Moab Docker Integration

Reference Guide for Moab 9.0.2 and Torque 6.0.2

August 2016

Adaptive Computing Enterprises, Inc.
1712 S. East Bay Blvd., Suite 300
Provo, UT 84606
+1 (801) 717-3700
www.adaptivecomputing.com

Scan to open online help

Contents

Welcome

This documentation supports Adaptive Computing's Docker Integration package. Contact your Adaptive Computing account manager for more details.

Welcome to the Moab Docker Integration Reference Guide.

Docker, from Docker, Inc. allows you to package an application with all of its dependencies into a standardized unit for software development and execution.

Moab and Torque support Docker containers in which you can run your workloads. Docker containers provide an isolated environment with the correct linux distribution version of all the libraries the user needs to run the workload. System administrators can preset the containers or users can create their own images and have the administrator upload their images to a central registry so the users can create containers from them. You can also configure job templates to force workloads and/or users to run inside Docker containers, as well as running preemptible or interactive jobs in containerized environments.

The following chapters will help you quickly get started:

- Chapter 1 Introduction to Moab Docker Integration on page 2 – Provides a brief introduction to using Docker containers. Also includes an illustration of the Docker container lifecycle.
- Chapter 2 Installation and Configuration on page 5 – Contains requirements information and procedures to set up Moab and Torque for Docker support. Also includes instructions for copying the Docker job start scripts.
- Chapter 3 Docker Job Submission on page 16 – Provides instructions and information on how to submit jobs using Docker containers.
- Chapter 4 Monitoring and Troubleshooting on page 26 – Provides useful information to monitor and troubleshoot your Moab Docker Integration.

Chapter 1 Introduction to Moab Docker Integration

This chapter provides information about Moab Docker Integration.

In this chapter:

About Moab Docker Integrationfo

Moab Docker Integration (also referred to as Docker support) is an additional package you can elect to include with your Moab configuration. This package enables Moab and Torque jobs to optionally run in a Linux container using Docker. Jobs may be run in a container of the user's choice and, within the container, the job script is run under the submitting user id. For single-node jobs, a container job may also be preempted (checkpointed at the container level; processes are not checkpointed) and resumed in the same container in which it was previously run.

Specifically, this Reference Guide will show you how to set up your CentOS 7 or SLES 12 Host OS to run a job in a CentOS Docker Container.

Docker Integration requires Torque Resource Manager as the Moab Resource Manager.

Docker Integration is available for Moab 9.0 and Torque 6.0.

Container Job Lifecycle

The following images show the job lifecycle when using Docker containers.

