In addition, id_rsa file is needed at the root of the project.

Invoke the logic to create the profile:

```
profile:
  extends: .abstract_profile
  variables:
    floating_ip_pool: ext-net-37
    machine_type: s1.small
    network_name: EBI-TSI-davidyuan_data_private
    disk_image_name: centos7
    cloud_provider: OSTACK
    os_project_name: EBI-TSI-davidyuan
    os_project_id: 7c1d8ce04aed460b88d87d4eb20d51fd
    os_auth_url: 'https://extcloud06.ebi.ac.uk:13000/v3'
    os_identity_api_version: 3
    os_region_name: regionOne
    os_user_domain_name: Default
```

It is intentional to hard-code the default config_name, param_name and cred_name in the library. This is to prevent the proliferation of ECP profiles.

Invoke the logic to deploy CPA:

```
deployment:
  extends: .abstract_deployment
  variables:
    APP: >
      {
          "repoUri": "https://github.com/EMBL-EBI-TSI/cpa-instance"
      }
    DPMT: >
      {
        "applicationName": "${app_name}",
        "applicationAccountUsername": "${app_account_name}",
        "configurationAccountUsername": "${config_account_name}",
        "attachedVolumes": [],
        "assignedInputs": [],
        "assignedParameters": [],
        "configurationName": "${config_name}"
      }
  #when: manual
```

Provide JSON input for repoUri and deployment descriptor for the deployment job defined in the library.

Create and invoke the logic to test the workload:

```
workload:
  extends: .abstract_workload
  script:
    - deployment_id=$(cat ${ARTIFACT_DIR}/deployment.json)
    - deployment_id=$(ecputil getDeploymentId "${deployment_id}")

    # accessIp may be null in deployment.json if instance was not deployed yet
    - data=$(ecp get deployment ${deployment_id} -j)
    - ip=$(ecputil getAccessIp "${data}")

    # Over simplified test case "df -h" on the new VM
    - ssh -o StrictHostKeyChecking=No -i ${ID_RSA_FILE} centos@${ip} "df -h" > $
↪{ARTIFACT_DIR}/workload.log

    - ecp 'login' '-r'
  when: delayed
  start_in: 1 minutes
  retry:
    max: 2
    when:
      - script_failure
  #when: manual
  dependencies:
    - deployment
```

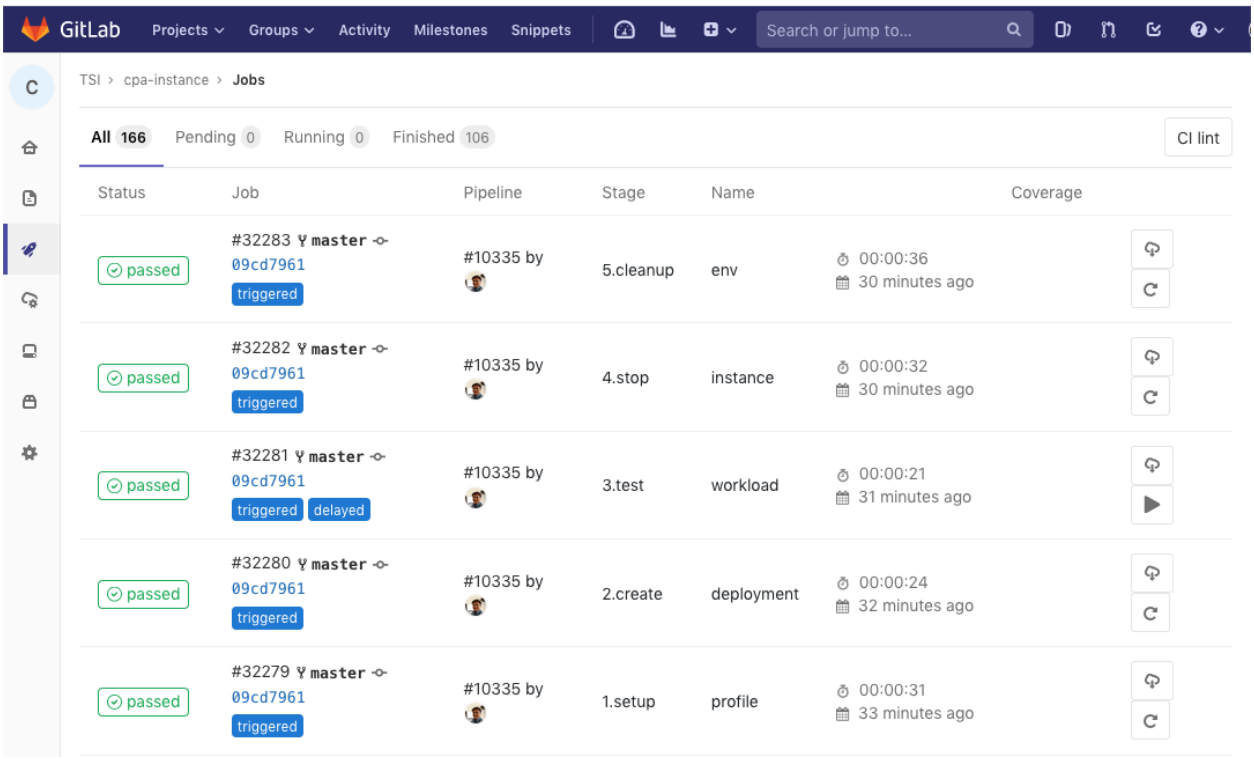
The logic to test a workload is highly specific. The job in the library is just a scalaton.

Once the pipeline files .gitlab-ci.yml are created or updated, use the Dashboard to run the test cases.

8.35.4 Downloading and viewing logs

Logs and intermediate results are exposed on GitLab if the files are stored under ARTIFACT_DIR. They can be downloaded for further analysis. This also means that the sensitive data should never be included as artifacts without encryption.

There is a run / rerun or download button beside each job or pipeline as shown below.



8.36 DevOps toolchain from IntelliJ to ReadtheDocs for publishing

The toolchain for documentation consists of the following tools:

1. IntelliJ IDEA CE
2. GitLab
3. Read the Docs

The source is stored in GitLab. Follow the instructions to install IntelliJ IDEA CE. Check out the project from source control repository. The easiest is to clone the project via HTTPS.

