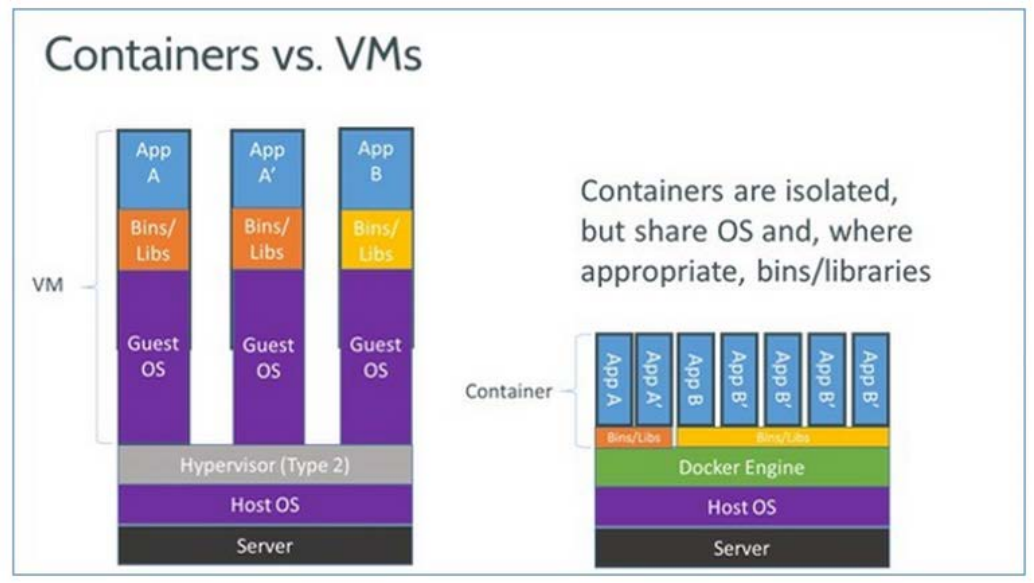
DYALOG

Belfast 2018

Workshop TP2
APL in the Cloud
Docker

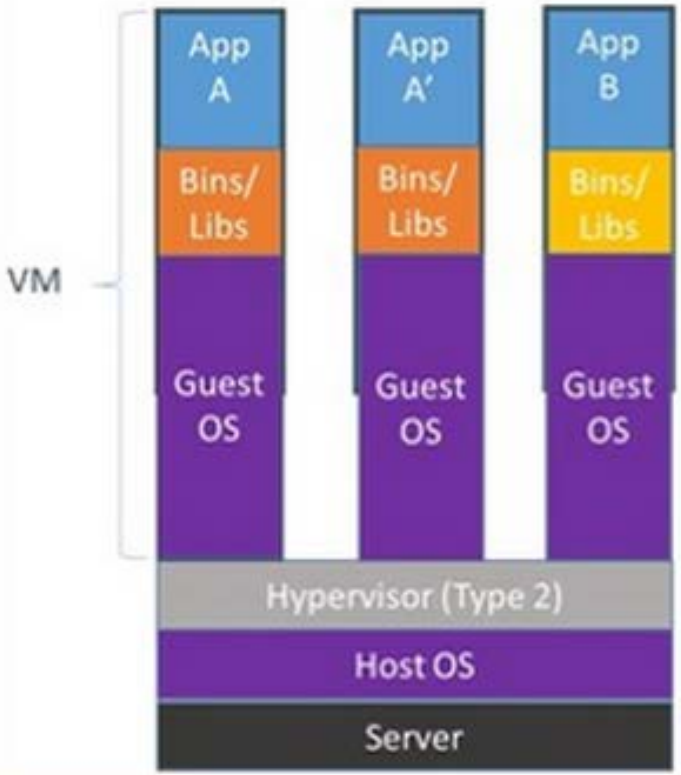
Morten Kromberg, CXO, Dyalog

Why Containers ?

