# ATLAS open database processing task

## Abstract

The successful deployment of a multiservice application for distributed processing of ATLAS open data has been demonstrated, with the capacity for scaling both the data reading and data processing steps. Communication between services has been implemented using a message broker, RabbitMQ, acting as a data transfer manager to allow for scaling of communication between services. A discussion of application scaling has been presented, including the evaluation of more advanced orchestration tools, and the use of commercial cluster services. This solution acts as a base to showcase the benefits of implementing a distributed approach to the Higgs Boson decay analysis, designed to scale well with increasing data volumes as well as with increasing availability of resources. Finally, limitations and subsequent improvements are proposed to further increase the efficiency of the processing script in a distributed context.

## Introduction

Over the past decade, there has been a massive effort in particle physics to expand the frontiers of knowledge and investigate phenomena involving the basic building blocks of matter. The A Toroidal LHC Apparatus (ATLAS) experiment was designed to leverage the power of the world's largest particle accelerator, the large hadron collider (LHC), to observe particle collisions in a multinational scientific effort. The magnitude of the ATLAS experiment allows for the individual capture and identification of particles colliding with energies up to 7 trillion eV. Using six different detecting subsystems wrapped concentrically around the collision point, collision debris is analyzed to investigate physics theories [1]. With billions of collisions occurring per second, the data generated from the detector is screened and interactions of interest are flagged for further study to reduce the load. However, even with a three level trigger data processing system, the volume of data required for the investigation of particles of interest is immense.

An example of the success of the ATLAS experiment is the discovery of the Higgs boson, initially proposed by Peter Higgs [2], Francois Englert and other theorists in 1964 as an additional elementary particle in the standard model of particle physics to explain the existence of mass. The proof for their claim was found in 2012, towards the end of the LHC's initial operational run, through a collaboration between the ATLAS and the compact muon solenoid (CMS) experiments. This was followed briskly by a Nobel Prize in Physics for the discovery of the Higgs boson in 2013 emphasizing the profound impact within the scientific community. As the data processing capacity increases, so does the probability of observing an interaction of interest, therefore recent advances in cloud computing have drawn interest from nuclear researchers as a means of increasing the processing efficiency. In 2011, the ATLAS experiment began its transition by initiating a cloud computing research and development project to explore the integration of cloud computing paradigms in the context of ATLAS' preexisting distributed computing model (ADC) [3].

On the back of successful investigatory applications, preparations were made to develop and integrate cloud-based operations by LHC run 2 with a pilot project on Amazon's commercial EC2 service at Brookhaven National Laboratory in 2014. Scaling tests showed successful completion of 437,000 event calculations in a 40,000-core scaling run using 8-core VMs over a week [4]. Further tests encountered

funding limitations and difficulties in resource procurement resulting in slowed integration, nonetheless, commercial cloud computing resources accounted for 14.4 % of all simulation event production in 2017 [5], a number expected to increase over the coming years. This contribution from cloud computing is not small, and with the expansion of commercially available cloud technologies holds great promise for scaling the data processing capacity of particle collision experiments such as ATLAS. Cloud resources provide an additional and flexible tool, on top of pledged resources from worldwide LHC computing grid (WLCG) sites, to expand the processing capacity, with the possibility for dynamic allocation to meet demand as needed [6].

With the clear advantages of utilizing cloud computing paradigms in data processing, a shift to cloud oriented design is occurring for future processing workflows. This project aims to demonstrate, using a locally hosted proof of concept, the benefits of adapting a sample processing script to use cloud computing technologies and evaluate the impact of distribution on the scalability of the analysis. This is achieved by implementing a cloud-based solution using Docker containers and the Docker swarm engine to create a packaged data processing application which can be deployed with ease over a network of nodes.

# Methods

To distribute the solution using cloud technology, the data processing script was decomposed into three tasks, one to retrieve the data, one for data processing, and the last to output the results. Each service was designed such that they could be run independently of each other thereby allowing them to run on individual nodes in a system. The solution was containerized using Docker to ensure that each service could be built on any computer regardless of operating system, and that the prerequisites of each service were installed with minimal interaction from the user. Docker containers are a lightweight and flexible solution in comparison to virtual machines (VM) with the separation of processes making it easier to update individual parts of the processing workflow [8]. They allow for design that is not centered around machine-specific configuration, moreover, they go hand in hand with a microservices design model.

The Docker swarm engine was used for container orchestration to demonstrate the capabilities of service distribution on the Docker network. While more controlled approaches such as Kubernetes are available, which allow for more hands-on distribution and load-balancing [9], Docker swarm was chosen due to its adaptability, quick scaling and testing capacity. It offers a streamlined, easy to use alternative for smaller applications, with automated load balancing and direct integration with the docker environment. Communication between services was implemented using a RabbitMQ message broker service, acting as a middleman, routing messages and ensuring they reach the correct destination [10]. This decoupling of services from each other allows them to operate with no direct interaction with each other, allowing for scaling of tasks independently.

