EUROPEAN LANGUAGE GRID

D1.3 Base Infrastructure (Second Release)

Authors:	Florian Kintzel (DFKI), Georg Rehm (DFKI)
Dissemination Level:	Public
Date:	31-12-2019

About this document

Project	ELG – European Language Grid
Grant agreement no.	825627 – Horizon 2020, ICT 2018-2020 – Innovation Action
Coordinator	Dr. Georg Rehm (DFKI)
Start date, duration	01-01-2019, 36 months
Deliverable number	D1.3
Deliverable title	Base Infrastructure (Second Release)
Type	Report
Number of pages	12
Status and version	Final – Version 1.0
Dissemination level	Public
Date of delivery	Contractual: 31-12-2019 – Actual: 26-12-2019
WP number and title	WP1: Grid Platform – Base Infrastructure
Task number and title	Task 1.2: Installation, setup and adaptation of base infrastructure
Authors	Florian Kintzel (DFKI), Georg Rehm (DFKI)
Reviewers	Ulrich Germann (UEDIN), Julija Melnika (TILDE)
Consortium	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
	Institute for Language and Speech Processing (ILSP), Greece
	University of Sheffield (USFD), United Kingdom
	Charles University (CUNI), Czech Republic
	Evaluations and Language Resources Distribution Agency (ELDA), France
	Tilde SIA (TILDE), Latvia
	Sail Labs Technology GmbH (SAIL), Austria
	Expert System Iberia SL (EXPSYS), Spain
	University of Edinburgh (UEDIN), United Kingdom
EC project officers	Philippe Gelin, Alexandru Ceausu
For copies of reports and other ELG-related information, please contact:	DFKI GmbH European Language Grid (ELG) Alt-Moabit 91c D-10559 Berlin Germany Dr. Georg Rehm, DFKI GmbH georg.rehm@dfki.de Phone: +49 (0)30 23895-1833 Fax: +49 (0)30 23895-1810 http://european-language-grid.eu © 2019 ELG Consortium

Table of Contents

List of Figures 3
List of Tables 3
List Of Terms 4
Abstract 5
1 Introduction 5
2 External Cloud Service Provider 5
2.1 Overview 5
2.2 Operational report for Year 1 6
2.3 Hardware Budget Planning 6
3 Managing and Deploying the ELG Platform 7
3.1 ELG Hardware Cluster Configuration 8
3.2 ELG Package Repository 8
3.3 ELG Cluster Admin Repository 8
3.4 ELG External Configuration 9
3.5 Docker Registries 10
3.6 ELG Components 10
3.6.1 Build-Bot 10
3.6.2 Cert-Manager 10
3.6.3 Cluster-Autoscaler 10
3.6.4 Nginx-Ingress 11
3.6.5 Keycloak 11
4 Continuous Integration 11
5 Conclusion and Outlook 12