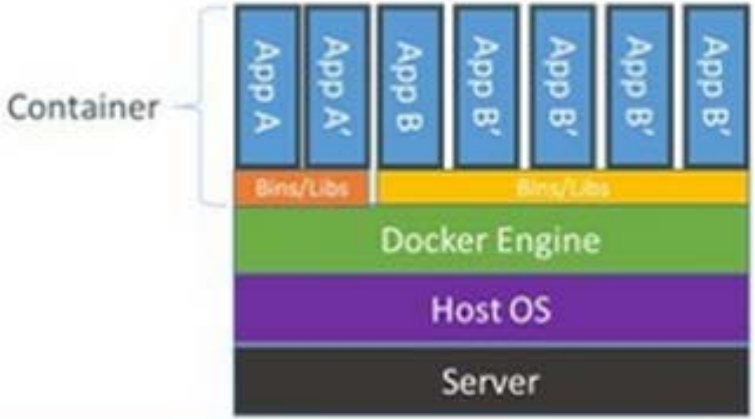


From:
http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries



Linux

Linux

- Container technology works best with Linux, due to the size of the kernel

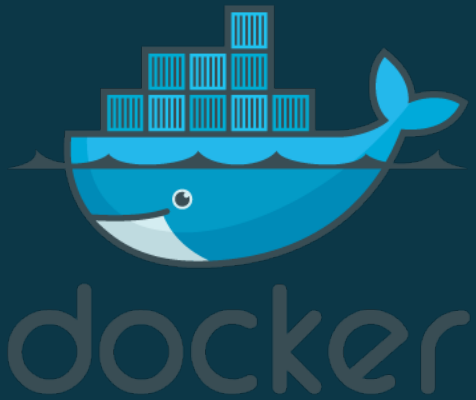
Linux

- Container technology works best with Linux, due to the size of the kernel
- Windows kernels are getting smaller but are still 10-20x as large as Linux (~0.5-1Gb vs 50Mb).

Linux

- Container technology works best with Linux, due to the size of the kernel
- Windows kernels are getting smaller but are still 10-20x as large as Linux (~0.5-1Gb vs 50Mb).
- Good News: Your Dyalog APL code will run unchanged under Linux.
  - So long as it doesn't call Windows APIs

Docker for Windows



- Docker for Windows uses Microsoft Hyper-V to run either Linux or Windows virtual machines.
- It provides the same command line interface as Docker under Linux

```
docker build –t myco/myapp-test .
```

```
docker push myco/myapp-test
```

```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```

Install Docker for Windows

Estimated reading time: 4 minutes

Docker for Windows is the Community Edition (CE) of Docker for Microsoft Windows. To download Docker for Windows, head to Docker Store.

Download from Docker Store

What to know before you install

- **README FIRST for Docker Toolbox and Docker Machine users**: Docker for Windows requires Microsoft Hyper-V to run. The Docker for Windows installer enables Hyper-V for you, if needed, and restart your machine. After Hyper-V is enabled, VirtualBox no longer works, but any VirtualBox VM images remain. VirtualBox VMs created with `docker-machine` (including the `default` one typically created during Toolbox install) no longer start. These VMs cannot be used side-by-side with Docker for Windows. However, you can still use `docker-machine` to manage remote VMs.
- **System Requirements**:
  - Windows 10 64bit: Pro, Enterprise or Education (1607 Anniversary Update, Build 14393 or later).
  - Virtualization is enabled in BIOS. Typically, virtualization is enabled by default. This is different from having Hyper-V enabled. For more detail see Virtualization must be enabled in Troubleshooting.

Edit this page
Request docs changes
Get support

On this page:

What to know before you install
About Windows containers
Install Docker for Windows desktop app
Start Docker for Windows
Where to go next

Install Docker for Windows

Estimated reading time: 4 minutes

Docker for Windows is the Community Edition (CE) of Docker for Microsoft Windows. To download Docker for Windows, head to Docker Store.

Download from Docker Store

What to know before you install

- **README FIRST for Docker Toolbox and Docker Machine users**: Docker for Windows requires Microsoft Hyper-V to run. The Docker for Windows installer enables Hyper-V for you, if needed, and restart your machine. After Hyper-V is enabled, VirtualBox no longer works, but any VirtualBox VM images remain. VirtualBox VMs created with `docker-machine` (including the `default` one typically created during Toolbox install) no longer start. These VMs cannot be used side-by-side with Docker for Windows. However, you can still use `docker-machine` to manage remote VMs.
- **System Requirements**:
  - Windows 10 64bit: Pro, Enterprise or Education (1607 Anniversary Update, Build 14393 or later).
  - Virtualization is enabled in BIOS. Typically, virtualization is enabled by default. This is different from having Hyper-V enabled. For more detail see Virtualization must be enabled in Troubleshooting.

Installing Docker + docker-compose

```
Ubuntu

sudo snap install docker
snap services docker
sudo groupadd docker
sudo usermod -aG docker mkrom
(log out and in again)