# Results and discussion

## *Solution overview*

The product application provides a scalable implementation of the Higgs boson analysis script using cloud technologies to distribute tasks amongst available resources, thereby leveraging parallelization to achieve more efficient processing. Figure 1 shows the tech stack for this solution, with the workflow beginning with the reading of data from the ATLAS open database. The processing step was chosen for parallelization

of the algorithm, with the processing service being replicable to increase processing capacity with available resources. The balance of work was considered by ensuring that workers completed their assigned task before moving onto the next, meaning that tasks are taken by the first available worker thereby minimizing their downtime. Outputs were collected by an output postprocessing service, combining data from individual workers into a single data set for final plotting and output. Finally, the output was presented using a Jupyter notebook webserver, showing a plot simulated events.

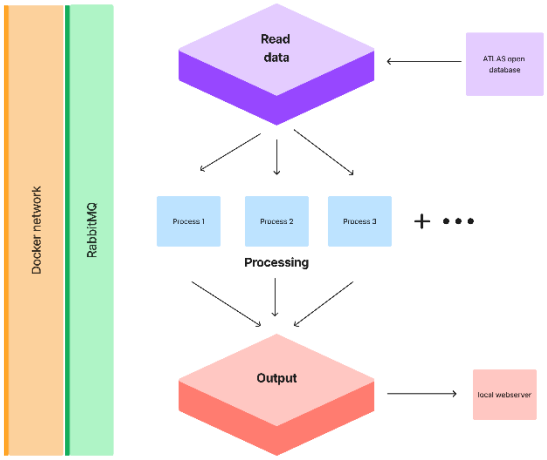


*Figure 1: Tech stack of HZZ analysis cloud processing solution*

Initially, communication between services was limited to writing data to files and reading those files from another service, this resulted in race conditions on files, lengthy write times and increased storage requirements for each service. A message broker, RabbitMQ, was used to create a message exchange and dedicated queues for messaging between different services, greatly simplifying data transfer between services. RabbitMQ acts as a workflow manager within this analysis, with the start of certain processes dictated using signaling along message queues. A summary of message queues and data flow within the analysis app is shown below, see figure 2, picturing the exchange of information within the workflow. While the docker swarm engine typically handles load balancing, in the context of RabbitMQ message queues, it is the message broker that distributes the messages to the first available worker therefore handling most of the load balancing in this case. Data volumes were used for the output of results as a figure to the user's directory, the plot is displayed in the form of a Jupyter notebook, displayed on a web page as a html file. The graphical summary of the analysis script gives the user a convenient way to analyze the results, with the possibility of further exploring the data themselves.

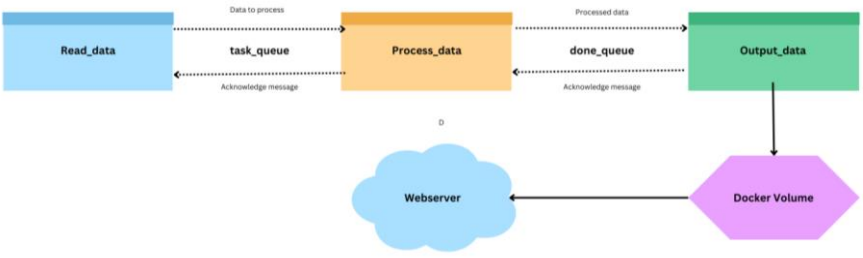


*Figure 2: Communication diagram within HZZ analysis*

The solution is designed to minimize the setup required by the user to allow for seamless processing from a remote data file. The code base was maintained on a GitHub repository, both to aid in distribution of the application, and to allow application development to occur on multiple machines.

## *Scaling*

The solution presented supports scaling by increasing the number of replicas created for each service. This is envisioned to be used on the read_data and process_data services, which could benefit greatly from scaling when increasing the data volume. The docker swarm engine gives the ability to scale a deployed docker stack using inbuilt functions, adjusting for load balancing and handling additional communications automatically. The parallelization of the data reading opens the possibility of processing multiple data files simultaneously, reducing the penalty for fetching data from a URL. Moreover, the distribution of the processing steps results in greatly reducing the bottleneck presented by the large number of calculations being performed when considering large volumes of data. The dynamic nature of scaling using docker allows for adjustments to be made on the fly, considering demand at the time.

Considering large scale deployment, the solution is ready to deploy to a cluster, this could be on local VM networks or commercial cluster services supporting docker swarm such as AWS EC2 or Microsoft Azure [11,12]. Using docker swarm mode, a leader node is designated, and additional nodes join the network as workers. The docker swarm engine will automatically assign work from the stack to separate nodes, efficiently managing the workload and maintaining high availability and fault tolerance. RabbitMQ plays a crucial role in scalability of communication between services, performing automated adjustments upon the addition of new workers in the network. Commercial services are a much easier path for the deployment of an application in comparison to clustering VMs locally, due to the automated setup and management of cluster resources, resulting in much more streamlined deployment. Despite the high availability of cloud computing resources, the cost-effectiveness of solutions should also be considered, with the creation and management of clusters often coming at a high price

Scaling by the introduction of greater volumes of data will require adjustment of the utils.py script with information about the samples in the new data set if the set contains different samples. Further changes are required in the get_data.py script to look for files in a new location, and minor adjustments to allow for the processing of multiple files using additional calls to the read_data function, which will run the processing. The data processing capacity can also be expanded upon by creating additional clusters, independent of each other and running these in parallel to process the data, with each cluster essentially acting as an individual rather than creating one large cluster with a lot of work. With a more complex cluster network, the need arises for additional controls regarding load balancing and networking, Kubernetes stands out as the leading container orchestration toolbox for larger infrastructures.

