Gravitational wave alert generation infrastructure on your laptop

Sara Vallero[1*], *Roberto* De Pietri[2], *Rhys* Poulton[3], *Pierre* Chanial[4] , *Alessio* Fiori[5] and *Daniele* Monteleone[1]

[1]INFN Torino, Via Pietro Giuria 1, 10125 Torino, Italy

[2]INFN gruppo collegato di Parma and Universià di Parma, 43124 PARMA, Italy

[3]European Gravitational Observatory (EGO), I-56021 Cascina, Pisa, Italy

[4]Université Paris Cité, CNRS, AstroParticule et Cosmologie, F-75013 Paris, France

[5]INFN Sezione di Pisa, L.go B. Pontecorvo 3, 56127 Pisa, Italy

Abstract. Multi-messenger astrophysics provides valuable insights into the properties of the physical Universe. These insights arise from the complementary information carried by photons, gravitational waves, neutrinos and cosmic rays about individual cosmic sources and source populations. When a gravitational wave (GW) candidate is identified by the Ligo, Virgo and Kagra (LVK) observatory network, an alert is sent to astronomers in order to search for electromagnetic or neutrino counterparts. The current LVK framework for alert generation consists of the Gravitational-Wave Candidate Event Database (GraceDB), which provides a centralized location for aggregating and retrieving information about candidate GW events, the SCiMMA Hopskotch server (a publish-subscribe messaging system) and GWCelery (a package for annotating and orchestrating alerts). The first two services are deployed in the Cloud (Amazon Web Services), while the latter runs on dedicated physical resources. In this work, we propose a deployment strategy for the alert generation framework as a whole, based on Kubernetes. We present a set of tools (in the form of Helm charts, Python packages and scripts) which conveniently allows running a parallel deployment of the complete infrastructure in a private Cloud for scientific computing (the Cloud at CNAF, INFN Tier-1 Computing Centre), which is currently used for integration tests. As an outcome of this work, we deliver to the community a specific configuration option for a sandboxed deployment on Minikube, which can be used to test the integration of other components (i.e. the low-latency pipelines for the detection of the GW candidate) with the alert generation infrastructure in an isolated local environment.

1 Introduction

Multi-messenger astrophysics [1] provides valuable insights into the properties of the physical Universe. These insights arise from the complementary information carried by photons, gravitational waves, neutrinos and cosmic rays about individual cosmic sources and source populations. When a gravitational wave (GW) candidate is identified by the Ligo [2], Virgo [3] and Kagra [4] (LVK) observatory network, an alert is sent to astronomers in order to search for electromagnetic or neutrino counterparts. This is achieved via the Low Latency Alert Generation Infrastructure (LLAI).

* Corresponding author: sara.vallero@to.infn.it

The goal of this work is to deliver to the LVK community a strategy for a sandboxed deployment of the LLAI as a whole. The proposed solution can be used to develop and test single LLAI components in an isolated local environment and to test the integration of other components (i.e., the search pipelines that look for a GW signal in incoming data) with the LLAI.

In the remainder of this section, we are going to describe how GWs are detected by means of laser interferometers and the architecture of the LLAI. Section 2 describes the *Mock Event Generator* [5], a package developed within the context of this work that is used to upload real past GW events on a given GraceDB instance. Section 3 describes our strategy for a sandboxed deployment of the LLAI on Kubernetes. In Section 4 we focus on a specific application of this work: the automated acceptance tests of one of the LLAI components.

1.1 Gravitational waves detection

Rapidly changing gravitational fields, like black hole mergers, generate gravitational waves. Gravitational waves travel at the speed of light and manifest as ripples of curvature in the fabric of space-time, transverse to the propagation direction [6]. Their effect is to stretch and squeeze the space between test masses, but since the effect is very small, to be able to observe them, the originating system should involve huge masses (i.e. 30 solar masses), relativistic velocities (i.e. 0.6 c) and short distances (i.e. 1.4 billion light years).

Gravitational waves are detected by means of laser interferometers (like Ligo, Virgo and Kagra), which are modified and enhanced versions of the Michelson interferometer. Each arm of the interferometer ends with a suspended mirror that reflects light and acts as a test mass. The passage of a GW creates a difference in length of the orthogonal arms, which translates into a phase difference between the laser beams circulating in the two arms. The optical signal measured at the detector output is proportional to the GW *strain* (relative difference in length of the orthogonal arms) [2].

