Using Docker to maintain a development environment

harleydk, 4 Jun 2018

.Would you contemplate using a containerized environment, that you might tear down at any moment, only to build it right back up again, indistinguishable from how it was before?I have recently begun to do just that. And it has greatly enhanced my workflow, and productivity.

Would you contemplate using a containerized environment, that you might tear down at any moment, only to build it right back up again, indistinguishable from how it was before?

I have recently begun to do just that. And it has greatly enhanced my workflow, and productivity. And I'll try and describe the how's and why's below.

When we think of Docker, we usually think of reasonably light-weight container instances, that we may assemble and congregate to form various applications. Each container yields usually a particular and highly individual service, for example a REST API or a database instance or a memory-cache or what have you, and we can compile these into low coupled solutions. And there's the potential of scaling, of the low overhead and cost that comes with only keeping the containers running for the duration of their workload. That's the norm. But really one of the fundamental aspects of using Docker is in how we can create a image of anything we wish, and the brevity - all things being equal - in how Docker image recipe makes it simple to conjure up the exact same environment to the detailed spec that is the Docker-file. Right now I'm running a number of only slightly different Ubuntu-based development environments in docker containers, that I access remotely. It's based on an Ubuntu 17.10, it's got Visual Studio Code installed, a few extensions to that, a browser, git. Postman, that I recommend for API testing.

Why would one wish to do that, 'dockerize' a full-fledged Ubuntu, weighing in at some odd gigabytes? I can think of several promoting arguments:

- The transparency of the environment. Setting up a dev machine can be tedious, and after a while it inevitably becomes cluttered with updates and remains of old installs and such. A Docker-initiated container won't suffer from this; any changes are made in the Docker-file and we'll get into the habit of tearing the machine down, and building it anew, soon enough. A Docker-container will never become polluted over time; it doesn't live long.
- The versioning of Docker-files becomes a major factor. We can have several distinct Docker-files, each suited to their individual project. Some will hold a different set of tools than others. Also, significant, it won't hold applications that aren't needed for solving the particular problem you're working; no e-mail app, no chat app, no office-app if that's not required. It's easy to not become distracted when the apps that might distract us aren't installed.
- Sharing is caring. I can work on the exact same development environment as my colleague, as we're both tied in to the same dev setup. I've so suffered in the past when for some reason I wasn't able to access the same resources, or install the same apps, as my colleague. Everything is the same; it narrows the focus on the project. And when I need to hand the project over to the developer next to me, I can be sure the environment will work the way I intended
- I needn't worry about my machine breaking down. Not that this has happened many times during my career, but there has been the odd irrecoverable crash. With a Docker environment, I just need to get my hand on a machine, any damn machine that will

run Docker. Pull the Docker image, spin up the container, get to work.

There are, of course, alternatives. I've often used Virtual Machines, that I've provisioned and enhanced with scripts, be it apt-get's on Linux or Choco on Windows. But they've never worked for me in the same fashion as does Docker-containers. They too became polluted over time, or it was a hassle to take them on the road with me. Docker containers are easier; the fire-and-forget nature appeals to me. A virtual machine takes up a chunk of space that you'll only get back when you stop using it; a Docker-container only takes up space for as long as you use it - though we do need to remember to clean up after ourselves.

It's not all pros, there are some cons, of course there are. If I'm not disciplined enough to commit my development changes to source control, well, let's just say that a Docker-container doesn't have a long attention-span! But that really all I can hold against the notion.

The potential in tying in the development environment with project-specific software and -configuration is, I can testify, quite productivity enhancing - and I, for one, won't be going back to maintaining my own development machine.