Downloaded docker-compose from
 github.com/docker/compose/releases
sudo mv Downloads/docker-compose-Linux-x86-64
/usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

```
Amazon Linux

sudo yum update -y
sudo yum install -y docker
sudo service docker start
sudo usermod -a -G docker ec2-user
(log out and in again)

sudo pip install docker-compose
docker-compose --version
sudo yum install git
```

Container Basics

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Container Basics

Base Image

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Container Basics

Base Image

Files to Add

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Container Basics

Base Image

Files to Add

Run ***during Build***

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Container Basics

- Base Image
- Files to Add
- Run ***during Build***
- Environment Vars

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Container Basics

Base Image

Files to Add

Run ***during Build***

Environment Vars

Run ***at Startup***

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Container Basics

Base Image

Files to Add

Run ***during Build***

Environment Vars

Run ***at Startup***

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

This "Dockerfile" completely describes a machine which will run "myapp".

Container Basics

Base Image

Files to Add

Run ***during Build***

Environment Vars

Run ***at Startup***

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp

RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS


ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Your Code

This "Dockerfile" completely describes a machine which will run "myapp".

Container Basics

Base Image

Files to Add

Run ***during Build***

Environment Vars

Run ***at Startup***

```
FROM ubuntu:18.04

ADD ./dyalog-unicode_17.0.34604_amd64.deb /


RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS
RUN git clone https://github.com/myco/myapp /myapp

ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp

CMD dyalog /JSS/JSONServer.dws
```

Your Code

Uses GitHub to load the source code for "myapp".

Building and Running the Docker Image

Dockerfile

```
FROM ubuntu:18.04
ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp
RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

Building and Running the Docker Image

Dockerfile

```
FROM ubuntu:18.04
ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp
RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

Build

```
docker build –t myco/myapp-test .
```

Building and Running the Docker Image

Dockerfile

```
FROM ubuntu:18.04
ADD ./dyalog-unicode_17.0.34604_amd64.deb /
ADD /myapp/v7/test /myapp
RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

Build

```
docker build –t myco/myapp-test .
```

Run

```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```

docker run syntax & common switches

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

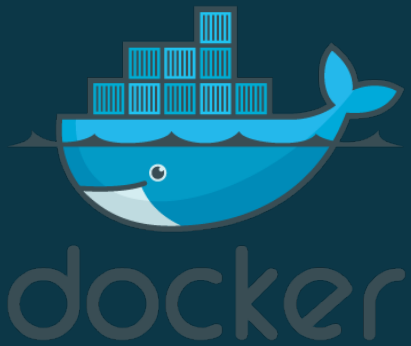
```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```

Switch	Description
`-p hhhh:cccc`	Map TCP port cccc in container to hhhh on host
`-e name=value`	Set environment variable inside the container
`-v /hfolder:/cfolder`	Mount /hfolder in container as /cfolder
`-t`	Allocate a pseudo-TTY
`-i`	Keep stdin open even if not attached
`--rm`	Discard changes when container terminates

Distributing the Image: DockerHub

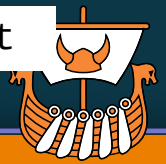
Build

```
docker build –t myco/myapp-test .
```



Run

```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```

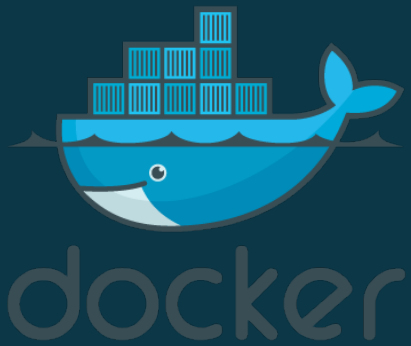


Distributing the Image: DockerHub

Build

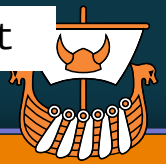
```
docker build –t myco/myapp-test .
```

We can "push" the image to DockerHub:



Run

```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```



Distributing the Image: DockerHub

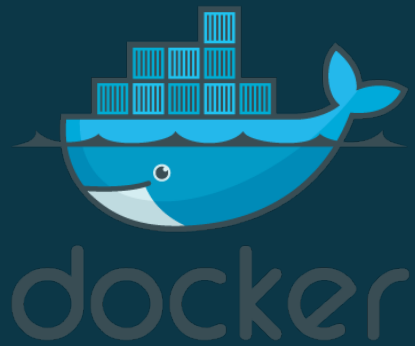
Build

```
docker build –t myco/myapp-test .
```

We can "push" the image to DockerHub:

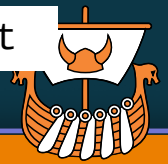
Push

