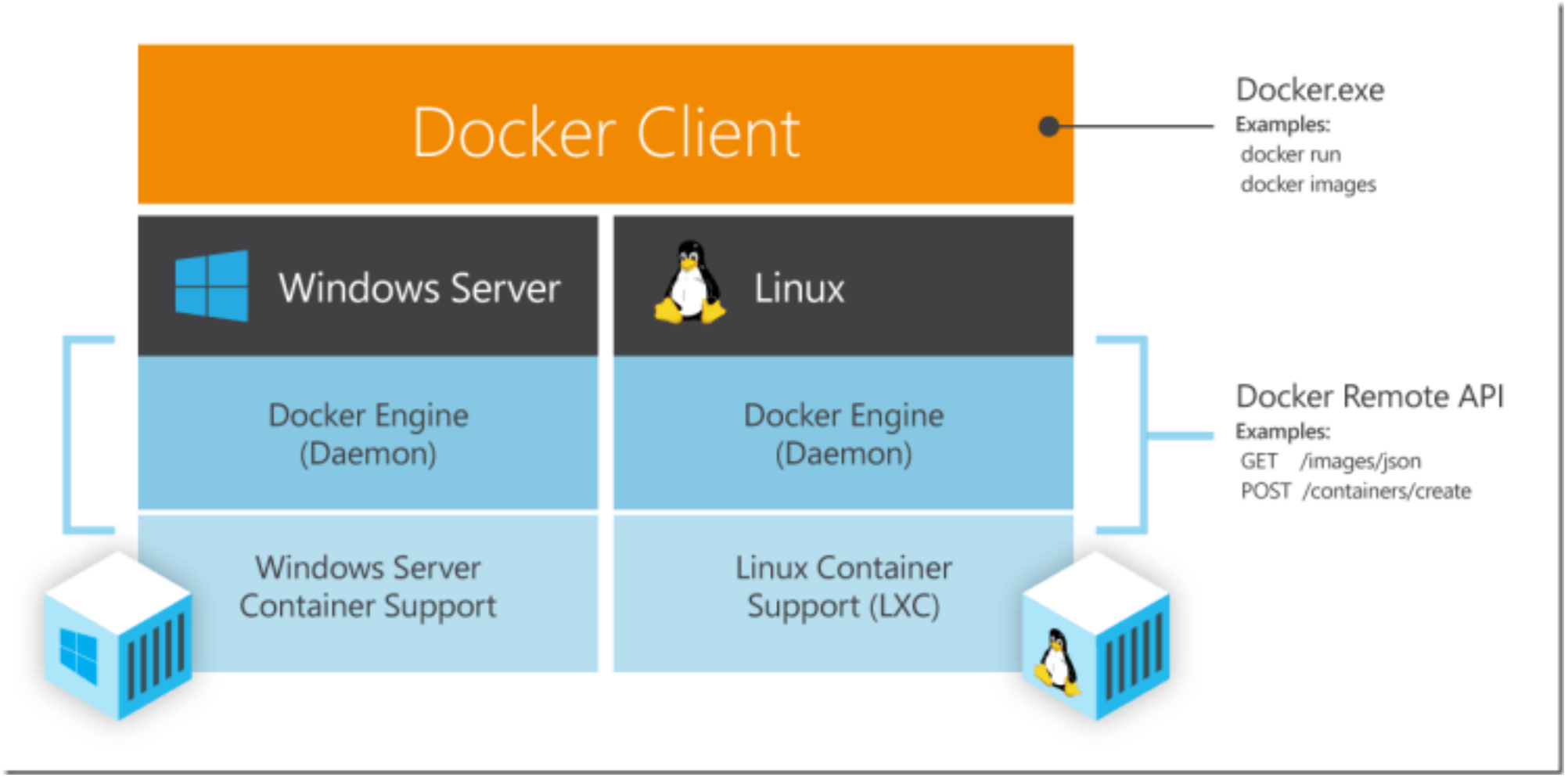
Розподілені системи та хмарні технології