After **.md* or **.rst* files are added or edited, a Personal Access Token is needed to push the changes instead of a regular password. This is because the 2FA authentication is enabled. Please make sure that the token has api scope and never expires. Make sure IntelliJ remembers the token / password so that it can be reused for pushes in the future.

The ReadtheDocs project is configured to pull the respsitory from GitLab. Build can be invoked by clicking "Build Version:". It is usually the *latest* version. It is possible to configure Read the Docs to trigger a build with every push. However, it makes little sense.

8.36.1 Local build with Sphinx

To create a local build environment, install *sphinx* with *pip*:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ sudo pip install sphinx
```

Initialize the project with *sphinx-quickstart*. Make sure the *rst* is selected:

```
C02XD1G9JGH7:tsi-ccdoc davidyuan$ sphinx-quickstart
Welcome to the Sphinx 1.8.2 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Selected root path: .

You have two options for placing the build directory for Sphinx output.
Either, you use a directory "_build" within the root path, or you separate
"source" and "build" directories within the root path.
> Separate source and build directories (y/n) [n]: y

Inside the root directory, two more directories will be created; "_templates"
for custom HTML templates and "_static" for custom stylesheets and other static
files. You can enter another prefix (such as ".") to replace the underscore.
> Name prefix for templates and static dir [_]:

The project name will occur in several places in the built documentation.
> Project name: tsi-cc
> Author name(s): David Yuan
> Project release []: 1.0

If the documents are to be written in a language other than English,
you can select a language here by its language code. Sphinx will then
translate text that it generates into that language.

For a list of supported codes, see
http://sphinx-doc.org/config.html#confval-language.
> Project language [en]:

The file name suffix for source files. Commonly, this is either ".txt"
or ".rst".  Only files with this suffix are considered documents.
> Source file suffix [.rst]:

One document is special in that it is considered the top node of the
"contents tree", that is, it is the root of the hierarchical structure
of the documents. Normally, this is "index", but if your "index"
document is a custom template, you can also set this to another filename.
> Name of your master document (without suffix) [index]:
Indicate which of the following Sphinx extensions should be enabled:
> autodoc: automatically insert docstrings from modules (y/n) [n]:
> doctest: automatically test code snippets in doctest blocks (y/n) [n]:
> intersphinx: link between Sphinx documentation of different projects (y/n) [n]:
> todo: write "todo" entries that can be shown or hidden on build (y/n) [n]:
> coverage: checks for documentation coverage (y/n) [n]:
> imgmath: include math, rendered as PNG or SVG images (y/n) [n]:
> mathjax: include math, rendered in the browser by MathJax (y/n) [n]:
> ifconfig: conditional inclusion of content based on config values (y/n) [n]:
> viewcode: include links to the source code of documented Python objects (y/n) [n]:
> githubpages: create .nojekyll file to publish the document on GitHub pages (y/n)␣
↪[n]:

A Makefile and a Windows command file can be generated for you so that you
only have to run e.g. `make html' instead of invoking sphinx-build
directly.
> Create Makefile? (y/n) [y]:
```

(continues on next page)

(continued from previous page)

```
> Create Windows command file? (y/n) [y]: n

Creating file ./source/conf.py.
Creating file ./source/index.rst.
Creating file ./Makefile.

Finished: An initial directory structure has been created.

You should now populate your master file ./source/index.rst and create other␣
↪documentation
source files. Use the Makefile to build the docs, like so:
   b.make builder
where "builder" is one of the supported builders, e.g. html, latex or linkcheck.
```

now, a local test build can be created:

```
c02xd1g9jgh7:tsi-ccdoc davidyuan$ make html
```

the index.html can be opened to see a skeleton project documentation:

```
c02xd1g9jgh7:tsi-ccdoc davidyuan$ open build/html/index.html
```

8.36.2 Using markdown with sphinx

the restructuredtext is powerful but its syntax is also a bit overwhelming. the markdown can be a alternative for simple documentation. the 'recommonmark' is needed to use markdown and restructuredtext in the same sphinx project.:

```
c02xd1g9jgh7:tsi-ccdoc davidyuan$ sudo pip install recommonmark
```

then, *conf.py* would also needs to be updated.:

```
from recommonmark.parser import commonmarkparser

source_parsers = {
  '.md': commonmarkparser,
}

source_suffix = ['.rst', '.md']
```

8.36.3 Reference

1. Restructured text primer
2. Getting started with sphinx
3. Markdown cheat sheet

8.37 HPC with Azure CycleCloud

Azure CycleCloud is designed to support HPC in the cloud environment, specifically on Azure. It is tightly integrated with the vendor technologies. The controller is containerized, which can run anywhere under Docker.

The only supported cloud is Azure. Thus, it makes little sense to run the container anywhere other that Azure. Install Azure CLI as documented in Deployment of Kubernetes Cluster onto Various Clouds to get started.

8.37.1 Cost

CycleCloud seems very expensive. It is roughly £1 per hour by just idling a single 4-core master node of a Slurm cluster.

8.37.2 HPC on CycleCloud

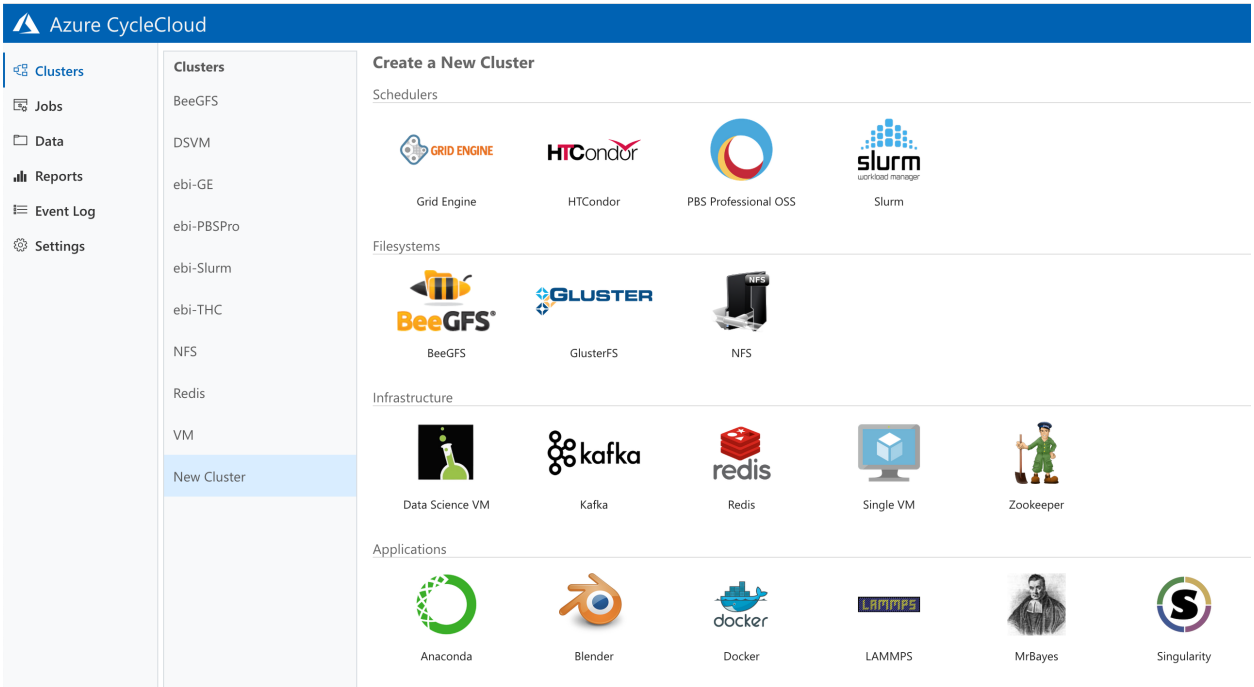
CycleCloud provides an FQDN (e.g. *cyclecloud.westeurope.azurecontainer.io* named in the deployment) mapping to an external IP once deployed. As shown on our deployment https://cyclecloud.westeurope.azurecontainer.io/cloud/cluster_list, the following schedulers are supported natively:

1. Slurm
2. PBS
3. HTCondor
4. Grid Engine

The following file systems are supported natively:

1. BeeGFS
2. GlusterFS
3. NFS

Most interesting to us is that both Docker and Singularity are supported natively by CycleCloud.



8.37.3 Configuration

Configure Azure CycleCloud by following the instructions on GUI. Tricky part is to provide the information of service principle so that an Azure subscription is available as a cloud provider.

8.37.4 CLI

The commandline interface is critical for any real workload. Download it from the GUI (e.g. https://<site_name>.<location>.azurecontainer.io/download/tools/cyclecloud-cli.zip). Unzip and run *install.sh*. */etc/paths* may need to be updated to include */Users/davidyuan/bin* in *PATH*.

Here are two tutorials to customize the Azure CycleCloud:

- To modify a cluster template
- To deploy customer application

8.37.5 Subscription

HPC consumes significantly amount of resources. It is a good idea to create a separate subscription for each project to force the separation of resources and accounting. It also makes scripting a bit easier by allowing some parameters hard-coded.

az login reports a list of subscriptions and which one is the default. The same information can also be found via *az account list*. Create a new one via the portal. It is always a good idea to set the present working subscription as default:

```
az account set --subscription "<subscription_id>"
```

8.37.6 Service principle

At least one service principle is needed to allow CycleCloud to access Azure cloud resources in a subscription. It must be created at the subscription scope:

```
az ad sp create-for-rbac --scopes="/subscriptions/<subscription_id>"
```

Take note of the JSON response. The information is needed to create cloud provider account in CycleCloud GUI. It is quite hard to find it again via *az ad sp list* and application secret will be hidden.

8.37.7 Resource group

Use *az account list-locations* to find a valid location code for a subscription. Note that not all services are available in all locations.

Create a resource group to organize resources for CycleCloud:

```
az group create --name ${CIName} --location ${Location}
```

8.37.8 Vnet and subnet

The CycleCloud requires three subnets for production. They are needed to create HPC clusters in GUI.

- cycle: The subnet in which the CycleCloud server is started in

- compute: A /22 subnet for the HPC clusters
- user: The subnet for creating user logins

For non-production, one subnet is enough:

```
az network vnet create --name ${CIName} --resource-group ${CIName} --address-prefix␣
↪10.0.0.0/16
az network vnet subnet create --resource-group ${CIName} --vnet-name ${CIName} --name␣
↪compute --address-prefix 10.0.0.0/22
```

8.37.9 Container instance

The CycleCloud is packaged as RPM, DEB or container. The container does not support Kubernetes at present. This means that it can not be running on AKS but can be installed on Azure Container Instances:

```
az container create \
  --resource-group ${CIName} \
  --location ${Location} \
  --name ${CIName} \
  --dns-name-label ${CIName} \
  --image mcr.microsoft.com/hpc/azure-cyclecloud \
  --ip-address public \
  --ports 80 443 \
  --cpu 2 \
  --memory 4 \
  -e JAVA_HEAP_SIZE=2048 FQDN="${FQDN}"
```

8.38 HPC with Slurm on GCP

A Slurm cluster can be created easily on GCP, following instructions in the git repository Slurm on Google Cloud Platform. Its Terraform script is in beta. Its Deployment Manager script is in production quality with excellent security design.

8.38.1 Accessing with CLI

The newly created cluster has a dedicated login node. In the most secure configuration, no public IPs are assigned to any nodes. The firewall only allows ICMP and TCP port 22. Follow instructions in Accessing GCP node from CLI to access the Slurm cluster via SSH, SCP, rsync, etc.

8.38.2 Enable GCSFuse

GCSFuse presents storage objects as files on shared directories. This allows you to access Petabytes of storage without pre-allocating anything. There is no downloading or uploading needed. You do not need to hard-code any keys or passwords, either.

Edit basic.tfvars if you are using Terraform script or slurm-cluster.yaml if you are using Deployment Manager to create Slurm clusters. Here are the steps in Terraform:

```
Add "https://www.googleapis.com/auth/devstorage.full_control" to compute_node_scopes,␣
↪for example:

  compute_node_scopes           = [
    "https://www.googleapis.com/auth/monitoring.write",
    "https://www.googleapis.com/auth/logging.write",
    "https://www.googleapis.com/auth/devstorage.full_control"
  ]

(Optional) If you are using a none default service account, make sure it has "Storage␣
↪Admin" role. If you are using the default service account (i.e., compute_node_
↪service_account = "default"), nothing needs to be done.

