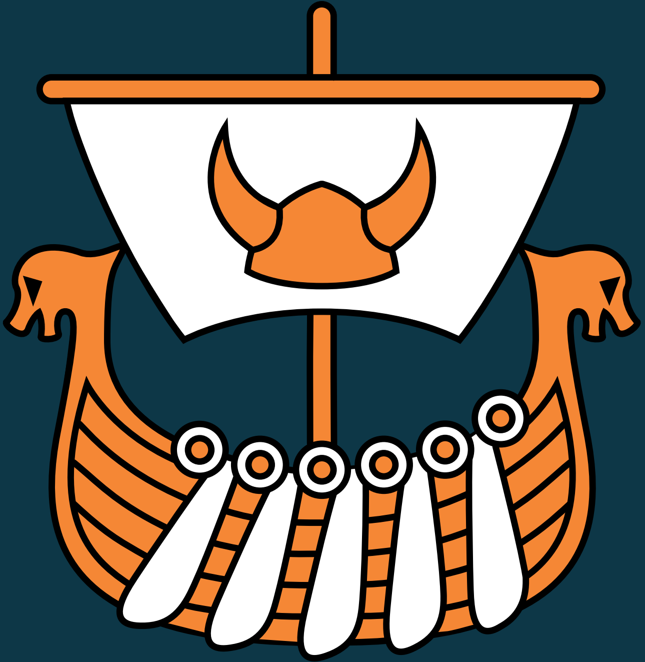


DYALOG

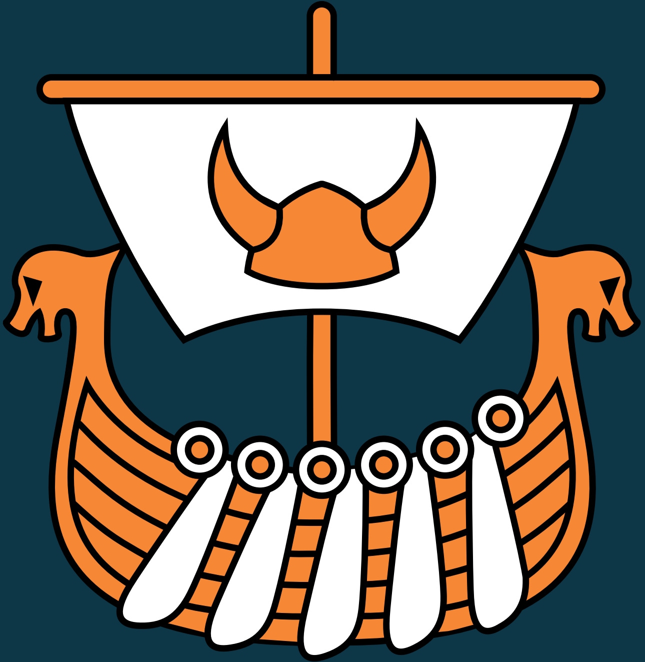
Belfast 2018

Cloud Computing with APL

Morten Kromberg, CXO, Dyalog

DYALOG

Belfast 2018



~~Cloud Computing~~ Docker with APL

Morten Kromberg, CXO, Dyalog

Cloud Computing: Definitions

Cloud Computing: Definitions

Cloud Computing = "Using someone elses computer"

Cloud Computing: Definitions

Cloud Computing = "Using someone elses computer"

- **SAAS** (Software As A Service): You use "**S**oftware" like gmail, dropbox, etc - with no idea of where it is running, or where your data is stored

Cloud Computing: Definitions

Cloud Computing = "Using someone elses computer"

- **SAAS** (Software As A Service): You use "**S**oftware" like gmail, dropbox, etc - with no idea of where it is running, or where your data is stored
- **PAAS**: Someone hosts your application on a "**P**latform" which runs specific development tools (php, mysql, ASP.NET, Wordpress...)

Cloud Computing: Definitions

Cloud Computing = "Using someone elses computer"

- **SAAS** (Software As A Service): You use "**S**oftware" like gmail, dropbox, etc - with no idea of where it is running, or where your data is stored
- **PAAS**: Someone hosts your application on a "**P**latform" which runs specific development tools (php, mysql, ASP.NET, Wordpress...)
- **IAAS** : Someone hosts your Virtual Machine on their "**I**nfrastructure". You can install anything you like.

Cloud Computing: Definitions

Cloud Computing = "Using someone elses computer"

- **SAAS** (Software As A Service): You use "**S**oftware" like gmail, dropbox, etc - with no idea of where it is running, or where your data is stored
- **PAAS**: Someone hosts your application on a "**P**latform" which runs specific development tools (php, mysql, ASP.NET, Wordpress...)
- **IAAS** : Someone hosts your Virtual Machine on their "**I**nfrastructure". You can install anything you like.

This talk is about installing and running Dyalog APL on **IAAS**.

Using IAAS

Using IAAS

- Pick an IAAS provider

Using IAAS

- Pick an IAAS provider
- Upload or Create a Virtual Machine

Using IAAS

- Pick an IAAS provider
- Upload or Create a Virtual Machine

- You save the hassle of
  - Buying [a] big enough computer[s]
  - Maintaining / replacing the hardware
  - Paying for a fast internet connection

Using IAAS – the Hard Part

Using IAAS – the Hard Part

- Picking a provider:
  - Amazon, Microsoft, Google, DigitalOcean, Oracle, RackSpace, Netrepid, IBM/Redhat, GreenCloud, Alibaba, Openstack, ...
  - Can't help you with that

Using IAAS – the Hard Part

- Picking a provider:
  - Amazon, Microsoft, Google, DigitalOcean, Oracle, RackSpace, Netrepid, IBM/Redhat, GreenCloud, Alibaba, Openstack, ...
  - Can't help you with that

- Installing the software that you want to run on the Virtual Machine:
  - APL Interpreter, Web Server or Service Framework, Database System, other tools ...
  - This is where Containers are "Pure Magic"

Containers Solve the Distribution Problem

```
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
```

Containers Solve the Distribution Problem

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
```

Containers Solve the Distribution Problem

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
```

Containers Solve the Distribution Problem

- 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
```

Containers Solve the Distribution Problem

Base Image →

```
FROM ubuntu:18.04
```

Files to Add →

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

Run ***during Build*** →

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

Environment Vars →

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

CMD dyalog /JSS/JSONServer.dws
```

Containers Solve the Distribution Problem

- 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
```

Containers Solve the Distribution Problem

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".

Containers Solve the Distribution Problem

- 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".

Containers Solve the Distribution Problem

Base Image

Files to Add

Run ***during Build***

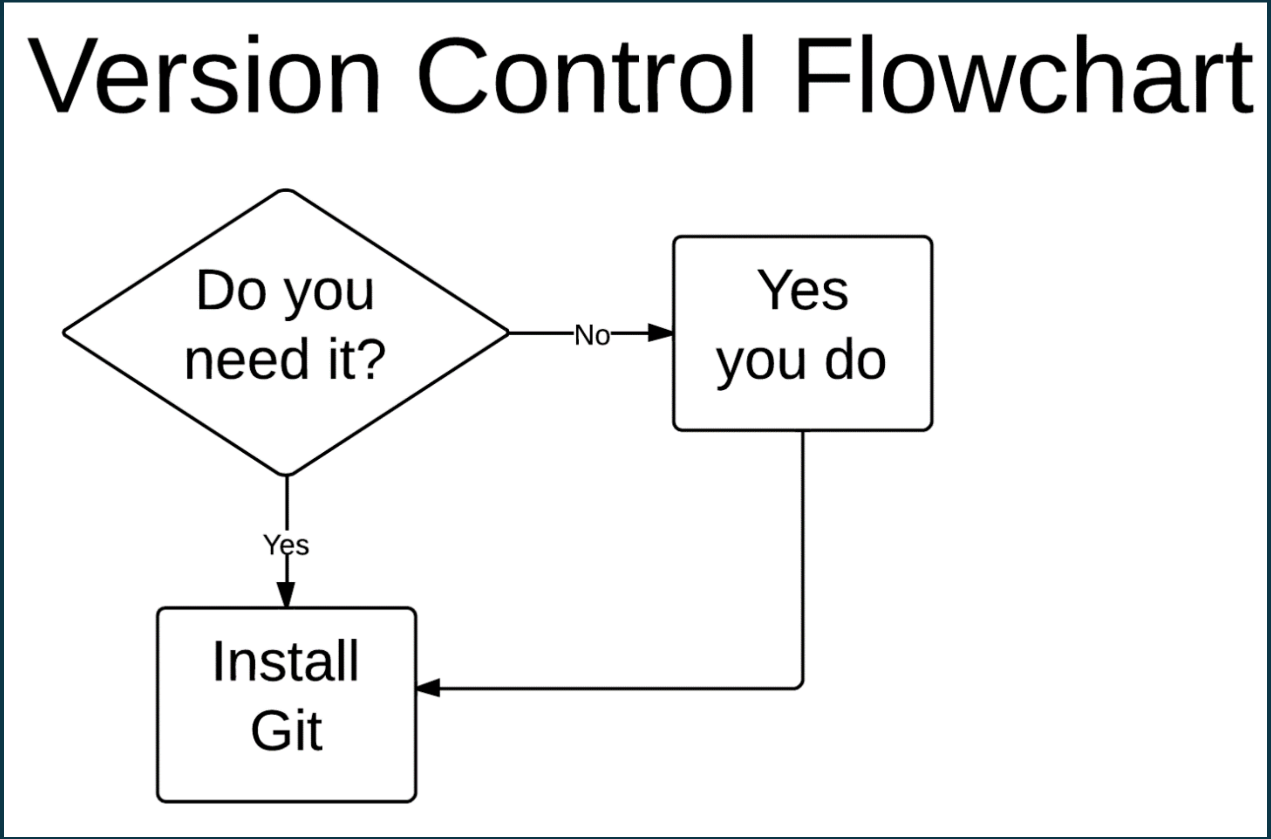
Environment Vars

Run ***at Startup***

Your Code

```
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
```

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


Version Control Flowchart
Do you need it?
No
Yes you do
Yes
Install Git

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
```

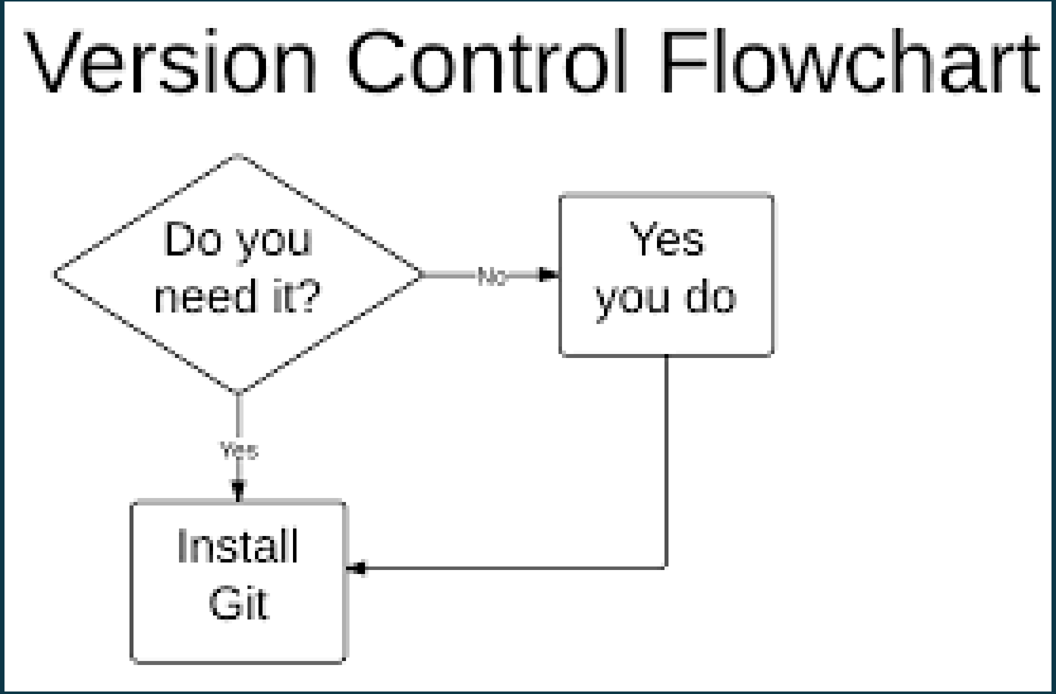
GitHub for source code distribution.
Code can be loaded at Image Build time,
OR when a Container is started.

