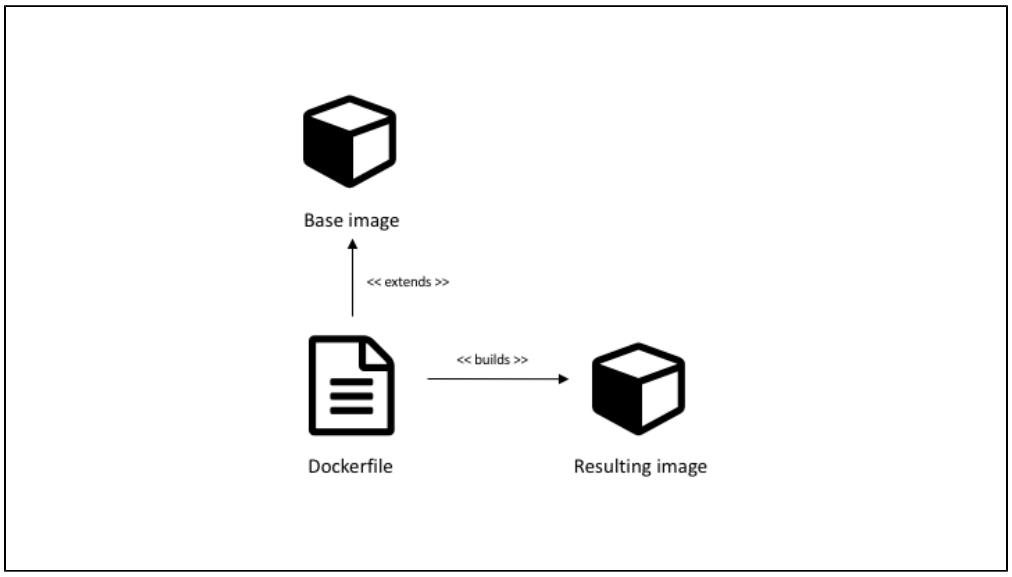
How to monitor Dockerized apps with Dynatrace AppMon

In this article, we'll propose two methods for integrating the Dynatrace Application Monitoring agent with a Dockerized application. These methods, referred to as the "inheritance-based" and "composition-based" approach in the following, have proven to be most effective by our customers. Depending on your particular situation, you may find the one or the other more suitable. To make your decision easier, we have listed known pros and cons.

Option A: Inheritance-based approach

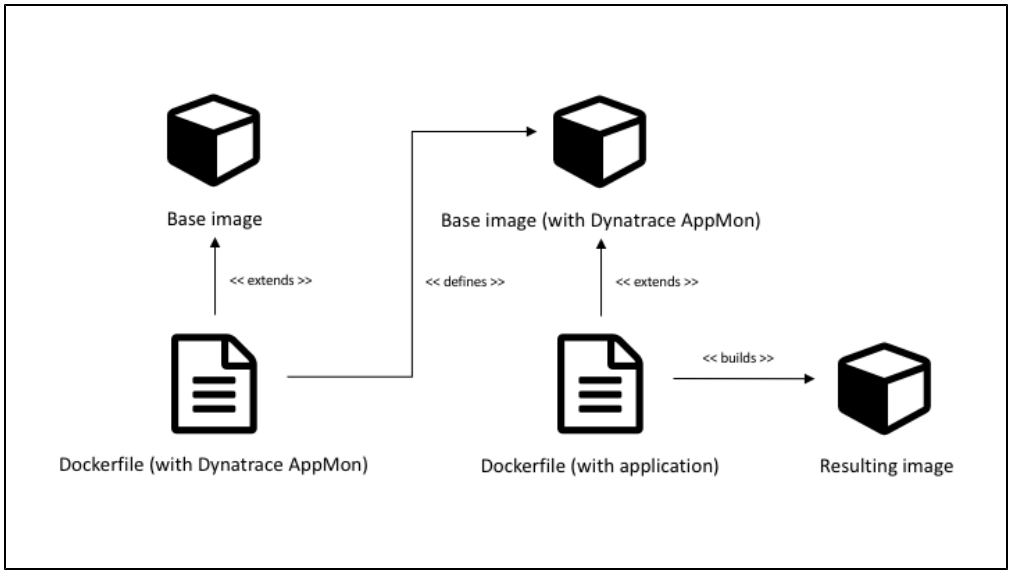
In technical terms, Dockerizing an application (process) typically involves two parts: a base image, such as `java:8` or `node:7`, which provides the basic execution environment, and a `Dockerfile` that augments selected base image with application-specific setup instructions. Finally, building your `Dockerfile` via the `docker build` command creates the desired Docker image.

