



Unleash Innovation

軟體部屬架構 與 Container

PLED/CID1 HungWei Chiu

Agenda

- 軟體部屬架構探討
- Container (容器化的介紹)
- 示範
 - 以 Docker Container 為範例示範
 - 搭配前述的 GitHub Repo

試想系上作業

- 需要完成一個作業來利用資料庫，其中需要下列元件
 - 有一個網頁，可以讓使用者操作
 - 有一個伺服器，可以連接資料庫
 - 一個資料庫可以儲存使用者的資料
- 如何將這些請求轉化成實際上佈署的面貌？



系統架構

- 真實上線面對使用者的專案，都會希望能夠追求高可用性
- 避免單點故障問題
- 畢竟沒有人會希望一個元件壞掉就使得所有使用者不能用
- 以前述的範例做來看，要如何設計整體架構來達到這個效果？

ChatGPT 4o

- 這類型的架構問題並不是第一天出現，網路上已經有各式各樣的資源，自然而然大家都可以透過 AI 去詢問得到一些想法與答案
- 以 GPT 4O 為範例去詢問這個問題，到底會得到什麼答案？

如果今天要撰寫一個學校作業，需要有一個前端，一個後端，也需要有一個資料庫

然後被要求要可以滿足高可用性，也要避免單點故障

我應該要怎麼設計與規劃我的部署模型

ChatGPT 4o

- 得到的答案包含
- 幫你規畫每個元件要用什麼專案
- 高可用性
 - 單一節點/元件掛點，服務藥可以運作
- 避免單點故障
 - 所有元件都需要有備援或 failover 機制
- 同時幫忙勾勒一個架構圖

- 前端：React、Vue 或純 HTML/CSS/JS。
- 後端：Node.js/Express、Flask、Django 等皆可。
- 資料庫：MySQL、PostgreSQL、MongoDB 等。
- 高可用性：即使某一節點或元件掛掉，服務仍可運作。
- 避免單點故障：所有元件都需有備援或 failover 機制。

💡 部署架構規劃（簡易版）

可以用下列模型來做設計：



ChatGPT 4o

- 繼續追問下去可以得到更多細節
- 如果 GPT 已經可以回答這些問題了，那我們還需要學習什麼？
- Q:
 - Load Balancer 的用途是什麼？
 - 為什麼使用者到 Frontend Server 需要 Load Balancer，但是 Frontend Server 到 Backend 不需要 Load Balancer？
 - Database 主從是什麼意思？
 - ...etc

- 前端：React、Vue 或純 HTML/CSS/JS。
- 後端：Node.js/Express、Flask、Django 等皆可。
- 資料庫：MySQL、PostgreSQL、MongoDB 等。
- 高可用性：即使某一節點或元件掛掉，服務仍可運作。
- 避免單點故障：所有元件都需有備援或 failover 機制。

💡 部署架構規劃（簡易版）

可以用下列模型來做設計：

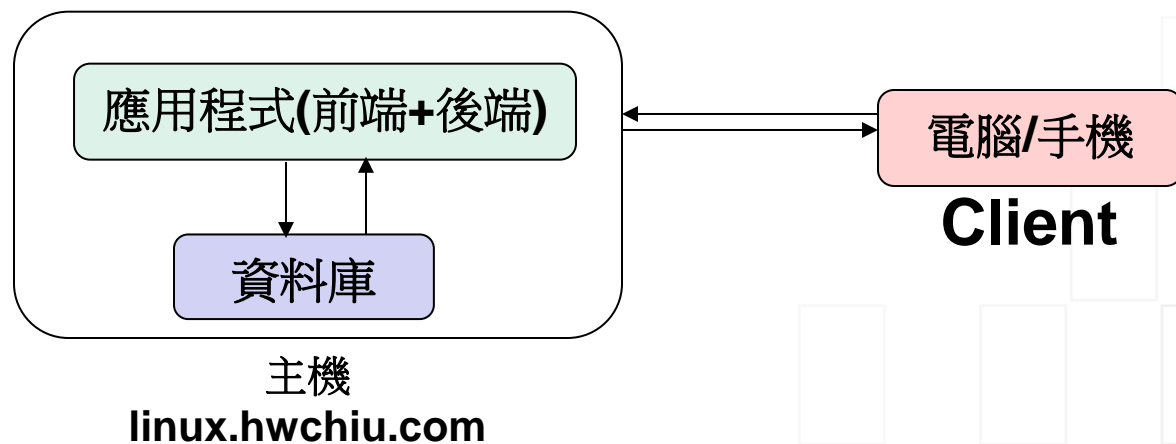


本課程期許

- 你會學會的
 - 如何思考與規劃一個應用程式佈署模型
 - 探討不同模型，並且比較不同模型的優缺點
 - 隨者架構複雜化，團隊要如何合作完成整體的開發與測試
 - **Container** 的概念，以及如何整合到上述流程
- 你不會學會的
 - 每個元件應該要用什麼程式語言，要用什麼專案
 - 每個元件的實際設定方式
 - **Container** 的指令教學

模型一

- 最直覺也是單人作業的開發方式
- 將前端與後端，撰寫到同一個應用程式中，並且與資料庫佈署到同一台機器中
- 特色
 - 一台機器就可以佈署所有服務
 - 前端與後端採用相同語言
- 缺點
 - 機器壞了，就代表三個元件一起壞
 - 可能性
 - 網路有問題
 - 硬碟有問題
 - 電源有問題



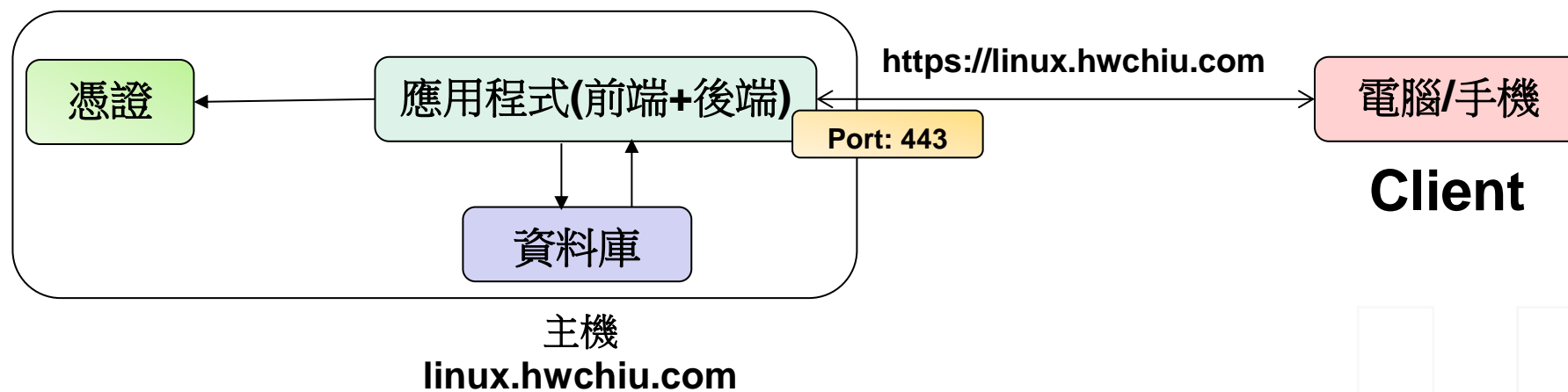
模型一 - 佈署方式

- 當網頁準備完畢後，會有兩種存取方式
 - HTTP
 - HTTPS
- 針對作業練習，使用 **HTTP** 都是可以接受的
- 但是工作實務上，所有對外服務都一定要有 **HTTPS** 來保護連線資訊
 - 為了達到 **HTTPS**，我們需要準備一些憑證來辨識彼此
 - **HTTPS** 的運作原理與憑證概念，請自行研究

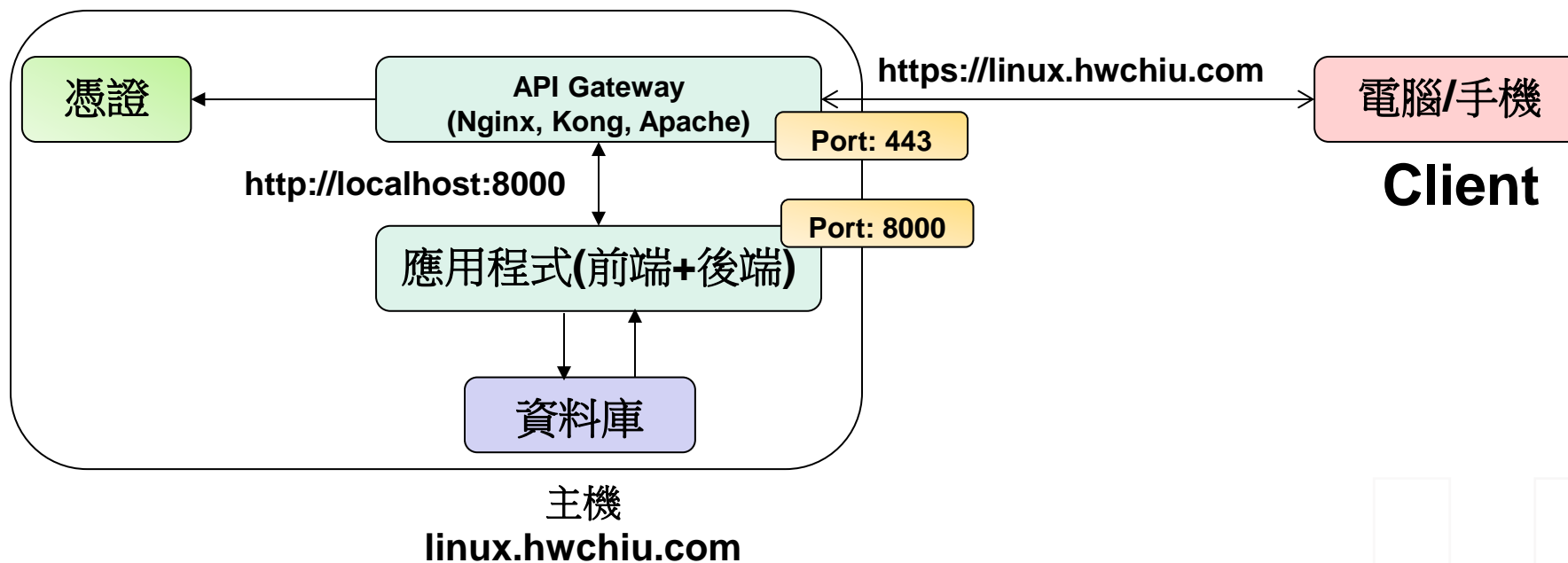
模型一 - 佈署方式

- 這個需求有兩種實作方法
- 你很熟悉憑證的所有需求與處理
 - 於自己的前端內導入憑證的處理與管理機制
 - 自己的前端會需要監聽 **port 443** 來處理 **HTTPS** 連線
- 不熟悉怎麼管理，因此決定導入開源專案當你應用程式的入口
 - 由該開源專案扮演一個 **API Gateway** 來處理 **HTTPS** 連線
 - 譬如 Nginx, Apache, HAProxy, Kong
 - 將收到的連線再轉發到你的應用程式
 - 由該 **API Gateway** 去監聽 **port 443** 來處理連線
 - 這意味你的應用程式就要改用別的 **Port** 來處理

模型一 - 佈署方式



模型一 - 佈署方式



模型一 - 佈署方式

- 第一個架構下，前端與後端需要處理
 - HTTPS 的實作
 - 相關憑證的處理
 - 憑證到期要可以自動延展
 - 服務需要重新載入檔案 會不會影響計有服務？

新聞

SSL/TLS憑證最長效期2029年將縮減成47天

避免長效期憑證遭駭客濫用，憑證產業論壇CA/Browser Forum宣布，截至2029年的SSL/TLS憑證效期調整計畫拍板定案，將由現行398天逐年縮短至47天，且每月需更新一次

