Chapter 5
Cloud Infrastructure of the European Language Grid

Florian Kintzel, Rémi Calizzano, and Georg Rehm

Abstract The European Language Grid (ELG) is a cloud-based platform, utilising a variety of software packages as well as infrastructure components and virtual hardware. The additional software components developed by the ELG project are usually provided as open source to facilitate re-use by third parties. This chapter provides an overview of the infrastructural setup used by the ELG cloud platform. The selected architecture also has implications for providers as well as users of the platform, e. g., in terms of the scaling behaviour of individual Language Technology (LT) services.

1 Introduction

One of the key technical goals of the ELG cloud platform is the ability to integrate functional Language Technology (LT) services from a variety of sources, i. e., to build a large platform and a corresponding community of providers and users of these services. The LT tools and services to be continuously integrated into the ELG platform are, thus, heterogeneous and vary in their technical setup, which is why a set of common approaches needs to be established to make the integration of the tools and services possible. One of the most basic joint technical approaches is the requirement for all functional services to be containerised so that they can run on the ELG cloud infrastructure. Providers can optionally benefit from utilising additional support functionality, e. g., source code repositories, container registries and deployment pipelines offered by the ELG platform.

Conceptually, the ELG platform consists of three layers, the user interface (UI) layer, the back end layer and the base infrastructure (see Figure 1). While the UI and back end are described in more detail in Chapters 2, 3 and 4, the present chapter focuses on the base infrastructure setup along with supporting functionality. Among others, this chapter is helpful for providers of functional LT tools and services or users interested in running parts of the ELG platform on their own hardware.

Florian Kintzel · Rémi Calizzano · Georg Rehm
Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Germany,
florian.kintzel@dfki.de, remi.calizzano@dfki.de, georg.rehm@dfki.de

© The Author(s) 2023

G. Rehm (ed.), *European Language Grid*, Cognitive Technologies,
https://doi.org/10.1007/978-3-031-17258-8_5

The rest of this chapter is structured as follows. First, Section 2 gives an overview of the building blocks of the ELG infrastructure. Section 3 provides information about the deployment side of the ELG platform, while Section 4 describes how the platform's scaling profile lends itself to usage in different real-world scenarios. Finally, Section 5 concludes the chapter with an overview of future work on the ELG platform infrastructure.

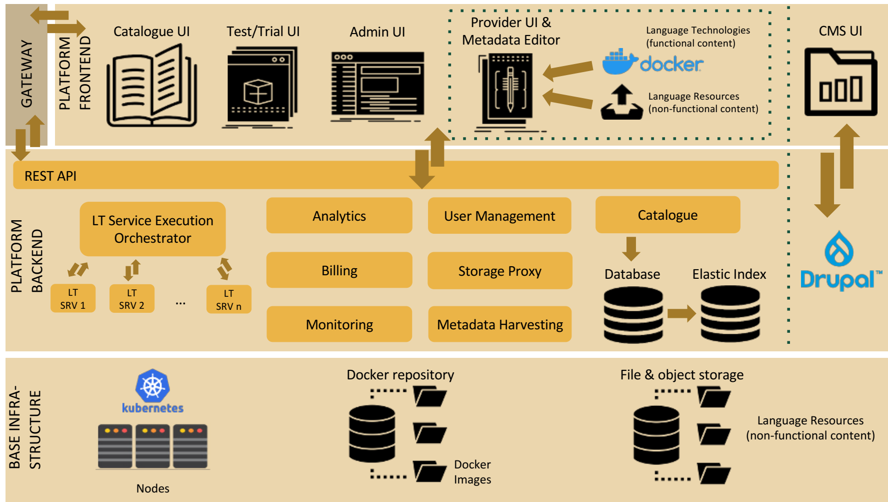

Fig. 1 ELG platform architecture

2 Cloud Infrastructure

The base infrastructure consists, first and foremost, of the compute nodes on which the European Language Grid runs, alongside their respective volume storage and networking facilities. On these, the Kubernetes[1] core components are installed (Section 2.1) including S3-compatible object storage (Section 2.2). We use a *managed* approach to Kubernetes, i. e., the installation, update and operation of the Kubernetes system itself is taken care of by a cloud provider. Together, this forms the hardware basis of the European Language Grid.