List of Figures

Figure 1: Grid Infrastructure Architecture 7
Figure 2: Continuous Integration 11

List of Tables

Table 3: Actual budget consumption for the first year (excl. December) 6

List Of Terms

Cluster (Kubernetes)	A set of machines (nodes), that run containerized applications managed by Kubernetes.
Container (Kubernetes)	A lightweight and portable executable image that contains software and all of its dependencies.
Controller (Kubernetes)	A loop that watches the shared state of a cluster through the API server and makes changes attempting to move the current state towards the desired state.
Deployment (Kubernetes)	A Kubernetes configuration that manages a replicated application.
Deployment	The action of making an installed and configured software application available to its intended internal or public users.
Docker	A software technology providing operating-system-level virtualization also known as containers.
Helm	Helm is a package manager for Kubernetes, i.e., a tool for the easy installation of pre-packaged components inside a Kubernetes cluster.
Helm chart	A configuration file for Helm listing all components of a package with their dependencies and additional configuration options.
Image	Stored instance of a container that holds a set of software; can be run as an application through software such as Docker.
Infrastructure as code	Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.
Instance (Cloud Provider)	A typical cost calculation unit as delivered by cloud IaaS providers. An instance can come in different sizes, e.g., small, medium, large with respect to cores and available RAM. The instance is represented by a virtual machine with the respective hardware resources.
Kubelet (Kubernetes)	An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
Namespace (Kubernetes)	An abstraction used by Kubernetes to support multiple virtual clusters on the same physical cluster.
Node (Kubernetes)	A worker machine in Kubernetes.
Object Storage	A storage architecture that manages data as objects with unique identifiers.
Pod (Kubernetes)	The smallest deployable units of computing that can be created and managed in Kubernetes.
S3	An object storage solution developed by Amazon. Today, often used to refer to solutions by other providers that have APIs compatible to the Amazon S3 API.
Service (Kubernetes)	An API object that describes how to access applications, such as a set of Pods, and can describe ports and load-balancers.
Volume (Kubernetes)	A directory containing data, accessible to the containers in a pod.

Abstract

This document is a follow-up to ELG Deliverable D1.2, *Base Infrastructure (First Release)*, and describes the current state of the ELG base infrastructure. It provides information about the current hardware setup, the base software and developer tooling, i.e., continuous integration. The architecture of the ELG platform proper is not part of this deliverable, but included in the upcoming platform deliverables (e.g., D2.4 in M16).

1 Introduction

The ELG development effort is structured along the respective architectural layers of the platform, which also correspond to the first three work packages, i.e., base infrastructure (WP1), platform (WP2), and front-end (WP3). While the platform and the front-end can be technically separated easily, it is a bit more difficult to specify exactly what constitutes the base infrastructure. The base infrastructure conceptually consists of:

- Provisioning the actual hardware the ELG runs on
- Providing development tooling, i.e., continuous integration, restricting user access to the cluster etc.
- Integrating off-the-shelf components by third-parties, if they are shipped and used as a single artefact, but not including libraries etc. used by the platform and front-end as part of their components
- Operation, deployment, scalability and monitoring of the ELG (as an outlook to later infrastructure releases, as the ELG is not yet publicly available)

Compared to the first release of this deliverable (ELG Deliverable D1.2), we applied some changes to the structure of this document, namely:

- The description of the APIs for the integration with LT services (ELG D1.2, Section 5) will be moved to the ELG platform deliverables (i.e., D2.4, D2.5, D2.6) and is no longer described here.
- The ELG “Infrastructure” Task Force has had various virtual meetings and two additional coding weeks (at the University of Sheffield and at R.C. Athena in Athens); these meetings and coding weeks are no longer described in detail in this document. Instead, we focus on a description of their results.

Topics covered in ELG Deliverables D1.1 or D1.2 are covered here only if there have been relevant changes.

2 External Cloud Service Provider

2.1 Overview

ELG contracted the cloud service provider SysEleven GmbH (Berlin).[1] At the end of the first year in the ELG project, the consortium has extended the contract to include a second hardware cluster, especially for demonstration purposes. The initial need for the second cluster was created by the demo foreseen to be given at META-FORUM 2019 on 8/9 October 2019; the new demo cluster will also be used for upcoming demonstrations.

[1] https://www.syseleven.de

As of November 2019, we, thus, use two clusters (“development” and “demo”). Each consists of four CPU / 16G RAM nodes along with 500G SSD network storage. This setup is sufficient for development purposes with a bit of headroom – the clusters are currently running with approx. 70% memory load in idle state.

2.2 Operational report for Year 1

In the first year of the project, no major issues have been encountered during the operation of the infrastructure cluster. As the ELG platform is not yet publicly available, the actual load of the system was limited. There has not been any concrete need yet for advanced reporting and monitoring tools. Nevertheless, existing notification and warning services offered by SysEleven GmbH have been set up, covering CVE[2] announcements, maintenance windows and incident notifications.

