The CLARIN ERIC deployment infrastructure and its applicability to reproducible research

Willem Elbers
CLARIN ERIC
willem@clarin.eu

Egon W. Stemle
Eurac Research
egon.stemle@eurac.edu

André Moreira
CLARIN ERIC
andre@clarin.eu

Alexander König
Eurac Research
alexander.koenig@eurac.edu

Luca Cattani
Eurac Research
luca.cattani@eurac.edu

Martin Palma
Eurac Research
martin.palma@eurac.edu

Abstract

This paper is describing the needs and technological preconditions of the CLARIN ERIC infrastructure. It introduces how containerization using Docker can help to meet these requirements and fleshes out the build and deployment workflow that CLARIN ERIC is employing to ensure that all the goals of their infrastructure are met in an efficient and sustainable way. In a second step, it is also shown how these same workflows can help researchers, especially in the fields of computational and corpus linguistics, to provide for more easily reproducible research by creating a virtual environment that can provide specific versions of data, programs and algorithms used for certain research questions and make sure that the exact same versions can still be used at a later stage to reproduce the results.

1 Introduction

CLARIN ERIC is the central institution at the heart of the European CLARIN project(Krauwer and Hinrichs, 2014) as a whole and especially of its technical infrastructure. A lot of the central services like the Virtual Language Observatory (VLO), the CLARIN Identity Provider or the Component Registry are managed by CLARIN ERIC directly and there exists an extraordinary need for stability and reliability for these services. Because of this, the system management division of CLARIN ERIC has to take great care to design the infrastructure and the related build and deployment workflows very precisely to ensure both the expected reliability but also keep the workflows simple enough that new versions or security bugfixes can be deployed with relative ease and, if needed, quickly. The growing field of containerization of software looks like it could be the solution to fulfill all these needs and this article will explain the approach taken by CLARIN ERIC in detail. It will also showcase how this same approach could be used by researchers to ensure reproducibility of their research results by also employing containerization.

2 CLARIN Docker best practices

The CLARIN ERIC infrastructure is running on a diverse collection of hardware resources, spanning different data centres and hosting providers[1]. In order to easily move service deployments around, these deployments must be as loosely coupled to their hosts as possible. Furthermore, services within the CLARIN infrastructure are often developed at external centres and can be handed over to CLARIN ERIC to be run centrally if services mature and become relevant for the community as a whole. To ease this process a standardized package and deploy approach is necessary, including example setups for easy local evaluation. In addition, having an immutable infrastructure (Fowler, 2013) greatly improves the deployability of the individual services and builds confidence in the ability to frequently and quickly update and roll back services. This resulted in: (1) portability of the services, (2) deployability of the services, (3) evaluability of the services and (4) immutability of the infrastructure as high level goals for

[1] Currently 16 servers across 4 hosting providers

the CLARIN infrastructure. Containerization via Docker[2], used by CLARIN since early 2015 to package, deploy and manage the infrastructure, was the first step in this direction. Over time this evolved into a set of best practices and a build workflow for containerization based on Docker.

2.1 Organizing Docker images

A set of base images are provided for a number of common cases in the CLARIN infrastructure, all of which are available on the CLARIN GitLab group[3]. These images are all based on alpine linux[4], following the best practice set by the official Docker images. The main advantages of alpine linux are little pre-installed packages keeping the footprint low and increasing security by minimizing the attack surface. In order to be able to run periodic tasks, the base images include Crond as a cron daemon. We noticed when migrating existing application stacks to Docker images, that services often write logs into multiple files which is not handled very well by Docker containers. In order to overcome this issue, we have decided to include a Fluentd[5] daemon which is responsible for tagging and aggregating log data from multiple files into the container's stdout and stderr streams. The base images provide an out of the box setup for Supervisord[6], running in the foreground of the container managing the Fluentd, Crond and other processes. As you can see, we do not follow a *one process per container* mantra but are following and also recommending the more pragmatic approach of *each container should have only one concern*, where a single container is responsible for one service process[7]. But if there is a strong reason to add more processes this should be acceptable.