Image 1-1: Single-node job

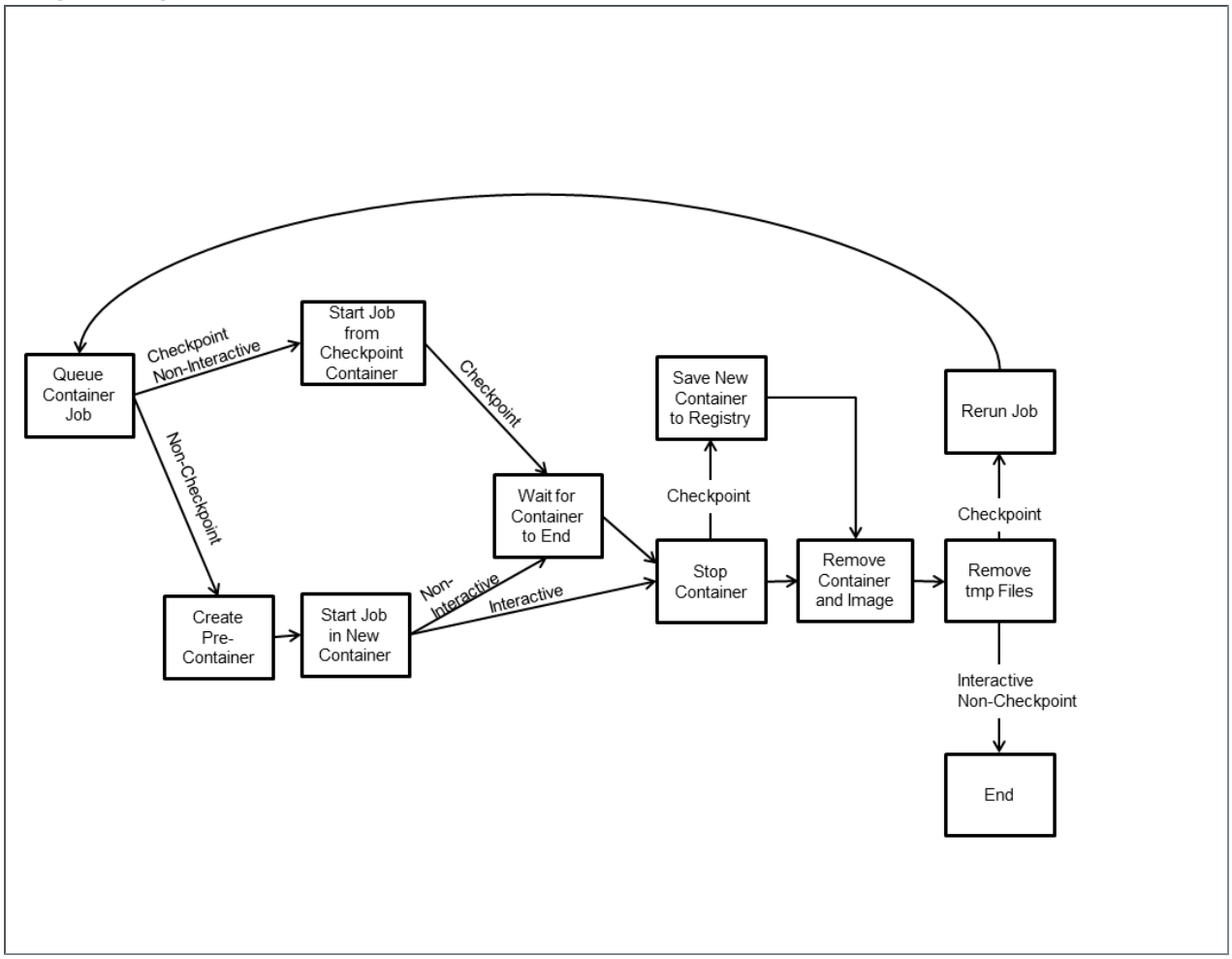
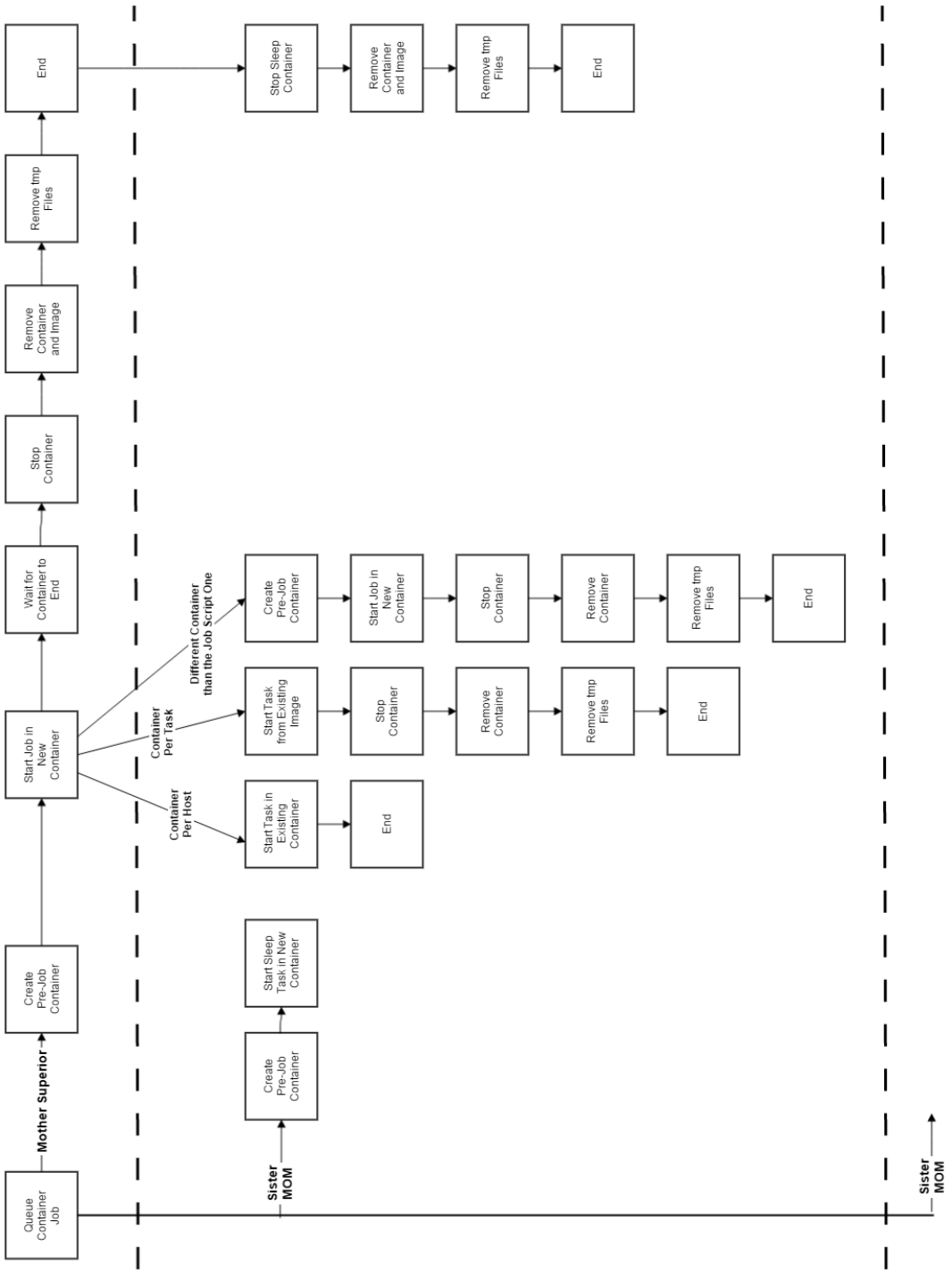


Image 1-2: Multi-node job (shown without checkpointing)



Chapter 2 Installation and Configuration

This chapter provides information on how to set up and configure Torque Resource Manager for Docker container support.

> This chapter assumes you have already installed Moab Workload Manager and Torque Resource Manager; including the Torque MOM Hosts (also referred to as compute nodes).

In this chapter:

Requirements

This topic contains information on the requirements for the Moab Server Host and the Torque MOM Hosts (also referred to as the compute nodes).