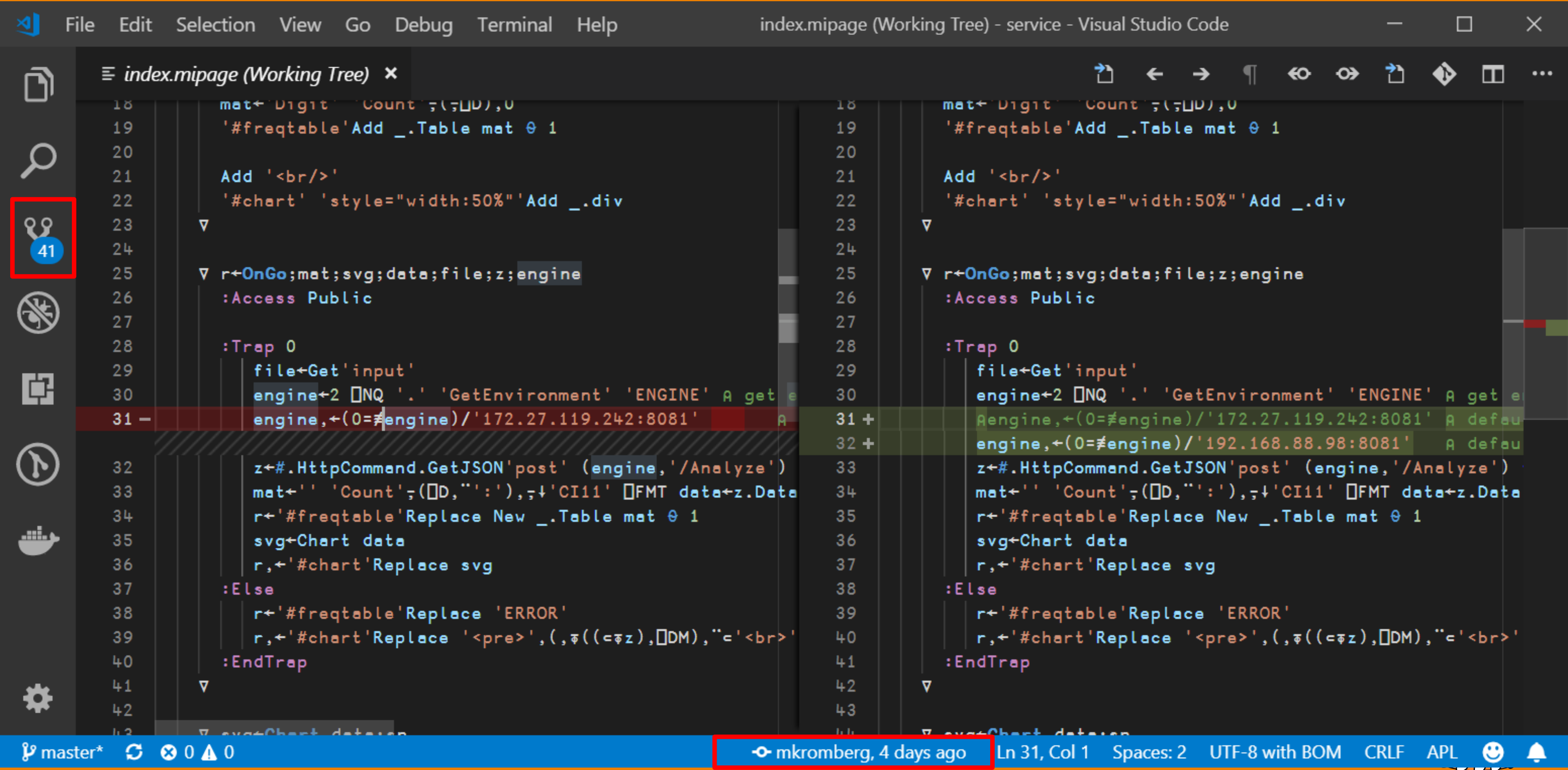
+

DockerHub for container distribution.

=

Simple distribution of applications and tools
to ANY machine – including IAAS VMs.


Version Control Flowchart
Do you need it?
No
Yes you do
Yes
Install Git



File Edit Selection View Go Debug Terminal Help
index.mipage (Working Tree) - service - Visual Studio Code
index.mipage (Working Tree)
41
master*
0 0
mkromberg, 4 days ago
Ln 31, Col 1
Spaces: 2
UTF-8 with BOM
CRLF
APL

```
 mat←'Digit' 'Count'⍪(⍪⎕D),0
 '#freqtable'Add _.Table mat ⊖ 1

 Add '<br/>'
 '#chart' 'style="width:50%"'Add _.div
∇

∇ r←OnGo;mat;svg;data;file;z;engine
 :Access Public

 :Trap 0
   file←Get'input'
   engine←2 ⎕NQ '.' 'GetEnvironment' 'ENGINE' ⍝ get e
-  engine,←(0=≢engine)/'172.27.119.242:8081'      ⍝

   z←#.HttpCommand.GetJSON'post' (engine,'/Analyze')
   mat←'' 'Count'⍪(⎕D,¨':'),⍪↓'CI11' ⎕FMT data←z.Data
   r←'#freqtable'Replace New _.Table mat ⊖ 1
   svg←Chart data
   r,←'#chart'Replace svg
 :Else
   r←'#freqtable'Replace 'ERROR'
   r,←'#chart'Replace '<pre>',(,⍕((⊂⍕z),⎕DM),¨⊂'<br>'
 :EndTrap
∇

∇ svg←Chart data;cp
```

```
 mat←'Digit' 'Count'⍪(⍪⎕D),0
 '#freqtable'Add _.Table mat ⊖ 1

 Add '<br/>'
 '#chart' 'style="width:50%"'Add _.div
∇

∇ r←OnGo;mat;svg;data;file;z;engine
 :Access Public

 :Trap 0
   file←Get'input'
   engine←2 ⎕NQ '.' 'GetEnvironment' 'ENGINE' ⍝ get e
+  ⍝engine,←(0=≢engine)/'172.27.119.242:8081' ⍝ defau
+  engine,←(0=≢engine)/'192.168.88.98:8081'    ⍝ defau
   z←#.HttpCommand.GetJSON'post' (engine,'/Analyze')
   mat←'' 'Count'⍪(⎕D,¨':'),⍪↓'CI11' ⎕FMT data←z.Data
   r←'#freqtable'Replace New _.Table mat ⊖ 1
   svg←Chart data
   r,←'#chart'Replace svg
 :Else
   r←'#freqtable'Replace 'ERROR'
   r,←'#chart'Replace '<pre>',(,⍕((⊂⍕z),⎕DM),¨⊂'<br>'
 :EndTrap
∇

∇ svg←Chart data;cp
```

Containers are STUNNING technology!

Containers are STUNNING technology!

In addition to making distribution very simple:

Containers are STUNNING technology!

In addition to making distribution very simple:

- Containers allow several applications to share the same host but remain isolated from each other

Containers are STUNNING technology!

In addition to making distribution very simple:

- Containers allow several applications to share the same host but remain isolated from each other
- The effect is similar to Virtual Machines but the Operating System kernel is shared

Containers are STUNNING technology!

In addition to making distribution very simple:

- Containers allow several applications to share the same host but remain isolated from each other
- The effect is similar to Virtual Machines but the Operating System kernel is shared
- Containers start and stop Containers in seconds
    - (the Operating System does not need to "Boot Up")

Containers are STUNNING technology!

In addition to making distribution very simple:

- Containers allow several applications to share the same host but remain isolated from each other
- The effect is similar to Virtual Machines but the Operating System kernel is shared
- Containers start and stop Containers in seconds
  - (the Operating System does not need to "Boot Up")
- Containers consume MUCH less resources than VMs

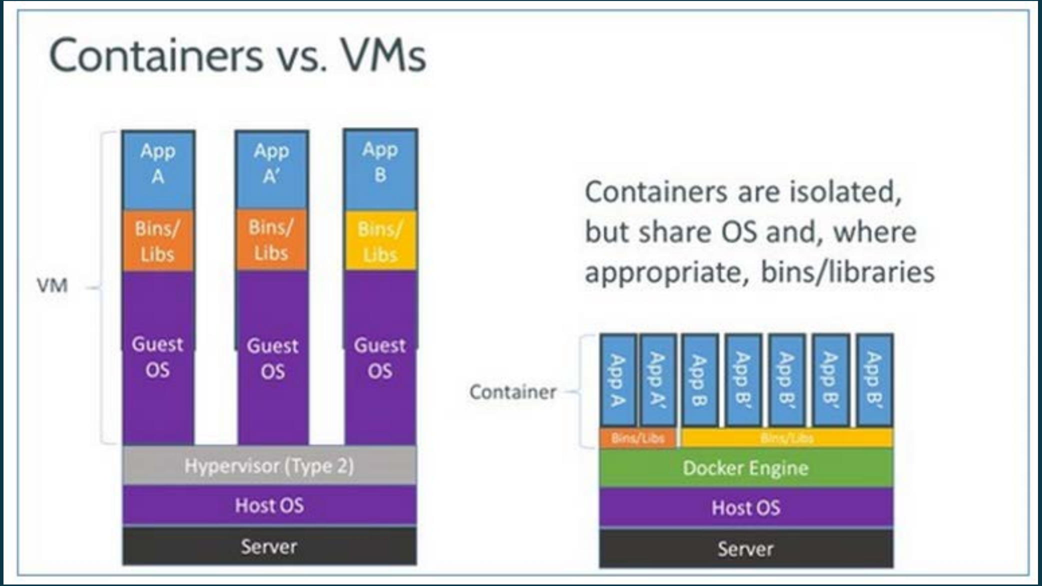
Containers are STUNNING technology!

ZDNET:

Docker is hotter than hot because it makes it possible to get far more apps running on the same old servers and it also makes it very easy to package and ship programs.

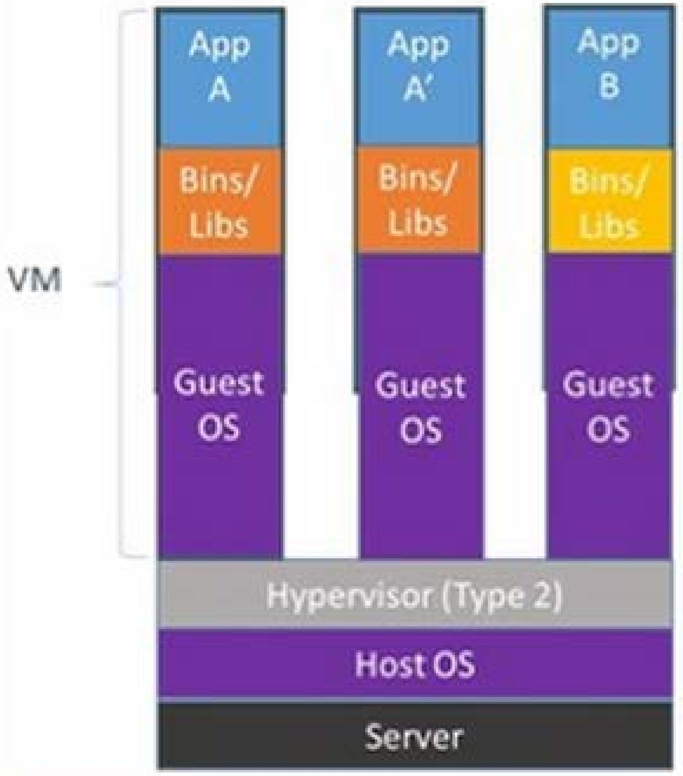
http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