```
docker login
docker push myco/myapp-test
```



Run

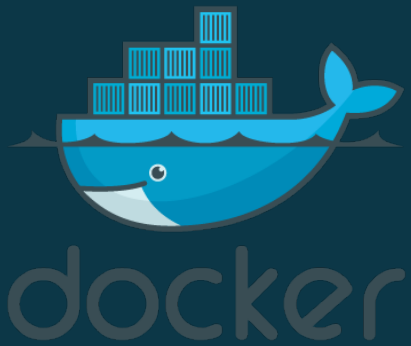
```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```



Distributing the Image: DockerHub

Build

```
docker build –t myco/myapp-test .
```



We can "push" the image to DockerHub:

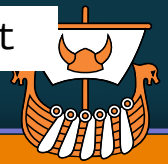
Push

```
docker login
docker push myco/myapp-test
```

Now, the following will work on ANY computer that has Docker installed
(assuming myco/myapp-test is a **PUBLIC** container)

Run

```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```



Distributing the Image: DockerHub

Build

```
docker build –t myco/myapp-test .
```

We can "push" the image to DockerHub:

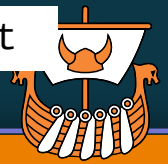
Push

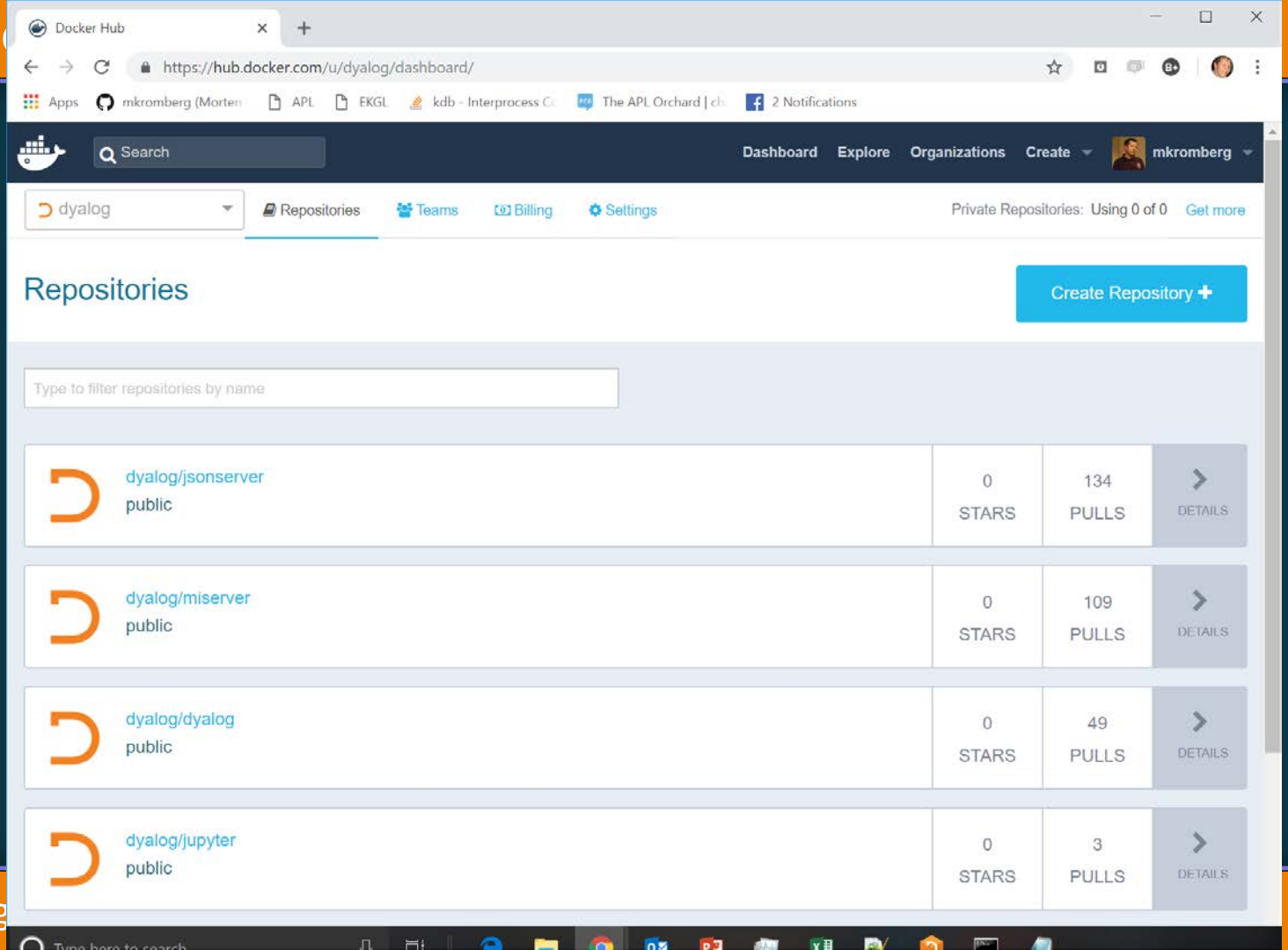
```
docker login
docker push myco/myapp-test
```

Now, the following will work on ANY computer that has Docker installed
(assuming myco/myapp-test is a **PUBLIC** container)

Run