A comprehensive set of base images is provided on the Docker Hub. I recommend reading Using Dockerfiles to Automate Building of Images and Best Practices for Writing Dockerfiles if you wish to dig deeper into this topic.



Examples

In the following, I will describe how you can create base images for your application, ready to be monitored by Dynatrace Application Montitoring. By weaving the agent into base images effectively auto-enables monitoring by making Dynatrace part of the "platform".



Example: Java

Here is a `Dockerfile` that extends the official openJDK Docker image base image and downloads and extracts the agent. We set a number of environment variables, mostly for convenience, such as `DT_AGENT_NAME` and `DT_AGENT_COLLECTOR`, which you'll have to populate with your own values. Additionally, we initialize `JAVA_OPTS` with a proper `-agentpath` that picks up these variables and points to the 64-bit agent in `DT_AGENT_LIB64`.

```
FROM openjdk:8

ENV DT_AGENT_INSTALLER_URL
"http://files.dynatrace.com/downloads/OnPrem/dynaTrace/6.5/6.5.0.1289/dynatrace-agent-
6.5.0.1289-unix.jar"

ENV DT                      "/dynatrace"
ENV DT_AGENT_LIB32          "${DT}/agent/lib/libdtagent.so"
ENV DT_AGENT_LIB64          "${DT}/agent/lib64/libdtagent.so"

ENV DT_AGENT_NAME           "java"
ENV DT_AGENT_COLLECTOR      "127.0.0.1:9998"

ENV JAVA_OPTS
"-agentpath:${DT_AGENT_LIB64}=name=${DT_AGENT_NAME},collector=${DT_AGENT_COLLECTOR}"

# Install the Dynatrace Application Monitoring Agent
RUN curl -L -o /tmp/`basename ${DT_AGENT_INSTALLER_URL}` ${DT_AGENT_INSTALLER_URL} &&
\
    java -jar /tmp/`basename ${DT_AGENT_INSTALLER_URL}` -t ${DT} && \
    rm -f /tmp/`basename ${DT_AGENT_INSTALLER_URL}`
```

Building this Dockerfile, e.g. via `docker build . -t openjdk:8-dtappmon -f ./Dockerfile`, creates a new Docker image with name `openjdk` and tag `8-dynatrace-appmon` in your local Docker registry. You (typically your CI/CD server) may now, with each application build, create a "latest" application image that extends `openjdk:8-dtappmon` like so (where `repo.internal` refers to a fictitious binary repository and `my-app` is a fictitious application). Note the possibility to override the `DT_AGENT_NAME` environment variable to be more specific than its definition in the base image.

```
FROM openjdk:8-dtappmon

ENV DT_AGENT_NAME "my-app"
ADD https://repo.internal/my-app/builds/latest.tar.gz /app

CMD java ${JAVA_OPTS} -jar /app/my-app.jar
```

