Introduction to Linux Containers with Docker and Alces Flight

Configuring, distributing and running containers

Stu Franks - Alces Flight Ltd
May 2017

Contents

Overview

This guide is intended for those who are already experienced with Docker containers and have used Alces Flight who wish to use their application images on Alces Flight instances.

Alces Flight

Alces Flight Cloud software aims to reduce the complexity and skills required to get started with Cloud computing - providing ready-to-go high performance compute (HPC) environments using popular Cloud providers including AWS. Flight software is delivered as a series of self-configuring products, bundled into configurable appliances designed to work together.

The Alces Flight range of Cloud software appliances have been designed with usability in mind - taking the time consuming configuration away and instead providing vital features required to get started with your research.

Leveraging popular and traditional techniques - the Alces Flight software appliances deliver an unparalleled computing experience to users and researchers of all backgrounds and skill levels.

Linux Containers

Linux Containers (LXC) provide a distribution-agnostic method for the development of containers. A container being an environment as close to an operating system as possible without the overhead that comes with a virtual machine.

Linux Containers provide tools to manage the resources allocated to container instances with cgroups and provide a neat base layer for developing portable applications.

Docker

Docker is open-source software that streamlines the Linux Container configuration by building a layer on top of the various tools for setting up containers. It automates the process of setting up and configuring development environments and reduces complexities with application distribution due to the environment being portable.

Useful Links

Alces Flight Environment Docs - http://docs.alces-flight.com/en/2017.1r1/index.html
Linux Container Homepage - https://linuxcontainers.org/
Docker Documentation - https://docs.docker.com/

Why Run Containers on Alces Flight?

Alces Flight provides a wide variety of software support (with the `alces gridware` command) for over a thousand different HPC applications. The Gridware utility automates the installation process and can even compile software against the specific hardware in the system to optimise performance. In certain circumstances, Gridware may not be the ideal solution for implementing your application, reasons for this may be that the application:

- is dependent on a different operating system than that used by Flight (CentOS)
- is an in-house application that isn't part of Gridware
- would benefit from portability between flight instances from a central storage location (usually the Docker Registry at https://hub.docker.com)
- has a complex installation routine that is in need of automation for quicker deployment

Alces Flight offers a range of standard features that will improve the utilisation and distribution of containers throughout the Flight environment.

Shared filesystems provide a quicker method for sharing container images between systems. No need to pull the image from the remote registry to each individual node when it can be loaded via an exported image shared from the local filesystem on the login node. Alternatively this can be achieved with a local registry server.

Job schedulers come with Flight and allow for the ease of running multiple instances of containers across many nodes in a matter of seconds.

Customization profiles can streamline the configuration process such that Docker can be automatically installed and images can be pulled upon node startup. This reduces the overhead required to prepare the cluster for running your containers. Customization profiles can also be applied to new Flight instances at the time of CloudFormation so it's completely ready for use after creation.

More info:

- Shared filesystems
  http://docs.alces-flight.com/en/2017.1r1/databasics/data_basics.html
- Job schedulers
  http://docs.alces-flight.com/en/2017.1r1/jobschedulers/jobschedulers.html
- Customization profiles
  http://docs.alces-flight.com/en/2017.1r1/customisation/customisation.html

Launching Docker Containers on Alces Flight

Setup

Before containers can be downloaded and run it is a matter of setting up Docker on the Flight nodes that are to be running containers. In our case, the official Docker release was installed by:

- Adding the Docker CentOS repository to yum (available here - https://blog.docker.com/2015/07/new-apt-and-yum-repos/)
- Installing the `docker-engine` package
- Adding the Flight user to the docker group

Once the above steps have been performed the user can begin to download images and run containers using the `docker` command.

For our testing, the login node and all compute nodes had docker installed and configured as above. To test the proper functionality of docker a simple C# script was written to be run with the official mono repository (available here - https://hub.docker.com/_/mono/). For more information on the container configuration see Appendix A: Hello World Container.

Once built, the container could run and print a message including the machine hostname to stdout. The hostname (in this case, 2fbcbf41b17c) is taken from the UUID of the docker container which is generated at creation time and set as the hostname:

```
[alces@login1(mycluster) ~]$ docker run cs-hello
Hello World from 2fbcbf41b17c!
```

To allow the image to be run from any node in the instance without requiring the image to be redownloaded, the image was exported to `/home/alces` so nodes simply have to import the new image.

```
[alces@login1(mycluster) ~]$ docker save -o
/home/alces/cs-hello-world.dock cs-hello
```

Serial Job

A serial job runs a single-threaded application, that is, it only uses a single CPU core on a node. Serial jobs are usually used for processes that are interactive, use very little

resources or are incapable of utilising multiple cores. Multiple serial jobs can be run side-by-side due to them only utilising the 1 CPU core.