```
docker run -p 8081:8080 -v /somefolder:/data –e DEBUG=1 myco/myapp-test
```




Docker Hub
https://hub.docker.com/u/dyalog/dashboard/
Apps
mkromberg (Morten
APL
EKGL
kdb - Interprocess C
The APL Orchard | ch
2 Notifications
Search
Dashboard
Explore
Organizations
Create
mkromberg
dyalog
Repositories
Teams
Billing
Settings
Private Repositories: Using 0 of 0
Get more
Repositories
Create Repository +
Type to filter repositories by name
dyalog/jsonserver
public
0
STARS
134
PULLS
DETAILS
dyalog/miserver
public
0
STARS
109
PULLS
DETAILS
dyalog/dyalog
public
0
STARS
49
PULLS
DETAILS
dyalog/jupyter
public
0
STARS
3
PULLS
DETAILS

Public Dyalog Containers

These currently for experimentation only and are based on UNSUPPORTED NON-COMMERCIAL Dyalog 17.1.

All run full development interpreters in interactive terminal mode.

`dyalog/dyalog:17.1-dbg`
- Linux + Dyalog APL Interpreter

`dyalog/jsonserver:dbg`
- dyalog:17.1-dbg + JSONServer

`dyalog/miserver:dbg`
- dyalog:17.1-dbg + MiServer

`dyalog/jupyter`
- dyalog:17.1-dbg + Python, Anaconda & Jupyter Notebook

Dyalog Scripts

Dyalog Scripts

`dyalog-c` `folder [rideport]`

- Starts container dyalog/dyalog:17.1-dbg

`folder` is always mounted as `/app` in the container

`rideport` is the optional port that RIDE can be attached to

Dyalog Scripts

`dyalog-c` `folder [rideport]`
- Starts container dyalog/dyalog:17.1-dbg

`jsonserver-c` `folder [[httpport] [rideport]]`
- Starts container dyalog/jsonserver-dbg

`folder` is always mounted as `/app` in the container
`httpport` is the application port that is always exposed by json- & mi-servers
`rideport` is the optional port that RIDE can be attached to

Dyalog Scripts

`dyalog-c` `folder [rideport]`
- Starts container dyalog/dyalog:17.1-dbg

`jsonserver-c` `folder [[httpport] [rideport]]`
- Starts container dyalog/jsonserver-dbg

`miserver-c` `folder [[httpport] [rideport]]`
- Starts container dyalog/miserver-dbg

`folder` is always mounted as `/app` in the container
`httpport` is the application port that is always exposed by json- & mi-servers
`rideport` is the optional port that RIDE can be attached to

Dyalog Scripts

`dyalog-c` `folder [rideport]`
- Starts container dyalog/dyalog:17.1-dbg

`jsonserver-c` `folder [[httpport] [rideport]]`
- Starts container dyalog/jsonserver-dbg

`miserver-c` `folder [[httpport] [rideport]]`
- Starts container dyalog/miserver-dbg

`jupyter-c` `[folder[/notebook]] [httpport]`
- Starts container dyalog/jupyter (Jupyter notebook server)

`folder` is always mounted as `/app` in the container
`httpport` is the application port that is always exposed by json- & mi-servers
`rideport` is the optional port that RIDE can be attached to

jsonserver-c.bat - Notepad

File Edit Format View Help

```
REM @ECHO OFF
SETLOCAL
REM Start JSONServer Container
REM Usage jsonserver-c directory [rideport]

