Darwin Harbour Water Quality monitoring program analysis application manual

Murray Logan

16/05/2025

Table of contents

About

This document comprises the manual for the Darwin Harbour Water Quality monitoring program analysis application. It provides information on:

- a broad overview of the structure of the application
- the application dependencies and how to install them
- starting the application
- progressing through the analysis pipeline
- visualising, interpreting and extracting outputs

Structural overview

R Graphical and Statistical Environment offers an ideal platform for developing and running complex statistical analyses as well as presenting the outcomes via professional graphical/tabular representations. As a completely scripted language it also offers the potential for both full transparency and reproducibility. Nevertheless, as the language, and more specifically the extension packages are community developed and maintained, the environment evolves over time. Similarly, the underlying operating systems and programs on which R and its extension packages depend (hereafter referred to as the *operating environment*) also change over time. Consequently, the stability and reproducibility of R codes have a tendency to change over time.

Docker containers

One way to attempt to future proof a codebase that must be run upon a potentially unpredictable operating environment is to **containerise** the operating environment, such that it is preserved to remain unchanged over time. Containers (specifically docker containers) are lightweight abstraction units that encapsulate applications and their dependencies within standardized, self-contained execution environments. Leveraging containerization technology, they package application code, runtime, libraries, and system tools into isolated units (*containers*) that abstract away underlying infrastructure differences, enabling consistent and predictable execution across diverse computing platforms.

Containers offer several advantages, such as efficient resource utilization, rapid deployment, and scalability. They enable developers to build, test, and deploy applications with greater speed and flexibility. Docker containers have become a fundamental building block in modern software development, enabling the development and deployment of applications in a consistent and predictable manner across various environments.

Shiny applications

Shiny is a web application framework for R that enables the creation of interactive and data-driven web applications directly from R scripts. Developed by Rstudio, Shiny simplifies the process of turning analyses into interactive web-based tools without the need for extensive web development expertise.

What makes Shiny particularly valuable is its seamless integration with R, allowing statisticians and data scientists to build and deploy bespoke statistical applications, thereby making data visualization, exploration, and analysis accessible to a broader audience. With its interactive and user-friendly nature, Shiny serves as a powerful tool for sharing insights and engaging stakeholders in a more intuitive and visual manner. ## Git and github

Git, a distributed version control system, and GitHub, a web-based platform for hosting and collaborating on Git repositories, play pivotal roles in enhancing reproducibility and transparency in software development. By tracking changes in source code and providing a centralized platform for collaborative work, Git and GitHub enable developers to maintain a detailed history of code alterations. This history serves as a valuable asset for ensuring the reproducibility of software projects, allowing users to trace and replicate specific versions of the codebase.

GitHub Actions (an integrated workflow automation feature of GitHub), automates tasks such as building, testing, and deploying applications and artifacts. Notably, through workflow actions, GitHub Actions can build docker containers and act as a container registry. This integration enhances the overall transparency of software development workflows, making it easier to share, understand, and reproduce projects collaboratively.

Figure 1 provides a schematic overview of the relationship between the code produced by the developer, the Github cloud repository and container registry and the shiny docker container run by user.

Installation

Installing docker desktop

To retrieve and run docker containers requires the installation of Docker Desktop on Windows and MacOSx

Windows

The steps for installing Docker Desktop are:

- **Download the Installer:** head to https://docs.docker.com/desktop/install/windows-install/ and follow the instructions for downloading the appropriate installer for your Windows version (Home or Pro).
- **Run the Installer:** double-click the downloaded file and follow the on-screen instructions from the installation wizard. Accept the license agreement and choose your preferred installation location.
- **Configure Resources (Optional):** Docker Desktop might suggest allocating some system resources like CPU and memory. These settings can be adjusted later, so feel free to use the defaults for now.

