Bringing the ATLAS HammerCloud setup to the next level with containerization

Benjamin Rottler[1,*], *Michael* Böhler[1], *Günter* Duckeck[2], *Alexander* Lory[2], *Christoph Anton* Mitterer[2], and *Jaroslava* Schovancova[3]

[1]Physikalisches Institut, Albert-Ludwigs-Universität Freiburg, Freiburg; Germany
[2]Fakultät für Physik, Ludwig-Maximilians-Universität München, München; Germany
[3]CERN, Geneva; Switzerland

Abstract. HammerCloud (HC) is a testing service and framework for continuous functional tests, on-demand large-scale stress tests, and performance benchmarks. It checks the computing resources and various components of distributed systems with realistic full-chain experiment workflows.
The HammerCloud software was initially developed in Python 2. After support for Python 2 was discontinued in 2020, migration to Python 3 became vital in order to fulfill the latest security standards and to use the new CERN Single Sign-On, which requires Python 3.
The current deployment setup based on RPMs allowed a stable deployment and secure maintenance over several years of operations for the ATLAS and CMS experiments. However, the current model is not flexible enough to support an agile and rapid development process. Therefore, we have decided to use a containerization solution, and switched to industry-standard technologies and processes. Having an "easy to spawn" instance of HC enables a more agile development cycle and easier deployment. With the help of such a containerized setup, CI/CD pipelines can be integrated into the automation process as an extra layer of verification.
A quick onboarding process for new team members and communities is essential, as there is a lot of personnel rotation and a general lack of personpower. This is achieved with the container-based setup, as developers can now work locally with a quick turnaround without needing to set up a production-like environment first. These developments empower the whole community to test and prototype new ideas and deliver new types of resources or workflows to our community.

1 Introduction

HammerCloud (HC) is a framework used and developed by the ATLAS [1] and CMS [2] experiments to test, commission, and benchmark Worldwide LHC Computing Grid (WLCG) [3] resources with realistic full-chain experiment workflows. It provides the capability to both run functional tests (FTs), where the resources are tested with a steady flow of test jobs, and stress tests, where the functionality of a resource or service is tested under a configured load intensity.

*e-mail: benjamin.rottler@cern.ch

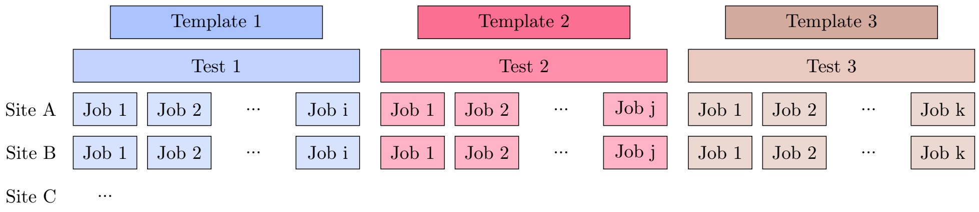


Figure 1: Representation of the relation between templates, tests, and jobs. Each template spawns a test at regular intervals. For the lifetime of the test, jobs are submitted regularly, usually such that there is one job running continuously at each site included in the template. At the end of the test lifetime all running and pending jobs are terminated.

The great benefit of using realistic full-chain experiment workflows as test jobs is that these jobs replicate the same actions, utilize identical environments, and access the exact same services as standard physics analysis jobs. Therefore, HammerCloud tests can be developed and configured in such a way that they reveal issues with the infrastructure without being affected by transient issues of the user payload.

The foundation of automated testing in HammerCloud revolves around the utilization of templates, as shown in Figure 1. A template defines the test parameters, i.e. the set of computing sites at which the test is executed, the runtime of the test, the frequency of test job submission, and the workload of the test job. For each template a test with a defined number of parallel jobs is started. Jobs are continuously dispatched to the respective sites during the lifetime of the test. Upon completion of the test, HammerCloud cancels all pending and running jobs and generates a fresh test to continue the automated testing process. For functional tests, the templates typically entail small workloads that are processed quickly and involve a significant number of sites. In contrast, stress test templates usually target only a small number of sites but have a higher intensity of jobs. Figure 1 illustrates a scenario primarily associated with functional tests rather than stress tests, i.e., it depicts a continuous flow of test jobs dispatched to numerous sites.

