This note will cover a variety of different fixes for volumes issues and file watchers not updating changes. The example commands have been tested and confirmed to be accurate as of this writing. Each terminal will require a slightly different syntax for the pwd (present working directory) variable.

Windows Home with Docker Toolbox

1. Pass the container the environment variable with fix directly in the docker run command (recommended):

`-e CHOKIDAR_USEPOLLING=true`

or

2. Create a **.env** file in the project root with the following inside of it:

`CHOKIDAR_USESPOLLING=true`

Working example commands for each terminal (be sure to run these commands in the root of your project directory)

Docker Quickstart (recommended)

```
docker run -it -p 3000:3000 -v /app/
node_modules -v ${PWD}:/app -e
CHOKIDAR_USEPOLLING=true IMAGE_ID
```

GitBash

```
winpty docker run -it -p 3000:3000 -v /app/
```

```
node_modules -v "/$(PWD)":/app -e
CHOKIDAR_USEPOLLING=true IMAGE_ID
```

PowerShell

```
docker run -it -p 3000:3000 -v /app/
node_modules -v /c/Users/username/frontend:/
app -e CHOKIDAR_USEPOLLING=true IMAGE_ID
```

Note: Currently, using the full path to your project directory is the only thing that seems to properly mount the volume. Please remember to replace your current working directory with what is shown in the command above.

VirtualBox Folder Sharing

By default, VirtualBox will share the C:\Users directory. If you are using an external or network drive or some other drive on your machine such as D:\ or F:\ it will not be shared and your volumes not be mounted properly.

Please make sure that the drive you are using has been shared by opening VirtualBox and clicking on "Settings", then "Shared Folders".

Windows Defender, Firewalls and Anti-Virus

Any of these services could possibly conflict with the volume mounting. When in doubt, disable each service one by one to see if the volumes begin working correctly.

Windows Pro with Docker Desktop

If you are using Windows and Docker Desktop v2.2 or higher and the latest Create React App v3.4.1, most of the file watching issues have been resolved. If you are using a version of Docker Desktop older than version v2.2 you should update to the latest version.

Working example commands for each terminal (be sure to run these commands in the root of your project directory)

GitBash

```
winpty docker run -it -p 3000:3000 -v /app/node_modules -v "/$(PWD)":/app CONTAINER_ID
```

PowerShell

```
docker run -it -p 3000:3000 -v /app/node_modules -v ${pwd}:/app CONTAINER_ID
```

Docker Desktop File Sharing

By default, your C:\ will be shared with Docker Desktop for volume mounting. If you are using an external or network drive or some other drive on your machine such as D:\ or F:\ it will not be shared and your volumes not be mounted properly.

Please make sure that the drive you are using has been shared by clicking Docker icon in the systray. Click "Settings", then "Resources" and finally "File Sharing".

Windows Defender, Firewalls and Anti-Virus

Any of these services could possibly conflict with the volume mounting. When in doubt, disable each service one by one to see if the volumes begin working correctly.

Recently, a bug was introduced with the latest Create React App version that is causing the React app to exit when starting with Docker Compose.

To Resolve this:

Add stdin_open property to your docker-compose.yml file

```
.     web:
.       stdin_open: true
```

Make sure you rebuild your containers after making this change with `docker-compose down && docker-compose up --build`

If you are using any version of Windows and your React app is not automatically reloading after a code change, you can add this environment variable to your compose file:

```
.   services:
.     web:
```

```
.       environment:
.         - CHOKIDAR_USEPOLLING=true
```

If you are using Windows Home you may have noticed that when adding a test or making a change to the App.test.js the tests are not re-running inside the container.

While this works on macOS (and likely Linux), Jest watchers seem to be completely broken on certain versions of Windows. We are looking into a potential fix or hack to get this working again and will update this note if we find one.

Note - Since the latest release of Create React App and Docker Desktop v2.2 this does not appear to be an issue for Windows Pro or Enterprise users.

