The SIMPLE Framework for deploying containerized grid services

Mayank Sharma[1,*], *Eraldo* Silva Junior[2,**], *Boris* Iliev Vasilev[3,***], *Maarten* Litmaath[1,****], and *Renato* Santana[2,†]

[1]CERN, Geneva, Switzerland
[2]COHEP, CBPF, Rio de Janeiro, Brazil
[3]University of Manchester, Manchester, United Kingdom

Abstract. The Worldwide LHC Computing Grid (WLCG) currently has about 170 sites. In order to support WLCG workloads, each site has to deploy and maintain a number of possibly complex grid services. Quite often, site managers require assistance of WLCG experts, for example when new software versions need to be deployed. Modern configuration management (e.g. Puppet, Ansible), container orchestration (e.g. Docker Swarm, Kubernetes) and containerization technologies (e.g. Docker, Podman) can help make such activities more lightweight by means of packaging sensible configurations of grid services and providing simple mechanisms to distribute and deploy them across the infrastructure available at a site. This article describes the SIMPLE project: a Solution for Installation, Management and Provisioning of Lightweight Elements. The SIMPLE framework leverages modern infrastructure management tools to deploy containerized grid services, such as popular compute elements (e.g. HTCondor, ARC), batch systems (e.g. HTCondor, Slurm), worker nodes, etc. Its architecture follows principles of sustainability, scalability and extensibility. We describe how system administrators can use the framework, as well as the first results, featuring the migration of computing resources to HTCondor at 2 sites. We conclude with an outlook on further developments.

1 Introduction

The Worldwide LHC Computing Grid (WLCG)[1] project is a collaboration of institutes across the world to provide a distributed computing infrastructure for storing and processing the data collected by the 4 main experiments at the Large Hadron Collider (LHC) at CERN: ALICE, ATLAS, CMS and LHCb. In order to support WLCG workloads, each site has to deploy and maintain a number of possibly complex grid services, often requiring significant assistance from WLCG experts. The amount of effort spent on such activities may outweigh the amount of resources provided by a site, particularly if the site is small. Through the use of containers with suitable orchestration and configuration management tools, the

*e-mail: mayank.sharma@cern.ch
**e-mail: esilvaju@cern.ch
***e-mail: boris.vasilev@student.manchester.ac.uk
****e-mail: maarten.litmaath@cern.ch
†e-mail: renato.santana@cern.ch

necessary effort may be reduced significantly. Required services can be prepackaged into Docker containers[2] along with configuration parameters preset to the extent possible, while site-specific values can be supplied through a configuration management system[3] and the containers then get deployed through an orchestration system. In [4] we already described the design principles and architecture of a framework providing such functionality: SIMPLE, a Solution for Installation, Management and Provisioning of Lightweight Elements. Here we focus on how system administrators can use the framework, as well as the first results already obtained in production. We also provide an outlook on further developments.

2 The SIMPLE Framework

A Platform as a Service (PaaS) software enables users to deploy and manage software applications without the complexity of maintaining the underlying infrastructure. A private PaaS is a software that facilitates the development, deployment and operations for IT services on a private infrastructure maintained by site administrators. The SIMPLE framework is a PaaS for orchestrating containerized services at scale. We have primarily focused on the private PaaS use case where site administrators deploy their *Config Master* locally. The Config Master is a single node in the infrastructure that holds the configuration of hosts and services managed through the framework. Lightweight Components are the nodes where the containerized grid services eventually get deployed. The Config Master and some of the Lightweight Components may also be external to the site, allowing remote deployment through SIMPLE acting as a public PaaS. In any case, the site administrator writes a single *site level configuration file* to describe the participating infrastructure and the configuration of the grid services that should be deployed on it. The framework automates the process of setting up, deploying and configuring the given infrastructure using popular configuration management technologies like Puppet[5] or Ansible[6] and container orchestration technologies like Docker Swarm[7] or Kubernetes[8].

2.1 Framework Components

The framework currently consists of the following core components. Each component of the framework has various functions that are executed during different stages of the framework's execution pipeline.

2.1.1 Component Repositories

A Lightweight Component Repository or simply, a component repository, is a Git repository that contains containerized grid services. It follows a standard structure dictated by the SIMPLE Specification Document[9] and comprises the following main files:

1. **Dockerfile**: Describes the Docker image for the grid service (Podman support can be added later).
2. **meta-info.yaml**: Describes the component repository to the framework's components.
3. **config-schema.yaml**: Lists the configuration variables specific to the grid service that a site administrator must specify to ensure proper configuration of the containerized grid service.
4. **default-data.yaml**: Lists the default values for those configuration variables that do not require site-specific values.