Edit network_storage for worker nodes, for example:

  network_storage = [
    {
      server_ip = "none"
      remote_mount = "dy-test-301718"
      local_mount = "/data"
      fs_type = "gcsfuse"
      mount_options = "file_mode=666,dir_mode=777,allow_other"
    }
  ]
```

This can be found in our slurm-master repo where we put the latest working code from official repo,

https://gitlab.ebi.ac.uk/TSI/slurm-main/-/blob/master/tf/examples/basic/basic.tfvars.sample

You can fork this repo, add your changes in tfvars and follow the README to setup CI/CD of your slurm cluster infrastructure.

8.38.3 Monitoring

Google Stackdrive should be enabled for monitoring. The dashboard can be accessed from Google Cloud Console, for example https://console.cloud.google.com/monitoring/dashboards/resourceList/gce_instance?project=citc-slurm&timeDomain=1h.

8.38.4 References

1. HPC made easy: Announcing new features for Slurm on GCP, https://cloud.google.com/blog/products/compute/hpc-made-easy-announcing-new-features-for-slurm-on-gcp
2. Slurm on Google Cloud Platform, https://github.com/SchedMD/slurm-gcp#stand-alone-cluster-in-google-cloud-platform

8.39 Installing OpenStack CLI on Mac OS X

8.39.1 Prerequisite

Python 2.7 and setuptools are installed by default. This can be verified –version option on them.

```
C02XD1G9JGH7:~ davidyuan$ python --version
Python 2.7.10
C02XD1G9JGH7:~ davidyuan$ easy_install --version
setuptools 18.5 from /System/Library/Frameworks/Python.framework/Versions/2.7/Extras/
↪lib/python (Python 2.7)
C02XD1G9JGH7:~ davidyuan$
```

8.39.2 Installation

Install pip 18.0 for Python 2.7.10 as root.

```
# easy_install pip
```

The package tornado is required by openstack client. It can be installed with pip as root first if not installed already. Otherwise, error message "matplotlib 1.3.1 requires nose, which is not installed. matplotlib 1.3.1 requires tornado, which is not installed." might be seen.

```
# pip install tornado
```

The installation of python-openstackclient may fail with the error of "Cannot uninstall 'six'. It is a distutils installed project and thus we cannot accurately determine which files belong to it which would lead to only a partial uninstall." User options –ignore-installed six –user to bypass it.

```
# pip install python-openstackclient --ignore-installed six --user
```

8.39.3 Configuration

Clients are installed under "/Users/davidyuan/Library/Python/2.7/bin". It is easier to add the directory to PATH.

```
C02XD1G9JGH7:bin davidyuan$ sudo vi /etc/paths
```

Connect to Horizon to download RC files under API access at https://extcloud05.ebi.ac.uk/dashboard/project/access_and_security/. Place the file EBI-TSI-DEV-openrc.sh under a working directory (e.g. /Users/davidyuan/). Make it executable.

```
chmod a+x EBI-TSI-DEV-openrc.sh
```

8.39.4 Verification

```
C02XD1G9JGH7:~ davidyuan$ ./EBI-TSI-DEV-openrc.sh
Please enter your OpenStack Password:
C02XD1G9JGH7:~ davidyuan$ nova list
+--------------------------------------+------------------------+---------+----------
↪--+-------------+-------------------------------------------------------+
| ID                                   | Name                   | Status  | Task␣
↪State | Power State | Networks                                              |
+--------------------------------------+------------------------+---------+----------
↪--+-------------+-------------------------------------------------------+
| d84f4480-7668-45d5-b865-2c15e5be3f27 | bastion                | SHUTOFF | -          ␣
↪  | Shutdown    | test_network=192.168.0.45                             |
| fbd97878-cdeb-40a6-80ba-34ce05c93e6e | dsds-test              | ACTIVE  | -          ␣
↪  | Running     | test_network=192.168.0.48                             |
| 473a240d-ed73-416a-a981-41c7cb856fed | gridftp1               | SHUTOFF | -          ␣
↪  | Shutdown    | test_network=192.168.0.41, 193.62.52.98               |
| 5b1b9d6a-ebaa-48bc-a241-786230eee487 | lsf-master             | SHUTOFF | -          ␣
↪  | Shutdown    | test_network=192.168.0.18                             |
```

(continues on next page)

(continued from previous page)

