Docker: Up and Running

Docker Images

Presented by Sean P. Kane

https://techlabs.sh/

Release 2023-10-20-1111

Instructor

Sean P. Kane

@spkane

./ techlabs

https://techlabs.sh

Follow Along Guide

Textual Slides

O'Reilly Online Sandbox VM

https://learning.oreilly.com/scenarios/devops-tools-sandbox/9781098126469/

NOTE: *VM sessions will expire after 60 minutes!*

Prerequisites (1 of 3)

NOTE: You *MUST* have open access to Github if you want to participate in the hands-on portion of class from your local computer system.

Prerequisites (2 of 3)

- A recent computer and OS
  - Recent/Stable Linux, macOS, or Windows 10+
  - root/admin rights
  - Sufficient resources to run one 2 CPU VM
  - CPU Virtualization MUST be enabled in your BIOS/EFI
  - Reliable and fast internet connectivity
- Docker Community/Desktop Edition (on Linux also install Docker Compose V2)

Prerequisites (3 of 3)

- A graphical web browser
- A text editor
- A software package manager
- `git` client
- General comfort with the command line will be helpful.
- [optional] `tar`, `wget`, `curl`, `jq`, `ssh` client
  - `curl.exe` for Windows - https://curl.se/windows/

A Note for Powershell Users

Terminal commands reflect the Unix bash shell. PowerShell users will need to adjust the commands.

- Unix Shell Variables
  - `export MY_VAR='test'`
  - `echo ${MY_VAR}`
- PowerShell Variables
  - `$env:my_var = "test"`
  - `Get-ChildItem Env:my_var`
  - `Remove-Item Env:\my_var`

