Code 004.453

CREATING A WEB SERVICE TO SERVE A LARGE NUMBER OF REQUESTS

Giga Kokaia

Sokhumi State University

Abstract

In web service systems today, there are often tasks which require high availability as well as fast processing of a large number of requests. To achieve this goal, an approach based on the use of Docker Swarm capabilities is proposed.

Аннотация

В современных сервисных системах мы часто сталкиваемся с задачами, в которых необходимо добиться высокой доступности сервиса, а также быстрой обработки большого количества запросов.
В данной публикации предлагается один из возможных путей достижения вышеупомянутой цели, основанный на использовании docker swarm.

Introduction

Docker is a set of platform as a service (PaaS) products that use an OS-level virtualization called “cgroups” to deliver software in packages called containers. The cgroup feature of the Linux kernel is used for isolation and limitation of resources for a group of processes. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and therefore use fewer resources than virtual machines. Containers are designed to run specific tasks and processes, not to host operating systems. The container created serves a single unit task. Once it has completed the given task, it stops. Therefore, the container life-cycle depends on the process ongoing within it. Once the process stops, the container stops as well. A “Dockerfile” defines this process. It is a script made up of instructions for how to build a Docker image. Commands such as FROM, RUN, COPY, CMD, etc. can be used within a Dockerfile. Dockerfile is a strict hierarchical system where it is necessary to describe one parent container with the FROM command; any container can be the parent of another container. If an empty container is necessary, a scratch image must be used.

This example shows an application which is written in Java and uses the Gradle build automation tool. For this it is necessary to prepare a Dockerfile with the following content.

```
FROM gradle:6.3.0-jdk11 as builder
WORKDIR /app
COPY . .
RUN /usr/bin/gradle bootWar
FROM openjdk:11-jre-slim
HEALTHCHECK --interval=10s --timeout=5s --start-period=10s --retries=5 CMD wget -q
http://127.0.0.1:8080/ping -O /dev/null || exit 1
COPY --from=builder /app/build/libs/app.war /app.war
CMD ["java”, “-jar”, “/app.war"]
```

These are multi-stage builds. Consider each stage:

FROM gradle:6.3.0-jdk11 as *builder*: The “FROM” command tells Docker to use the “gradle:6.3.0-jdk11” image, with Gradle version 6.3.0 and jdk 11. So “gradle:6.3.0-jdk11” is the image wherein Gradle and jdk 11 are already installed.

WORKDIR /app: This instruction sets the working directory to /app folder for any RUN, CMD, COPY instructions that follow it in the Dockerfile
COPY . . : Copies files and directories from the current directory and adds them to the filesystem of the container in the workdir.
RUN /usr/bin/gradle bootWar: This command tells Docker to run the Gradle program with the bootWar argument to build the program. The result is the “app.war” file, which will contain all the application files.
FROM openjdk:11-jre-slim: This tells Docker to use another image where only jre 11 is installed.
HEALTHCHECK --interval=10s --timeout=5s --start-period=10s --retries=5 CMD wget -q http://127.0.0.1:8080/ping -O /dev/null || exit 1: This HEALTHCHECK instruction tells Docker how to test a container to check that it is still working. If it gets a response, the service is considered to be healthy. This can detect cases such as a web server that is stuck in an infinite loop and unable to handle new connections, even though the server process is still running.
COPY --from=*builder* /app/build/libs/app.war /app.war: - Copies the file built in the previous stage to the current container stage.The final image will contain only files from the openjdk:11-jre-slim image ; the other files will be deleted.
CMD ["java”, “-jar”, “/app.war"]: - This command specifies which app should be started in order to run the container.

The advantage of using multi-stage builds is that the final version does not include the Gradle program and the libraries required for it, only the built file is necessary.
After building a Docker image, the image file must be put into the containers’ registry. To execute the “FROM gradle” command, Docker tries to find the image locally. If it does not exist, then it is downloaded from hub.docker.com. Images can be stored on hub.docker.com or any other existing containers’ registry, or one can be created for this purpose.

To build the Docker image, run the "**docker build -t app_image .** " command; it will build the image and store it locally. Afterwards, it can be uploaded to a registry with the "**docker push app_image**" command.

A Docker Swarm is a group of either physical or virtual machines that are running the Docker application and that have been configured to join together in a cluster. When a machine joins the cluster, it becomes a node in that swarm. Docker Swarm can be used as a load balancer, but other balancers such as HAProxy or Traefik can also be used.

The command "**docker service create -replicas 2 --name app_service app_image**" can be used to launch the app in the Docker Swarm. The argument “ **-replicas 2**” tells the swarm to run the app on two nodes.
After a new release, app_service can be updated with the command “**docker service update --force --update-order start-first --update-failure-action rollback --image app_image app_service**”.
With the argument "**--update-order start-first**", the old version of the service will not be turned off until the new Docker has checked the new service’s health. It stops the old version only after the newer version has been successfully released.

The argument "**--update-failure-action rollback**" says that if a new version of the service is not healthy, Docker will keep the older version.
When the app_service state is healthy, the Docker Swarm will redirect traffic to it, and will stop the old version. The switching process is almost instant, so requests will not fail during service updates.
In addition to Docker Swarm, there are other alternatives (OpenShift and Kubernetes). However, the basic principles of operation are very similar.

The approach discussed in this paper is a program written in the Java programming language using the Spring Framework, which is a service for Technical Inspection Centers. The aim of the service is to collect the results of vehicle technical inspections throughout the country. The program also generates visualizations of data in the form of charts.