SET pname=%~0

IF "%1"=="" (GOTO USAGE) ELSE (SET folder="%~f1")
IF "%2"=="" (SET httpport=8080) ELSE (SET httpport=%2)
IF NOT EXIST %folder% (GOTO NOTEXIST) ELSE (
  IF %3=="" (
     START docker run -p %httpport%:8080 -it -v %folder%:/app dyalog/jsonserver:dbg
     ) ELSE (
     START docker run -p %httpport%:8080 -p %3:4502 -it -v %folder%:/app dyalog/jsonserver:dbg
     )
  IF NOT %ERRORLEVEL%==0 (GOTO Exit) ELSE (
     START http://localhost:%httpport%
     goto DONE
  )
)

:ERROR
ECHO "%pname% Docker returned error %ERRORLEVEL%"
goto DONE
```

jsonserver-c - Notepad

File Edit Format View Help

```
#!/bin/bash

pname=`basename $0`

if [ $# = 0 ]; then echo "Usage: $pname directory [httpport] [rideport]" >& 2 ; exit 128 ; fi
if [ ! -d $1 ]; then echo "$pname: \"$1\" does not exist" >&2 ; exit 1; fi

folder=`readlink -e $1`
httpport="${2:-8080}"
if [ -z $3 ] ; then rideport="" ; else rideport="-p $3:4502 "; fi

( sleep 3 ; firefox http://localhost:$httpport 2>/dev/null ) &

docker run $rideport -p $httpport:8080 -it -v $folder:/app dyalog/jsonserver:dbg

e=$?
if [ $e != 0 ] ; then echo "$pname: docker returned error code $e" ; exit 1; fi
```

Exercise 3

Wherever you have docker installed:

```
git clone https://github.com/mkromberg/dyalog-docker
```

Depending on your OS, try one of:

```
dyalog-docker/winscripts/dyalog-c /yourapp 4502
dyalog-docker/bashscripts/dyalog-c /yourapp 4502
```

Verify that you can connect RIDE to localhost:4502

Exercise 4

Use the jsonserver-c script to launch your folder as a containerized JSONServer:

```
jsonserver-c /yourapp 8080 4502
```

Verify that you can connect a browser to both ports and call your function from the browser front-end (using the data you noted in Exercise 1)

Exercise 5

- View the docker cheat sheet: https://github.com/wsargent/docker-cheat-sheet

- Experiment with
```
docker ps
docker stop image_name
```

- If you don't)off from RIDE, you will need docker stop to stop your continers

Benefits of Dyalog's Public Containers

Without Public Containers

```
FROM ubuntu:18.04
ADD ./dyalog-unicode_17.0.34604_amd64.deb /
RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS
ADD /myapp/v7/test /myapp
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

Benefits of Dyalog's Public Containers

Without Public Containers

```
FROM ubuntu:18.04
ADD ./dyalog-unicode_17.0.34604_amd64.deb /
RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS
ADD /myapp/v7/test /myapp
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

With Public Containers

```
FROM dyalog/jsonserver:dbg
ADD /myapp/v7/test /myapp
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

Benefits of Dyalog's Public Containers

Without Public Containers

```
FROM ubuntu:18.04
ADD ./dyalog-unicode_17.0.34604_amd64.deb /
RUN dpkg -i /dyalog*.deb
RUN git clone https://github.com/dyalog/JSONServer /JSS
ADD /myapp/v7/test /myapp
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

With Public Containers

```
FROM dyalog/jsonserver:dbg
ADD /myapp/v7/test /myapp
ENV RIDE_INIT="SERVE:*:4502"
ENV CodeLocation=/myapp
CMD dyalog /JSS/JSONServer.dws
```

Or even without a Dockerfile

```
docker run -p 8080:8080 -p 4502:4502 -v /myapp/v7/test:/myapp
           -e RIDE_INIT="SERVE:*:4502" -e CodeLocation=/myapp dyalog/jsonserver
```

Secure JSONServer

```
FROM dyalog/jsonserver:dbg

ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

ENV Secure=1
ENV SSLValidation=64
ENV RootCertDir=/certs/ca
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

ADD test-certs /certs


ADD backend /app

CMD dyalog /JSONServer/Distribution/JSONServer.dws
```

Runs ZodiacService backend as a secure service

Secure JSONServer

APL+JSONServer included →

```
FROM dyalog/jsonserver:dbg

ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

ENV Secure=1
ENV SSLValidation=64
ENV RootCertDir=/certs/ca
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

ADD test-certs /certs


ADD backend /app

CMD dyalog /JSONServer/Distribution/JSONServer.dws
```

Runs ZodiacService backend as a secure service

Secure JSONServer

APL+JSONServer included

Basic JSONServer Settings

