

Dockerfile

<https://docs.docker.com/engine/reference/builder/>

https://philipzheng.gitbook.io/docker_practice/dockerfile/basic_structure

docker build 指令

- Docker can build images automatically by reading the instructions from a Dockerfile.
- A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image.
- Using docker build users can create an automated build that executes several command-line instructions in succession.

Dockerfile

- 使用 Dockerfile 建立自定義 映像檔
- 四部分：
 - 基底映像檔資訊
 - 維護者資訊
 - 映像檔操作指令
 - 容器啟動時執行指令

#註解

```
1 # This dockerfile uses the ubuntu image
2 # VERSION 2 - EDITION 1
3 # Author: docker_user
4 # Command format: Instruction [arguments / command] ..
5
6 # 基本映像檔，必須是第一個指令 基底映像檔資訊
7 FROM ubuntu
8
9 # 維護者: docker_user <docker_user at email.com> (@docker_user) 維護者資訊
10 MAINTAINER docker_user docker_user@email.com
11
12 # 更新映像檔的指令 映像檔操作指令 (RUN)
13 RUN echo "deb http://archive.ubuntu.com/ubuntu/ raring main universe" >> /etc/apt/s
14 RUN apt-get update && apt-get install -y nginx
15 RUN echo "\ndaemon off;" >> /etc/nginx/nginx.conf
16
17 # 建立新容器時要執行的指令
18 CMD /usr/sbin/nginx 容器啟動時執行指令 (CMD)
```

每運行一條 RUN 指令，
映像檔就會新增一層。

指令

FROM

格式為 FROM <image> 或 FROM <image>:<tag> 。

第一條指令必須為 FROM 指令。並且，如果在同一個 Dockerfile 中建立多個映像檔時，可以使用多個 FROM 指令（每個映像檔一次）。

MAINTAINER

格式為 MAINTAINER <name>，指定維護者訊息。

RUN

格式為 RUN <command> 或 RUN ["executable", "param1", "param2"] 。

前者將在 shell 終端中運行命令，即 /bin/sh -c；後者則使用 exec 執行。指定使用其它終端可以透過第二種方式實作，例如 RUN ["/bin/bash", "-c", "echo hello"] 。

每條 RUN 指令將在當前映像檔基底上執行指定命令，並產生新的映像檔。當命令較長時可以使用 \ 來換行。

CMD

支援三種格式

CMD ["executable","param1","param2"] 使用 exec 執行，推薦使用；

CMD command param1 param2 在 /bin/sh 中執行，使用在給需要互動的指令；

CMD ["param1","param2"] 提供給 ENTRYPOINT 的預設參數；

指定啟動容器時執行的命令，每個 Dockerfile 只能有一條 CMD 命令。如果指定了多條命令，只有最後一條會被執行。

如果使用者啟動容器時候指定了運行的命令，則會覆蓋掉 CMD 指定的命令。

指令

例如: -p 8080:80
這邊開放的是container內的 80 port
host 的 8080 port 則是在 docker run 的時候指定

EXPOSE

格式為 EXPOSE <port> [<port>...]。

設定 Docker 伺服器容器對外的埠號，供外界使用。在啟動容器時需要透過 -P，Docker 會自動分配一個埠號轉發到指定的埠號。

ENV

格式為 ENV <key> <value>。指定一個環境變數，會被後續 RUN 指令使用，並在容器運行時保持。

例如

ENV PATH /usr/local/postgres-\$PG_MAJOR/bin:\$PATH

ADD

格式為 ADD <src> <dest>。

該命令將複製指定的 <src> 到容器中的 <dest>。

其中 <src> 可以是 Dockerfile 所在目錄的相對路徑；

也可以是一個 URL；

還可以是一個 tar 檔案（其複製後會自動解壓縮）。

指令

COPY

格式為 COPY <src> <dest> 。

複製本地端的 <src> (為 Dockerfile 所在目錄的相對路徑) 到容器中的 <dest> 。

當使用本地目錄為根目錄時，推薦使用 COPY 。

ENTRYPOINT

兩種格式：

ENTRYPOINT ["executable", "param1", "param2"]

ENTRYPOINT command param1 param2 (shell中執行) 。

指定容器啟動後執行的命令，並且不會被 docker run 提供的參數覆蓋。

每個 Dockerfile 中只能有一個 ENTRYPOINT，當指定多個時，只有最後一個會生效。

VOLUME

格式為 VOLUME ["/data"] 。

建立一個可以從本地端或其他容器掛載的掛載點，一般用來存放資料庫和需要保存的資料等。

指令

USER

格式為 USER daemon。

指定運行容器時的**使用者名稱**或 UID，後續的 RUN 也會使用指定使用者。

當服務不需要管理員權限時，可以透過該命令指定運行使用者。並且可以在之前建立所需要的使用者，例如：RUN groupadd -r postgres && useradd -r -g postgres postgres。要臨時取得管理員權限可以使用 **gosu**，而不推薦 sudo。

WORKDIR

格式為 WORKDIR /path/to/workdir。

為後續的 RUN、CMD、ENTRYPOINT 指令指定工作目錄。

可以使用多個 WORKDIR 指令，後續命令如果參數是相對路徑，則會基於之前命令指定的路徑。

例如：

WORKDIR /a

WORKDIR b

WORKDIR c

RUN pwd

則最終路徑為 /a/b/c。

指令

ONBUILD

格式為 ONBUILD [INSTRUCTION]。

指定當建立的映像檔作為其它新建立映像檔的基底映像檔時，所執行的操作指令。

例如，Dockerfile 使用以下的內容建立了映像檔 image-A。

```
1 [...]
2 ONBUILD ADD . /app/src
3 ONBUILD RUN /usr/local/bin/python-build --dir /app/src
4 [...]
```

如果基於 image-A 建立新的映像檔時，新的 Dockerfile 中使用 FROM image-A 指定基底映像檔時，會自動執行 ONBUILD 指令內容，等於在後面新增了兩條指令。

```
1 FROM image-A
2
3 #Automatically run the following
4 ADD . /app/src
5 RUN /usr/local/bin/python-build --dir /app/src
```