2.2 Building, testing and releasing Docker images

Within the CLARIN infrastructure many different environments, OSX, Windows, Linux and remote CI (continuous integration) pipelines are used. Unifying the build workflow across all environments is important as well as the ability to test Docker images before making them available in a public registry. Running a Docker container from an image requires some configuration. Since we want our services to be easy to evaluate, providing a standardized approach to run a container with sensible configuration presets is a must. Furthermore, we have experienced that the lack of a formal connection between the Dockerfile and the resulting Docker image – you are free to name and tag a Docker image as you like with each build command – can cause inconsistencies. Creating a tight coupling between a Dockerfile and the resulting Docker image is a strong requirement in an immutable infrastructure. Therefore goals set for our build workflow are as follows: (1) unify the build workflow across environments, (2) add support to test Docker images, (3) add support to easily run Docker images and (4) ensure a tight coupling of Dockerfiles to Docker images. With these goals in mind a project layout and build script have been created. The script provides an approach to build the Docker image inside Docker. The project layout requires the inclusion of a test and run configuration enabling easy testing and running of the Docker image. The project's directory name is used as the image name and the current git tag or revision as the image tag, resulting in a Docker image which is tightly coupled to the git project. CLARIN ERIC is using GitLab to host the Docker image git repositories because of the integrated CI environment together with a hosted Docker registry. This enables automated building, testing and publishing of Docker images.

2.3 Deploying

When deploying production applications based on Docker images, some extra aspects must be taken into consideration. Most applications will require a set of configuration parameters specific to a host or contain sensitive information which should be kept out of the Docker image. To accommodate these requirements, CLARIN ERIC has developed a deployment structure where secret files and parameters only exist on the host server. Because of the 'easy to evaluate' requirement, sample values are provided

[2] `https://docker.com`

[3] `https://gitlab.com/groups/CLARIN-ERIC`

[4] `https://alpinelinux.org/`

[5] `https://www.fluentd.org/`

[6] `http://supervisord.org/`

[7] See also the *Best practices for writing Dockerfiles* `https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#each-container-should-have-only-one-concern`

to allow the image to be run out-of-the-box, while at the same time an overriding mechanism is used to ensure that the real secrets available at the host server will replace the sample ones. In addiction to this, some applications are susceptible to require minor changes to their configuration rather often. If all configuration is supplied inside an image built as described in the previous points, one would always need a new build in order to make any changes to the configuration. This brings some welcome advantages such as immutability of the configuration. On the other hand, it can become cumbersome and slow, to change the configuration of a running production image. In order to tackle this problem, CLARIN ERIC has developed a strategy which allows rapid changes to be made to the configuration of the images, while still offering immutability.

This strategy is based on the following: (1) using docker-compose to group together the various images used for an application, (2) docker-compose is responsible for setting the default secret parameters, (3) docker-compose is responsible for overriding the default secret parameters with the real ones by specifying them in a *.env* file, (4) an *.env* symlink is created to bring in secrets from a location outside the docker-compose project if needed, (5) secret files, such as certificates and keys, are supplied to the containers using Docker host mounts. Otherwise the containers will generate them upon initialization, (6) non-secret configuration files are kept in the docker-compose project, stored on Git, allowing for quick immutable configuration changes without the need to rebuild the images, (7) a special user is available to deploy new releases, (8) a second special user is available with more permissions for testing and debugging purposes.

This setup also offers the ability to run automated end-to-end tests, which is a critical feature in order to comprehensively test applications composed of more than one Docker image.

3 Containers for reproducible research

Compiling language data for research is an intricate task. On the technical side—this is compared to for example the legal side—this may start as basic as the need to digitise data that had not already been born-digital. This is usually followed by both automatic and manual processing to some extent. Automatic processing stages towards linguistically enriched raw texts usually include tokenisation, lemmatisation and part-of-speech tagging, and sometimes it can also include named entity recognition and syntactic parsing.

The unification of all the necessary NLP tools into a unified processing framework is more the exception than the norm. This inevitably leads to a rich bouquet of individual solutions with individual installation procedures, development life cycles with maintenance and update schedules, etc. Furthermore, linguistic models that are often at the heart of NLP tools are also subject to change, and this change need not necessarily be synchronised with the tool itself. Finally, once the linguistic data is ready with its metadata and all annotations, analyses can be computed to answer research questions, or the data can be explored with query interfaces.