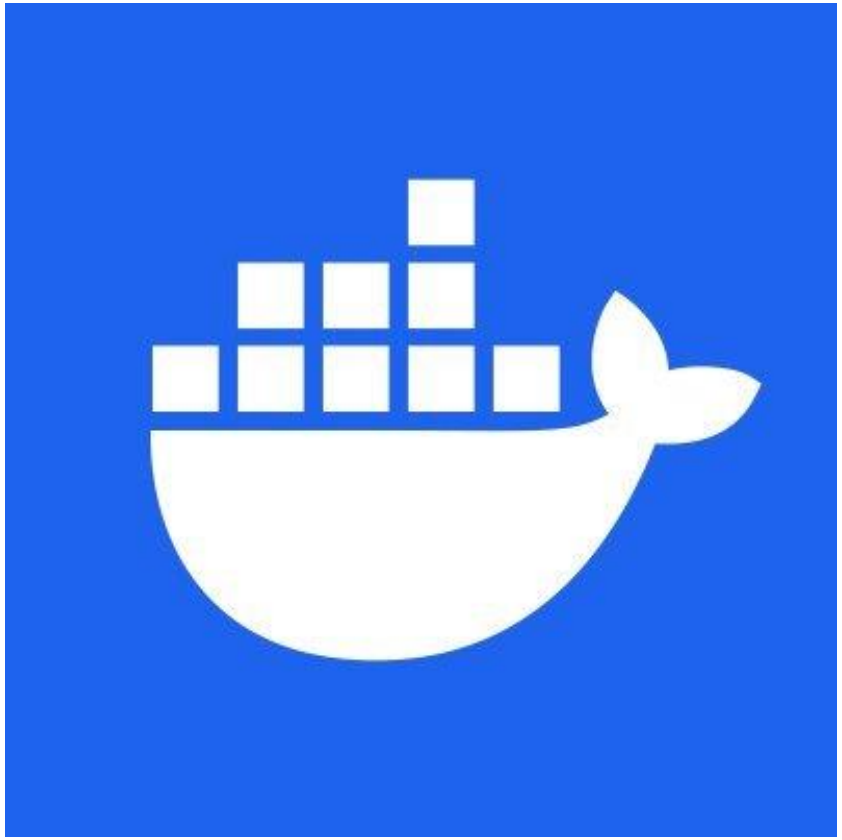
Контейнери в хмарних середовищах

Контейнер

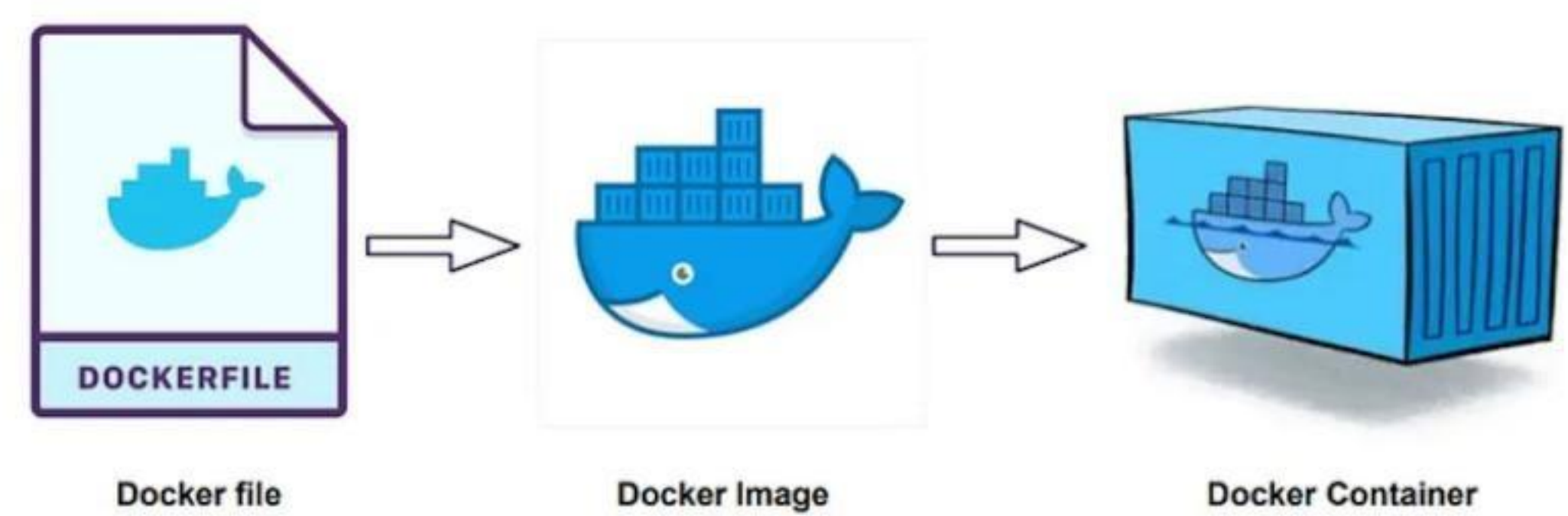
Windows vs Linux containers



Docker



Термінологія



Команди в Dockerfile

FROM
RUN
CMD
LABEL
MAINTAINER
EXPOSE
ENV
ADD
COPY
ENTRYPOINT
VOLUME
USER
WORKDIR
ARG
ONBUILD
STOPSIGNAL
HEALTHCHECK
SHELL