At present, there are component repositories available particularly for HTCondor CE[10], HTCondor Batch[11] and HTCondor Workers[12]. The framework was first released and used with component repositories for CREAM CE, Torque Batch and Torque Workers, all of which have become obsolete in the meantime.

While the detailed description of the components is included in the SIMPLE Specification Document [9], below we provide a quick overview of the core components of the framework.

2.1.2 Site Level Configuration File and Schema

The primary function of the site level configuration file is to collect the information from the site administrator that is required by the rest of the framework's components to perform their functions. It is a YAML file whose structure is dictated by the site level config schema [13]. The information present in the site level config is divided into the following sections:

1. **site_infrastructure**: description of the nodes available at the site.
2. **lightweight_components**: description of the services that should be configured at the site.
3. **runtime_variables**: detailed information required for the component repositories used by the site administrator in the lightweight_components section.
4. **supported_virtual_organizations** and **voms_config**: site-specific settings related to the virtual organizations to be supported by the services.
5. **site**: general site-specific information like security email, support email, timezone etc.
6. **preferred_tech_stack**: specific underlying technologies that the framework should use for configuration management and container orchestration.

2.1.3 Site Level Defaults

For site administrators to spend less effort in the configuration of various grid services, the framework already defines several essential configuration variables by default. These variables are listed in the site level defaults file[13] and include configuration parameters for the LHC experiments among others.

2.1.4 YAML Compiler

The YAML compiler or simply, the compiler, is a standalone component of the framework that takes the site level configuration file as input and generates an *augmented* site level configuration file with its corresponding schema as output. The augmented site level configuration file contains complete definitions of all the used configuration variables as well as meta-data related to the deployment of the grid services. The compiler supports the inclusion of information from subsidiary configuration files as well as the reuse of specific definitions to help avoid duplication of information. [4]

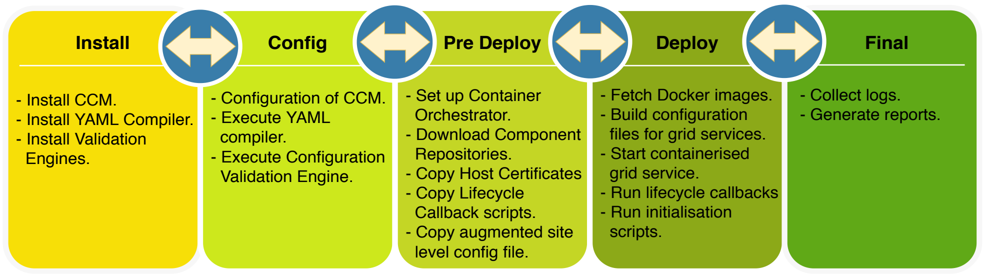
Install
Config
Pre Deploy
Deploy
Final
- Install CCM.
- Install YAML Compiler.
- Install Validation Engines.
- Configuration of CCM.
- Execute YAML compiler.
- Execute Configuration Validation Engine.
- Set up Container Orchestrator.
- Download Component Repositories.
- Copy Host Certificates
- Copy Lifecycle Callback scripts.
- Copy augmented site level config file.
- Fetch Docker images.
- Build configuration files for grid services.
- Start containerised grid service.
- Run lifecycle callbacks
- Run initialisation scripts.
- Collect logs.
- Generate reports.

Figure 1: The stages in the Execution Pipeline of the SIMPLE framework.

2.1.5 Validation Engines

In order to avoid common pitfalls (e.g. typos, missing or incompatible configuration information) associated with configuration and deployment of grid services, the framework provides validation engines responsible for validation of the configuration and the infrastructure at various stages of the execution pipeline.

The configuration validation engine ensures that the configuration parameters provided in the site level configuration file have the correct data types and that the values abide by the validation rules specified in the augmented site level configuration schema. It can help detect not only common configuration errors, but also combinations of parameters that are known not to work.

The infrastructure validation engine ensures that the lightweight component nodes meet the minimum system requirements. After the execution of each stage in the pipeline of the framework, the infrastructure validation engine also ensures that the nodes and grid services are in their expected states. Any discrepancies are reported so that the site administrator can take appropriate actions to address the underlying issues.