Providing extensive network features as well as automated service discovery, security and self-healing functions, Kubernetes holds great potential in the context of increasing the scalability of the presented solution [13]. Moreover, Kubernetes can manage multiple clusters in parallel which would greatly help in achieving the data processing capacity expansion previously mentioned. Another option for the distribution of the services on commercial clusters is to deploy using services provided by the chosen cloud solution, for example, Amazon elastic container service (ECS) which provides managed orchestration. Managed clusters and orchestration services have some of the simplest deployment and management requirements with the majority of configuration managed by the cluster provider making them ideal choices for hands-off deployment of a solution to scale.

### *Critical analysis*

A limitation of the proposed approach lies in the splitting of data over processing services, a data file containing samples of multiple data points is split by sample and sent for processing. With samples containing multiple data points, there is potential for further parallelization to remove the dependency of the maximum number of processing service replicas on the number of samples in the data file. Secondly, while the transfer of data by message broker is undoubtedly more efficient than writing to files using docker volumes, further investigation should be conducted into the most efficient methods of data transfer. Improvements could be made by combining individual transfers where possible, additionally, data could be compressed prior to being sent, reducing the overhead of TCP/IP transmission. Finally, creating an interactive webpage, perhaps using tools such as iPyWidgets to create a graphical user interface, from which the user could re-run the analysis and control the number of service replicas would increase the accessibility of the application and allow for reproducibility of results, or further observations.

While the proposed approach exhibits certain limitations, it is important to recognize the vast potential for improvement and optimization using cloud technologies. The application presented acts as a proof-of-concept demonstration, showing the initial application of distributed programming using cloud paradigms such as services and deployment on cluster networks, and their benefits. With the constant drive for innovation in the scientific community, the demand, and in turn, supply for cloud technologies will undoubtedly grow, making them both more cost-effective and accessible. By leveraging advances in cloud technologies such as Kubernetes, extremely complex applications can be maintained, deployed and managed; resulting in solutions with improved resilience with great scalability. To conclude, the chosen cloud technologies work well for the scope of this project, with more complex tools available for further optimization in deployment. The resulting application is scalable to large volumes of data, and adaptable to resources available.

## References


1. ATLAS website available at: https://atlas.cern/about
2. P Higgs*. "Broken Symmetries and the Masses of Gauge Bosons". Physical Review Letters.*1964 ***13*** *(16): 508–509.*
3. Sergey Panitkin *et al* “ATLAS cloud R&D” *J. Phys.: Conf. Ser.* **2014** 513 062037
4. R. P. Taylor, C. J. D. Cordeiro, D. Giordano, J. Hover, T. Kouba, P. Love, A. McNab, J. Schovancova, R. Sobie, on behalf of the ATLAS Collaboration, "*Consolidation of cloud computing in ATLAS*," *J. Phys.: Conf. Ser.,* **2015,** *664,* 022038.
5. Ryan P Taylor et al “the evolution of cloud computing in ATLAS” J. Phys.: Conf. Ser. 2015 664 022038
6. F. B. Megino, on behalf of the ATLAS experiment, “*Accelerating science: the usage of commercial clouds in ATLAS Distributed Computing”,* presented at CHEP **2023**, Norfolk, USA
7. F. B. Megino, H. S. Bawa, K. De, J. Elmsheuser, A. Klimentov, M. Lassnig, C. Serfon, T. Wegner, “*Seamless integration of commercial Clouds with ATLAS Distributed Computing”* **2021**
8. Merkel D. Docker: lightweight linux containers for consistent development and deployment. Linux journal. 2014;2014(239):2.
9. Kubernetes documentation available at:https://kubernetes.io/docs/home/ -v1.28
10. RabbitMQ documentation “Complete reference guide” available at: https://www.rabbitmq.com/amqp-0-9-1-reference.html

11. Amazon web services documentation available at: https://docs.aws.amazon.com/
12. Azure documentation available at: https://azure.microsoft.com/en-us/get-started/
13. F. H. B. Megino, J. R. Albert, F. Berghaus, K. De, F. Lin, D. MacDonell, T. Maeno, R. B. Da Rocha, R. Seuster, R. P. Taylor, M.-J. Yang, "Using kubernetes as an ATLAS computing site" *EPJ Web Conf.,* **2020,** *245,* 07025. https://doi.org/10.1051/epjconf/202024507025

## Appendix

Code archive: https://github.com/kon-218/ATLAS-cloud-processing

Please check Readme.md for important information on running the application and deployment to a swarm.