Using the SLURM job scheduler, the docker image was able to be imported and run through a simple job script on a single core of a node (an example of the job script used can be found under Appendix B: Hello World Job Scripts). This job wrote the command output to `/home/alces/jobid.out` to show the results.

```
[alces@login1(mycluster) ~]$ cat 2.out
Loaded image: cs-hello:latest
Hello World from bb9499814663!
```

The output shows that the node was able to import the image stored on the filesystem and execute it locally.

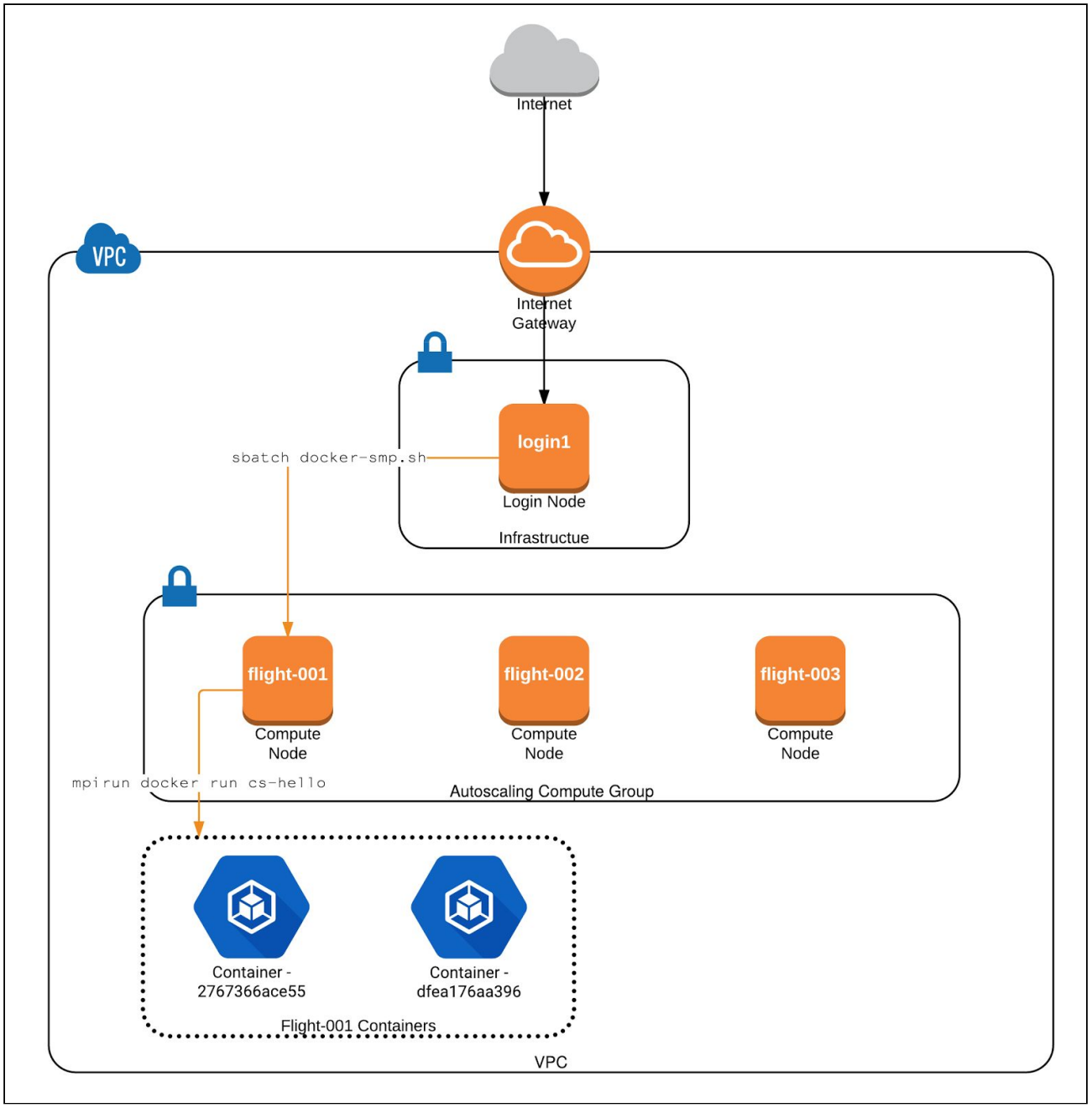
SMP Job

SMP, or multi-threaded, jobs run applications which utilise two or more CPU cores on a node. This can allow for faster processing of results when compared to serial jobs due to the larger allocation of resources. SMP jobs are usually used for resource hungry applications or those that are able to use multiple cores to reduce runtime.

