WC$_{e}nv_{m}anagerdocumentation$

Release 0.0.1

Karr Lab

Mar 20, 2020

Contents

wc_env_manager helps modelers and software developers setup customizable computing environments for developing, testing, and running whole-cell (WC) models and WC modeling software. This makes it easy for modelers and software developers to install and configure the numerous dependencies required for WC modeling. This helps modelers and software developers focus on developing WC models and software tools, rather than on installing, configuring, and maintaining complicated dependencies.

In addition, *wc_env_manager* facilitates collaboration by helping WC modelers and software developers share a common base computing environment with third party dependencies. Furthermore, *wc_env_manager* helps software developers anticipate and debug issues in deployment by enabling developers to replicate similar environments to those used to test and deploy WC models and tools in systems such as Amazon EC2, CircleCI, and Heroku.

wc_env_manager uses Docker to setup a customizable computing environment that contains all of the software packages needed to run WC models and WC modeling software. This includes

- Required non-Python packages
- Required Python packages from PyPI and other sources
- WC models and WC modeling tools
- Optionally, local packages on the user's machine such as clones of these WC models and WC modeling tools

wc_env_manager supports both the development and deployment of WC models and WC modeling tools:

- **Development:** *wc_env_manager* can run WC models and WC modeling software that is located on the user's machine. This is useful for testing WC models and WC modeling software before committing it to GitHub.
- **Deployment:** *wc_env_manager* can run WC models and WC modeling software from external sources such as GitHub.

CHAPTER 1

Contents

1.1 Installation

1.1.1 Requirements

First, install the following requirements. Detailed installation instructions are available in An Introduction to Whole-Cell Modeling.

- git
- Docker
- Docker Compose
- Pip >= 19.0
- Python >= 3.5

1.1.2 Installing the latest revision from GitHub

Second, run the following command to install the latest version of *wc_env_manager* from GitHub:

```
pip install git+https://github.com/KarrLab/wc_env_manager.git#egg=wc_env_manager
```

1.2 Overview

1.2.1 Features

wc_env_manager provides a high-level interface for the following modeling tasks

- Build WC modeling computing environments (Docker images and containers)

1. Copy files (such as configuration files and authentication keys) into image
2. Install GitHub SSH key into image
3. Install WC models and WC modeling tools from GitHub into image
4. Mount host directories (e.g. with clones of WC models and WC modeling tools) into container
5. Install Python packages in mounted directories (e.g. clones of WC models and WC modeling tools) from host

- Copy files to/from containers
- List containers of the images
- Get CPU, memory, network usage statistics of containers
- Login to DockerHub
- Push and pull images to/from DockerHub
- Remove images and containers

1.2.2 How *wc_env_manager* works

wc_env_manager is based on Docker images and containers which enable virtual environments within all major operating systems including Linux, Mac OSX, and Windows, and the DockerHub repository for versioning and sharing virtual environments.

1. *wc_env_manager* creates a Docker image, *wc_env_dependencies* with the third-party dependencies needed for WC modeling or pulls this image from DockerHub. This image represents an Ubuntu Linux machine.
2. *wc_env_manager* uses this Docker image to create another Docker image, *wc_env* with the WC models, WC modeling tools, and the configuration files and authentication keys needed for WC modeling.
3. *wc_env_manager* uses this image to create a Docker container to run WC models and WC modeling tools. Optionally, the container can have volumes mounted from the host to run code on the host inside the Docker container, which is helpful for using the container to test and debug WC models and tools.

The images and containers created by *wc_env_manager* can be customized using a configuration file.

1.2.3 Caveats and troubleshooting

- Code run in containers created by *wc_env_manager* can create host files and overwrite existing host files. This is because *wc_env_manager* mounts host directories into containers.
- Containers created by *wc_env_manager* can be used to run code located on your host machine. However, using different versions of Python between your host and the Docker containers can create Python caches and compiled Python files that are incompatible between your host and the Docker containers. Before switching between running code on your host your and the Docker containers, you may need to remove all `__pycache__` subdirectories and `*.pyc` files from host packages mounted into the containers.
- Code run in Docker containers will not have access to the absolute paths of your host and vice-versa. Consequently, arguments that represent absolute host paths or which contain absolute host paths must be mapped from absolute host paths to the equivalent container path. Similarly, outputs which represent or contain absolute container paths must be mapped to the equivalent host paths.
- Running code in containers created with *wc_env_manager* will be slower than running the same code on your host. This is because *wc_env_manager* is based on Docker containers, which add an additional layer of abstraction between your code and your processor.