An array of detectors (currently Ligo Hanford, Ligo Livingston and Virgo) is necessary to localize the GW source via triangulation and to rule out transient noise, which could mimic the GW signal but it's very unlikely to appear at the same time in detectors positioned at separate geographical locations.

The output signals of the interferometers array are processed by search pipelines, which are able to identify a GW within ~ 1 minute from the arrival of the data [7]. These pipelines provide GW candidates, which are the input data products for the LLAI described in the next subsection.

1.2 The low-latency alert generation infrastructure

The LVK Low Latency Alert Generation Infrastructure (LLAI) is responsible for issuing astronomical alerts corresponding to GW transient sources (i.e. the merger of a compact binary system of black holes or neutron stars). The system receives as input candidate GW events from search pipelines. As depicted in Figure 1, candidate GW events go through four sequential processing phases:

- *event enrichment:* computationally cheap tasks like checking on detector status or data quality that can be performed on each incoming event
- *event aggregation:* grouping of events possibly related to the same astrophysical source into a *superevent*
- *superevent enrichment:* computationally intensive tasks like source classification and sky localisation
- *vetting:* human or automatic decision concerning the publication or retraction of a GW detection.

The main components of the LLAI (namely the ones taken into account by this work) are:

- *The Gravitational-Wave Candidate Event Database (GraceDB):* provides a centralized location for aggregating and retrieving information about candidate GW events. It's a Django [9] application, integrated with an authentication service (Shibboleth [10]), a backend web server (Gunicorn [11]), a frontend web server (Apache [12]) and a backend database (PostgreSQL [13]).
- *GWCelery* [14]*:* is the service responsible for the enrichment and aggregation tasks described above. It is based on the Celery [15] distributed task queue and relies on Redis [16] as an internal database.
- *The SCiMMA Hopskotch server* [17]*:* it's a publish-subscribe messaging system based on Apache Kafka [18] and integrated with a custom authentication/authorisation layer.

The (simplified) workflow involving the three components above is the following: when a new event is uploaded by search pipelines to GraceDB, the database posts a message in a dedicated Hopskotch server queue (Kafka topic). One of the GWCelery workers constantly listens to that channel, starts processing any new event and eventually posts the *superevent* or any other required annotation to GraceDB. GWCelery is also responsible for sending the public alert via the Hopskotch server (on a dedicated topic). Alerts are also sent via the NASA General Coordinates Network (GCN) [19], but this aspect is not covered by the work presented here. Alerts are sent both in the form of machine readable public notices and via plain text. The completion of the main GWCelery actions is registered on GraceDB as a *label* associated with the corresponding *superevent* entry.

2 The Mock Event Generator

The Mock Event Generator (MEG) [5] package regenerates past gravitational wave pipeline events by time-translating them. It fetches already existing events from a GraceDB server into a cache, shifts the time references as if they were created now and sends a request to the GraceDB server to create a new event. By specifying a *superevent* id, all the online events that are part of it are recreated, either all at the same time, in a specific time interval, or as they were originally uploaded to GraceDB. This allows the possibility to recreate and upload the sequence of triggers, associated with detection, that would be generated by online search pipelines together with their specific delays.

3 Kubernetes deployment of the low-latency alert generation infrastructure

In this work, we deploy each of the LLAI components, described in the previous section, as a Kubernetes [19] application. For each component, we have prepared a separate Helm [20] chart. User documentation can be found in [21].

The GraceDB chart installs three Kubernetes *StatefulSets* [19]: one for the web application, one for Shibboleth and one for PostrgeSQL. A Traefik [22] *IngressController* [19] handles the routing to the web and authentication services. GraceDB can be accessed either via a browser or programmatically through an API. In the latter case, a client certificate needs to be provided (no Shibboleth authentication) and the Traefik controller is properly configured to accept certificates signed by the certification authority used by Ligo (InCommon [23]). The application can be installed either in a distributed or in a single instance mode. In the first case, the Kubernetes cluster should provide a *StorageClass* [19] of type *ReadWriteMany* (RWX) used to create the volume where the files uploaded to GraceDB are stored. Moreover, it is possible to choose a configuration option for a sandboxed deployment, in which a self-signed server certificate is automatically created leveraging *CertManager* [24], login to the web UI is provided by username/password (no Shibboleth authentication) but users can still be imported from the Ligo LDAP [25] for programmatic access.

