ComputeOps: container for High Performance Computing

The High Performance Computing (HPC) domain aims to optimize code in order to use the last multicore and parallel technologies including specific processor instructions. In this computing framework, portability and reproducibility are critical requirements for HPC applications. A way to handle these requirements is to use Linux containers. These “light virtual machines” allow to encapsulate applications within its environment in Linux processes. Containers have been recently rediscovered due to their abilities to provide both multi-infrastructure environnement for developers and system administrators and reproducibility due to image building file. Two main container solutions are emerging: Docker for micro-services and Singularity for computing applications. We present here a white paper summarizing several studies that has been made within the ComputeOps project [1] and ideas that we think that can be usefull for the IN2P3 prospective.

Introduction

Containers provide flexible strategies for packing, deploying and running isolated application processes within multi-user systems and enable scientific reproducibility. Recently, the re-discovery of Linux containers has given rise to an innovative way to encapsulate and share codes and services. Containers are based on specific Linux processes which allow to isolate one task execution from another with a specific set of process ids, user identifiers and filesystem namespace. The emergence of several solutions has changed the developer workflow from the code prototyping level to the production phase on computing centers. As for virtual machines (VM) in the cloud, a wide ecosystem of components is provided to handle specific properties of containers. Furthermore, catalogs of images have been created to offer a central place for sharing images between users. In a production environment, orchestration of containers can be managed by specific tools such as schedulers or orchestrators which allow to combine distributed infrastructures (cluster, cloud) in a hybrid way.

Solution evaluation

Since 2005, several container solutions are emerging: Docker, Rkt, Singularity, uDocker, Shifter, CharlieCloud, Kata containers etc. Among these solutions, Docker [1] has driven the community by providing a complete ecosystem: Docker daemon, client, Hub/private Registry, Compose, Swarm, Machine, etc. The Docker solution is evolving rapidly and it offers a user friendly environment for deploying micro-services. The micro-service concept has been quickly adopted by the industry and the web community due to the simple description of the system architecture. Furthermore, Docker containers can be run on multi-infrastructures such as bare metal system, local computers, cloud computing virtual machines (VMs) and on container clusters managed by an orchestrator.

The Docker's paradigm is based on the representation of a container as a lightweight VM that should host a single service. But this idea of containers as miniature VMs is a wrong approach for the HPC community. Indeed, micro-services can not be used on computing clusters due to security constraints or working methods different from the typical Docker use case. For example, on computing clusters and particularly on supercomputers, the container's permissions has to be set in the unprivileged mode similar to the user's home on the computing infrastructure. This rule is broken for Docker containers where processes in the container can be accessed as root. Moreover, software solutions should be ready to deploy on existing infrastructure or should be deployed easily without interfering with existing systems. The Docker solution is based on a daemon and a client which can create an overhead and/or installation problems depending on the infrastructure. Furthermore, the software solutions must take advantage of the hardware capabilities and low level librairies (i.e. CUDA/ CUDnn for GPU, MPI, etc.). Finally, the Docker container format is not easily portable: scientists are looking for simple solutions to share, publish and ensure the reproducibility of codes and calculations. For all these reasons, Docker is not the optimal solution for executing scientific applications in an HPC environment. Fortunately, there are other solutions at this time that take advantage of the benefits of containerization, while adapting to the scientific environment.

Several criteria has been chosen in order to compare container solutions for HPC application:

- Compatibility with Docker: as Docker is the first container solution already used in a large amount of projects, other technologies have to be compatible with Docker;
- Security: allowing unprivileged mode;
- I/O: transparency allowing a compatibility with MPI processes and X11 graphical export;
- Scheduler: native integration with schedulers. Container images can be submitted and executed as a job on a computing cluster;
- GPU: easy GPU integration in containers;
- Mobility of computing: ability to define, create and maintain a workflow and be confident that the workflow can be executed on different hosts, operating systems (as long as it is Linux) and service providers. This characteristic can also be managed by Workflow Management Systems;
- Single filesystem: containers run a single image file which is the complete representation of all the files within the container. This feature which facilitates mobility also facilitates the reproducibility of computing;

The conclusion of this evaluation has shown that the Singularity solution [2] is the best compromise for the time being. The solution can be installed easily on local cluster and on computing center (as the CC-IN2P3, a High Throughput

Computing (HTC) computing center) and can be used without privileged mode. Singularity is the most widely used container solution in HPC centers and it has also the larger user community: on supercomputer (original use case), on grid infrastructure (LCG), on HTC and HPC computing centers and on the cloud (in order to build Singularity images for Mac and Windows users). It is also fully compatible with Continuous Integration (CI) wich is an important feature for a large community of researchers. It is important to note that all solutions are evolving quickly. In a close future, others technologies can become a good alternative to Singularity (see CharlieCloud for instance) and next solutions seem very promising due to the flexibility they will offer.