With a few modifications to the serial job script and the addition of MPI software the container can be run in parallel on multiple cores of the same node (an example of the job script used can be found under Appendix B: Hello World Job Scripts). The job output is written to a similar file as that which was used for the batch job.

```
[alces@login1(mycluster) ~]$ cat 5.out
Loaded image: cs-hello:latest
Loaded image: cs-hello:latest
Hello World from 2767366ace55!
Hello World from dfea176aa396!
```

The diagram below reflects the network, nodes and containers as the SMP job was running.



The virtual private cloud (VPC) is the collection of resources that were generated on-demand for the Flight instance. The VPC is isolated from the rest of the public cloud for your personal use.

Running Alces Gridware as a Container

Gridware Containers

With the latest version of Gridware the ability to build containers has been added. This allows for images to be packaged up for distribution to non-Flight instances. For example, a build of Gridware along with everything required for the memtester package can be stored into a container image. This image can then be imported to non-Flight systems or run through cloud based container services.

Alces Flight provides a neat process to automate the Docker installation and configuration process using feature profiles, the docker feature profile can be enabled using the customize utility.

```
[alces@login1(mycluster) ~]$ alces customize apply
feature/configure-docker
Running event hooks for configure-docker
Running configure hook:
/opt/clusterware/var/lib/customizer/feature-configure-docker/conf
igure.d/configure-docker
<snip>
No node-started hooks found in
/opt/clusterware/var/lib/customizer/feature-configure-docker
No member-join hooks found in
/opt/clusterware/var/lib/customizer/feature-configure-docker
No member-join hooks found in
/opt/clusterware/var/lib/customizer/feature-configure-docker
No member-join hooks found in
/opt/clusterware/var/lib/customizer/feature-configure-docker
```

Once the feature profile has been applied, a container can be created from any available Gridware application

```
[alces@login1(mycluster) ~]$ alces gridware docker build
apps/memtester/4.3.0
Building Gridware container 'gridware-apps-memtester-4.3.0'...

  >>> Sending build context to Docker daemon  2.048kB
  >>> Step 1/5 : FROM docker.io/alces/gridware-base
<snip>
  >>> Removing intermediate container 96e738e02294
  >>> Successfully built 8264914fcf6b
```

```
  >>> Successfully tagged
alces/gridware-apps-memtester-4.3.0:latest
```

Using Gridware with Docker - http://docs.alces-flight.com/en/2017.1r1/apps/docker.html

EC2 Container Service (ECS)

ECS is Amazon's front-end on top of EC2 for deploying container instances to auto-scaling groups of clusters. This allows for automated load balancing and scaling of node instances. While this is primarily designed to host long running services (like webservers) it can still be used to effectively launch containers as tasks on the cluster nodes.

Running Gridware inside a container on the ECS service can be beneficial for users who are:

- Familiar with GUIs and prefer them over CLI
- Wanting to run an application multiple times with slightly different arguments (due to the task definition acting as a template where arguments can be overridden at runtime)