Containers & Docker

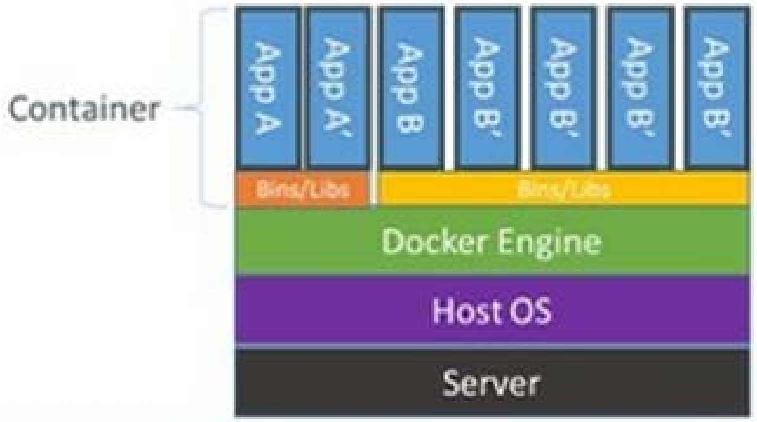


From:
http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

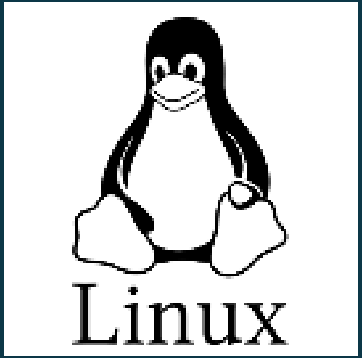
Containers vs. VMs



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

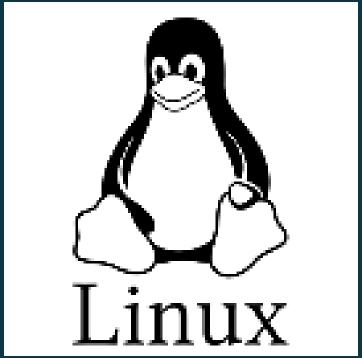


Linux



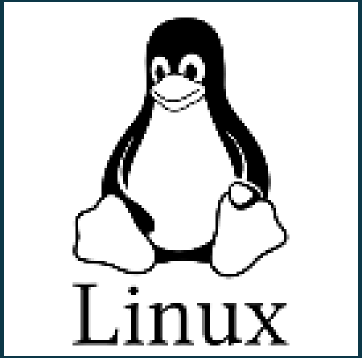
Linux

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



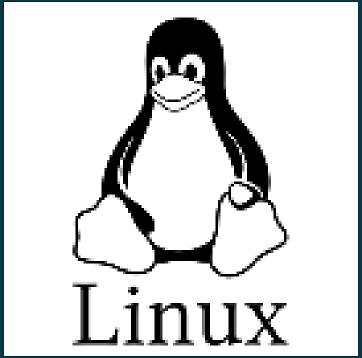
Linux

- 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



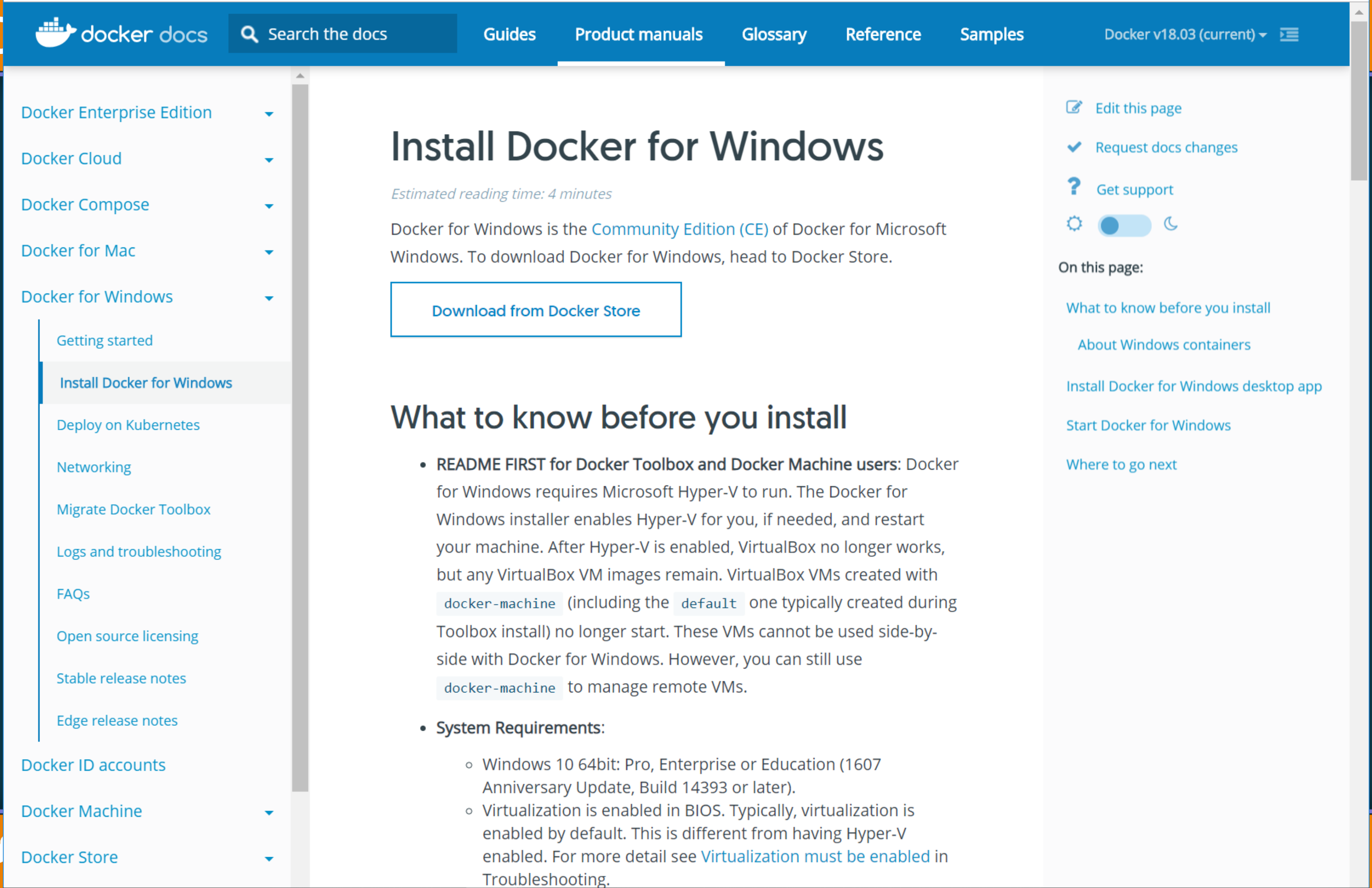
Docker for Windows

- 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

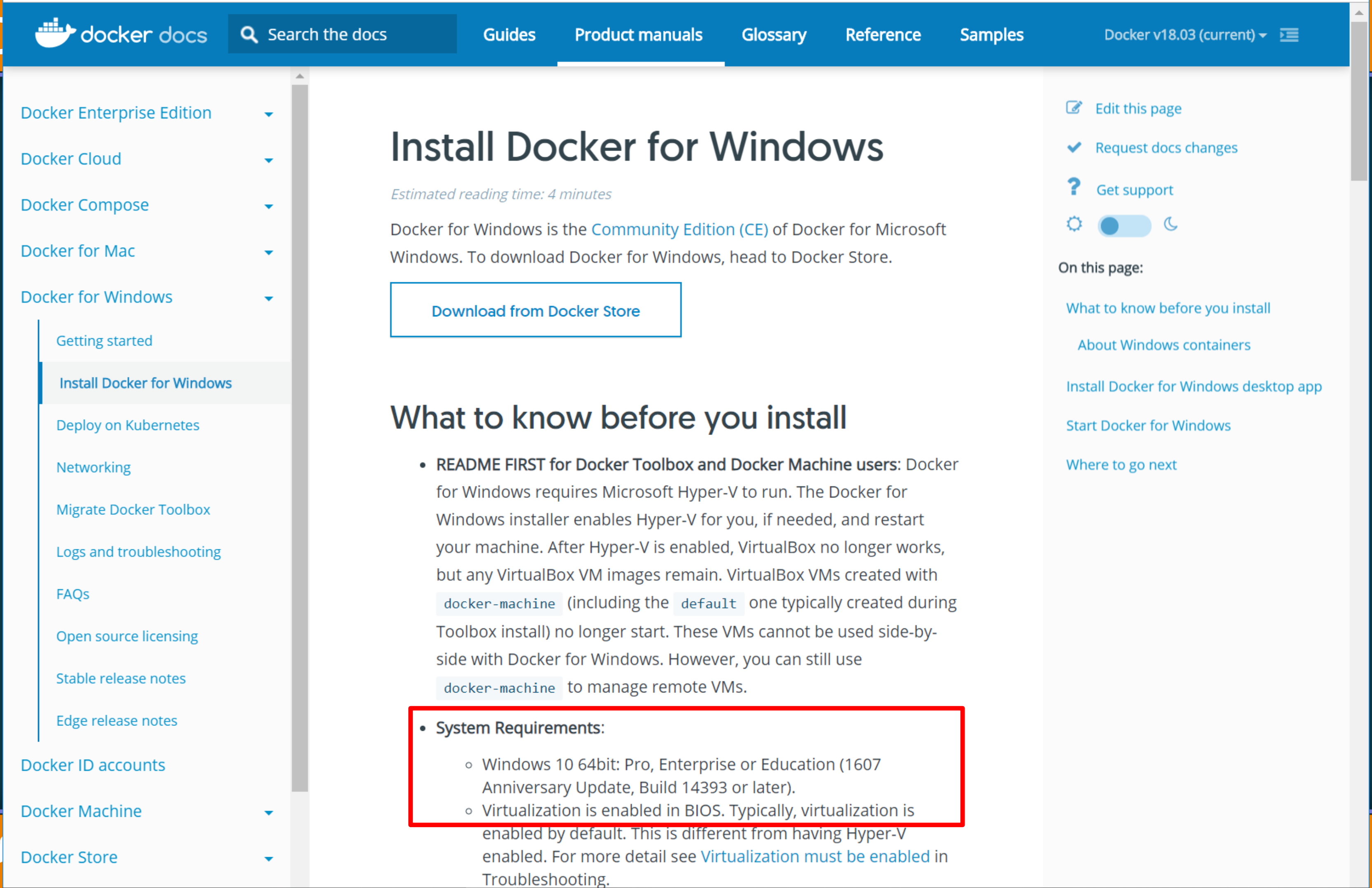
Estimated reading time: 4 minutes

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.



