Infrastructure Requirements

David de Boer Maarten van Gompel Jaap Blom Femmy Admiraal
Hennie Brugman Mario Mieldijk Enno Meijers Roeland Ordelman
Ronald Siebes Thomas Vermaut Menzo Windhouwer

version 1.0 (CONCEPT), June 2022

Developed in the CLARIAH-PLUS project, funded by the NWO under grant 184.034.023

Contents

Introduction 2

Principles 3
- SEP . . . 3
  - Example . . . 3
- AUTO . . . 3
  - Example . . . 3
- REL . . . 3
  - Example . . . 3

Definitions 4

Requirements 4
1. The infrastructure *MUST* run applications that are packaged as *OCI containers*. (SEP) . . . 4
2. The infrastructure *MUST* connect to a **container registry**. (SEP) . . . 4
3. The infrastructure *MUST* be able to run *stateful applications*. . . . 4
4. The infrastructure *MUST configure applications* through environment variables. (SEP) . . . 4
5. The infrastructure *MUST capture application logs* at stdout. (SEP) . . . 4
6. The infrastructure *MUST aggregate logs*. (REL) . . . 4
7. The infrastructure *MUST automatically build* the application. (AUTO) . . . 5
8. The infrastructure *MUST automatically run application tests* when commits are pushed to the application repository. (AUTO) . . . 5
9. The infrastructure *MUST automatically deliver* new application versions to an acceptance and/or production environment. (AUTO) . . . 5
10. The infrastructure *MUST* be *highly available* corresponding to the infrastructure supplier's SLA (REL) . . . 5
11. The infrastructure *MUST* expose applications at *public HTTPS web endpoints*. . . . 5
12. The infrastructure *MUST* provision and renew *TLS certificates* for the web endpoints. (REL) . . . 5
13. The infrastructure *MUST* back up all application data at an agreed upon interval and has working restore functionality. (REL) . . . 5
14. The infrastructure *MUST* be *GDPR-compliant*, for instance in the way it stores data. . . . 5
15. The infrastructure *MUST* be configured with *security* in mind. (REL) . . . 6
16. The infrastructure *SHOULD* support *zero-downtime deployments*. (REL) . . . 6
17. The infrastructure configuration *SHOULD* be *declared in code*. (AUTO) . . . 6
18. The infrastructure *SHOULD capture metrics*. (Should have, REL) . . . 6
19. The infrastructure *SHOULD send alerts*. (AUTO) . . . 6
20. The application repository *SHOULD* be *open source*. (REL) . . . 6
21. The infrastructure *SHOULD* automatically configure *DNS*. (AUTO) . . . 6
22. The infrastructure *SHOULD automatically scale* when usage changes. (AUTO) . . . 6
23. The infrastructure *MAY* have an optimized way of hosting *static files*. (REL) . . . 7

License 7

Introduction

This set of requirements is largely derived from earlier work done in the scope of the Netwerk Digitaal Erfgoed (NDE).

CLARIAH aims to develop a service-oriented infrastructure consisting of a wide variety of different applications, this is often denoted as CLARIAH-as-a-Service (CLaaS) for short.

The user adoption of applications depends not only on their usefulness and quality, but also on their reliability and performance. When applications are slow or buggy, users will drop out. For foundational services such as those delivered by CLARIAH, on which large groups of users depend, this problem is even more prominent.

Infrastructure plays an important role in delivering reliable software. This document describes requirements for a fault tolerant, performant and reliable infrastructure. The requirements are based on modern DevOps and other best practices, including Twelve-Factor and FAIR4RS. The current NDE infrastructure is based on these requirements and CLARIAH would do good to follow suit here.

This document is aimed at organisations and their IT operators that would like to take ownership of CLARIAH applications. By joining us in following these practices, organisations will contribute to the longevity and adoption of the applications.

For technical requirements from the perspective of the software developer/technology provider, we refer to the CLARIAH Technical Requirements for Software and Services.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Principles

Guiding principles of these requirements are:

SEP

A clear separation between the application and the infrastructure it runs on. The application code should not contain infrastructure specifics and be portable so application instances can be hosted by multiple parties, including developers that want to run or debug the application locally. The infrastructure, in turn, should not have to change to accommodate the application.