Conceptually, the base infrastructure also consists of a larger set of Git[2] repositories and container registries which are described in Sections 2.3 and 2.4.

[1] https://kubernetes.io

[2] https://git-scm.com

2.1 Kubernetes and Cloud Native

Kubernetes is an open source system for automating deployment, scaling, and management of containerised applications. It has seen widespread usage in recent years as *the* container orchestration tool of choice. Adoption of Kubernetes in a *managed* setup was still in a relatively early stage at the time the ELG project was exploring different cloud providers in early 2019. While various products by the typical hyperscalers already existed, European providers had only very recently started offering comparable solutions.

Our selection of Kubernetes as the framework of choice for ELG was primarily based on the following criteria:

- Kubernetes provides self-healing capabilities that can detect common failure situations and restart affected containers automatically.
- Through the use of a managed approach to Kubernetes, failures of the core Kubernetes system itself are the responsibility of the cloud provider.
  These first two criteria together allowed the ELG project to have a relatively small footprint in terms of operational complexity as failures are either self-healed or taken care of by the cloud provider, at least in theory. While exceptions *do* exist, this still has reduced the operational effort considerably.
- Kubernetes facilitates the usage of OCI-compatible containers.[3] As ELG aims to integrate different technologies used for the implementation of LT services and tools, OCI-compatible containers form a common approach for integration.
- Kubernetes provides off-the-shelf functionality for scaling up resources based on dynamic load. As ELG integrates hundreds of different LT tools and services, this functionality was deemed essential.
- Kubernetes namespaces[4] are useful to separate the different platform components from one another.
- Continuous adoption of Kubernetes within the industry assures continued support and development of this technology.

An ecosystem of compatible technologies has been established around Kubernetes with the Cloud Native Computing Foundation (CNCF).[5] CNCF promotes the use of a large set of base technologies for solving, e. g., authentication, monitoring, deployment and other common challenges. Most supporting technologies used in ELG (Section 3.2) are part of CNCF. Alongside this, a set of architecture patterns has emerged that aim to support properties such as Gannon et al. (2017):

- Cloud-native applications often operate at the global level.
- Cloud-native applications must scale well with thousands of concurrent users.
- Built on the assumption that infrastructure is fluid and failure is constant.
- Designed so that upgrade and test occur without disrupting production.
- Security must be part of the underlying application architecture.

[3] https://opencontainers.org – Open Container Initiative

[4] https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces

[5] https://www.cncf.io

2.2 Storage

The various components of the European Language Grid platform utilise persistent storage differently, as follows:

- Static Language Resources, i. e., corpora, models etc. available for direct download on the European Language Grid platform are persisted on S3-compatible object storage and can be fetched from there.
- The major infrastructural part of the ELG platform – the hundreds of LT tools services – do not utilise persistent storage at all, as they are designed stateless. All application code is shipped within an OCI-compatible container. This includes additional resources needed to run the service, e. g., language models and additional configuration files.
- The core ELG platform components (catalogue, authentication, CMS etc.) utilise network block storage attached to their running containers for persistence. This block storage is in turn backed up to the object storage on a regular basis.

Therefore, static resources *can* potentially be available for direct download and be included in the respective service container image as well. We decided for this approach to simplify deployment and management of images and resources, e. g., for a local installation of a set of LT services, it is only necessary to pull and run the respective images, i. e., no additional language resources need to be handled. Though this potentially results in duplication of resource files (within an image and as an additional separate file for download) it was deemed a necessary trade-off to keep the deployment model easier.