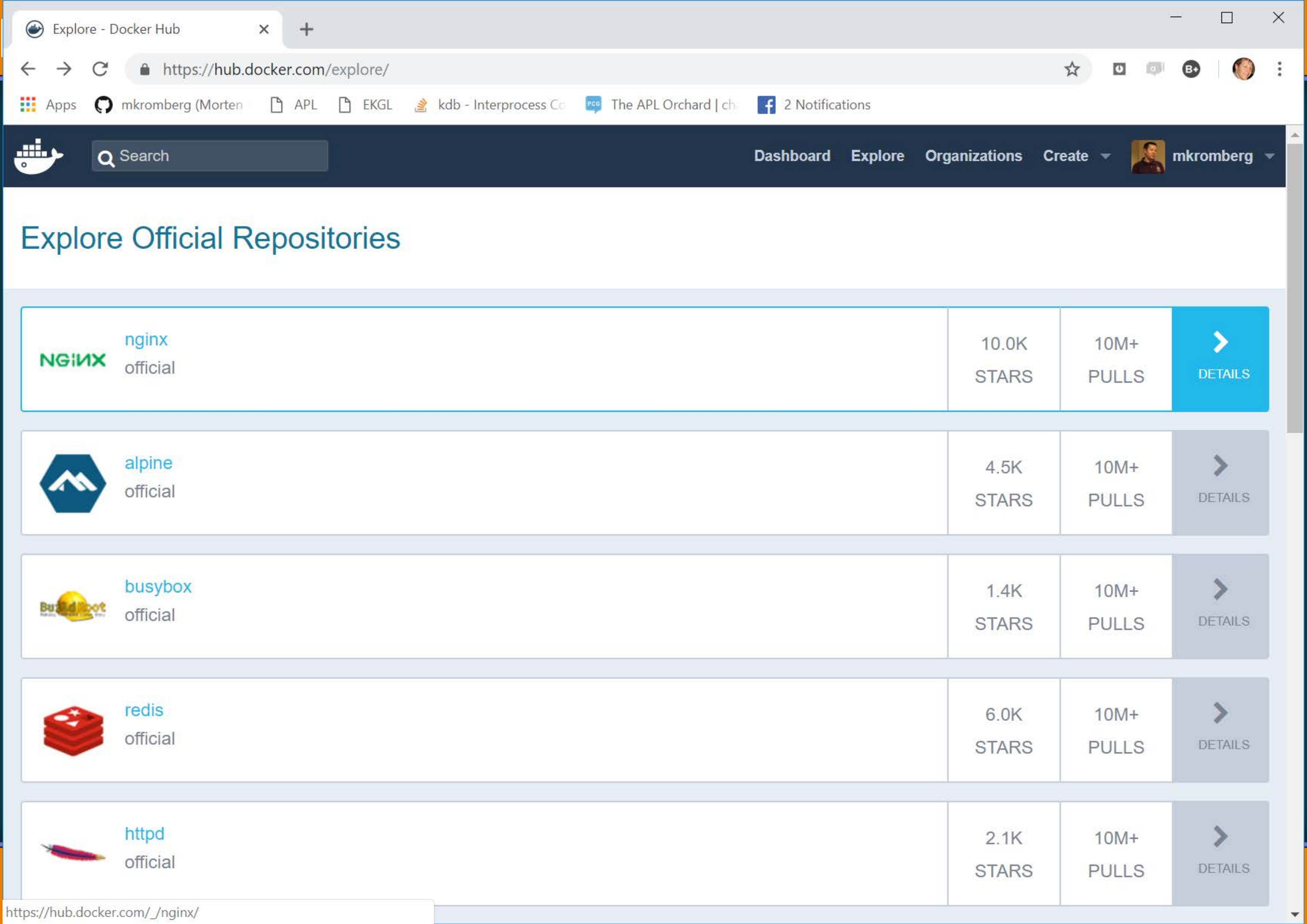
Install Docker for Windows

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.

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.



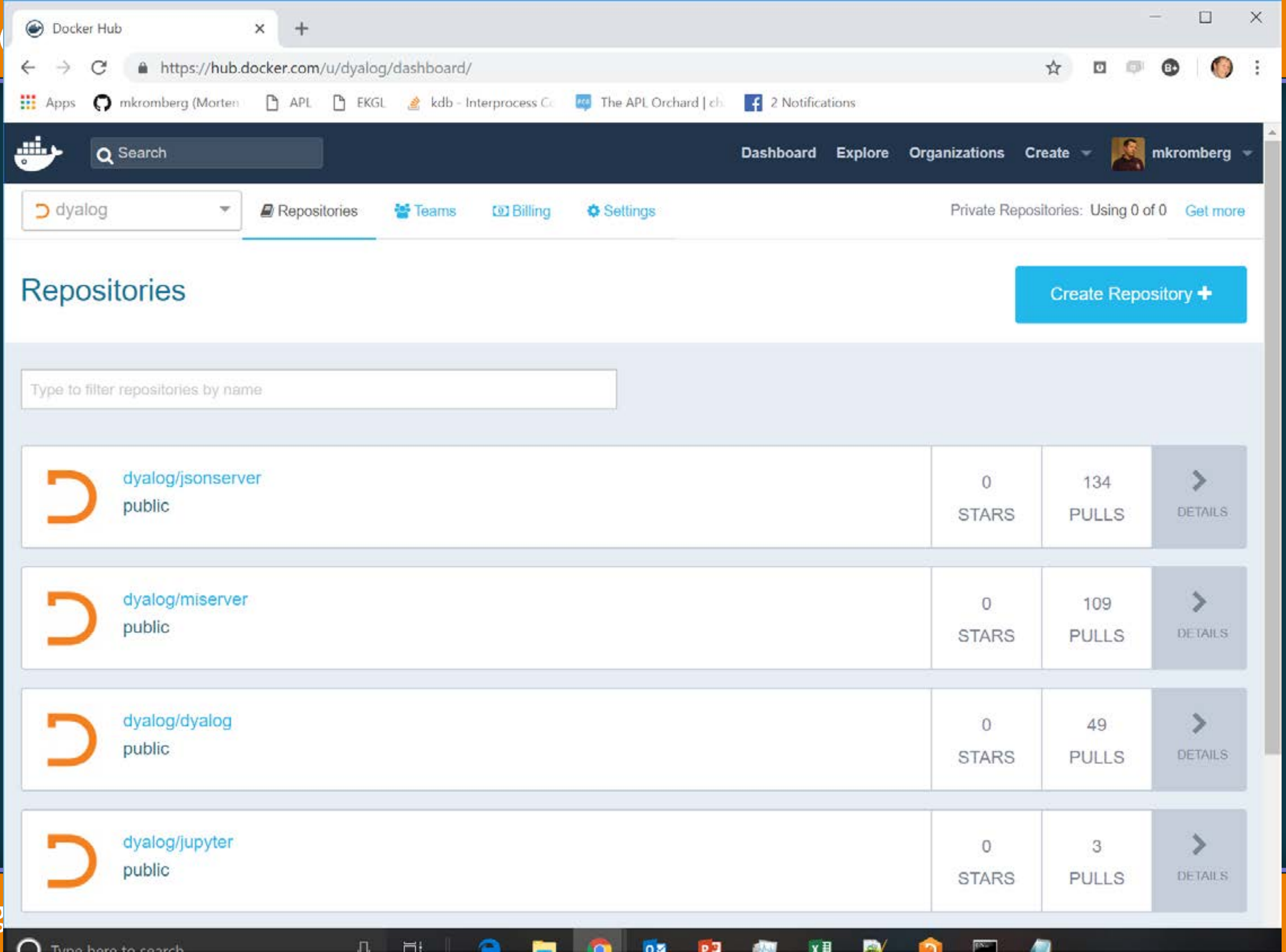
Explore - Docker Hub

https://hub.docker.com/explore/

Explore Official Repositories

nginx official	10.0K STARS	10M+ PULLS	DETAILS
alpine official	4.5K STARS	10M+ PULLS	DETAILS
busybox official	1.4K STARS	10M+ PULLS	DETAILS
redis official	6.0K STARS	10M+ PULLS	DETAILS
httpd official	2.1K STARS	10M+ PULLS	DETAILS

https://hub.docker.com/_/nginx/


Docker Hub

https://hub.docker.com/u/dyalog/dashboard/

dyalog | Repositories | Teams | Billing | Settings

Private Repositories: Using 0 of 0 Get more

Repositories

Create Repository

Repository	Stars	Pulls
dyalog/jsonserver public	0	134
dyalog/miserver public	0	109
dyalog/dyalog public	0	49
dyalog/jupyter public	0	3

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

Benefits of 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 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 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
```

Demo: 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

Demo: 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

Demo: 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

Demo: 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

Demo: 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

Demo: 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
```

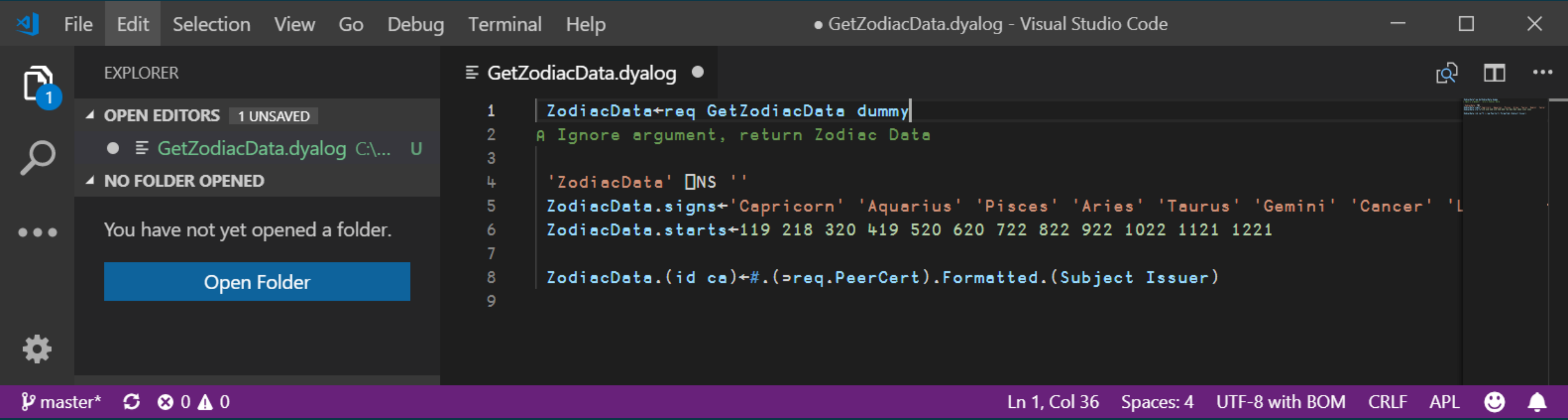
Runs ZodiacService backend as a secure service

Demo: Secure JSONServer

- APL+JSONServer included
- Basic JSONServer Settings
- Secure Options
- Add Certificates
- Application Code
- Start JSONServer

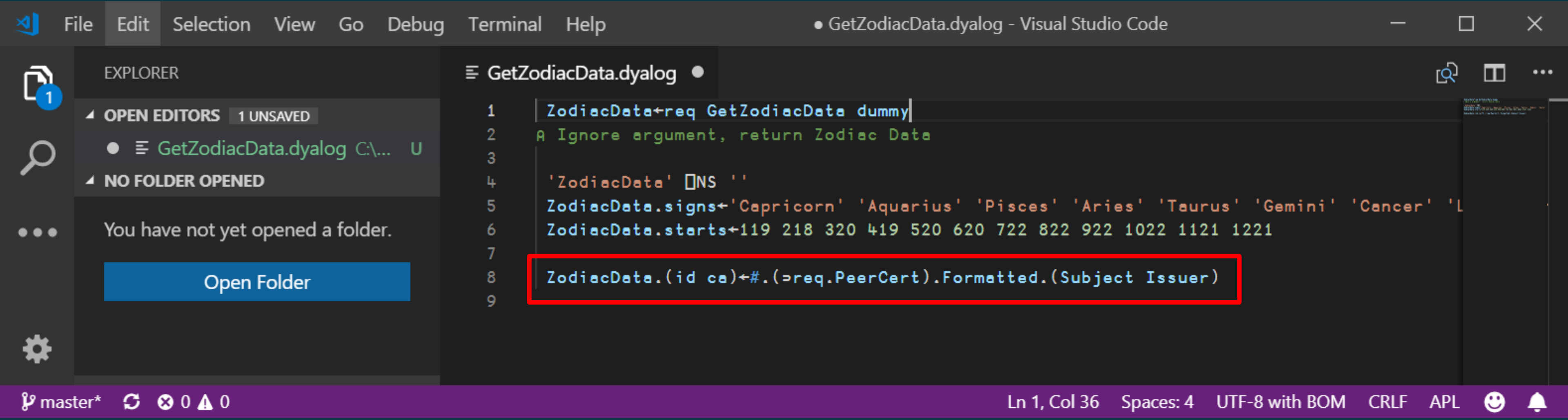
```
FROM dyalog/jsonserver:dbg

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



You have not yet opened a folder.

