Using AIQ with Docker/Containers

What is Docker?

Docker is a container management service that allows developers to build applications, ship them into containers and run them anywhere.

What is a container?

Is the encapsulation os an application with it dependencies, appears to be a lightweight form of virtual machine (VMs).

Facts about containers

- Share resources with the host OS (more efficient).
- Can be started and stopped in a fraction of the time a virtual machine would do it.
- Incurs into little to no overhead compared to applications running natively on the host OS.
- Allows changes in the running environment easily.
- A developer can run dozens of containers at the same time, making it possible to emulate a production-ready distributed system.
- Allows run complex applications without needing to spend hours of configuration and installation issues or worrying about the changes required for their system.
- This concept is not new, it's an old concept in UNIX systems, they had the chroot command, there have been different implementations across the time and different distributions. coming to what we are discussing today.

Docker Hub

Get started by downloading Docker Desktop, and learn how you can build, tag and share a sample image on Hub, please take a look to https://hub.docker.com/, create an account and log in, you will find an enormous number of public container images for download, avoiding duplicates work already done by others; in our case could use one of those images as the base to create a new one for AIQ.

Docker philosophy "Batteries included but replaceable", meaning that a complete solution would be provided, but parts could be swapped out.

One of the biggest use cases and strongest drivers behind the uptake of containers are microservices.

Installing Docker

The official way to install docker as below.

$ culr http://get.docker.com > /tmp/install.sh

A good install practice is checking the changes to be performed, do it as below.

$ cat /tmp/install.sh

Change the access permission for /tmp/install.sh.

$ chmod +x /tmp/install.sh

Issue the script file as below

$ /tmp/install.sh

A few checks, then install docker using the appropriate package for your system, this documentation does not include the installation procedure for Windows platforms, there are a lot of considerations beginning by Docker is native for Unix alike systems and there are not strong implementations and experiences in Windows.

In case of Red Hat based distributions, it is recommended you run SELLinux in permissive mode, which will log rather than enforce errors, check mode by issuing as below.

$ sestatus

If the current mode is "enforcing" then change into the permissive mode as follows.

$ setenforce 0

Additional tasks

As Docker is a privileged binary, it will need to prefix commands with sudo to avoid that add the current user to the docker group as follows.

$ sudo usermod -aG docker

The last command is for Ubuntu but would be similar for other Linux distributions, now restart the Docker service by performing.

$ sudo service docker restart

In order to test that the Docker has been properly installed run the command as follows.

$ docker run debian echo "hello world"

Some verbosity will be obtained, the last line must be "hello world"

Building images from Dockerfiles

Dockerfile is a plain text file containing instructions to create an image, a Docker image is a file, comprised of multiple layers, used to execute code in a Docker container.

Most common instructions are:

- FROM, specify the base image to use, the format for this instruction is "FROM <image>:<version>" latest version will be used if no version is specified e.g. (FROM ubuntu:18.04)
- WORKDIR, specify the default folder in the container.
- ENV, specify environmental variables in pairs e.g. (ENV folderTemp /tmp)
- LABEL, customized labels e.g. (version=1.0)
- EXPOSE, informs Docker that the container listens on the specified network ports at runtime e.g. (EXPOSE 80/tcp)
- USER, sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for anyRUN,CMDandENTRYPOINTinstructions that follow it in theDockerfile e.g. (USER patrick)
- VOLUME, creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers, these volumes will persist the data.
- COPY, This copies one or many local files or folders into the destination within your Docker image e.g. (COPY Gemfile Gemfile.lock ./)
- ADD, same that COPY but additionally If <source> is a local tar archive in a recognized compression format, then it is automatically unpacked as a directory into the Docker image. e.g. (ADD rootfs.tar.xz /)
- RUN, executes the command(s) that you give in a new layer and creates a new image. This is mainly used for installing a new package, e.g. (RUN apt-get update && apt-get install -y apache2)
- ENTRYPOINT, Entrypoint sets the command and parameters that will be executed first when a container is run, e.g. (ENTRYPOINT ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
- CMD, The main purpose of a CMD (Dockerfiles) / command (Docker Compose files) is to provide defaults when executing a container. These will be executed after the entrypoint.

There are more instructions and details over this stuff, but this document is not intended to be a Docker course but the necessary tools to implement AIQ in containers.

Best Practices

- Needs to be easy to destroy.
- One service per container.
- Avoid undesired files by declaring them in .dockerignore file.
- As few layers as possible.
- Use multiline in instructions for readability and layers reduction.
- Have the more arguments as possible.
- Avoid installing unnecessary packages.
- Adding labels that help and guide the usage.

AIQ container planning

Use containers to make the execution environment repeatable, but keep the AIQ controller, test-node, browser, network, and results-storage requirements aligned with the deployment topology. Validate image contents, mounted paths, credentials, and outbound connectivity before running a customer workload.

Operational checklist

- Pin the base image and document the Java, browser, driver, and AIQ versions included in it.
- Keep secrets outside the image and provide them through the approved CI/CD secret mechanism.
- Persist required scripts, reports, and logs using the documented workspace or artifact mechanism.
- Run a smoke scenario first, then compare container and non-container results before scaling execution.

Last updated: October 15, 2026

If you have any feedback on this topic, please contact us at documentation@appvance.ai

---

Revision #4
Created 2026-07-31 15:38:17 UTC by Admin
Updated 2026-11-04 16:58:59 UTC by AI_Mananger