文/ 林妍臻 | 2025-04-15 發表

讚 287

分享

模型一 - 佈署方式

- 第二個架構下，把 **HTTPS** 的實作轉嫁到 **API Gateway** 身上
 - 我們的前端只需要專注處理 **API** 的格式與資料
 - 為了避免 **Port** 衝突，我們的應用程式要使用不同的 **Port**
 - 8080
 - **API Gateway** 將連線轉發給我們開發的前端服務
 - **Nginx/Apache** 設定相關憑證，並且處理 **HTTPS** 的連線，最後將連線轉發到後面的應用
- 各自的優缺點？
 - 要不要引入更多元件來額外處理？
 - **API Gateway** 能不能有更多用途？
 - 殺雞焉用牛刀？
 - 沒有標準答案，就很看需求跟團隊能力

模型一 - 不同使用者

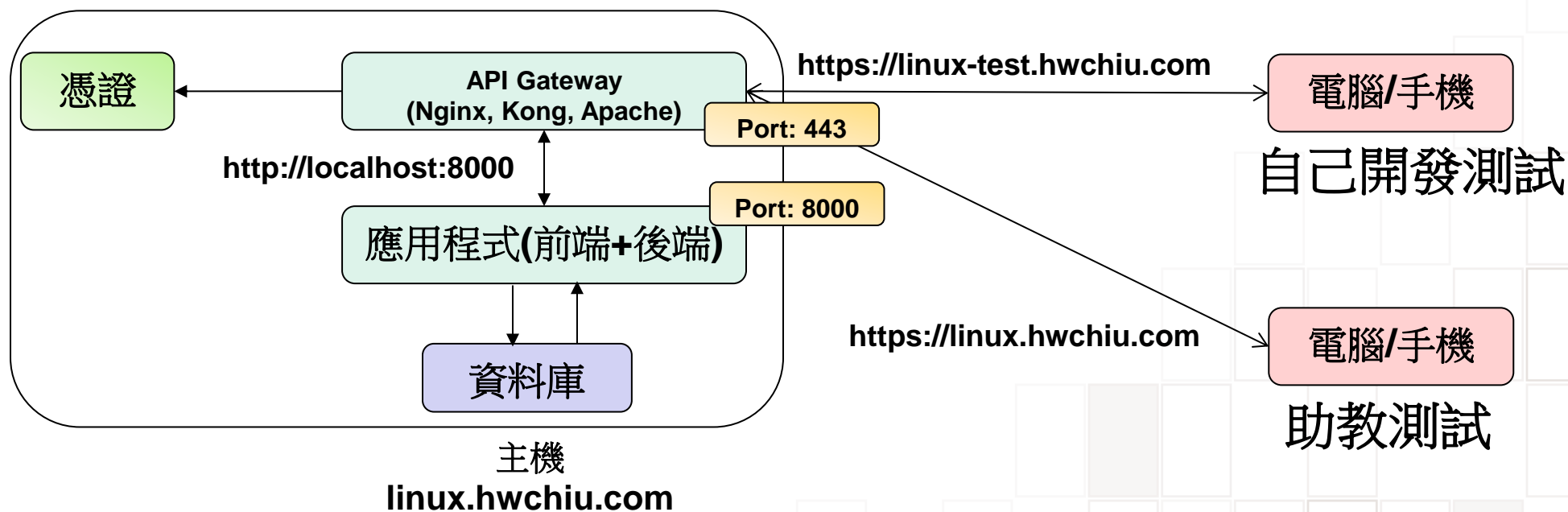
- 以模型一來說，程式開發者必須要完成下列需求
- 第一種
 - 研究合適的框架與語言，一次搞定前端與後端，並且掌握編譯與運行過程
 - 掌握憑證的準備與整合
 - 選擇一個合適的資料庫，並且運行資料庫
- 第二種
 - 研究合適的框架與語言，一次搞定前端與後端，並且掌握編譯與運行過程
 - 選擇一個合適的資料庫，並且運行資料庫
 - 選擇一個合適的 **API Gateway**，並且學會設定與運行

模型一 - 不同使用者

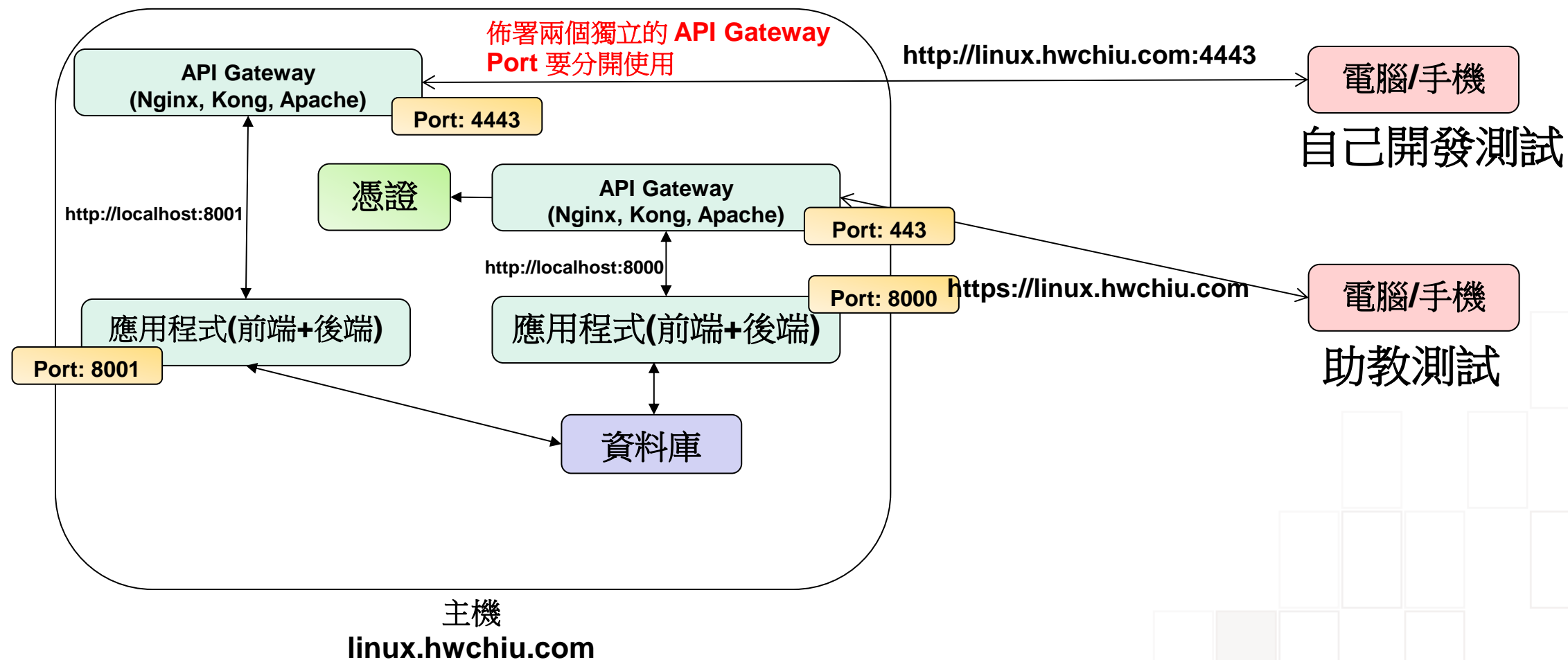
- 為了完成這個作業，你必須要開發與測試，最後還要 **Demo** 給助教看
 - 因此會有
 - 開發環境
 - **Demo** 環境
- 假設你只有一台機器，你要如何同時支援 開發環境的測試與真正給助教的 **Demo** 環境
 - 服務會不會衝突？
 - 要怎麼運行多個？
- 假設有兩台環境，你要如何確保你的 **Demo** 環境與你的開發環境是一致的(程式碼版本/各種設定)
 - **Config** 一致
 - 資料庫要安裝多次，還要確保設定符合環境

模型一 - 不同使用者

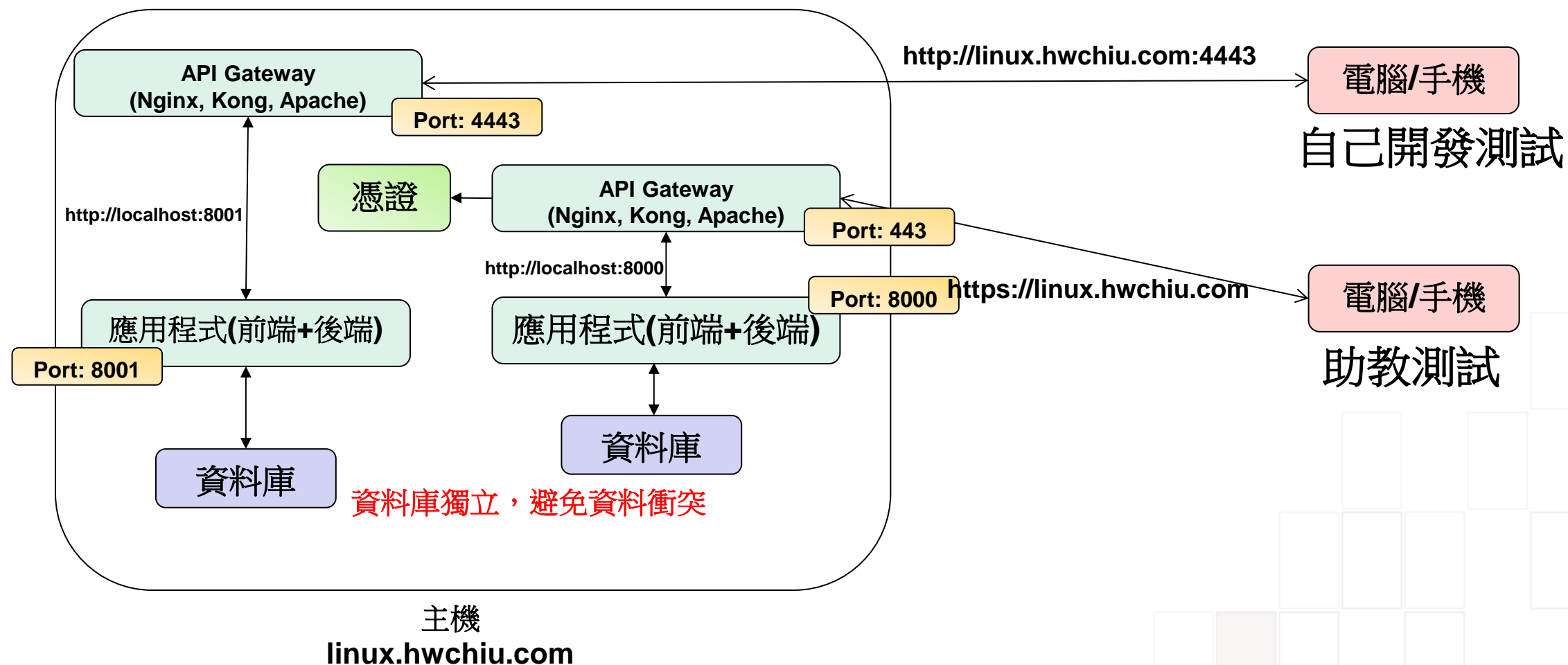
- 我們希望一台電腦可以讓你跟助教連接到同一台機器，但是分別存取到不同的服務
 - 自行開發跟長期給助教 Demo 的是不一樣的
 - 又不希望平常開發的直接影響到給助教 Demo 評分用的
- 該怎麼辦？