Now, in recent years and decades, the empirically-based method has become established in linguistics, and work with corpora in particular has been widely adopted. The underlying corpora are becoming more numerous and extensive These corpora and tools are of great value – not only to the scientists who collected and created them, but also to the research community as a whole: objective, data-oriented reproducibility is required by the scientific method. There may also exist a different, more practical need for access to NLP tools in their prior working state: for long-lasting project (e.g. longitudinal studies) with a need for reprocessing data it makes sense to process all data in the same manner – or at least be able to reconstruct the earlier processing pipelines. Therefore, the careful preparation and easy retrieval, and reusability of corpora and tools is of paramount importance.

As a case in point we consider the Korpus Südtirol (Abel and Anstein, 2011), an ongoing corpus linguistic initiative since 2005. Its goal is to collect, file and process texts in order to make them available to the public and document the use of written German in South Tyrol. One prominent subcorpus that has been used for several studies is the KoKo corpus, which was created with the aim to investigate and describe the writing skills of German-speaking secondary-school pupils at the end of their school career

by analysing authentic texts produced in classrooms, and to take external socio-linguistic factors (e.g. gender, socioeconomic background etc.) into consideration.

The ongoing nature of the project brings about the usual updates to the processing tools and interfaces, e.g. to the lemmatiser and part-of-speech tagger, the IMS TreeTagger[8]; to the corpus managing and querying tools, the IMS Open Corpus Workbench (CWB)[9]. But also brings about changes to interfaces or additions of tools, e.g. we added the NoSketch Engine[10], a limited version of the software empowering the Sketch Engine service[11]; and added ANNIS[12], a web browser-based search and visualization architecture for complex multilayer linguistic corpora with diverse types of annotation.

But any of these update might bring about changes to implicit default settings or to explicitly documented behaviour, which, in turn, may cause changes in the base data used for later analyses. To mediate any effects it is necessary to track changes to all of the processing tools and interfaces involved. Thinking about all this from the aspect of comparability, it can even become difficult to compare results with results from the same corpus at a different time. But research should be objective, repeatable and reproducible. Traditionally, this could be achieved by keeping different functional versions of the individual tools on hand. But early on, and at last as soon as there exist numerous dependencies, this task becomes intractable.

Pipelines built on Docker images are a cure in this situation because the allow for encapsulated, fully functional units that can be kept for later re-use and ensure identical behaviour.

4 Future work

The advantages brought by containerizing the central CLARIN infrastructure and the advantages containers can bring to create reproducible research pipelines, as discussed in this paper, set a clear path into the (near) future to strengthen our adoption of containerization technologies. While the Docker best practices feel stable for now there are still improvements we can make in the build and deploy workflow. The build workflow would benefit from a static vulnerability scanning (Evans and Larochelle, 2002) stage, based on clair[13] by CoreOS, integrated into the CI pipeline. The deploy workflow is still a work in progress so improvements are expected. Orchestration solutions, such as kubernetes (Burns et al., 2016), look very promising to even further enhance the deploy workflow especially with an integration into GitLab where automatic rolling deployments could be triggered after building and releasing a new version of a container. Within the CLARIN infrastructure some centres, such as Eurac Research, are already working with advanced orchestration solutions such as kubernetes.

References

Andrea Abel and Stefanie Anstein. 2011. Korpus Südtirol - Varietätenlinguistische Untersuchungen. In Andrea Abel and Renate Zanin, editors, *Korpora in Lehre und Forschung*, pages 29–54. bu,press - Bozen-Bolzano University Press.

Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, omega, and kubernetes. *Queue*, 14(1):10:70–10:93, January.

D. Evans and D. Larochelle. 2002. Improving security using extensible lightweight static analysis. *IEEE Software*, 19(1):42–51, Jan.

Chad Fowler. 2013. Trash your servers and burn your code: Immutable infrastructure and disposable components.

Steven Krauwer and Erhard Hinrichs. 2014. The clarin research infrastructure: resources and tools for e-humanities scholars. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 1525–1531. European Language Resources Association (ELRA).

[8] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/
[9] http://cwb.sourceforge.net/
[10] https://nlp.fi.muni.cz/trac/noske
[11] https://www.sketchengine.co.uk/
[12] http://corpus-tools.org/annis/
[13] https://github.com/coreos/clair