```
FROM dyalog/jsonserver:dbg

ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

ENV Secure=1
ENV SSLValidation=64
ENV RootCertDir=/certs/ca
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

ADD test-certs /certs


ADD backend /app

CMD dyalog /JSONServer/Distribution/JSONServer.dws
```

Runs ZodiacService backend as a secure service

Secure JSONServer

APL+JSONServer included

Basic JSONServer Settings

Secure Options

```
FROM dyalog/jsonserver:dbg

ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

ENV Secure=1
ENV SSLValidation=64
ENV RootCertDir=/certs/ca
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

ADD test-certs /certs


ADD backend /app

CMD dyalog /JSONServer/Distribution/JSONServer.dws
```

Runs ZodiacService backend as a secure service

Secure JSONServer

APL+JSONServer included

Basic JSONServer Settings

Secure Options

Add Certificates

```
FROM dyalog/jsonserver:dbg

ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

ENV Secure=1
ENV SSLValidation=64
ENV RootCertDir=/certs/ca
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

ADD test-certs /certs


ADD backend /app

CMD dyalog /JSONServer/Distribution/JSONServer.dws
```

Runs ZodiacService backend as a secure service

Secure JSONServer

APL+JSONServer included

Basic JSONServer Settings

Secure Options

Add Certificates

Application Code

```
FROM dyalog/jsonserver:dbg

ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

ENV Secure=1
ENV SSLValidation=64
ENV RootCertDir=/certs/ca
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

ADD test-certs /certs


ADD backend /app

CMD dyalog /JSONServer/Distribution/JSONServer.dws
```

Runs ZodiacService backend as a secure service

Secure JSONServer

APL+JSONServer included

Basic JSONServer Settings

Secure Options

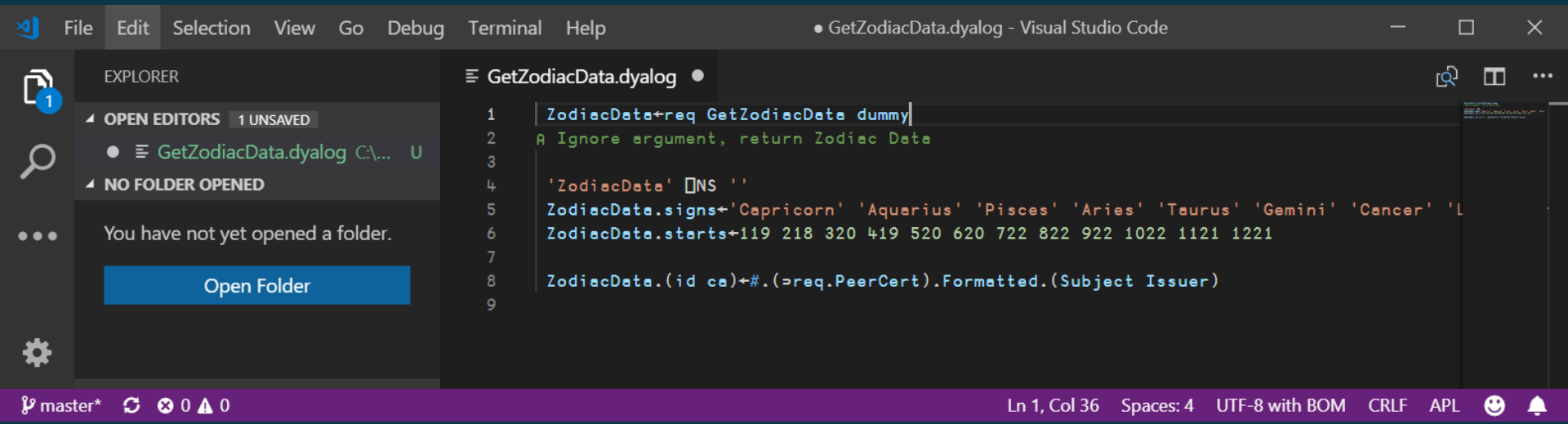
Add Certificates

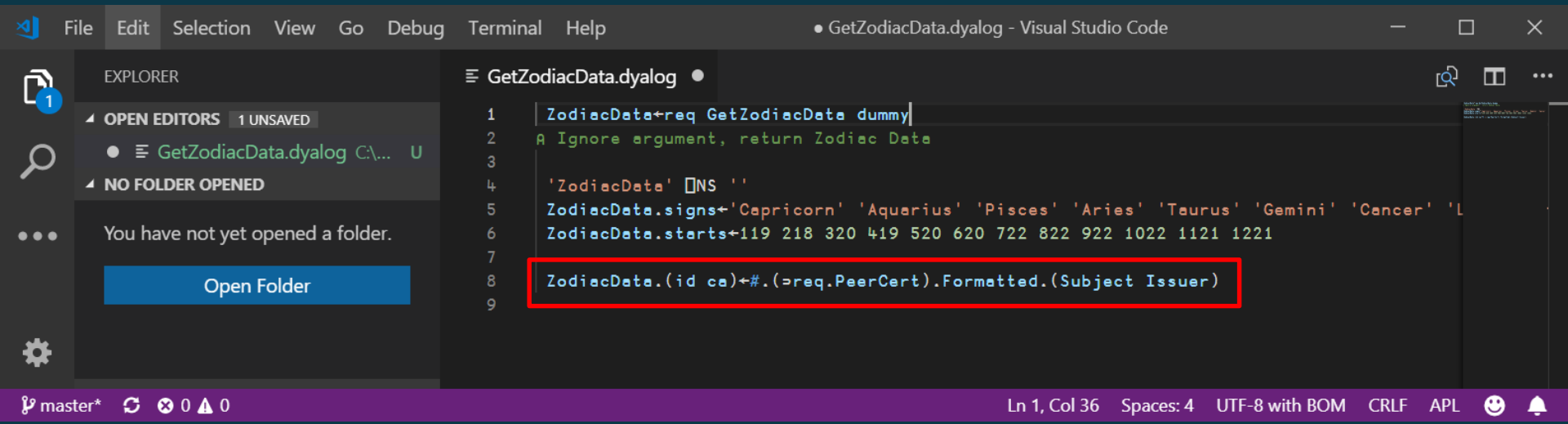
Application Code

Start JSONServer