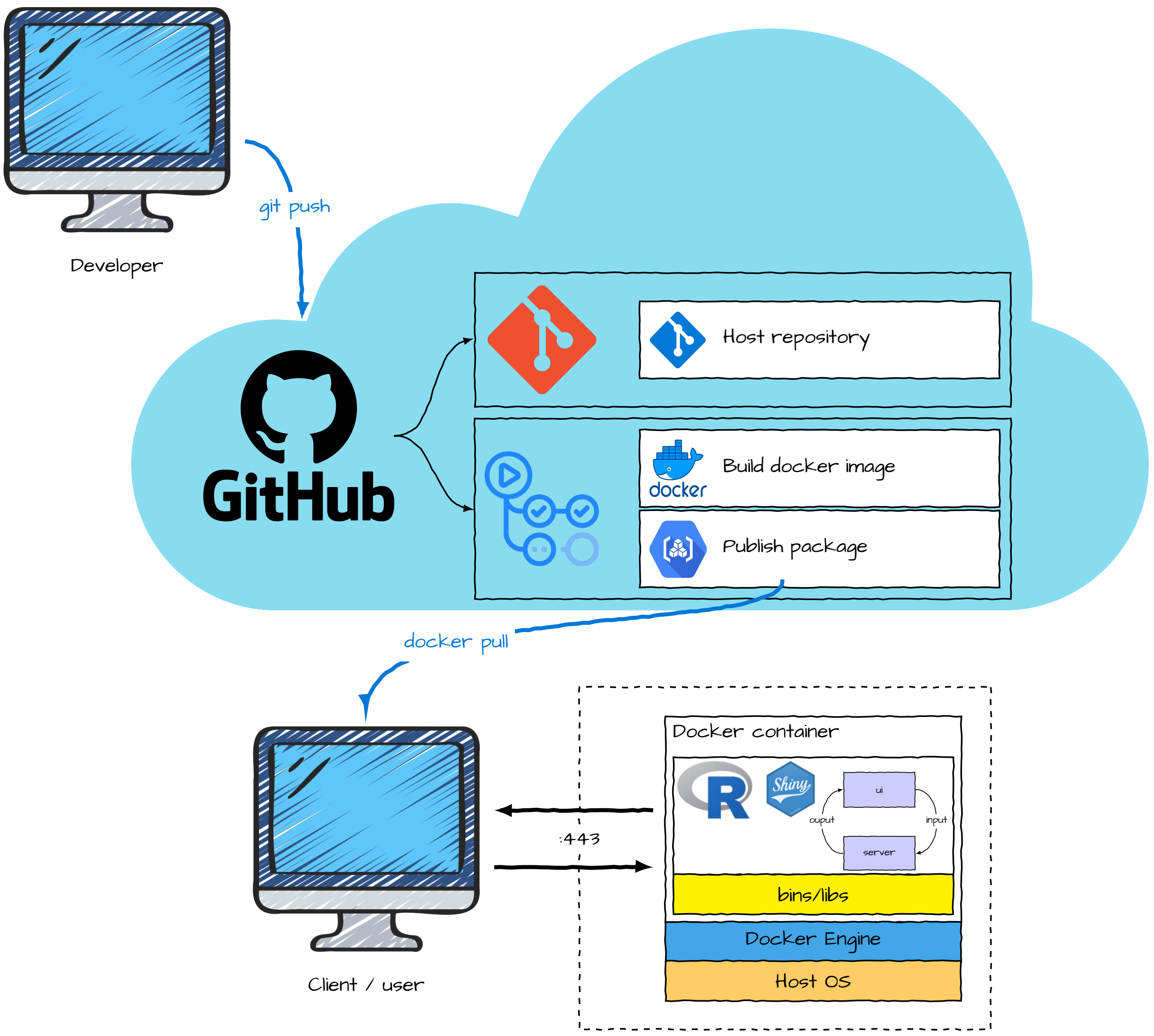


Figure 1: Diagram illustrating the relationship between the code produced by the developer and the shiny docker container utilised by user with a Github cloud conduit. The developed codebase includes a Shiny R application with R backend, `Dockerfile` (instructions used to assemble a full operating environment) and github workflow file (instructions for building and packaging the docker image on github via `actions`).

- **Start the Docker Engine:** once installed, click the "Start Docker Desktop" button. You may see a notification in the taskbar - click it to confirm and allow Docker to run in the background.
- **Verification:** open a terminal (or Powershell) and run `docker --version`. If all went well, you should see information about the installed Docker Engine version.

Additional Tips:

- Ensure Hyper-V (virtualization) is enabled in your BIOS settings for optimal performance.

Installing the and running the app

The task of installing and running the app is performed via a single **deploy script** (`deploy_wq.bat` on Windows or `deploy_wq.sh` on Linux/MacOSX/wsl). For this to work properly, the deploy script should be placed in a folder along with two additional folders (one called `input` and the other called `parameters`) that contains the input datasets (in csv format) and run time parameters. This structure is illustrated below for Windows.