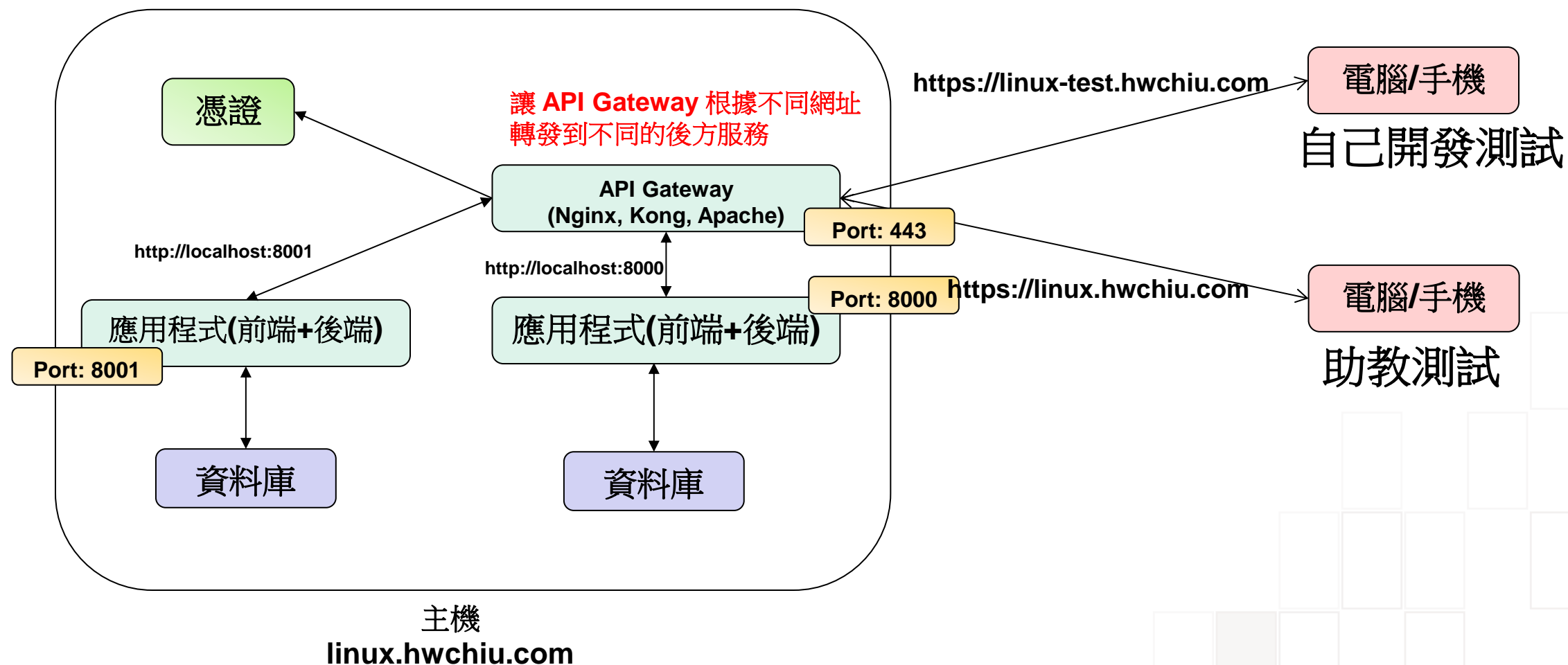
模型一 - 不同使用者



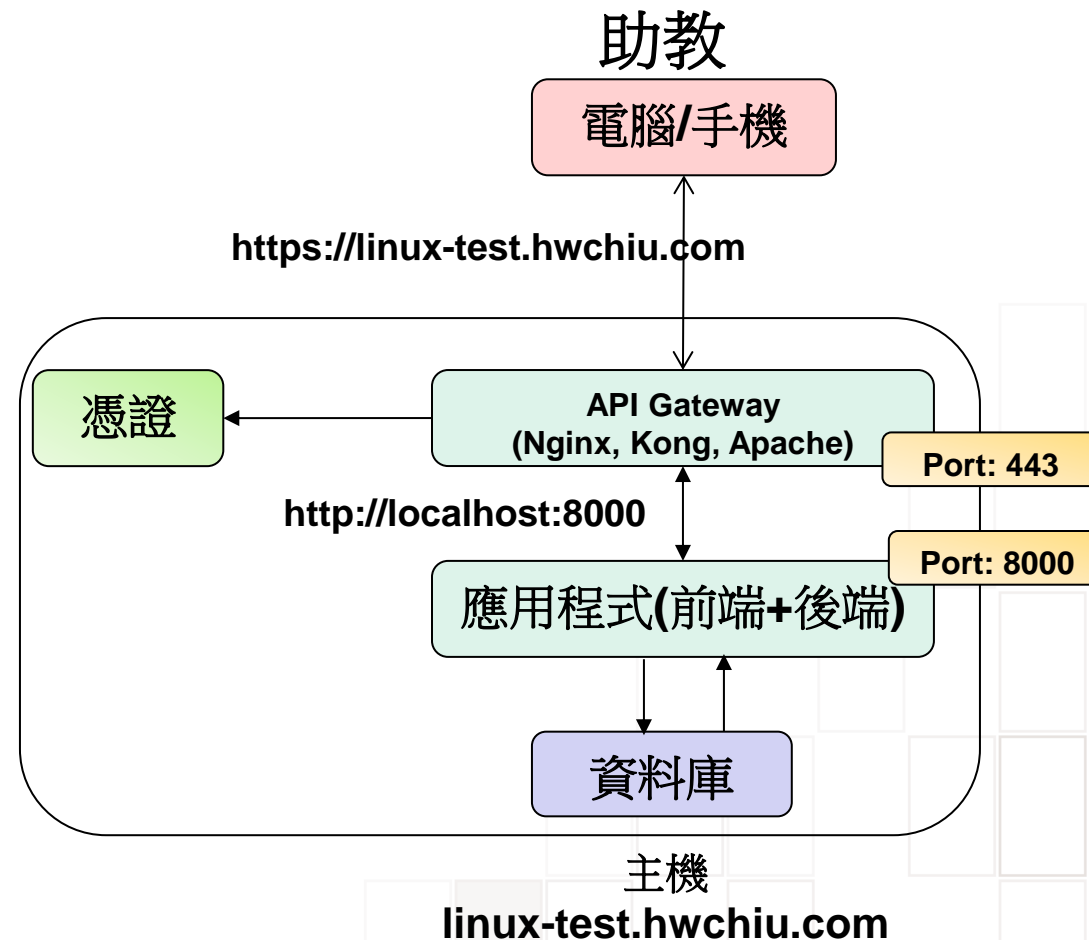
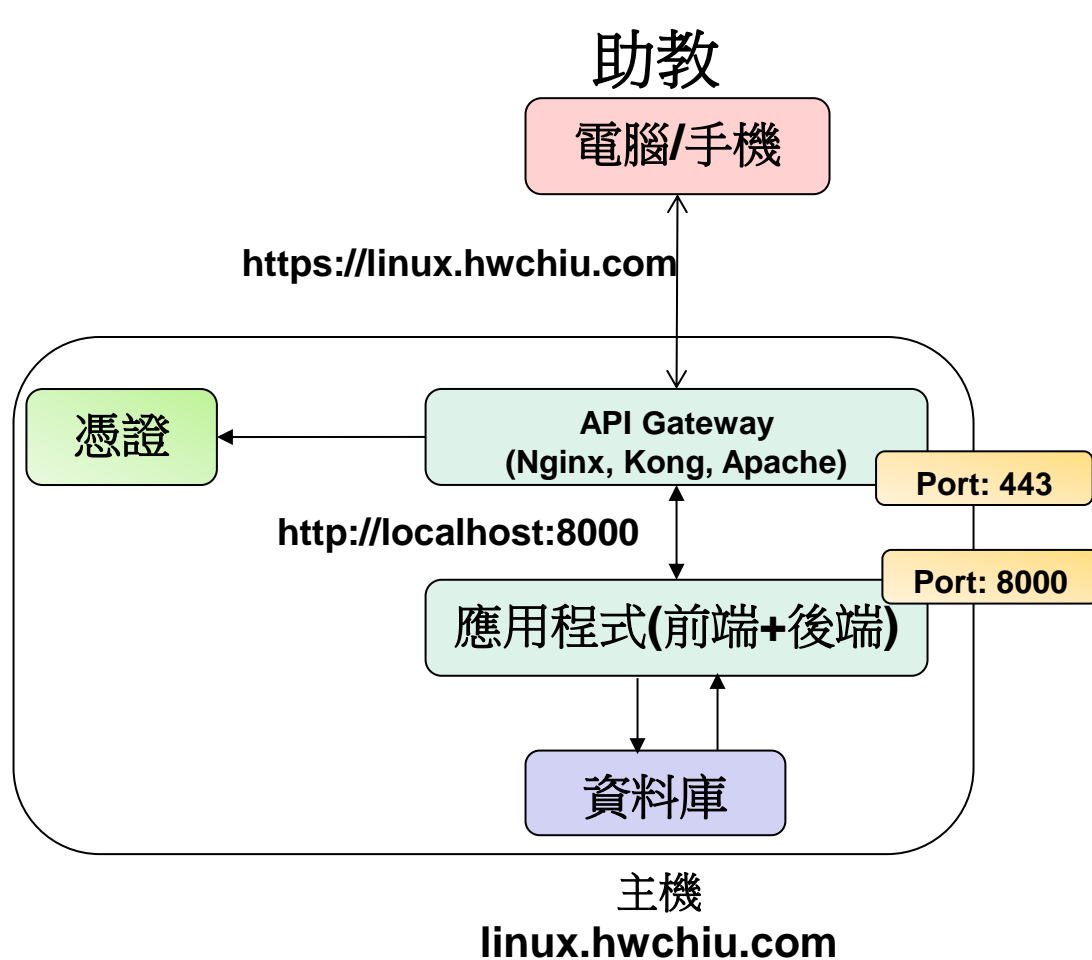
模型一 - 不同使用者



模型一 - 不同使用者



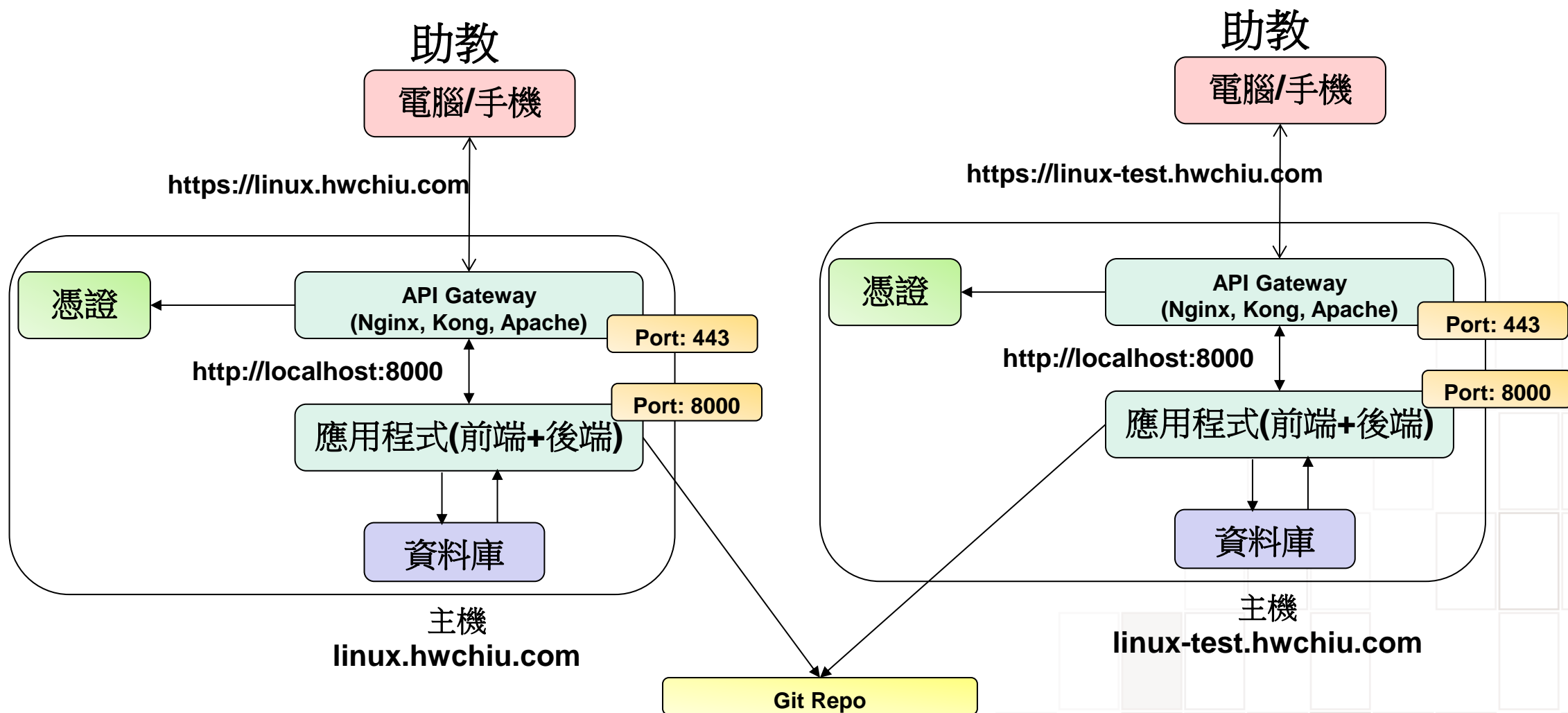
模型一 - 不同使用者



模型一 - 不同使用者

- 採用兩台電腦的方式，可以讓架構單純，有更多需求就是複製更多一樣環境的電腦
 - 避免過多複雜的環境以及特異需求
- 但是只要電腦愈來愈多，就會有程式運行與佈署的版本控管問題
- 解法
 - 仰賴 **Git** 專案管理，來確保程式碼是有版本控制，同時去檢查每個環境佈署的程式碼版本