使用 ONBUILD 指令的映像檔，推薦在標籤中註明，例如 ruby:1.9-onbuild。

建立映像檔

編輯完成 Dockerfile 之後，可以透過 `docker build` 命令建立映像檔。

基本的格式為 `docekr build [選項] 路徑`，

該命令將讀取指定路徑下（包括子目錄）的 Dockerfile，並將該路徑下所有內容發送給 Docker 伺服器端，由伺服器端來建立映像檔。

因此一般會建議放置 Dockerfile 的目錄為空目錄。

也可以透過 `.dockerignore` 檔案（每一行新增一條排除模式：`exclusion patterns`）來讓 Docker 忽略路徑下的目錄和檔案。

要指定映像檔的標籤資訊，可以透過 `-t` 選項，例如

```
$ sudo docker build -t myrepo/myapp /tmp/test1/
```

```
1 # Nginx
2 #
3 # VERSION          0.0.1
4
5 FROM      ubuntu
6 MAINTAINER Victor Vieux <victor@docker.com>
7
8 RUN apt-get update && apt-get install -y inotify-tools nginx apache2 openssh-server
9
10 # Firefox over VNC
11 #
12 # VERSION          0.3
13
14 FROM ubuntu
15
16 # Install vnc, xvfb in order to create a 'fake' display and firefox
17 RUN apt-get update && apt-get install -y x11vnc xvfb firefox
18 RUN mkdir /.vnc
19 # Setup a password
20 RUN x11vnc -storepasswd 1234 ~/.vnc/passwd
21 # Autostart firefox (might not be the best way, but it does the trick)
22 RUN bash -c 'echo "firefox" >> /.bashrc'
23
24 EXPOSE 5900
25 CMD ["x11vnc", "-forever", "-usepw", "-create"]
26
27 # Multiple images example
28 #
29 # VERSION          0.1
30
31 FROM ubuntu
32 RUN echo foo > bar
33 # Will output something like ==> 907ad6c2736f
34
35 FROM ubuntu
36 RUN echo moo > oink
37 # Will output something like ==> 695d7793cbe4
38
39 # You'll now have two images, 907ad6c2736f with /bar, and 695d7793cbe4 with
40 # /oink.
```

Example 1: 建立一個 Docker website

Step 1:

vi dockerfile

<貼上 Dockerfile 內容>

dockerfile

FROM ubuntu

井字號開頭是註解

USER root

<= 使用root身分

RUN apt-get update && apt-get install -y \

<= 安裝軟體

python3 \

curl && \

rm -rf /var/lib/apt/lists/*

RUN groupadd -r nonroot && \

useradd -r -g nonroot -d /home/nonroot -s /sbin/nologin -c "Nonroot User" nonroot && \

mkdir /home/nonroot && \

chown -R nonroot:nonroot /home/nonroot

USER nonroot

<= 建立帳號

WORKDIR /home/nonroot/

RUN curl -o index.html http://info.cern.ch/hypertext/WWW/TheProject.html

<= 下載網頁檔案

USER nonroot

<= 使用nonroot身分

EXPOSE 3000

WORKDIR /home/nonroot/


CMD ["python3", "-m", "http.server", "3000"]

<= 執行python程式

Example 1: 建立一個 Docker website

Step 2: `docker build -t=docker_website .`

```
root@cc1lab-virtual-machine:~/dockerfile01# vi dockerfile
root@cc1lab-virtual-machine:~/dockerfile01# docker build -t=docker_website .
Sending build context to Docker daemon 3.584kB
Step 1/11 : FROM ubuntu:14.04
14.04: Pulling from library/ubuntu
2e6e20c8e2e6: Extracting [=====>] 22.28MB/70.69MB
30bb187ac3fc: Download complete
b7a5bcc4a58a: Download complete
```

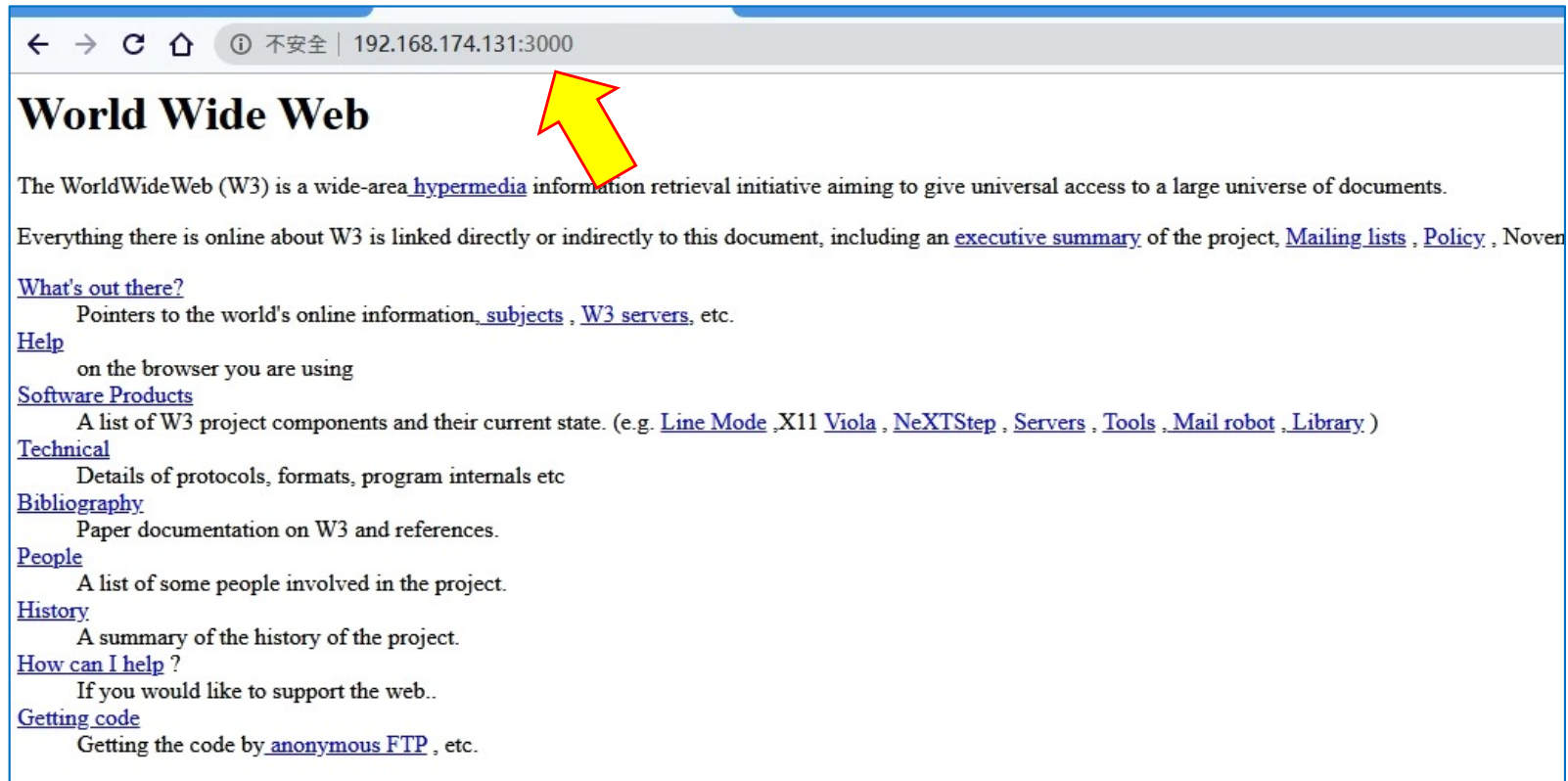


```
root@cc1lab-virtual-machine:~/dockerfile01# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
docker_website	latest	1ceef750d52d	About a minute ago	200MB
node	slim	0e2e78467169	7 days ago	165MB
ubuntu	latest	1d622ef86b13	7 weeks ago	73.9MB
ubuntu	18.04	c3c304cb4f22	7 weeks ago	64.2MB
ubuntu	14.04	6e4f1fe62ff1	6 months ago	197MB

根據dockerfile 建了一個 docker image, 名稱為 docker_website

Step3: `docker run -p 3000:3000 -i -t docker_website`



執行 `docker ps -a` 可看到已正在執行中的 container