Hardware resources were, for the most part, sufficient for the current development effort, only during peaks at META-FORUM 2019 and with the running of a second cluster for demonstration purposes did we exceed the foreseen monthly budget (but not the average foreseen per month). Hardware utilization will linearly increase with the addition of new tools and LT services. In fact, it is currently impossible to run the full set of LT services already available in the platform (approx. 90 different LT services) concurrently on the current hardware setup. This is why we manually disabled those services not currently under development by utilizing the ELG Helm chart configuration. For live operations of the platform at a later stage, this manual configuration will be replaced by an automatic solution for detecting which services are currently needed and to enable and disable them automatically using the Cluster-Auto-Scaler.

2.3 Hardware Budget Planning

The budget distribution was originally planned as follows (see Deliverable D1.1)[3]:

- Year 1: 11,000€
- Year 2: 33,000€
- Year 3: 66,000€

Month	Cost	Comment
ELG Platform 11/2019	1,702.66€	Extended hardware for META-FORUM 2019
ELG Platform 10/2019	1,131.12€	Second cluster created (“demo”)
ELG Platform 09/2019	910.28€	–
ELG Platform 08/2019	910.28€	–
ELG Platform 07/2019	910.28€	–
ELG Platform 06/2019	910.28€	–
ELG Platform 05/2019	910.28€	First cluster (“dev”) created beginning of May 2019
Total	**7,385.18€**	

Table 1: Infrastructure budget consumption for the first year (excl. December)

Table 1 shows that the actual cost is well in range of the budget foreseen for the first year of the project.

[2] CVE: Common Vulnerabilities and Exposures: Information regarding software security concerns

[3] ELG Deliverable D1.1 (M4): *Requirements and Architectural Specification of the Base Infrastructure*

3 Managing and Deploying the ELG Platform

For managing and deploying the ELG platform (core components and LT services) the following packages and configurations are required.

- Hardware cluster configuration
- ELG package repository
- ELG cluster administrator repository
- ELG external configuration

They are described in their following Sections 3.1 to 3.4 while Sections 3.5 and 3.6 describe additional components needed to deploy a complete ELG platform instance. Together, these components form the basis of the infrastructure configuration. The components do not only form one cluster, but, rather, can be combined to create multiple instances of the ELG platform (Figure 1).

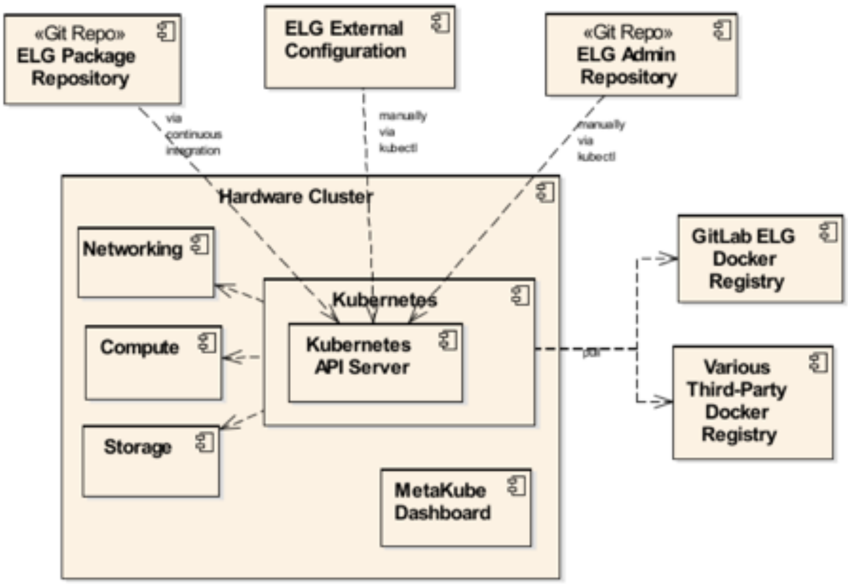

Figure 1: ELG Base Infrastructure Configuration and Deployment

The currently available instances are:

- ELG development cluster (“dev cluster”)
- ELG demonstration cluster (“demo cluster”)
- Various ELG instances on the local machines or clusters of the ELG development team

A third main instance, i.e., the ELG production cluster, will be created in 2020.