2.3 Software Repositories

ELG is comprised of various independent software packages for, e. g., platform components and individual LT services. The main ELG GitLab project repository[6] is set up as a GitLab group, consisting of various sub-groups and repositories. The different repositories in this group can be categorised as follows.

- The ELG Infrastructure Repository consists of a set of configuration files, mostly in the form of Helm[7] charts (see Section 3.1). These define which packages, i. e., containers, the ELG system consists of, as well as numerous additional configuration parameters such as the number of replicas and package-specific configurations. It can be used to set up multiple clusters. We maintain different branches within the repository, usually at least one for the development and one for the production cluster. The branches are not only used to distinguish between specific configurations for each cluster, but present different versions

[6] https://gitlab.com/european-language-grid

[7] https://helm.sh

of the ELG system as it matures during development. This is used to facilitate a staged roll-out to the production cluster. The actual source code for these components is not part of this repository. It only includes references to the container registries with the specific components. When installing the ELG cluster, these images are then downloaded ("pulled") from these registries.

- The ELG Cluster Admin repository holds cluster-specific configurations for each ELG instance that are applied separately from the settings of the ELG Infrastructure Repository. These mostly consist of the list of active administrative users for accessing the ELG infrastructure (those needing access to the infrastructure the ELG is running on, not users of the ELG platform), their roles and access rights as well as the configuration for build-bot, our continuous integration utility of choice. Included are also various utilities to manage the cluster. This repository is not needed for local deployment of the ELG, as such a deployment is usually only meant for a single user, typically a developer, and does not participate in continuous deployment.
- The main ELG GitLab platform project repository.[8] This repository hosts the individual components that make up the the core ELG platform and ELG website. These are mainly the platform (catalogue back end and front end components and the website content management system, along with a larger set of internal supporting and utility components.
- Individual sub-groups with repositories for individual LT services, grouped by provider. These consist solely of the LT services provided by members or associates of the ELG project consortium.

Implementation code for LT services not provided by ELG project consortium members is not usually held in the ELG GitLab group but rather managed via provider-specific repositories.

2.4 Container Registries

The images for instantiating containers in the ELG cluster are stored in various container registries. The Kubernetes installation powering ELG pulls the images from these registries on demand. These can be categorised as follows.

- The ELG GitLab project registry[9] is the registry that corresponds to the main ELG GitLab group, it hosts all images for all ELG core platform components (e. g., UI, back end, utilities) and for several ELG LT services developed by ELG project consortium partners. This registry allows public access to facilitate download and re-use of ELG components.
- Public registries for various externally implemented third-party components such as database system, identity and access management.

[8] https://gitlab.com/european-language-grid/platform

[9] registry.gitlab.com/european-language-grid

- Private registries of partners who do not publish their LT services under an open source license (proprietary LT services) or need to use their own registries for technical reasons.
- Various other public registries for open source LT services.
- The dedicated ELG registry.[10] As LT service images are partly pulled from registries external to the ELG project, this registry was set up to serve as a point to collect LT service images when they are ingested into ELG in order to perform versioning. Using this approach, ELG can ensure the availability of older versions of certain tools even if their original site is no longer serving them.

3 Installation

ELG utilises a GitOps approach (see, e. g., Beetz and Harrer 2021) to deployment, i. e., the configuration necessary to set up the compute cluster is managed by version control. The base artefact for deployment is the Helm chart.[11] Helm charts are used to manage the installation and update the ELG platform. Each chart bundles a set of components along with their configuration. All custom charts are defined in the ELG platform repository GitLab group (Section 3.1). Alongside the custom charts, a larger set of third-party charts is utilised to set up the respective components (Section 3.2).

We apply the charts to the cluster using a Continuous Integration (CI) approach, i. e., automatic deployment happens whenever changes to the configuration are detected by the CI (Figure 2).