Example: NGINX

Here is a `Dockerfile` that extends the official NGINX Docker image base image and downloads and extracts the agent. We set a number of environment variables, mostly for convenience, such as `DT_WSAGENT_NAME` and `DT_WSAGENT_COLLECTOR`, which you'll have to populate with your own values. Additionally, we provide a proper agent configuration in `DT_WSAGENT_INI` and override the container's `CMD` to load the agent with NGINX.

```
FROM nginx:1.9

ENV DT_WSAGENT_INSTALLER64_URL
"http://files.dynatrace.com/downloads/OnPrem/dynaTrace/6.5/6.5.0.1289/dynatrace-wsagen
t-6.5.0.1289-linux-x86-64.tar"

ENV DT                          "/dynatrace"
ENV DT_WSAGENT_INI              "${DT}/agent/conf/dtwsagent.ini"
ENV DT_WSAGENT_BIN64            "${DT}/agent/lib64/dtwsagent"
ENV DT_WSAGENT_LIB64            "${DT}/agent/lib64/libdtagent.so"

ENV DT_WSAGENT_NAME             "nginx"
ENV DT_WSAGENT_COLLECTOR        "127.0.0.1:9998"
ENV DT_WSAGENT_LOG_LEVEL        "info"

# Install the Dynatrace Application Monitoring Agent
ENV DT_INSTALL_DEPS             "curl"
RUN apt-get update && apt-get install -y --no-install-recommends ${DT_INSTALL_DEPS} &&
\
    curl -L -o /tmp/`basename ${DT_WSAGENT_INSTALLER64_URL}`
${DT_WSAGENT_INSTALLER64_URL} && \
    tar -C /tmp -xf /tmp/`basename ${DT_WSAGENT_INSTALLER64_URL}` && sh
/tmp/dynatrace-wsagent-*.sh && mv /dynatrace-* /dynatrace && \
    rm -f /tmp/dynatrace-* && apt-get remove --purge -y ${DT_INSTALL_DEPS} && rm -rf
/var/lib/apt/lists/* /tmp/*

# Configure the Dynatrace Application Monitoring WebServer Agent
RUN sed -i -r "s/^#?Name.*/Name ${DT_WSAGENT_NAME}/;s/^#?Server.*/Server
${DT_WSAGENT_COLLECTOR}/;s/^#?Loglevel.*/Loglevel
${DT_WSAGENT_LOG_LEVEL}/;s/^#?ConsoleLoglevel.*/ConsoleLoglevel
${DT_WSAGENT_LOG_LEVEL}/" ${DT_WSAGENT_INI}
CMD sh -c "(${DT_WSAGENT_BIN64} &) && LD_PRELOAD ${DT_WSAGENT_LIB64} nginx -g 'daemon
off;'"
```

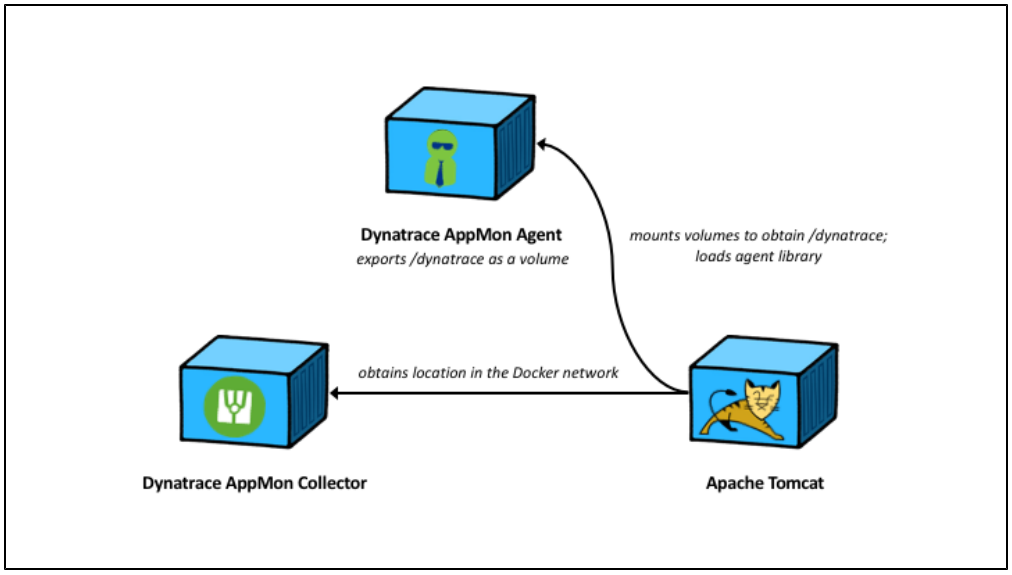
Analysis

- On the **pros** side we find that this approach neatly reduces the integration of a monitoring solution like Dynatrace to less frequent, preparatory work that does not add any overhead to the much more frequent process of building, shipping and running Dockerized applications.
- On the **cons** side, depending on your particular use-case and on the technologies you use, you'll have to integrate with each of these technologies by hand. And since this approach tightly ties together the agent with a particular technology in the same base image, these base images may altogether have to be recreated when switching to a new version of either the technology or of Dynatrace Application Monitoring.

Option B: Composition-based approach

With the "composition-based" approach, we equip you with the dynatrace/agent Docker image that includes all variants of the Dynatrace Application Montitoring agent -- that you can attach to your existing Docker containers by simple means of configuration!

Technically speaking, this approach uses a feature of Docker that allows a Docker container to export parts of its file system as Docker volumes a nd thereby make them obtainable by other, interested containers. Let's see what this looks like.