```
| 65430b10-16f2-490a-8b3a-3f20064c2ed6 | lsf-node1                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.15                                |
| c7ddf10a-36c7-4c01-9f0f-673d8f42b575 | lsf-node10               | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.24                                |
| 9976bd27-6d09-4561-a278-0a43e221a3a1 | lsf-node11               | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.20                                |
| 8ad71500-27a5-4895-ae18-1b82dac1a864 | lsf-node12               | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.21                                |
| 6b9812ab-6f0d-414b-89a7-fc408a2c05ad | lsf-node13               | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.22                                |
| 795c57c6-1695-4d1b-bda2-b87788830106 | lsf-node2                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.25                                |
| 06f122d0-25b1-4a65-aa52-4421f7309b98 | lsf-node3                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.26                                |
| 6633ae23-227c-43ac-b1ec-a3adf887db23 | lsf-node4                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.16                                |
| 7fe0532a-4b00-4069-b47e-335bef1c727d | lsf-node5                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.17                                |
| a0cf962f-04a5-4052-a3eb-bbd211ed8907 | lsf-node6                | ACTIVE  | -
↪  | Running     | test_network=192.168.0.12                                |
| 246ebd68-be1e-4a28-8cd8-d7f780b77c8a | lsf-node7                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.13                                |
| bfc75e2a-b6d5-4abf-ac35-eee511398aed | lsf-node8                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.14                                |
| c154aafb-715e-42da-9e15-0b828c855d17 | lsf-node9                | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.23                                |
| c9f8192c-a0ec-40e1-b15e-e40e1a37aec8 | nfs-server-1             | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.34, 193.62.52.67                  |
| 00fb3ad6-a57f-458e-9b99-1437f2041f1d | pymol-test               | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.37, 193.62.52.122                 |
| b32bb42b-79b3-4cc9-adbb-114e30d0c163 | tesk-k8s-prod-master     | ACTIVE  | -
↪  | Running     | tesk-k8s-prod-network=10.0.0.5, 193.62.55.68             |
| a1c83dae-b244-4683-855e-56c6de5c79b9 | tesk-k8s-prod-node-0     | ACTIVE  | -
↪  | Running     | tesk-k8s-prod-network=10.0.0.9                           |
| 6a881aea-701a-447e-90cc-855d6ffa31e0 | tesk-k8s-prod-node-1     | ACTIVE  | -
↪  | Running     | tesk-k8s-prod-network=10.0.0.10                          |
| 6f2857b8-80c6-4623-9328-df57975ac7f9 | tesk-k8s-prod-node-2     | ACTIVE  | -
↪  | Running     | tesk-k8s-prod-network=10.0.0.6                           |
| 3094c4e6-2d99-43cd-8452-27e56a464edf | tesk-k8s-prod-node-3     | ACTIVE  | -
↪  | Running     | tesk-k8s-prod-network=10.0.0.7                           |
| e783e094-e781-4908-8d8f-906bba472c4e | tesk-k8s-prod-node-4     | ACTIVE  | -
↪  | Running     | tesk-k8s-prod-network=10.0.0.8                           |
| 75fd4c5a-4b6b-4062-ae3a-5bacba7bf6f9 | tesk-k8s-testing-master  | ACTIVE  | -
↪  | Running     | tesk-k8s-testing-network=10.0.0.105, 193.62.55.65        |
| 26f576f9-f053-48e5-a9e4-f24555be514f | tesk-k8s-testing-node-0  | ACTIVE  | -
↪  | Running     | tesk-k8s-testing-network=10.0.0.104                      |
| 8e7b66ec-5f47-41af-bad0-8c6a5b35155d | tesk-k8s-testing-node-1  | ACTIVE  | -
↪  | Running     | tesk-k8s-testing-network=10.0.0.103                      |
| cf1d3824-1654-4e2e-b3a7-61281c7e75d2 | test_comps               | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.11, 193.62.52.84                  |
| 41e000ca-7369-415e-b64a-a2640c6752ed | tsi1535451150700-1       | SHUTOFF | -
↪  | Shutdown    | test_network=192.168.0.44, 193.62.52.105                 |
+--------------------------------------+--------------------------+---------+----------
↪--+-------------+----------------------------------------------------------+
```

8.39.5 Configuration

Import or export key

OpenStack images tend to have password logon disabled for security. You can either import a public key into OpenStack or export a private key from OpenStack under Access & Security" > "Key Pairs".

Import public key

It is easier and safer to take this approach. Perform the following on the SSH client to generate a RSA key.

```
ssh-keygen -t rsa -f cloud.key
```

Copy and paste the content of cloud.key after clicking "Import Key Pair"

Export private key

For multiple SSH clients and SFTP clients, the better options is to export the private key from OpenStack. Click "Create Key Pair" and save the private key locally. Change permission to 600 to secure the key immediately.

```
chmod 600 ./id_rsa
```

Move the key to the default location.

```
mv ./id_rsa ~/.ssh
```

Store the key in the keychain.

```
ssh-add -K ~/.ssh/id_rsa
```

Generate public key from a private key

It is impossible to generate a private key from a public key on most servers due to limited computing power. However, it is easy and can be handy to keep the matching public key around by running.

```
ssh-keygen -y -f ./id_rsa > ./id_rsa.pub
```

8.39.6 Reference

https://pypi.org/project/python-openstackclient/

8.40 Kubeflow for Machine Learning

Kubeflow is a cloud-native platform for machine learning based on Google's internal ML pipelines. The complete documentation is at https://www.kubeflow.org/docs/.

Kubeflow is under active development. There is one new release roughly every month. Here is a link to the release notes: https://github.com/kubeflow/kubeflow/releases/.

8.40.1 Setting up Kubeflow on GKE

Kubeflow can run on any environment with Kubernetes. If it is used for ML, model, quota and performance of GPUs become a major decision factor. Embassy Hosted Kubernetes does not have GPUs. GKE is tried first as it is the most mature environment for Kubernetes, Kubeflow and ML with GPU acceleration.

1. Create a GKE cluster. Choice of a zone is important if GPUs are needed. See *GPU availability at europe-west* for GPU accelerators in europe-west.
2. Follow instructions at https://www.kubeflow.org/docs/started/k8s/kfctl-existing-arrikto/ to deploy Kubeflow. It requires a Kubernetes cluster with LoadBalancer support. Create MetalLB if needed.
3. Get the credentials for the newly created cluster *gcloud container clusters get-credentials ${CLUSTER} –zone ${ZONE} –project ${PROJECT}*.
4. Get the IP address and open Kubeflow dashboard.

Accessing Kubeflow with the following commands:

```
IP_KUBEFLOW=$( kubectl get svc -n istio-system istio-ingressgateway -o jsonpath='{.
↪status.loadBalancer.ingress[0].ip}' )
open https://${IP_KUBEFLOW}
```

Alternatively, use port forward on local host:

```
kubectl port-forward -n istio-system svc/istio-ingressgateway 8443:443
open https://localhost:8443
```

8.40.2 Setting up Jupyter notebook

Jupyter notebooks can be created easily on the Kubeflow dashboard.

1. Click *Notebook Servers* to create as many as you want.
2. Click *CONNECT* to start using a notebook server.

Note that the number of GPUs in a cluster can be listed as documented in https://www.kubeflow.org/docs/notebooks/setup/, for example:

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,GPU:.status.allocatable.
↪nvidia\.com/gpu"

NAME                                          GPU
gke-tsi-gpu-1-default-pool-c5d48ec2-8ckx      <none>
gke-tsi-gpu-1-default-pool-c5d48ec2-lqmr      <none>
gke-tsi-gpu-1-gpu-pool-1-695efd18-wlq9        1
```

If GPUs are to be used, the Docker image for GPU and extra resource needs to be provided:

```
{"nvidia.com/gpu": 1}
```

The generated sts yaml contains resource limite of Nvidia GPUs, for example:

```
spec:
  containers:
  - env:
    - name: NB_PREFIX
      value: /notebook/davidyuan/jupyter-3
```

(continues on next page)

(continued from previous page)

```
    image: gcr.io/kubeflow-images-public/tensorflow-1.13.1-notebook-gpu:v0.5.0
    imagePullPolicy: IfNotPresent
    name: jupyter-3
    ports:
    - containerPort: 8888
      name: notebook-port
      protocol: TCP
    resources:
      limits:
        nvidia.com/gpu: "1"
      requests:
        cpu: 500m
        memory: 1Gi
```