Enough talk, let's try it out. Install the Docker community edition if you don't have it already. Then take a glance at my template Docker-file, below, for an Ubuntu 17.10 with VS Code and a few related extensions installed, as well as a Firefox Browser, the Postman API development tool and a few other tools of my personal liking. It's heavily annotated so I'd like to try and let the comments speak for themselves:

Hide Shrink ▲ Copy Code

```
# Get the distro:
FROM ubuntu:17.10
ENV DEBIAN_FRONTEND=noninteractive

# Do some general updates.
RUN apt-get clean && rm -rf /var/lib/apt/lists/* &&  apt-get clean &&  apt-get update -y &&  apt-get
upgrade -y
RUN apt-get install -y software-properties-common
RUN add-apt-repository universe
RUN apt-get install -y cups curl sudo libgconf2-4 iputils-ping libxss1 wget xdg-utils libpango1.0-0 fonts-
liberation
RUN apt-get update -y && apt-get install -y software-properties-common && apt-get install -y locales

# Let's add a user, so we have a chance to, well, actually use the machine.
ENV USER='thecoder'
ENV PASSWORD='password'
RUN groupadd -r $USER -g 433 \
    && useradd -u 431 -r -g $USER -d /home/$USER -s /bin/bash -c "$USER" $USER \
    && adduser $USER sudo \
    && mkdir /home/$USER \
    && chown -R $USER:$USER /home/$USER \
    && echo $USER':'$PASSWORD | chpasswd

# Let's add a super user, too. Same password
ENV SUDOUSER='theadmin'
ENV PASSWORD='password'
RUN groupadd $SUDOUSER \
    && useradd -r -g $SUDOUSER -d /home/$SUDOUSER -s /bin/bash -c "$SUDOUSER" $SUDOUSER \
    && adduser $SUDOUSER sudo \
    && mkdir /home/$SUDOUSER \
    && chown -R $SUDOUSER:$SUDOUSER /home/$SUDOUSER \
    && echo $SUDOUSER':'$PASSWORD | chpasswd

# Configure timezone and locale to en_US. __Change locale and timezone to whatever you want.__
 ENV LANG="en_US.UTF-8"
 ENV LANGUAGE=en_US
 RUN locale-gen en_US.UTF-8 && locale-gen en_US
 RUN echo "Europe/Copenhagen" > /etc/timezone && \
     apt-get install -y locales && \
     sed -i -e "s/# $LANG.*/$LANG.UTF-8 UTF-8/" /etc/locale.gen && \
     dpkg-reconfigure --frontend=noninteractive locales && \
     update-locale LANG=$LANG

# Create an keyboard-layout file, so we won't have to set it every time the machine starts. Just replace
```

```
XKBLAYOUT="dk" with your layout, ex. "us".
# RUN printf '# Consult the keyboard(5) manual
page.\nXKBMODEL="pc105"\nXKBLAYOUT="dk"\nXKBVARIANT=""\nXKBOPTIONS=""\nBACKSPACE="guess"\n'"" >
/etc/default/keyboard
# - doesn't work :(
# set danish keyboard layout
# RUN setxkbmap dk  - doesnt work :(

# Install some much needed programs - nano, midnight commander, guake terminal
RUN apt-get install nano -y
RUN apt-get install mc -y
RUN apt-get install guake -y

# Use the Xfce desktop. Because it's nice to look at, in my opinion.
RUN apt-get update -y && \
    apt-get install -y xfce4