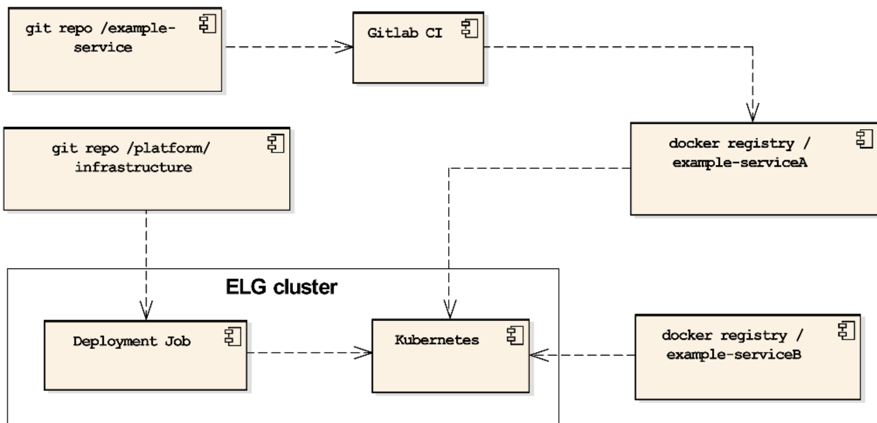

Fig. 2 ELG continuous integration

[10] registry.european-language-grid.eu

[11] https://helm.sh

If a new version of the infrastructure setup is detected, the CI checks out the respective changes and applies them to the cluster state. Any new container versions are then pulled from their distributed container registries. The Kubernetes cluster is updated with the latest configuration and takes care of gracefully shutting down and instantiating new containers.

Continuous integration regarding the ELG infrastructure only deals with updating the ELG cluster with the latest set of images (as specified by their version number) and configuration. It does not deal with building the respective images themselves.

3.1 ELG Charts

These charts were specifically developed for ELG and control its setup and installation. The packages are meant to be installed together, though it is possible to install only a subset for specific use cases (e. g., custom local installations). The architecture of the ELG is described in Chapter 2 as well as, e. g., Rehm et al. (2021), which is why we focus only on the software packages themselves.

- The ELG core package consists of definitions for various supporting functionalities of ELG. These are the Ingress[12] definitions for routing incoming traffic into the ELG cluster, the configuration for the rest server component as well as the configuration for the temporary storage component (used for large file operations). Various smaller configurations can also be found here, e. g., priority classes for pod scheduling, support for maintenance operations and others.
- The ELG back end chart consists of the definitions for the main back end components, the Django[13] and React[14] powered applications that form the ELG catalogue and the ELG back end and administrative applications. Included in this chart are also a set of utility functions that deal with housekeeping.
- The ELG LT services chart bundles the whole set of individual LT services installed in ELG. It is actually a collection of charts that follow a common structure, each sub-chart consisting of the definitions for the LT services of a specific LT services provider as well as a common chart for open source LT services by providers who only offer a small set of services. A definition for each individual LT service consists at the minimum of the reference to its image location, but can consist of numerous additional configurations, e. g., specific hardware requirements, helper images, parameters for scaling the service up and down and various other parameters.

[12] https://kubernetes.io/docs/concepts/services-networking/ingress

[13] https://www.djangoproject.com

[14] https://reactjs.org

3.2 Third-Party Charts

Apart from the core components, we use a set of third-party components, which provide their functionality to the ELG cluster. In the following, we briefly describe the main third-party components.