8.40.3 Customising images for Notebook servers

It is a good idea to start customisation from an official image in GCR. Docker needs to be configured to access the images at https://gcr.io/kubeflow-images-public/

```
gcloud components install docker-credential-gcr
gcloud auth configure-docker
```

The following must be included in a Dockerfile if starting with a custom image, according to https://www.kubeflow.org/docs/notebooks/custom-notebook/. This is likely a long way:

```
ENV NB_PREFIX /
CMD ["sh","-c", "jupyter notebook --notebook-dir=/home/jovyan --ip=0.0.0.0 --no-
↪browser --allow-root --port=8888 --NotebookApp.token='' --NotebookApp.password='' --
↪NotebookApp.allow_origin='*' --NotebookApp.base_url=${NB_PREFIX}"]
```

8.40.4 Pipeline SDK

Set up and activate a venv undere ~/.kube, for example. Ensure that it is 3.5 or later. Install SDK in the venv:

```
cd ~/.kube
virtualenv venv
source venv/bin/activate
python --version

pip install https://storage.googleapis.com/ml-pipeline/release/latest/kfp.tar.gz --
↪upgrade
which dsl-compile
```

If IntelliJ is used as IDE, ensure project structure is updated with SDK in this venv. When a Python script is run, it compiles itself into Argo manifest in ZIP, which can be uploaded to Kubeflow pipeline:

```
if __name__ == '__main__':
  kfp.compiler.Compiler().compile(align_and_vc, __file__ + '.zip')
```

Alternately, compile the Python pipeline in CLI:

```
dsl-compile --py [path/to/python/file] --output [path/to/output/tar.gz]
```

8.40.5 Authentication

The authentication can be done via external IdP (e.g. Google, LinkedIn, etc.), static users or LDAP. More details can be found at https://www.kubeflow.org/docs/started/k8s/kfctl-existing-arrikto/#accessing-kubeflow.

8.40.6 Accessing Git repository

Git client is already installed. SSH terminal is available. The **.ipynb* notebooks can be easily checked in and out of Git repositories on persistent storage. Here is a cheatsheet of the most used Git commands.

- https://tsi-ccdoc.readthedocs.io/en/master/Tech-tips/DevOps-toolchain-docker.html?highlight=git%20commit#appendix-command-line-instructions

8.40.7 Accessing data via Tensorflow

As documented in https://www.kubeflow.org/docs/pipelines/sdk/component-development/, *tf.gfile* module supports both local and cloud storage paths:

```
#!/usr/bin/env python3
import argparse
import os
from pathlib import Path
from tensorflow import gfile # Supports both local paths and Cloud Storage (GCS) or S3

# Defining and parsing the command-line arguments
parser = argparse.ArgumentParser(description='My program description')
parser.add_argument('--input1-path', type=str, help='Path of the local file or GCS␣
↪blob containing the Input 1 data.')
parser.add_argument('--param1', type=int, default=100, help='Parameter 1.')
parser.add_argument('--output1-path', type=str, help='Path of the local file or GCS␣
↪blob where the Output 1 data should be written.')
parser.add_argument('--output1-path-file', type=str, help='Path of the local file␣
↪where the Output 1 URI data should be written.')
args = parser.parse_args()

gfile.MakeDirs(os.path.dirname(args.output1_path))
# Opening the input/output files and performing the actual work
with gfile.Open(args.input1_path, 'r') as input1_file, gfile.Open(args.output1_path,
↪'w') as output1_file:
  # do_work(input1_file, output1_file, args.param1)

# Writing args.output1_path to a file so that it will be passed to downstream tasks
Path(args.output1_path_file).parent.mkdir(parents=True, exist_ok=True)
Path(args.output1_path_file).write_text(args.output1_path)
```

This API fits for relatively small files in GB range if from cloud storage. There is no good solution for large files in TB range and large volume in PB range.

In addition, there is https://github.com/tensorflow/io and https://github.com/google/nucleus libraries to process specific file types. In particular, Nucleus handles special file types for genomic sequence processing. It branched from Deep Variant https://github.com/google/deepvariant.

8.40.8 Accessing data via OneData

OneClient can be installed. However, it can not be run without root privilege. It is a serious security risk to run a container with root privilege. Kubeflow will never agree to that. The integration of OneData with Kubeflow is out of the question:

```
conda install -c onedata oneclient=18.02.2
oneclient --help
oneclient: error while loading shared libraries: libprotobuf.so.19: cannot open␣
↪shared object file: No such file or directory
```

The only sensible option is for OneData to provide a provisioner as other storage vendors https://github.com/kubernetes-incubator/external-storage, where a common library by Kubernetes SIG is available at https://github.com/kubernetes-sigs/sig-storage-lib-external-provisioner.

8.40.9 GPU quotas

Apply filters to the service of *Compute Engine API*, the metric of *GPUs* and the location of *europe-west1*, for example, to find out GPU quota at https://console.cloud.google.com/iam-admin/quotas?_ga=2.179486372.-1491760115.1547984356&project=extreme-lore-114513&folder&organizationId=817248562955.

8.40.10 Separate GPU node pool

Always create a separate GPU pools in a cluster. When adding a GPU node pool to an existing cluster that already runs a non-GPU node pool, GKE automatically taints the GPU nodes with the following node taint:

- Key: nvidia.com/gpu
- Effect: NoSchedule

Additionally, GKE automatically applies the corresponding tolerations to Pods requesting GPUs by running the ExtendedResourceToleration admission controller.

This causes only Pods requesting GPUs to be scheduled on GPU nodes, which enables more efficient autoscaling: your GPU nodes can quickly scale down if there are not enough Pods requesting GPUs.

