Summary

There are three main tasks, where the first task consists of two parts:

- Your version of what DevOps is.
- Gitlab pipeline with sample C# console application. The focus here is on pipeline, whereas C# console application is to check development knowledge/skills.
- Load balancer local Docker environment setup and configuration.

All questions and the full solution should be sent to tine.lazar@cosylab.com.
Subtasks marked as **[Optional]** are considered bonus points. Implement them if they don't significantly prolong the time you need to complete the task.

DevOps definition

In your own words define what DevOps is.

Prepare a Gitlab pipeline with C# console application

Gitlab pipeline

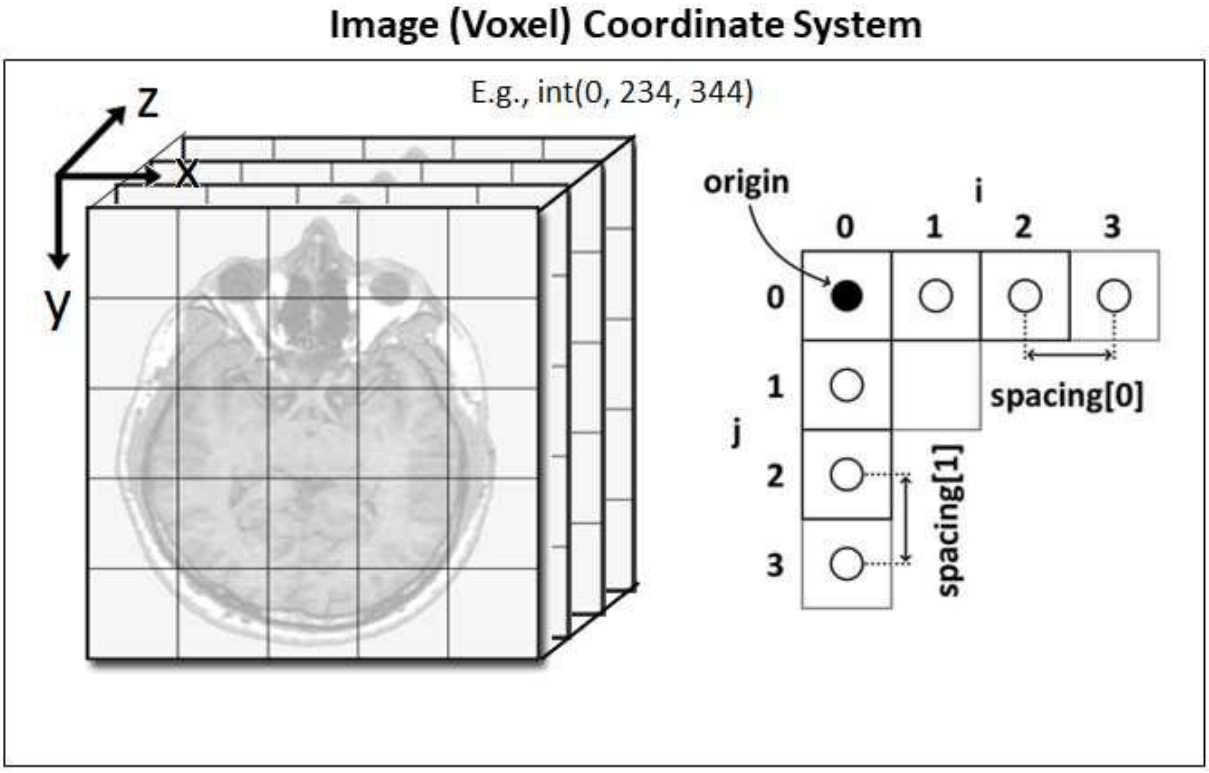
You can use Gitlab.com for this task. There is no confidential information used.

The main task is:

1. Prepare a Gitlab pipeline for building, testing, and Nuget deployment/publishing of the sample C# console application, described below.

Application description and inputs

In cancer radiotherapy, CT imaging is often used as a diagnostic tool that helps doctors pinpoint the position of cancerous tissue. A single CT image can be thought of as a 3D volume consisting of cuboid sub-elements called voxels. Voxels make up a CT image similarly as pixels make up a 2D image. Each voxel within a CT image has a numeric value, a Hounsfield unit, which quantitatively describes its radiological density. It is also possible to think of a CT image as a collection of 2D slices, each slice occupying a different depth represented by a Z coordinate. Example of a CT slice is shown in the picture below:



Picture source: https://github.com/ymirsky/CT-GAN

Here, a single CT slice is shown as a 2D picture in the XY plane at a specific Z location. The 2D array representing the slice can be a flat array with indices increasing from left to right, and from top to bottom:

the top-left element has an index 0, while the bottom-right element has an index (N x M) – 1, where N is slice width and M is slice height.

For the exercise, all CT slices will be stored as simplified text files located in a single directory.

Each CT slice will be represented as a single .txt file with an arbitrary name and the content of the following form:

```
SliceZPosition
SliceWidth SliceHeight
Space-separated flat array of values representing Hounsfield units of the slice
```

For instance, slice0.txt could contain a slice with 25 elements:

```
12.58
5 5
0 0 0 0 0 0 68 955 192 527 69 719 240 -576 -312 921 -294 -324 607 -882 221 0 2 5 0
```

In practice, slice dimensions will be larger, e.g. 512 x 400 or similar. A single directory will contain many such CT slice files – up to several hundreds. You can assume that no other files will be present in the directory.

Console application features

The console application shall have the following features:

1. The application shall consist of a Main project and two Subprojects. Subproject 2 depends on Subproject 1. Both subprojects need to be deployed also as Nuget packages. All projects must have at least one unit test.
   Content of Subprojects is not important.
   When a project is built locally, it shall use ProjectReference, when it is built in pipeline, it shall use PackageReference.
2. It should be possible to enter path to a directory containing CT slices. On selection confirmation, the application will start asynchronously loading the slices from the directory and reporting progress.
3. The processing operation will, for each loaded slice, find the minimum and maximum value within the slice, as well as the average of all slice values. For minimum and maximum values, the row and column of the respective element within the CT slice should be shown as well.
   For instance, the Min value (row, column) column value that reads -998 (0, 12) would mean that the minimum value found within the given slice was -998 at row 0 and column 12.
4. The app should print a table like this filled with data:

Slice Position	Dimensions	Min value (row, column)	Max value (row, column)	Average value
-10.0	520 x 520			
-8.0	520 x 520			
-6.0	520 x 520			
...	...			

5. The program should also use a data type that maintains the following values:
   - global minimum – the smallest value among all slice minimums.
   - global maximum – the largest value among all slice maximums.
   - global average – the average value of all slice averages.

   For global minimum and maximum, the Z location of the associated slices should be recorded as well.
6. Print the global result in a table like this (don't mind the format too much):

Global min	-815	Slice Z: -115.25
Global max	1225	Slice Z: 230.00
Global average	425.86	

7. In case there are multiple solutions (e.g. multiple elements with the same maximum or minimum), you can show just one (anyone) of them.
8. **[Optional]** The total processing time in seconds should be shown after the processing has been completed.

Load balancer evaluation task

1. Prepare a local Docker test environment with two HAProxy load balancers in HA setup, which balance between three Nginx servers in the backend. Nginx server 1 shall display "Hello One", Nginx server 2 shall display "Hello Two", Nginx server 3 shall display "Hello Three".
2. Simulate fail of Nginx server 2 and verify that traffic is not sent to it any more.
3. Simulate fail of the primary HAProxy and verify that the secondary HAProxy takes over and that the Nginx servers are still accessible.