1.3 Tutorial for users of WC models and WC modeling tools

Users of WC models and WC modeling tools should follow these steps to use *wc_env_manager* to use WC models and WC modeling tools

1. Use *wc_env_manager* to pull existing computing environments for WC modeling (Docker images)
2. Use *wc_env_manager* to create Docker containers for WC modeling
3. Run models and tools inside the Docker containers created by *wc_env_manager*

1.3.1 Pulling existing Docker images

First, use the following command to pull existing WC modeling Docker images. This will pull both the base image with third part dependencies, *wc_env_dependencies*, and the image with WC models and modeling tools, *wc_env*.:

```
wc-env-manager pull
```

The following commands can also be used to pull the individual images.:

```
wc-env-manager base-image pull
wc-env-manager image pull
```

1.3.2 Building containers for WC modeling

Second, use the following command to use *wc_env* to construct a network of Docker containers.:

```
wc-env-manager container build
```

This will print out the id of the WC container that was built. This is the main container that you should use to run WC models and WC modeling tools.

1.3.3 Using containers to run WC models and WC modeling tools

Third, use the following command to execute the container. This launches the container and runs an interactive *bash* shell inside the container.:

```
docker exec --interactive --tty <container_id> bash
```

Fourth, use the integrated WC modeling command line program, *wc_cli*, to run WC models and WC modeling tools. For example, the following command illustrates how to get help for the *wc_cli* program. See the *wc_cli* documentation for more information.:

```
container >> wc-cli --help
```

1.3.4 Using WC modeling computing environments with an external IDE such as PyCharm

The Docker images created with *wc_env_manager* can be used with external integrated development environments (IDEs) such as PyCharm. See the links below for instructions on how to use these tools with Docker images created with *wc_env_manager*.

- Jupyter Notebook
- PyCharm Professional Edition
- Other IDEs:
  1. Install the IDE in a Docker image
  2. Use X11 forwarding to render graphical output from a Docker container to your host. See Using GUI's with Docker for more information.

1.3.5 Exiting and removing containers

Next, exit the container by executing *exit* or typing control-d. The container can be restarted using the following commands:

```
docker restart <container_id>
docker exec --interactive --tty <container_id> bash
```

Finally, remove the container by executing the following command:

```
wc-env-manager container remove
```

1.4 Tutorial for developers of WC models and WC modeling tools

Developers should follow these steps to build and use WC modeling computing environments (Docker images and containers) to test, debug, and run WC models and WC modeling tools.

1. Use *wc_env_manager* to pull existing WC modeling Docker images
2. Use *wc_env_manager* to create Docker containers with volumes mounted from the host and installations of software packages contained on the house
3. Run models and tools inside the Docker containers created by *wc_env_manager*

1.4.1 Pulling existing Docker images

First, use the following command to pull existing WC modeling Docker images. This will pull both the base image with third part dependencies, *wc_env_dependencies*, and the image with WC models and modeling tools, *wc_env*.:

```
wc-env-manager pull
```

The following commands can also be used to pull the individual images.:

```
wc-env-manager base-image pull
wc-env-manager image pull
```

1.4.2 Building containers for WC modeling

Second, set the configuration for the containers created by *wc_env_manager* by creating a configuration file *./wc_env_manager.cfg* following the schema outlined in */path/to/wc_env_manager/wc_env_manager/config/core.schema.cfg* and the defaults in */path/to/wc_env_manager/wc_env_manager/config/core.default.cfg*.

- Configure additional Docker containers that should be run and linked to the main container. For example, the configuration below will generate a second container based on the `postgres:10.5-alpine` image with the host name `postgres_hostname` on the `wc_network` Docker network and the environment variable `POSTGRES_USER` set to `postgres_user`. The main Docker image will also be added to the same `wc_network` Docker network, which will make the second image accessible to the main image with the host name `postgres_hostname`. In this example, it will then be possible to login to the Postgres service from the main container with the command `psql -h postgres_hostname -U postgres_user <DB>`.