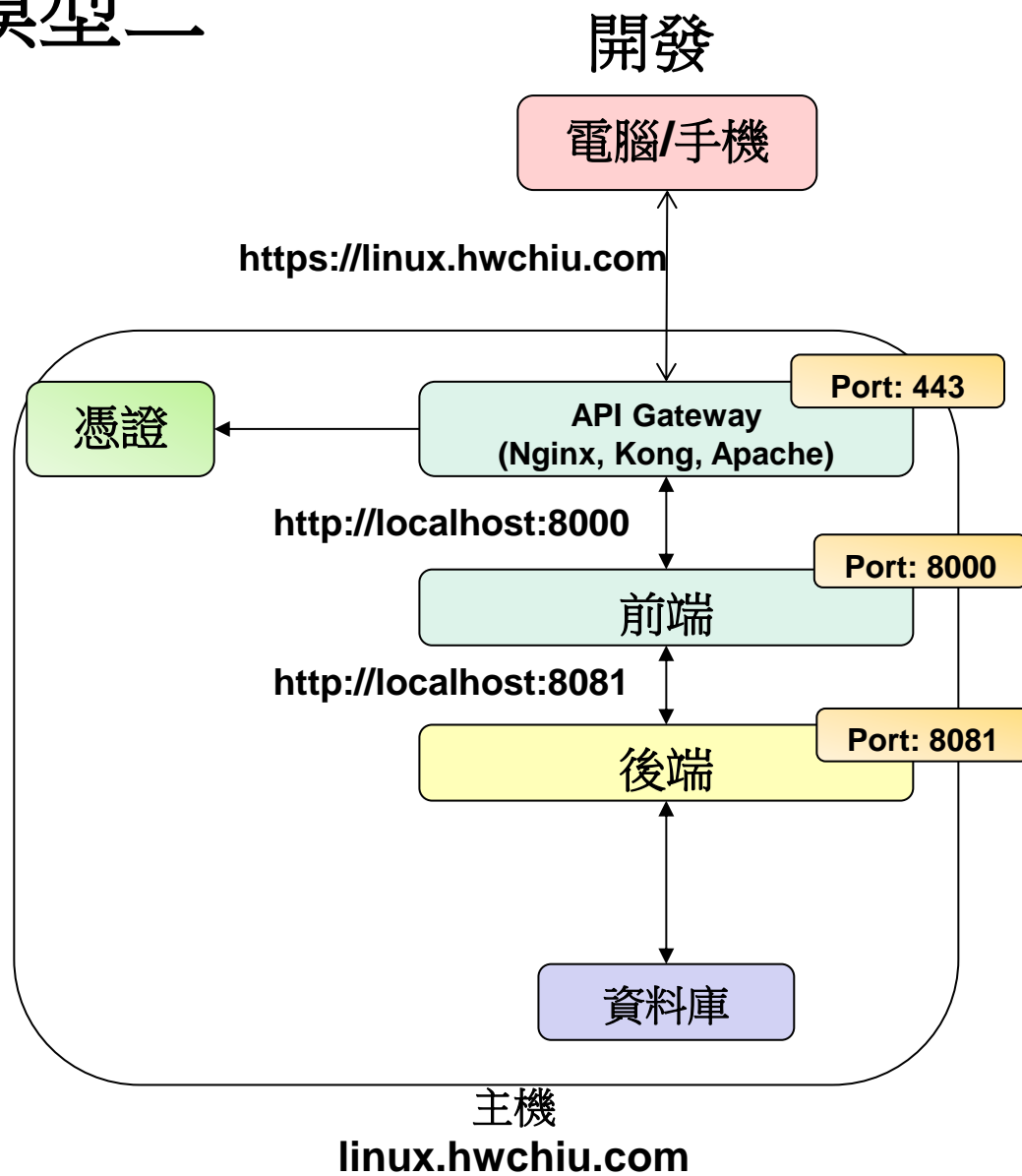
模型一 - 不同使用者



模型二

- 模型一過於簡單，同時要求前端與後端採用相同的程式語言與框架
- 因此模型二就是將其分離
 - 前端與後端可以獨立開發與測試
 - 採用不同的程式語言，不同人撰寫
- 基本上與模型一相似，唯一差異是
 - 前後端的開發分離
 - 多人合作下可以根據專長進行分工，有人專心寫前端，有人專心寫後端

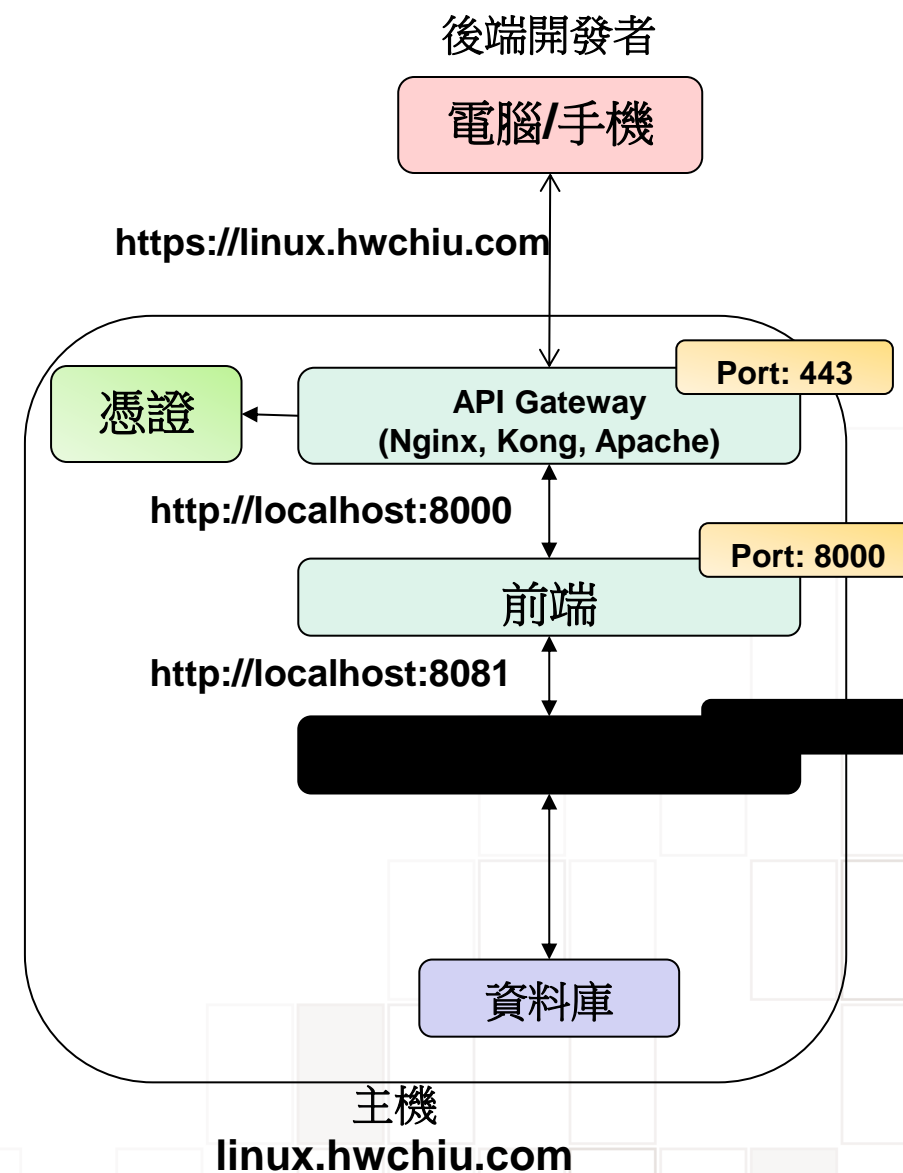
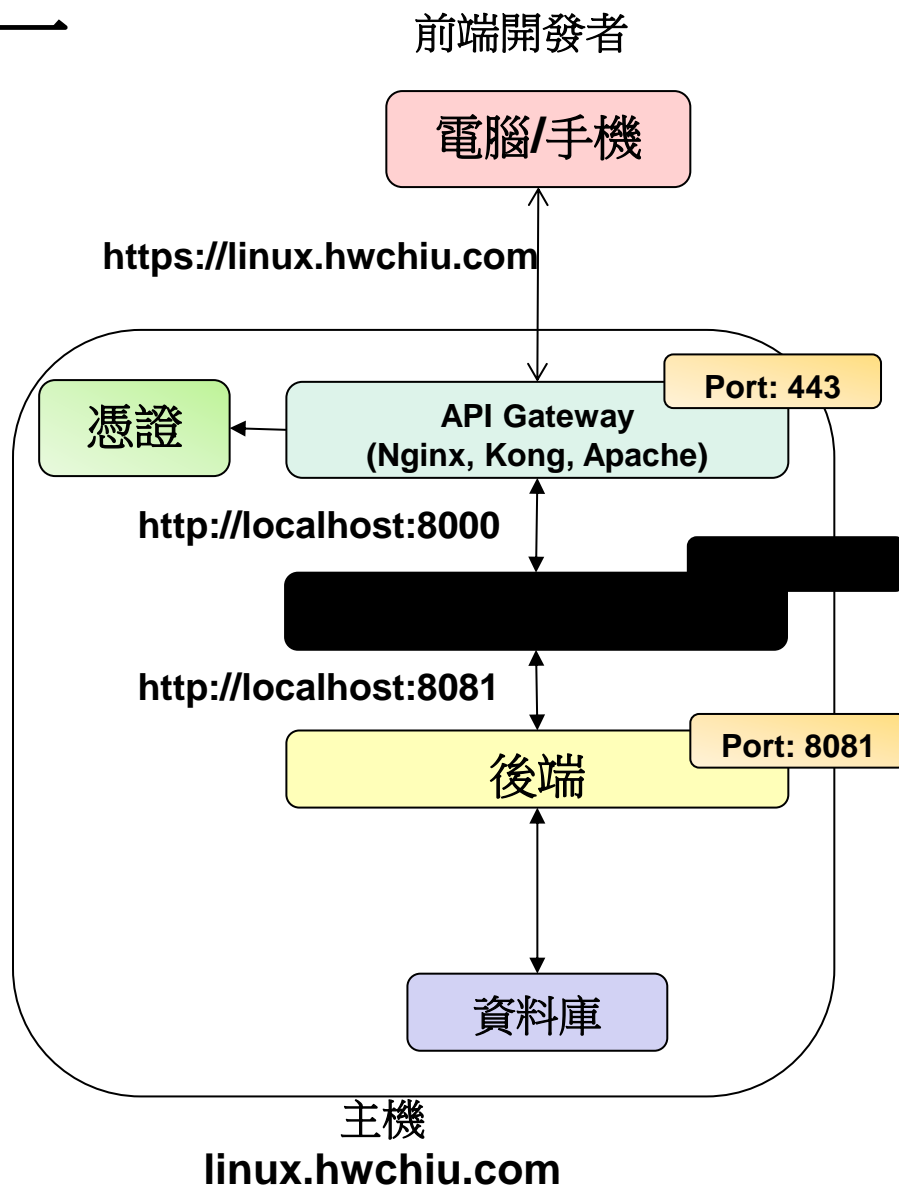
模型二