- Cert-manager[15] is a tool to manage issuing and updating of TLS certificates. It is used to install and refresh TLS certificates to allow for the encryption of all HTTPS traffic that reaches the cluster via one of the configured ingress-rules.
- The Horizontal Pod Autoscaler (HPA)[16] is a standard Kubernetes component used to scale pods based on their load and runtime behaviour. For scalability and load monitoring, Kubernetes collects certain metrics, e. g., CPU and memory load, from each pod. Therefore, it is necessary to have at least one instance of each type of pod to be up and running at all times. Otherwise, no metrics can be collected. This setup is useful to scale ELG core components, e. g., the portal website and back end. It cannot be utilised as is to scale the hundreds of LT services offered by the platform, as these need to be scaled down to zero replicas if they are not needed to not exceed the cluster capacity. Therefore, we introduced KNative (see below), which is feeding the standard autoscaler with a new metric "concurrency", based on the number of active requests to that LT service. Scaling those services still makes use of cluster-autoscaler functionality, but with the new metric also being available if no active replica of an LT service is instantiated.
- KNative[17] and Kourier[18] give ELG the possibility to scale down LT services based on the current number of parallel requests to them (concurrency). The concurrency metric is available even if there is no active replica of an LT service. KNative buffers HTTP requests to one of the ELG APIs until the specific LT service's container has started and keeps track of the concurrency metric to terminate the replica if it is no longer needed. We cannot overstate the importance of this functionality for ELG as the platform consists of hundreds of individual LT service components, not all of which need to run all the time, i. e., it would not be efficient to have all these services consume resources while in idle state. Starting up a container takes a certain amount of time though, while the service initialises. Using a service after it has not been used in a while therefore requires a certain spin-up time. KNative does not natively provide facilities to reduce the spin-up time further, but additional methods might be helpful in the ELG context, e. g., predictive auto scaling (Nanayakkara 2021). If frequent traffic is expected for a particular service, it can easily be configured to have one or more instances running at any given time, depending on hardware availability.

[15] https://cert-manager.io/docs

[16] https://kubernetes.io/de/docs/tasks/run-application/horizontal-pod-autoscale

[17] https://knative.dev/docs

[18] https://github.com/3scale-archive/kourier

- Ingress-Ningx[19] is installed to act as ingress-controller, i. e., handling HTTP traffic received and forwarding them to their respective endpoint within the cluster.
- Keycloak[20] is an open source solution for authentication and authorisation. It interfaces with front end, back end and LT services to provide single-sign on.
- Elasticsearch[21] is used to index the catalogue database for fast faceted search.
- Prometheus[22], Grafana, Loki and AlertManager form the ELG monitoring solution. They collect and analyse logs and metrics from all running components in the cluster (including the hardware) and provide visualisations in the form of dashboards and diagrams (Figure 3).

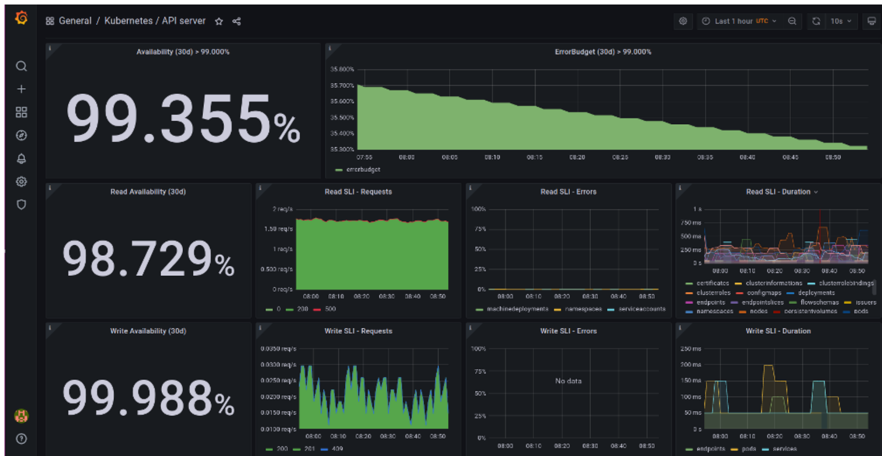

Fig. 3 Monitoring ELG using Prometheus and Grafana

- The ELG back end database uses PostgreSQL[23], a well-supported open source database engine. It holds all relevant data concerning the ELG catalogue, e. g., projects, organisations, LT resources, LT service as well as user information.
- MariaDB[24] is used for persistence of the Drupal CMS that powers the ELG portal. We plan to move this over to PostgreSQL for ease of maintenance.
- Not an off-the-shelf component, but rather specifically adapted for ELG, the s3proxy[25] facilitates the upload of LT resources (models, corpora, but also project and organisation logos etc.) to ELG. It acts as a proxy to the S3-compatible object storage that takes care of validating upload authorisation with the ELG back end and streams data to the object storage.