  [wc_env_manager]

  [[network]] name = wc_network [[[containers]]]

  [[[[postgres_hostname]]]] image = postgres:10.5-alpine [[[[[environment]]]]]

  POSTGRES_USER = postgres_user

- Configure environment variables that should be set in the Docker container. The following example illustrates how to set the `PYTHONPATH` environment variable to the paths to *wc_lang* and *wc_sim*. Note, we recommend using *pip* to manipulate the Python path rather than directly manipulating the `PYTHONPATH` environment variable. We only recommend manipulating the `PYTHONPATH` environment variable for packages that don't have `setup.py` scripts or for packages that `setup.py` scripts that you temporarily don't want to run.:

```
[wc_env_manager]
    [[container]]
        [[[environment]]]
            PYTHONPATH = '/root/host/Documents/wc_lang:/root/host/Documents/wc_
↪utils'
```

- Configure the host paths that should be mounted into the containers. Typically, this should including mounting the parent directory of your Git repositories into the container. For example, this configuration will map (a) the Documents directory of your host (*${HOME}/Documents*) to the */root/host/Documents* directory of the container and (b) your the WC modeling configuration directory of your host (*${HOME}/.wc*) to the WC modeling configuration directory of the container (*/root/.wc*). *${HOME}* will be substituted for the path to your home directory on your host.:

```
[wc_env_manager]
    [[container]]
        [[[paths_to_mount]]]
            [[[[${HOME}/Documents]]]]
                bind = /root/host/Documents
                mode = rw
            [[[[${Home}/.wc]]]]
                bind = /root/.wc
                mode = rw
```

- Configure the WC modeling packages that should be installed into *wc_env*. This should be specified in the *pip* requirements.txt format and should be specified in terms of paths within the container. The following example illustrates how to create editable installations of clones of *wc_lang* and *wc_utils* mounted from the host into the container.:

```
[wc_env_manager]
    [[container]]
        python_package = '''
            -e /root/host/Documents/wc_lang
            -e /root/host/Documents/wc_utils
            '''
```

- Configure additional command(s) that should be run when the main Docker container is created. These commands will be run within a bash shell. For example, this configuration could be used to create and import the

content of a database when the container is created.:

```
[wc_env_manager]
    [[container]]
        setup_script = '''
            create db
            restore db
            '''
```

- Configure the ports that should be exposed by the container. The following example illustrates how to expose port 8888 as 8888.:

```
[wc_env_manager]
    [[container]]
        [[[ports]]]
            8888 = 8888
```

- Configure all credentials required by the packages and tools used by the container. These should be installed in config (*.cfg) files that can be accessed by *wc-env-manager*. *~/.wc* is a standard location for whole-cell config files. Failure to install credentials will likely generate *Authentication error* exceptions. Docker images and containers may need to be cleaned up if *wc-env-manager* fails. See the *docker* command help for instructions.

Third, use the following command to use *wc_env* to construct a network of Docker containers.:

```
wc-env-manager container build
```

This will print out the id of the WC container that was built. This is the main container that you should use to run WC models and WC modeling tools.

1.4.3 Using containers to run WC models and WC modeling tools

Fourth, use the following command to execute the container. This launches the container and runs an interactive *bash* shell inside the container.:

```
docker exec --interactive --tty <container_id> bash
```

Fifth, use the integrated WC modeling command line program, *wc_cli*, to run WC models and WC modeling tools. For example, the following command illustrates how to get help for the *wc_cli* program. See the *wc_cli* documentation for more information.:

```
container >> wc-cli --help
```

1.4.4 Using containers to develop WC models and WC modeling tools

Sixth, use command line programs inside the container, such as *python*, *coverage* or *pytest*, to run WC models and tools. Note, only mounted host paths will be accessible in the container.

1.4.5 Using WC modeling computing environments with an external IDE such as PyCharm

The Docker images created with *wc_env_manager* can be used with external integrated development environments (IDEs) such as PyCharm. See the links below for instructions on how to use these tools with Docker images created with *wc_env_manager*.

- Jupyter Notebook
- PyCharm Professional Edition
- Other IDEs:
    1. Install the IDE in a Docker image
    2. Use X11 forwarding to render graphical output from a Docker container to your host. See Using GUI's with Docker for more information.

1.4.6 Exiting and removing containers

Next, exit the container by executing *exit* or typing control-d. The container can be restarted using the following commands:

```
docker restart <container_id>
docker exec --interactive --tty <container_id> bash
```

Finally, remove the container by executing the following command:

```
wc-env-manager container remove
```

1.5 Tutorial for administrators of the *wc_env* and *wc_env_dependencies* images

Administrators should follow these steps to build and disseminate the *wc_env* and *wc_env_dependencies* images.