The GWCelery chart has a dependency on the Redis chart distributed by Bitnami [26]. The GWCelery application itself is installed as a set of Kubernetes *Deployments* [19], one for each of the Celery workers, besides the worker running the sky localisation which is installed as a *StatefulSet* to guarantee persistence of the volume holding the reference waveform data needed for the computation. For one of the Celery workers, an additional *SidecarContainer* [19] is deployed to import some data needed for performing checks and validation on the flux of real-time streamed detector data from an external Kafka instance. Moreover, two additional *Deployments* are started: one for the Flower [27] monitoring of the Celery tasks and one for a Flask [28] application that provides forms to manually initiate certain tasks.

The Hopskotch chart installs a simple *Deployment* based on a Docker container prepared by the SCiMMA organization [29]. The container includes Kafka, Zookeeper [30] and the SCiMMA server processes. We consider a possible future activity: the refactoring of the Hopskotch installation in a microservice-oriented fashion in which these processes are run as separate containers.

Figure 1 shows a schematic representation of the LLAI deployment on Kubernetes. The different applications are connected over a private overlay network (L3) provided by Kubernetes. Within each application, coloured boxes represent either:

- a specific container type like Shibboleth (SSO or Single Sign On), Django (Web) or the various Celery workers
- a standalone service like PostgreSQL, Redis or the Kafka data importer
- the shared storage

The Helm chart code for each of the services above belongs to a different GitLab project. The Continuous Integration (CI) pipeline of each of the projects builds the chart

and uploads it to a central repository whenever a *tag* is created. The Helm repository is hosted on GitLab and has been implemented following [31].

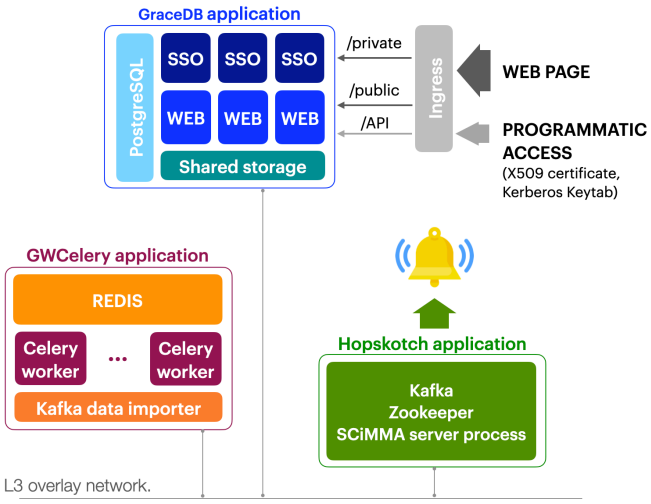

Fig. 1. Schematic representation of the LLAI deployment on Kubernetes. Coloured boxes represent either a specific container type, a standalone service or the shared storage.

3.1 CNAF and Minikube deployments

Two instances of GraceDB are currently running at CNAF [32]: one for production and one for development. In the production deployment, TLS termination is secured by an InCommon certificate and the use of a client certificate for API access. The development instance has a sandboxed configuration as described in the previous section.

The production deployment leverages the SCiMMA Hopskotch server on a dedicated set of topics and is used in conjunction with a test instance of GWCelery deployed on bare metal at the California Institute of Technology (on which the GWCelery librarians actively develop and test new features). The development instance leverages local installations of the Hopskotch server and GWCelery deployed on Kubernetes as described in the previous section. This development LLAI instance is used to test new features in the Kubernetes deployment strategy.

The performance of the production and development LLAI deployments at CNAF is monitored via a Prometheus [33] and Grafana [34] stack installed on the same Kubernetes infrastructure. Besides monitoring regular container metrics such as CPU, memory and storage consumption of the different components, for the development instance, we also import metrics on Celery task execution times and success rates from Flower.

Moreover, we provide a specific configuration option for a sandboxed deployment on a local machine using Minikube [35]. Minikube is a local single-worker Kubernetes cluster made for development and testing purposes. It relies on Docker or similar Virtual Machine engines. The setup has been tested on Linux and macOS. In this case, GraceDB uses a self-signed certificate and we provide instructions on how to reach the service from the local machine using a meaningful DNS name and to import the self-signed CA certificate in the

local browser for a simplified user experience. The installation of all the LLAI services (GraceDB, the Hopskotch server and GWCelery) requires at least 8 CPU cores and 16 GiB of RAM.

This sandboxed deployment can be useful for local development of the LLAI components in a scenario in which the component that needs to be developed is deployed on bare metal and the other components are easily installed on Minikube. Indeed, the different LLAI components are tightly interconnected and a standalone deployment would provide little insight in the correct behavior of the component in its ecosystem.