2.1.6 Central Configuration Manager

The Central Configuration Manager (CCM) binds all of the other components in the framework together. It ensures that functions of the various components of the framework are executed in the correct environment and in the proper sequence which is defined by the execution pipeline. The CCM also enables the site administrator to have control over the framework's execution pipeline by providing functions to traverse the execution pipeline in forward and backward directions. After the execution of a stage, any discrepancies between the intended and the observed configuration are reported by the validation engines. The site administrator can resolve such discrepancies by rolling back to the previous stage, applying an appropriate fix and re-running the current stage. This functionality helps provide a robust mechanism to handle configuration discrepancies and to ensure that the final stage is ready for production. The same functionality can be used to apply configuration changes and software updates.

2.2 Execution Pipeline

The execution pipeline of the framework is divided into five stages. In each stage, one or more functions of one or more components are executed. An overview of the stages of the execution pipeline is shown in Figure 1.

2.3 Features

The primary aim of the framework is to abstract the empirical knowledge required for deploying and operating grid services, reduce the amount of configuration required for them and help avoid common pitfalls that accompany the process, while leveraging modern and popular technologies that site administrators may already be familiar with.

The execution pipeline has been designed such that the services are production-ready in the final stage. The outcome of the execution pipeline is independent of the choice of the underlying technologies used for implementing the framework's components. This separation between technologies and functionalities allows the framework to evolve towards the use of alternative technologies internally, while keeping the high-level interfaces unchanged. Where multiple technologies are available for a particular functionality, the site administrator might prefer to use one that permits local enhancements. At present, our implementation utilizes Python for the YAML compiler and Validation Engines, Puppet for the Central Configuration Manager and Docker Swarm for container orchestration. In future releases, we aim to provide alternative implementations based on different technologies such as Kubernetes for container orchestration and Ansible for the CCM.

The framework just expects its users to have basic Linux system administrator skills. Experience with configuration management or containers is not required. Writing a site-level configuration file of just 100-300 lines of YAML may already be sufficient for a successful deployment. The YAML compiler web utility further helps with validating such files beforehand. Advanced system administrators may modify or extend the framework components as desired and propose patches for inclusion upstream. They can also make use of advanced features like the injection of scripts to provide additional functionality.

3 SIMPLE - Flow of Configuration Data

When grid service experts create component repositories, the framework allows them to indicate required configuration information through the config-schema.yaml files and guarantees that such information will be present in the augmented site level configuration file, which then is supplied to the component repositories involved. Site administrators may further tailor the configuration of component repositories by providing supplemental documented parameters, extra files or scripts that are appropriate for the grid service containers defined in the component repository. They may propose patches for incorporation upstream. Figure 2 represents these interactions between site administrators, SIMPLE framework and component repositories provided by grid service experts. The steps shown in the figure are described below:

1. Grid service experts containerize a service by creating a Dockerfile. To a certain extent, packages and services can be pre-configured in the container images.

2. Additional configuration parameters required to configure the service are indicated in meta information files.

3. These meta information files and the Dockerfile are packaged and pushed to GitHub. This is what we call the component repositories.

4. A site administrator creates a site level configuration file, starting with the hostnames and IP addresses of the lightweight component nodes, and uses the component repositories to deploy the grid services on the nodes.

5. The configuration parameters required by the meta information files of the component repositories must be provided in the site level configuration file, which will be checked by the framework.

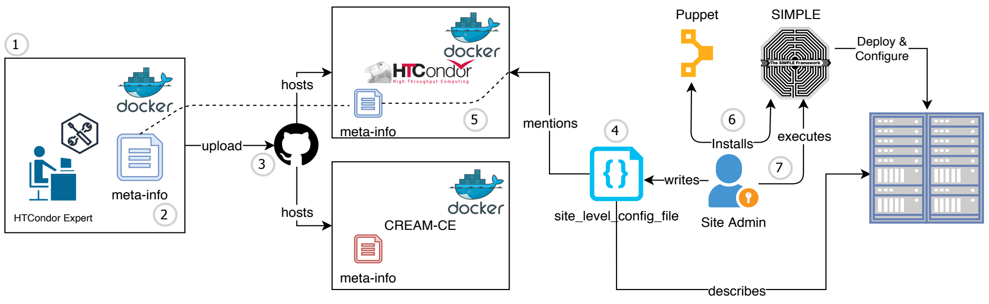

Figure 2: Flow of configuration information in the SIMPLE framework.