2 HammerCloud in ATLAS

HammerCloud is an integral part of the ATLAS Distributed Computing operations (ADC Ops) automation suite. It mainly is used to identify infrastructure issues through functional testing of the resources. The results of a subset of the functional tests are utilized to automatically exclude resources from the distributed computing system when they are malfunctioning. After the issue has been resolved, the resources are automatically included again. Furthermore, the test results are used to spot system-wide outages. Moreover, HammerCloud is used in the commissioning and integration of new resources, such as GPUs, as well as the commissioning of new components of distributed computing systems, such as the pilot for the ATLAS Workload Management System (WMS) PanDA [4]. It is also used to test new versions of the ATLAS core software. Lastly, HammerCloud is utilized to conduct performance benchmarks, where a standardized workflow such as HEPScore23 [5] is used to measure the performance of a resource. Static and dynamic information about the ATLAS resource topology is collected by HammerCloud from the Computing Resource Information Catalog (CRIC) [6] in order to create tailored tests for different types of resources. Currently, HammerCloud manages 30 distinct templates for ATLAS. As shown in Figure 2, this encompasses between 60 000 and 80 000 jobs per day. Jobs that test sites for physics analysis workflows, i.e., running a physics analysis, and central production workflows, such as the simulation of

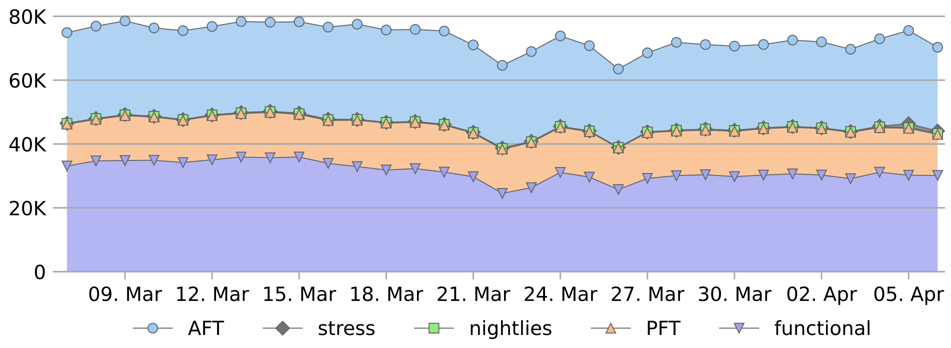


Figure 2: Total number of ATLAS jobs submitted by HammerCloud between March 7, 2023, and April 6, 2023. The number of jobs is shown in a stacked diagram and is grouped into several categories: Results of the Analysis Functional Tests (denoted as "AFTs") and Production Functional Tests (denoted as "PFTs") are used for the automatic exclusion of resources, tests for the ATLAS core software are labeled as "nightlies", remaining functional tests are grouped in the "functional" category, and stress tests are denoted as "stress".

collision events, are denoted as Analysis Functional Tests (AFTs) and Production Functional Tests (PFTs), respectively. The results of these jobs are used for the automatic exclusion of resources.

3 Architecture of HammerCloud

HammerCloud is designed as a distributed multi-node setup, comprising a collection of services. This setup consists of three distinct types of nodes.

The *core* node is responsible for central business logic like assigning tests to a submission node, performing the automatic exclusion and recovery of resources, and adding new sites to running tests. Furthermore, it collects static and dynamic information about the resource topology from CRIC and the ATLAS Distributed Data Management (DDM) system Rucio [7]. The *submission* nodes are responsible for submitting jobs to PanDA, collecting the job outcomes, and writing the results to the database. The *web server* node hosts a Django web server [8], allowing users to access the status of ongoing and previous tests. Additionally, the Django web server enables HammerCloud operators to schedule new tests and to define new templates. The whole setup is backed by a MySQL [9] database, where all configuration and test results are stored. The submission nodes are organized as a Celery cluster [10], an asynchronous task queue supported by a Redis database [11]. This arrangement facilitates the distribution of resource-intensive tasks, like interactions with PanDA, among multiple nodes. It allows for an improved resource efficiency and the simultaneous execution of time-consuming operations. An overview of the architecture of the HammerCloud service with an example of a test submission is shown in Figure 3. Currently, the configuration comprises one core node, one web server, and eight submission nodes.

HammerCloud has been initially developed using Python 2.7 in conjunction with Django 1.1 for the web server component. The migration to Python 3 and Django 4 is currently ongoing. For deploying new versions, RPM packages are directly installed onto virtual machines provided by CERN IT.