# There's also the MATE desktop-enviroment. Bit more light-weight.
#RUN apt-get update -y && \
#   apt-get install -y mate-desktop-environment-extras

# Install git
RUN apt-get install -y git
# Install Python.
# Whoa, waitaminute - Ubuntu 17.10 already comes with Python 3.6 as default. Just run python3 to invoke it.

# Install Firefox
RUN apt-get install firefox -y

# Install Postman to 'opt' dir
RUN wget https://dl.pstmn.io/download/latest/linux64 -O postman.tar.gz
RUN tar -xzf postman.tar.gz -C /opt
RUN rm postman.tar.gz

# Install Visual Studio Code
ENV VSCODEPATH="https://go.microsoft.com/fwlink/?LinkID=760868"
RUN curl -fSL "${VSCODEPATH}" -o vscode.deb && dpkg -i vscode.deb

# To make it easier to automate and configure VS Code, it is possible to list, install,
# and uninstall extensions from the command line. When identifying an extension, provide
# the full name of the form publisher.extension, for example donjayamanne.python.
USER $USER
WORKDIR /home/$USER

# Enable viewing git log, file history, compare branches and commits -
https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory
RUN code --install-extension donjayamanne.githistory
# Install Ms' python - linting, debugging, intellisense, etc.
RUN code --install-extension ms-python.python
# Install code outline provider - better code visualization in the explorer pane
RUN code --install-extension patrys.vscode-code-outline


# Annnnnd back to root for the remainder of this session.
USER root

# Install nomachine, the remote-desktop server that enables us to remote into the container image.
# You don't have to rely on my choice of NoMachine - just go to their website and get a different one, if
you want.
ENV NOMACHINE_PACKAGE_NAME nomachine_6.1.6_9_amd64.deb
ENV NOMACHINE_MD5 00b7695404b798034f6a387cf62aba84

RUN curl -fSL "http://download.nomachine.com/download/6.1/Linux/${NOMACHINE_PACKAGE_NAME}" -o nomachine.deb
\
&& echo "${NOMACHINE_MD5} *nomachine.deb" | md5sum -c - \
&& dpkg -i nomachine.deb


# Create an executable file that starts the NoMachine remote desktop server.
# A unix executable .sh-file must start with #!/bin/bash. '\n' means 'newline'.
# Note how the file ends with a /bin/bash-command. That's deliberate, it allows
# us do - don't ask me how - keep the container running when we use it later.
RUN printf '#!/bin/bash\n/etc/NX/nxserver --startup\n/bin/bash'"" > /etc/NX/nxserverStart.sh
# Now make the executable _actually_ executable ...
RUN chmod +x /etc/NX/nxserverStart.sh
# ... and start the nomachine-remote server when the container runs, and ...
CMD ["/etc/NX/nxserverStart.sh"]
#... happy developing! Use a NoMachine-client program to log into the server.
# PS: remember to run the container with the -d and -t arguments.
# Check the readme.md file, https://github.com/harleydk/linuxRemoteDocker/blob/master/README.md
```