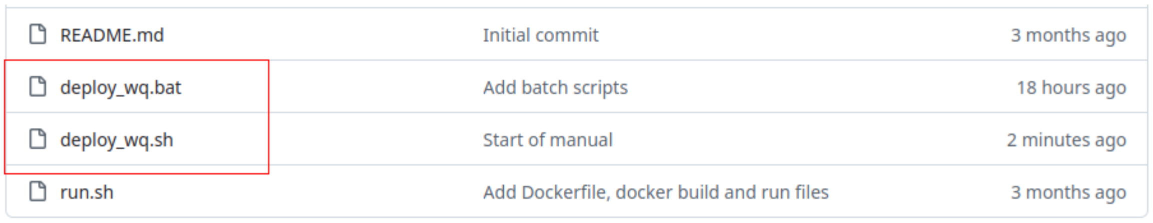
```
\
|- deploy_wq.bat
|- input
   |- 16_wq.csv
   |- 17_wq.csv
   |- overwrites.csv
   |- weights_m.csv
   |- weights_s.csv
|- parameters
   |- config.ini
   |- water_quality_guidelines.csv
   |- spatial.csv
   |- GIS
      |- RCZ_rev24.*
      |- SBZone_upper.*
      |- Middle_Harbour_Upper.*
```

Note

In the above illustration, there are two example water quality datasets (`16_wq.csv` and `17_wq.csv`). To ensure that the application correctly identifies them as water quality datasets, it is important that they are named according to the following format: `<yy>_wq.csv` where the `<yy>` represents a two digit year (e.g. 16 for 2016). For additional information on the contents of these files, please see **?@sec-data-requirements**.

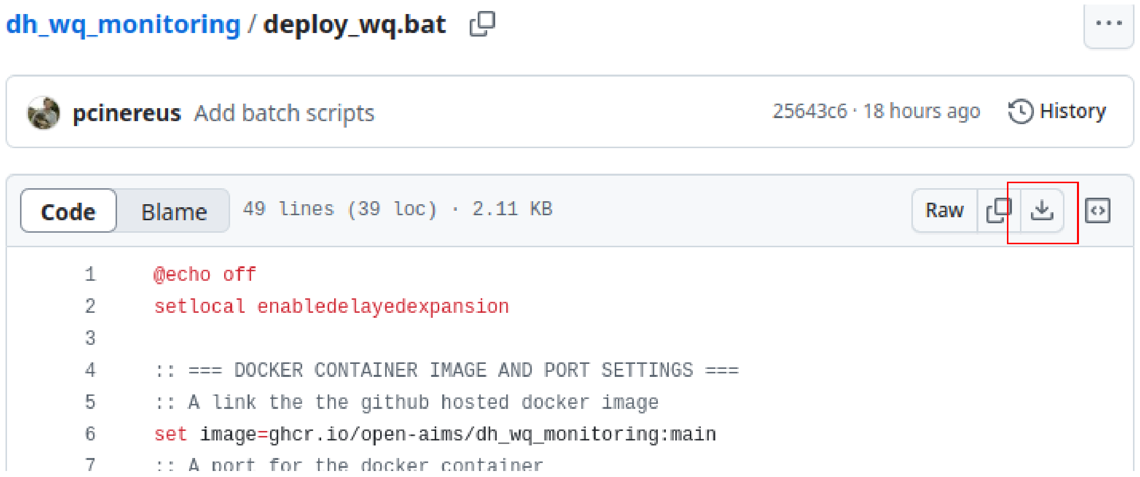
To set up the above structure:

1. create a new folder on your computer in a location of your choice that you are likely to remember and easily locate (e.g. on the desktop). Whilst the name of the folder is not important, it is recommended that it be named after the project (e.g. `darwin_harbour_sediment_monitoring`).
2. download the deploy script from the projects github repository
   a. go to the projects github repository (https://github.com/open-AIMS/dh_wq_monitoring.git) in a browser
   b. click on either the `deploy_wq.bat` (Windows) or `deploy_wq.sh` (Linux/MacOSX/wsl).



README.md	Initial commit	3 months ago
deploy_wq.bat	Add batch scripts	18 hours ago
deploy_wq.sh	Start of manual	2 minutes ago
run.sh	Add Dockerfile, docker build and run files	3 months ago

   c. click on the download button and select the project folder as the location to download the file to. If the file is automatically downloaded to a downloads folder, move the file to the project folder.


dh_wq_monitoring / deploy_wq.bat
pcinereus Add batch scripts
25643c6 · 18 hours ago
History
Code
Blame
49 lines (39 loc) · 2.11 KB
Raw

```
@echo off
setlocal enabledelayedexpansion

:: === DOCKER CONTAINER IMAGE AND PORT SETTINGS ===
:: A link the the github hosted docker image
set image=ghcr.io/open-aims/dh_wq_monitoring:main
:: A port for the docker container
```

3. within the project folder, create folders called `inputs` and `parameters` (as outlined above) and place all the appropriate data sets into these folders

To run the app, navigate inside of the project folder and run (typically double click) on the deploy script. Upon doing so, you will be presented with a directory selection window that is prompting for the path of the project folder. Navigate to and select the project folder before clicking the "OK" button. Shortly thereafter, the application will appear in a browser tab.