Moab Server Host

This is the host on which Moab Server resides.

Requirements:

- Moab Workload Manager 9.0.2

Torque MOM Hosts

These are the hosts on which the Torque MOM Client reside (also referred to as the compute nodes).

Requirements:

- Red Hat 7-based or SUSE 12-based Host OS

  > For Red Hat 7-based systems, Security Enhanced Linux (SELinux) may be in any mode (enforcing, permissive, or disabled).

- Python 2.7; assumed to be installed with the Host OS

- Docker 1.8.2 and after
- Torque 6.0.2 and after
- If using preemption, all nodes should have a shared $HOME file system

Installing and Configuring Docker

This topic provides instructions on installing and configuring Docker as part of your Moab Docker Integration.

If preemption is part of your system configuration, you will also need to repeat these steps on the host your local registry will reside.

Install and Configure Docker

On each Torque MOM Host (and on the local registry host; if using preemption), do the following:

1. Install Docker.
   - Available by default in the OS.
     - Red Hat 7-based systems; packaged with version 1.8.2

       ```
       yum install docker
       ```

     - SUSE 12-based systems; packaged with version 1.9.1

       ```
       # Enable container module (Optional)
       SUSEConnect -p sle-module-containers/12/x86_64 -r ' '

       zypper install docker
       ```

   - Optional, available from the Docker repository. See also https://docs.docker.com/engine/installation/linux/.
     - Red Hat 7-based systems *only*; version 1.10.1

       ```
       # Update package manager
       yum update

       curl -sSL https://get.docker.com/ | sh
       ```

2. Configure Docker.
   - Red Hat 7-based systems
     - If you installed Docker using the default version in the OS, edit `/etc/sysconfig/docker` as follows:

- Remove `--selinux-enabled`.
- Append the following line to OPTIONS.

```
-s devicemapper --storage-opt dm.fs=xfs --storage-opt dm.no_warn_on_loop_
devices=true --exec-opt
```

- If you installed Docker using the Docker repository *(Red Hat 7-based systems only)*, do the following:

a. Create the service directory.

```
mkdir /etc/systemd/system/docker.service.d
```

b. In the directory you just created, create the `execstart_override.conf` file and add "[Service]" and "ExecStart=" as the first and second line; respectively.

c. From `/lib/systemd/system/docker.service`, copy the ExecStart line and place it as the last line in the `execstart_override.conf` file.

d. In the `execstart_override.conf` file, append the last line to include `-s devicemapper --storage-opt dm.fs=xfs --exec-opt native.cgroupdriver=cgroupfs`.

The following is an example of the configured `execstart_override.conf` file.

```
[Service]
ExecStart=
ExecStart=/usr/bin/docker daemon -H fd:// -s devicemapper --storage-opt
dm.fs=xfs --storage-opt dm.no_warn_on_loop_devices=true --exec-opt
native.cgroupdriver=cgroupfs
```

e. Reload the systemd manager configuration.

```
systemctl daemon-reload
```

- SUSE 12-based systems

No additional configuration required.

3. Start/restart Docker.

```
systemctl restart docker.service
```

4. Set Docker to start after reboot.

```
systemctl enable docker.service
```

5. Verify the installation.

```
docker run hello-world
```

6. Download any Docker images you want to make available to users on your system. For example on each MOM host:

```
docker pull centos:6
```

Configuring Torque

This topic provides additional configuration requirements for your Torque Server Host and Torque MOM Hosts as part of your Moab Docker Integration.

You must have installed Moab Workload Manager and Torque Resource Manager; including the Torque MOM Hosts (also referred to as compute nodes) before proceeding with this topic.

If container usage will be tracked with the jobs, Torque should be built with cgroup support. See Customizing the Install in the *Torque Resource Manager Administrator Guide* for more information.

In this topic:

Configure the Torque Server Host

On the Torque Server Host, do the following:

1. Set root to be a queue manager on each Torque MOM Host.

```
qmgr -c "s s managers += root@<MOM Host>"
```

2. Set kill delay to 30 seconds.

This will set the kill delay on the system level. Alternately, you can set the kill delay on a queue or a user job.

```
qmgr -c "s s kill_delay=30"
```

If you have a slow file system (or network) and the Docker images are taking awhile to clean up, you may need to increase kill_delay to a higher value to give the job starter enough time to cleanup after the container has been terminated.

Configure the Torque MOM Hosts

> The following instructions assume your Torque home directory is `/var/spool/torque` on each Torque MOM Host. If your home directories are different, you will need to adjust the instructions in this topic accordingly.

On *each* Torque MOM Host, do the following:

1. Install and enable cgroups. See Torque NUMA-Aware Configuration in the *Torque Resource Manager Administrator Guide* for detailed instructions.
2. Configure Torque to run the job starter with elevated privileges. Append these lines to `/var/spool/torque/mom_priv/config`:
   - $job_starter /var/spool/torque/mom_priv/job_starter
   - $job_starter_run_privileged true
   - $spool_as_final_name true (required only if using preemption)
3. Restart pbs_mom.

```
systemctl restart pbs_mom.service
```

Installing and Configuring Docker Job Start Scripts

This topic provides instructions on how to install and configure the Docker job start scripts.

