Containerized HPCC Systems® Platform

Boca Raton Documentation Team

Containerized HPCC Systems® Platform

Boca Raton Documentation Team

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com>

Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

2021 Version 8.4.6-1

Containerized HPCC Overview

Starting with version 8.0, the HPCC Systems® Platform is focusing on containerized deployments. This is useful for cloud-based deployments (large or small) or local testing/development deployments.

Docker containers managed by Kubernetes (K8s) is a new target operating environment, alongside continued support for traditional “bare metal” installations using .deb or .rpm installer files. Support for traditional installers continues and that type of deployment is viable for bare metal deployments or manual setups in the Cloud.

This is not a *lift and shift* type change, where the platform runs its legacy structure unchanged and treat the containers as just a way of providing *virtual machines* on which to run, but a significant change in how components are configured, how and when they start up, and where they store their data.

This book focuses on containerized deployments. The first section is about using Docker containers and Helm charts locally. Docker and Helm do a lot of the work for you. The second part uses the same techniques in the cloud.

For local small deployments (for development and testing), we suggest using Docker Desktop and Helm. This is useful for learning, development, and testing.

For Cloud deployments, you can use any flavor of Cloud services, if it supports Docker, Kubernetes, and Helm. This book, however, will focus on Microsoft Azure for Cloud Services. Future versions may include specifics for other Cloud providers.

If you want to manually manage your local or Cloud deployment, you can still use the traditional installers and Configuration Manager, but that removes many of the benefits that Docker, Kubernetes, and Helm provide, such as, instrumentation, monitoring, scaling, and cost control.

HPCC Systems adheres to standard conventions regarding how Kubernetes deployments are normally configured and managed, so it should be easy for someone familiar with Kubernetes and Helm to install and manage the HPCC Systems platform.

Note: The traditional bare-metal version of the HPCC Systems platform is mature and has been heavily used in commercial applications for almost two decades and is fully intended for production use. The containerized version is new and is not yet 100% ready for production. In addition, aspects of that version could change without notice. We encourage you to use it and provide feedback so we can make this version as robust as a bare-metal installation.

Bare-metal vs Containers

If you are familiar with the HPCC Systems platform, there are a few fundamental changes to note.

Processes and pods, not machines

Anyone familiar with the existing configuration system will know that part of the configuration involves creating instances of each process and specifying on which physical machines they should run.

In a Kubernetes world, this is managed dynamically by the K8s system itself (and can be changed dynamically as the system runs).

Additionally, a containerized system is much simpler to manage if you stick to a one process per container paradigm, where the decisions about which containers need grouping into a pod and which pods can run on which physical nodes, can be made automatically.

Helm charts

In the containerized world, the information that the operator needs to supply to configure an HPCC Systems environment is greatly reduced. There is no need to specify any information about what machines are in use by what process, as mentioned above, and there is also no need to change a lot of options that might be dependent on the operating environment, since much of that was standardized at the time the container images were built.

Therefore, in most cases, most settings should be left to use the default. As such, the new configuration paradigm requires only the bare minimum of information be specified and any parameters not specified use the appropriate defaults.

The default **environment.xml** that we include in our bare-metal packages to describe the default single-node system contains approximately 1300 lines and it is complex enough that we recommend using a special tool for editing it.

The **values.yaml** from the default helm chart is under 100 lines and can be edited in any editor, and/or modified via helm's command-line overrides. It also is self-documented with extensive comments.

Static vs On-Demand Services

In order to realize the potential cost savings of a cloud environment while at the same time taking advantage of the ability to scale up when needed, some services which are always-on in traditional bare-metal installations are launched on-demand in containerized installations.

For example, an eclccserver component launches a stub requiring minimal resources, where the sole task is to watch for workunits submitted for compilation and launch an independent K8s job to perform the actual compile.

Similarly, the eclagent component is also a stub that launches a K8s job when a workunit is submitted and the Thor stub starts up a Thor cluster only when required. Using this design, not only does the capacity of the system automatically scale up to use as many pods as needed to handle the submitted load, it scales down to use minimal resources (as little as a fraction of a single node) during idle times when waiting for jobs to be submitted.

ESP and Dali components are always-on as long as the K8s cluster is started--it isn't feasible to start and stop them on demand without excessive latency. However, ESP can be scaled up and down dynamically to run as many instances needed to handle the current load.