The rationale for the way we configured the platform is to have a core common part, that is identical in all instances (the ELG package repository) as well as specific parts (“configurations”), for each of the instances of the ELG (hardware cluster configuration, cluster administration repository, external configuration). The split of the latter (the specific instance configuration) into three parts is done for either technical reasons (the hardware cluster configuration) and for security reasons (administration repository and ELG external configuration).

When creating or updating an instance of the ELG, it is necessary to combine the core configuration with the respective specific configuration for the cluster in question.

3.1 ELG Hardware Cluster Configuration

This configuration manages the (virtual) hardware of each cluster. This configuration is maintained with the help of the tools provided by the respective cloud provider. It handles the provisioning of compute, storage, networking resources as well as the setup of the base Kubernetes system on top of this hardware.

The cloud-based ELG instances (in the SysEleven data centre) are managed and maintained via the SysEleven MetaKube Dashboard. The local installations (for development and testing) are managed via the MiniKube or kubeadm applications on individual workstations. The hardware setup thus configured defines the hardware limitations of the instance (i.e., CPU cores, RAM, storage, etc.), which is made known to Kubernetes to manage. Kubernetes manages the distribution of workloads (containers) to the available hardware. Kubernetes does *not* manage the actual hardware as configured by the cloud provider's toolbox. While the actual hardware configuration is specific to each cluster, the Kubernetes API offers a unified interface for workload management.

3.2 ELG Package Repository

The ELG package repository[4] consists of a set of configuration files, mostly in the form of Helm charts. Helm charts define which packages, i.e., containers, the ELG system consists of, as well as certain additional configuration parameters for these packages, e.g., the number of replicas and package-specific configuration.

The ELG package repository can be used to set up multiple clusters, as there is no cluster-specific configuration inside the repository. However, we maintain different branches within the repository to hold different versions of the ELG package repository, usually at least one for the development and one for the demo cluster. The branches are not used to distinguish between specific configurations for each of the clusters, but rather present different versions of the ELG system as it matures during development. This will later also be used to facilitate a staged rollout to the production cluster, i.e., first rolling out a new version on a test cluster and later, once successfully tested, to the production cluster.

This Git repository is currently marked as private, but might at a later time, be made public to facilitate third-parties to make use of the ELG base system, e.g., ICT-29b projects and others.

3.3 ELG Cluster Admin Repository

This private repository[5] holds cluster-specific configurations for each ELG instance:

- The list of active users for each instance
- Their roles and access rights within each instance
- The code and configuration for build-bot, the ELG continuous integration utility
- Utility scripts for the cluster administrators to, e.g., create new users for the cluster

The users mentioned above are users that need access to the actual Kubernetes API and resources, i.e., administration users, ELG developers for debugging, etc. They are different and managed separately from actual users of the ELG platform, i.e., those accessing certain sections of the ELG platform proper or calling LT services. The latter set of users is not in scope of the infrastructure configuration but of the platform itself.

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

[5] https://gitlab.com/european-language-grid/platform-admin/env-dev

This repository is not needed for local deployment of the ELG, as such a deployment is usually only meant for a single user and does not participate in continuous deployment.

The current set of defined user roles is as follows:

- elg-backend-dev-role: Role for ELG backend developers, giving access to the elg-backend-dev namespace
- elg-portal-dev-role: Role assigned to portal developers, giving access to the elg-portal-dev namespace
- expsys-role: Specific role for partner Expert System, giving access to their respective namespace
- saillabs-role: Specific role for partner Sail Labs,
- srv-provider-role: Standard role, assigned as default to all users, allows access to the namespaces elg-srv-dev namespace

We use Kubernetes namespaces to separate the different platform components from one another. A namespace in Kubernetes terminology is a set of resources exclusively grouped within it, i.e., a resource is only part of a single namespace. A resource in Kubernetes terms means a container, pod, ingress-role or any other Kubernetes object with the exception of cluster-wide resources, e.g., pods and persistent volumes.

