Tool Administrators

The procedures for maintaining and updating tool execution environments are presented in this section. Anaconda (Jupyter Notebook) changes require membership in the apps group. Container changes described in the Installing Tool Dependecies section requires login access to execution hosts and membership in the docker group. Tool Administrator responsibilities and procedures. CMS changes described in the White list section require access to the administrator website.

Installing Tool Dependencies

Occasionally additional software needs to be installed in a tool environment to support tool execution. Please follow these instructions in the given order. Keep in mind that an environment maybe shared by multiple tools.

Install a operating system package in the tool execution container

Tool execution containers are based on Docker images. At minimum Debian 7 (wheezy) and Debian 10 (buster) Docker images are provided. Operating system choice is not limited to Debian. We have used CentOS and Ubuntu images as the basis for images to meet tool requirements. Regardless of operating system choice there is a minimum set of scripts and configuration files that must be included to enable the middleware to manage container related processes.

The pairing of Docker image and tool is made according to the following convention:

1. Find an image tag matching tool name and revision.
2. Find an image tag matching the tool name.
3. A default image specified in the middleware.

Multiple Docker tags may be applied to the same image thus reducing the storage requirement for multiple images. Docker images may be modified or created using standard practices based on Dockerfiles. If all Docker images are not made available on all execution hosts, host requirements may be used to aid in execution host selection.

Manually install software in the 'use' infrastructure

Typically we create and run Hapi (HUBzero Apps Program Installers) scripts to download, configure, compile, and install software in the tool execution environment. This is especially important when multiple versions of the same software must be available to server different tools. For example, Tool A may require R version 3.6.3 while Tool B may require R version 4.2.1. These requirements are the result of an ever evolving software landscape. "Use" is very helpful in this case, among others, providing a way to load specific software versions as requirements demand.

All operations for manually installing dependent software for tools must be done by the apps user from a workspace terminal.

As the apps user clone the Hapi repo into the apps home directory:

```
git clone https://github.com/hubzero/hapi.git
```

In the hapi/scripts directory you will find a collection of shell scripts and csv files for software typically installed for use in tool environments. If a script exists for software that you need, just run it! Feel free to add new Hapi scripts of your creation to the GitHub repo by submitting pull requests. We occasionally add scripts as well.

If, after doing a git update on your repository, a script doesn't exist for the software you need, copy an existing Hapi script and modify it. Hapi scripts make your life much easier by downloading, configuring, compiling, installing and even adding the required 'use' environ.d file to the appropriate location.

All tool dependencies are installed by the apps user in an operating system specific directory such as /apps/share64/debian7 and should be owned by the apps user and group. All files must be readable by everyone and all directories must be searchable by everyone. No files or directories should be writable by everyone (seriously don't do this)

* For a full manual run "man use" from a tool session terminal.

USE(1) User Commands USE(1)

NAME

use, unuse - adjust the shell environment

SYNOPSIS

use [options]... [ENVIRONMENT]
unuse [options]... [ENVIRONMENT]

DESCRIPTION

The use command incorporates the specified ENVIRONMENT to the current shell. The unuse command removes it. It optionally records the selection persistently so that subsequent shells will use the ENVIRONMENT. These commands are independent of the shell being run.

An ENVIRONMENT is specified by a configuration file of the same name as found in one of the configuration directories. The ENVIRON_CONFIG_DIRS environment variable specifies a list of directories in which to search for configurations. Each configured ENVIRONMENT specifies a environment variables to set or prepend, shell variables to set, and shell aliases to set.

Some environments are configured to conflict with others. The use command will ask if conflicting ENVIRONMENT should be replaced.

With no arguments, the use and unuse commands will print a synopsys of options and lists all available environments.

-h print available help for a named ENVIRONMENT.

-e environment only. Do not ask about preserving the selection.

-p modify the environment and preserve selection. Do not ask about preserving the selection.

-k keep any conflicting environment. Do not ask about replacing it.

-r replace any conflicting environment without asking.

-x quietly ignore the command if the named ENVIRONMENT cannot be found.

MAKING IT WORK

The following command will describe an environment named xyz:

use -h xyz

The following command will incorporate the xyz environment preserving the environment for future shell invocations. It will also not override any conflicting environments:

use -p -k xyz

The following command will remove the xyz environment but retain its use for future sessions:

unuse -e xyz

INTERNAL OPERATION

use and unuse are actually implemented as shell functions (or as aliases in the case of csh derivatives). The functions pass their arguments to the /etc/environ script which determines the commands that the shell should execute to satisfy the new environment configuration. The script prints these commands, the shell function receives them and evals them.