```
root@cc1lab-virtual-machine:~# docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
870bdb002420   docker_website "python3 -m http.ser...  2 minutes ago Up 2 minutes  0.0.0.0:3000->3000/tcp             condescending_haibt
```

隨堂作業:

- 根據前面的 dockerfile 檔案，建立一個 docker_website²
- 更改網頁內容為

```
<html>
```

```
Hello docker! <br>
```

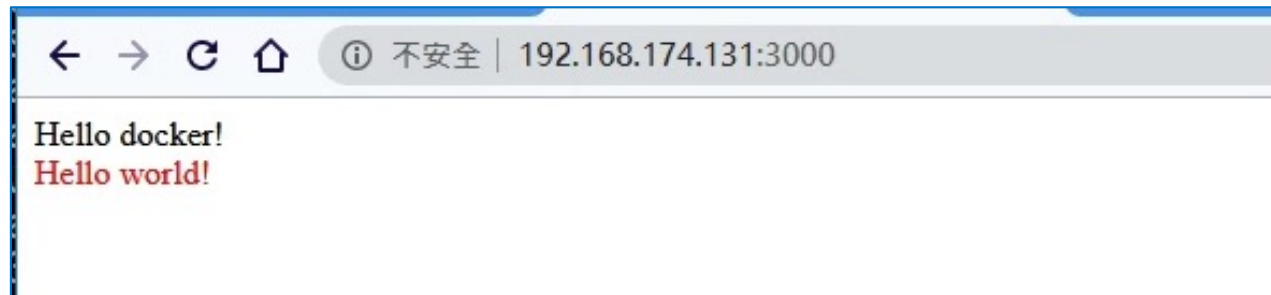
```
<font color='red'>Hello world!</font>
```

```
</html>
```

- 用瀏覽器連線觀看

思考:
如何更改網頁的原始碼?

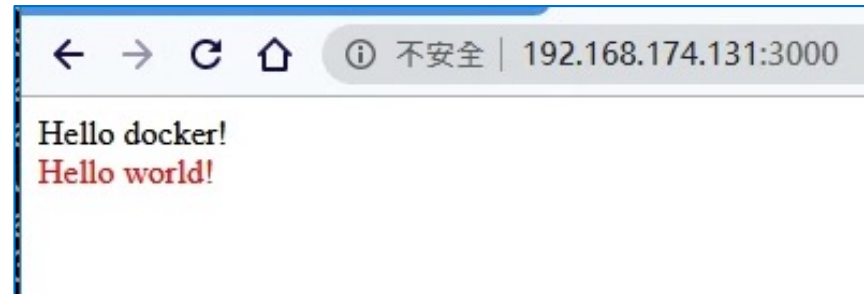
執行效果



把答案上傳到 e-learning (0421 作業區)

- word 檔案

- 執行結果截圖



- 說明文字:

- Step 1

- Step 2

- Step 3

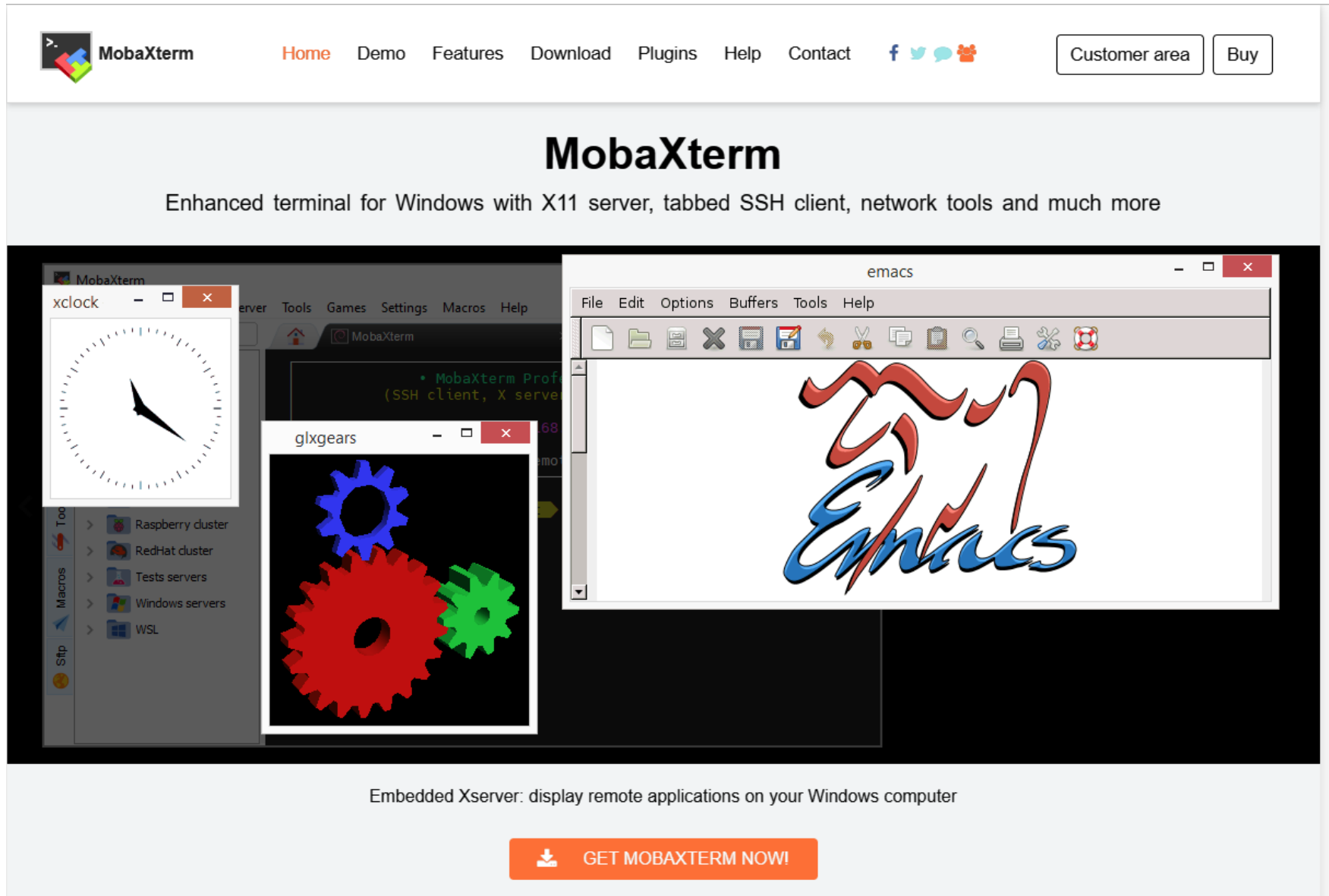
- ...

Example 3: 建立一個視窗container

2023/04/21 更新

Step 0: 安裝 mobaXterm

- <https://mobaxterm.mobatek.net/>



The screenshot displays the MobaXterm website. The header includes the MobaXterm logo, navigation links (Home, Demo, Features, Download, Plugins, Help, Contact), social media icons, and buttons for 'Customer area' and 'Buy'. The main heading is 'MobaXterm', followed by the tagline 'Enhanced terminal for Windows with X11 server, tabbed SSH client, network tools and much more'. Below this is a large image showing the MobaXterm application interface. The interface features a sidebar with categories like 'Tools', 'Games', 'Settings', 'Macros', and 'Help'. The main window displays a terminal with a green prompt, and several X11 applications are running: 'xclock' (a clock), 'glxgears' (gears), and 'emacs' (showing the Emacs logo). At the bottom, the text 'Embedded Xserver: display remote applications on your Windows computer' is visible, along with an orange button labeled 'GET MOBAXTERM NOW!' with a download icon.

MobaXterm

Home Demo Features Download Plugins Help Contact

Customer area Buy

MobaXterm

Enhanced terminal for Windows with X11 server, tabbed SSH client, network tools and much more

xclock

glxgears

emacs

Embedded Xserver: display remote applications on your Windows computer

GET MOBAXTERM NOW!

Step 1: 找出你的 windows 內部IP

- 開始 > cmd > ipconfig
- 尋找 跟 vmware ubuntu 一樣開頭的 IP

