In this scenario, you will learn how the new Healthcheck instruction provides visibility into the state of applications running inside containers.

Step 1 - Creating Service
The new Healthcheck functionality is created as an extension to the Dockerfile and defined when a Docker image is built.

Create HTTP Service with a Healthcheck
The Dockerfile below extends an existing HTTP service and adds a healthcheck.

The healthcheck will curl the HTTP server running every second to ensure it's up. If the server responds with a non-200 request, curl will fail and an exit code 1 will be returned. After three failures, Docker will mark the container as unhealthy. The format of the instruction is HEALTHCHECK [OPTIONS] CMD command.

DOCKERFILE :
```
FROM katacoda/docker-http-server:health
HEALTHCHECK --timeout=1s --interval=1s --retries=3 \
  CMD curl -s --fail http://localhost:80/ || exit 1
```

Currently, Healthcheck supports three different options:

interval=DURATION (default: 30s). This is the time interval between executing the healthcheck.

timeout=DURATION (default: 30s). If the check does not finish before the timeout, consider it failed.

retries=N (default: 3). How many times to recheck before marking a container as unhealthy.

The command executing must be installed as part of the container deployment. Under the covers, Docker will use docker exec to execute the command.

This Dockerfile directive adds a health check to a Docker image that is built from the katacoda/docker-http-server base image.

The HEALTHCHECK directive specifies that the Docker daemon should periodically check the health of the container, and take action if the container is determined to be unhealthy. In this case, the health check is configured to run every 1 second (--interval=1s) with a timeout of 1 second (--timeout=1s) and to retry the check up to 3 times (--retries=3) before considering the container unhealthy.

The CMD instruction specifies the command to run as part of the health check. In this case, the command uses curl to make an HTTP GET request to http://localhost:80/ and check if the request is successful (--fail). If the request is unsuccessful, the command exits with status code 1, indicating that the container is unhealthy.

Overall, this health check ensures that the Docker container running the HTTP server is responsive and functioning properly, and alerts the Docker daemon to take action if the container becomes unresponsive or stops functioning as expected.

Build and Run
Before continuing, build and run the HTTP service.
docker build -t http .

```
$ docker build -t http .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM katacoda/docker-http-server:health
health: Pulling from katacoda/docker-http-server
12b41071e6ce: Pull complete
fb1cef6edba2: Pull complete
1061ea2815dd: Pull complete
Digest: sha256:fee2132b14b4148ded82aacd8f06bdcb9efa535b4dfd2f1d88518996f4b2fb1d
Status: Downloaded newer image for katacoda/docker-http-server:health
 ---> 7f16ea0c8bd8
Step 2/2 : HEALTHCHECK --timeout=1s --interval=1s --retries=3   CMD curl -s --fail http://localhost:80/ || exit 1
 ---> Running in b6d11d457545
Removing intermediate container b6d11d457545
 ---> 00b6feb0b015
Successfully built 00b6feb0b015
Successfully tagged http:latest
$
```

By default, it will start in a healthy state.
docker run -d -p 80:80 --name srv http

```
Successfully tagged http:latest
$ docker run -d -p 80:80 --name srv http
6c745c404f65268580241a11abfacb048b47458e89d9bf78ab5e8b7ab1ab283c
$
```

In the next steps we will cause the HTTP Server to start throwing errors.

Step 2 - Crash Service
With the HTTP server running as a container, the Docker Daemon will automatically check the healthcheck based on the options. It will return the status when you list all the running containers, for example
docker ps.

```
6c745c404f65268580241a11abfacb048b47458e89d9bf78ab5e8b7ab1ab283c
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                            PORTS               NAMES
6c745c404f65        http                "/app"              About a minute ago  Up About a minute (healthy)       0.0.0.0:80->80/tcp  srv
$ curl http://docker/unhealthy
$
```

Set Unhealthy
The HTTP server has a special endpoint which will cause it to start reporting errors.
Make a http request to
curl http://docker/unhealthy
The service will now go into error mode. In the next step, we'll look at how Docker handles this.

Step 3 - Verify Status
As the HTTP server is in an error state, the healthcheck should fail. Docker will report this as part of the metadata.

Detecting Errors
Docker will report the health status in various different places. To get the raw text stream, useful during automation, use Docker Inspect to pull out the Health Status field.
docker inspect --format "{{json .State.Health.Status }}" srv

```
$ docker inspect --format "{{json .State.Health.Status }}" srv
"unhealthy"
$ docker inspect --format "{{ .State.Health.Status }}" srv
unhealthy
```

The Health state stores a log of all the failures and any output from the command. This is useful for debugging why a container is considered unhealthy.
docker inspect --format "{{json .State.Health }}" srv

```
$ docker inspect --format "{{json .State.Health }}" srv
{"Status":"unhealthy","FailingStreak":262,"Log":[{"Start":"2023-04-16T14:17:45.900947887Z","End":"2023-04-16T14:17:45.955606139Z","ExitCode":1,"Output":""},{"Start":"2023-
04-16T14:17:46.960785915Z","End":"2023-04-16T14:17:47.014698944Z","ExitCode":1,"Output":""},{"Start":"2023-04-16T14:17:48.020591125Z","End":"2023-04-16T14:17:48.073637012Z
","ExitCode":1,"Output":""},{"Start":"2023-04-16T14:17:49.078862382Z","End":"2023-04-16T14:17:49.132523895Z","ExitCode":1,"Output":""},{"Start":"2023-04-16T14:17:50.137714
243Z","End":"2023-04-16T14:17:50.191411578Z","ExitCode":1,"Output":""}]}
$
```