If you're entirely new to Docker, hopefully the comments should make it clearer in what's going on here. We start out by downloading a Linux distribution, then do a bunch of RUN commands which execute when we progress to build the image. This includes also setting up a user and a super-user. Save the file as 'Dockerfile', without extension, and execute, in the directory of this new Docker-recipe, the ...

Hide Copy Code

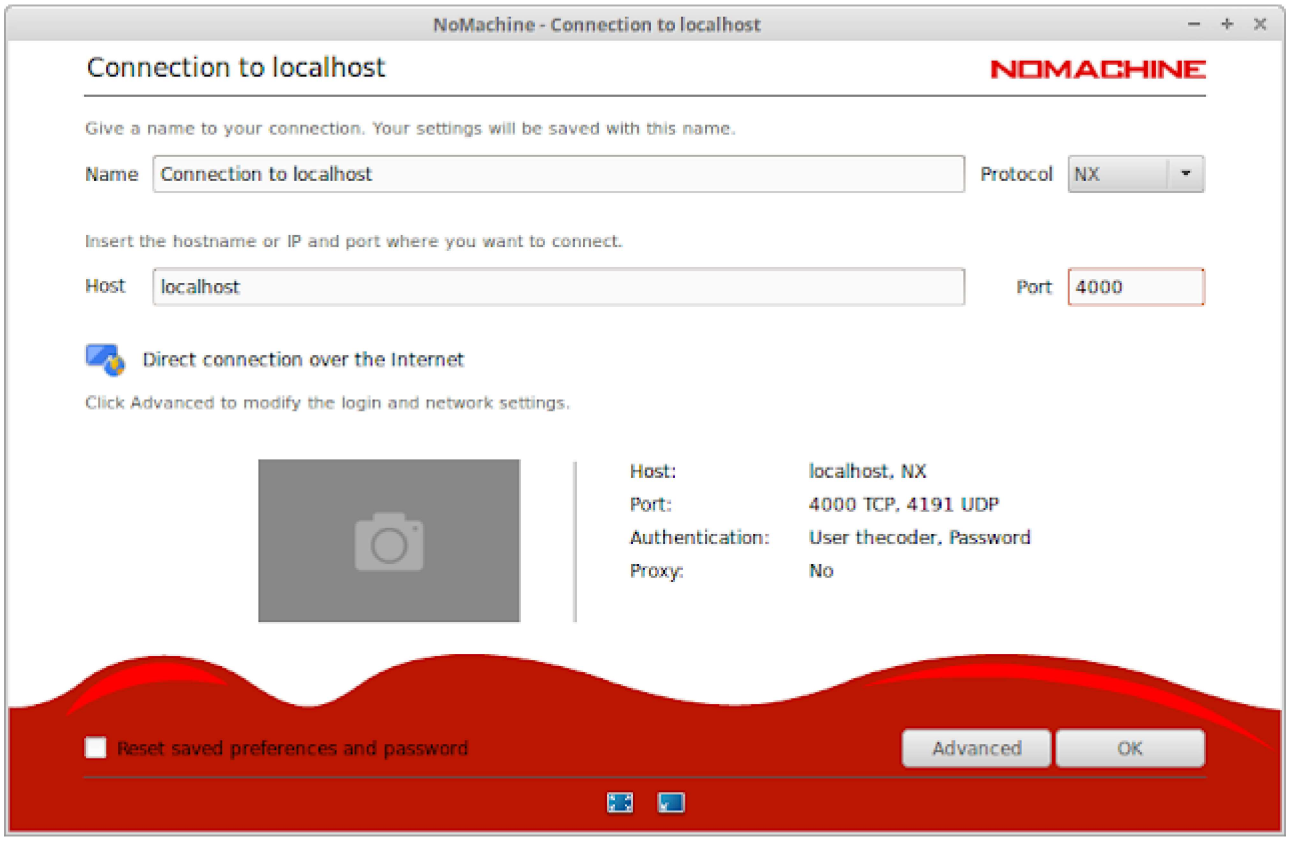
```
docker build -t ‘linux_remote_pc’ .<br />
```