ENVIRONMENT CONFIGURATION FILES

Configuration files are interpreted shell scripts. Several predefined functions are available to make the the process automatic.

alias NAME "Replacement"

Set a command alias in the shell.

conflict VARNAME
: Define an environment variable to indicate that a type of an ENVIRONMENT is in use. All conflicting ENVIRONMENT configurations should specify the same conflict. An ENVIRONMENT configuration may specify multiple conflicts.

desc "A short description..."
: A short description of the ENVIRONMENT.

help "A lengthy description..."
: A long description of the ENVIRONMENT and how to use it. This description will be formatted when printed.

prepend VARNAME ADDITION
: Prepend ADDITION to the environment variable VARNAME separated with a colon.

setenv VARNAME REPLACEMENT
: Set or replace the environment variable VARNAME with REPLACEMENT.

shellset VARNAME REPLACEMENT
: Set or replace the shell variable VARNAME with REPLACEMENT.

Jupyter Notebooks

Adding additional packages to Jupyter Notebooks

Google the desired package (python, R packages, not OS packages) and review the installation instructions. They might recommend a different conda repository than the default.
Be careful if conda says it wants to downgrade packages. If it is a minor downgrade, it is probably OK. Do not proceed if many packages must be downgraded or critical packages are to be downgraded. Remember that changes to the Ananconda environment will affect all tools using the environment and a downgrade could cause tools not to function (if a feature is no longer available, for example).

- Start a workspace tool. If your HUB supports multiple operating system (vendor or version) based tool containers choose the workspace tool for the appropriate container.
- As an alternative, start the Jupyter Notebook tool based on the Anaconda environment that you want to modify and start a new terminal.
- From the open terminal switch to the apps user (your account must be a member of the apps group).
    - `sudo su - apps`
- Load the Anaconda environment that you wish to modify (there may be multiple Anaconda environments available. The list of Anaconda environments can be listed with the command
    - `use |& grep anaconda`
- Execute the command "use", choosing the appropriate Anaconda (X represents the particular version choice) environment.
    - `use -e -r anaconda-X`
- Install the desired package via conda or mamba if it is installed. mamba is typically much more efficient and provides more useful information should it fail. Another option is to use pip. Extra care must be taken when mixing conda and pip package installations in the same environment. Typical installation command like
    - `conda install -c conda-forge -c defaults <pkgname>`
    - `pip install -U --upgrade-strategy only-if-needed <pkgname>`

The installation may take a few minutes

- **Important! Fix any world writable files by doing:**
    - `chmod -R o-w /apps/share64/<OS>/anaconda/anaconda-X`

Creating a separate anaconda environment.

In some circumstances it is beneficial to spawn a new named environment from the standard base environment. The named environments are referred to as kernels in Jupyter notebooks. Once a Jupyter notebook is running the developer or user has the opportunity to change to a different kernel. The kernel used to run the notebook is embedded within the notebook metadata therefore only has to be changed once. This works well for developers and means the administrator does not have to create a Jupyter Notebook tool for each kernel.

- Set base Anaconda environment
    - `use -e -r anaconda-X`
- Create new named environment
    - `conda create -n <name>`
- Activate named environment
    - `source activate <name>`
- Do package installation as before
    - `conda install -c conda-forge -c defaults <pkgname>`
    - `pip install -U --upgrade-strategy only-if-needed <pkgname>`
- Register named environment as kernel in base environment
    - `python -m ipykernel install --sys-prefix --name <name> --display-name "Python3 (<name>)"`
    - `python -m ipykernel install --prefix /apps/share64/<OS>/anaconda/anaconda-X --name <name> --display-`

```
name "Python3 (<name>)"
```

- Deactivate named environment
    - `conda deactivate`

References:

conda user guide

Ipython kernel instalation

Updating hublib

The HUBzero utility library should be installed in every base Anaconda environment and in each named environment as well. The hublib package is available only through pip.

- Set Anaconda base environment
    - `use -e -r anaconda-X`
- Install hublib
    - `pip install -U hublib`

White list of directories through the CMS

The parameter passing tool execution feature provides a mechanism for passing file names to a tool through the URL used to launch the tool session. As a security measure the passed parameters must conform to a simple schema and only files in white listed directories may be referenced. The passed parameters are validated by the middleware. The files passed as parameters are restricted to the directories listed in the **Directory Parameter Whitelist.** If specified files lie outside of the white listed locations the the tool launch will fail.

Any super-user can whitelist a directory via the /administrator interface. Follow these steps to complete this request:

1. Navigate to the **/administrator** interface and login
2. Hover over **Components** then click on **Tools**
3. Click the **Options** button and under **Directory Parameter Whitelist**, modify the comma separated list of directories. At minimum the list should include /home.
4. Click **Save & Close**
5. Make sure that you complete this task on all the requested systems (i.e. production, stage, dev, qa, etc.)