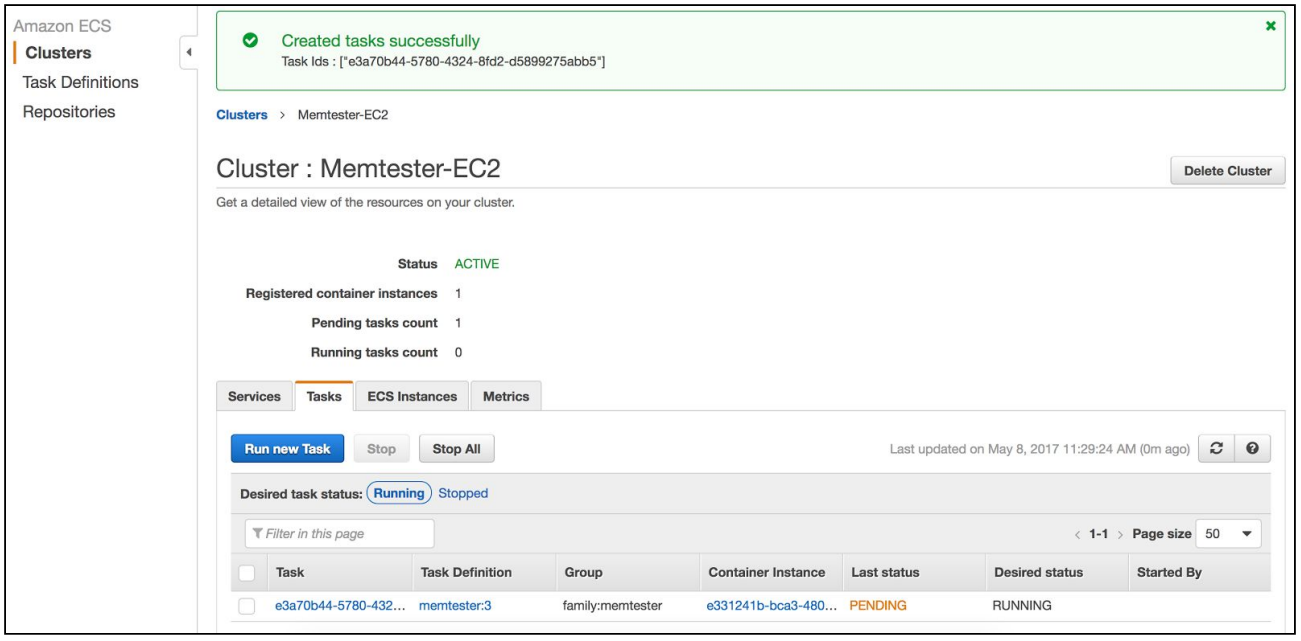
There may also be other reasons that a user would prefer to use ECS over alternative solutions.

The cluster and tasks themselves are setup through the AWS web interface but the output from the container can only be viewed from a terminal session (unless CloudWatch has been configured for output logging).

Using ECS consists of:

- Pushing a container image to an AWS repository (details on using the EC2 Container Repository are available in the AWS documentation - http://docs.aws.amazon.com/AmazonECR/latest/userguide/ECR_GetStarted.html)
- Creating a task definition which utilises the container image (and allows for additional configuration of usage limits, advanced networking and much more)
- Generating a cluster instance using EC2 instances

Once the task and cluster has been configured, a copy of the task can be run on the cluster.



Checking the log output from the container that was launched (after identifying the name of the instance with `docker ps`):

```
[ec2-user@ip-10-0-1-10 log]$ docker logs c76ccc75ea4f
memtester version 4.3.0 (64-bit)
Copyright (C) 2001-2012 Charles Cazabon.
Licensed under the GNU General Public License version 2 (only).

pagesize is 4096
pagesizemask is 0xfffffffffffff000
want 1024MB (1073741824 bytes)
got  1024MB (1073741824 bytes), trying mlock ...locked.
Loop 1/1:
  Stuck Address       : ok
  Random Value        : ok
  Compare XOR         : ok
  Compare SUB         : ok
  Compare MUL         : ok
  Compare DIV         : ok
```

AWS Batch

Much like ECS, AWS Batch is built over EC2 with the purpose of streamlining the container launch process. Unlike ECS, Batch is designed to be a web-based job scheduler using container images.

