GO-Docker

A batch scheduling system with Docker containers

Olivier Sallou *(Author)*
GenOuest BioInformatics Platform
IRISA/Univ. Rennes 1
Rennes, France
olivier.sallou@irisa.fr

Cyril Monjeau *(Author)*
GenOuest BioInformatics Platform
IRISA
Rennes, France
cyril.monjeau@irisa.fr

Abstract *Multi user open source batch scheduling software based on Docker containers with custom scheduler and executor plugin mechanisms.*

Tracks: Programming and System Software: Cluster System Software/Operating Systems, Cloud-enabling Cluster Technologies and Virtualization

Tags: Docker, container, cluster, open source

I. Introduction

Lightweight virtualization technologies gained attention by offering performance and effective scalability across cloud and physical architecture. GO-Docker[1] is a new open source batch scheduling tool that provides container support (Docker [2]). It is based on proven technologies and tools to provide job isolation and custom images for user jobs.

Its architecture scales to handle large configurations and provides end-user easy access with a Web UI, CLI tools and API access for external programs integration.

Containers provide job isolation, preventing resources overlap, and easier management for the cluster administrators. For the end-user, it provides a choice of operating systems, pre-built configurations and possible root access to the container.

Its plugin architecture eases the integration of new scheduling algorithms or other execution/control mechanisms.

The software targets multi-user systems with a central authentication (ldap, …) and shared storage (home directory, shared data, etc.) and manages Docker access for users, leveraging security concerns with container access.

II. Features

GO-Docker is in active development but is functional for the most common features. Additional features are planned.

Common cluster features are available:

- job submission (execute a command on a node)
- job resources specification (cpu and memory requirements, queues)
- job arrays (submit N jobs with a parent job)
- list active/terminated jobs
- accounting
- kill a job
- suspend/resume a job
- interactive jobs (SSH access)
- user/project quota

Due to container technology, the following features are also available:

- optional root access to the containers
- allow/disable network access in the containers
- job isolation
- job resources consumption monitoring
- past/live monitoring of the job resource usage
- mount of a list of directories selected by user in the container (home, ...)

III. Architecture

A. Access

Go-Docker can be managed via a web interface, a CLI tool or a REST API. API is documented and used by the web interface and the CLI too.

User authenticates using an ACL plugin mechanism. A default LDAP plugin is available. Other mechanisms can be added as new plugin. A unique key, per user, ensures the user authenticity. All transactions are encrypted via HTTPS access and no sensitive data is stored on user computer.

B. Components

The software is built on 4 main components (Figure [1]):

1. web server: manages the API and the GUI. Basically it will only store new job requests and manage queries for the current status of the jobs.
2. scheduler: it is in charge of taking pending jobs , order them according to the selected algorithm and submit them to the executor. It is also based on a plugin architecture to add other scheduling algorithms. A basic FIFO algorithm and Fair Share policy algorithm are provided.
3. Watcher: checks the running jobs status, kill requests, and update database accordingly.
4. Executor: implementations for Docker Swarm [2] and Mesos[3] are provided. Executor is in charge of executing the job on an available node based on requested resources. Additional executors can be added as a plugin. Nodes management is left to the used framework (Swarm or Mesos).

To record job data, two databases are used: MongoDB [2] and Redis . MongoDB is used for long term storage while Redis is used as storage for running jobs for quicker access.

InfluxDB [7] is optionally used to record time-based related data (accounting, number of jobs, mean cpu, mean ram, …)

Cadvisor [6] is also an optional component to get live usage statistics for the container.

At last, a shared directory is used to store job logs and data. A default pair-tree implementation is available but other storage strategies can be easily added.

C. *Scalability*

The architecture is horizontally scalable. One can put as many web servers as needed to manage users load, as well as several watchers to monitor jobs. Scheduler is a single instance as it needs to have a complete view of the pending jobs to schedule them.

The executor is in charge of managing the workload against the execution nodes.

Docker Swarm is a kind of proxy in front of other Docker daemons running on nodes. Mesos [3] integration can scale the solution to manage a very large number of nodes.

IV. Jobs in containers

When a job is executed, it is executed in a Docker container on an execution node, selected according to the job requirements (cpu, ram, queue constraints). A container runs an image (selected by user) on the node host, but the image runs the selected operating system (running on the host Linux Kernel). This choice provides the user the possibility to execute jobs in different OS/configurations or images, containing the software needed for their job, and use the increasing number of containers available.

In the container, the executed command can create new processes, they will run in the same container and within the same boundaries. This prevents different jobs overlapping or over using the same resources. If job is killed by user, for example, all container processes will be killed, leaving the node in a clean state.

Go-Docker offers the possibility for the user to run the task as root or with his Linux id/gid in the container (left to administrator choice). If user is root in the container, he can install new libraries etc.. in the system. If user decides to be root, different ACLs will be used and the volumes (directories) selected by the user to be mounted in the container may be either rejected or mounted in read-only mode.

Several directories can be mounted (according to configuration) in the container. This will usually map to user home directory and other common directories (common data, …). They can be mounted as read-write or read-only. The authentication plugin will check the user requirements and adjust them. An additional directory is mounted, unique, specific to the job. As root, this is the only place where user can write persistent data. After completion, all files in this directories will be set with user origin id/gid so that user access them with his usual access rights. It also contains the standard output log of the job command.

Interactive jobs are SSH access to the container using the user SSH public key. User can get concurrent session access to the same container.

V. target

The software targets local or cloud systems needing a multi-user cluster. Thanks to its plugin architecture, it is easy for researchers to test new schedulers or executors, independently of the task management itself or user interaction. The container usage will also help to get easily some custom environments for the jobs from local created or community created container images.

VI. Status and future work

Software is in active development and several features are already in road-map. Main features are working and a first official release is planned in September 2015.

Next steps are the test of the software at scale on a large infrastructure (supported by an AWS in Education Research Grant award) and the test/evolution of the fair-share scheduler.

References

[1] Go-Docker: http://osallou.bitbucket.org/go-docker/index.html

[2] *Docker https://docs.docker.com/ Docker swarm https://docs.docker.com/swarm/*

[3] *Apache Mesos: http://mesos.apache.org/*

[4] *mongoDB: https://www.mongodb.org/*

[5] *Redis: http://redis.io/*

[6] *cadvisor: https://github.com/google/cadvisor*

[7] *InfluxDB: http://influxdb.com/*

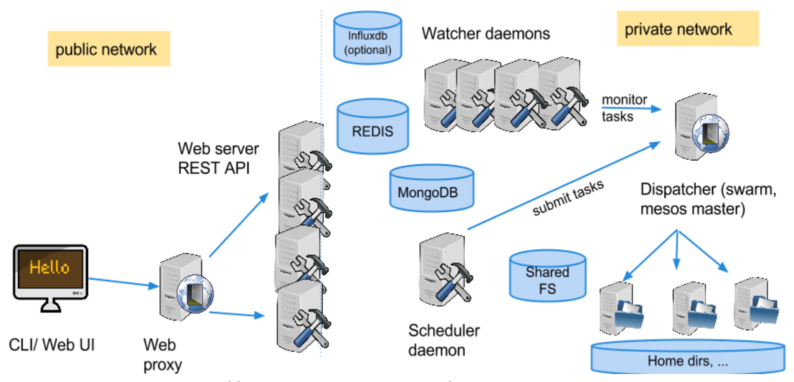

Illustration 1: Architecture