Examples

The following examples assume that you are already running a `dynatrace/agent` Docker container by name `dtagent` that exports the `/dynatrace` installation folder as a volume. Haven't heard the news? Our Dynatrace in Docker project on GitHub includes scripts to do exactly that and even allows you to conveniently set up an entire Dynatrace Application Montitoring environment in Docker. More information can be found in our Performance Clinic on "Dynatrace in Docker for Apps on Docker".

Example: Apache Tomcat

This `docker-compose.yml` file mounts the exported volumes of a container `dtagent` and initializes the `CATALINA_OPTS` environment variable with a proper `-agentpath`.

```
tomcat:
  image: tomcat
  ports:
  - 8080
  volumes_from:
  - dtagent
  environment:
    CATALINA_OPTS:
"-agentpath:/dynatrace/agent/lib64/libdtagent.so=name=tomcat,collector=127.0.0.1:9998"
  command: catalina.sh run
```

Example: NGINX

This `docker-compose.yml` file mounts the exported volumes of a container `dtagent` and overrides the container's `CMD` to load the agent with NGINX.

```
nginx:
  image: nginx
  ports:
  - 80
  volumes_from:
  - dtagent
  environment:
    DT_AGENT_NAME: "nginx"
    DT_AGENT_COLLECTOR: "127.0.0.1:6698"
    LD_PRELOAD: "/dynatrace/agent/lib64/libdtagent.so"
  command: sh -c “/dynatrace/run-wsagent.sh && nginx -g 'daemon off;'"
```

Analysis

- On the **pros** side we find that this approach neatly contributes to a clean separation of concerns, which is a design principle in the containers world. Also, you don't have to care about getting agents into your base images, a simple configuration at runtime is all you need to get your containers monitored.
- On the **cons** side, while the Docker runtime has great support for exchanging volumes between containers, doing so on a container orchestration platform such as Kubernetes or OpenShift can render your application configurations overly complex.

Q&A

Can I monitor applications running on Alpine in Docker?

As of now, the Dynatrace Application Montitoring agent does not support monitoring applications which are statically or dynamically linked against musl standard C library (instead of the more widely distributed and common glibc). As a temporary workaround (that will still save you a considerable amount of disk space) you could use a vanilla Alpine Docker image and install *glibc* together with your runtime environment, such as *Java*, on top. This means that you cannot install Alpine's OpenJDK using *apk* since it, like all Alpine packages, will depend on musl. Therefore, you may want to refer to anapsix/alpine-java as your base image, which, at the time of writing, combines *Alpine 3.4*, *glibc 2.23-r3* and *Oracle Java 1.8.0_71* out of the box.

```
FROM anapsix/alpine-java

ENV DT_AGENT_INSTALLER_URL
"http://files.dynatrace.com/downloads/OnPrem/dynaTrace/6.5/6.5.0.1289/dynatrace-agent-
6.5.0.1289-unix.jar"

ENV DT                    "/dynatrace"
ENV DT_AGENT_LIB32        "${DT}/agent/lib/libdtagent.so"
ENV DT_AGENT_LIB64        "${DT}/agent/lib64/libdtagent.so"

ENV DT_AGENT_NAME         "java"
ENV DT_AGENT_COLLECTOR    "127.0.0.1:9998"

ENV JAVA_OPTS
"-agentpath:${DT_AGENT_LIB64}=name=${DT_AGENT_NAME},collector=${DT_AGENT_COLLECTOR}"

# Install the Dynatrace Application Monitoring Agent
RUN wget -O /tmp/`basename ${DT_AGENT_INSTALLER_URL}` ${DT_AGENT_INSTALLER_URL} && \
    java -jar /tmp/`basename ${DT_AGENT_INSTALLER_URL}` -t ${DT} && \
    rm -f /tmp/`basename ${DT_AGENT_INSTALLER_URL}`
```

If you need an application server like Apache Tomcat on top, I suggest you have a look at one of their (glibc-based) Dockerfiles and augment the above example as needed, e.g. with Apache Tomcat 7 on Java 8.

Can I monitor Dockerized applications on Kubernetes or OpenShift?

Yes. Please look up the second part of this series.

Can I run easyTravel in Docker?

Sure! Dynatrace easyTravel has been fully Dockerized and is available at the Dynatrace easyTravel in Docker GitHub project. You can easily instrument it using Dynatrace in Docker.