```
root@ccllab-virtual-machine:~# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:e9ff:fe73:8535 prefixlen 64 scopeid 0x20<link>
    ether 02:42:e9:73:85:35 txqueuelen 0 (Ethernet)
    RX packets 42452 bytes 507252560 (507.2 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 72323 bytes 90486613 (90.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 co

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.197.129 netmask 255.255.255.0 bro
    inet6 fe80::b8f1:35cd:4a7e:219d prefixlen 64 s
    ether 00:0c:29:c2:c5:f6 txqueuelen 1000 (Ether
    RX packets 186597 bytes 202300311 (202.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 400335 bytes 532201898 (532.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 co

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 9662 bytes 100368610 (100.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9662 bytes 100368610 (100.3 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 co

root@ccllab-virtual-machine:~#
```

這是 UBUNTU 的 IP

選取 命令提示字元

這是 windows 的內部 IP

```
媒體狀態 . . . . . : 媒體已中斷連線
連線特定 DNS 尾碼 . . . . . :

無線區域網路介面卡 區域連線* 2:

媒體狀態 . . . . . : 媒體已中斷連線
連線特定 DNS 尾碼 . . . . . :

乙太網路卡 VMware Network Adapter VMnet1:

連線特定 DNS 尾碼 . . . . . :
連結-本機 IPv6 位址 . . . . . : fe80::4433:ea87:20ed:a4af%22
IPv4 位址 . . . . . : 192.168.70.1
子網路遮罩 . . . . . : 255.255.255.0
預設閘道 . . . . . :

乙太網路卡 VMware Network Adapter VMnet8:

連線特定 DNS 尾碼 . . . . . :
連結-本機 IPv6 位址 . . . . . : fe80::5159:c3aa:c022:db60%16
IPv4 位址 . . . . . : 192.168.197.1
子網路遮罩 . . . . . : 255.255.255.0
預設閘道 . . . . . :

乙太網路卡 藍牙網路連線:

媒體狀態 . . . . . : 媒體已中斷連線
連線特定 DNS 尾碼 . . . . . :
```

Step 3: 編輯 Dockerfile 內容

```
FROM ubuntu
```

```
RUN apt-get update && apt-get install -y gedit
```

```
# Replace 1000 with your user / group id
```

```
RUN export uid=1000 gid=1000 && \
```

```
    mkdir -p /home/developer && \
```

```
    echo "developer:x:${uid}:${gid}:Developer,,,:/home/developer:/bin/bash" >> /etc/passwd && \
```

```
    echo "developer:x:${uid}:" >> /etc/group && \
```

```
    chown ${uid}:${gid} -R /home/developer
```

```
USER developer
```

```
ENV HOME /home/developer
```

```
CMD /usr/bin/gedit
```

Step 4: 製作 docker image

根據 Dockerfile 建立 image (建立名為gedit的image)

```
docker build -t=gedit .
```

Step 5: 設定 \$DISPLAY

- 在 Virtual machine UBUNTU 內設定

export DISPLAY=你的IP:0

承Step 1:

輸入 export DISPLAY=192.168.64.1:0

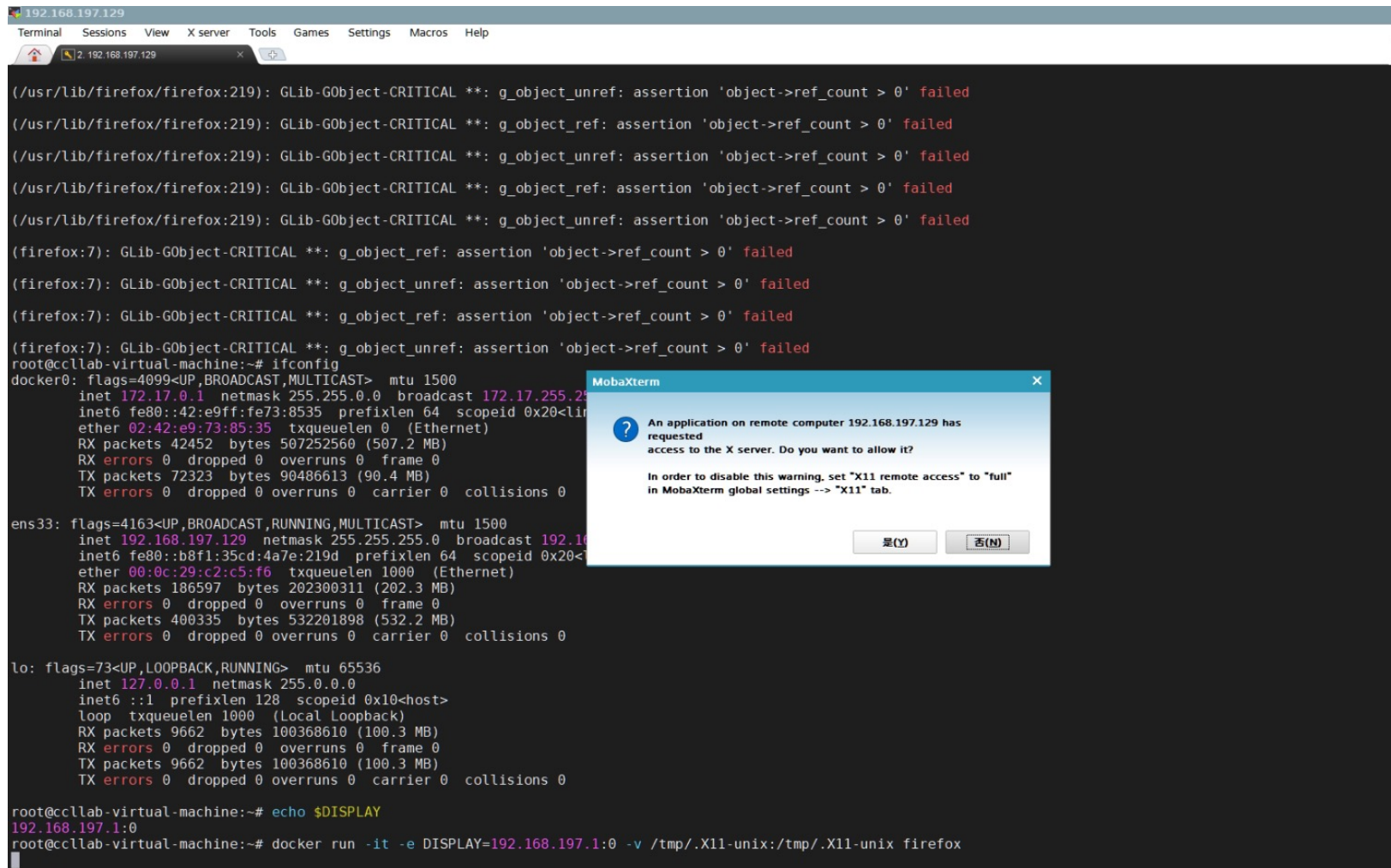
執行 echo \$DISPLAY 確認IP有設定成功