```
 ZodiacData←req GetZodiacData dummy
⍝ Ignore argument, return Zodiac Data

 'ZodiacData' ⎕NS ''
 ZodiacData.signs←'Capricorn' 'Aquarius' 'Pisces' 'Aries' 'Taurus' 'Gemini' 'Cancer' 'L
 ZodiacData.starts←119 218 320 419 520 620 722 822 922 1022 1121 1221

 ZodiacData.(id ca)←#.(⊃req.PeerCert).Formatted.(Subject Issuer)
```

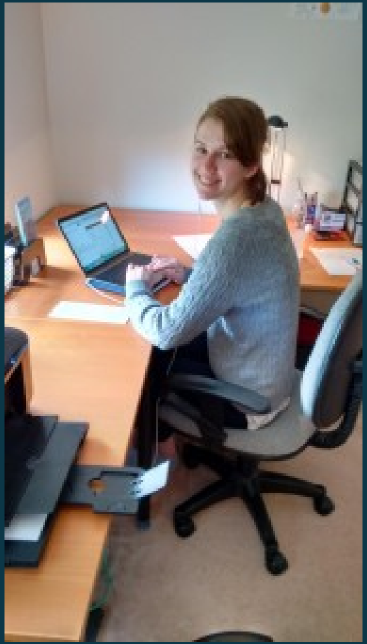

File
Edit
Selection
View
Go
Debug
Terminal
Help
● GetZodiacData.dyalog - Visual Studio Code
EXPLORER
OPEN EDITORS 1 UNSAVED
● GetZodiacData.dyalog C:\... U
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
GetZodiacData.dyalog ●
master* 0 0
Ln 1, Col 36 Spaces: 4 UTF-8 with BOM CRLF APL

```
ZodiacData←req GetZodiacData dummy
⍝ Ignore argument, return Zodiac Data

 'ZodiacData' ⎕NS ''
 ZodiacData.signs←'Capricorn' 'Aquarius' 'Pisces' 'Aries' 'Taurus' 'Gemini' 'Cancer' 'L
 ZodiacData.starts←119 218 320 419 520 620 722 822 922 1022 1121 1221

 ZodiacData.(id ca)←#.(⊃req.PeerCert).Formatted.(Subject Issuer)

```

Demo Time

Demo Time

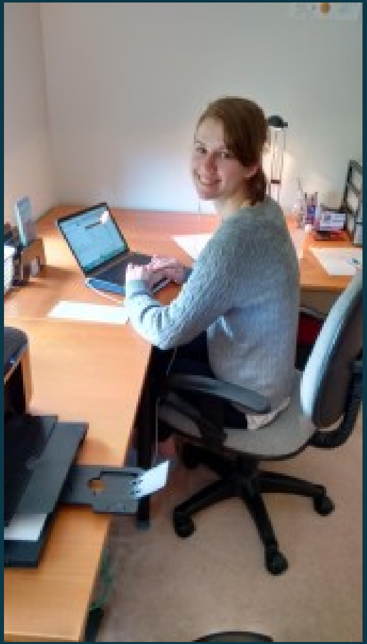
On each machine, we have already:



Demo Time

On each machine, we have already:

- Installed git

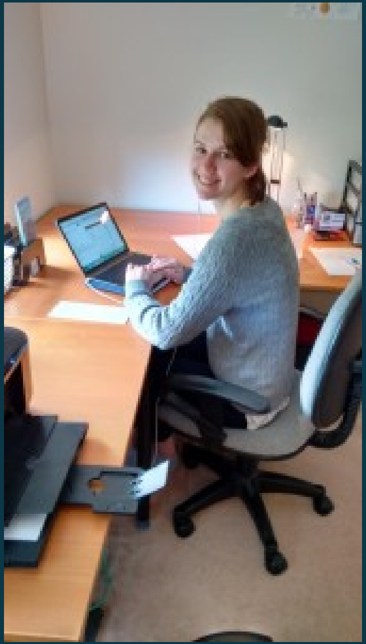


Demo Time

On each machine, we have already:

- Installed git

```
yum install git
```

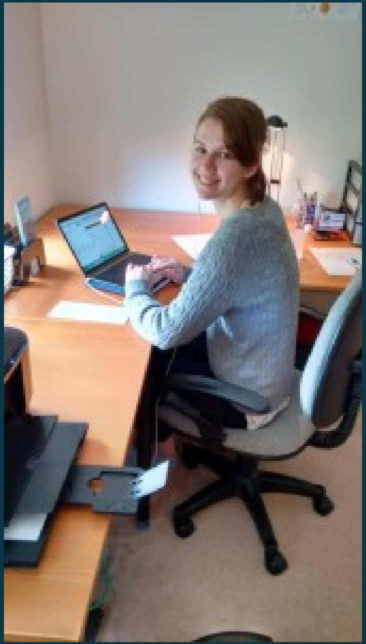


Demo Time

On each machine, we have already:

- Installed git
- Installed docker

```
yum install git
```



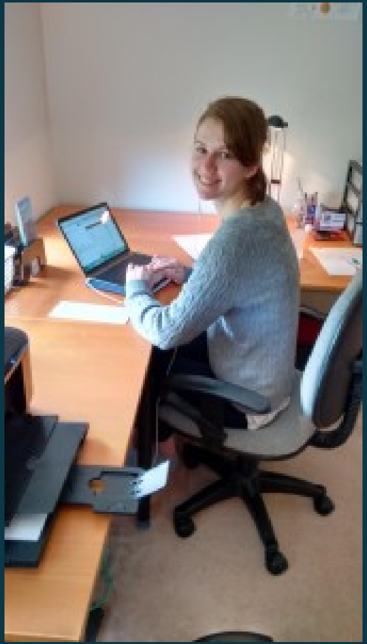
Demo Time

On each machine, we have already:

- Installed git
- Installed docker

```
yum install git
```

```
yum install -y docker
usermod -a -G docker mary
```



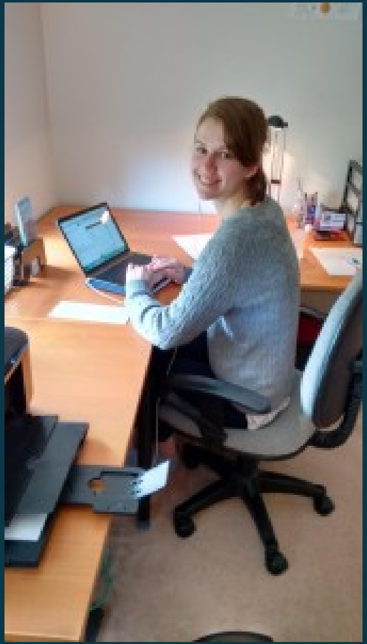
Demo Time

On each machine, we have already:

- Installed git
- Installed docker
- Installed the Docker Util Scripts

```
yum install git
```

```
yum install -y docker
usermod -a -G docker mary
```



Demo Time

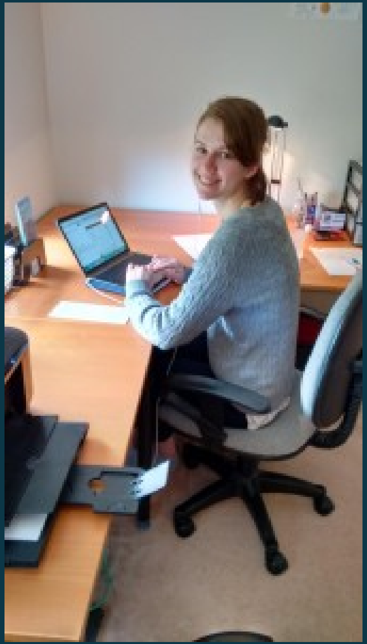
On each machine, we have already:

- Installed git
- Installed docker
- Installed the Docker Util Scripts

```
yum install git
```

```
yum install -y docker
usermod -a -G docker mary
```

- (and put them on the PATH)



Demo Time

On each machine, we have already:

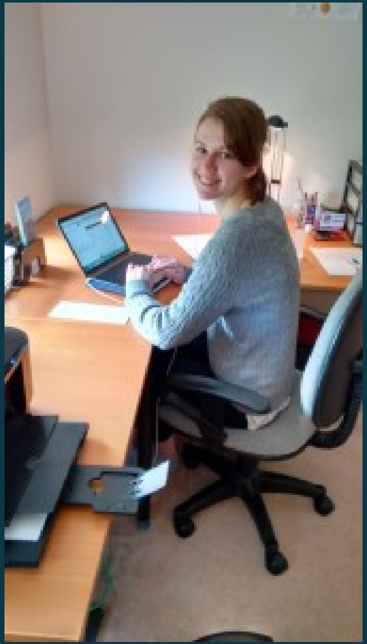
- Installed git
- Installed docker
- Installed the Docker Util Scripts

