11falsenonelisttrue

storageN

The use of `storageN` within these documents indicates that any storage platform can be used.

Current available storage platforms:

- storage1
- storage2

Features of the `docker` application (a.k.a. the docker wrapper) are accessed by setting environment variables before executing `bsub`. All the examples below show how to set the values inline with submitting the job, and multiples of these settings are given by including all of them on the same command-line:

Or they can also be set within the shell by "exporting" them, essentially making them apply to every command that follows:

LSF_DOCKER_ADD_HOST

Adds one more more host -> IPaddress mappings to the docker container. The value of `LSF_DOCKER_ADD_HOST` is a space-separated list of mappings in the same format as the `--add-host` argument of `docker run`, ie. `hostname:IPaddress`

LSF_DOCKER_ADD_HOST_FILE

Specify one more more space-separated files containing host -> IPaddress mappings. These files should have one mapping per line in the same format used by `LSF_DOCKER_ADD_HOST`:

These file names are then included in the `LSF_DOCKER_ADD_HOST_FILE` variable:

LSF_DOCKER_CGROUP

Set the value for the `--cgroup-parent` argument to `docker run`. There is no mechanism for creating custom cgroups at this time, so this should not be used unless specifically directed to by RIS.

LSF_DOCKER_ENTRYPOINT

Overrides the container's default `ENTRYPOINT`, and becomes the `--entrypoint` argument to `docker run`

LSF_DOCKER_ENV_FILE

Used to set envrionment variables within the running container. Expects a space-separated list of file names containing environment variables. The files should have one variable per line, and be formatted as `variable=value`:

To make use of these files, set the `LSF_DOCKER_ENV_FILE` and submit the job:

The file names are then used as `--env-file` arguments to `docker run`.

LSF_DOCKER_IPC

Control the `--ipc` argument to `docker run`. This will become useful when building *MPI* applications.

LSF_DOCKER_NETWORK

Make use of docker's `--network` argument. Most notablly for `--network=host`. See the Docker networking tutorial.

LSF_DOCKER_PORTS

Specify network ports to expose from your docker container. Note that these must come from a range of "approved ports" (8000-8999) and should be "reserved" by the job scheduler. Here we map port 80 inside the container to port 8001 on the execution node, and we "reserve" port 8001 on that execution node through the scheduler.

The `LSF_DOCKER_PORTS` variable works with most expressions accepted by the `-p` option of `docker run`:

- `8001`: Expose a single port
- `8001:80`: Expose port 8001 and forward it to port 80 within the container

`8001-8010`: Expose a range of ports

`8001-8010:2001-2010`: Expose a range of forwarded ports

`8001/tcp`: Expose only TCP port 8001

`8001-8010/udp`: Expose a range of UDP ports

Additionally, more than one port may be exposed by separating each request with a space. For example:

See also Exposing Ports from Within Containers.

Note: LSF_DOCKER_PORTS is affected by the LSF_DOCKER_NETWORK type, and does not work with "host" type networking.

LSF_DOCKER_PRESERVE_ENVIRONMENT

This is a boolean value instructing the job launcher to either preserve your shell environment or not. Valid values are the strings `true` and `false`. The default value is `true`.

When false, the container will have a minimal environment with only a few variables, including `HOSTNAME` set to the exec host the job runs on, and those variables defined in the container's Dockerfile with `ENV`.

When true, the container will inherit most environment variables set when the job is submitted, except for a few:

HOSTNAME - becomes the exec host's hostname

LSB_INTERACTIVE - removed within an interactive job so jobs started within that job are not interactive by default

values containing newlines - these variables are removed due to a bug in docker's handling of environment variables

LSF_DOCKER_RUN_LOGLEVEL

Set the log level for the wrapper script:

Valid values are, in descending order of verbosity: CRITICAL, ERROR, WARNING, INFO, DEBUG

LSF_DOCKER_SHM_SIZE

Control the `--shm-size` argument to `docker run`. This will become useful when building *MPI* applications.

LSF_DOCKER_USE_LUCID_AUTH

Getting authentication through libnss-sss working for containers based on Ubuntu Lucid Lynx (10.04) requires extra files and devices from the host system to be mounted in the container. This option should not be used unless you are having trouble getting authentication working on a container based on Ubuntu Lucid.

LSF_DOCKER_WORKDIR

The default initial working directory within the container is the same as the working directory when the job was submitted. This can be overridden with `LSF_DOCKER_WORKDIR` and becomes the `-w` option to `docker run`, and must be a directory accessible within the container.

LSF_DOCKER_VOLUMES

This is a space separated list of filesystem locations to pass into your docker container by means of the `--mount` flag. The format is `src:dst`, where `src` means the filesystem location outside the container should be mapped to `dst` inside the container. (See also `--mount` at Docker Volumes.)

Elements in LSF_DOCKER_VOLUMES must be *directories*. Docker would allow you to pass in files, such that the following might be expected to work:

LSF_DOCKER_VOLUMES="$HOME/etc/myfile:/etc/somefile"

But the RIS environment explicitly prevents this due to a security vulnerability.

A volume can be mounted read-only by appending `:ro` to any of the mounts, for example: `src:dst:ro`.

LSF_DOCKER_RUN_RAW_ARGS

This variable tells bsub to use the command or entrypoint within the Docker image instead of requesting the command or entrypoint from the user as is the regular interaction without this variable.

This means that Docker images like the basic hello-world Docker image can be run on the Compute Platform.

The command also makes use of `LSB_DOCKER_PLACE_HOLDER` which is, as the name suggests, just a placeholder as bsub still needs an input to run. This is NOT passed as a command to the Docker container.

Example Command:

Output:

is submitted to queue . <> <> Using default tag: latest latest: Pulling from library/hello-world Digest: sha256:ffb13da98453e0f04d33a6eee5bb8e46ee50d08ebe17735fc0779d0349e889e9 Status: Image is up to date for hello-world:latest docker.io/library/hello-world:latest Hello from Docker! This message shows that your installation appears to be working correctly. To generate this message, Docker took the following steps: 1. The Docker client contacted the Docker daemon. 2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64) 3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading. 4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal. To try something more ambitious, you can run an Ubuntu container with: $ docker run -it ubuntu bash Share images, automate workflows, and more with a free Docker ID: https://hub.docker.com/ For more examples and ideas, visit: https://docs.docker.com/get-started/]]>