Example

An infrastructure provider runs the Red Hat Linux distribution and require developers to package their applications as RPMs. This violates the SEP principle because the artifact that is deployed is no longer portable to other, non-Red Hat, environments.

AUTO

Processes are automated, which strengthens transparency, prevents human error and reduces the lead time between a feature request, code changes to build that feature, and the availability of the feature to users.

Example

When a new application needs to be deployed, system administrators manually configure a server and copy the code to it. When changes are made to the application code, developers have to ask administrators to deploy the latest code version. This slows down delivery times and therefore violates the AUTO principle. A better way would be to automatically build and deploy the application when commits are pushed to the application repository.

REL

The infrastructure is reliable, which helps adoption of the applications. Both software developers and end-users need reliable and performant services to get their work done.

Example

An infrastructure provider hosts many applications on a single virtual machine (VM). Application usage increases, and so does server load. To increase the VM's resources, the system administrator takes down the VM and deploys a larger instance. During that time the applications are unavailable to users. This violates the REL principle. Horizontal scaling, by running multiple application containers in parallel, would solve this issue.

Definitions

Requirements

1. The infrastructure *MUST* run applications that are packaged as *OCI containers*. (SEP)

Containers are a form of lightweight virtualization, striking a good balance between isolation and performance. Containers are self-sufficient (without external dependencies) and uniform (they look the same on the outside). This makes them decoupled from infrastructure specifics (such as the OS used by the infrastructure provider). The same container can be run on any Linux distribution, cloud provider (including AWS, Azure, Google and DigitalOcean) as well as on a developer's local Mac or Windows machine.

(This corresponds to point 14 of the Software/Service Requirements (SR))

2. The infrastructure *MUST* connect to a container registry. (SEP)

Built containers (see requirement 7) are pushed to a container registry. This can be an external registry, such as GHCR or one provided by the infrastructure itself.

The built artifacts must be accessible not only to the infrastructure itself but also to outside collaborators, who can then access and reuse those same containers, for example to run them locally (see requirement 1).

(This corresponds to point 14.1 of the Software/Service Requirements (SR))

3. The infrastructure *MUST* be able to run *stateful applications*.

The infrastructure can run both stateless applications (that store no state) and stateful applications, such as triple stores. For the latter to work, the infrastructure must be able to persist data between application runs (for instance on data volumes).

(This corresponds to point 14.5 of the Software/Service Requirements (SR))

4. The infrastructure *MUST configure applications* through environment variables. (SEP)

Configuration values (such as database connection strings, API URLs or secrets) must be parameterized. The best way to do so is with environment variables, a language- and OS-agnostic standard to specify key/value pairs at a high level of granularity. The infrastructure must therefore support providing these environment variables to the application.

(This corresponds to point 15.3 of the Software/Service Requirements (SR))

5. The infrastructure *MUST capture application logs* at stdout. (SEP)

Traditionally, logs are written by applications to a log file or an API (such as Logstash) as dictated by the infrastructure. This couples the application to the infrastructure it runs on. To safeguard proper separation, the infrastructure must capture application logs at the standard output stream (`stdout`).

(This corresponds to point 15.4 of the Software/Service Requirements (SR))

6. The infrastructure *MUST aggregate logs*. (REL)

To see what is going on in the running application (observability), developers as well as operators must be able to search through logs efficiently. This is especially relevant when multiple container instances of the application are running in parallel. Logs can be made available through a command-line and/or a web interface.

7. The infrastructure *MUST automatically build* the application. (AUTO)

When software is released, through tagging in the Version Control System (VCS), the infrastructure must automatically (re)build the application, producing an OCI container build artefact. This facilitates *continuous delivery and deployment*.

8. The infrastructure *MUST automatically run application tests* when commits are pushed to the application repository. (AUTO)

When commits are pushed to the source code in a Version Control System (VCS), the infrastructure must automatically run the test suite associated with the software (*continuous integration*). Automated tests prevent regressions only when they are run automatically on a standardized environment.

(This corresponds to point 8 of the Software/Service Requirements (SR))

9. The infrastructure *MUST automatically deliver* new application versions to an acceptance and/or production environment. (AUTO)

Manual steps slow down delivery to users. Preferably, code changes that have been committed, tested (requirement 8) and approved are delivered continuously and automatically to users at the production environment (*continuous deployment*).