Due to a change in how the Jest library works with Create React App, we need to make a small modification:

```
.   script:
.     - docker run USERNAME/docker-react npm run test -- --
    coverage
.
```

instead should be:

```
script:
  - docker run -e CI=true USERNAME/docker-react npm run test
```

Additionally, you may want to set the following property to the top of your .travis.yml file:

```
language: generic
```

There is a slight change that will be required, otherwise you will get an error when Travis attempts to run your code.

The code will now look like this:

```
access_key_id: $AWS_ACCESS_KEY
secret_access_key: $AWS_SECRET_KEY

```

If you still see a failed deployment, try the following fixes:

1) Use an Unnamed Builder

Currently, there seems to be a bug with Elasticbeanstalk and the multi-stage builder step is failing. If you pull the AWS logs, you will see:

"docker pull" requires exactly 1 argument

The quick fix will be to use an unnamed builder, rather than a named builder:

By default, the stages are not named, and you refer to them by their integer number, starting with 0 for the first `FROM` instruction.
https://docs.docker.com/develop/develop-images/multistage-build/#name-your-build-stages

The Dockerfile would now look like this:

```
FROM node:alpine
WORKDIR '/app'
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx
EXPOSE 80
COPY --from=0 /app/build /usr/share/nginx/html
```

2) Upgrade to t2small instance

The `npm install` command frequently times out on the t2.micro instance that we are using. An easy fix is to bump up the instance type that Elastic Beanstalk is using to a t2.small.

Note that a t2.small is outside of the free tier, so you will pay a tiny bit of money (likely less than one dollar if you leave it running for a few hours) for this instance. Don't forget to close it down! Directions for this are a few videos ahead in the lecture titled 'Environment Cleanup'.

3) Refactor COPY line

Try editing the 'COPY' line of your Dockerfile like so:

`COPY package*.json ./`

Sometimes AWS has a tough time with the '.' folder designation and prefers the long form `./`

Tested with the new Platform without issue: Docker running on 64bit Amazon Linux 2/3.0.3

Initial Setup

1. Go to AWS Management Console

2. Search for Elastic Beanstalk in "Find Services"

3. Click the "Create Application" button

4. Enter "docker" for the Application Name

5. Scroll down to "Platform" and select "Docker" from the dropdown list. Leave the rest as defaults.

6. Click "Create Application"

7. You should see a green checkmark after some time.

8. Click the link above the checkmark for your application. This should open the application in your browser and display a Congratulations message.

Change from Micro to Small instance type:

Note that a t2.small is outside of the free tier. t2 micro has been known to timeout and fail during the build process.

1. In the left sidebar under Docker-env click "Configuration"

2. Find "Capacity" and click "Edit"

3. Scroll down to find the "Instance Type" and change from *t2.micro* to *t2.small*

4. Click "Apply"

5. The message might say "No Data" or "Severe" in Health Overview before changing to "Ok"

Add AWS configuration details to .travis.yml file's deploy script

1. Set the *region*. The region code can be found by clicking the region in the toolbar next to your username.

eg: 'us-east-1'

2. *app* should be set to the Application Name (Step #4 in the Initial Setup above)

eg: 'docker'

3. *env* should be set to the lower case of your Beanstalk Environment name.

eg: 'docker-env'

4. Set the *bucket_name*. This can be found by searching for the S3 Storage service. Click the link for the elasticbeanstalk bucket that matches your region code and copy the name.

eg: 'elasticbeanstalk-us-east-1-923445599289'