Install Docker Job Scripts

On each Torque MOM Host, do the following:

1. Copy these job starter scripts to `/var/spool/torque/mom_priv` after customization listed in the following steps:
   - job_starter
   - epilogue
   - prologue.parallel
   - epilogue.parallel
   - job_starter_common.pyc
   - job_starter_config.py
   - job_starter_container_config.json

Customize the Configuration Files

There are two configuration files you should review and customize before starting Docker jobs:

- job_starter_config.py – Contains configuration values required for the Job Start scripts.
- job_starter_container_config.json – Contains configuration values that may be set on a per-container basis. The values in this file control how containers will be started for jobs on your system.

Job Start Scripts Configuration File

The contents of this file provide values used by the job starter script. The file is Python-based; lines should contain a simple "name = value" format or begin with a "#" to indicate a comment. The file you use should list these values. If you are just using the provided one, default values are listed.

- DOCKER_CMD – Path to the "docker" command on your system. Default: "/usr/local/bin/docker".
- GETENT_CMD – Path to the "getent" command on your system. Default: "/usr/bin/getent".
- TORQUE_BIN_PATH – Path to the Torque binaries on your system. Default: "/usr/local/bin".
- TORQUEHOME – Path to Torque's home directory. Default: "/var/spool/torque".
- TORQUE_CPUSET_ROOT – Name of the Torque cpuset root in the cgroup hierarchy. Default: "/torque/".
- REGISTRY_URL – Address of your local registry (if you use checkpoint/restart). Default: "" (no registry).
- SCRATCH_DIR – Path to your scratch file system (used for writing small files). Default: "/tmp".
- CHECKPOINT_SIGNAL – Signal to use for initiating a checkpoint operation on a container job. Default: signal.SIGUSR1 (basically a USR1 signal).
- USE_SYSLOG – Indicates whether to log job starter messages to syslog (True/False). Default: True.
- USE_CGROUPS – Indicates whether to tell Docker to use Torque cgroups when starting containers (True/False). Default: True.
- CONTAINER_CONFIG_FILE – Path to the container configuration file. Default: "/var/spool/torque/mom_priv/job_starter_container_ config.json"
- TOKEN – Random string of text of any length. It is used to created hashed container names. Default: "some-random-string-of-text".

- KILL_DELAY_DEFAULT_SECONDS – Default number of seconds between SIGTERM and SIGKILL when stopping a running container. This value is used *only* if there is not a user (job), queue or server kill_delay value available. Default: 60.
- KILL_DELAY_OFFSET_SECONDS – Kill delay offset seconds. After retrieving the user, queue or server kill_delay value in seconds, the value is decreased by the offset and used with "docker stop -t" so that the container will be stopped before the Torque job. Default: 15.

Container Configuration File

The contents of this file provide values used by the job starter script when starting a container and allow you to control how containers with specific names should be started. The file should be in JSON format as an array of objects.

File Basic Structure

The following is the basic structure of this file.

```
[
  {"<pattern>": { "volumes": <volume-list>,
                "net_mode_default": <net-mode>,
                "net_mode_user_settable": <boolean>,
                "port_ranges_user_settable": <boolean>,
                "port_ranges_allowed_master": <master-port-list>,
                "port_ranges_allowed_worker": <worker-port-list>,
                "linux_capadd": <capadd-list>,
                "linux_capdrop": <capdrop-list>}}
]
```

Where:

- <pattern> is a regular expression. This pattern is compared against the container name that is being started. The remaining values control how the container is started via "docker run".
- <volume-list> is an array of volume strings. Each string translates to a "-v" option to docker run and indicates which files or file systems are to be mounted within the file system. Default: "" (no volumes mounted).

  Special values in the strings may be used and will be expanded at container run time:

  - $HOME – The job owner's home directory.
  - $TORQUEHOME – Torque home directory (as specified in job_starter_config.py by the TORQUEHOME variable).
  - $PBS_JOBID – the job ID.

  Consult the "-v" option in the "docker-run" man page for more information.

- <net-mode> is a string indicating the network mode of the container. Common values are "host", "bridge", or "none". See the --net option in the "docker-run" man page for more information. Default: "bridge".
- <boolean> is a true or false value.
  - "net_mode_user_settable" entry – If set to true, user jobs are allowed to request the network mode of their job containers via the PBS_CONTAINERNETMODE environment variable. False forbids this request. Default: false.
  - "port_ranges_user_settable" entry – If set to true, user jobs are allowed to request ports to be forwarded from their job containers via the PBS_CONTAINERPORTS_MASTER or PBS_CONTAINERPORTS_WORKER environment variables. False forbids the request. Default: false.
- <master-port-list> is an array of port strings. Each string translates to a "-p" option to docker run and indicates which ports are to be published to the master container (the one running the job script) when the network mode is "bridge". The ranges *must* be unique and valid. See the "-p" option in the "docker-run" man page for more information. Default: "" (no master ports published).
- <worker-port-list> is an array of port strings. Similar to <master-port-list> but for worker containers (containers not running the job script). Default: "" (no worker ports published).
- <capadd-list> is an array of Linux capability strings. Each string translates to a "--cap-add" option to docker run and indicates which Linux capabilities are to be added to the container. See the "--cap-add" option in the "docker-run" man page and the "capabilities" man page for more information. Default: [] (empty list--no Linux capabilities added).
- <capdrop-list> is an array of Linux capability strings. Similar to <capadd-list> but for Linux capabilities to be dropped. Default: ["SETUID", "SETGID"] (processes within containers may not manipulate user and group ids).