10. The infrastructure *MUST* be *highly available* corresponding to the infrastructure supplier's SLA (REL)

The infrastructure absorbs faults so local problems in infrastructure components do not lead to failures that users will notice. For instance, when a server crashes or becomes unreachable, the infrastructure automatically migrates the applications to another server. The application remains available with as little interruption as possible.

11. The infrastructure *MUST* expose applications at *public HTTPS web endpoints*.

CLARIAH services span a shared infrastucture over multiple partners, most services are directly intended for public consumption, therefore they must be publicly accessible over HTTPS.

(This relates to point 13.5 of the Software/Service Requirements (SR))

12. The infrastructure *MUST* provision and renew *TLS certificates* for the web endpoints. (REL)

Users must never get a 'certificate expired' error in their browser, so the infrastructure must check expirations and renew certificates automatically, for instance using Let's Encrypt.

13. The infrastructure *MUST* back up all application data at an agreed upon interval and has working restore functionality. (REL)

Data loss is unacceptable, especially when that data has been provided by users in stateful applications. Both backup and restore functionality must be tested.

14. The infrastructure *MUST* be *GDPR-compliant*, for instance in the way it stores data.

If the infrastructure stores personal data, for instance logs, this must be done in a GDPR-compliant manner.

(This relates to point 13.4 of the Software/Service Requirements (SR))

15. The infrastructure *MUST* be configured with *security* in mind. (REL)

Any administrative access to the infrastructure itself *MUST* be restricted to a minimum of authorized persons. Preferably, only automated processes have access to change the infrastructure (see requirement 17).

All infrastructure and application components (such as containers) *SHOULD* be automatically scanned for security vulnerabilities.

All software components, including the infrastructure's OS and other packages, *MUST* be continuously updated to incorporate security patches.

16. The infrastructure *SHOULD* support *zero-downtime deployments*. (REL)

When new versions of the applications are deployed (requirement 9), this should happen without service interruption. Solutions include Blue/Green or Rolling Updates.

17. The infrastructure configuration *SHOULD* be *declared in code*. (AUTO)

The infrastructure's state is not the result of manual interventions (for example, running one-off commands), but a reflection of the configuration as it is declared in code and stored in a version control system (see also Software Requirement 1). Changes are checked into version control and automatically rolled out (see also requirement 9, an approach known as Infrastructure as Code (IaC).

Having the configuration as code makes it transparent and declarative. Preferably, the code is opened up to developers, so they can view it and propose the changes to it that are needed to run their applications.

(This relates to points 1 and 17 of the Software/Service Requirements (SR))

18. The infrastructure *SHOULD capture metrics*. (Should have, REL)

Logs (requirement 5) are used to diagnose and fix problems that have already occurred. Metrics, on the other hand, help prevent failures. They are values that measure the infrastructure resources and applications. For example by measuring CPU usage, capacity can be increased in time so user service will not be interrupted. Like log output (requirement 6), the metrics must be made available for reporting and analysis, preferably in a web interface.

19. The infrastructure *SHOULD send alerts*. (AUTO)

Having a centralized place to view logs (requirement 6) is only part of a monitoring solution. Viewing logs is pull-based, so a push mechanism for software failure notifications, on channels such as e-mail or Slack, must be added. The infrastructure, therefore, must have a way to send out alerts based on both logs and metrics thresholds.

20. The application repository *SHOULD* be *open source*. (REL)

Collaboration between developers, both inside and outside the organisation, yields reliable software. The ability to quickly and transparently report and fix bugs engages software developers with the application. It is therefore best if anyone can submit issues and propose changes.

21. The infrastructure *SHOULD* automatically configure *DNS*. (AUTO)

Automating all steps for launching new applications (including registering new hostnames and configuring DNS records) makes that process more reliable.

22. The infrastructure *SHOULD automatically scale* when usage changes. (AUTO)

When usage of the application increases and decreases, the infrastructure automatically adjusts capacity within set limits.

In the case of horizontal scaling (scaling in and out) this requires load-balancing the requests to the instances.

23. The infrastructure *MAY* have an optimized way of hosting *static files*. (REL)

If the infrastructure is able to directly serve static files, without configuring a web server first, this speeds up delivery. This is useful mainly for static HTML/JavaScript applications.

License