[19] https://nginx.org

[20] https://github.com/keycloak/keycloak

[21] https://github.com/elastic/elasticsearch

[22] https://prometheus.io

[23] https://www.postgresql.org

[24] https://mariadb.org

[25] https://gitlab.com/european-language-grid/platform/s3proxy

4 Scalability of LT Tools and Services

ELG is optimised for stateless LT tools and services. Its database systems are exclusively used by the platform back end for the metadata catalogue, user data etc. LT services do not have persistence enabled for them, with the exception of temporary files used for large file uploads. In the following, we describe our approach for scaling up individual LT services and describe its impact for service usability.

4.1 Implementation

With the goal of hosting thousands of individual LT tools and services with very different hardware needs, it is neither feasible nor practical to have all of them instantiated at the same time as this would require hundreds of Gigabytes of RAM even in idle mode, i. e., even if none of them are actually used. Therefore, ELG leverages the capabilities of KNative[26] which make is possible to automatically scale down services not currently in use to zero replicas. In this state, an LT service does not consume any hardware resources.

Scaling up an LT service happens automatically to an initial number of replicas once a request has been received for that individual service. Requests are buffered while new containers are starting up. This setup is especially suitable for services seeing little or irregular traffic. Further scale-up happens when a configurable threshold of concurrent requests for a given service is exceeded.

LT services deployed on ELG need to be aware that their life-cycle is exclusively controlled by Kubernetes and they need to expect to be started, stopped and horizontally scaled regularly, e. g., when the scheduler detects low resource situations on one of the nodes, if a container fails to respond, if high traffic is received to an LT service and other situations. LT services, therefore, highly benefit from quick start-up times and this is one of the reasons, why we opted for LT services to include necessary resources like models into their OCI images directly.

4.2 Use Cases

Given its scalability (Section 4.1), a number of use cases can be solved with ELG.

- Demonstration of service functionality: providers of LT tools and services can freely deploy their services to the platform and can expect to be discoverable via the platform's catalogue. For the try out functionality of services, a certain spin-up time from idle mode will not impact its usefulness. More performant installations of a given service could, e. g., be offered by the providers themselves.

[26] https://knative.dev/docs

- Batch processing of multiple documents: as the containers of an individual LT service will stay instantiated for some time after usage before scale-down happens, ELG is a good fit for batch processing as the initial scale-up time will not be a major contributing factor to processing time.
- For services intended to power applications where quick response times are required (e. g., mobile apps), however, the time it takes to spin up a container is likely too long (some seconds, depending on a service's implementation). This is why services on ELG can be configured to stay instantiated all the time and still benefit from dynamic scaling in high load situations. To be feasible, dedicated hardware is necessary, which service providers will be able to reserve on the ELG platform for a fee in the future so their services will show the responsiveness and performance they require.
- Remote processing is a second alternative for LT service providers who want to offer their services to the public. In this setup, the ELG platform uses a proxy to forward user requests to an external installation of a service, managed by the service providers themselves. This offers a flexible approach for providers to tune the hardware setup according to their own requirements.
- Management of non-functional LT resources, where only bandwidth limits scalability instead of compute capacity.

5 Conclusions

The ELG platform is growing continuously and the capacity, availability, operational readiness and tooling support of the base infrastructure need to evolve accordingly. We foresee a need to evolve in the following areas in particular.