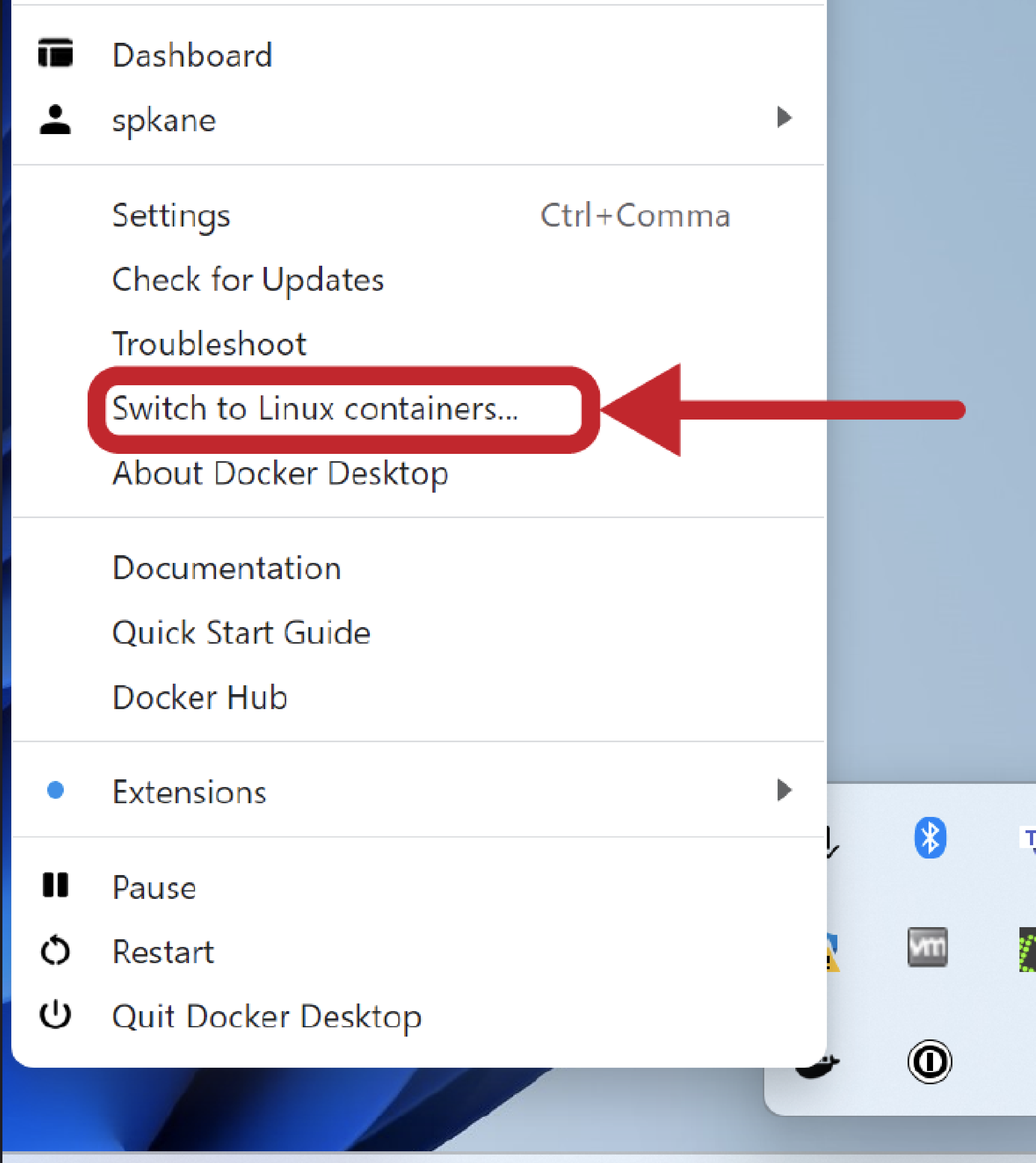
Translation Key

`\` - Unix Shell Line Continuation
`` ` `` - Powershell Line Continuation (sort of)

`${MY_VAR}` - Is generally a place holder in the slides.

Linux Container Mode

- On the Windows platform make sure that you are running in Linux Container mode.
- *WARNING:* On Windows, avoid using Git Bash. WSL2 is highly recommended instead.



A Note About Proxies & VPNs

Proxies can interfere with some Docker activities if they are not configured correctly and VPNs can increase audio and video streaming issues in class.

- Configure `HTTP_PROXY`, `HTTPS_PROXY`, and your web browser.

If required, you can configure a proxy in Docker Desktop via the preferences.

- Docker
- Docker-Compose

Instructor Environment

- **Operating System**: macOS (v14.X.X+)
- **Terminal**: iTerm2 (Build 3.X.X+) - https://www.iterm2.com/
- **Shell Prompt Theme**: Starship - https://starship.rs/
- **Shell Prompt Font**: Fira Code - https://github.com/tonsky/FiraCode
- **Text Editor**: Visual Studio Code (v1.X.X+) - https://code.visualstudio.com/

-
  ```
  export BUILDKIT_COLORS=run=green:warning=yellow:error=red:cancel=cyan
  ```

Docker in Translation

- **Docker client**
  - The docker command used to control most of the Docker workflow and talk to remote Docker servers.
- **Docker server**
  - The dockerd command used to launch the Docker daemon. This turns a Linux system into a Docker server that can have containers deployed, launched, and torn down via a remote client.

The Docker Server

- **Virtual Machine** or **Bare Metal**
    - In general, the Docker server can only be run on Linux. If you are running Linux on your system, you may have the server running natively on bare metal. On Windows and macOS, it is common to utilize a Linux virtual machine to run the server and Docker Desktop Edition takes care of this for you.
    - In class, it makes no difference. If I refer to the "virtual machine" or the "docker server" know that these terms are interchangeable for class and it does not really matter how you are setup locally.

Docker in Translation

- **Docker/OCI images**
  - Docker images consist of one or more filesystem layers and some important metadata that represent all the files required to run a Dockerized application. A single Docker image can be copied to numerous hosts. A container will typically have both a name and a tag. The tag is generally used to identify a particular release of an image.
  - OCI - Open Container Initiative

Docker in Translation

- **Linux Containers**
  - A Linux Container is a single instantiation of a Docker or OCI-standard image. A specific container can only exist once; however, you can easily create multiple containers from the same image.