5. Set the *bucket_path* to 'docker'

6. Set *access_key_id* to $AWS_ACCESS_KEY

7. Set *secret_access_key* to $AWS_SECRET_KEY

Create an IAM User

1. Search for the "IAM Security, Identity & Compliance Service"

2. Click "Create Individual IAM Users" and click "Manage Users"

3. Click "Add User"

4. Enter any name you'd like in the "User Name" field.

eg: docker-react-travis-ci

5. Tick the "Programmatic Access" checkbox

6. Click "Next:Permissions"

7. Click "Attach Existing Policies Directly"

8. Search for "beanstalk"

9. Tick the box next to "AWSElasticBeanstalkFullAccess"

10. Click "Next:Tags"

11. Click "Next:Review"

12. Click "Create user"

13. Copy and / or download the *Access Key ID* and *Secret Access Key* to use in the Travis Variable Setup.

Travis Variable Setup

1. Go to your Travis Dashboard and find the project repository for the application we are working on.

2. On the repository page, click "More Options" and then "Settings"

3. Create an *AWS_ACCESS_KEY* variable and paste your IAM access key from step #13 above.

4. Create an *AWS_SECRET_KEY* variable and paste your IAM secret key from step #13 above.

Deploying App

1. Make a small change to your src/App.js file in the greeting text.

2. In the project root, in your terminal run:

```
. git add.
. git commit -m “testing deployment"
. git push origin master
```

3. Go to your Travis Dashboard and check the status of your build.

4. The status should eventually return with a green checkmark and show "build passing"

5. Go to your AWS Elasticbeanstalk application

6. It should say "Elastic Beanstalk is updating your environment"

7. It should eventually show a green checkmark under "Health". You will now be able to access your application at the external URL provided under the environment name.

```
docker run -it -p 8000:3000 -v /
```

```
Users/anmolrajarora/google\ drive/
work/2020/devops/frontend:/app
<image_id>

docker run -it -p 8000:3000 -v “path
till frontend directory”:/app
<image_id>

docker run -it -p 8000:3000 -v /app/
node_modules -v "$(pwd)":/app
<image_id>
```

Once you have generated the React app you will need to delete the local git repository that Create React App may have automatically initialized.

Inside the newly created client directory, run `rm -r .git`

If you miss this step, the client folder will be considered a submodule and pushed as an empty folder to GitHub.

Due to a recent update in the official Postgres image hosted on the DockerHub we need to make a few changes to our environment variables, package versions, and code.

1) Update the Postgres environment variable

Add the following to your docker-compose.yml file in the **postgres** service:

```
  postgres:
    image: "postgres:latest"
    environment:
      - POSTGRES_PASSWORD=postgres_password

```

2) Pin a specific version of the Node pg library

In your **server/package.json** file:

Change from this:

```
    "pg": "^8.0.3",
```

to this:

```
    "pg": "8.0.3",
```

3) Ensure that we delay the table query until after a connection is made.

In your **server/index.js** file:

Change these lines (likely lines 21 to 25):

```
pgClient.on('error', () => console.log('Lost PG connection'));

pgClient
  .query('CREATE TABLE IF NOT EXISTS values (number INT)')
```

```
.      .catch(err => console.log(err));
.
```

to this:

```
.   pgClient.on('connect', () => {
.     pgClient
.       .query('CREATE TABLE IF NOT EXISTS values (number
    INT)')
.       .catch((err) => console.log(err));
.   });
.
```

After making all of these changes make sure to run:

`docker-compose down  && docker-compose up --build`

Recently, a bug was introduced with the latest Create React App version that is causing the React app to exit when starting with Docker Compose.

To Resolve this:

Add stdin_open property to the client service of your docker-compose.yml file

```
.     client:
.       stdin_open: true
```

Make sure you rebuild your containers after making this change with `docker-compose down && docker-compose up --build`

An edge case where the nginx upstream connection is failing after you run `docker-compose up --build`:

connect() failed (111: Connection refused) while connecting to upstream, client:[DOCKER INTERNAL IP], server:, request: "GET / HTTP/1.1", upstream: [NETWORK IP]

The solution is to add this to your nginx service in the docker-compose.yml file:

```
.     nginx:
.       depends_on:
.         - api
.         - client
```

Its entirely possible that you might run into a bug or two when you start up the set of containers with docker-compose.

Are you able to enter a number into the react app, but it appears to never be calculated?