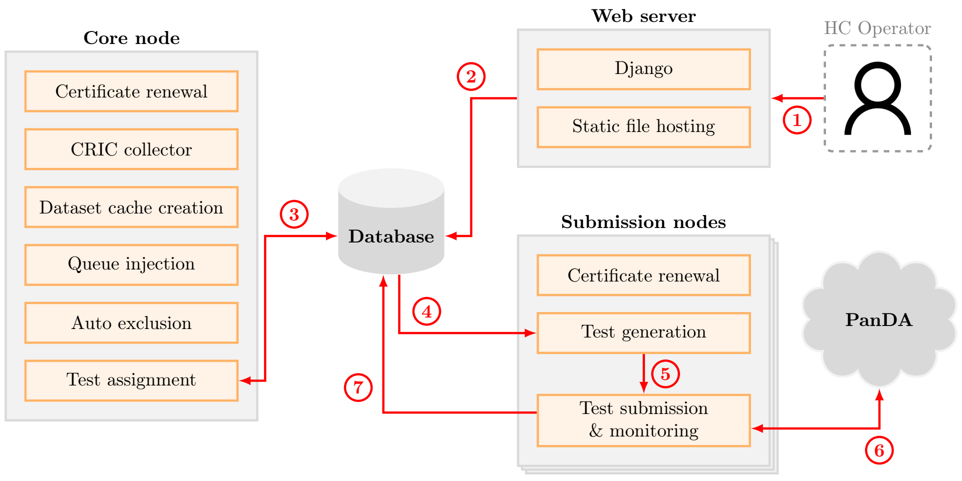


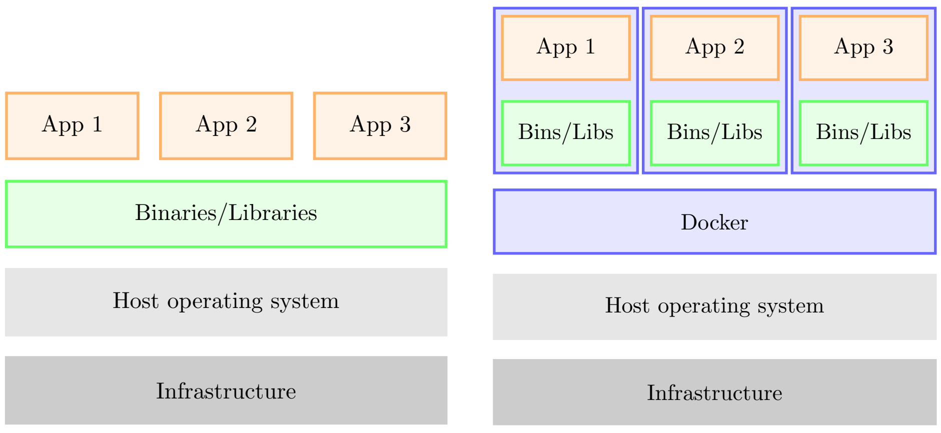
Figure 3: Overview of the HammerCloud architecture. The core node is responsible for most of the business logic like test assignment, auto exclusion, and collection of static and dynamic information about the resource topology. Job submission and monitoring is handled by the submission nodes. New tests and templates are defined and scheduled with the web server. The red lines show the flow of a test submission. 1 + 2: A HammerCloud operator schedules a test via the interface provided by the web server. 3: The core node assigns the test to a submission node. 4 + 5: The submission node prepares for job submission by generating all required configuration files. 6 + 7: Jobs are submitted to the ATLAS Workload Management System PanDA. The jobs are being monitored, and the results are written into the central database, where they can be accessed via the web server.

4 Containerization of HammerCloud

The primary drawback of the current setup is that it relies on RPMs being directly deployed onto virtual machines. This limits the utilization of binaries and libraries to only those specific versions that are provided by the host operating system, as depicted in Figure 4 (a). Moreover, this setup necessitates careful management of compatibility among various HammerCloud services that rely on these shared binaries and libraries. Furthermore, dedicated machines are required specifically for development purposes.

Using a containerized setup, i.e. HammerCloud services running inside Docker containers [12], offers various advantages over the current setup. For example, a container provides an isolated environment with its own dependencies, allowing for greater flexibility, as shown in Figure 4 (b). This makes the current migration of HammerCloud from Python 2.7 and Django 1.1 to Python 3 and Django 4 much easier. Additionally, the adoption of containers allows for a more agile development process by eliminating the need for dedicated development machines. Development can then be conducted locally on personal computers or laptops by spinning up a local instance of the container for testing. Two of these local development setups are detailed in the next sections. Furthermore, containerization enables the application of concepts such as continuous integration (CI) and continuous delivery (CD). This involves executing unit tests and integration tests with every revision made in the version control system, automatically generating the HammerCloud Docker containers, and deploying these containers seamlessly into both testing and production environments.