Testing the Docker Setup

```
$ docker image ls
$ docker container run -d --rm --name quantum \
    --publish mode=ingress,published=18080,target=8080 \
    docker.io/spkane/quantum-game:latest
$ docker container ls
```

- In a web browser, navigate to port 18080 on your Docker server.

```
$ docker container stop quantum
$ docker container ls
```

Exploring the Dockerfile

```
$ cd ${HOME}
$ mkdir class
$ cd ${HOME}/class
$ git clone https://github.com/spkane/balance_game.git
$ cd balance_game
```

- Open & explore Dockerfile in your text editor

Full Documentation:
https://docs.docker.com/engine/reference/builder/

Registering with Docker Hub

Create an account at: https://hub.docker.com/

Create Your Image Repository

- **Login**: `https://hub.docker.com/`
- **Click**: `Create Repository+`
- **Enter name**: `balance_game`
- **Set visibility**: `public`
- **Click**: `Create`

Docker Login

```
$ docker login
$ cat ~/.docker/config.json
```

Registry Authentication

```
{
    "auths": {
        "https://index.docker.io/v1/": {
            "auth": "q378t348q7tb78bfs387b==",
            "email": "me@example.com"
        }
    }
}
```

Building Your First Image

```
$ export HUB_USER=${USER}
$ export BUILDKIT_COLORS=run=green:warning=yellow:error=red:cancel=cyan
$ docker image build -t docker.io/${HUB_USER}/balance_game:latest .
```

Testing and Uploading the Image

```
$ docker container run -d --rm --name balance_game \
    --publish mode=ingress,published=18081,target=80 \
    docker.io/${HUB_USER}/balance_game:latest
$ docker container stop balance_game
$ docker image push docker.io/${HUB_USER}/balance_game:latest
$ docker search ${HUB_USER}
```

Docker Hub API Examples

```
$ curl -s -S \
  "https://registry.hub.docker.com/v2/repositories/library/alpine/tags/" \
  | jq '."results"[]["name"]' | sort
```

Launch a Typical Build Process

```
$ docker image build --progress=plain --no-cache \
    -t docker.io/${HUB_USER}/balance_game:latest .
```

A Typical Build Process - Simplified (1 of 2)

1. Client loads build definition from `Dockerfile`.
2. Client loads the `.dockerignore` file.
3. Client transfers the build context to the server.
4. Client loads metadata for the base image.
5. Client authentication against the registry.
6. Client pulls base image.

A Typical Build Process - Simplified (2 of 2)

7. New intermediate container created from base image.
    - or empty container created, if using `FROM scratch`.
8. Command executed inside intermediate container.
9. New image/layer created from intermediate container snapshot.
10. If there is another step, a new intermediate container is created from the last step, and then the build goes back to step 7.

Advanced Dockerfile Techniques

Keep it Small

- Each and every layer's size matters
- Don't install unnecessary files

```
$ cd ${HOME}/class
$ docker container run --rm -d --name outyet-small \
    --publish mode=ingress,published=8090,target=8080 \
    docker.io/spkane/outyet:1.9.4-small
$ docker container export outyet-small -o export.tar
$ tar -tvf export.tar
$ docker container stop outyet-small
$ rm export.tar
```

Debugging an Image

- If your image has a shell installed you can access it using a command like this:

```
$ docker container run --rm -ti docker.io/spkane/outyet:1.9.4 \
    /bin/sh
```

Debugging an Image

- But without a shell in the image this will fail.

```
$ docker image ls
$ docker container run --rm -ti docker.io/spkane/outyet:1.9.4-small \
    /bin/sh
```

Debugging an Image

So, let's fix this:

```
$ git clone https://github.com/spkane/outyet.git
$ cd outyet
```

Multi-Stage Images

```
FROM golang:1.9.4 as builder

COPY . /go/src/outyet
WORKDIR /go/src/outyet

ENV CGO_ENABLED=0
ENV GOOS=linux

RUN go get -v -d && \
    go install -v && \
    go test -v && \
    go build -ldflags "-s" -a -installsuffix cgo -o outyet .
```