```
FROM dyalog/jsonserver:dbg

ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

ENV Secure=1
ENV SSLValidation=64
ENV RootCertDir=/certs/ca
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

ADD test-certs /certs


ADD backend /app

CMD dyalog /JSONServer/Distribution/JSONServer.dws
```

Runs ZodiacService backend as a secure service


File Edit Selection View Go Debug Terminal Help
• GetZodiacData.dyalog - Visual Studio Code
EXPLORER
OPEN EDITORS 1 UNSAVED
GetZodiacData.dyalog C:\... U
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
GetZodiacData.dyalog
1 ZodiacData←req GetZodiacData dummy
2 ⍝ Ignore argument, return Zodiac Data
3
4 'ZodiacData' □NS ''
5 ZodiacData.signs←'Capricorn' 'Aquarius' 'Pisces' 'Aries' 'Taurus' 'Gemini' 'Cancer' 'L
6 ZodiacData.starts←119 218 320 419 520 620 722 822 922 1022 1121 1221
7
8 ZodiacData.(id ca)←#.(⊃req.PeerCert).Formatted.(Subject Issuer)
9
master* 0 0
Ln 1, Col 36 Spaces: 4 UTF-8 with BOM CRLF APL


File Edit Selection View Go Debug Terminal Help
• GetZodiacData.dyalog - Visual Studio Code
EXPLORER
OPEN EDITORS 1 UNSAVED
GetZodiacData.dyalog C:\... U
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
GetZodiacData.dyalog
1 ZodiacData←req GetZodiacData dummy
2 ⍝ Ignore argument, return Zodiac Data
3
4 'ZodiacData' ⎕NS ''
5 ZodiacData.signs←'Capricorn' 'Aquarius' 'Pisces' 'Aries' 'Taurus' 'Gemini' 'Cancer' 'L
6 ZodiacData.starts←119 218 320 419 520 620 722 822 922 1022 1121 1221
7
8 ZodiacData.(id ca)←#.(⊃req.PeerCert).Formatted.(Subject Issuer)
9
master* 0 0
Ln 1, Col 36 Spaces: 4 UTF-8 with BOM CRLF APL

Exercise 6

- Add your own function to the "secure" service
- Have it use certificate information in some way
- Build it
- Run it, exposing port 8080 and 4502
- Experiment with

```
docker container ls
docker container rm image_id
```

Exercise 7 / Demo

- Load, modify and run the SSHDemo function.

Ideas for Future Containers

`Runtime and Development/Debug versions of all containers.`

`dyalog/tamstat`

- Runs HTML/JS version of Tamstat "anywhere"
- Looks for data in mapped folder `/data`

`dyalog/isolate`

- Runs an isolate server
- If `/workspace.dws` is found, each isolate will be intialised from it
- `/isolate.config` will set security rules and other options