If that's the case, then you can try adding in environment variables to the 'worker' entry in the docker-compose file, like so:

```
.   worker:
.     environment:
```

```
.       - REDIS_HOST=redis
.       - REDIS_PORT=6379
```

Also, you can add depends_on to the api:

```
.   api:
.     depends_on:
.       - postgres
```

After any changes, make sure you run a `docker-compose down` and then `docker-compose up --build`

Instead of running:

`brew cask install minikube`

We only need to run:

`brew install minikube`

Driver Options

Minikube now supports the use of many different drivers. Hyperkit is the current recommended default for macOS. If you do not have Docker Desktop installed, then you may need to install it using Homebrew:

`brew install hyperkit`

To start minikube with hyperkit:

```
minikube start --driver=hyperkit
```

To set Hyperkit as the default driver:

```
minikube config set driver hyperkit
```

Find the IP address of your cluster:

```
minikube ip
```

Docker Driver - Important

Currently, the docker driver is not supported for use. It currently does not work with any type of ingress.

These instructions are for setting up and installing Minikube and its dependencies for use on Windows Pro with Docker Desktop and HyperV

Install Kubectl

1. Create a new directory that you will move your kubectl binaries into. A good place would be `C:\bin`

2. Download the latest kubectl executable from the link on the Kubernetes doc page:

https://kubernetes.io/docs/tasks/tools/install-kubectl/

#install-kubectl-on-windows

3. Move this downloaded .exe file into the bin directory you created.

4. Use Windows search to type “env” then select “Edit the system environment variables”

5. In the System Properties dialog box, click “Environment Variables”.

6. In System Variables click on the “Path” Variable and then click “Edit”

7. Click “New” and then type `C:\bin`

8. Drag the newly created path so that it is higher in order than Docker's binaries. This is very important and will ensure that you will not have an out of date kubectl client.

9. Click "OK"

10. Restart your terminal and test by typing `kubectl` into it. You should get the basic commands and help menu printed back to your screen. If this doesn't work try restarting your machine.

11. Run `kubectl version --client` to verify that you are using the newest version and not the out of date v1.10 version

Install Minikube

1. Download the Windows installer here:

https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe

2. Double click the .exe file that was downloaded and run the installer. All default selections are appropriate.

3. Open up your terminal and test the installation by typing `minikube`. You should get the basic commands and help menu printed back to your screen. If this doesn't work try restarting your machine.

Starting Up Minikube

Since by default Minikube expects VirtualBox to be used, we need to tell it to use the hyperv driver instead, as well as the Virtual Switch we created earlier.

Start up a terminal as an Administrator. Then, in your terminal run:

```
minikube start --vm-driver hyperv
```

Important note, all minikube commands must be run in the context of an elevated Administrator.

These instructions are for setting up and installing Minikube and its dependencies for use on Windows Home editions that are using Docker Toolbox.

Install Kubectl

1. Create a new directory that you will move your kubectl binaries into. A good place would be `C:\Users\YOURUSERNAME\bin` since these path variables will also be made available to the "Docker Quick Start" terminal.

2. Download the latest kubectl executable from the link on the Kubernetes doc page:

https://kubernetes.io/docs/tasks/tools/install-kubectl/#install-kubectl-on-windows

3. Move this downloaded .exe file into the bin directory you created.

4. Use Windows search to type “env” or “edit” then select “Edit the system environment variables”

5. In the System Properties dialog box, click “Environment Variables”.

6. In User Variables click on the “Path” Variable and then click “Edit”

7. Click “New” and then type `C:\Users\YOURUSERNAME\bin`

8. Click "OK"

9. Restart your terminal and test by typing kubectl into it. You should get the basic commands and help menu printed back to your screen. If this doesn't work try restarting your machine.

Install Minikube

VirtualBox should already be installed from setting up Docker Toolbox, so we wont need to install this again.

To install Minikube we can use the standalone installer, which is available by clicking this link:

https://github.com/kubernetes/minikube/releases/latest/download/minikube-installer.exe

After the installer finishes, restart your terminal and test your installation:

`minikube status`

Running `minikube start` will provision the VirtualBox machines and startup your Kubernetes services.

These instructions should be valid for Debian / Ubuntu / Mint Linux distributions. Your experience may vary if using an RHEL / Arch / Other distribution or non desktop distro like Ubuntu server, or lightweight distros which may omit many expected tools.