Multi-Stage Images

```dockerfile
# To support debugging, let's use alpine instead of scratch
FROM alpine:latest as deploy

# Since we are using alpine we can simply install these
RUN apk --no-cache add ca-certificates

WORKDIR /
COPY --from=builder /go/src/outyet/outyet .
# (or) COPY --from=0 /go/src/outyet/outyet .

EXPOSE 8080

CMD ["/outyet", "-version", "1.9.4", "-poll", "600s", "-http", ":8080"]
```

Building the Improved Image

```
$ docker image build -f Dockerfile -t outyet:1.9.4-local .
```

Debugging an Image

- Now that we have a shell, let's try this again:

```
$ docker image ls
$ docker container run --rm -ti outyet:1.9.4-local /bin/sh
```

- Once inside the new container:

```
$ ls -lFa /outyet
$ exit
```

Debugging a Broken Build (1 of 4)

- Break the `Dockerfile` and then try building it again.

```
RUN go getit -v -d && \
    go install -v && \
    go test -v && \
    go builder -ldflags "-s" -a -installsuffix cgo -o outyet .
```

```
$ docker image build -t outyet:debug --no-cache .
```

Debugging a Broken Build (2 of 4)

- Add another stage and build the known working stages.

```
FROM builder as debug
RUN go getit -v -d && \
    go install -v && \
    go test -v && \
    go builder -ldflags "-s" -a -installsuffix cgo -o outyet .
```

```
$ docker image build -t outyet:debug --target builder --no-cache .
```

Debugging a Broken Build (3 of 4)

- Let's debug the last successful image in that build

```
$ docker container run --rm -ti outyet:debug
```

- Once inside the new container:

```
$ go getit -v -d
$ go get -v -d
$ exit
```

Debugging a Broken Build (4 of 4)

- **Important**: Always, remember to apply your fixes to the `Dockerfile` and remove the extra stage that you added before committing the changes.

```
RUN go get -v -d && \
    go install -v && \
    go test -v && \
    go builder -ldflags "-s" -a -installsuffix cgo -o outyet .
```

Smart Layering

- Each and every layer's size matters
- Clean up, inside of each step.

```
$ cd ${HOME}/class
$ cd balance_game
$ docker image build -f Dockerfile.fedora .
$ docker image tag ${IMAGE_ID} size${#}
$ docker image history size${#}
```

Smart Layering

- edit `Dockerfile.fedora`, build, and re-examine size

```
RUN dnf install -y httpd
RUN dnf clean all
```

```
RUN dnf install -y httpd && \
    dnf clean all
```

Timing commands in Windows

- In the next exercise we will be timing commands using a Unix utility. If you are on Windows and want to try to time these commands locally, you can try something like this in Powershell.

```
PS C:\> $t = Measure-Command { docker image build --no-cache . }
PS C:\> Write-Host That command took $t.TotalSeconds to complete.
```

Order Matters

- Keep commands that change towards the end of your Dockerfile.

```
$ cd ${HOME}/class
$ cd balance_game
$ time docker image build --no-cache .
$ time docker image build .
```

Order Matters

- edit `start.sh`

```
$ time docker image build .
```

Order Matters

- Add to the top of Dockerfile:

```
ADD start.sh /
```

- And then remove the same line from the bottom of the Dockerfile.