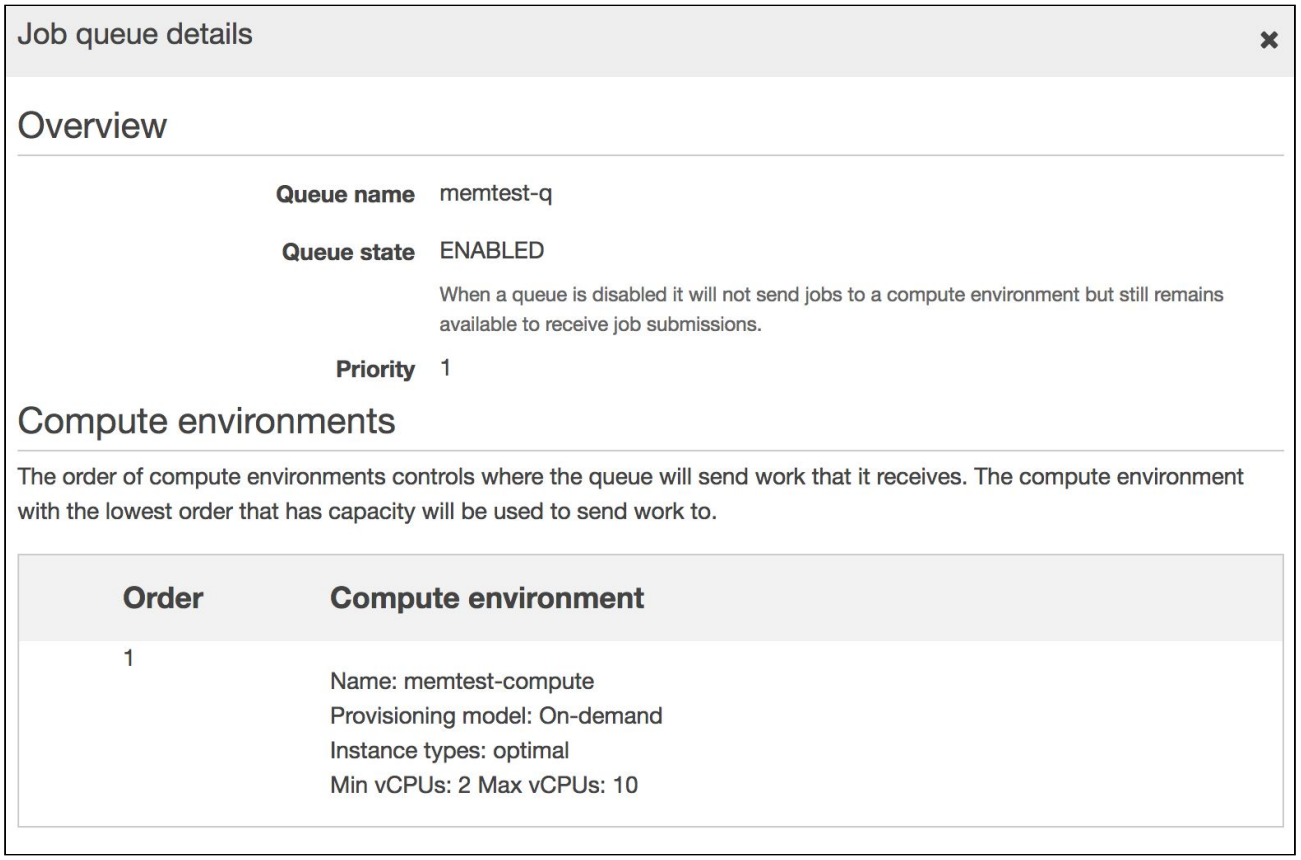
AWS Batch reworks the ECS solution for running containers in the cloud. It provides a different workflow that mimics that of a queue system. While the reasons you may opt to use ECS still apply to Batch, you may additionally want to use Batch for users who are:

- Accustomed to using queuing systems
- In need of job priority control (whether it be to manage their own workload or if multiple users are running on the system)

In order to run a container through Batch, the following needs to be setup:

- Compute environment - the ECS instance types to be used which can be automatically scaled as resources are required
- Job queue - for assigning the compute environment to and submitting jobs to
- Job definition - A template for job submission, this references the container and default limits to be used

An example of a job queue with a compute environment assigned to it can be seen below.



Below is an example of a job definition which specifies the image to use along with any default arguments (which can be overwritten at submission time).

Job definition details

Job definition attributes

Job definition name : revision	memtester:4
Job definition ARN	arn:aws:batch:us-east-1:700366075446:job-definition/memtester:4
Status	ACTIVE

Resource requirements

Job role ARN	
Image	700366075446.dkr.ecr.eu-west-1.amazonaws.com/gridware-memtester:latest