```
root@ccllab-virtual-machine:~# echo $DISPLAY
192.168.197.1:0
root@ccllab-virtual-machine:~#
```


Step 6: 執行 container

--rm 執行完後就自動刪除

`docker run -it --rm -e DISPLAY=192.168.64.1:0 -v /tmp/.X11-unix:/tmp/.X11-unix gedit`



The screenshot shows a terminal window with the following content:

```
192.168.197.129
Terminal Sessions View X server Tools Games Settings Macros Help
2. 192.168.197.129
(/usr/lib/firefox/firefox:219): GLib-GObject-CRITICAL **: g_object_unref: assertion 'object->ref_count > 0' failed
(/usr/lib/firefox/firefox:219): GLib-GObject-CRITICAL **: g_object_ref: assertion 'object->ref_count > 0' failed
(/usr/lib/firefox/firefox:219): GLib-GObject-CRITICAL **: g_object_unref: assertion 'object->ref_count > 0' failed
(/usr/lib/firefox/firefox:219): GLib-GObject-CRITICAL **: g_object_ref: assertion 'object->ref_count > 0' failed
(/usr/lib/firefox/firefox:219): GLib-GObject-CRITICAL **: g_object_unref: assertion 'object->ref_count > 0' failed
(firefox:7): GLib-GObject-CRITICAL **: g_object_ref: assertion 'object->ref_count > 0' failed
(firefox:7): GLib-GObject-CRITICAL **: g_object_unref: assertion 'object->ref_count > 0' failed
(firefox:7): GLib-GObject-CRITICAL **: g_object_ref: assertion 'object->ref_count > 0' failed
(firefox:7): GLib-GObject-CRITICAL **: g_object_unref: assertion 'object->ref_count > 0' failed
root@ccllab-virtual-machine:~# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    inet6 fe80::42:e9ff:fe73:8535 prefixlen 64 scopeid 0x20<link>
    ether 02:42:e9:73:85:35 txqueuelen 0 (Ethernet)
    RX packets 42452 bytes 507252560 (507.2 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 72323 bytes 90486613 (90.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.197.129 netmask 255.255.255.0 broadcast 192.168.197.255
    inet6 fe80::b8f1:35cd:4a7e:219d prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:c2:c5:f6 txqueuelen 1000 (Ethernet)
    RX packets 186597 bytes 202300311 (202.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 400335 bytes 532201898 (532.2 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 9662 bytes 100368610 (100.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 9662 bytes 100368610 (100.3 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