Sample File

The following is a sample container configuration file. The first entry uses a regular expression that matches all container names. The second entry uses a regular expression that matches only the container name "centos:6".

You may include as many comma-separated entries and you like.


Successive patterns override or to add to each previous one.

```
[
  {"^.+$": { "volumes": ["/tmp/:/tmp/:rw",
              "$HOME:$HOME:rw",
              "$PBS_HOME/aux/$PBS_JOBID:$PBS_HOME/aux/$PBS_JOBID:ro",
              "$PBS_HOME/server_name:$PBS_HOME/server_name:ro",
              "/usr/local/systems:/usr/local/systems:rw"],
             "net_mode_default": "host",
             "net_mode_user_settable": false,
             "port_ranges_user_settable": true,
             "port_ranges_allowed_master": [ "30000:40000", "60000:65535", "8001-
8010:8001-8010" ],
             "port_ranges_allowed_worker": [ "40000:40000"],
             "linux_capadd": [],
             "linux_capdrop": ["SETUID", "SETGID"]}},

   {"^centos:6$": { "net_mode_default": "bridge" }}
]
```

Setting Up the Local Registry (Preemption Only)

If preemption is part of your configuration, you will need to set up the local registry.

⚠ Preemption is only available for single-node jobs.

Set Up the Local Registry

ℹ In the following instructions, "myrepo.host.com" is used as the local registry name. Change this to match your local registry information.

Do the following:

1. On the local registry host *and* on each Torque MOM Host, do the following:
   a. Configure Docker
      - Red Hat 7-based systems.
        - If you installed Docker using the default version in the OS, edit `/etc/sysconfig/docker` to uncomment the "INSECURE_REGISTRY=" line *and* add the the registry host.

          ```
          INSECURE_REGISTRY='--insecure-registry myreg.host.com:5000'
          ```

        - If you installed Docker using the Docker repository, do the following:
          a. Edit the `/etc/systemd/system/docker.service.d/execstart_override.conf` file to append `--insecure-registry`

`myreg.host.com:5000` to the last ExecStart line.

b. Reload the systemd manager configuration.

```
systemctl daemon-reload
```

- SUSE 12-based systems.

  Edit `/etc/sysconfig/docker` to add `--insecure-registry myreg.host.com:5000` to the DOCKER_OPTS value.

b. Restart Docker.

```
systemctl restart docker.service
```

2. On the local registry host, do the following:

   a. Set up the local registry. Complete the steps in https://docs.docker.com/registry/deploying/.

   b. Start the registry.

```
docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

Preparing for Parallel Jobs

A parallel launch agent like pbsdsh (Torque) or mpirun (OpenMPI package) *must* be available and built for each container in which you wish to run tasks.

For example if you are using CentOS 6 for your job and using pbsdsh for your launcher, you will need Torque binaries available for a CentOS 6 system. The following steps identify how to do this for Torque:

1. Select path to store the binaries.
2. Run container with path mounted:

```
docker run -it -v /usr/local/centos6:/usr/local/centos6:rw -v
/usr/local/packages:/usr/local/packages:rw centos:6 /bin/bash
```

3. Install Torque required packages.

   - Red Hat 7-based systems

```
[root]# yum install libtool openssl-devel libxml2-devel boost-devel gcc gcc-c++
```

   - SUSE 12-based systems

```
[root]# zypper install libopenssl-devel libtool libxml2-devel boost-devel gcc
gcc-c++ make gmake
```

4. `cd /usr/local/packages` (assume Torque source or tarball is here).

5. Run `configure --prefix=/usr/local/centos6 ….`
6. Run make install.

At the completion of these steps you will have CentOS 6 Torque binaries that can be used in subsequent job submissions. See Job Submission Using Multiple Job Hosts on page 22 for an example job submission.

Chapter 3 Docker Job Submission

The Moab Docker Integration component provides several options for submitting jobs.

You can submit interactive, non-interactive, or preemptible jobs (single-node jobs only). You can also choose whether to submit those jobs directly to Torque or by using a Moab job template.

In this chapter:

Related Topics

Container Job Lifecycle on page 3

Submitting Interactive and Non-Interactive Jobs

This topic provides instructions for submitting interactive and non-interactive jobs.

Submit Interactive Job

Use the following commands to submit an interactive job from Moab or Torque.

- Torque

```
qsub -I -v PBS_CONTAINERINFO=<imagename> job.pbs
```

- Moab

```
msub -I -v PBS_CONTAINERINFO=<imagename> job.pbs
```

Submit Non-Interactive Job

Use the following commands to submit a non-interactive job from Moab or Torque.

- Torque

```
qsub -v PBS_CONTAINERINFO=<imagename> job.pbs
```

- Moab

```
msub -v PBS_CONTAINERINFO=<imagename> job.pbs
```

Terminating Jobs

This topic provides instructions for terminating jobs.

Terminate a Job

- Torque

```
qdel <jobid>
```

- Moab
  - ```
    mjobctl -N TERM <jobid>
    ```

    The preferred method for terminating Docker jobs. This command removes both non-preemptible and preemptible (checkpointed) Docker jobs.
  - ```
    mjobctl -c <jobid>
    ```

    or

    ```
    canceljob <jobid>
    ```

    For preemptible (checkpointed) Docker jobs, sends SIGUSR1 to the Docker job causing it to checkpoint. For non-preemptible Docker jobs, these commands terminate Docker jobs similar to how these commands terminate non-Docker jobs.

Using Templates for Docker Jobs

You can set up job templates in Moab to automatically push the PBS_CONTAINERINFO environment variable for your users and to set up different user policies in Moab.

Create a Template

On the Moab Server Host, add the template information to moab.cfg. The following example creates a template for Docker jobs using a CentOS 5 container.