Приклад Dockerfile

```dockerfile
FROM mcr.microsoft.com/dotnet/sdk:8.0@sha256:35792ea4ad1db051981f62b313f1be3b46b1f45cadbaa3c288cd0d3056eefb83 AS build-env
WORKDIR /App

# Copy everything
COPY . ./
# Restore as distinct layers
RUN dotnet restore
# Build and publish a release
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:8.0@sha256:6c4df091e4e531bb93bdbfe7e7f0998e7ced344f54426b7e874116a3dc3233ff
WORKDIR /App
COPY --from=build-env /App/out .
ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]
```

Команди в Dockerfile

FROM

```
FROM [--platform=<platform>] <image> [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
```

Or

```
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]
```

Команди в Dockerfile

WORKDIR

```
WORKDIR /path/to/workdir
```

Команди в Dockerfile

COPY

COPY has two forms. The latter form is required for paths containing whitespace.

```
COPY [OPTIONS] <src> ... <dest>
COPY [OPTIONS] ["<src>", ... "<dest>"]
```

Команди в Dockerfile

COPY

COPY has two forms. The latter form is required for paths containing whitespace.

```
COPY [OPTIONS] <src> ... <dest>
COPY [OPTIONS] ["<src>", ... "<dest>"]
```

Команди в Dockerfile

COPY --from

By default, the `COPY` instruction copies files from the build context. The `COPY --from` flag lets you copy files from an image, a build stage, or a named context instead.

```
COPY [--from=<image|stage|context>] <src> ... <dest>
```

To copy from a build stage in a multi-stage build, specify the name of the stage you want to copy from. You specify stage names using the `AS` keyword with the `FROM` instruction.

```
# syntax=docker/dockerfile:1
FROM alpine AS build
COPY . .
RUN apk add clang
RUN clang -o /hello hello.c

FROM scratch
COPY --from=build /hello /
```

Команди в Dockerfile

COPY --chown --chmod

Note

Only octal notation is currently supported. Non-octal support is tracked in moby/buildkit#1951.

```
COPY [--chown=<user>:<group>] [--chmod=<perms> ...] <src> ... <dest>
```

```
COPY --chown=55:mygroup files* /somedir/
COPY --chown=bin files* /somedir/
COPY --chown=1 files* /somedir/
COPY --chown=10:11 files* /somedir/
COPY --chown=myuser:mygroup --chmod=644 files* /somedir/
```

Команди в Dockerfile

COPY --chown --chmod

Note

Only octal notation is currently supported. Non-octal support is tracked in moby/buildkit#1951.

```
COPY [--chown=<user>:<group>] [--chmod=<perms> ...] <src> ... <dest>
```

```
COPY --chown=55:mygroup files* /somedir/
COPY --chown=bin files* /somedir/
COPY --chown=1 files* /somedir/
COPY --chown=10:11 files* /somedir/
COPY --chown=myuser:mygroup --chmod=644 files* /somedir/
```

Команди в Dockerfile

RUN

The `RUN` instruction will execute any commands to create a new layer on top of the current image. The added layer is used in the next step in the Dockerfile. `RUN` has two forms:

```
# Shell form:
RUN [OPTIONS] <command> ...
# Exec form:
RUN [OPTIONS] [ "<command>", ... ]
```

For more information about the differences between these two forms, see shell or exec forms.

The shell form is most commonly used, and lets you break up longer instructions into multiple lines, either using newline escapes, or with heredocs:

```
RUN <<EOF
apt-get update
apt-get install -y curl
EOF
```

Команди в Dockerfile

ENTRYPOINT

An `ENTRYPOINT` allows you to configure a container that will run as an executable.