模型二 - 測試問題

- 常見問題
 - 一個完整的測試需要有完整的系統
 - 使用者透過網頁操作資料庫
- 撰寫前端的人要如何於自己的環境進行測試？
 - 要學會後端的 (編譯+運行+設定) 加上 資料庫的安裝與設定
- 撰寫後端的人要如何於自己的環境進行測試？
 - 要學會前端的 (編譯+運行+設定) 加上 資料庫的安裝與設定
- 另外，資料庫誰負責？
 - 前端？後端？第三人？

模型二



模型二 - 測試問題

- 一人團隊，一條龍服務
 - 前端，後端，資料庫，**API Gateway** 都是由一個人搞定全部
 - 因此就沒有上述的分工問題
- 多人團隊，互相協作
 - 分工進行，開發效率更好
 - 測試與整合上則是會有更多問題
 - 不同人寫程式的習慣與技巧都不同
 - 架設環境測試
- 普遍作業都是一人團隊，然而工作實務上都是多人團隊的開發，小到 2 人大到數十人都會有，甚至百人大專案。因此團隊內如何有效的協作一直以來都是所有團隊追求的目標

模型二 – 測試問題

- 簡易解法
- 程式碼共享: 透過 **Git** 共享
- 指令說明與解釋: 透過 **README** 文件解釋
 - 撰寫良好的文件，描述所有環境需求
- 環境架設與操作: 撰寫良好的自動化程式去安裝與設定
 - 透過一些腳本將操作給簡單化，減少人為錯誤
 - 讓所有人都能夠有辦法重現開發環境
- **GitHub** 上如同有滿滿的 **README.md** (markdown 語法)
- 自動化腳本
 - 各種 **shell script**
 - **Makefile**
 - 容器化環境

模型二 – 測試問題

I just want it built and run it on my cluster

First, point the `Makefile` to the Docker registry of your choice:

```
export DOCKER_PREFIX=index.docker.io/myrepo
export DOCKER_TAG=mybuild
```



Then build the manifests and images:

```
make && make push && make manifests
```



Note: If you see failures related to fetching some modules, try increasing bazel's timeout with:

```
export PULLER_TIMEOUT=10000
```



模型三

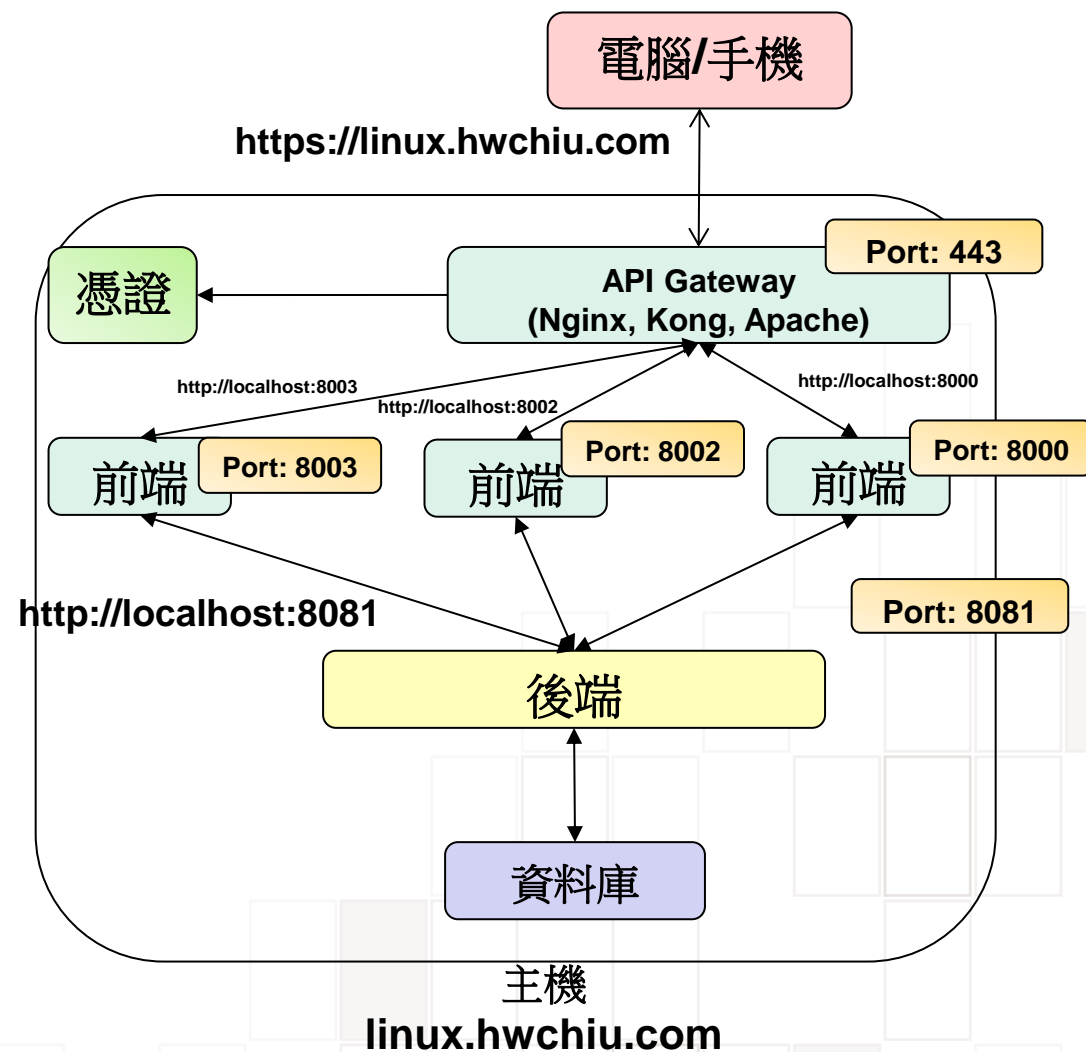
- 前述的模型都還是專注於單一機器的環境，該機器壞掉的話，所有的服務就會一起壞掉
- 因此架構要追求高可用性，因此嘗試佈署看看多個副本
- 不過到底哪個元件要變成多副本？
 - 前端
 - 後端
 - 資料庫
- 理論上應該全部都要？
- 先從前端多副本看看

模型三

- 多副本的情況下，到底需要幾台機器？
 - 看需求與瓶頸
- 如果今天多副本想要解決的是本來的效能問題，也許一台機器就可以
 - 譬如我的應用程式只能吃一個 **CPU**，多副本就可以利用多 **CPU**
- 如果目的則是希望可以跨機器來提供高可用性，那就要多台機器
- 來檢視一下這兩種情境下的佈署情況

模型三 – 佈署模型一

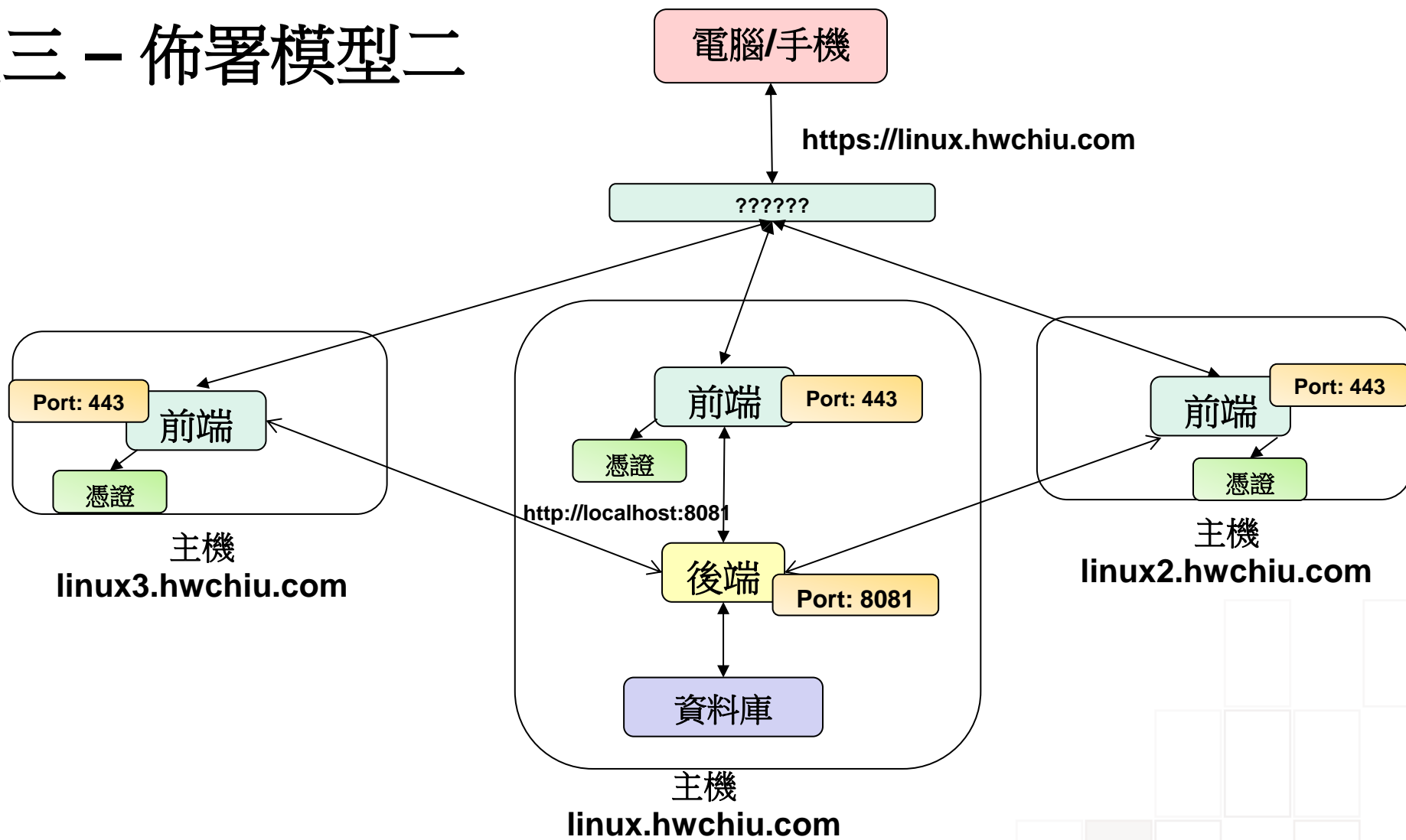
- 透過 API Gateway 去幫忙轉發多副本的流量
 - 有不同的演算法去決定流量要給後續的誰
- 這些前端應用都在同一台上，因此需要使用不同的 Port
 - 這意味 API Gateway 上面的設定就要很多種，根據有多少服務就設定多少