Topology settings – Clusters vs queues

In bare-metal deployments, there is a section called **Topology** where the various queues that workunits can be submitted to are set up. It is the responsibility of the person editing the environment to ensure that each named target has the

appropriate eclccserver, hThor (or ROXIE) and Thor (if desired) instances set up, to handle workunits submitted to that target queue.

This setup has been greatly simplified when using Helm charts to set up a containerized system. Each named Thor or eclagent component creates a corresponding queue (with the same name) and each eclccserver listens on all queues by default (but you can restrict to certain queues only if you really want to). Defining a Thor component automatically ensures that the required agent components are provisioned.

Local Deployment (Development and Testing)

While there are many ways to install a local single node HPCC Systems Platform, this section focuses on using Docker Desktop.

Prerequisites

Windows 10	Mac	Linux
• Docker Desktop & WSL 2 • Helm OR • Docker Desktop & Hyper-V • Helm OR • Docker • Kubectl • Helm • Minikube	• Docker Desktop • Helm	• Docker • Helm • Minikube

Add a repository

To use the HPCC Systems helm chart, you must add it to the helm repository list, as shown below:

```
helm repo add hpcc https://hpcc-systems.github.io/helm-chart/
```

Expected response:

```
"hpcc" has been added to your repositories
```

To update to the latest charts:

```
helm repo update
```

Expected response:

```
Update Complete. Happy Helming!
```

Start a default system

The default helm chart starts a simple test system with Dali, ESP, eclccserver, two eclagent queues (ROXIE and hThor mode), and one Thor queue.

To start this simple system:

```
helm install mycluster hpcc/hpcc --version=8.2.2
```

The --version argument is optional, but recommended. It ensures that you know which version you are installing. If omitted, the latest non-development version is installed. This example uses 8.2.2, but you should use the version you want.

Expected response:

```
NAME: mycluster
LAST DEPLOYED: Tue Mar 23 13:26:55 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing the HPCC chart.

This chart has defined the following HPCC components:
dali.mydali
dfuserver.dfuserver
eclagent.hthor
eclagent.roxie-workunit
eclccserver.myeclccserver
esp.eclwatch
esp.eclservices
esp.eclqueries
esp.esdl-sandbox
esp.sql2ecl
roxie.roxie
thor.thor
sasha.dfurecovery-archiver
sasha.dfuwu-archiver
sasha.file-expiry
sasha.wu-archiver
```

To check status:

```
kubectl get pods
```

Expected response:

```
NAME                                    READY   STATUS    RESTARTS   AGE
eclqueries-7fd94d77cb-m7lmb             1/1     Running   0          2m6s
eclservices-b57f9b7cc-bhwtm             1/1     Running   0          2m6s
eclwatch-599fb7845-2hq54                1/1     Running   0          2m6s
esdl-sandbox-848b865d46-9bv9r           1/1     Running   0          2m6s
hthor-745f598795-ql9dl                  1/1     Running   0          2m6s
mydali-6b844bfcfb-jv7f6                 2/2     Running   0          2m6s
myeclccserver-75bcc4d4d-gflfs           1/1     Running   0          2m6s
roxie-agent-1-77f696466f-tl7bb          1/1     Running   0          2m6s
roxie-agent-1-77f696466f-xzrtf          1/1     Running   0          2m6s
roxie-agent-2-6dd45b7f9d-m22wl          1/1     Running   0          2m6s
roxie-agent-2-6dd45b7f9d-xmlmk          1/1     Running   0          2m6s
roxie-toposerver-695fb9c5c7-9lnp5       1/1     Running   0          2m6s
```

```
roxie-workunit-d7446699f-rvf2z                1/1     Running   0          2m6s
sasha-dfurecovery-archiver-78c47c4db7-k9mdz   1/1     Running   0          2m6s
sasha-dfuwu-archiver-576b978cc7-b47v7         1/1     Running   0          2m6s
sasha-file-expiry-8496d87879-xct7f            1/1     Running   0          2m6s
sasha-wu-archiver-5f64594948-xjblh            1/1     Running   0          2m6s
sql2ecl-5c8c94d55-tj4td                       1/1     Running   0          2m6s
thor-eclagent-6b8f564f9c-qnczz                1/1     Running   0          2m6s
thor-thoragent-56d788869f-7trxk               1/1     Running   0          2m6s
```

Note: It may take a while before all components are running, especially the first time as the container images need to be downloaded from Docker Hub.