```
# CentOS 5
JOBCFG[centos5template] SELECT=TRUE
JOBCFG[centos5template] FLAGS=RESTARTABLE
JOBCFG[centos5template] ENV=PBS_CONTAINERINFO=localhost:5000/centos:5
JOBCFG[centos5template] ENV=PBS_CONTAINERHOSTNAME=centos5
```

Submit Jobs Using a Template

On the Moab Server Host, do the following:

```
msub -l template=centos5template myjob.sh
```

Enabling Preemptible Docker Jobs

This topic provides information and instructions on how to enable preemptible Docker jobs.


Preemption is only available for single-node jobs.

In this topic:

Configure Moab to Preempt Docker Jobs

On the Moab Server Host, do the following:

1. Edit the `moab.cfg` file to add the checkpoint parameters.

```
# Signal to send to RM when job is checkpointed
RMCFG[pbs] CHECKPOINTSIG=SIGUSR1

# How long to wait for a job to checkpoint before canceling it
RMCFG[pbs] CHECKPOINTTIMEOUT=1:00
GUARANTEEDPREEMPTION TRUE
PREEMPTPOLICY CHECKPOINT
```

2. Create a new or use an existing job template for preemption and then edit the `moab.cfg` file to make that template restartable (all jobs submitted

using this template can be restarted). Add this line (where [centos5template] is your template name):

```
JOBCFG[centos5template] FLAGS=RESTARTABLE
```

See Using Templates for Docker Jobs on page 17 for additional information on templates.

3. Restart Moab

```
[root]# systemctl restart moab.service
```

Communicate With the Local Registry

> In the following instructions, "myrepo.host.com" is used as the local registry name. Change this to match your local registry information.

Do the following:

1. If you have not already done so, set up the local registry. See Setting Up the Local Registry (Preemption Only) on page 13.
2. On each Torque MOM Host, edit the REGISTRY_URL parameter in the job_starter_config.py script to point to the local registry. The REGISTRY_URL is the host name of the local registry host and the port number. For example:

```
REGISTRY_URL='myrepo.host.com:5000'
```

Request a Preemptible Docker Job

Once the local registry, a template, and the moab.cfg file are configured for preemptible Docker jobs, you only need to submit the job using the template configured for preemption (centos5template is used in this documentation).

Test Docker Job Preemption

This section contains instructions on how to test preemption for a Docker job. In these instructions you will modify the job's checkpoint values and submit the job using a restartable job template (centos5template).

On the Moab Server Host, do the following:

1. Submit a job script that records its progress and can recover from where it left off. For example:

```
cat sample-checkpoint.sh
#!/bin/bash
# where to store our progress
CHECKPOINT_FILE=/var/tmp/checkpoint.dat
# how many steps do you want to iterate through?
NSTEPS=20
# how many seconds to sleep for in each step
SLEEP_AMOUNT=5

START=1
if test -f $CHECKPOINT_FILE
then
        echo "Checkpoint data found! Resuming execution..."
        START=`cat $CHECKPOINT_FILE`
fi
echo "Starting from step $START..."
while test $START -lt `expr $NSTEPS + 1`
do
        echo "I'm in step $START"
        # increment counter
        START=`expr $START + 1`
        # sleep for $SLEEP_AMOUNT seconds
        sleep $SLEEP_AMOUNT
        # log progress...
        echo $START > $CHECKPOINT_FILE
done
```

2. Edit the variables that control the application's behavior.

   In the previous step, you will find these variables that can control the application's behavior:

   - CHECKPOINT_FILE
   - NSTEPS
   - SLEEP_AMOUNT

   Modify those variables for your test. In this example, the job will iterate through 20 different steps, sleeping 5 seconds between each of the steps and recording its progress to a checkpoint.dat file. The other variables do not need to be changed.

3. Submit the job using a template that is configured for preemption. For example:

   ```
   msub -l template=centos5template sample-checkpoint.sh

   142
   ```

   This will submit the job requesting the centos5 template. After the job has been running for a few seconds, if you ask Moab to checkpoint it, Moab will push a SIGUSR1 signal to your job and it will then archive a copy of your container's file system in the central registry. The next time it restarts the job, it can pick up where it left off.

4. Wait a few seconds to give the job time to run and then preempt it:

```
mjobctl -C 142

job 142 successfully preempted
```

5. Wait a few more seconds and then you should see the job getting requeued and restarted.
6. After the job has finished, review its output file. The output file will show that the job was able to pick up where it left off.