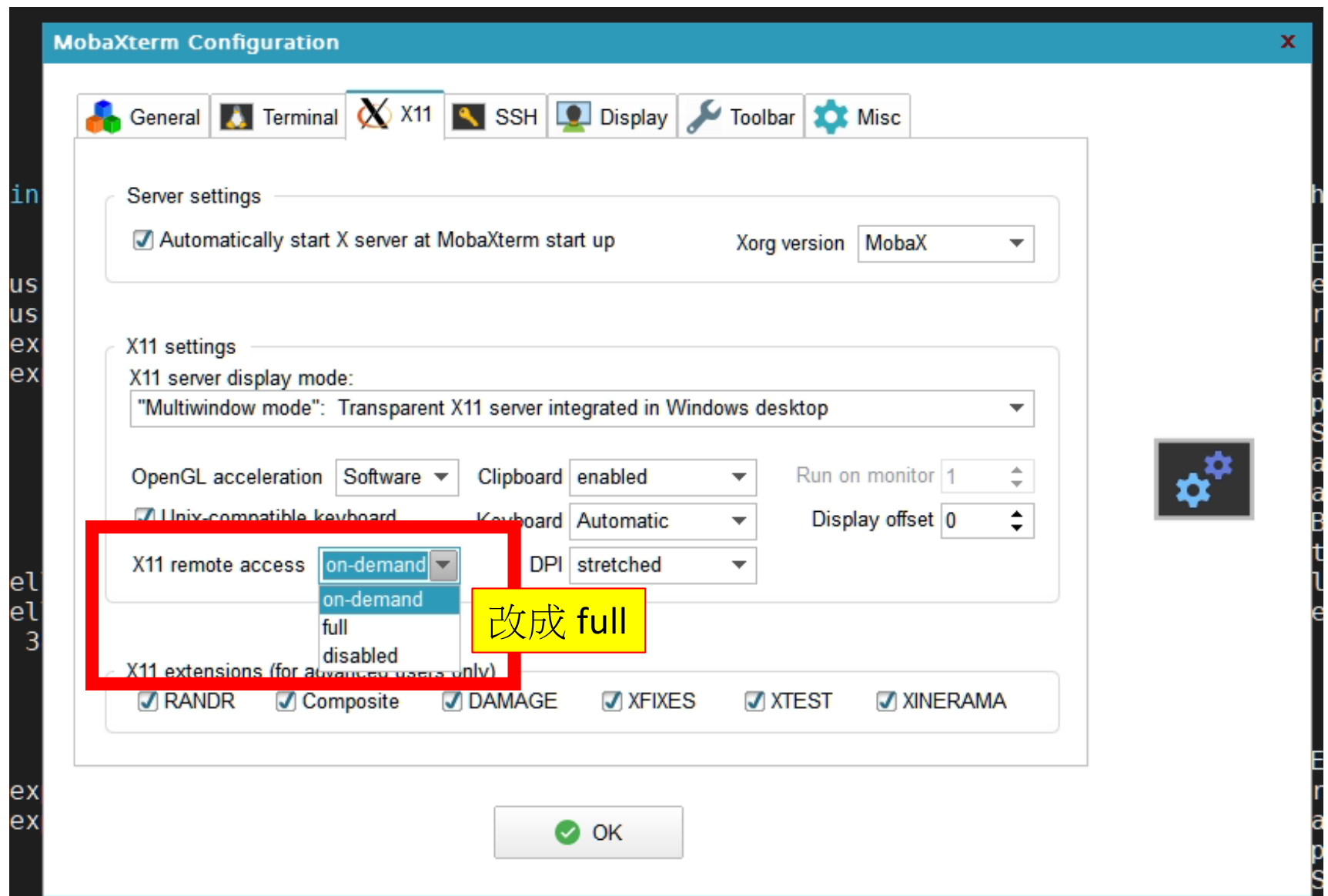
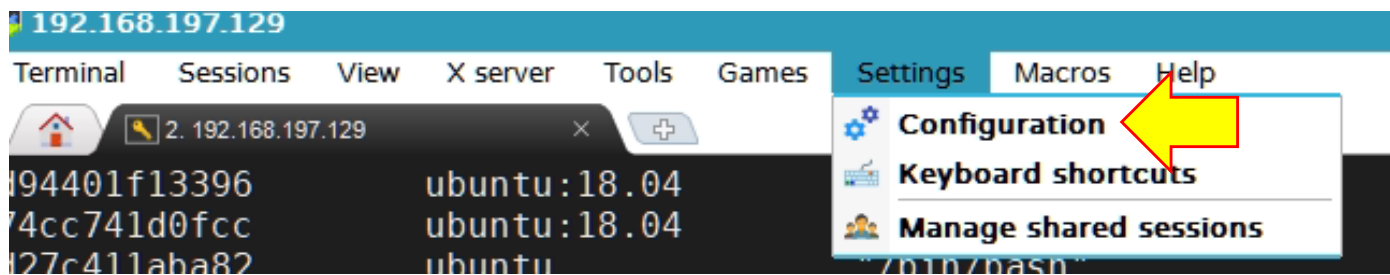
root@ccllab-virtual-machine:~# echo $DISPLAY
192.168.197.1:0
root@ccllab-virtual-machine:~# docker run -it -e DISPLAY=192.168.197.1:0 -v /tmp/.X11-unix:/tmp/.X11-unix firefox
```

A MobaXterm dialog box is overlaid on the terminal, asking for permission to access the X server:

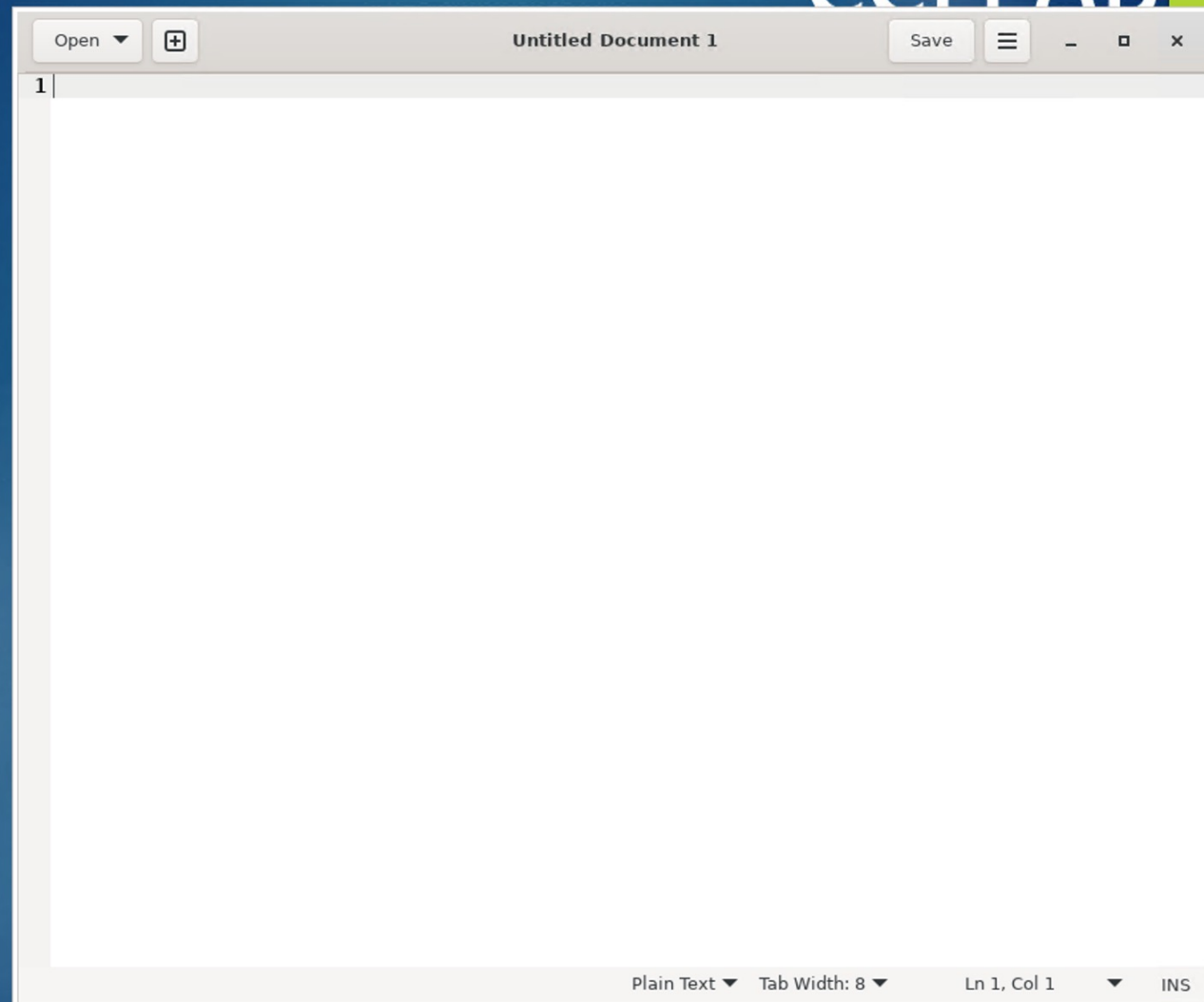
An application on remote computer 192.168.197.129 has requested access to the X server. Do you want to allow it?

In order to disable this warning, set "X11 remote access" to "full" in MobaXterm global settings --> "X11" tab.

Buttons: 是(Y) No(N)



CCILAB



CWL

Common workflow language

串接程式?

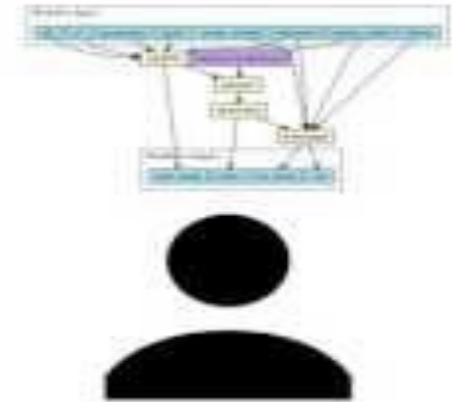
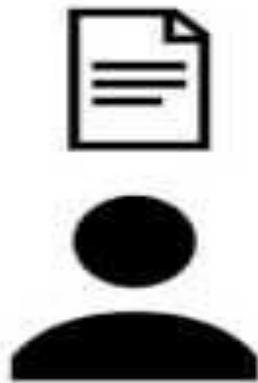
```
#!/bin/bash
```