6. The site administrator installs the CCM, which includes Puppet and the Puppet module for the SIMPLE framework.

7. The administrator runs the framework which automates the deployment and configuration of the grid services across the specified nodes.

4 Deployments and Use Cases

The SIMPLE framework provides a useful tool e.g. for migrating computing resources from the CREAM-CE, which reaches its end-of-life in December 2020, to HTCondor-CE. Centro Brasileiro de Pesquisas Físicas (CBPF), a WLCG Tier-2 site located in Rio de Janeiro, has been an early adopter of the SIMPLE framework and a major contributor to the project. The migration of their 1,640 cores from CREAM CE and Torque to HTCondor CE and batch system was carried out by means of the SIMPLE framework.

Anticipating a similar migration, the ALICE Tier-3 site at Cibinong, Indonesia, used SIMPLE to set up a small HTCondor cluster and has investigated possible scenarios to support local users in such a setup. This is currently a work in progress.

The generic architecture of the SIMPLE framework also enables the deployment of non-WLCG services using the same machinery. This flexibility was demonstrated in deploying an analysis cluster for economics studies, featuring Apache Spark, Hadoop, Yarn, HDFS and a Jupyter notebook, for the graduation project of a student from Plekhanov Russian University of Economics, Moscow[14].

5 Conclusions

We describe how the components and the execution pipeline of the SIMPLE framework can empower site administrators to deploy and maintain production-ready grid services with just basic Linux system administration skills. We provide examples of the current production use cases of the framework, particularly the migration of computing resources at CBPF and Cibinong to the HTCondor-CE and HTCondor batch system.

We have put in effort to improve the user experience in several ways. The Web YAML compiler can help detect compilation and validation errors early as well as assist site administrators in writing the site level configuration files. A dedicated command line interface, available since March 2020, helps the administrator go through the steps required by the framework, providing specific checks and detailed logs along the way. We aim to introduce support for Kubernetes and Ansible as underlying technologies. We look forward to community contributions, particularly for increasingly popular grid services such as ARC CE[15] and Slurm[16].

References

[1] *Worldwide LHC Computing Grid*, `http://wlcg.web.cern.ch/`

[2] B. Bashari Rad, H. Bhatti, M. Ahmadi, IJCSNS International Journal of Computer Science and Network Security **173**, 8 (2017)

[3] *What is configuration management?*, `https://www.redhat.com/en/topics/automation/what-is-configuration-management`

[4] M. Sharma, M. Litmaath, E. Silva Junior, R. Santana, EPJ Web of Conferences **214**, 07019 (2019)

[5] *Powerful infrastructure automation and delivery | Puppet | Puppet.com*, `https://puppet.com/`

[6] *Ansible: Simple IT Automation*, `https://www.ansible.com/`

[7] *Swarm mode | Docker*, `https://docs.docker.com/engine/swarm/`

[8] *Kubernetes*, `https://kubernetes.io/`

[9] M. Sharma, M. Litmaath, E. Silva Junior, *SIMPLE Framework: Specification Document - Google Docs* (2018), `http://cern.ch/go/X7cr`

[10] M. Sharma, E. Silva Junior, B. Vasilev, *GitHub - simple-framework/simple_htcondor_ce: The component repository for HTCondor-CE*, `https://github.com/simple-framework/simple_htcondor_ce`

[11] M. Sharma, E. Silva Junior, B. Vasilev, *GitHub - simple-framework/simple_htcondor_batch: The central manager, collector and negotiator daemons of HTCondor are run by this component repossitory*, `https://github.com/simple-framework/simple_htcondor_batch`

[12] M. Sharma, E. Silva Junior, B. Vasilev, *GitHub - simple-framework/simple_htcondor_worker: The component repository for HTCondor Executor node*, `https://github.com/simple-framework/simple_htcondor_worker`

[13] M. Sharma, S. Sinha, A. Arisal, *GitHub - simple-framework/simple_grid_site_repo: Default variables that can be used by site_level_configuration_file in the simple_grid framework*, `https://github.com/simple-framework/simple_grid_site_repo`

[14] I. Gavrilenko, M. Sharma, M. Litmaath, T. Tikhomirova, CEUR Workshop Proceedings **2507**, 300 (2019)

[15] *NorduGrid | ARC Compute Element*, `http://www.nordugrid.org/arc/ce/`

[16] *Slurm Workload Manager - sbatch*, `https://slurm.schedmd.com/sbatch.html`