Currently, we have defined the following namespaces:

- cert-manager-v0-9-1: Separate namespace for installation of the cert-manager package
- elg-admin: Admin-only namespace dealing mostly with the continuous integration
- elg-backend-dev: The ELG backend components (e.g., the catalogue)
- elg-core-dev: ELG core components (currently the nginx-ingres controller and RabbitMQ)
- elg-portal-dev: ELG portal components (i.e., the portal website)
- elg-srv-dev: All open source LT services components
- elg-srv-expsys: Closed source LT service components from partner ExpertSystems
- elg-srv-saillabs: Closed source LT service components from partner SailLabs
- users: Definitions for all users of the cluster (i.e., the content of the ELG cluster admin repository)
- syseleven-kubernetes-dashboard: Reserved by SysEleven GmbH
- velero: Reserved by SysEleven GmbH
- webterminal: Reserved by SysEleven GmbH
- a small set of kube-* namespaces, reserved for Kubernetes itself

3.4 ELG External Configuration

Certain parts of the cluster configuration that are not meant to be stored in any repository for security reasons. These mostly consists of:

- The user credentials, i.e., in the form of kubeconfig files. These are distributed to their specific user once and not stored anywhere within the system. They are specific to each cluster.
- Access credentials to access third-party closed source images that need to be installed inside the cluster. This currently consists mostly of the credentials to access the private docker registries of the partners ExpertSystems, SailLabs and Tilde.

This information is only stored inside each cluster manually (as Kubernetes secrets) and is accessible only to authorised users with the specific role to do so. The secrets are maintained by the respective partners.

3.5 Docker Registries

The images for instantiating containers inside the ELG cluster are stored in various Docker registries. There are multiple registries from which Kubernetes pulls the specified images, including:

- The ELG GitLab project registry that hosts the images for many ELG core platform components (e.g., UI, backend components) and for several ELG LT services.
- The public registries for third-party components like cert-manager, RabbitMQ etc (e.g., DockerHub).
- Private Docker registries from partners who do not publish their LT services under an open source license.

These registries are not part of the ELG infrastructure as such. The building of images for example is using different continuous integration solutions, e.g., GitLab CI for the ELG backend components and other in-house CI solutions for the components developed by ELG's commercial partners.

3.6 ELG Components

Apart from the custom core components, developed by the ELG project, that form the ELG platform and LT services, a set of third-party components is also installed as is, providing their functionality to the cluster.

3.6.1 Build-Bot

Build-bot is a small utility that is responsible for delivering the last part of the continuous integration chain, i.e., the update of the given cluster with the latest state of the respective branch of the ELG package repository. The continuous integration is described in more detail in the section about continuous integration.

3.6.2 Cert-Manager

Cert-manager[6] is a tool to manage issuing and updating of TLS certificates from Let's Encrypt[7]. It is used to install and refresh TLS certificates to allow for the encryption of all HTTP traffic reaching the cluster via one of the configured ingress-rules.

3.6.3 Cluster-Autoscaler

The Horizontal Pod Autoscaler is a standard Kubernetes component used to scale pods based on their current load and runtime behavior. It is not yet configured to allow for smooth scaling the ELG core components and LT services components. It is included here to explain the concept designed within the ELG project on how to deal with the fact that when handling thousands of different LT services, eventually the point will be reached when having them all instantiated at the same time is not feasible hardware-wise.

Unfortunately, Kubernetes is not currently able to natively scale down replicas to zero. 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 all the time. Else, no metrics can be collected. For the ELG, this restriction is problematic, due to the expected high number of LT services

[6] https://github.com/jetstack/cert-manager

[7] https://letsencrypt.org

that are planned to be deployed inside the grid. This is a rather unusual requirement, as most systems only consist of a limited and static set of components. To solve this, we plan to explore KNative[8], which among others, allows for downscaling to zero replicas.