```
yum install git
```

```
yum install -y docker
usermod -a -G docker mary
```

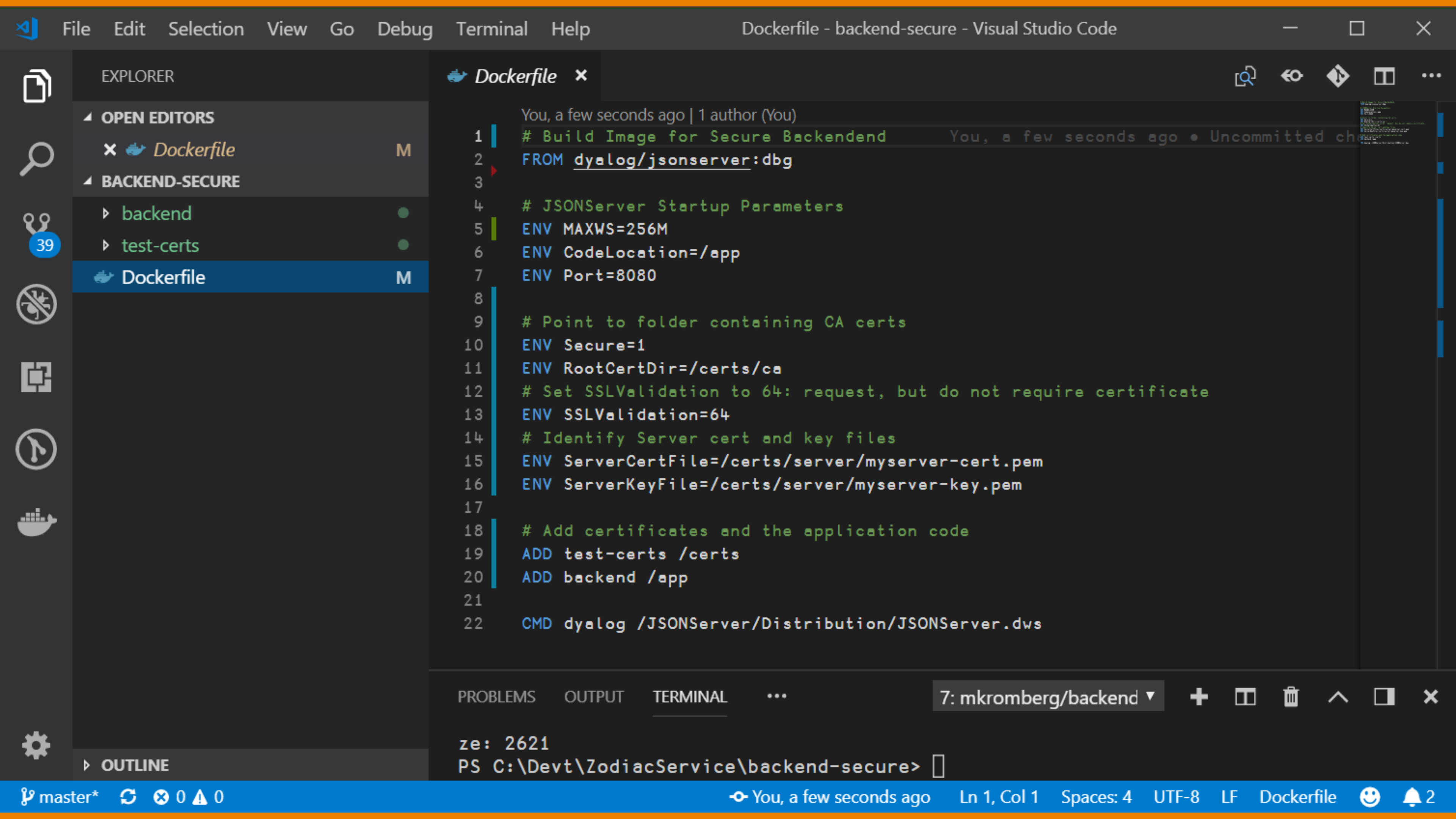
```
git clone https://github.com/mkromberg/docker-utils
```

- (and put them on the PATH)



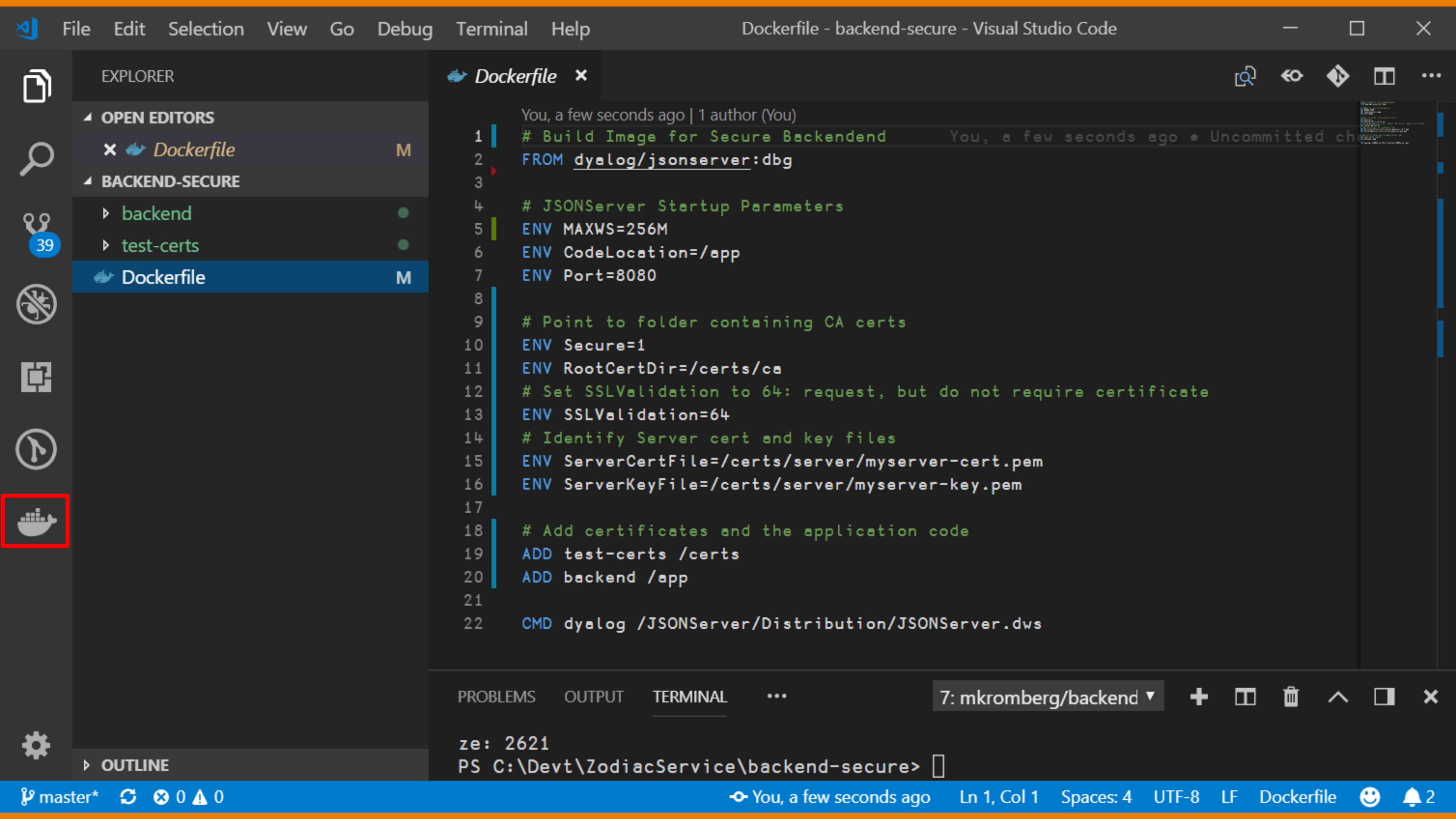
Demo Time

- Build the "secure" service
- Push it to DockerHub
- Login to an AWS EC2 instance
- Start the service
- Test it from a Web Browser


File Edit Selection View Go Debug Terminal Help
Dockerfile - backend-secure - Visual Studio Code
EXPLORER
OPEN EDITORS
Dockerfile M
BACKEND-SECURE
backend
test-certs
Dockerfile M
OUTLINE
39
You, a few seconds ago | 1 author (You)
You, a few seconds ago • Uncommitted che
PROBLEMS OUTPUT TERMINAL
7: mkromberg/backend
master* 0 0 You, a few seconds ago Ln 1, Col 1 Spaces: 4 UTF-8 LF Dockerfile 2

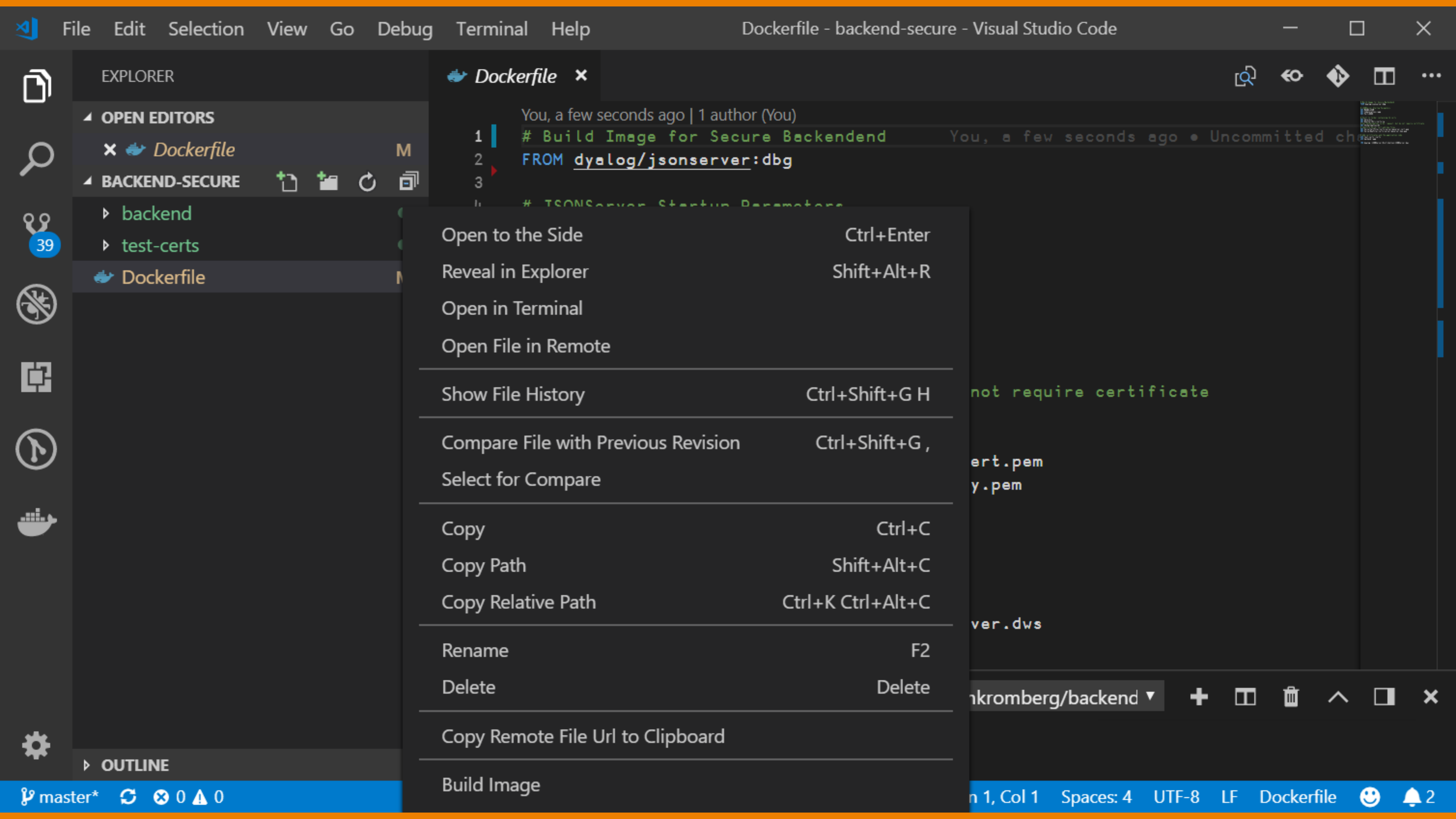
```
# Build Image for Secure Backendend
FROM dyalog/jsonserver:dbg

# JSONServer Startup Parameters
ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

# Point to folder containing CA certs
ENV Secure=1
ENV RootCertDir=/certs/ca
# Set SSLValidation to 64: request, but do not require certificate
ENV SSLValidation=64
# Identify Server cert and key files
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

# Add certificates and the application code
ADD test-certs /certs
ADD backend /app

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

```
ze: 2621
PS C:\Devt\ZodiacService\backend-secure>
```


File Edit Selection View Go Debug Terminal Help
Dockerfile - backend-secure - Visual Studio Code
EXPLORER
OPEN EDITORS
Dockerfile M
BACKEND-SECURE
backend
test-certs
Dockerfile M
OUTLINE
39
You, a few seconds ago | 1 author (You)
PROBLEMS OUTPUT TERMINAL
7: mkromberg/backend
master* 0 0 You, a few seconds ago Ln 1, Col 1 Spaces: 4 UTF-8 LF Dockerfile 2

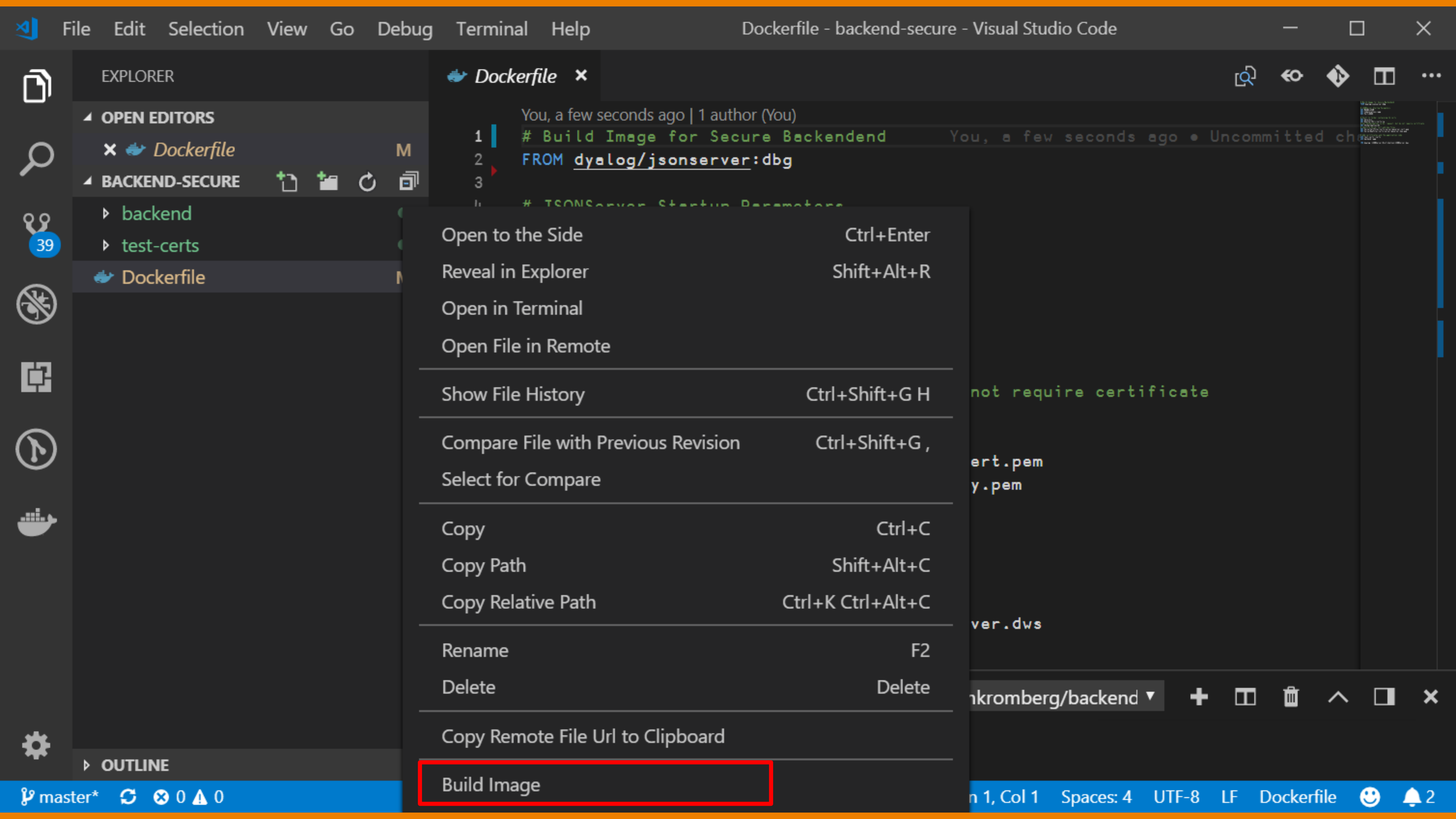
```
# Build Image for Secure Backendend
FROM dyalog/jsonserver:dbg

# JSONServer Startup Parameters
ENV MAXWS=256M
ENV CodeLocation=/app
ENV Port=8080

# Point to folder containing CA certs
ENV Secure=1
ENV RootCertDir=/certs/ca
# Set SSLValidation to 64: request, but do not require certificate
ENV SSLValidation=64
# Identify Server cert and key files
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

# Add certificates and the application code
ADD test-certs /certs
ADD backend /app

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