More specific information about the `deploy_wq.bat` script

The `deploy_wq.bat` script performs the following:

1. defines paths to the project repository and local project folder
2. checks if `docker` is installed and available from the command line for the current user
3. checks if `docker` is running
4. query the user for the location of the project folder
5. determine whether there are any updates to the `docker` image and if so pull them down
6. run the `docker` container
7. open the shiny app in a browser

The Darwin Harbour Water Quality Monitoring Program Analysis App

This Shiny application is designed to ingest very specifically structured water quality datasets containing Darwin Harbour Water Quality monitoring data and produce various analyses and visualisations. The application is served from a docker container to the localhost and the default web browser.

Docker containers can be thought of a computers running within other computers. More specifically, a container runs an instance of image built using a series of specific instructions that govern the entire software environment. As a result, containers run from the same image will operate (virtually) identically regardless of the host environment. Furthermore, since the build instructions can specify exact versions of all software components, containers provide a way of maximising the chances that an application will continue to run as designed into the future despite changes to operating environments and dependencies.

This shiny application comprises five pages (each accessable via the sidebar menu on the left side of the screen):

1. a **Landing** page (this page) providing access to the settings and overall initial instructions
2. a **Dashboard** providing information about the progression of tasks in the analysis pipeline
3. a **Data** page providing overviews of data in various stages
4. a **QAQC** page providing graphical QAQC outputs
5. a **Summaries** page providing summaries of the bootstrap aggregation of indices

6. a **Manual** page that displays the online manual for the application

Each page will also contain instructions to help guide you through using or interpreting the information. In some cases, this will take the from of an info box (such as the current box). In other cases, it will take the form of little symbols whose content is revealed with a mouse hover.

There are numerous stages throughout the analysis pipeline that may require user review (for example examining any data validation issues as well as the QAQC figures to confirm that the data are as expected). Consequently, it is advisable for the user to manually trigger each successive stage of the pipeline. The stages are:

- Stage 1 - Prepare environment

  More info

  This stage is run automatically on startup and essentially sets up the operating environment.

  - load any R package dependencies
  - get runtime settings from `../params/config.ini`. These include:
    - `focal_year`: usually the final year of sampling, all artifacts (data/graphics) will be stored in a folder reflecting this year
    - `method`: the index method to apply when calculating indices
    - `foldcap`: the folding cap to apply when calculating indices
    - `tuning`: the tuning to apply when calculating indices
    - `size`: the number of bootstrapp samples
    - `seed`: the random seed to apply to bootstrapping

- Stage 2 - Obtain data

  More info

  This stage comprises of the following steps:

  - read in the water quality guidelines from `../parameters/water_quality_guidelines.csv`.
  - read in each of the water quality data files from `../input/`. These files are in the format of `<number>_wq.csv`, where `<number>` is a two digit number representation of the sampling year.
  - read in each of the overwrites file from `../input/overwrites.csv`.
  - read in each of the measures weights file from `../input/weights_m.csv`.
  - read in each of the spatial weights file from `../input/weights_s.csv`.
  - read in the aggregation hierarchy file from `../input/hierarchy.csv`.
  - read in the spatial settings file from `../parameters/spatial.csv`.
  - validating each of the sources of input data according to a set of validation rules

  The tables within the **Raw data** tab of the **Data** page will also be populated (but wont be available for review until after the data have been processed in Stage 3).

- Stage 3 - Prepare spatial data

  More info

  This stage comprises of the following steps:

  - read in individual shapefiles from `../parameters/GIS`. The files are:
    - `RCZ_rev24.shp`
    - `SBZone_upper.shp`
    - `Middle_Harbour_Upper.shp`
  - combine all shapefiles into a single shapefile

  The tables within the **Processed data** tab of the **Data** page will also be populated.