During the implementation of the container solution for HammerCloud, the Docker philosophy is adhered to, which advocates assigning a single responsibility to each container



(a) Bare metal. (b) Containers.

Figure 4: Comparison between a bare-metal setup (a) and a containerized setup (b). In the bare-metal setup, binaries and libraries are shared among various applications, operating directly on the host operating system. The containerized setup introduces an additional layer between the host operating system and the applications. Each application within a container possesses its individual set of binaries and libraries, offering enhanced isolation and encapsulation.

instance. Consequently, the consolidation of all services associated with a specific node type, i.e., core, submission, and web, into a single common container instance is avoided. Instead, each service operates within its dedicated container instance. This approach grants us finer control over individual services, facilitating effective management and customization.

4.1 Web server development setup

The simple three-container setup illustrated in Figure 5 allows for local development of the web server. Here, the central component is the container that hosts the Django web server. A second container is used to operate a local MySQL database, because the Django web server requires a database that it can connect to. To closely emulate the production environment, a Nginx [13] server is deployed in a third container. It acts as a reverse proxy, i.e., it forwards the HTTP requests from the developer to the Django web server, while also serving static files directly.

All containers can be easily instantiated using `docker compose`. A simplified version of the corresponding `docker-compose.yml` configuration file is shown in Figure 6. Configuration files and static files are persistently stored on the Docker host, and bind mounts are used to include them within the container instances.

4.2 Test submission development setup

The configuration for the local development of the test submission loop is depicted in Figure 7. The same three containers utilized in the web server development setup are employed, but the setup is expanded with additional containers. Firstly, two container instances are established to run the services responsible for collecting information from CRIC and Rucio, respectively. Moreover, a container dedicated to the test assignment service is created, which

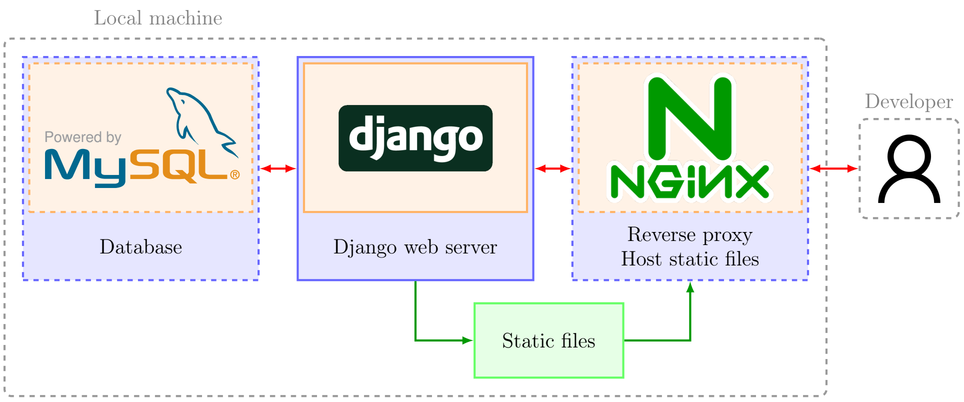


Figure 5: Local development setup for the HammerCloud web server based on three containers. The web server container is running a Django instance, which is connected to a local database that is running in a second container instance. In addition, a Nginx container is used as reverse proxy and to host static files, that are generated by Django. The static files are stored persistently on the local machine and are included in the web server and Nginx container instances via bind mounts. Containers running HammerCloud services are shown with a solid border, while containers running third-party services have a dashed border. In production, the database container is replaced by a MySQL database provided by CERN-IT.

handles the task of assigning new tests to a specific submission node. The central element of a submission node is the test generation service, which is now running in its own container. This service monitors for new tests that are assigned to the node on which the service operates on. For each newly assigned test, the test generation service starts a fresh container instance that then runs the logic for test submission and monitoring.