Environment

Command	memtester 1G 1
vCPUs	1
Memory (MB)	2048

Retry strategy

Attempts	1

Parameters

Environment variables

When the job has been submitted to the queue and run, the output from the container is stored in CloudWatch. An example log from CloudWatch can be seen below.

```
Filter events                         all 30s 5m 1h 6h 1d 1w custom
Time (UTC +00:00)  Message
2017-05-12
                   No older events found at the moment. Retry.
12:02:20           memtester version 4.3.0 (64-bit)
12:02:20           Copyright (C) 2001-2012 Charles Cazabon.
12:02:20           Licensed under the GNU General Public License version 2 (only).
12:02:20           pagesize is 4096
12:02:20           pagesizemask is 0xfffffffffffff000
12:02:20           want 1024MB (1073741824 bytes)
12:02:21           got 1024MB (1073741824 bytes), trying mlock ...locked.
12:02:21           Loop 1/1:
12:02:25           Stuck Address : setting 0testing 0setting 1testing 1setting 2testing 2setting 3testing 3setting 4testing 4setting 5testing 5setting 6testing 6setting 7testing 7s
12:02:28           Random Value : \|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/-\|/
12:02:28           Compare XOR : ok
12:02:28           Compare SUB : ok
12:02:29           Compare MUL : ok
12:02:30           Compare DIV : ok
12:02:30           Compare OR : ok
12:02:30           Compare AND : ok
12:02:30           Sequential Increment: ok
```

Non-Alces Machine

As well as running through various container services, it is possible to launch Gridware containers on any machine running an up-to-date installation of Docker. Running applications in this manner is beneficial for users who:

- Don't always have access to the cloud (due to a lack of Internet connectivity - a Gridware container can be run on the local machine at any point)
- Wish to create a distributable and repeatable workflow from a stable image (avoiding issues caused by different library or application versions)

In the example below, the Gridware container with memtester installed is imported onto a MacBook Pro running Docker for Mac.

```
stu@asuka ~$ docker pull stuts/apps-memtester-4.3.0
Using default tag: latest
latest: Pulling from stuts/apps-memtester-4.3.0
17385548ba54: Pull complete
ae704dd74737: Pull complete
f6837a239af8: Pull complete
e75b3df20ef1: Pull complete
d02a087ce4e6: Pull complete
a90dce94a1ab: Pull complete
44925f3ec33e: Pull complete
06c058c02097: Pull complete
Digest:
sha256:c026033668fc5cb0ae5305139f5db5f50177d7a2e9e2b3720f2a8af96d
15a423
Status: Downloaded newer image for
stuts/apps-memtester-4.3.0:latest
```

The image can now be seen in the local repository.

```
stu@asuka ~$ docker images
REPOSITORY                    TAG                 IMAGE ID
CREATED             SIZE
stuts/apps-memtester-4.3.0    latest              bc94c9ed88e2
13 minutes ago      587 MB
```

A container instance of the memtester image can be run to test 1GB of memory for a single loop (output has been snipped for readibility, the `--ulimit` argument was required to allow for 1G of memory to be locked)