The status of all the containers can be viewed using
docker ps

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                          PORTS               NAMES
6c745c404f65        http                "/app"              6 minutes ago       Up 6 minutes (unhealthy)        0.0.0.0:80->80/tcp  srv
$
```

Step 4 - Fix Service
Use an extra HTTP endpoint to make the service healthy again. curl http://docker/healthy

View Healthy Status
Once the service is healthy again, Docker will update the status.
docker ps
docker inspect --format "{{json .State.Health.Status }}" srv

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                     PORTS                  NAMES
6c745c404f65        http                "/app"              6 minutes ago       Up 6 minutes (unhealthy)   0.0.0.0:80->80/tcp     srv
$ curl http://docker/healthy
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                   PORTS                  NAMES
6c745c404f65        http                "/app"              7 minutes ago       Up 7 minutes (healthy)   0.0.0.0:80->80/tcp     srv
$ docker inspect --format "{{json .State.Health.Status }}" srv
"healthy"
$
```

Step 5 - Healthchecks with Swarm
Docker Swarm can use these health checks to understand when services need to be restarted/recreated.

Initialise a Swarm cluster and deploy the newly created image as a service with two replicas.
docker rm -f $(docker ps -qa);
docker swarm init
docker service create --name http --replicas 2 -p 80:80 http

```
$ docker ps -qa
6c745c404f65
$ docker rm -f $(docker ps -qa);
6c745c404f65
$ docker swarm init
Swarm initialized: current node (nno0roo38au45vxsnae5i5a15) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-3ayvc7zhs0ij7q9icmfz9frk3e3cy6itnvkjtkgh2cjr76et5p-4ywrwm4ajn0blsh3jgs5dtsf9 10.0.0.11:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

$ docker service create --name http --replicas 2 -p 80:80 http
image http:latest could not be accessed on a registry to record
its digest. Each node will access http:latest independently,
possibly leading to different nodes running different
versions of the image.
```

You should see two containers responding
The docker service create command is used to create a new Docker service, which is a scalable group of containers that can be distributed across multiple Docker hosts. In this case, the command creates a new service named "http" with 2 replicas and maps port 80 on the Docker host to port 80 in the container.

Here's a breakdown of the command:
docker service create: Creates a new Docker service
--name http: Sets the name of the service to "http"
--replicas 2: Specifies that the service should have 2 replicas (i.e., 2 instances of the container)
-p 80:80: Maps port 80 on the Docker host to port 80 in the container
http: Specifies the image to use for the service, in this case "http"
Overall, this command creates a Docker service that runs two instances of a container based on the "http" image, each of which exposes port 80. This makes the service accessible through port 80 on the Docker host, which can be accessed by clients through a web browser or other HTTP client.

You should see two containers responding - curl host01
Randomly cause one of the nodes to be unhealthy with - curl host01/unhealthy
You should only see one node processing requests as Swarm has automatically removed it from the load balancer: - curl host01
Swarm will now restart the unhealthy service automatically. - docker ps
After Swarm has restarted the service you should see two nodes again: - curl host01

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                     PORTS               NAMES
03f1126dacd0        http:latest         "/app"              32 seconds ago      Up 28 seconds (healthy)    80/tcp              http.1.67ovq7awph2j6x8wavjbza6ht
c2a9741ad0d0        http:latest         "/app"              3 minutes ago       Up 3 minutes (healthy)     80/tcp              http.2.g5rs5arq0i0ygwlummjw6qpai
$ curl host01
<h1>A healthy request was processed by host: c2a9741ad0d0</h1>
$ curl host01/unhealthy
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                     PORTS               NAMES
03f1126dacd0        http:latest         "/app"              3 minutes ago       Up 3 minutes (unhealthy)   80/tcp              http.1.67ovq7awph2j6x8wavjbza6ht
c2a9741ad0d0        http:latest         "/app"              6 minutes ago       Up 6 minutes (healthy)     80/tcp              http.2.g5rs5arq0i0ygwlummjw6qpai
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                     PORTS               NAMES
c2a9741ad0d0        http:latest         "/app"              6 minutes ago       Up 6 minutes (healthy)     80/tcp              http.2.g5rs5arq0i0ygwlummjw6qpai
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                                  PORTS               NAMES
0816d1f67f9e        http:latest         "/app"              4 seconds ago       Up Less than a second (health: starting)   80/tcp              http.1.im1hfbu0ua1wemgl9cg7c
53hj
c2a9741ad0d0        http:latest         "/app"              6 minutes ago       Up 6 minutes (healthy)                  80/tcp              http.2.g5rs5arq0i0ygwlummjw6
qpai
$
```