Moreover, the Minikube deployment in conjunction with MEG is being used for debugging the LLAI workflow by replaying corner case *superevents* in which the expected workflow was not executed correctly.

4 Use-case: automated acceptance tests

We have prepared an integration GitLab project that automatically spins-up the full LLAI deployment on a Kubernetes cluster and performs automated acceptance tests of the GWCelery application. These tests consist of checking for the correct creation of the *superevent* and validating all the files uploaded and the labels applied by GWCelery to the new *superevent*. This is achieved via the project's CI pipeline running on the Kubernetes cluster at CNAF. The installation needs an instance of Vault [36] to be available to the CI pipeline for a secure handling of application passwords and GitLab tokens.

The pipeline has different stages:

- *Build:* this stage builds the Docker image used to run the GWCelery workers and pushes it to the Gitlab container registry of the integration project. The *Dockerfile* [37] allows for a GWCelery installation either from a specific *tag* or *branch* of the code hosted in GitLab, or from a local directory. The former option is used by the CI pipeline. The Docker image can be based on a configurable Python version.
- *Secrets:* at this stage, the CI JSON Web Token (JWT) [38] is saved on Vault for later retrieval by the jobs running on the Kubernetes cluster. The JWT is used in the *Deploy* stage to retrieve the Helm charts from the GitLab repository. Moreover, the GraceDB authentication credentials are retrieved from Vault and used to generate a proxy client certificate for GWCelery programmatic authentication to GraceDB. The Kubernetes *Secret* [20] containing the proxy certificate is also installed on the cluster.
- *Deploy:* this stage installs the Hopskotch, GraceDB and GWCelery Helm charts on the Kubernetes cluster.
- *Configuration:* this stage creates the GraceDB user account to be used by GWCelery.
- *Test:* at this stage, MEG is used to upload to the newly created instance of GraceDB all the events belonging to a past real *superevent*. Then, a test script performs the acceptance tests on the GWCelery application.
- U*ndeploy:* this stage uninstalls the Hopskotch, GraceDB and GWCelery Helm charts on the Kubernetes cluster.

The test script outputs a JSON file containing the timestamps at which the different labels are applied or files are uploaded. The same information is also provided as a plot, as in the example provided in Figure 2.

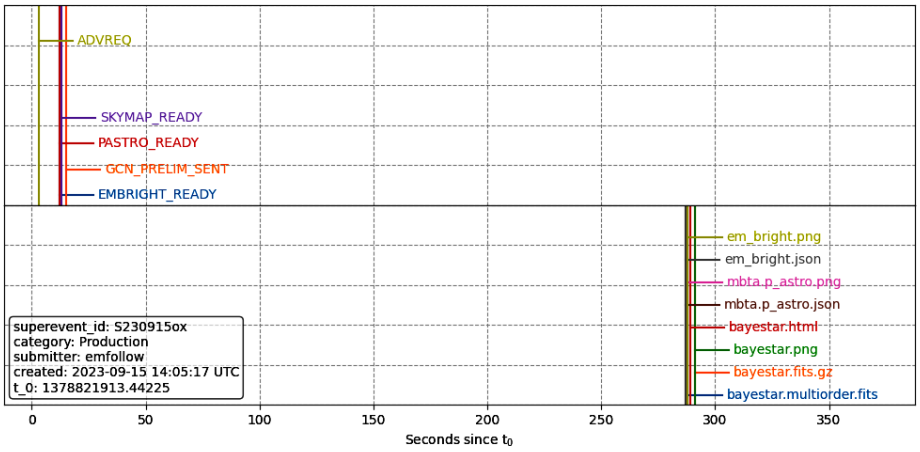

Fig. 2. Output of the automated GWCelery acceptance tests on a sandboxed LLAI installation on Kubernetes. The top panel shows the times at which labels are applied by GWCelery to a given *superevent* in GraceDB. The bottom plot shows the times at which files associated to the *superevent* are uploaded to GraceDB. t_0 is the time of the astrophysical event.

5 Conclusions

We have prepared a set of tools (in the form of Helm charts, scripts and packages) to deploy the LVK Low Latency Alert Generation Infrastructure (LLAI) on a Kubernetes cluster. This opens the possibility of deploying the LLAI on a broad range of cloud infrastructures, including private ones dedicated to scientific computing, with the only requirement of providing a Kubernetes cluster (which nowadays is a very loose constraint).

Moreover, our strategy includes options for a sandboxed deployment that can be used for the development and testing of single LLAI components in an isolated local environment, as well as to test the integration of other components.