```
ze: 2621
PS C:\Devt\ZodiacService\backend-secure>
```


File
Edit
Selection
View
Go
Debug
Terminal
Help
Dockerfile - backend-secure - Visual Studio Code
EXPLORER
OPEN EDITORS
Dockerfile M
BACKEND-SECURE
backend
test-certs
Dockerfile
OUTLINE
39
Dockerfile
You, a few seconds ago | 1 author (You)
You, a few seconds ago • Uncommitted ch
Open to the Side Ctrl+Enter
Reveal in Explorer Shift+Alt+R
Open in Terminal
Open File in Remote
Show File History Ctrl+Shift+G H
Compare File with Previous Revision Ctrl+Shift+G ,
Select for Compare
Copy Ctrl+C
Copy Path Shift+Alt+C
Copy Relative Path Ctrl+K Ctrl+Alt+C
Rename F2
Delete Delete
Copy Remote File Url to Clipboard
Build Image
kromberg/backend
master*
0 0
1, Col 1
Spaces: 4
UTF-8
LF
Dockerfile
2

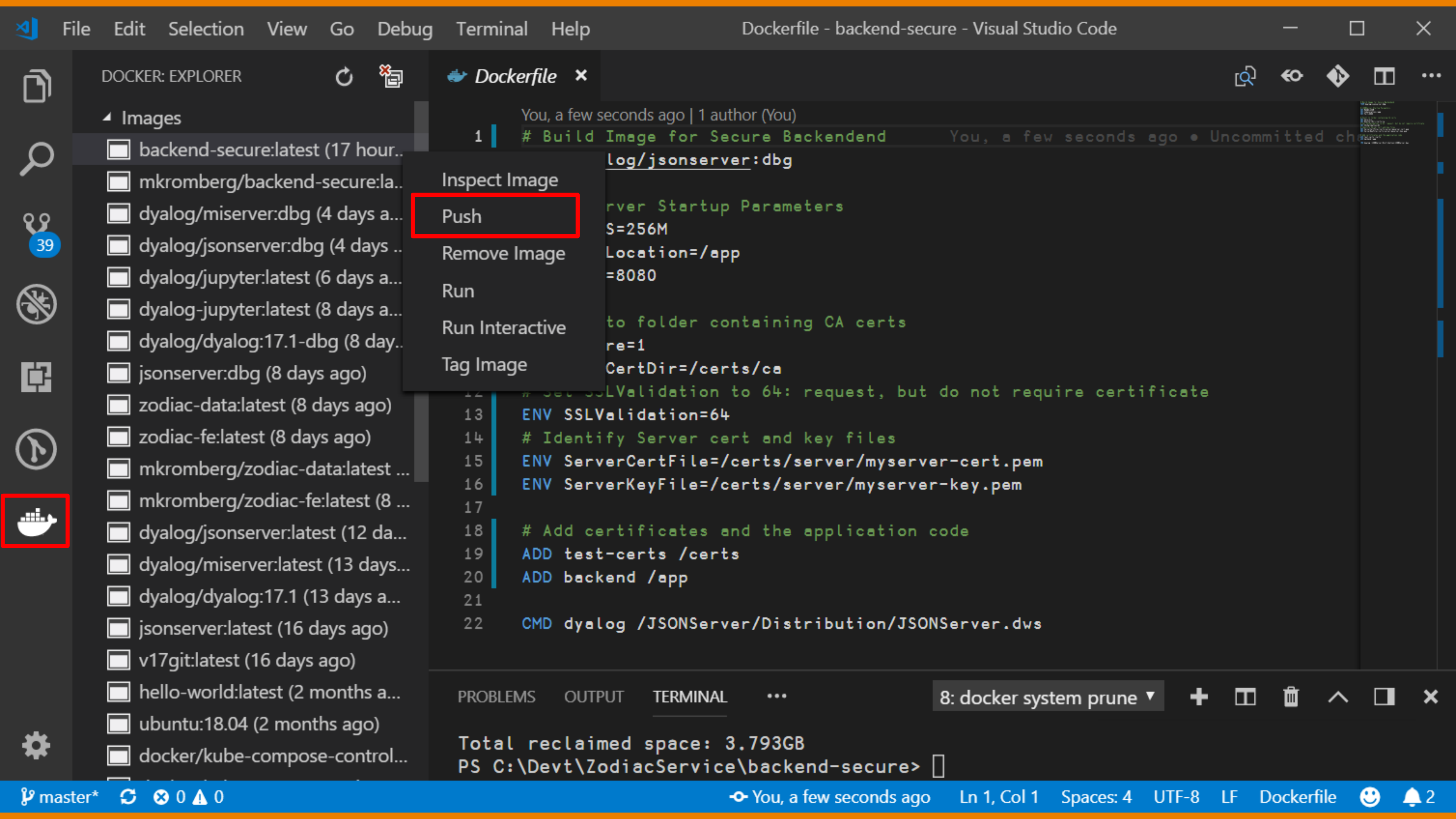
```
# Build Image for Secure Backendend
FROM dyalog/jsonserver:dbg

# JSONServer Startup Parameters
```

```
not require certificate
ert.pem
y.pem
ver.dws
```


File Edit Selection View Go Debug Terminal Help
Dockerfile - backend-secure - Visual Studio Code
EXPLORER
OPEN EDITORS
Dockerfile M
BACKEND-SECURE
backend
test-certs
Dockerfile
OUTLINE
Dockerfile
You, a few seconds ago | 1 author (You)

```
# Build Image for Secure Backendend        You, a few seconds ago • Uncommitted che
FROM dyalog/jsonserver:dbg

# JSONServer Startup Parameters
...not require certificate
...ert.pem
...y.pem
...ver.dws
```

Open to the Side	Ctrl+Enter
Reveal in Explorer	Shift+Alt+R
Open in Terminal	
Open File in Remote	
Show File History	Ctrl+Shift+G H
Compare File with Previous Revision	Ctrl+Shift+G ,
Select for Compare	
Copy	Ctrl+C
Copy Path	Shift+Alt+C
Copy Relative Path	Ctrl+K Ctrl+Alt+C
Rename	F2
Delete	Delete
Copy Remote File Url to Clipboard	
Build Image	


File Edit Selection View Go Debug Terminal Help
Dockerfile - backend-secure - Visual Studio Code
DOCKER: EXPLORER
Images
backend-secure:latest (17 hour...
mkromberg/backend-secure:la...
dyalog/miserver:dbg (4 days a...
dyalog/jsonserver:dbg (4 days ...
dyalog/jupyter:latest (6 days a...
dyalog-jupyter:latest (8 days a...
dyalog/dyalog:17.1-dbg (8 day...
jsonserver:dbg (8 days ago)
zodiac-data:latest (8 days ago)
zodiac-fe:latest (8 days ago)
mkromberg/zodiac-data:latest ...
mkromberg/zodiac-fe:latest (8 ...
dyalog/jsonserver:latest (12 da...
dyalog/miserver:latest (13 days...
dyalog/dyalog:17.1 (13 days a...
jsonserver:latest (16 days ago)
v17git:latest (16 days ago)
hello-world:latest (2 months a...
ubuntu:18.04 (2 months ago)
docker/kube-compose-control...
Inspect Image
Push
Remove Image
Run
Run Interactive
Tag Image
Dockerfile
You, a few seconds ago | 1 author (You)
39
PROBLEMS OUTPUT TERMINAL ...
8: docker system prune
master* 0 0 You, a few seconds ago Ln 1, Col 1 Spaces: 4 UTF-8 LF Dockerfile 2

```
# Build Image for Secure Backendend        You, a few seconds ago • Uncommitted che
log/jsonserver:dbg

rver Startup Parameters
S=256M
Location=/app
=8080

to folder containing CA certs
re=1
CertDir=/certs/ca
  SSLValidation to 64: request, but do not require certificate
ENV SSLValidation=64
# Identify Server cert and key files
ENV ServerCertFile=/certs/server/myserver-cert.pem
ENV ServerKeyFile=/certs/server/myserver-key.pem

# Add certificates and the application code
ADD test-certs /certs
ADD backend /app

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

```
Total reclaimed space: 3.793GB
PS C:\Devt\ZodiacService\backend-secure> 
```

Dyalog Public Scripts

Dyalog Public Scripts

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

Dyalog Public Scripts

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

In parallel folders `bashscripts` and `winscripts`:

Dyalog Public Scripts

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

In parallel folders `bashscripts` and `winscripts`:

`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 Public Scripts

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

In parallel folders `bashscripts` and `winscripts`:

`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 Public Scripts

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

In parallel folders `bashscripts` and `winscripts`:

`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 Public Scripts

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

In parallel folders `bashscripts` and `winscripts`:

`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

Demo Time

Demo Time

Let's build

Demo Time

Let's build
an APL Based

Demo Time

Let's build
an APL Based
Web Site

Demo Time

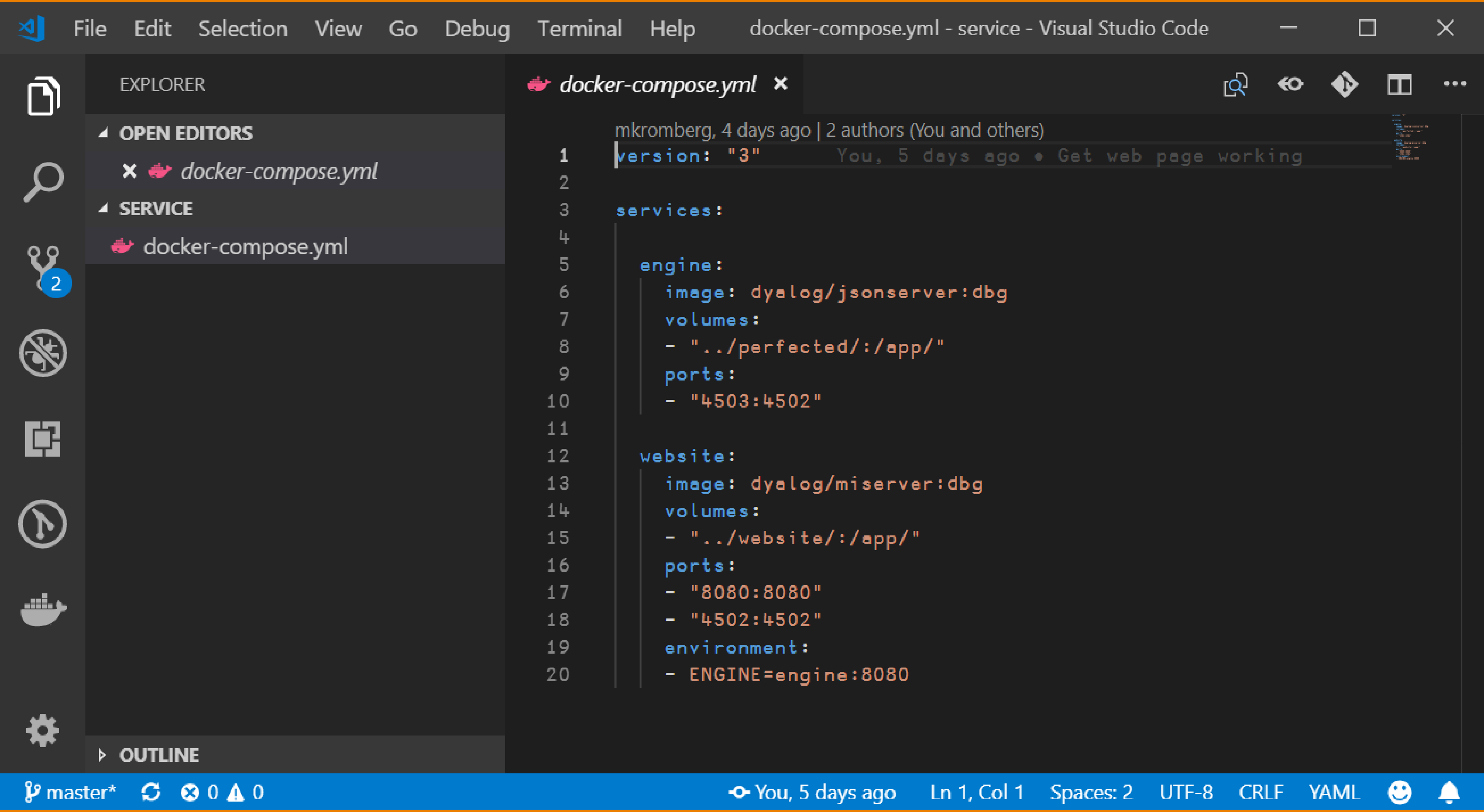
Let's build
an APL Based
Web Site
From Zero

Demo Time

Let's build
an APL Based
Web Site
From Zero

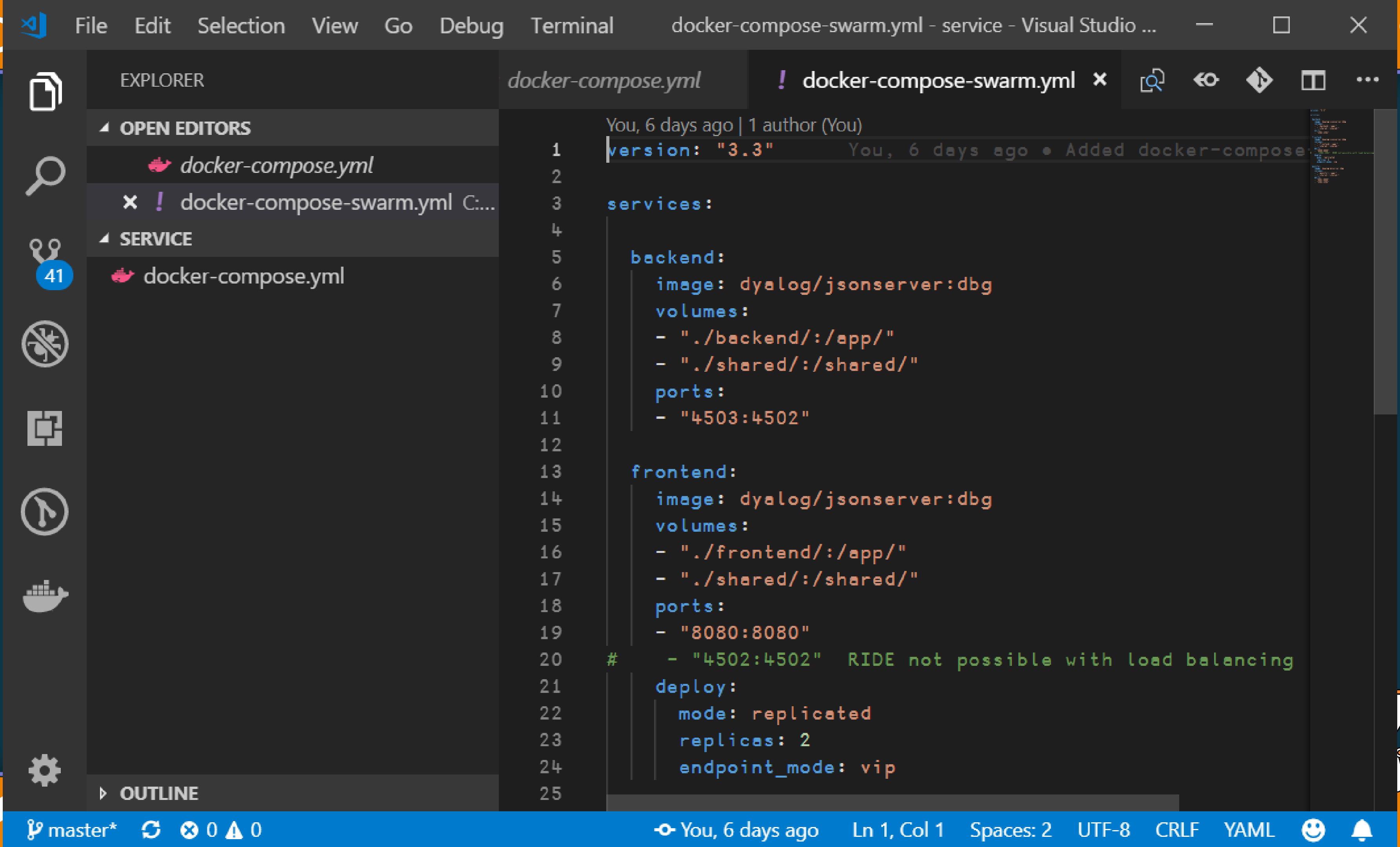
In ABOUT 2 minutes...

docker-compose (multiple services)

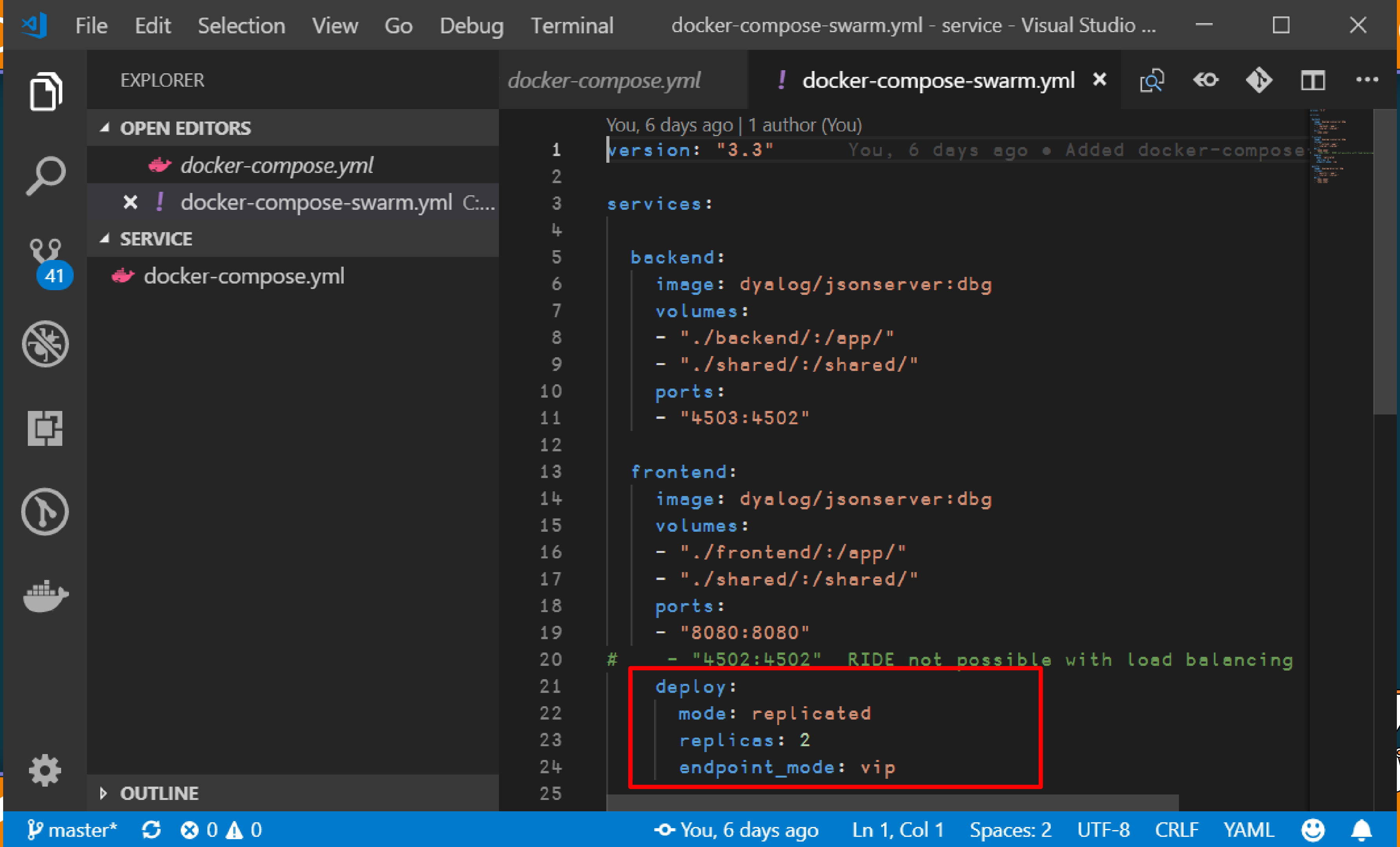

File Edit Selection View Go Debug Terminal Help
docker-compose.yml - service - Visual Studio Code
EXPLORER
OPEN EDITORS
docker-compose.yml
SERVICE
docker-compose.yml
OUTLINE
docker-compose.yml
mkromberg, 4 days ago | 2 authors (You and others)
You, 5 days ago • Get web page working
master*
0 0
You, 5 days ago
Ln 1, Col 1
Spaces: 2
UTF-8
CRLF
YAML

```yaml
version: "3"

services:

  engine:
    image: dyalog/jsonserver:dbg
    volumes:
    - "../perfected/:/app/"
    ports:
    - "4503:4502"

  website:
    image: dyalog/miserver:dbg
    volumes:
    - "../website/:/app/"
    ports:
    - "8080:8080"
    - "4502:4502"
    environment:
    - ENGINE=engine:8080
```

scaling (replicated services)



```yaml
version: "3.3"        You, 6 days ago • Added docker-compose

services:

  backend:
    image: dyalog/jsonserver:dbg
    volumes:
    - "./backend/:/app/"
    - "./shared/:/shared/"
    ports:
    - "4503:4502"

  frontend:
    image: dyalog/jsonserver:dbg
    volumes:
    - "./frontend/:/app/"
    - "./shared/:/shared/"
    ports:
    - "8080:8080"
#     - "4502:4502"  RIDE not possible with load balancing
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: vip
```



```yaml
version: "3.3"      You, 6 days ago • Added docker-compose

services:

  backend:
    image: dyalog/jsonserver:dbg
    volumes:
    - "./backend/:/app/"
    - "./shared/:/shared/"
    ports:
    - "4503:4502"

  frontend:
    image: dyalog/jsonserver:dbg
    volumes:
    - "./frontend/:/app/"
    - "./shared/:/shared/"
    ports:
    - "8080:8080"
#     - "4502:4502"  RIDE not possible with load balancing
    deploy:
      mode: replicated
      replicas: 2
      endpoint_mode: vip
```

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

Conclusion

- It is already easy to deploy APL applications to the cloud (and debug them there)
- Many more public containers and tools to come.
  - Also "Premium Images" that you can run on cloud systems and pay for Dyalog APL "indirectly" throuhg the service provider.
- Follow the Dyalog Webinar series for more news and examples