Run the following commands to add a separate GPU pool to an existing cluster, for example:

```
gcloud container node-pools create pool-gpu-1 --accelerator type=nvidia-tesla-p100,
↪count=1 --zone ${ZONE} --cluster ${CLUSTER} --num-nodes 1 --min-nodes 0 --max-nodes␣
↪2 --enable-autoscaling
kubectl apply -f https://raw.githubusercontent.com/GoogleCloudPlatform/container-
↪engine-accelerators/master/nvidia-driver-installer/cos/daemonset-preloaded.yaml
```

8.40.11 GPU availability at europe-west

To see a list of all GPU accelerator types supported in each zone, run the following command:

```
gcloud compute accelerator-types list | grep europe-west

nvidia-tesla-k80       europe-west1-d             NVIDIA Tesla K80
nvidia-tesla-p100      europe-west1-d             NVIDIA Tesla P100
nvidia-tesla-p100-vws  europe-west1-d             NVIDIA Tesla P100 Virtual␣
↪Workstation
```

(continues on next page)

(continued from previous page)

```
nvidia-tesla-k80       europe-west1-b             NVIDIA Tesla K80
nvidia-tesla-p100      europe-west1-b             NVIDIA Tesla P100
nvidia-tesla-p100-vws  europe-west1-b             NVIDIA Tesla P100 Virtual␣
↪Workstation
nvidia-tesla-p100      europe-west4-a             NVIDIA Tesla P100
nvidia-tesla-p100-vws  europe-west4-a             NVIDIA Tesla P100 Virtual␣
↪Workstation
nvidia-tesla-v100      europe-west4-a             NVIDIA Tesla V100
nvidia-tesla-p4        europe-west4-c             NVIDIA Tesla P4
nvidia-tesla-p4-vws    europe-west4-c             NVIDIA Tesla P4 Virtual Workstation
nvidia-tesla-t4        europe-west4-c             NVIDIA Tesla T4
nvidia-tesla-t4-vws    europe-west4-c             NVIDIA Tesla T4 Virtual Workstation
nvidia-tesla-v100      europe-west4-c             NVIDIA Tesla V100
nvidia-tesla-p4        europe-west4-b             NVIDIA Tesla P4
nvidia-tesla-p4-vws    europe-west4-b             NVIDIA Tesla P4 Virtual Workstation
nvidia-tesla-t4        europe-west4-b             NVIDIA Tesla T4
nvidia-tesla-t4-vws    europe-west4-b             NVIDIA Tesla T4 Virtual Workstation
nvidia-tesla-v100      europe-west4-b             NVIDIA Tesla V100
```

In summary, the following models are availabe as permanent or preemptible at present:

1. NVIDIA K80
2. NVIDIA V100
3. NVIDIA P100
4. NVIDIA P4
5. NVIDIA T4

Note that P100, P4 and T4 are also available as virtual workstation.

According to https://cloud.google.com/kubernetes-engine/docs/how-to/gpus, GKE nodepools can be created with all the GPUs above.

1. Kubernetes version > 1.9 for Container-optimised OS or > 1.11.3 for Ubuntu node image
2. Manually install GPU driver via a DaemonSet https://cloud.google.com/kubernetes-engine/docs/how-to/gpus#installing_drivers

GPUs are used on accelerators to VMs or Kubernetes clusters. The virtual infrastructure passes requests through to GPUs on the same region / zone. GPUs are now available on all regions and zones. However, quota varies. If a workload requires GPU, GPU quota needs to be checked when a region / zone is selected.

8.40.12 GPU pricing

GPU pricing seems the same in all regions. More details can be found at https://cloud.google.com/compute/all-pricing#gpus.

Regions and zones https://cloud.google.com/compute/docs/regions-zones/

8.40.13 References

- https://cloud.google.com/gpu/
- https://cloud.google.com/compute/docs/gpus/
- https://www.kubeflow.org/docs/started/k8s/kfctl-existing-arrikto/

- https://www.kubeflow.org/docs/
- https://onedata.org/docs/doc/using_onedata/onedatafs.html
- https://github.com/kubeflow/pipelines/

8.41 Tips and tricks with Docker

8.41.1 Adding $USER to docker group

If the current user can't access the docker engine, because you're lacking permissions to access the unix socket to communicate with the engine. The error "Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.37/info: dial unix /var/run/docker.sock: connect: permission denied" would occur. The easiest fix is to add the user to docker group and log back in.

```
sudo usermod -a -G docker $USER
```

8.41.2 Mounting a volume from host

With Docker bind mount, a volume or a file system can be made available to a container when started. The source needs to be the file system: local or mounted remotely from another host. The target can be any arbitrary path, which does not exist in the Docker image. If an existing path is used, the behaviour is undefined.

```
docker run -it --mount type=bind,source=/mnt/nfs,target=/app/pvol nginx
```

8.41.3 Size of a Docker image

There is no absolute rule what the best size is. In general, the performance of a container at both build time and runtime is the inverse of its size. Images over 1 GB should be looked at at present.

More importantly, it is the good old trade off of "cohesion vs. coupling" to be concerned about. It is the choice of architecture of "layered vs. microservice" to be debated. It is the principle of separating data from computing, user data from prod data, transient data from persistant data to be abid by. Architects in the Cloud Consulting Team could provide suggestions and second opinions.

In practice, always rebuild an image from scratch instead of an existing image. Always start with an empty directory with `Dockerfile` only. Always pick a minimal base image for software dependencies. Use `.dockerignore` to get rid of unwanted files and directories.

8.41.4 Running processes in a container as nonroot

By default, root user is assumed inside a container when it is started even by a nonroot user. This is because the Docker image was built as root by default. This is the norm in Docker. This security exposure has long be criticized.

A container can be run as any arbitrary user, for example as the current user with `--user $(id -u):$(id -g)`.

```
[centos@tsi1539957622607-k8s-master ~]$ docker run -it --user $(id -u):$(id -g) --
↪mount type=bind,source=/mnt/nfs,target=/app/pvol nginx
I have no name!@66b482bb3e63:/$ id
uid=1000 gid=1000 groups=1000
I have no name!@66b482bb3e63:/$  exit
exit
[centos@tsi1539957622607-k8s-master ~]$ id
```

(continues on next page)

(continued from previous page)

```
uid=1000(centos) gid=1000(centos) groups=1000(centos),4(adm),10(wheel),190(systemd-
↪journal),993(docker) context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
[centos@tsi1539957622607-k8s-master ~]$
```

`I have no name!?` This is just an indication that it is an useless attempt to try running a container with a user it knows nothing about. All the files and directories in the ontainer are still owned by `root:root`. There is no `/home/$USER`. There is no `sudo`. `su -` does not work, either.