```
APP1.exe --cpus 4 --output app1.output
```

```
APP2.exe --mem 16G --input app1.output --output app2.output
```

Introduction to CWL

<https://www.youtube.com/watch?v=86eY8xs-Vo8>



Common Workflow Language

Support [gitter](#) [join chat](#) [stars](#) 700

Donate

The Common Workflow Language (CWL) is a specification for describing analysis workflows and tools in a way that makes them portable and scalable across a variety of software and hardware environments, from workstations to cluster, cloud, and high performance computing (HPC) environments. CWL is designed to meet the needs of data-intensive science, such as Bioinformatics, Medical Imaging, Astronomy, Physics, and Chemistry.



CWL is developed by a multi-vendor working group consisting of organizations and individuals aiming to enable scientists to share data analysis workflows. The CWL project is maintained on Github and we follow the Open-Stand.org principles for collaborative open standards development. Legally CWL is a member project of Software Freedom Conservancy and is formally maintained by them.

CWL builds on technologies such as

User Guide

The CWL user guide provides a

[CWLの日本語での解説ドキュメント](#)

Specification

For developers and advanced users

Citation

To reference the CWL standard

Peter Amstutz, Michael R. Crus

Scales, Stian Soiland-Reyes, Lu

doi:10.6084/m9.figshare.311513

A collection of existing reference



CWL是一種描述命令行的工具並能夠使用命令行創建工作流，由於CWL是一種要求（類似規範）並不是一種特定的工具。

CWL描述的工作流能夠在支持CWL標準的多種平臺上使用。

CWL任務是獨立的，使用者必須明確輸入和輸出。

這種明確和獨立是複雜，可移植和可擴展的：工具和CWL描述的工作流能夠輕易勝任一些工作例如在不同的平臺上CWL Docker的實現。

CWL非常適合描述大規模集羣數據流，雲數據流以及多點平行運行的高性能計算環境。

CWL是一種規則，採用的是命令行與輸入輸出分開的方式。

jkovich, Matt

[rg/cwl/v1.0/](#)

<https://www.twblogs.net/a/5b8838f22b71775d1cdb1dc1>

<https://www.commonwl.org/>

Implementations


Software	Description	CWL support	Platform support
cwltool	Reference implementation of CWL	build failing	Linux, OS X, Windows, local execution only
Arvados	Distributed computing platform for data analysis on massive data sets. Using CWL on Arvados	build passing	AWS, GCP, Azure, Slurm
Toil	Toil is a workflow engine entirely written in Python.	build failing	AWS, Azure, GCP, Grid Engine, LSF, Mesos, OpenStack, Slurm, PBS/Torque
Rabix Bunny	An open-source, Java-based implementation of Common Workflow Language with support for multiple drafts/versions. See Rabix.io for details.	build failing	Linux, OS X, GA4GH TES (experimental)
Galaxy	Web-based platform for data intensive biomedical research.	alpha	-
cwl-tes	CWL engine backended by the GA4GH Task Execution API	build passing	Local, GCP, AWS, HTCondor, Grid Engine, PBS/Torque, Slurm
CWL-Airflow	Package to run CWL workflows in Apache-Airflow (supported by BioWardrobe Team, CCHMC)	build passing	Linux, OS X
REANA	RE usable ANALyses	Yes	Kubernetes, CERN OpenStack (OpenStack Magnum)
Cromwell	Cromwell workflow engine	build unstable alpha	local, HPC, Google, HtCondor
CWLEXEC	Apache 2.0 licensed CWL executor for IBM Spectrum LSF, supported by IBM for customers with valid contracts.	build unstable Yes, except for ExpressionTool	IBM Spectrum LSF 10.1.0.3+
Xenon	Run CWL workflows using Xenon	alpha	any Xenon backend : local, ssh, SLURM, Torque, Grid Engine
Consonance	orchestration tool for running SeqWare workflows and CWL tools	-	AWS, OpenStack, Azure
Apache Taverna	Domain-independent Workflow Management System	alpha	-
AWE	Workflow and resource management system for bioinformatics data analysis.	alpha	-
yacle	Yet Another CWL Engine	-	-

準備一個 Ubuntu 1804 VM

- `sudo -s`
- `apt-get update`
- `apt-get install vim`
- `apt-get install openssh-server`

```
root@ubuntu:~# sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2019-05-15 03:13:50 PDT; 1min 7s ago
     Docs: https://docs.docker.com
    Main PID: 16954 (dockerd)
      Tasks: 8
     CGroup: /system.slice/docker.service
             └─ 16954 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

May 15 03:13:49 ubuntu dockerd[16954]: time="2019-05-15T03:13:49.350857589-07:00" level=warni
May 15 03:13:49 ubuntu dockerd[16954]: time="2019-05-15T03:13:49.351015296-07:00" level=warni
May 15 03:13:49 ubuntu dockerd[16954]: time="2019-05-15T03:13:49.351124067-07:00" level=warni
May 15 03:13:49 ubuntu dockerd[16954]: time="2019-05-15T03:13:49.369873423-07:00" level=info
May 15 03:13:49 ubuntu dockerd[16954]: time="2019-05-15T03:13:49.740818826-07:00" level=info
```



Install CWLtool (Ubuntu 1804)