... command. Then sit back and wait for a bit; the individual steps of the recipe will be executed and the image built. Now's the time to put it to use. The image installs and runs a NoMachine service, enabling remote desktop connections via a NoMachine-client application. It's fast and free. So go ahead and install a client from their web-site, https://www.nomachine.com/download, and spin up a Docker container from the Docker image just built, by issuing the following command:

Hide Copy Code

```
docker run -d -t -p 4000:4000 --name=linux_remote_pc --cap-add=SYS_PTRACE linux_remote_pc_image
```

This will spin up an instance, a 'Docker container' as it's called, with the port 4000 of the Docker host mapped to port 4000 on the container - the port that we'll connect to, as we remote access the container. So, on a machine on the same network as the Docker host, start the NoMachine client and connect:


NoMachine - Connection to localhost

Connection to localhost

Give a name to your connection. Your settings will be saved with this name.

Name: Connection to localhost
Protocol: NX

Insert the hostname or IP and port where you want to connect.

Host: localhost
Port: 4000

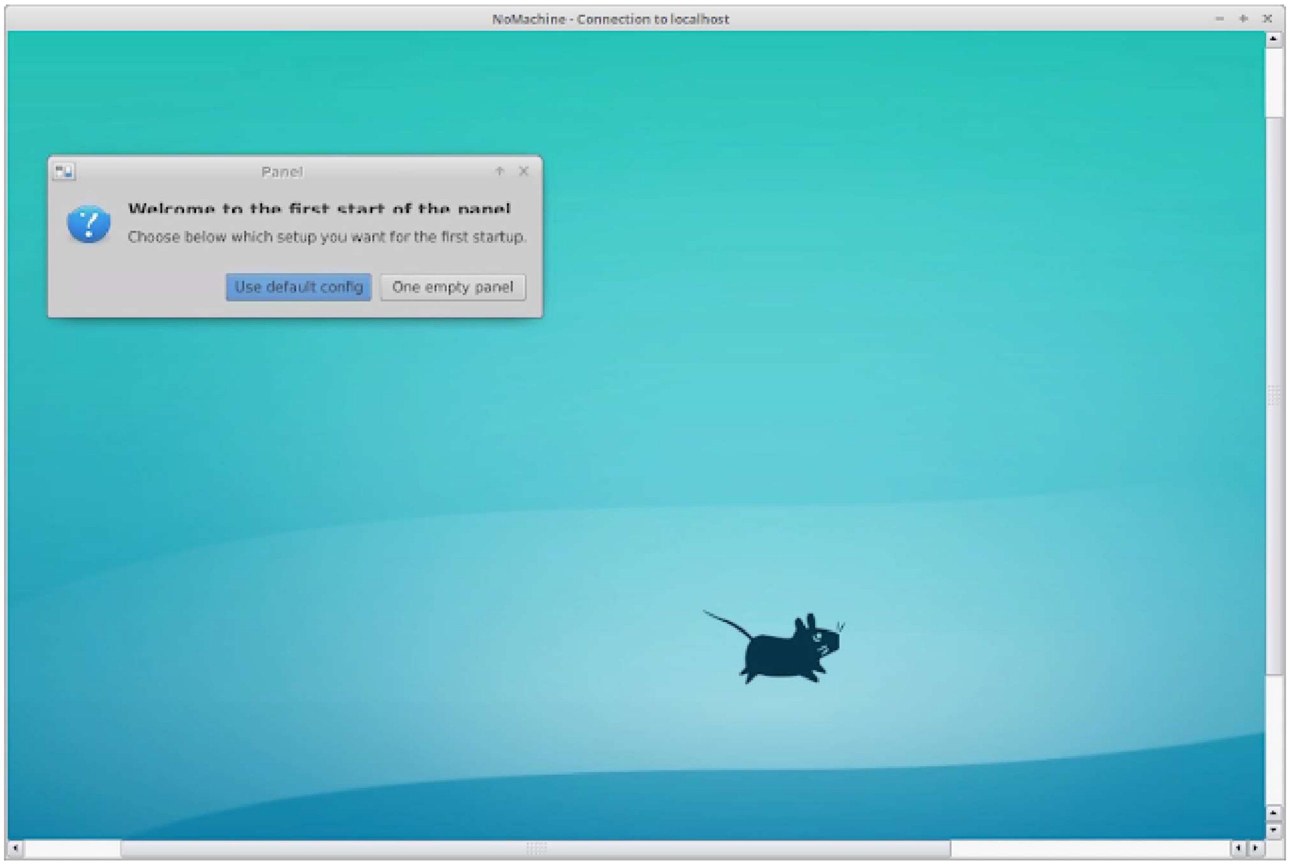
Direct connection over the Internet

Click Advanced to modify the login and network settings.

Host:	localhost, NX
Port:	4000 TCP, 4191 UDP
Authentication:	User thecoder, Password
Proxy:	No

Reset saved preferences and password

...Where 'localhost' would of course be the host-name of the machine where the Docker-container is running. Having connected to the container, this should be the first that you see:

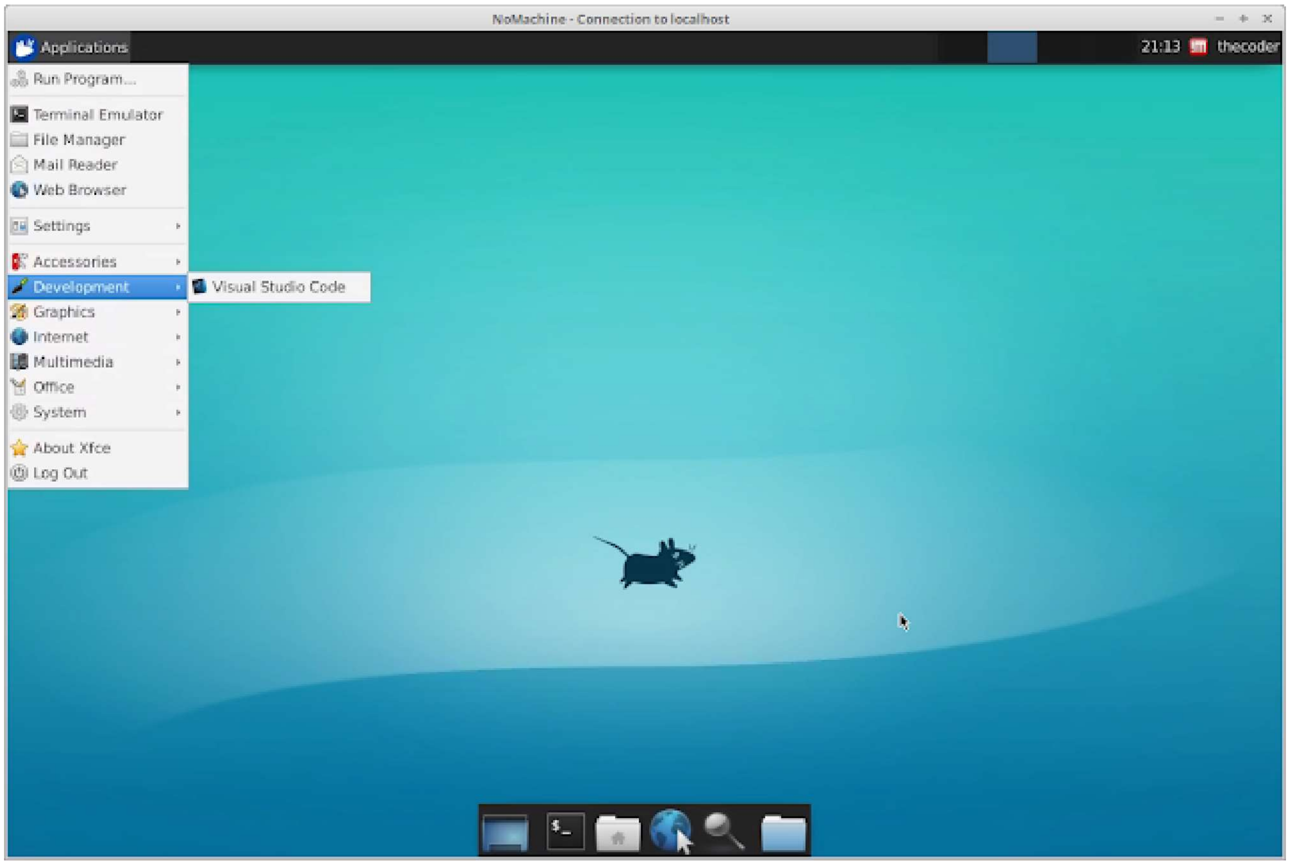


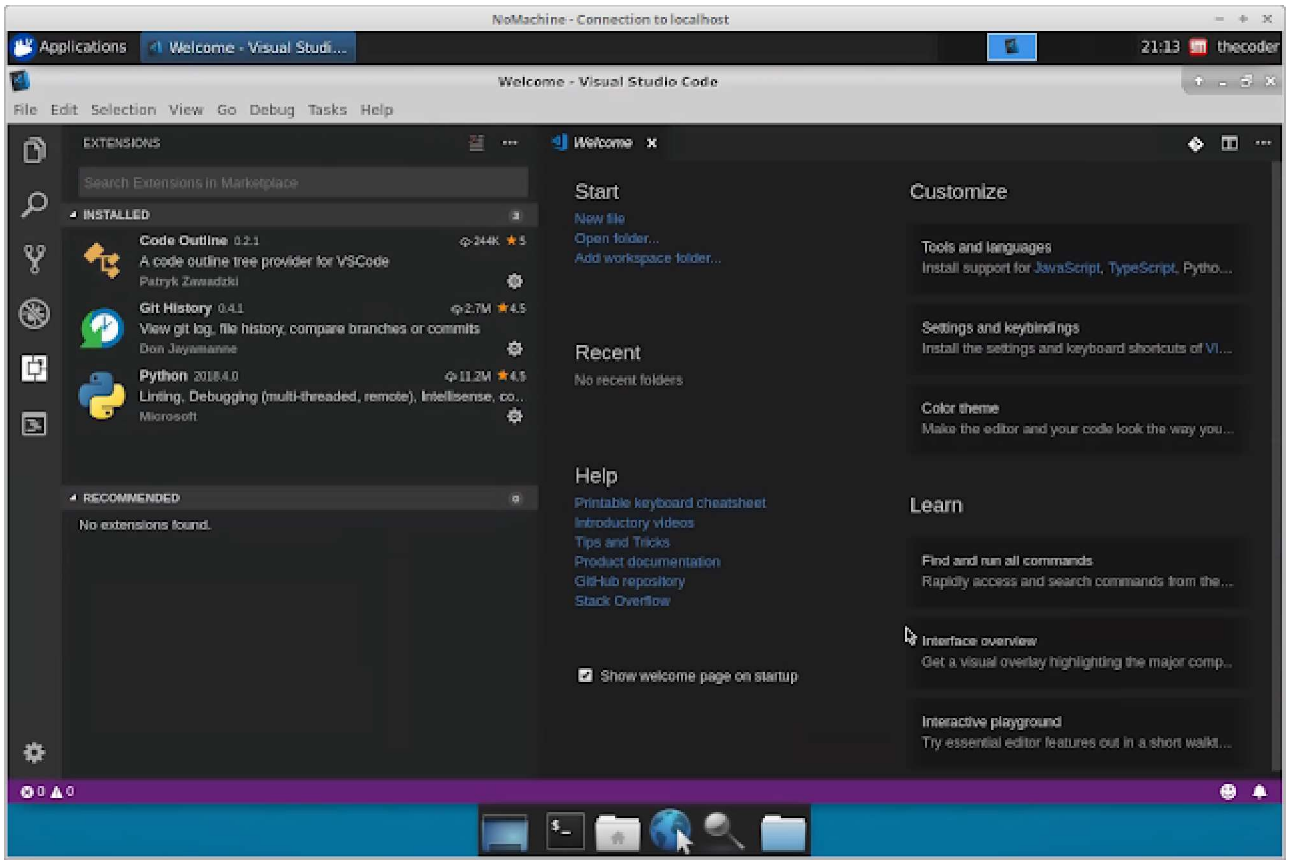
Welcome to the first start of the panel

Choose below which setup you want for the first startup.

The machine tells us it lacks some basic configuration - that’s because we installed it unattended, so as not to have to deal initially with all those setup-choices. Just press ‘use default config’ and be done with it.

That’s it - you now have a machine with VS Code installed:




NoMachine - Connection to localhost
Applications
Welcome - Visual Studi...
21:13 thecoder
Welcome - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXTENSIONS
Search Extensions in Marketplace
INSTALLED
Code Outline 0.2.1
A code outline tree provider for VSCode
Patryk Zawadzki
Git History 0.4.1
View git log, file history, compare branches or commits
Don Jayamanne
Python 2018.4.0
Linting, Debugging (multi-threaded, remote), Intellisense, co...
Microsoft
RECOMMENDED
No extensions found.
Welcome
Start
New file
Open folder...
Add workspace folder...
Recent
No recent folders
Help
Printable keyboard cheatsheet
Introductory videos
Tips and Tricks
Product documentation
GitHub repository
Stack Overflow
Show welcome page on startup
Customize
Tools and languages
Install support for JavaScript, TypeScript, Pytho...
Settings and keybindings
Install the settings and keyboard shortcuts of Vi...
Color theme
Make the editor and your code look the way you...
Learn
Find and run all commands
Rapidly access and search commands from the...
Interface overview
Get a visual overlay highlighting the major comp...
Interactive playground