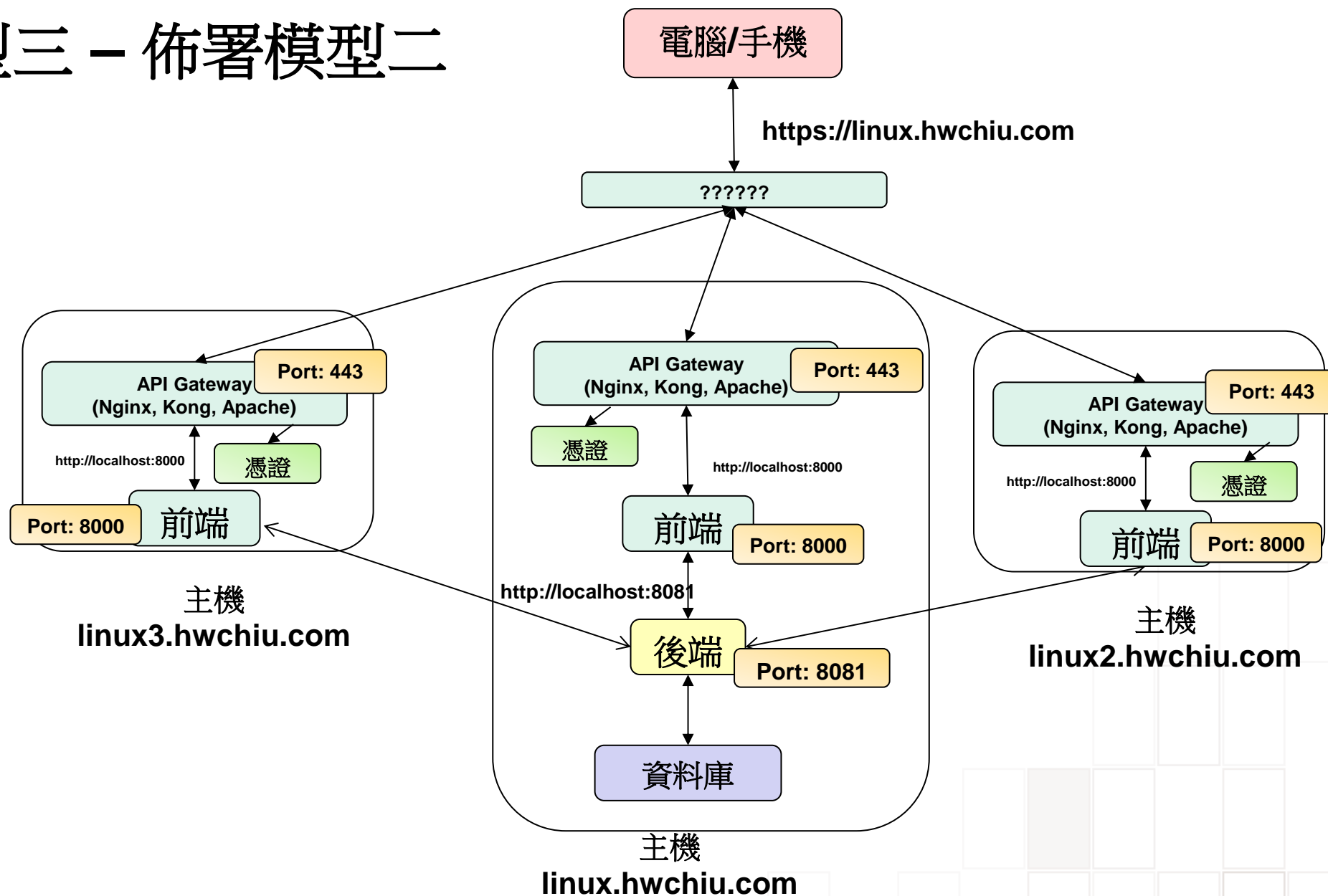
模型三 – 佈署模型二

- 前述的模型還是在同一台機器，因此只能解決單一機器上面的單一 **Process** 掛掉的問題，若機器壞掉則還是全部壞掉
- 因此我們需要嘗試多機器佈署，確保一台機器壞掉還是有可以用的前端去處理使用者的請求
- 前端的請求牽扯到 **HTTPS**，這個 **HTTPS** 該怎麼處理？
- 每台機器上面佈署
 - 單純前端
 - Nginx + 前端

模型三 - 佈署模型二



模型三 – 佈署模型二



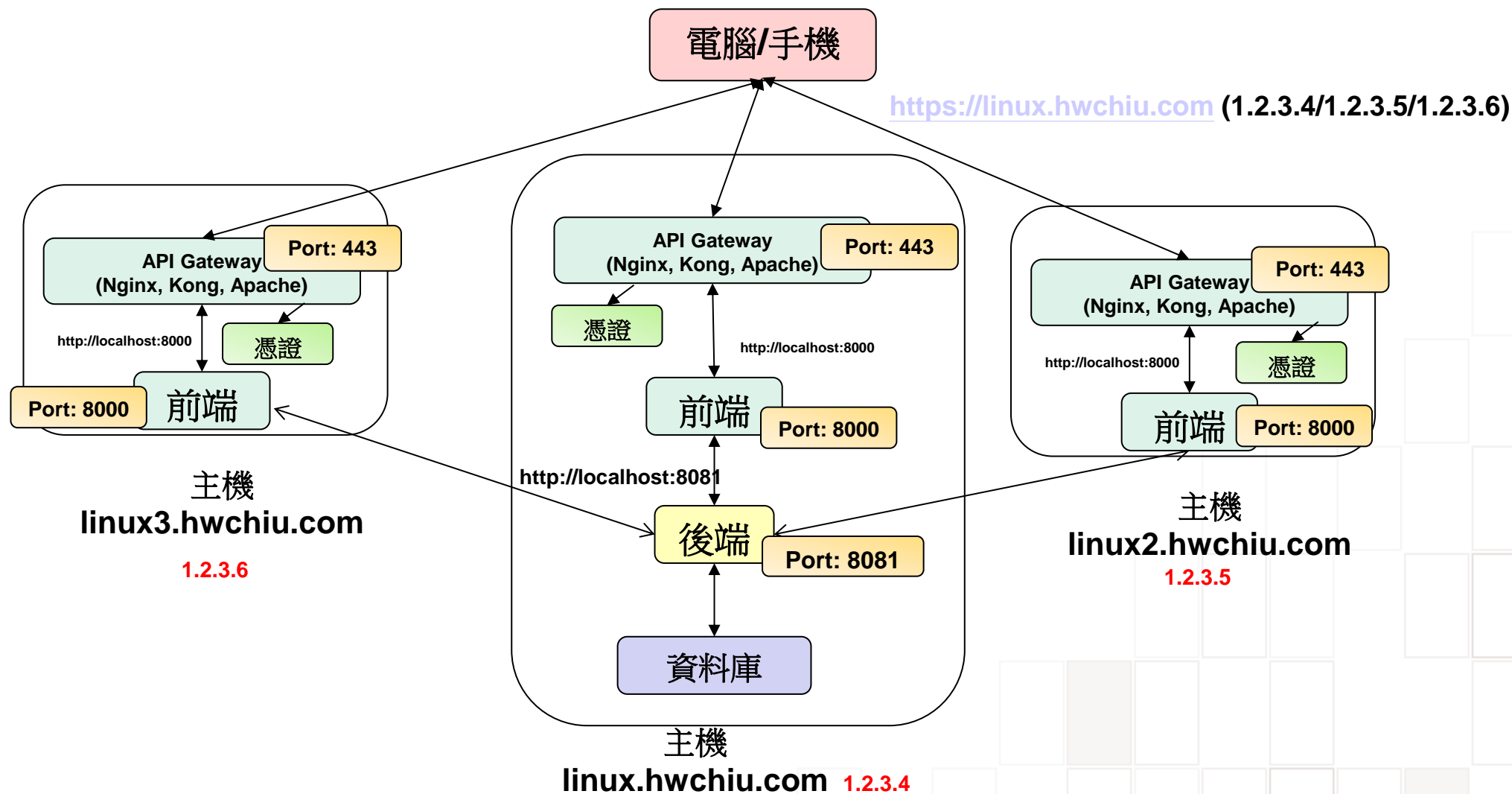
模型三 – 佈署模型二

- 上述兩種模型看起來好像可以分散，但是又導致其他問題
- 使用者到底是連接到哪一個前端/API Gateway
 - 憑證這樣就會有每個地方都要有，但是憑證又牽扯到 Hostname
 - HTTPS 的憑證都跟名稱有關
 - 因此一個 linux.hwchiu.com 要三台都可以連接得到
- 因為後端跟資料庫都沒有多副本，所以其他的前端要怎麼打回去給後端？
 - 這些後端不能用 localhost 存取，需要額外開啟網路服務去給存取

模型三 – 佈署模型二

- 第一種解法是透過 DNS
- 過往我們都習慣 一個 DNS 解析到一個 IP，譬如
- linux.hwchiu.com
 - 1.2.3.4
- 我們可以讓 DNS 解析到多個 IP，這樣使用者就有機會被導向不同的機器
 - 1.2.3.4
 - 1.2.3.5
 - 1.2.3.6

模型三 – 佈署模型二

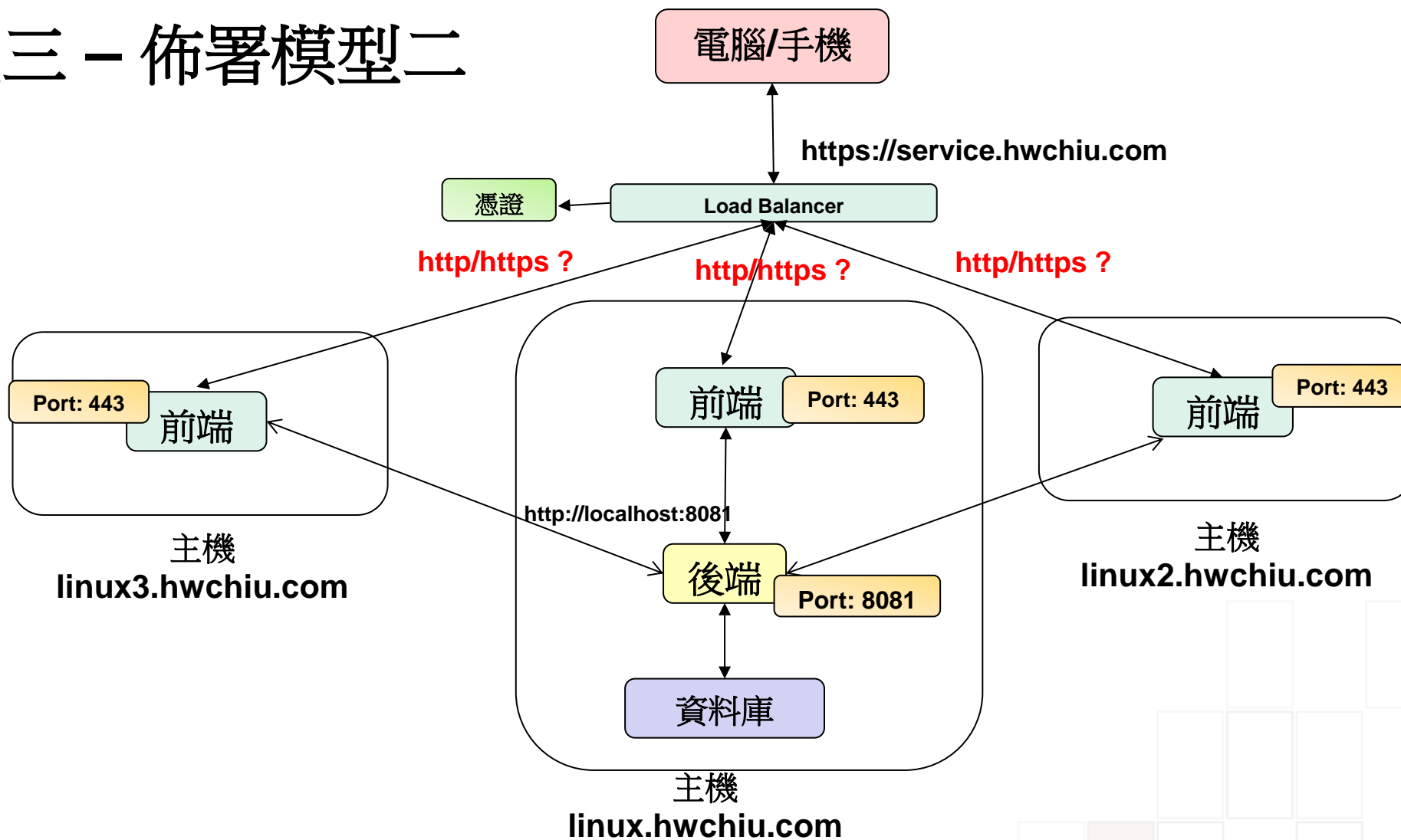


模型三 – 佈署模型二

- 第二種解法
- 佈署一個額外的服務，可以想成把 **API Gateway** 給抽出來，變成獨立的元件
 - 有時候我們也會將這個元件稱為 **Load Balancer**
- 該 **Load Balancer** 可以處理
 - 多副本的演算法分配
 - **HTTPS** 的憑證處理