Use the default system

Your system is now ready to use. The usual first step is to open ECL Watch.

Note: Some pages in ECL Watch, such as those displaying topology information, are not yet fully functional in containerized mode.

Use this command to get a list running services and IP addresses:

```
kubectl get svc
```

Expected response:

```
NAME                  TYPE          CLUSTER-IP      EXTERNAL-IP  PORT(S)           AGE
eclqueries            LoadBalancer  10.108.171.35   localhost    8002:31615/TCP    2m6s
eclservices           ClusterIP     10.107.121.158  <none>       8010/TCP          2m6s
eclwatch              LoadBalancer  10.100.81.69    localhost    8010:30173/TCP    2m6s
esdl-sandbox          LoadBalancer  10.100.194.33   localhost    8899:30705/TCP    2m6s
kubernetes            ClusterIP     10.96.0.1       <none>       443/TCP           2m6s
mydali                ClusterIP     10.102.80.158   <none>       7070/TCP          2m6s
roxie                 LoadBalancer  10.100.134.125  localhost    9876:30480/TCP    2m6s
roxie-toposerver      ClusterIP     None            <none>       9004/TCP          2m6s
sasha-dfuwu-archiver  ClusterIP     10.110.200.110  <none>       8877/TCP          2m6s
sasha-wu-archiver     ClusterIP     10.111.34.240   <none>       8877/TCP          2m6s
sql2ecl               LoadBalancer  10.107.177.180  localhost    8510:30054/TCP    2m6s
```

Notice the **eclwatch** service is running on **localhost:8010**. Use that address in your browser to access ECL Watch.

Inside ECL Watch, press the ECL button and go to the Playground tab.

From here you can use the example ECL or enter other test queries and pick from the available clusters available to submit your workunits.

Terminate (Decommission) the system

To check which Helm charts are currently installed, run this command:

```
helm list
```

To stop the HPCC Systems pods, use helm to uninstall:

```
helm uninstall mycluster
```

This stops the cluster, deletes the pods, and with the default settings and persistent volumes, it also deletes the storage used.

Storage

Persistent Storage for a Local Deployment

When running on a single-node test system such as Docker Desktop, the default storage class normally means that all persistent volume claims (PVCs) map to temporary local directories on the host machine. These are typically removed when the cluster is stopped. This is fine for testing but for any real application, you want persistent storage.

To persist data with Docker Desktop, the first step is to make sure the relevant directories exist:

1. Create data directories using a terminal interface:

   For Windows, use this command:

   ```
   mkdir c:\hpccdata
   mkdir c:\hpccdata\dalistorage
   mkdir c:\hpccdata\queries
   mkdir c:\hpccdata\sasha
   mkdir c:\hpccdata\hpcc-data
   mkdir c:\hpccdata\mydropzone
   ```

   For macOS, use this command:

   ```
   mkdir -p /Users/myUser/hpccdata/{dalistorage,queries,sasha,hpcc-data,mydropzone}
   ```

   Note: If all of these folders do not exist, your cluster may not start.

2. Download the HPCC Platform Helm charts.

   These are available in the HPCC Systems **HPCC-Platform** repository on GitHub (https://github.com/hpcc-systems/HPCC-Platform).

   If you want only the helm charts, the use the **helm-chart** repository (https://github.com/hpcc-systems/helm-chart).

3. Open a terminal and navigate to the **helm** folder of the repository you just downloaded.

4. Install the Helm chart from the **examples/local** directory in your local repository.

   This chart creates persistent volumes based on host directories you created earlier.

   ```
   # for a WSL2 deployment:
   helm install hpcc-localfile examples/local/hpcc-localfile
              --set common.hostpath=/run/desktop/mnt/host/c/hpccdata

   # for a Hyper-V deployment:
   helm install hpcc-localfile examples/local/hpcc-localfile --set common.hostpath=/c/hpccdata

   # for a macOS deployment:
   helm install hpcc-localfile examples/local/hpcc-localfile --set common.hostpath=/Users/myUser/hpccdata
   ```

   The **--set common.hostpath=** option specifies the base directory:

   The path **/run/desktop/mnt/host/c/hpccdata** provides access to the host file system for WSL2.