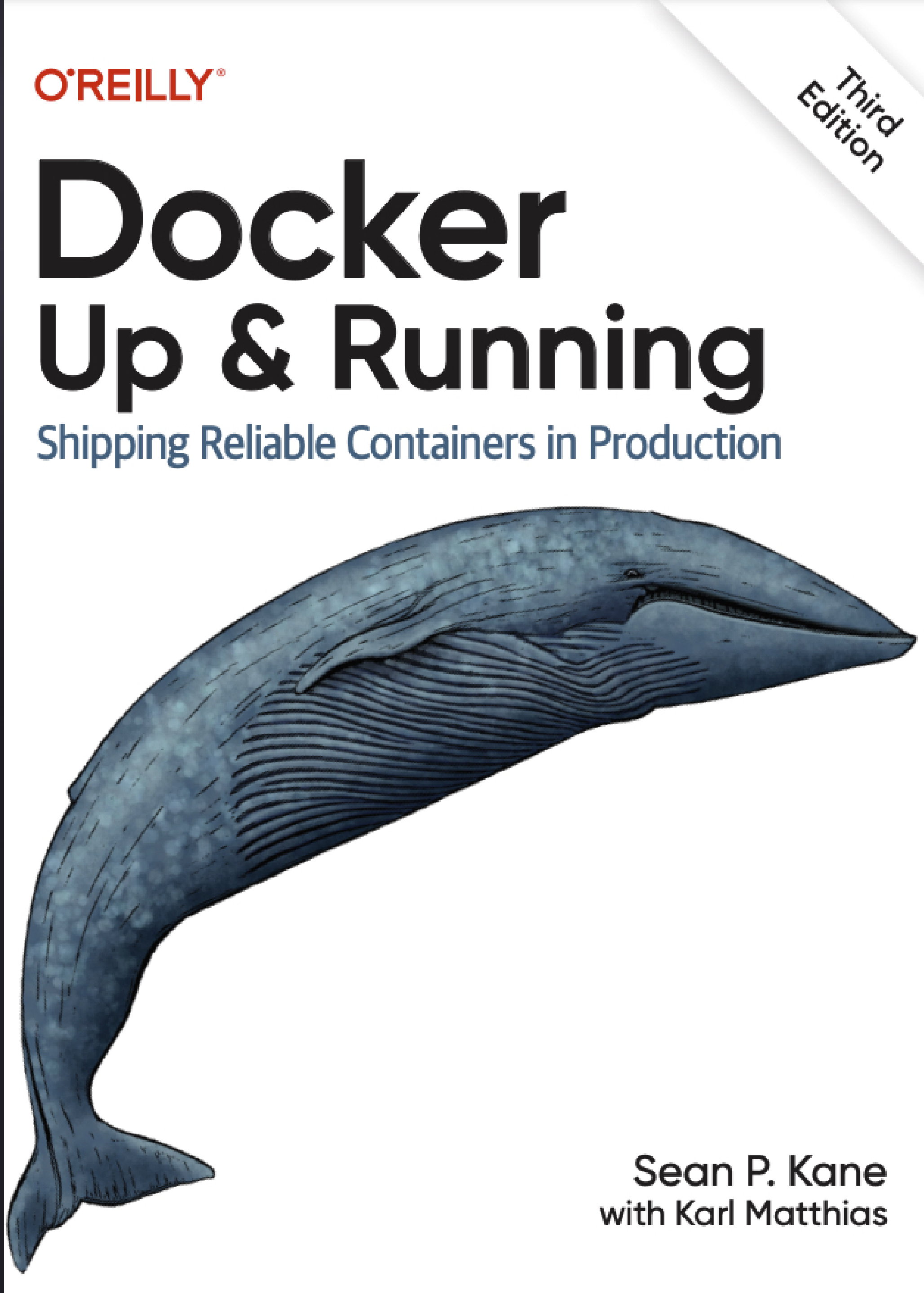
```
$ time docker image build --no-cache .
$ time docker image build .
$ vi start.sh
$ time docker image build .
```

What We Have Learned

- What a Dockerfile is
- Building a Docker image
- Using Docker Hub
- Keeping Images Small
- Keeping Builds Fast
- Multi-Stage Dockerfiles
- Debugging Images

Additional Reading

- The 12-Factor App
  - http://12factor.net/
- Official Docker Documentation
  - https://docs.docker.com/
- Docker: Up and Running
  - https://dockerupandrunning.com/



Additional Learning Resources

https://learning.oreilly.com/

Student Survey

Please take a moment to fill out the class survey linked to from the bottom of the ON24 audience screen.

O’Reilly and I value your comments about the class.

Thank you!

Any Questions?

[Sean P. Kane](https://techlabs.sh/)

techlabs

Hands-on technical training and engineering

https://techlabs.sh/