5 Conclusions

HammerCloud, a versatile framework utilized for monitoring, commissioning, and benchmarking WLCG resources, has been presented within the context of the ATLAS Experiment. Furthermore, the architecture of HammerCloud has been discussed. In the ever-changing landscape of software development it becomes imperative to make a transition from the current RPM-based setup to a more contemporary approach centered around containerization. The advantages of containerization have been discussed, encompassing local development capabilities and the adoption of agile development processes like continuous integration and continuous deployment (CI/CD). Furthermore, two distinct container-based setups have been presented: one for the local development of the web server and another one for the test submission loop. These setups exemplify the flexibility and adaptability that containerization brings to the HammerCloud framework, enabling efficient and streamlined development workflows.

```
services:
  web:
    build: .
    image: hammercloud-atlas-docker
    expose:
      - 8000
    depends_on:
      - "db"
      - "nginx"
    env_file:
      - ./.env.dev.web
    volumes:
      - ./files/config:/data/hc/apps/atlas/config
      - web-static:/data/hc/web/static
  nginx:
    build: ./nginx
    ports:
      - 8000:80
    volumes:
      - web-static:/data/hc/web/static
  db:
    image: "mysql"
    volumes:
      - db-data:/var/lib/mysql
    env_file:
      - ./.env.dev.mysql
volumes:
  web-static:
  db-data:
```

Figure 6: Simplified `docker-compose.yml` configuration file of the local development setup for the HammerCloud web server.

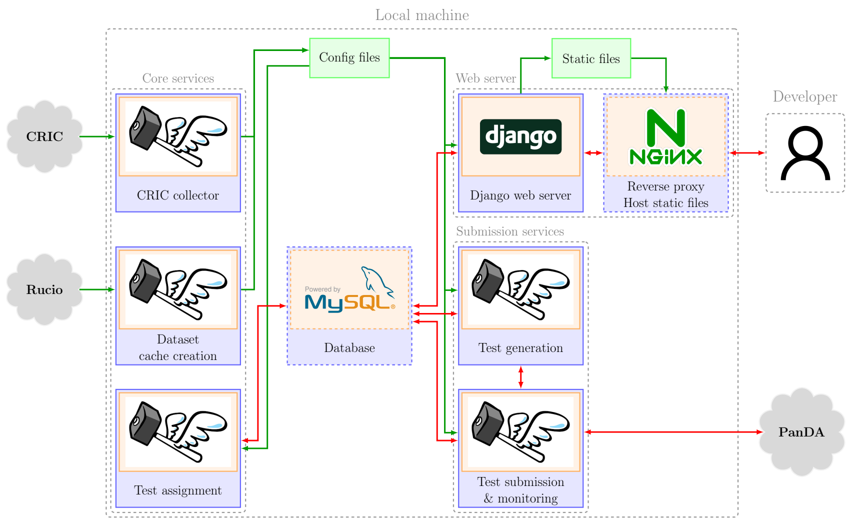


Figure 7: Local development setup for the test submission loop. The core services are shown on the left side, encompassing two services responsible for gathering data from the Computing Resource Information Catalog (CRIC) and the ATLAS Distributed Data Management (DDM) system Rucio, alongside the test assignment service. The collected data is utilized to configure other container instances within the setup. For test submission, a container providing the test generation service is established. Each new test triggers the creation of a separate container that runs the test submission and monitoring logic. Interaction with the setup is enabled through the Django web server, connected to a Nginx server acting as a reverse proxy. The flow of configuration files is represented by green lines, while red lines depict the flow of information for the submission of a new test. Containers running HammerCloud services are shown with a solid border, while containers running third-party services have a dashed border. In production, the database container is replaced by a MySQL database provided by CERN-IT.

References

[1] ATLAS Collaboration, JINST **3**, S08003 (2008)
[2] CMS Collaboration, JINST **3**, S08004 (2008)
[3] WLCG, https://wlcg.web.cern.ch, accessed 6th July 2023
[4] ATLAS Collaboration, J. Phys. Conf. Ser. **898**, 052002 (2017)
[5] HEPiX, HEPScore23, https://w3.hepix.org/benchmarking.html, accessed 6th July 2023
[6] A. Anisenkov, J. Andreeva, A. Di Girolamo, P. Paparrigopoulos, B. Vasilev, EPJ Web Conf. **245**, 03032 (2020)
[7] M. Barisits, T. Beermann, F. Berghaus, B. Bockelman, J. Bogado, D. Cameron, D. Christidis, D. Ciangottini, G. Dimitrov, M. Elsing et al., Computing and Software for Big Science **3**, 11 (2019)
[8] Django Software Foundation, https://djangoproject.com, accessed 6th July 2023
[9] MySQL database, https://www.mysql.com, accessed 6th July 2023
[10] Celery: Distributed Task Queue, http://www.celeryproject.org, accessed 6th July 2023
[11] Redis, https://redis.io, accessed 6th July 2023
[12] D. Merkel, Linux Journal **2014**, 2 (2014)
[13] W. Reese, Linux Journal **2008** (2008)