- Stage 4 - Process data

  More info

  This stage comprises of the following steps:

  - combine all the water quality data into a single data set
  - process the dates from strings into genuine date objects
  - filter data to the bounds either defined in `../parameters/config.ini` or the data

  - select only measures for which there are guideline values
  - if the `focal_year` is undefined, define it based on the maximum date
  - pivot the data into a longer format that is more suitable for analysis and graphing
  - join in the guidelines information
  - use the spatial information in the shapefiles to assign spatial domains such as Regions and Zones.
  - apply any unit conversions to the values
  - apply limit of detection rules (to Dissolved Oxygen)
  - join in the aggregation hierarchy

  The tables within the **Processed data** tab of the **Data** page will also be populated and the `Data` page will be available for review.

- Stage 5 - Calculate indices

  More info

  This stage comprises of the following steps:

  - retrieve the processed data.
  - calculate the indices
  - prepare for bootstrapping

- Stage 6 - QAQC

  More info

  This stage comprises of the following steps:

  - retrieve the processed data.
  - construct outlier plots
  - contruct an LOR table
  - contruct boxplots for each Measure for the Focal Year for each Zone
  - construct timeseries boxplots for each Measure/Zone
  - construct boxplots for each Measure for the Focal Year conditional on Zone

  The QAQC figures of the **QAQC** page will also be populated.

- Stage 7 - Bootstrapping

  More info

  This stage comprises of the following steps:

  - generate bootsrapping schematic diagram
  - retrieve the processed data
  - retrieve the indices
  - process the overwrites
  - process the weights
  - aggregate to Zone/Measure/Source level
  - aggregate to Zone/Measure level
  - aggregate to Zone/Subindicator level
  - aggregate to Zone/Indicator level
  - aggregate to Region/Measure level
  - aggregate to Region/Subindicator level
  - aggregate to Region/Indicator level
  - aggregate to WH/Measure level
  - aggregate to WH/Subindicator level
  - aggregate to WH/Indicator level

- Stage 8 - Summaries

  More info

  This stage comprises of the following steps:

  - retrieve the processed data
  - compile all the indice scores
  - generate Zone/Measure/Source level

- collate Zone/Measure level scores
- collate Zone/Subindicator level scores
- collate Zone/Indicator level scores
- collate Region/Measure level scores
- collate Region/Subindicator level scores
- collate Region/Indicator level scores
- collate WH/Measure level scores
- collate WH/Subindicator level scores
- collate WH/Indicator level scores
- generate trend plots
- calculate effects (between years)
- generate effects plots

The trend and effects figures of the **Summaries** page will also be populated.

Underneath the sidebar menu there are a series of buttons that control progression through the analysis pipeline stages. When a button is blue (and has a play icon), it indicates that the Stage is the next Stage to be run in the pipeline. Once a stage has run, the button will turn green. Grey buttons are disabled.

Clicking on button will run that stage (or stages in some cases). While the stage is in progress, a popup will be displayed over the buttons. This popup serves two purposes. Firstly, some tasks within some stages are computationally intense and thus take some time to perform. For such tasks, a progress bar will be displayed in the popup to inform you of the progress through this task. Secondly, as some stages/tasks are slow, it provides visual feedback about when a stage has truly started and completed and prevents the temptation to repeatedly click on a button when nothing appears to be happening.

Once a stage is complete, the button will change to either green (success), yellow (orange) or red (failures) indicating whether errors/warnings were encountered or not. If the stage was completed successfully, the button corresponding to the next available stage will be activated.

Sidebar menu items that are in orange font are active and clicking on an active menu item will reveal an associated page. Inactive menu items are in grey font. Menu items will only become active once the appropriate run stage has been met. The following table lists the events that activate a menu item.

Menu Item	Trigger Event
Landing	Always active
Dashboard	Always active
Data	After Stage 4
QAQC	After Stage 6
Summaries	After Stage 8
Manual	Always active

Note, it is also possible to make all menu and buttons active using the **Run in sequence** toggle, however this should only be used if the full sequence of stages has already been run and you are returning to the analyses in a later session

Figure 2 provides a schematic overview of the sequence of filesystem events that occur during the development, deployment and running of this app.

1. the developed codebase is pushed to github and if necessary continuous integration (github actions) is triggered. The continuous integration will re-build and host a docker image as well as rebuild the manual.
2. when the client runs the `deploy_wq.bat` (or `deploy_wq.sh`) script, it will check whether docker is running and get input from the user about the location of the project directory.
3. github will be queried to discover if a new docker image is available. If so, then the new image will be pulled down locally and run (if docker is runnning).
4. the docker container will be run and this will trigger git within the container to pull down the latest version of the codebase from github to a temporary repo in the container. As the container is starting up, it will mount the project folder so that its contents are available to the environment within container and outputs produced within the container are available to the host.
5. some of the files in the temporary repo will be copied to a folder within the project folder.
6. the shiny app will start up on `port 3838` of the localhost and this will be offered to the default browser.