```
sudo apt-get install cwltool
```

Run on the command line

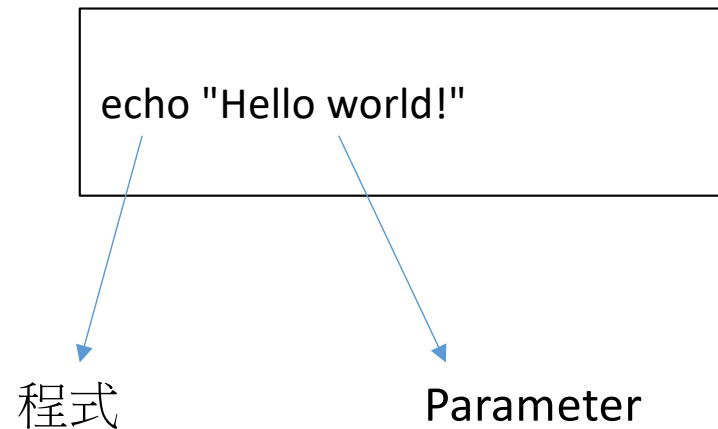
cwl-runner [tool-or-workflow-description] [input-job-settings]

Example: Grep and Count Workflow

Metadata	<pre>class: Workflow cwlVersion: v1.0</pre>	<pre>steps: grep: run: grep.cwl in: pattern: pattern infile: infiles scatter: infile out: [outfile] wc: run: wc.cwl in: infiles: grep/outfile out: [outfile]</pre>	Workflow Steps
Environment	<pre>requirements: - class: ScatterFeatureRequirement</pre>		
Input Parameters	<pre>inputs: pattern: string infiles: File[]</pre>		
Output Parameters	<pre>outputs: outfile: type: File outputSource: wc/outfile</pre>		

A Gentle Introduction to the CWL

- First example



想法: 把 程式 跟 參數 分開，保持日後修改參數的彈性

A Gentle Introduction to the CWL

- First example

1st-tool.cwl

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo
inputs:
  - id: message
    type: string
    inputBinding:
      position: 1
outputs: []
```

程式

Parameter

如有output放這邊
本例子 [] 表示沒有
輸出檔案

echo-job.yml

```
message: Hello world!
```

```
ccllab@ccllab-virtual-machine:~$ cwl-runner 1st-tool.cwl echo-job.yml
```

```
/home/ccllab/venv/bin/cwl-runner 1.0.20180502225535
```

```
Resolved '1st-tool.cwl' to 'file:///home/ccllab/1st-tool.cwl'
```

```
[job 1st-tool.cwl] /tmp/tmp0b6fk9uh$ echo \
```

```
'Hello world!'
```

```
Hello world!
```

```
[job 1st-tool.cwl] completed success
```

```
{}
```

```
Final process status is success
```

要執行的指令

指令執行結果

echo [-f] -i {example_int} --example-string {example_string} [--file={example_file}]

task-definition.cwl

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: echo
inputs:
  {
    example_flag:
      type: boolean
      inputBinding:
        position: 1
        prefix: -f
  }
  {
    example_string:
      type: string
      inputBinding:
        position: 3
        prefix: --example-string
  }
  {
    example_int:
      type: int
      inputBinding:
        position: 2
        prefix: -i
      separate: false
  }
  {
    example_file:
      type: File?
      inputBinding:
        prefix: --file=
        separate: false
        position: 4
  }
```

outputs: []

Argument name

cwltool task-definition.cwl input-job.yaml

input-job.yaml

```
example_flag: true
example_string: hello
example_int: 42
example_file:
  class: File
  path: whale.txt
```

Returning output files

tar.cwl

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: [tar, xf]
inputs:
  - id: tarfile
    type: File
    inputBinding:
      position: 1
outputs:
  - id: example_out
    type: File
    outputBinding:
      glob: hello.txt
```

tar-job.yml

```
tarfile:
  class: File
  path: hello.tar
```

(先產出 input files)

`touch hello.txt && tar -cvf hello.tar hello.txt`

`cwltool tar.cwl tar-job.yml`

```
root@ubuntu:~# cwltool tar.cwl tar-job.yml
/usr/bin/cwltool 1.0.20180302231433
Resolved 'tar.cwl' to 'file:///home/ccllab/tar.cwl'
[job tar.cwl] /tmp/tmpFPIdeP$ tar \
    xf \
    /tmp/tmpvwRe5G/stgaac3b7f9-e87a-44cc-9827-75448a006545/hello.tar
[job tar.cwl] completed success
{
  "example_out": {
    "checksum": "sha1$da39a3ee5e6b4b0d3255bfeef95601890afd80709",
    "basename": "hello.txt",
    "location": "file:///home/ccllab/hello.txt",
    "path": "/home/ccllab/hello.txt",
    "class": "File",
    "size": 0
  }
}
Final process status is success
```

Running tools inside Docker

docker.cwl

```
cwlVersion: v1.0
class: CommandLineTool
baseCommand: node
hints:
  - class: DockerRequirement
    dockerPull: node:slim
inputs:
  - id: src
    type: File
    inputBinding:
      position: 1
outputs: []
```

docker-job.yml

```
src:
  class: File
  path: hello.js
```

`sudo -s` (切換到 root 權限才能執行 docker)

`echo "console.log(\"Hello World\");" > hello.js`

`cwl-runner docker.cwl docker-job.yml`

```
root@ubuntu:~# vi docker.cwl
root@ubuntu:~# vi docker-job.yml
root@ubuntu:~# echo "console.log(\"Hello World\");" > hello.js
root@ubuntu:~# cwl-runner docker.cwl docker-job.yml
/usr/bin/cwl-runner 1.0.20180302231433
Resolved 'docker.cwl' to 'file:///home/ccllab/docker.cwl'
['docker', 'pull', 'node:slim']
slim: Pulling from library/node
743f2d6c1f65: Extracting [=====>] 15.6MB/22.49MB
89252b028f01: Download complete
204dbfab402e: Download complete
5a6179c77f5e: Download complete
```

```
root@ubuntu:~# cwl-runner docker.cwl docker-job.yml
/usr/bin/cwl-runner 1.0.20180302231433
Resolved 'docker.cwl' to 'file:///home/ccllab/docker.cwl'
['docker', 'pull', 'node:slim']
slim: Pulling from library/node
743f2d6c1f65: Pull complete
89252b028f01: Pull complete
204dbfab402e: Pull complete
5a6179c77f5e: Pull complete
Digest: sha256:65142740ff7461aed0a83b1a903c73017667bcff3b32277735e2d26db0a440bc
Status: Downloaded newer image for node:slim
[job docker.cwl] /tmp/tmpxjRzyZ$ docker \
    run \
    -i \
    --volume=/tmp/tmpxjRzyZ:/var/spool/cwl:rw \
    --volume=/tmp/tmpitgexZ:/tmp:rw \
    --volume=/home/ccllab/hello.js:/var/lib/cwl/stgd98f5cef-d95b-47fe-b09e-3370fbff90fb/hello.js:ro \
    --workdir=/var/spool/cwl \
    --read-only=true \
    --user=0:0 \
    --rm \
    --env=TMPDIR=/tmp \
    --env=HOME=/var/spool/cwl \
    node:slim \
    node \
    /var/lib/cwl/stgd98f5cef-d95b-47fe-b09e-3370fbff90fb/hello.js
Hello World
[job docker.cwl] completed success
{}
Final process status is success
```