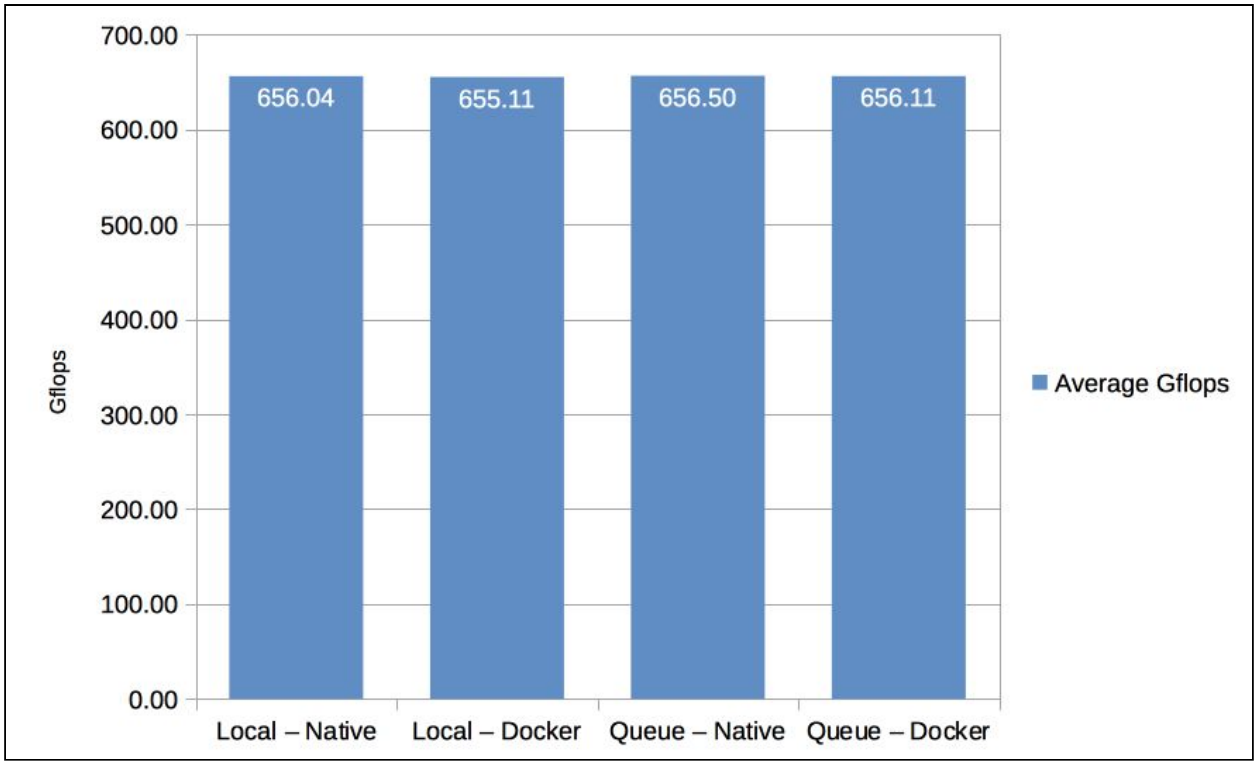
```
stu@asuka ~$ docker run --ulimit memlock=-1:-1
stuts/apps-memtester-4.3.0 memtester 1G 1
memtester version 4.3.0 (64-bit)
Copyright (C) 2001-2012 Charles Cazabon.
Licensed under the GNU General Public License version 2 (only).

pagesize is 4096
pagesizemask is 0xfffffffffffff000
want 1024MB (1073741824 bytes)
got  1024MB (1073741824 bytes), trying mlock ...locked.
Loop 1/1:
  Stuck Address       : ok
  Random Value        : ok
  Compare XOR         : ok
  Compare SUB         : ok
  Compare MUL         : ok
  Compare DIV         : ok
```

Container Performance Discussion

For evaluating the performance of containers on Alces Flight, a recent High Performance Linpack (HPL) from Intel was used (available here - https://software.intel.com/en-us/articles/intel-mkl-benchmarks-suite). This benchmark suite contains precompiled binaries and run scripts for launching an array of arithmetic tests to calculate the Gflop compute power of the system. The HPL test is used to benchmark the Top500 SuperComputers so can be seen as an 'industry standard' test.

The container tests were run on an Alces Flight solo system on AWS consisting of a *t2.large* login node and a *c4.8xlarge* compute node (more info on AWS instance types - https://aws.amazon.com/ec2/instance-types/). The theoretical max Gflop performance for the compute node hardware was calculated using the HPL calculator (available at - http://hpl-calculator.sourceforge.net/) which returned a 100% performance rating of 835 Gflops. A system will not typically achieve theoretical peak performance as this is calculated purely off of the hardware specs and doesn't take into account memory caching, disk I/O or any other operational factors that occur in actual use of systems.

Below is a graph showing the results of the Intel HPL benchmark locally on the node either through a native run or through Docker (HPL job scripts can be found in Appendix C: HPL Job Scripts). These 2 tests were then repeated through the queuing system to show the additional overhead that comes with batch submission. In all cases, the max Gflop performance value reached has been used.



The results above go to show that any performance impact from running containers is negligible (at least as far as HPL is concerned). The results for both native and Docker runs show deviations of less than 1% from repeated runs. When compared to one another there is no observable withdrawal in terms of performance for Docker containers.

While there is no visible difference with the HPL tests, other applications may display larger performance differences due to the functionality so this will need to be considered when deciding between native and containerised applications.

The performance efficiency (when compared to the theoretical max of 835 Gflops) is 78-79% which is to be expected for the reasons explained earlier regarding how theoretical max is calculated.

These results also go to show that running locally to the system or submitting through the queuing system has no observable effect on the performance of the application.

Conclusion

Docker Containers on Flight

When it comes to running non-Gridware applications as containers on Flight instances, not only is it possible but the addition of Alces Flight features improves the ease of distribution, automation and execution of container images. On a standard OS installation there are many steps to configuring the environment for distributed container usage whereas Flight provides multiple tools and customisations to make this process take minutes. In a matter of minutes a container image can be pulled and run across multiple nodes via a queuing system with very little effort.