3.6.4 Nginx-Ingress

Ingress-Ningx[9] is installed to act as ingress-controller, i.e., handling HTTP traffic received and forwarding them to their respective endpoint within the cluster.

3.6.5 Keycloak

Keycloak[10] is an open-source solution for authentication and authorization. It interfaces with front-end, back-end and LT services to provide a single-sign on experience.

4 Continuous Integration

Continuous Integration during development of the ELG is achieved using a combination of Git repositories, GitLab CI, webhooks and Docker registries.

Continuous integration regarding the grid infrastructure only deals with updating a given ELG cluster with the latest set of images (as specified by their version number) and configuration. It does *not* deal with the building of the respective images. The latter is out of scope for this document and depends on the CI workflows of the respectives service providers. Figure 2 gives an overview of the setup.

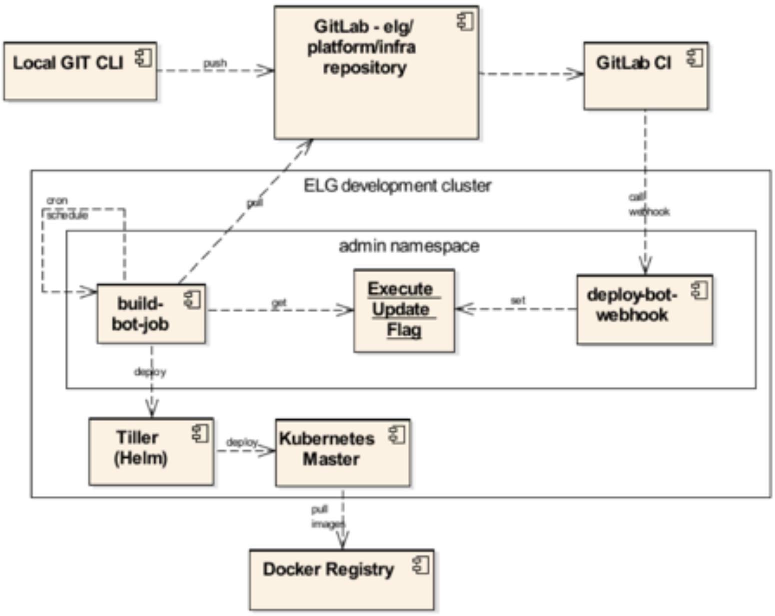

Figure 2: Continuous Integration Workflow

[8] https://knative.dev

[9] https://kubernetes.github.io/ingress-nginx/

[10] https://www.keycloak.org

Each push to the ELG infrastructure Git repository triggers a webhook on GitLab, which notifies the respective cluster as identified by the branch where the push is being made (development and demo currently). Inside the respective cluster, the notification webhook is received by deploy-bot-webhook components which are, in turn, queried by the build-bot-job which is triggered at regular intervals.

If a new version of the infrastructure setup is detected, the build-bot-job checks out the respective branch and executes the deployment scripts from within the cluster. As the configuration mostly consists of a set of container images and their versions, the respective images are then pulled from their distributed docker registries when needed. The Kubernetes cluster is updated with the latest configuration and takes care of gracefully shutting down and instantiating new pods whenever a configuration has changed.

5 Conclusion and Outlook

This document describes the current state of the ELG base infrastructure as of December 2019. In order to accommodate the European Language Grid in production, the following steps need to be taken:

- New hardware for the production cluster needs to be procured
- Operational procedures for the live cluster need to be specified, e.g., incident handling
- More components need to be installed, as required by the ongoing development effort by the front-end and back-end developments, e.g., permanent storage such as an S3 compatible object storage
- KNative or another solution needs to be introduced to allow for replicas to scale down to zero
- Backup- and recovery procedures need to be specified and integrated
- Monitoring tools for all parts of the grid need to be specified and integrated

These tasks will form the main focus of the next development cycles.