Install VirtualBox:

Find your Linux distribution and download the .deb package, using a graphical installer here should be sufficient. If you use a package manager like apt to install from your terminal, you will likely get a fairly out of date version.

https://www.virtualbox.org/wiki/Linux_Downloads

After installing, check your installation to make sure it worked:

```
VBoxManage —version
```

As an alternative you can use (or maybe you have to use) KVM instead of VirtualBox. Here are some great instructions that can be found in this post (Thanks to Nick L. for sharing):

https://computingforgeeks.com/install-kvm-centos-rhel-ubuntu-debian-sles-arch/

Install Kubectl

In your terminal run the following:

```
curl -LO https://storage.googleapis.com/
kubernetes-release/release/$(curl -s https://
storage.googleapis.com/kubernetes-release/
release/stable.txt)/bin/linux/amd64/kubectl
```

```
chmod +x ./kubectl
```

```
sudo mv ./kubectl /usr/local/bin/kubectl
```

Check your Installation:

```
kubectl version
```

See also official docs:
https://kubernetes.io/docs/tasks/tools/install-kubectl/
#install-kubectl-on-linux

Install Minikube

In your terminal run the following:

```
curl -Lo minikube https://
storage.googleapis.com/minikube/releases/
latest/minikube-linux-amd64 && chmod +x
```

`minikube`

`sudo install minikube /usr/local/bin`

Check your installation:

`minikube version`

Start Minikube:

`minikube start`

See also official docs:

https://kubernetes.io/docs/tasks/tools/install-minikube/

These instructions are for using Docker Desktop's builtin Kubernetes instead of minikube which is discussed in the lectures.

Windows Pro / Enterprise

1. Right click the Docker icon in the Windows Toolbar

2. Click Settings

3. Click "Kubernetes" in the dialog box sidebar (If you don't see Kubernetes listed in the sidebar you may have Windows Containers enabled instead of Linux Containers - you can toggle this setting by right clicking the Docker toolbar icon and clicking "Switch to Linux Containers")

4. Check the “Enable Kubernetes” box

5. Click "Apply"

6. Click Install to allow the cluster installation (This may take a while). If on Windows, you may get a dialog box warning from Windows Defender about Docker's vpnkit. Check the boxes to allow it to communicate on your network and click "Allow Access"

macOS

1. Click the Docker icon in the top macOS toolbar

2. Click Preferences

3. Click "Kubernetes" in the dialog box menu

4. Check the “Enable Kubernetes” box

5. Click "Apply"

6. Click Install to allow the cluster installation (This may take a while).

Usage

Going forward, any minikube commands run in the lecture videos can be ignored. Also, instead of the IP address used in the video lectures when using minikube, we use localhost.

For example, in the first project where we deploy our simple React app, using minikube we would visit:

192.168.99.101:31515

Instead, when using Docker Desktop's Kubernetes, we would visit: **localhost:31515**

Due to a change in how the Jest library works with Create React App, we need to make a small modification:

```
.   script:
.     - docker run USERNAME/react-test npm test -- --coverage
```
instead should be:

```
  script:
    - docker run -e CI=true USERNAME/react-test npm test
```

You can read up on the CI=true variable here:

https://facebook.github.io/create-react-app/docs/running-tests#linux-macos-bash

and enviornment variables in Docker here:

https://docs.docker.com/engine/reference/run/#env-environment-variables

Additionally, you may want to set the following property if your travis build fails with “rakefile not found” by adding to the top of your .travis.yml file:

```
  language: generic

```

You may get a blank page with an error in your console when you attempt to run the pod or deployment:

react-dom.production.min.js:209 TypeError: this.state.seenIndexes.map is not a function

This is because we added the following line to our **client/nginx/default.conf** file:

try_files $uri $uri/ /index.html;

This line was added to resolve some React Router issues our client app was having. However, it will break this demo because we have no Nginx container or Ingress service in place.

The best way to resolve this is to use Stephen's Client image in the pod and deployment for these demos, instead of your own:

```
image: stephengrider/multi-client
```

To be clear, this issue will not affect the multi-container Kubernetes project we will be building in Section 14. This is just a quick demo and we will not be reusing these files.