With Flight, the *feature profile* for Docker reduces the setup and configuration of docker to two commands. This allows for more time to test and run Docker containers without having to worry about possible issues with installation on all the compute nodes.

The *shared filesystems* on Flight are automatically configured when the instance is created, no additional changes are required to start sharing files between the login and compute nodes.

Running containers through the *queue system* will automatically manage resources and distribution of container jobs. Additionally, multiple jobs can be prepared to run even if resources aren't immediately available. This removes the manual overhead required to run jobs local to the compute nodes as the queuing system takes care of this for you.

As the testing above shows, there is no observable difference in performance between native and container runs of the same application. This does not necessarily imply that no applications will see a change in performance when containerised so each HPC application to be used should be tested before deciding on how it will be deployed. If there is a performance difference (no matter how negligible) this will need to be taken into consideration. Conversely, any performance loss may be an acceptable tradeoff for better usability of the system with the containers reducing OS level dependency conflicts to allow the application to be launched effectively.

Gridware as a Container

As a result of the new docker components in Gridware it has become a lot easier to create images with one of hundreds of open-source HPC packages inside. With only a couple of commands, a distributable image is ready to be imported by any machine that's running Docker and can even be run on various cloud-based container services.

This increases the usability of the Gridware utility for application management. Allowing software that has been used on Flight to be conveniently bundled together to be ran

elsewhere, this can help with verifying results on other systems using identical software builds without the complications of configuring the software for non-Flight systems.

Flight vs Container-Only Services

When compared to container-only services like ECS and AWS Batch, Flight has many additional features for managing containers for specific workflows and provides flexibility in system functionality so the system can adapt to your workflow.

AWS Batch has multiple steps to perform before a container instance can be run, including configuring clusters and a queuing system before a job can be run on the nodes. Alces Flight instances come prepared with auto-scaling cluster nodes and a queuing system which is ready to handle jobs.

Appendix A: Hello World Container

The C# hello world script which was saved as `~/hello-world.cs`

```
// A Hello World! program in C#.
using System;
using System.Net;
namespace HelloWorld
{
    class Hello
    {
            static void Main()
            {
                string hostName = Dns.GetHostName();
                string message = "Hello World from " + hostName +
"!";
                Console.WriteLine(message);
            }
    }
}
```

The Docker file used for container creation, saved as `~/Dockerfile`

```
# Use onbuild mono container
FROM mono:latest
# Add file to docker
ADD . /home/alces/
# Compile script
RUN mcs /home/alces/hello-world.cs
# Run app
CMD ["mono", "/home/alces/hello-world.exe"]
```

Appendix B: Hello World Job Scripts

The jobs for the hello world C# script were submitted through the SLURM queuing system. The scripts below are configured such that they are compatible with SLURM. Scripts used to submit to other queuing systems will vary.

The script to run the serial job, saved as `~/docker-serial.sh`

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -o /home/%u/%j.out
docker load -i $HOME/cs-hello-world.dock
docker run cs-hello
```

The script to run the SMP job, saved as `~/docker-smp.sh`

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH -n 2
#SBATCH -o /home/%u/%j.out
module load mpi/openmpi
docker load -i $HOME/cs-hello-world.dock
mpirun docker run cs-hello
```

Appendix C: HPL Job Scripts

The below script, `~/hpl-node.sh`, was used to launch the native HPL run on a node

```
#!/bin/bash -l
#SBATCH -o /home/%u/native.%jout
cd
$HOME/l_mklb_p_2017.2.015/benchmarks_2017/linux/mkl/benchmarks/li
npack/
./runme_xeon64
```

The `Dockerfile` below was used to create the Docker container for running the HPL benchmark

```
# Use CentOS base image
FROM centos:7.3.1611
# Install wget
RUN yum -y install wget
# Download HPL
RUN wget
http://registrationcenter-download.intel.com/akdlm/irc_nas/9752/l
_mklb_p_2017.2.015.tgz
# Untar HPL
RUN tar xzf l_mklb_p_2017.2.015.tgz
# Run benchmark
CMD ["sh", "-c", "cd
/l_mklb_p_2017.2.015/benchmarks_2017/linux/mkl/benchmarks/linpack
&& ./runme_xeon64"]
```

The above file was used to create the Docker container image `linpack-benchmark` which was run on a node using the below job script, `~/hpl-docker.sh`

```
#!/bin/bash -l
#SBATCH -o /home/%u/docker.%j.out
docker run linpack-benchmark
```