**Exercise 1**

A sample simple web app built with Node.js that responds on port 8080 is included along with this file. Commit it to a GitHub repository where you’ll be creating the workflows/pipelines.

1. Using GitHub Actions, create a workflow/pipeline that will run a simple CI process with the included web app. Use a Github-hosted runner
   1. Use a Dockerfile and docker to containerize the application with the “latest” tag, as well as a tag with the git commit hash
      1. Make sure to expose the right ports
      2. Use a nodejs base image
      3. Ensure you include both files (index.js, package.json)
      4. The start command of the container should be “npm install && npm start”
   2. Publish the sample application container to Docker Hub
2. Using GitHub Actions, create a workflow/pipeline that will run a simple CD process with the included web app. Use a self-hosted runner on your local machine, a VM, or WSL on Windows
   1. Download the container from Docker Hub that has the “latest” tag
   2. Run the container for the first time, or update the existing container to run the new image if it’s already running
3. Using GitHub Actions, create a workflow/pipeline that will automatically run when a new commit is pushed to the main/master branch of the repo
   1. Run the CI and CD jobs in sequence (one after the other)
   2. The CD will not run if the CI fails
   3. If EITHER the CI or CD jobs fail, add a new line to a log file with the timestamp and the name of the repository (Use a self-hosted runner on your local machine, a VM, or WSL on Windows)
   4. Make sure the web app can be accessed on port 3000 from the host

You should be able to create a GitHub and a Docker Hub account for free for this exercise.

Test that the deployment works by sending an HTTP request to the running container on port 3000 and see if you receive back “Hello World!”. You can do this with a script, cURL, or an HTTP request program like Postman.

**Exercise 2**

Using Python, create a simple Flask RESTful endpoint that will return the JSON and XML versions of a list, use a POST endpoint to append at least 3 more names to the employee list

employees = [

    { 'name': Joe, 'last\_name': 'Smith', 'age': 25 },

    { 'name': Allen, 'last\_name': 'Jones', 'age': 21 },

    { 'name': Sam, 'last\_name': 'Andrews', 'age': 35 }

]

Output should return the names by ascending order of Age

Endpoints:

/add - should add an entry to the list

/json - should return the list as a JSON string

/xml - should return the list in XML format