Sharing container image

In order to host the container images of micro-services and scientific applications, catalogues of images such as Hubs, registries and Marketplaces are traditionally provided in the container's ecosystem. These catalogues of images matche the experience gained in the cloud computing development but with improvements. Indeed, catalogues of images are linked with source code management tools hosting the recipe used to build the image in a full transparency approach. We will see in this section that other ways can be used for sharing container images between users. An other important aspect will be also approached with the automatic construction of the container's image. Indeed, the GitLab [3] source code management tool also provide a Continuous Integration/Continuous Deployment (CI/CD) plateform which allows to automatically build images of applications within their environment and deploy them on catalogue of images or direcly on infrastructures.

Hubs, registries and Marketplaces

With the arrival of Linux containers, catalogues of images have been proposed to user communities within container solutions in order to share images, tag the version and give information (owner, recipe, code repository, base image, etc.). Public Hubs such as DockerHub [4] and Singularity Hub [5] are respectively based on open source solutions: the registry service and a Web portal. Both Hubs offer to any users the facility to use public images and manage their own container images.

For specific user community such as the IN2P3 research community, private Hubs have also started to flourish. They are based on:

- an authentification mechanism relying on a GitLab/GitHub account;
- collections of images for specific projects;
- manageable workflows;
- the ability for users to rate available containers and view recipe sources.

In order to manage quality and security issues in container images, image scanning can be used in a CI process to check if the image is compliant with the last security releases. Several tools such as Clair [6] and Trivy [7] has been developed these last years in order to provide a way to search Common Vulnerabilities and Exposures (CVEs) inside container images.

File system

CVMFS is a software distribution system allowing to deploy software on world-wide distributed infrastructures. It has been developped to assist High Energy Physics (HEP) collaborations to deploy their software on all sites they are running jobs [8]. The software deployment is usually made on one dedicated server (so called stratum-0) which will mirror the software on all the registered servers. Thanks to that, one can ensure that the required versions of the software are deployed and accessible in all participating sites and configured client machines. CVMFS also provides interesting features, such as a cache system on local machine, to minimize the bandwith required to read the requested files.

CVMFS will then appear quite convenient for sharing and publishing container´s images (definition files will however rather be stored in GitLab). The preferred image's format will be the Singularity unpacked-like (directory tree) images as this format will allow I/O optimisation thanks to advanced CVMFS features. However, the small overhead due to the use of Singularity is mostly coming from the creation of the container (taking into account the user's specifications), so even compressed images can be stored in that system. This latter format is the most used and convenient one as it preserves portability and reproducibility.

A Comprehensive Software Archive Network (CSAN)

Dans le cadre de leurs analyses/simulations, les chercheurs et ingénieurs sont souvent obligés d'installer manuellement les programmes dont ils ont besoin. Les sources de ces programmes se trouvent dans une forêt de sites web ou de dépôts publics et privés. Quelques programmes sont packagés via des outils (comme Brew, Conda, Guix, etc) facilitant ainsi leur installation. Mais cette diversité d'outils ajoute finalement des difficultés pour les utilisateurs et du travail supplémentaire pour les équipes de développement qui doivent se soumettre aux exigences diverses de ces outils et au suivi sans fin de leurs dernières versions au risque de laisser des brèches de sécurité et de négliger l'optimisation des codes. Cette étape chronophage fait que chaque programme supporte rarement plus de deux outils de packaging. Bien entendu, ce packaging peut être fait par de tierces personnes. Les programmes peuvent également être mis à disposition via des conteneurs sur des Marketplace. Se pose alors un problème de qualité et de sécurité de ces solutions.

Ces problèmes sont bien connus depuis des années par la communauté informatique. Une solution éprouvée réside dans les Comprehensive Network Archive (CPAN pour Perl, CTAN pour Tex ou CRAN pour R). Dans ce contexte, une nouvelle archive, la Comprehensive Software Network Archive (CSAN), pourrait émerger. Cette proposition part du constat que les applications scientifiques dans les différents mésocentres sont souvent les mêmes selon les secteurs (bio-info, physique, etc.). Le CSAN permettra aux auteurs de logiciels d'y déposer les releases de leurs sources (ou de pointer vers des dépôts versionnés), charge alors au groupe d'experts de CSAN de traiter ce code, via des méthodes d'intégration et de développement continu, afin de le rendre accessible et installable sans effort sur différents systèmes d'exploitation (Linux, Mac et Windows).