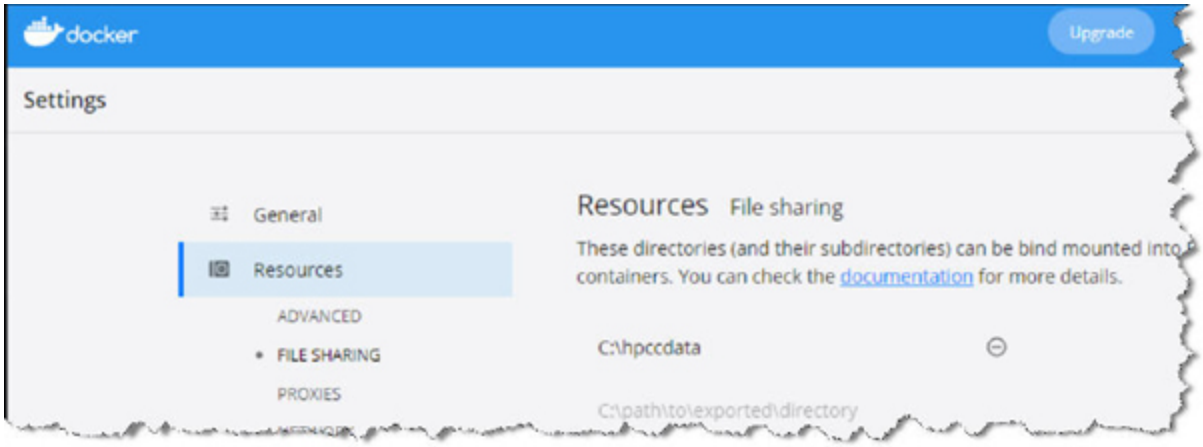
   The path **/c/hpccdata** provides access to the host file system for Hyper-V.

The path **/Users/myUser/hpccdata** provides access to the host file system for Mac OSX.

Note: The value passed to --set common-hostpath is case sensitive.

5. If you are using Docker Desktop with Hyper-V, add the shared data folder (in this example, C:\hpccdata) in the Docker Desktop settings.

   This is **not** needed in a macOS or WSL 2 environment.



6. Finally, install the hpcc Helm chart, and provide a yaml file that provides storage information that uses the PVCs created by the previous step.

   The example directory contains a sample yaml file that can be used in this case:

```
helm install mycluster hpcc/ --version=8.2.2
             -f examples/local/values-localfile.yaml
```

   The --version argument is optional, but recommended. It ensures that you know which version you are installing. If omitted, the latest non-development version is installed. This example uses 8.2.2, but you should use the version you want.

7. To test, create some data files and workunits by submitting to Thor some ECL code like the following:

```
LayoutPerson := RECORD
  UNSIGNED1 ID;
  STRING15  FirstName;
  STRING25  LastName;
END;
allPeople := DATASET([ {1,'Fred','Smith'},
                       {2,'Joe','Jones'},
                       {3,'Jane','Smith'}],LayoutPerson);
OUTPUT(allPeople,,'MyData::allPeople',THOR,OVERWRITE);
```

8. Use the helm uninstall command to terminate your clusters, then restart.

9. Open ECL Watch and notice your workunits and logical files are still there.

Import: Storage Planes and how to use them

Storage planes provide the flexibility to configure where the data is stored within an HPCC Systems platform, but it doesn't directly address the question of how to get data onto the platform in the first place.

Containerized platforms support importing data in two ways:

- Upload to a Landing Zone and Spray (not yet implemented in the containerized version)
- Copy to a Storage Plane and access directly

Beginning with version 7.12.0, new ECL syntax was added to access files directly from a storage plane. This is similar to the **file::** syntax used to directly read files from a physical machine, typically a landing zone.

The new syntax is:

```
~plane::<storage-plane-name>::<path>::<filename>
```

Where the syntax of the path and filename are the same as used with the **file::** syntax. This includes requiring uppercase letters to be quoted with a ^ symbol. For more details, see the Landing Zone Files section of the *ECL Language Reference.*

If you have storage plane configured as in the previous section, and you copy the **originalperson** file to **C:\hpccdata\hpcc-data\tutorial**, you can then reference the file using this syntax:

```
'~plane::data::tutorial::originalperson'
```

Note: The **originalperson** file is available from the HPCC Systems Web site (https://cdn.hpccsystems.com/install/docs/3_8_0_8rc_CE/OriginalPerson)