Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker

Charles Zheng and Douglas Thain
What is a container and why we want to use it?
• A container is a way of "jailing" an application so that it cannot see other applications.

• Instead of virtualizing hardware, containers rest on top of a single Linux instance, which leaves us with a small, neat capsule containing our application.
How is it used in the wider world?
● Used as a tooling mechanism to improve resource management, process isolation and security in application virtualization

● Be incorporated into most DevOps applications, including Puppet, Chef and Vagrant
What is workflows?
A workflow is a form of a workflow management system designed for executing a series of computation and data analysis.
What is Makeflow and Work Queue and how they work together?
Makeflow

A command line workflow engine using a syntax similar to classic Make

Example:

index: seq.fasta bwa

./bwa -index seq.fasta
Work Queue

A user-level execution engine, which consists of a master library and many worker processes that deployed across clusters.
Put them together
How to create execution environment for workflow?
● Install desired operating system and applications on all computing nodes

● Create sandbox directories and place all dependencies in the sandbox

● Use virtual machines to define a consistent environment.
The approach we proposed and how we implement it
• Use Lightweight container (Docker) to define the execution environment

• Integrating Docker into different level of Makeflow and Work Queue system
  ○ Wrapper-script
  ○ Worker-in-container
  ○ Container-in-worker
  ○ Shared-container
What is Docker and why we chose it?
Docker

An operating-system level container technology used for deliver desired execution environment.

Example:

docker run ubuntu bwa -index seq.fasta
● Big user community and well supported

● Global Docker Hub, which is easy to use

● Build once, run anywhere
● New docker related projects

○ Docker Swarm: Turns a pool of Docker hosts into a single, virtual host
  https://docs.docker.com/swarm/

○ Docker Machine: help you easily deploy Docker Engines on cloud providers and in your own data center
  https://docs.docker.com/machine/

○ Docker Compose: help you to define a multi-container application in a single file
  https://docs.docker.com/compose/
The implementation details of four different ways to manage containers
Wrapper-script

Wrapping each task at the workflow level

Example:

`makeflow --docker ccl/debian_cctools`

`makeflow --docker ccl/debian_cctools --docker-tar debian_cctools.tar`
Worker-in-container

Running each worker inside a container
Container-in-worker

Running each task inside a container

Example:

work_queue_worker
--docker ccl/debian_cctools
localhost 9123
Shared-container

Running multiple compatible tasks inside one container

Example:

`work_queue_worker --docker-preserve ccl/debian_cctools`  
`localhost 9123`
Three different ways to manage container images
● **Dockerfile:** Describes the entire procedure of building an image.

● **Tarball:** Encoding and saving a binary image into a tarball and later loading the image from the tarball.

● **Image registry:** Global container image repository, which can be used to store container images.
The desired approach:
- Workflow manager build image on the submit machine
- Exported the generated image as a tarball
- Included it as an input dependency for each task
- Distribute the tarball across all computing nodes
How we evaluate the performance
● **Hardware we used:** Notre Dame Data intensive science cluster (DISC), which consisted of 24 8-core Intel Xeon E5620 CPUs each with 32 GB RAM, 12 2TB disks, 1Gb Ethernet, running Red Hat Enterprise Linux 6.5.

● **Experimental workload:** Burrows-Wheeler Alignment (BWA) bioinformatics workflow, which has 4082 short-running tasks.
The evaluation result we got
Base architecture

![Histogram of Execution Time](chart)

- X-axis: Execution Time (s)
- Y-axis: # Tasks

The chart shows the distribution of execution times for various tasks, with a peak between 0 and 10 seconds.
Wrapper-script
Worker-in-container
Container-in-worker
Shared-container

![Histogram of Task Execution Times](image)

- **X-axis**: Execution Time (s)
- **Y-axis**: Number of Tasks

The histogram shows a peak in task execution times around 5 to 10 seconds, with a rapid decrease in the number of tasks as execution times increase.
What we have learned
There are tradeoffs between performance, isolation and consistency.

- If most of tasks in the workflow require same execution environment, the Shared-container configuration should be chosen
- If isolation is required across tasks, the Container-in-worker configuration should be chosen
The management of containers is better done by distributed system rather than user.

- If we are using both Makeflow and Work Queue, Work Queue should be adopted to manage containers.
Further thing remain to be done
● Develop an advanced container management mechanism, which aims to control the number of private and shared containers.

● Integrate other container runtime into the Makeflow and Work Queue system, like Rocket.
Where can you get the software
Lab link: [http://ccl.cse.nd.edu/](http://ccl.cse.nd.edu/)


Stable version, 4.4.2

Release version, 4.5.0
● Download a source package from the download page. And

gnuzip cctools-4.4.2-source.tar.gz
tar xvf cctools-4.4.2-source.tar
cd cctools-4.4.2-source
./configure
make
make install
export PATH=${HOME}/cctools/bin:$PATH

● Go to the makeflow working directory and run

makeflow example.makeflow
Contact us

Charles Zheng: czheng2@nd.edu

Douglas Thain: dthain@nd.edu