References

1. M. Branchesi, *Multi-messenger astronomy: gravitational waves, neutrinos, photons, and cosmic rays*, J. Phys.: Conf. Ser. **718** 022004 (2016)
2. J. McIver, D. H. Shoemaker, *Discovering gravitational waves with Advanced LIGO*, Contemp. Phys. **61:4** 229-255 (2020)
3. D. Bersanetti, B. Patricelli, O.J. Piccinni, F. Piergiovanni, F. Salemi, V. Sequino, *Advanced Virgo: Status of the Detector, Latest Results and Future Prospects,* Universe **7** 322 (2021)
4. T. Akutsu et al., *Overview of KAGRA: Detector design and construction history*, Progr. Theo. Exp. Phys. **5** 05A101 (2021)
5. LVK Collaboration, *Mock Event Generator*, Available at: https://emfollow.docs.ligo.org/mock-event-generator/, https://pypi.org/project/mock-event-generator/
6. M. Maggiore, *Gravitational Waves Volume 1: theory and experiments*, Oxford University Press (2008)
7. B.P. Abbott et al., *Low-latency Gravitational-wave Alerts for Multimessenger Astronomy during the Second Advanced LIGO and Virgo Observing Run,* Astrophys. J. **875**/2 161 (2019)

8. Django Software Foundation, *Django*, Available at: https://djangoproject.com
9. M. Needleman, *The Shibboleth Authentication/Authorization System*, Ser. Rev. **30**/3 (2004)
10. B. Chesneau, *Gunicorn*, Available at: https://gunicorn.org/
11. Apache Software Foundation, *Apache HTTP server*, Available at: https://httpd.apache.org/
12. The PostgreSQL Global Development Group, *PostgreSQL*, Available at: https://www.postgresql.org/
13. LVK Collaboration, *GWCelery*, Available at: https://rtd.igwn.org/projects/gwcelery/en/latest/
14. A. Solem and contributors, *Celery*, Available at: https://docs.celeryq.dev/en/stable/#
15. Redis Ltd., *Redis*, Available at: https://redis.io/
16. SCiMMA Collaboration, *SCiMMA Hopskotch server*, Available at: https://scimma.org/hopskotch
17. Apache Software Foundation, *Apache Kafka*, Available at: https://kafka.apache.org/
18. NASA, *General Coordinates Network*, Available at: https://gcn.nasa.gov/
19. Cloud Native Computing Foundation, *Kubernetes*, Available at: https://kubernetes.io/
20. Cloud Native Computing Foundation, *Helm*, Available at: https://helm.sh/
21. *LLAI deployment on Kubernetes*, Available at: https://virgo.docs.ligo.org/computing/wp6-multimessengerastronomy-mma/low-latency-test-facility-at-cnaf/igwn-kube-gracedb/index.html
22. Traefik Labs, *Traefik*, Available at: https://traefik.io/traefik/
23. InCommon, *Trusted Access for Education and Research*, Available at: https://incommon.org/
24. Cloud Native Computing Foundation, *CertManager*, Available at: https://cert-manager.io/
25. *Lightweight Directory Access Protocol*, Available at: https://ldap.com/
26. Bitnami by VMware, *Redis Helm Chart*, Available at: https://github.com/bitnami/charts/tree/main/bitnami/redis
27. M. Movsisyan, *Flower*, Available at: https://flower.readthedocs.io/en/latest/
28. Pallets Projects, *Flask*, Available at: https://palletsprojects.com/p/flask/
29. SCiMMA Collaboration, *SCiMMA Hopskotch server container,* Available at: https://github.com/scimma/scimma-server-container
30. Apache Software Foundation, *Zookeeper*, Available at: https://zookeeper.apache.org/
31. GitLab, *Helm charts in the Package Registry:* https://docs.gitlab.com/ee/user/packages/helm_repository/
32. CNAF: https://www.cnaf.infn.it/en/institute/
33. Cloud Native Computing Foundation, *Prometheus*, Available at: https://prometheus.io/
34. Grafana Labs, *Grafana*, Available at: https://grafana.com/
35. The Kubernetes Authors, *Minikube*, Available at: https://minikube.sigs.k8s.io/docs/
36. HashiCorp, *Vault*, Available at: https://www.vaultproject.io/
37. Docker Inc., *The Dockerfile reference*, Available at: https://docs.docker.com/engine/reference/builder/
38. Okta Inc., *JSON Web Tokens*, Available at: https://jwt.io/