`ENTRYPOINT` has two possible forms:

- The exec form, which is the preferred form:

```
ENTRYPOINT ["executable", "param1", "param2"]
```

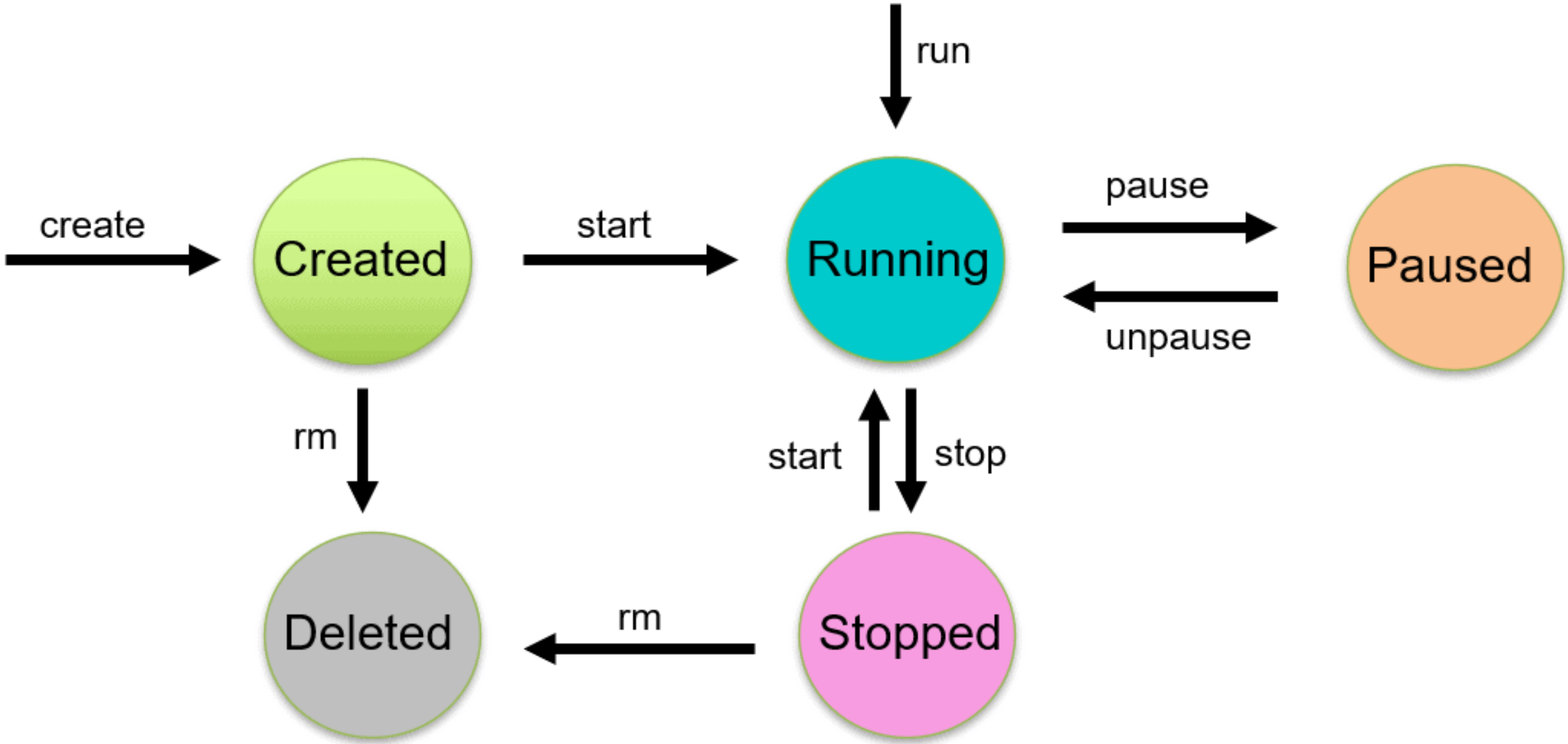
- The shell form:

```
ENTRYPOINT command param1 param2
```

For more information about the different forms, see Shell and exec form.

The following command starts a container from the `nginx` with its default content, listening on port 80:

```
$ docker run -i -t --rm -p 80:80 nginx
```


run
create
Created
start
Running
pause
Paused
unpause
rm
start
stop
Deleted
rm
Stopped

DEMO

- Run a simple Nginx and Ubuntu containers locally
- Run a containerized .NET app
- Container states
- Azure Container Registry
- Azure Container Instance