Zurich University
of Applied Sciences

School of
Engineering

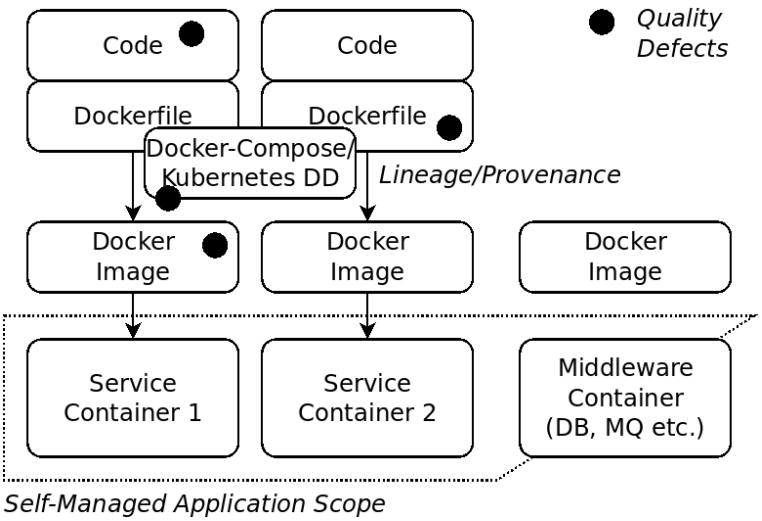
SCAD P07 – Container Image Deployment and Quality Checks

This is a two-weeks lab, i.e. SW 9, submission before SW 11 lecture.

In this lab, your task is to ensure the operation of a high-quality service in the cloud, where “quality” for now refers to only the static quality of all involved artefacts such as containers and orchestration files, as well as corresponding lineage artefacts.

1. Docker Images Preparation

If you have successfully built and hub-registered Docker images for long-running services in P05, use these ones along with a correponding Docker Compose file as starting point. If not, or out of curiosity, for instance if you want to use new microservice-oriented programming languages and frameworks, you are free to build your own additional images. Eventually you need a Docker Compose file referencing at least 2 long-running container images built by yourself; at least one of those should have an implementation in a language that allows interfacing with a self-hosted or managed stateful middleware service (e.g. database or message queue).



2. Kubernetes-Hosted Application

After finishing the preparation steps, the first task is then to turn the composition into something that can be deployed to Kubernetes and interacts with a stateful service according to Kubernetes scaling rules, resource limits and other characteristics.

First, use the `kompose` tool (from `https://kompose.io/`) to convert your Docker Compose file into a first Kubernetes manifest file. Check that the result is working by importing it into your Kubernetes or OpenShift account, e.g. on APPUiO (using `kubectl apply -f` or alternatively `oc create -f`) or in the ZHAW Cloud Lab (see `https://k8sbeta.init-lab.ch/login` and `https://info.cloudlab.zhaw.ch/pages/k8s.html`).

Export the application deployment descriptors (e.g. using `kubectl get -o yaml` or `oc export`) to get a second Kubernetes manifest file. Compare it with the Kompose-generated files. Are there any meaningful changes after the roundtripping, and what are the implications for application portability?

3. Quality Analysis

Inspect your source code files, Dockerfiles, Docker-Compose, OpenShift/Kubernetes files, and Docker container images. Investigate at least n (with n being equal to number of students in the team) potential weaknesses and improvements across these artefacts. The analysis should first be conducted with manual exploration based on best practices documents and experience-based peer review. For each weakness, describe or measure the original behaviour or sitation. Then, try at least one potential improvement, and again describe or measure the new behaviour or situation. Finally, perform a systematic analysis on at least one of these artefacts using systematic quality check tools and linters. Read the 2021 revision of the ‘CCEM 2020’ tutorial transcript (see Moodle) for inspiration on how to install and operate these tools; however you can also go scouting for similar tools. Extend your description with a paragraph on which tool was used and which results were obtained (and if the results encouraged you to improve your artefacts).

Generally, use the team composition to divide the work effectively. Not all weaknesses apply equally to all container images and related artefacts, therefore prioritise according to your understanding of how your containers could be improved.

The following pointers may guide your work specifically for container images:

- using different base images because the current one is too big on disk/slow to start
- combining layers or using multi-stage builds to cut down image size and/or startup time
- removal of unnecessary packages and files
- replacement of entrypoint command to achieve faster startup
- elimination of local write access within the running container; may be complemented with outsourcing critical writes to mounted volume
- reduction of privileged commands running as root user
- addition of a health check to simplify container management

- splitting the container into two, linked via a compose file
- adaptivity: in case a remote functionality is not available or not working (e.g. a cloud function), dynamically provide a local fallback, or vice-versa
- identification and labelling of hardware limitations beyond the main CPU architecture

Additional exploration and improvement possibilities can be discussed with the lab instructor. In any case, document the achieved improvements.