**Practical Exercise 17 - Working with Artifacts**

**Exercise Description**

**In this practical exercise, our goal is to explore how we can leverage artifacts to share build outputs between jobs. We will also explore how to combine caching and artifacts in the same workflow.**

In order to achieve that, we will leverage a React application that we will scaffold with the help of the create-react-app utility. Check the tips section for the specific command to scaffold your React application. Here are the instructions for the exercise:

1. Generate a React application:
   1. Create a new folder named 14-artifacts at the root of the repository.
   2. Using a terminal, cd into this directory and scaffold a React application inside a react-app directory. You can either create the directory yourself or let the create-react-app utility do it for you.
   3. Once the React setup is done, you should see a success message.
   4. Take a few moments to inspect the files and get familiar with the application folder structure.
2. Create the first version of the workflow:
   1. Create a file named 14-artifacts.yaml under the .github/workflows folder at the root of your repository.
   2. Name the workflow 14 - Working with Artifacts.
   3. Add the following triggers to your workflow:
      1. workflow\_dispatch
   4. Add a job named test-build to the workflow:
      1. The job should run on ubuntu-latest
      2. The job should set 14-artifacts/react-app as the default working-directory option for run commands.
      3. The job should contain seven steps:
         1. The first one, named Checkout code, should checkout the repository code into the current working directory.
         2. The second one, named Setup Node, should setup Node with the version 20.x.
         3. The third one, named Download cached dependencies, should:
            1. Have an id of cache.
            2. Use the actions/cache@v3 third-party action.
            3. Pass the following inputs to the action:

The key should be deps-node-modules-${{ hashFiles('14-artifacts/react-app/package-lock.json') }}

It's important to provide a key that remains stable as long as the list of dependencies does not change, and that changes as soon as the list of dependencies changes. This is easily achieved by hashing the package-lock.json file, since this file contains a list of all the dependencies, both direct and indirect, of our application.

The path where to restore the cache should be 14-artifacts/react-app/node\_modules. It's important to provide the entire path, since the working-directory option set as default at the job level applies only to run commands.

* + - 1. The fourth one, named Install dependencies, should execute the npm ci command, and it should be executed if and only if there was no cache hit in the Download cached dependencies step.
      2. The fifth one, named Unit tests, should execute the npm run test command.
      3. The sixth one, named Build code, should execute the npm run build command.
      4. The seventh one, named Upload build files, should:
         1. Use the actions/upload-artifact@v4 third-party action.
         2. Pass two inputs to the action: The name input, with a value of app, which is used for later download, and the path input, with a value of 14-artifacts/react-app/build, which is used to determine which folders and/or files should be uploaded as an artifact.
  1. Add a job named deploy to the workflow:
     1. The job should run on ubuntu-latest
     2. The job should have a dependency to the test-build job.
     3. The job should contain two steps:
        1. The first one, named Download build files, should:
           1. Use the actions/download-artifact@v4 third-party action.
           2. Pass two inputs to the action: The name input, with a value of app, should have the same value as the name input from the upload-artifact action, and the path input, with a value of build, which is used to determine into which directory the artifact files should be placed.

1. Commit the changes and push the code. Trigger the workflow manually and take some time to inspect the results of the workflow run. Can you find the uploaded artifacts and download them manually? How do they look like?
2. Extend the workflow to upload test coverage reports as artifacts:
   1. Replace the hard-coded name input from the existing upload-artifact and download-artifact actions to a workflow-level env variable named build-artifact-key. The value of this variable should be app-${{ github.sha }}.
   2. Add a second workflow-level env variable named test-coverage-key and with value test-coverage-${{ github.sha }}.
   3. Change the script from the Unit tests step to generate a coverage report. If you are not sure how to do that, check the **Tips** section below.
   4. Add another step after the Unit tests steps. The step should:
      1. Be named Upload test results.
      2. Upload the contents of the 14-artifacts/react-app/coverage folder, which is generated when running the test command with the coverage option turned on. Use the value of the test-coverage-key as the name input.
3. Commit the changes and push the code. Trigger the workflow manually and take some time to inspect the results of the workflow run.

**Tips**

**Generate a coverage report for unit tests with Jest**

To generate a coverage report when using Jest, simply pass the --coverage flag to the Jest command. When using npm to run the test script, you have to add an additional -- before passing the --coverage flag, since the --coverage flag must be passed to the underlying jest command instead of the npm command. Therefore, the correct script to run is npm run test -- --coverage.