Pour construire cette nouvelle archive, celle-ci doit se baser sur un système de conteneurisation (Singularity) et un outil de packaging (Guix [9]). En effet, la conteneurisation seule ne peut apporter le principe reproductibilité indispensable dans la science. Le choix d'un outil de packaging comme Guix est donc motivé par le fait qu'il est l'un des rares à répondre au besoin de reproductibilité. Concernant Singularity, nous avons pu voir qu'il est à ce jour la technologie de conteneurisation la plus répandue dans le monde des sciences et du HPC.

L'archive CSAN pourra se présenter sous la forme d'un portail de catalogue d'applications dont chaque version renseigne des spécifications matérielles et logicielles (options de compilation, type de processeur, etc. . .) ainsi que les coordonnées d'un ou des mainteneur(s) et toutes les sources et recettes pour reproduire le package de zéro. Les paquets pourront être consultés et évalués via une plateforme web et un Hub (type Docker ou Sregistry). Chaque paquet du catalogue devra être testé et validé par un groupe d'experts composé d'un ou plusieurs référents scientifiques et techniques de manière à proposer à la communauté des applications vérifiées et validées dans des environnements de calcul scientifique identifiés.

Le portail CSAN offrira donc un service stratégique aux développeurs, aux utilisateurs finaux et aux administrateurs système. Ce service permettra d'aboutir à un gain de temps substantiel pour tous. Il permettra aussi une meilleure diffusion des logiciels produits tout en favorisant la notion de reproductibilité.

CI with container

CI has become a standard development practice in the recent years, answering some of the needs that have emerged with Agile and DevOps movements in the software development community. Besides its trending methodological context, CI is now embodied into software systems which provide tools to help software development and collaborative work.

Containers are intensively used in CI workflow in order to package and distribute applications by automatically building container image. The CI workflow involving containers is made of several steps:

- Code commit: the code hosted on the GitLab-CI starts a new pipeline on each commits;
- Pipeline runner: GitLab-CI instance provides a container runner based on Docker. The runner works with any Docker images stored on the Docker Hub or on the GitLab registry (Docker private Registry);
- Pipeline execution: the pipeline running in the container runner can build a Docker image by using for the runner a Docker-in-Docker (DinD) image (Docker image with privileged mode). The DinD provides a Docker daemon inside the container. The image is finaly pushed/updated on the GitLab registry of the project;
- Singularity-in-Docker: to automatically build Singularity images and publishes/updates images on Hubs. By using a pre-packaged Docker image with Singularity (Singularity-in-Docker) and SRegistry client (client for managing Singularity images in version 2), the pipeline builds an image and stores it in GitLab as an artifact or on a Singularity Hub. In order to authenticate on the Singularity Hub, the user token is stored as an environment variable settings;
- Deploy in production: automatically build images can be pulled from the Singularity Hub and executed within the scheduler;
- Image scanning: check the CVEs in the final image.

This automation workflow for building images is an important part in the portability and reproducibility requirements.

Container with Accelerators

See Reprises prospective [10] for GPU description.

Accelerated hardware (HW) can be accessed from containers by mounting from the host the corresponding devices and the relevant software - e.g., driver libraries. Nevertheless, security rules from the OS (seccomp, apparmor, permissions…) could prevent this kind of operation. Hence, container industry added some convenient stacks (a flag at runtime bind-mounting librairies and devices, specific developments (e.g. nvidia-docker)), to automate this process.

In the case of NVidia GPUs, the software stack needed to exploit cards is conveniently factorized. From the host point of view, drivers and libraries needed to manage the cards are HW specific and OS independent, while the CUDA libraries for running applications are compatible with all recent GPUs cards. Thus the container can mount the suitable runtime environment from the host while carrying the desired application environment (OS, CUDA version, etc…). This is made particularly simple to exploit since most container technologies have native options implementing the correct operations. Thus working with NVidia GPU from inside a container is often matter of simply adding the suitable flag.

On the other hand NVidia itself delivers optimized working environments in Docker containers (which can be easily ported to Singularity). This includes environment for machine learning and Notebooks (e.g. DIGITS). But sometimes it is hard to convert the Docker image provided by NVidia into a Singularity image. That is the reason why it can be necessary to keep building local Singularity images for GPU's applications.

Other accelerated HW vendors are also delivering their runtime environments already in containerized version. For example Xilinx provides last version of its drivers and software development kits in Docker containers.

Orchestrator vs Scheduler

Scheduler

On computing center, scheduler solutions are used to manage job distribution from the master node to the cluster of computing nodes. Classical scheduler solutions are: Torque/Maui, Slurm, Grid Engine for LSF for HPC centers and HTCondor for HTC centers.