1. Create contexts for building the *wc_env* and *wc_env_dependencies* Docker images.
2. Create Dockerfile templates for the *wc_env* and *wc_env_dependencies* Docker images.
3. Set the configuration for *wc_env_manager*.
4. Use *wc_env_manager* to build the *wc_env* and *wc_env_dependencies* Docker images.
5. Use *wc_env_manager* to push the *wc_env* and *wc_env_dependencies* Docker images to DockerHub.

1.5.1 Create contexts for building the *wc_env* and *wc_env_dependencies* images

First, create contexts for building the images. This can include licenses and installers for proprietary software packages.

1. Prepare CPLEX installation
    a. Download CPLEX installer from https://ibm.onthehub.com
    b. Save the installer to the base image context
    c. Set the execution bit for the installer by running *chmod ugo+x /path/to/installer*
2. Prepare Gurobi installation
    a. Create license at http://www.gurobi.com/downloads/licenses/license-center
    b. Copy the license to the *gurobi_license* build argument for the base image in the *wc_env_manager* configuration
3. Prepare Mosek installation

a. Request an academic license at https://license.mosek.com/academic/

b. Receive a license by email

c. Save the license to the context for the base image as *mosek.lic*

4. Prepare XPRESS installation

   a. Install the XPRESS license server on another machine

      i. Download XPRESS from https://clientarea.xpress.fico.com

      ii. Use the *xphostid* utility to get your host id

      iii. Use the host id to create a floating license at https://app.xpress.fico.com

      iv. Save the license file to the context for the base image as *xpauth.xpr*

      v. Run the installation program and follow the onscreen instructions

   b. Copy the IP address or hostname of the license server to the *xpress_license_server* build argument for the base image in the *wc_env_manager* configuration.

   c. Save the license file to the context for the base image as *xpauth.xpr*.

   d. Edit the server property in the first line of *xpauth.xpr* in the context for the base image. Set the property to the IP address or hostname of the license server.

1.5.2 Create Dockerfile templates for *wc_env* and *wc_env_dependencies*

Second, create templates for the Dockerfiles to be rendered by Jinja, and save the Dockerfiles within the contexts for the images. The default templates illustrate how to create the Dockerfile templates.

- */path/to/wc_env_manager/wc_env_manager/assets/base-image/Dockerfile.template*
- */path/to/wc_env_manager/wc_env_manager/assets/image/Dockerfile.template*

1.5.3 Set the configuration for *wc_env_manager*

Third, Set the configuration for *wc_env_manager* by creating a configuration file *./wc_env_manager.cfg* following the schema outlined in */path/to/wc_env_manager/wc_env_manager/config/core.schema.cfg* and the defaults in */path/to/wc_env_manager/wc_env_manager/config/core.default.cfg*.

- Set the repository and tags for *wc_env* and *wc_env_dependencies*.
- Set the paths for the Dockerfile templates.
- Set the contexts for building the Docker images and the files that should be copied into the images.
- Set the build arguments for building the Docker images. This can include licenses for proprietary software packages. For example,:

```
[wc_env_manager]
    [[base_image]]
        [[[build_args]]]
            gurobi_version = 8.0.1
            gurobi_license = ...
            ...
```

- Set the WC modeling packages that should be installed into *wc_env*. For example,:

```
[wc_env_manager]
    [[image]]
        python_packages = '''
            pytest
            pytest-cov
            '''
```

- Set your DockerHub username and password.

1.5.4 Build the *wc_env* and *wc_env_dependencies* Docker images

Use the following command to build the *wc_env* and *wc_env_dependencies* images:

```
wc-env-manager build
```

1.5.5 Push the *wc_env* and *wc_env_dependencies* Docker images to DockerHub

Use the following command to push the *wc_env* and *wc_env_dependencies* images to GitHub:

```
wc-env-manager push
```

1.6 About

1.6.1 License

The software is released under the MIT license

1.6.2 Development team

This package was developed by the Karr Lab at the Icahn School of Medicine at Mount Sinai in New York, USA.

1.6.3 Acknowledgements

1.6.4 Questions and comments

Please contact the Karr Lab with any questions or comments.