7. as the shiny app progresses through each of the analysis stages, more data will be added to various folders of the project directory.

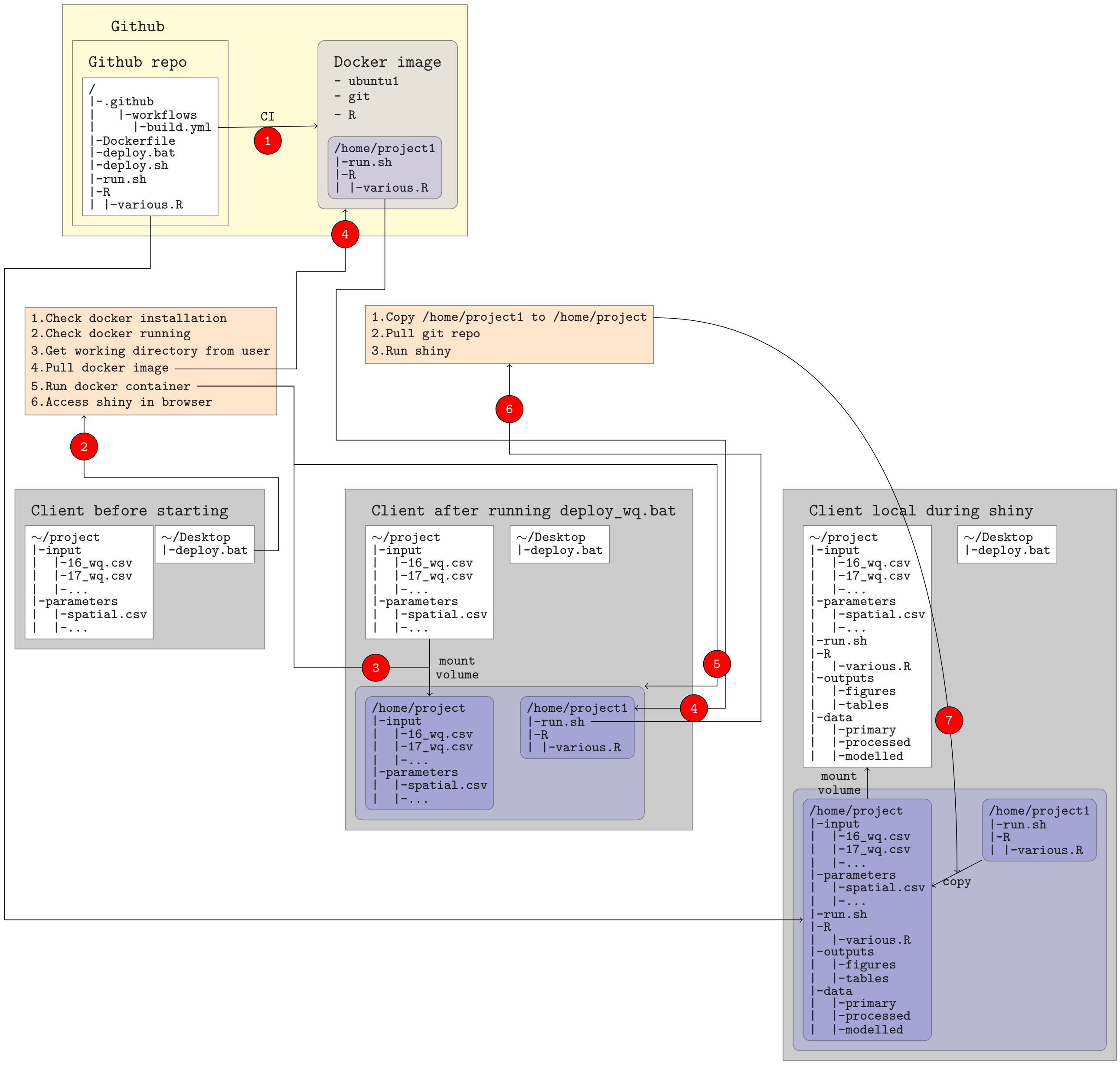


Figure 2: Diagram illustrating the sequence of filesystem events that occur during the development, deployment and running of this app.