The only option to is to create a nonroot user and changes the default to that one at the build time. For example,

```
FROM openjdk:8-jdk
RUN useradd --create-home -s /bin/bash nonrootuser
WORKDIR /home/nonrootuser
USER nonrootuser
```

Now, you are stuck with the arbitrary non-root user, which host many have no idea about. Without `sudo` command, You have have to built the user into root group and depend on `su -` to accommoplish many tasks. A lot of complications arises from there. `root:root` is really "the lesser of two evils".

8.41.5 Containers should be immutable & ephemeral

Docker containers serve dual purposes for both Dev and Ops:

1. Delivering software package for a very specific purpose.
2. Providing a controlled runtime environment for a specific function.

Many people tend to treat Docker containers as a "hamster cages". They include not only the software (a.k.a. the hamster) but also everything the hamster needs and produces (i.e. input, output and waste). This makes Docker containers not only larger than necessary but also mutable and irreplaceable. This makes such containers unmanageable. It can not be upgraded, replaced or even relocated.

Containers should not be self-sufficient units. They should be stateless and ephemeral. In other words, containers should be designed to address their Ops aspects first and foremost - controlled runtime environments. A good rule to apply is that all containers should be read-only. Logs should be mapped to external storage. Temporary files should be on temporary file systems. Input should always be taken from external sources. Containers should never be self-sufficient. They always need to be managed by additional software such as Kubernetes, and interconnected with other containers, external storage, message queues, etc..

Overall, Docker containers do not bring in simplicity unless they are used in extremely simple situations with quick and dirty solutions (picturing yourself hamster cages with smells and droppings). It is quite opposite that Docker containers exposes complexities and management overhead for complex applications.

8.41.6 Paying attention to build context

The current working directory, where `docker build` is issued, is called build context. Everything (i.e. every file under the current directory and all sub-directories) under the build context is sent to Docker daemon, and ended up in the image built. If the build command is issued under root directory, the final image is as large as the entire file system. If the build command is issued under different directories every time, the size and content of the images are different in each build.

To avoid such casual mistake, always create a build script to drive docker build. Always create and empty directory, and issue `docker build` from there.

For any real project, a CI/CD toolchain should be created to make sure that an image is built consistently and all the changes are traceable. See https://tsi-ccdoc.readthedocs.io/en/master/Tech-tips/DevOps-toolchain-docker.html for details how to create a toolchain.

In particular, the YAML file to invoke `docker build` should look like the following:

```
  script:
    - git clone ${CI_GIT_URL}
    - cd ${CI_GIT_REPO_NAME}
    - docker build -f ${CI_DOCKER_FILE} -t ${CI_SOURCE_IMAGE} . | tee ${ARTIFACT_DIR}/
↪build.log
    - docker tag ${CI_SOURCE_IMAGE} ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG}
    - docker push ${CI_REGISTRY_IMAGE}:${CI_COMMIT_REF_SLUG} | tee ${ARTIFACT_DIR}/
↪push.log
```

1. Create and update `Dockerfile` in an IDE such as IntelliJ.
2. Push build changes into Github or Gitlab.
3. Create and update CI/CD script `.gitlab-ci.yml`. Make sure that the good practices are coded in the script.
    1. Create a fresh copy of build repository to a new directory via `git clone`.
    2. Change directory to the new directory for build repository.
    3. Issue `docker build` from the root of the build repository. This ensures that everything in the build repository is included in the image, and nothing else.
    4. Tag an image before push it to Docker Hub.
    5. Log `docker build` and `docker push` for detailed analysis later.
    6. If something in the build repository should not be included into the image, use `.dockerignore` to keep the size of the image even smaller.

8.41.7 Best practices for Dockerfile

Docker has published a document with extensive hints and tips how to write a good Dockerfile. Here is a link: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

8.41.8 Recommendations for the packaging and containerizing of bioinformatics software

https://f1000research.com/articles/7-742

8.42 Tips and tricks with Terraform

Terraform is available as a single binary package, which can be downloaded from https://www.terraform.io/downloads.html for various operating systems. If desired, update PATH to include `terraform` by updating `~/.profile` or `~/.bashrc` or by moving it to a well-known location such as `/usr/local/bin`. Remember to initialize it with `terraform init` in the directory with *.tf.

8.42.1 Getting started

It is highly recommended to read through Getting Started Guide if you are new to the language.

8.42.2 Benefit to use Terraform

Terraform is a language to provide common syntax to interact with multiple cloud providers. Unlike Java, which can be written once and run everywhere, it does not mask the differences from the cloud providers supported: AWS, GCP, Azure or OSK. DevOps engineers still have to understand the details of each cloud provider and write scripts specific to it.

In addition to one language for all major cloud providers, there are sizable collections of modules for them. DevOps engineers are reuse the modules for faster development and better code quality.

8.42.3 Drawback to use Terraform

Terraform is not an official language by any cloud providers. It is always lag behind any improvement by the cloud providers. DevOps engineers will still have to use native languages if latest features are required.

Terraform support is heavily skewed towards AWS with GCP as distant second. The support to OSK is limited to very basic features.

8.42.4 What, not how

Terraform is a declarative language. DevOps engineer describes what the target environment would be instead of how to build it. The engine would manage to figure out how to create a deployment. It is important to keep tracking the state of each deployment. Terraform does not have build-in registry for this. DevOps engineers need to set up good convention for this.

*.tf files for a deployment needs to be in the same directory. Different deployments can not be mixed in the same directory.

8.42.5 Plan ahead

It is always a good idea to plan ahead before actual deployment. This provides a quick way for try-and-error without actual deployment, for example:

```
terraform plan ~/IdeaProjects/cpa-kubernetes.github/ostack/terraform/
```

8.42.6 Custom modules and submodules

A module must contain a main.tf. It is a good idea to place custom modules in a Git repository or to publish them to Terraform registry. Modules can be nested. Submodules are invisible outside a module. It is important to nest modules logically. This makes the code reuse much easier.

```
module "consul" {
  source = "hashicorp/consul/aws"
  version = "0.1.0"
}
```

```
module "consul" {
  source = "github.com/hashicorp/example"
}
```

8.42.7 Provisioners

It can be handy to call a provisioner for some simple configuration. Ansible can be used for more complex configurations.

```
provisioner "file" {
  source      = "script.sh"
  destination = "/tmp/script.sh"
}

provisioner "remote-exec" {
  inline = [
    "chmod +x /tmp/script.sh",
    "/tmp/script.sh args",
  ]
}
```

8.42.8 Reference

1. Terraform Recommended Practices