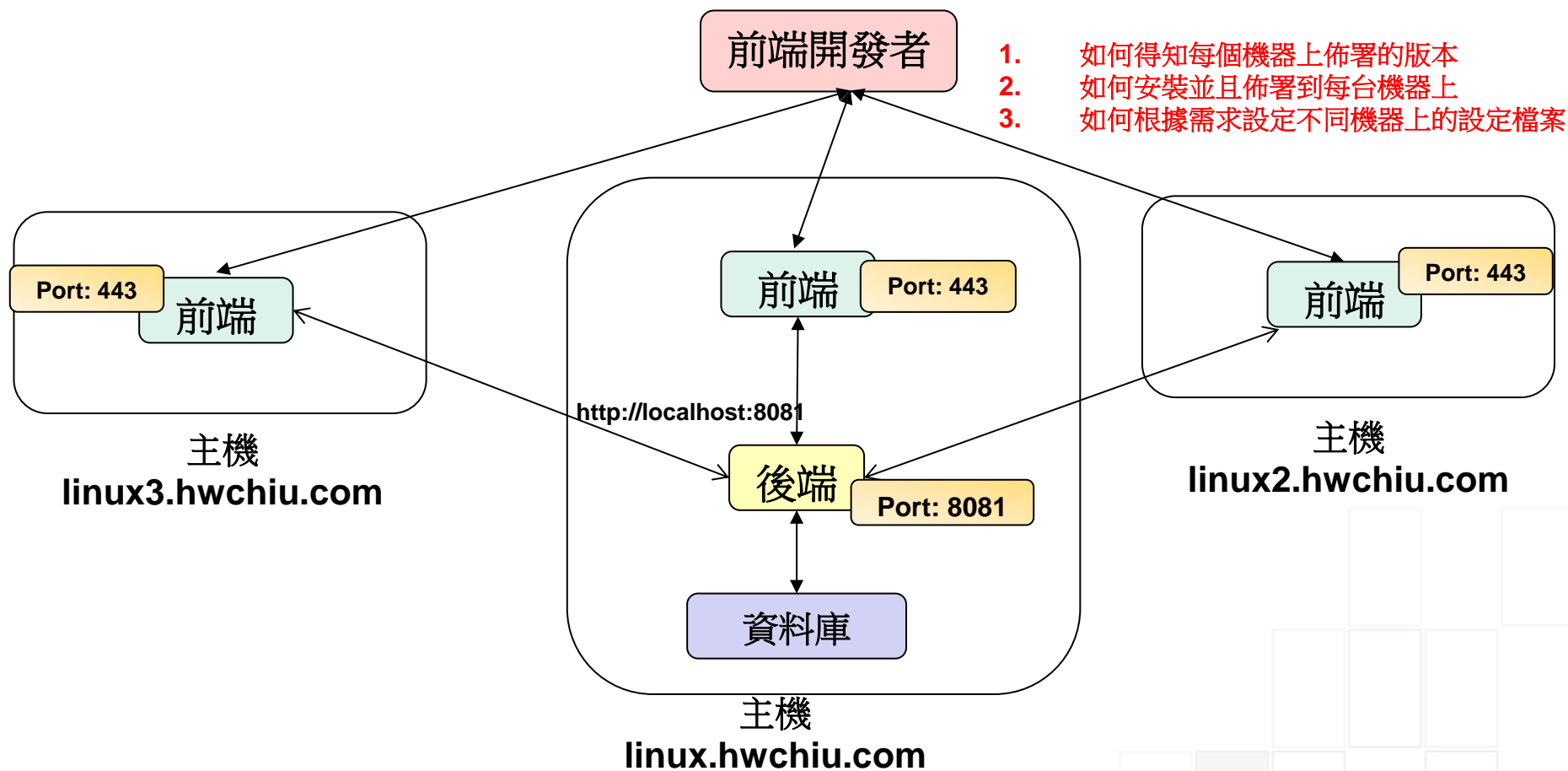
模型三 – 佈署模型二



模型三 – 痛點

- 當前端應用程式有多副本的時候，下一個痛點就是如何佈署這些副本
- 我們希望達到
 - 多個機器上的副本版本一樣
 - 簡化佈署多個機器的流程
 - 每台都手動 ssh 連接?
 - 如果有 20 台機器該怎麼辦?
 - ◆ 手動執行 20 次?
 - ◆ 需要程式處理
 - 隨者開發迭代，我們需要知道每個環境佈署的版本
 - 如果每次都要靠一些運作行為來猜版本，人生會浪費太多時間
 - 必須要有一個統一且最好可以自動化的方式知道目前佈署的版本
 - 每台的設定檔案都要能夠妥善設定，避免大量的人為介入

模型三 – 佈署模型二



模型三 – 痛點

- 可能解法
- 佈署方式
 - 透過 **ssh**，每台進去下指令
 - 把指令寫好，複製貼上
 - 撰寫腳本，整合 **ssh** 與相關指令，自動進行
 - 使用 **Ansible** 等工具
- 版本控制
 - 程式碼內塞一個方式獲取版本
 - Curl <http://xxxx/version> ?
 - 檔案內放一個 **VERSION**?
 - 其他好方式?

已知問題

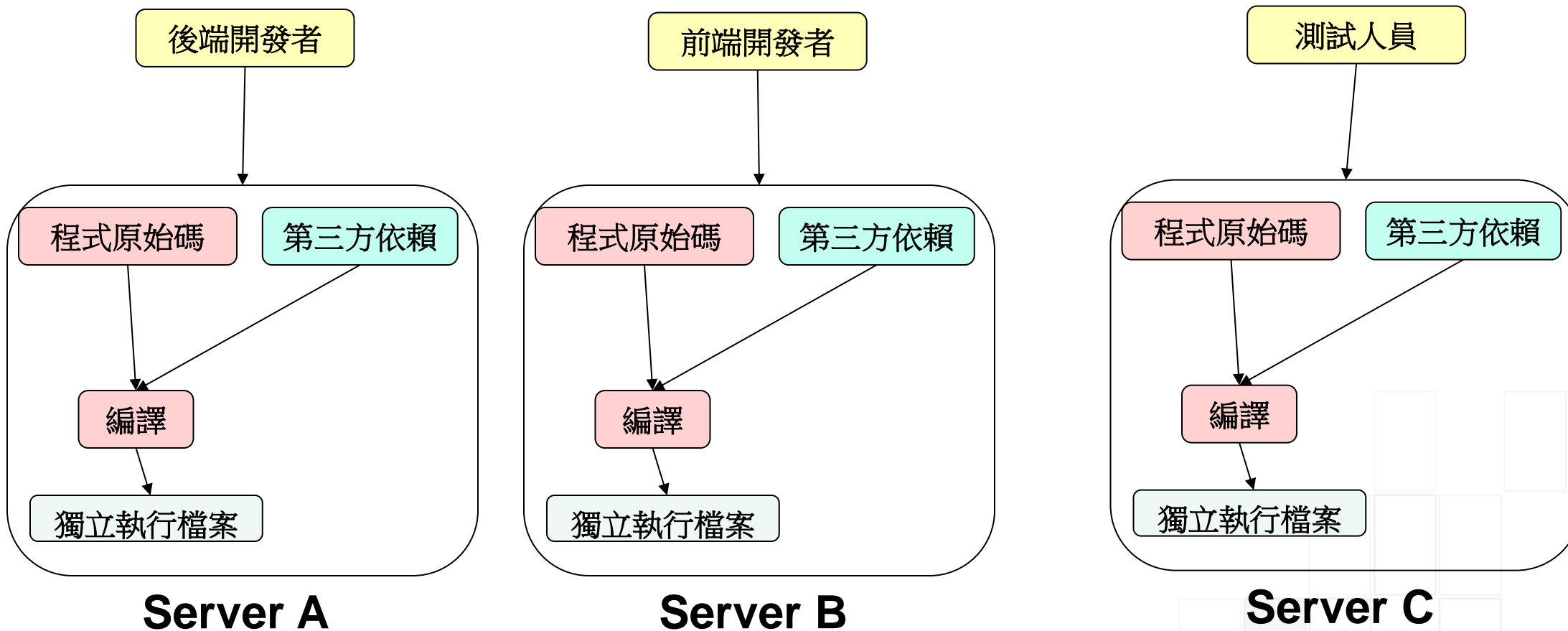
- 前後端分離且多人協作上，測試環境不好架設
 - 需要各自的專業知識與工具用法
- 高可用性(HA) 需求下，會需要佈署到多台機器
 - 如何簡化我們的佈署與設定行為？



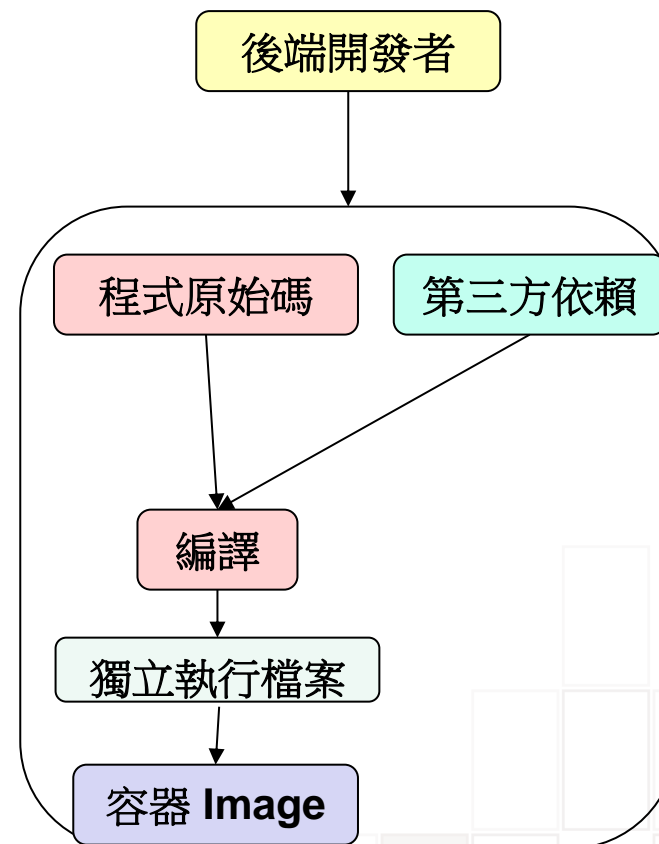
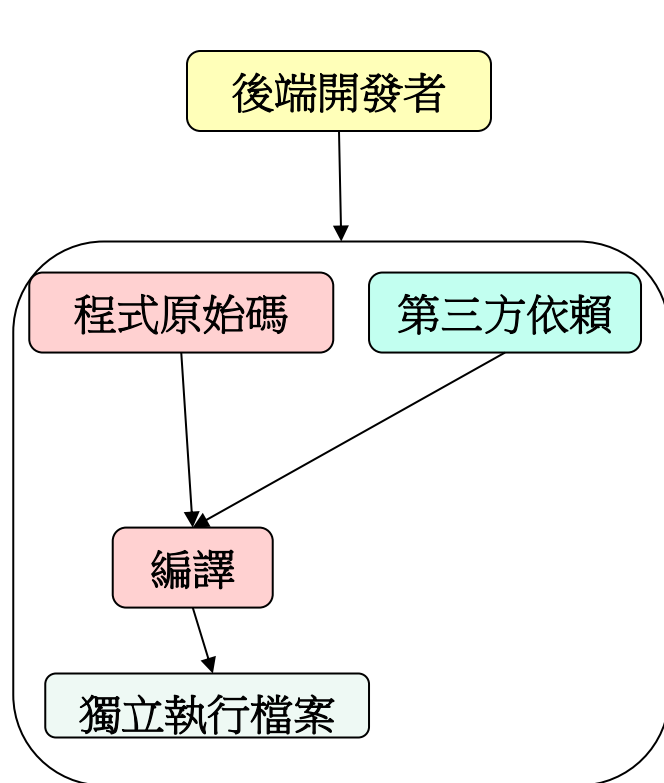
容器化