```
cat sample-checkpoint.sh.o142
Starting from step 1...
I'm in step 1
I'm in step 2
I'm in step 3
I'm in step 4
I'm in step 5
I'm in step 6
Checkpoint data found! Resuming execution...
Starting from step 6...
I'm in step 6
I'm in step 7
I'm in step 8
I'm in step 9
I'm in step 10
I'm in step 11
I'm in step 12
I'm in step 13
I'm in step 14
I'm in step 15
I'm in step 16
I'm in step 17
I'm in step 18
I'm in step 19
I'm in step 20
```

Related Topics

Using Templates for Docker Jobs on page 17

Running Parallel Jobs with Docker

This topic provides information and instructions on how to run parallel jobs with Docker.

In this topic:

Job Submission Using Multiple Job Hosts

The following steps provide an example of a job submission that uses multiple job hosts.

This example assumes that Torque has been installed under a CentOS 6 system and that the path where the binaries have been stored on the host have been mounted in /usr/local/systems/centos6/bin in the container requested by the job. See Installing and Configuring Docker Job Start Scripts on page 9 for more information about how to do this.

1. Submit an interactive job using a CentOS 6 container.

   ```
   qsub -I -v PBS_CONTAINERINFO=centos:6 -l nodes=3:ppn=1
   ```

   When the command prompt appears, the job has been started in a CentOS 6 container. Confirm the container by running:

   ```
   cat /etc/redhat-release
   ```

2. At the command prompt within the CentOS 6 container, launch "`cat /etc/redhat-release`" on *each* of the job nodes.

   Unless indicated otherwise, the remote tasks will be started using containers of the name passed to the job via the PBS_CONTAINERINFO environment variable.

   The following example:

   - Assumes that Torque has been installed under a CentOS 6 system and that the path where the binaries have been stored on the host have been mounted in `/usr/local/systems/centos6/bin` in the container requested by the job. See "Installing and Configuring Docker Job Start Scripts" in the *Torque Resource Manager Administrator Guide* for more information about how to do this.
   - Uses Torque's pbsdsh as the parallel launcher. Other launchers, such as OpenMPI (when built with Torque's TM library) may be used. See Open MPI"Open MPI" in the *Torque Resource Manager Administrator Guide* for more information on building OpenMPI with Torque.

   ```
   export PATH=$PATH:/usr/local/systems/centos6/bin
   pbsdsh cat /etc/redhat-release
   ```

3. Launch "`cat /etc/os-release`" on each of the job nodes in Ubuntu 14.04 containers.

   ```
   pbsdsh -e PBS_CONTAINERINFO=ubuntu:14.04 cat /etc/os-release
   ```

Container Modes

This section provides information on the different container modes you can use.

By default, tasks on job nodes should start very quickly in existing containers that were started at job startup. See the diagram on page 4.

In the default mode, one container per host is created and any parallel launches will start tasks in this single container running on each job node. For each job, you will have one container running on each host.

However, if you wish to create one container per task (and potentially have multiple containers running per host), you can set the PBS_CONTAINERPERTASK environment variable to "true". This may be set at submission time with qsub or msub, or at parallel launch time via pbsdsh.

If you wish to start a different container on each job host than the one specified at job start time, you can set the PBS_CONTAINERINFO environment variable on the launcher. Setting PBS_CONTAINERINFO to a different value than the one used at submit time will force a new container to be created for each task launched via pbsdsh. When running in this mode, each job host may have many containers launched via pbsdsh (even if PBS_CONTAINERPERTASK has been set to false).

Container Startup or Shutdown Process

Tasks started on job nodes may experience a small delay when starting and ending as containers and possibly images are created and destroyed.

This section describes the basic process that happens during parallel task launch and explains delays that may be observed. The actual values may vary based on how your system is setup.

- When a job is started, each job MOM host starts a Docker container of the name specified by the user in PBS_CONTAINERINFO. This container is in a running state and runs a sleep process which consumes very little resource.
- When the job launches a parallel run:
    - via pbsdsh, the command or script specified by pbsdsh will be run as follows:
        - On the mother superior host (the host where the job is running): In the job container.
        - On each sister host (remaining hosts allocated to the job): In the container started at the beginning of the job.

      Using this mode, tasks launched via pbsdsh will start very quickly, usually within a second or less, since no new container on each job host is constructed.
    - via pbsdsh with the PBS_CONTAINERPERTASK environment variable set to true, each host associated with the job will start a new container for the command or script specified by pbsdsh. Tasks

launched this way will start after a small delay, usually a few seconds, while a new container is constructed on each host.

- via pbsdsh with the PBS_CONTAINERINFO value different from the one used at job submission time, each host associated with the job will construct an intermediate container with the user defined in it and then start the command or script within a new container run as the user. Tasks launched this way will start after a delay of a minute or so.

The following identifies the order of launch types from fastest to slowest:

- pbsdsh …
- pbsdsh -e PBS_CONTAINERPERTASK=true …
- pbsdsh -e PBS_CONTAINERINFO=<name> … (where <name> is not equal to the <name> used with PBS_CONTAINERINFO when the job was submitted)

- When the tasks have finished running, the following container/image cleanup activities take place:
  - pbsdsh: no cleanup since tasks run in existing running containers. No delay.
  - pbsdsh -e PBS_CONTAINERPERTASK=true …: container cleaned up. Delay is on the order of a few seconds.
  - pbsdsh -e PBS_CONTAINERINFO … (where <name> is not equal to the <name> used with PBS_CONTAINERINFO when the job was submitted): container cleaned up and task image cleaned up. Delay is usually under about 10 seconds.

Related Topics

Container Environment Variables on page 24
Container Job Lifecycle on page 3 Container Job Lifecycle on page 3

Container Environment Variables

These container environment variables may be used when running serial *or* parallel jobs:

- PBS_CONTAINERHOSTNAME – Host name of the job container when using the "bridge" network mode. Controls the "--hostname=" option to docker run. This sets the hostname (if allowed) on the job container *only* and not the worker containers.
- PBS_CONTAINERINFO – Name of container to start. Example: centos:6
- PBS_CONTAINERNETMODE – Desired network mode for the running

container. Controls the "--net=" option to docker run. See the docker-run man page for options. Example: host

These container environment variables may be used *only* when running parallel jobs that use container network mode "bridge" (see the Docker man page for more information on the "bridge" network mode):

- PBS_CONTAINERPORTS_MASTER – Container ports to use with master container. Controls the "-p' option to docker run. A comma separated list of ports or ranges of ports will be split into separate -p options to docker run. See the docker-run man page for options.

  The ranges *must* be unique and valid *and* must not conflict with those specified in the container configuration file for the named container. Example: 5001:5001,5100-5200:5100-5200
- PBS_CONTAINERPORTS_WORKER – Container ports to use with worker container(s). Similar to PBS_CONTAINERPORTS_MASTER. Example: 50000:50000
- PBS_CONTAINERPERTASK – (True/False) If set to "true", each host associated with a parallel job will start a new container for the command or script specified by pbsdsh.

Chapter 4 Monitoring and Troubleshooting

This chapter provides useful information to monitor and troubleshoot your Moab Docker Integration.

Also see Container Job Lifecycle beginning on page 3 for examples of single-node and multiple-node job lifecycles.

In this chapter:

Script Monitoring

This topic provides information about monitoring your Moab Docker Integration.

Enable Scripts to Log to syslog

The "USE_SYSLOG" value *must* be set to "True" in the `job_starter_config.py` file to enable scripts to log their actions to syslog.

job_starter script

The job starter script logs its actions to syslog with this format:

```
job_starter[<pid>]: <function-name>: <job-id>: <message>
```

Where:

- <pid> - process id of the job_starter script
- <function-name> - name of function within job_starter script
- <job-id> - job id
- <message> - string of text logging a particular event or action

The following shows two sample lines in the syslog:

```
Mar 21 15:07:37 testsystem job_starter[26178]: prepare_container: 483.testsystem:
calling docker run with ['/usr/bin/docker', 'run', '--detach=true', '--
volume=/tmp/483.testsystem.JHD5ny.docker.workingfiles/483.testsystem.docker.host_init_
script:/483.testsystem.docker.host_init_script:ro', '--
cidfile=/tmp/483.testsystem.JHD5ny.docker.workingfiles/483.testsystem.docker.cid', '--
cgroup-parent=/torque/483.testsystem', 'centos:6', '/bin/bash',
'/483.testsystem.docker.host_init_script']

Mar 22 11:58:56 testsystem epilogue.parallel[18881]: post_job_cleanup: 508.testsystem:
returning with 0
```

prologue.parallel, epilogue.parallel and epilogue scripts

prologue.parallel, epilogue.parallel and epilogue scripts log messages in a similar format to the job_starter script to syslog.

The following shows two sample lines in the syslog:

```
Mar 22 11:55:27 testsystem prologue.parallel[18667]: sister_mom_prologue_launch:
508.testsystem: returning with: 0

Mar 21 15:07:39 testsystem job_starter[26178]: prepare_container: 483.testsystem:
docker run returned 0
```

Related Topics

Script Errors on page 28

Known Issues or Limitations

This topic lists current known issues (bugs) and limitations (not supported functionality).

Known Issues

- "Docker logs --follow" command has a known issue where it may repeat the first line of output. As a result, non-interactive jobs may log the first line of output to stdout twice.
- A SIGKILL sent to the job without a SIGTERM sent first *and* without enough time after the SIGTERM to handle the container cleanup, can result in any or all of the following:
  - container left running after batch job
  - container image not cleaned up
  - temporary files "/tmp/<*jobid*>" not removed
- Docker image clean up. It is recommended that you set a kill_delay of at least 30 seconds on the system, queue, or job via Torque .

Limitations

- Only serial (single-node) jobs can be checkpointed.
- Interactive jobs cannot be checkpointed.
- Job prologue/epilogue scripts run outside container job.
- Moab "mjobctl -c <*jobid*>" or "canceljob <*jobid*>", when used with preemptible (checkpointed) Docker jobs, sends SIGUSR1 to the Docker job causing it to checkpoint. See Terminating Jobs on page 17 for additional information on terminating preemptible and non-preemptible Docker jobs.
- Cluster environments of mixed Red Hat 7 -based and SUSE 12 -based hosts running Docker have not been tested.
- Parallel task launching using pbsdsh is *only* supported when the job container network mode is "host".

Script Errors

When contacting Adaptive Computing Support regarding script errors, please do the following first:

1. Capture the syslog output related to the failed script. See Script Monitoring on page 26 for more information.
2. Obtain the job_starter script version and commit strings information.

   ```
   cd <directory-where-job_starter-installed>
   ./job_starter --version
   ```

   <directory-where-job_starter-installed> is typically TORQUEHOME/mom_priv. TORQUEHOME is defined in job_starter_config.py.
3. Obtain the tracelogs (Python backtraces) for the failed script. The job_starter, prologue.parallel, epilogue.parallel, and epilogue script tracelogs can be found in:

   ```
   TORQUEHOME/job_starter_tracelogs/<script-name>.<pid>.traceback.
   ```

   TORQUEHOME is set in job_starter_config.py. If the directory it points to does not exist, /tmp is used instead.