Try essential editor features out in a short walkt...

If you're new to Docker there're lots of tutorials and documentation available. Highly recommend trying build commands on a 'hello world' Dockerfile instead of the slightly more complex one, above. If just to get a feel for the command-line interface and command responses.

The Docker-file above is maintained at my github repository, https://github.com/harleydk/linuxRemoteDocker, which you can clone if you like. There's a bit of further documentation available there.

I'd much like to hear from you, if you find it useful and/or if I can help out in some way. So drop me a note in the comments or create a new github issue, https://github.com/harleydk/linuxRemoteDocker/issues and I'll see what I can do.

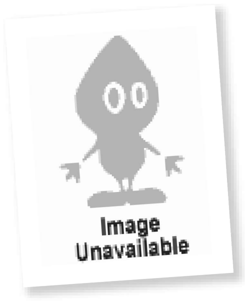
Thanks for reading & happy developing.

License

Share

TWITTER FACEBOOK

About the Author



harleydk
No Biography provided

D
e
n
m
a
r
k

You may also be interested in...

Public, Private, and Hybrid Cloud: What's the difference?

A Solution Blueprint for DevOps

Dockerize ASP.NET Boilerplate Projects

DaReport PrintDocument - A Printing Utility for .NET

A look at Docker

Building a DevOps CI/CD Pipeline for ASP.NET

Core with VSTS

Comments and Discussions

You must **Sign In** to use this message board.

Search Comments

First Prev Next

How does this compare to using Vagrant?
wheelman570z 5-Jun-18 12:03

Thanks for a useful idea... having just built a VM with my new Dev Environment...
Kirk 10389821 5-Jun-18 7:38

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.