- Hardware capacity and cost distribution: through the use of cloud technology, ELG has the technical capability to grow horizontally as required by the encountered load. In practice, though, the available hardware is restricted by budget considerations. Batches of utilised compute resources would need to be individually matched to the user requesting them or the provider offering them, to allow the ELG to calculate operational costs on a per request basis. With this and the emerging payment functionality, individual resource usage can be reimbursed.
- Hardware acceleration: ELG currently runs on CPUs exclusively. Already now, a larger number of LT services in ELG would benefit from GPU support. Apart from higher costs, GPU support will pose a number of technical challenges, among them a need to map LT services to specific compute nodes (with or without GPU support).
- Integration and deployment support: the initial integration of a functional LT service will need further automation and tooling support to be able to cope with increased demand and an increased number of running services.
- Workflow support: ELG would benefit from a possibility for easy workflow composition, spanning multiple LT services. Initial efforts have been started towards this goal (Moreno-Schneider et al. 2020).

- Gaia-X: in the Gaia-X[27] project OpenGPT-X[28] the ELG platform is currently being integrated into the wider Gaia-X ecosystem, i. e., ELG is further extended so that it complies to the technical Gaia-X specifications. This will enable all ELG LT services and resources to be discoverable and usable within Gaia-X.

This list only includes a selection of likely areas of improvement. Many additional use cases and requirements for ELG can be imagined – the platform infrastructure will need to grow and evolve as required.

References

Beetz, Florian and Simon Harrer (2021). “GitOps: The Evolution of DevOps?” In: *IEEE Software* 39.4, pp. 70–75. DOI: 10.1109/MS.2021.3119106.

Gannon, Dennis, Roger Barga, and Neel Sundaresan (2017). “Cloud-Native Applications”. In: *IEEE Cloud Computing* 4.5, pp. 16–21. DOI: 10.1109/MCC.2017.4250939.

Moreno-Schneider, Julián, Peter Bourgonje, Florian Kintzel, and Georg Rehm (2020). “A Workflow Manager for Complex NLP and Content Curation Pipelines”. In: *Proc. of the 1st Int. Workshop on Language Technology Platforms (IWLTP 2020, co-located with LREC 2020)*. Ed. by Georg Rehm, Kalina Bontcheva, Khalid Choukri, Jan Hajic, Stelios Piperidis, and Andrejs Vasiljevs. Marseille, France, pp. 73–80. URL: https://www.aclweb.org/anthology/2020.iwltp-1.12.pdf.

Nanayakkara, Pallage Kamindu (2021). “Serverless Performance Improvement for Knative using Predictive Auto Scaling”. PhD thesis. Sri Lanka: Informatics Institute of Technology. URL: http://dlib.iit.ac.lk/xmlui/handle/123456789/702.

Rehm, Georg, Stelios Piperidis, Kalina Bontcheva, Jan Hajic, Victoria Arranz, Andrejs Vasiļjevs, Gerhard Backfried, José Manuel Gómez Pérez, Ulrich Germann, Rémi Calizzano, Nils Feldhus, Stefanie Hegele, Florian Kintzel, Katrin Marheinecke, Julian Moreno-Schneider, Dimitris Galanis, Penny Labropoulou, Miltos Deligiannis, Katerina Gkirtzou, Athanasia Kolovou, Dimitris Gkoumas, Leon Voukoutis, Ian Roberts, Jana Hamrlová, Dusan Varis, Lukáš Kačena, Khalid Choukri, Valérie Mapelli, Mickaël Rigault, Jūlija Meļņika, Miro Janosik, Katja Prinz, Andres Garcia-Silva, Cristian Berrio, Ondrej Klejch, and Steve Renals (2021). “European Language Grid: A Joint Platform for the European Language Technology Community”. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations (EACL 2021)*. Kyiv, Ukraine: ACL, pp. 221–230. URL: https://www.aclweb.org/anthology/2021.eacl-demos.26.pdf.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

[27] https://www.gaia-x.eu, https://www.data-infrastructure.eu

[28] https://www.opengpt-x.de