What is a Docker Health Check?

Introduction

Docker health check is a feature provided as part of the Docker container platform. It lets you assess and report the health of a running container based on the application it hosts. You can use a health check to identify if the application running in the Docker container is functioning correctly or not.

A Docker health check operates by executing a command within the container at regular intervals. This command, defined by the user, could be anything that represents the health of the application within the container. If the command executes successfully, the container is considered healthy. If it fails, the container is flagged as unhealthy.

This feature is important because it helps you keep track of the operational status of containerized applications. This can improve availability and reliability and also aid in early problem detection, failover, scalability, and auditing.

Common Uses for Docker Health Checks

Application Crashes

One common way Docker containers can fail is due to application crashes. An application could crash due to various reasons, such as bugs in the code, unexpected input, or resource exhaustion.

Docker health checks can, for example, attempt to reach an application endpoint or check a process’s status. If the health check command returns a failure due to an application crash, the container is marked as unhealthy. You can then restart the container or stop it and shift traffic to other instances.

Dependency Failures

Another reason for Docker container failure is dependency failures. Applications often rely on external dependencies, such as databases, web services, or other components. If these dependencies fail or become unavailable, the application itself can fail, leading to a Docker container failure.

In cases of dependency failures, health checks can determine if the application in a container can still communicate with these external services. For example, a health check could attempt to establish a connection with a database or ping a dependent web service. Failure in these checks implies an issue with the dependencies. You can then restart the container or attempt to re-establish connections.

Resource Limitations

Resource limitations can also lead to Docker container failure. If a Docker container exhausts its allocated resources, such as CPU, memory, or storage, it can cause the application within the

container to fail. This is why it's vital to monitor resource usage and ensure that Docker containers are adequately provisioned.

Health checks can assist in monitoring the resource utilization of containers. By executing commands that inspect the usage of CPU, memory, or storage, the health check can identify when the resources are nearing exhaustion. On detecting such a scenario, you can reallocate resources or scale the application by adding more containers.

Misconfigurations

Lastly, misconfigurations can cause Docker containers to fail. This includes incorrect configuration of the Docker container itself, the application within the container, or the underlying environment. Misconfigurations can lead to a range of issues, from application errors to complete container failure.

Health checks play a vital role in identifying misconfigurations by attempting to execute operations that would fail if configurations are incorrect. For example, a health check might try to read a configuration file or make a network request using environment variables. Upon failure, you can investigate potential misconfigurations, revert to a previous known-good configuration, or take other remedial actions.

How to use HEALTHCHECK instruction in a Dockerfile

The HEALTHCHECK instruction lets you specify any valid command-line instruction, which will be executed within the container's context. The result of this command will determine whether the container is healthy or not.

To use the HEALTHCHECK instruction, you simply add it to your Dockerfile. The syntax is as follows:

```
HEALTHCHECK [OPTIONS] CMD <command>
```

If the command exits with a `0` status, the container is considered healthy. If it exits with a `1` (which usually means the application closed or crashed), the container is unhealthy. If the command exits with any other code, the test is considered inconclusive, and the health status is left unchanged.

For instance, if you're running a web server in your container, you could use a curl command to check if the server is still responding. Your HEALTHCHECK instruction could look something like this:

```
HEALTHCHECK CMD curl --fail http://localhost:8080 || exit 1
```

This command uses curl to send a request to the server. If the server responds, curl will exit with a 0 status, indicating that the container is healthy. If the server does not respond, curl will exit with a 1 status, indicating that the container is unhealthy.

HEALTHCHECK options

In addition to the CMD keyword, the HEALTHCHECK instruction also supports several options that allow you to configure the behavior of the health check:

- The `interval` option specifies the time between health checks. By default, Docker will perform a health check every 30 seconds. However, you can adjust this interval to suit your needs.
- The `timeout` option sets the maximum time Docker should wait for a health check to complete. If a health check takes longer than this time, Docker will consider the check to have failed. By default, the timeout is set to 30 seconds.
- The `start-period` option specifies the amount of time to wait before starting health checks. This can be useful if your container takes a while to start up. By default, Docker will start performing health checks immediately after a container is started (0 sec delay), but you can use the start-period option to delay this.
- The `retries` option controls the number of consecutive failures needed to mark a container as unhealthy. By default, Docker considers a container unhealthy after three (03) consecutive failures.

Example

```
FROM node:alpine
# Install curl for health checks
RUN apk add --no-cache curl
WORKDIR /app
COPY package*.json ./
RUN npm install
# Then copy the rest
COPY . .
EXPOSE 3000
# healthcheck settings
HEALTHCHECK --interval=30s --timeout=3s --retries=3 \
  CMD curl -f http://localhost:3000/health || exit 1
# start npm
CMD ["npm", "start"]
```

Let's break down what this health check does:

- `–interval=30s:` Checks every 30 seconds
- `–timeout=3s:` Waits up to 3 seconds for a response
- `–retries=3:` Tries 3 times before marking as unhealthy
- `curl -f http://localhost:3000/health:` The actual check - tries to access your app
- `|| exit 1:` If the curl fails, the health check fails

Custom Scripts for Complex Health Checks

For more complex health checks, you can write custom scripts that perform a series of tests on your

application. These scripts can be written in any scripting language that your container supports, such as bash or python.

To use a custom script, you need to include it in your Dockerfile and then use the HEALTHCHECK instruction to run it. Here's an example:

```
COPY healthcheck-script.sh /usr/local/bin/
HEALTHCHECK CMD /usr/local/bin/healthcheck-script.sh
```

In your script, you should aim to test the most important functions of your application. The specific tests will depend on your application, but they could include checking if a database is up and running, if a web server is responding, or if an API endpoint is available.

> Remember, if your script exits with a 0 status, Docker will consider the container healthy; if it exits with a 1, the container is considered unhealthy. Make sure your script is designed to return the correct exit status based on the results of your tests.

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m347/learningunits/lu03/lu03a/healthcheck

Last update: **2025/05/15 15:48**