Schedulers are designed to handle jobs with finite time. They are highly efficient for that purpose when dealing with multiple users, providing job queueing mechanisms and complex fair resources sharing rules. They are designed for traditional computing centers, that usually provide a complete stack of softwares, but don't allow a lot of user customization. Hence the potential impact of containers in this context, lessened by the security problems that the de facto standard Docker poses in term of user management. At the opposite, Singularity containers are managed by scheduler as a job process.

When workflow applications are tightly coupled with workload managers, they cannot be easily executed on another system or infrastructure, nor automatically tested with a CI service. In other words, tight coupling (between workflow applications workload managers) results in a loss of runtime mobility. This issue can however been solved by adding another software layer, a so-called Workflow Management System (WMS), which will properly handle the workflow and its constraints. In the HPC world, there are quite a lot of WMS in use. Some of the most commonly used ones are Pegasus [11], FireWorks [12] and Makeflow [13]. Though some are more dedicated to a specific scientific area, they all provide the same basic features: batch system interoperability, handling of the workflow and its constraints, providing a GUI or at least command lines to follow the proper execution of the workflow. The most mature of these tools will also provide native container integration, for at least Docker and Singularity. However, Singularity integration will always be possible by using a job wrapper as one only needs a command line to properly define how to instantiate a Singularity container.

Orchestrator: the kubernetes era

Orchestrators have been designed to manage containerized workloads, providing primarily resource management, but also a lot of nice features like service reliability, isolation, scalability, etc. However, they were designed to host long running services, not computing jobs. Indeed they have been designed to manage micro-service container such as web services in production. Consequently, they lack finite job scheduling mechanism (no management of potential constraints and links between jobs, input data and output data), and multi user management. But as describe below, some solutions have started to take into account these problems and they will be probably overcame in a close future.

Orchestration solutions are also used to manage job processing on hybrid infrastructures (ie. cloud VMs and HPC clusters) that can lead to connection problems. The execution of containers on these distributed infrastructure can be orchestrated by several tools: Docker Swarm, Mesos/Chronos [14] and Kubernetes [15]. These solutions proactively manage related problems of orchestration such as resource optimization, per-task data I/O staging, error recovery, etc.

Kubernetes is the world's most popular production-grade container orchestration platform, and the most important project of the Cloud Native Computing Foundation (CNCF) [16], which provides the following advantages:

- Velocity: evolving quickly, while staying available;
- Scalability: services are replicated and they support auto-scaling using pre-defined configurations;
- Portability: applications deployed using Kubernetes can be easily transferred between environments;
- Efficiency: applications can be co-located on the same machine without impacting the application themselves due to containerization.

In order to propose a job scheduling mechanism, Kubernetes has a job processing feature allowing to run job to completion. Furthermore, Kubernetes is now orchestrating Singularity containers for compute-driven workloads which shows that a slight convergence between scheduler and orchestrator has started.

Conclusion

Containers have proven that they can provide a pragmatic and efficient solution to the problem of packaging complex software dependencies into self-contained, ready-to-run executable runtimes that can be easily deployed in a portable manner. Singularity, in particular, provides an HPC-friendly container runtime that streamlines adoption in data centers and facilities in which these kinds of applications are generally executed.

References

[1] C. Cavet, A. Bailly-Reyre, D. Chamont, O. Dadoun, A. Dehne Garcia, P.-E. Guérin, P. Hennion, O. Lodygensky, G. Marchal-Duval, E. Medernach, V. Mendoza, J. Pansanel, R. Randriatoamanana, A. Sartirana, M. Souchal, J. Tugler, ComputeOps: container for High Performance Computing, CHEP 2018 Conference (2019), epjconf182860

[1] Docker: https://www.docker.com/

[2] Singularity: http://singularity.lbl.gov/

[3] GitLab: https://about.gitlab.com/

[4] DockerHub: https://hub.docker.com

[5] Singularity Hub: https://singularity-hub.org/

[6] Clair: https://github.com/dctrud/clair-singularity and https://vsoch.github.io/2018/stools-clair/

[7] Trivy: https://github.com/aquasecurity/trivy

[8] CVMFS: https://cernvm.cern.ch/portal/filesystem

[9] Guix: https://guix.gnu.org/

[10] Reprises prospective: https://gitlab.in2p3.fr/CodeursIntensifs/Reprises/tree/master/prospective

[11] Pegasus: https://pegasus.isi.edu/

[12] Fireworks: https://materialsproject.github.io/fireworks/

[13] Makeflow: http://ccl.cse.nd.edu/software/makeflow/

[14] Mesos: http://mesos.apache.org/

[15] Kubernetes: https://kubernetes.io/

[16] CNCF: https://www.cncf.io/