- 一個常見的做法就是容器(Container)化，將你的應用程式給打包成一個容器，透過打包容器來隱藏
 - 編譯，佈署細節的工具
- 只需要透過一樣的工具 (docker) 加上一個有共識的設定檔案
 - 設定檔案可以有很多格式，沒有標準 (YAML, Json, TOML, XML...etc)
- Docker 容器化的過程，會產生一個俗稱 Container Image 的產物，該產物可以有版本編號
 - 你可以幫你的應用程式打上編號: 0.0.1, 0.0.2

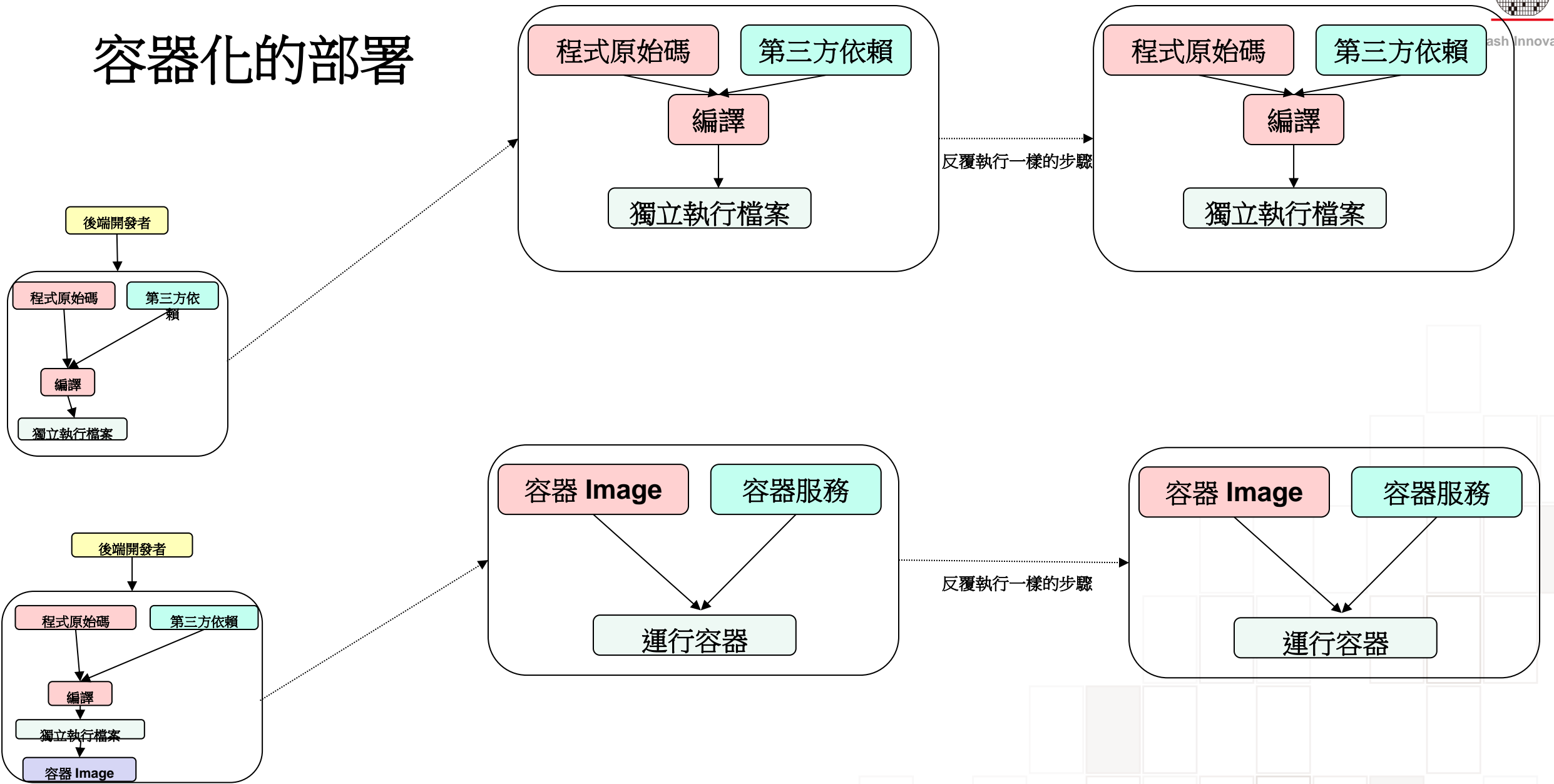
環境建置



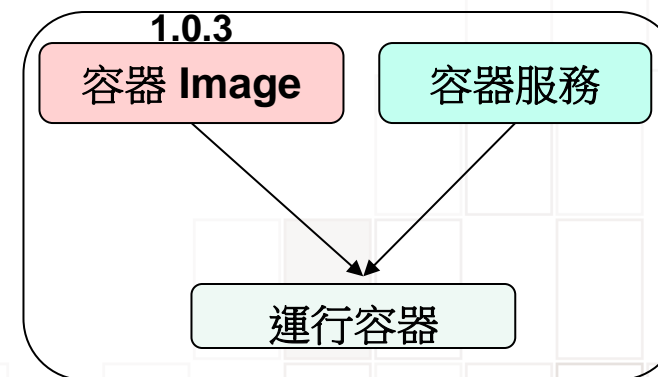
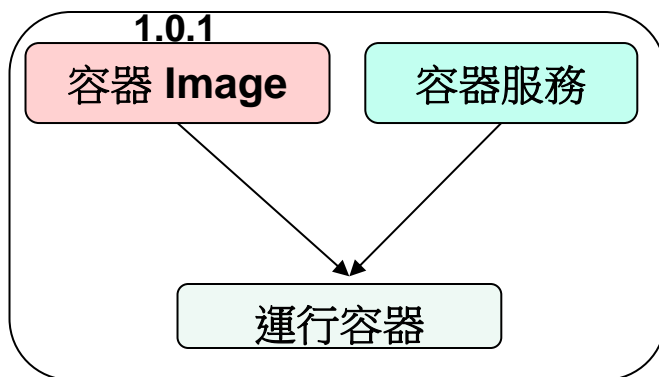
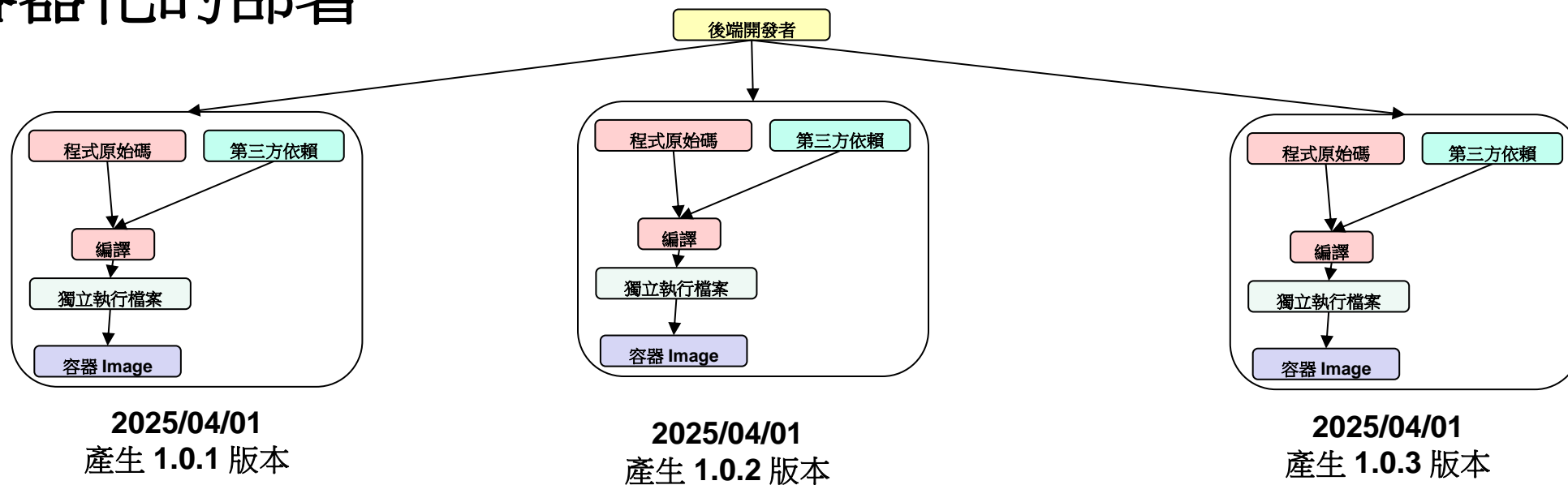
環境建置 容器化



容器化的部署



容器化的部署



容器化

- 版本的寫法
 - Major.Minor.Patch
- 推薦可參閱 <https://semver.org/> 學習
- 常見作法
 - 補上時間戳
 - 補上 Git Commit Hash
- 常見原則
 - Immutable image tag
 - 相同的 image 版本，不應該有兩個不同的內容
- 最怕的問題就是
 - 這個東西昨天可以動，今天不能動，結果發現其實裡面的程式碼不一樣

容器化

- **Nginx**

Quick reference

- Maintained by:
[the NGINX Docker Maintainers](#)
- Where to get help:
[the Docker Community Slack](#), [Server Fault](#), [Unix & Linux](#), or [Stack Overflow](#)

Supported tags and respective **Dockerfile** links

- `1.25.5`, `mainline`, `1`, `1.25`, `latest`, `1.25.5-bookworm`, `mainline-bookworm`, `1-bookworm`, `1.25-bookworm`, `bookworm`
- `1.25.5-perl`, `mainline-perl`, `1-perl`, `1.25-perl`, `perl`, `1.25.5-bookworm-perl`, `mainline-bookworm-perl`, `1-bookworm-perl`, `1.25-bookworm-perl`, `bookworm-perl`
- `1.25.5-otel`, `mainline-otel`, `1-otel`, `1.25-otel`, `otel`, `1.25.5-bookworm-otel`, `mainline-bookworm-otel`, `1-bookworm-otel`, `1.25-bookworm-otel`, `bookworm-otel`
- `1.25.5-alpine`, `mainline-alpine`, `1-alpine`, `1.25-alpine`, `alpine`, `1.25.5-alpine3.19`, `mainline-alpine3.19`, `1-alpine3.19`, `1.25-alpine3.19`, `alpine3.19`
- `1.25.5-alpine-perl`, `mainline-alpine-perl`, `1-alpine-perl`, `1.25-alpine-perl`, `alpine-perl`, `1.25.5-alpine3.19-perl`, `mainline-alpine3.19-perl`, `1-alpine3.19-perl`, `1.25-alpine3.19-perl`, `alpine3.19-perl`

容器化

Repository Tags

Compact

Expanded

Show Signatures

1 - 25 of 2391

<

>

Filter Tags...

TAG















LAST MODIFIED ↓

SECURITY SCAN

SIZE

EXPIRES

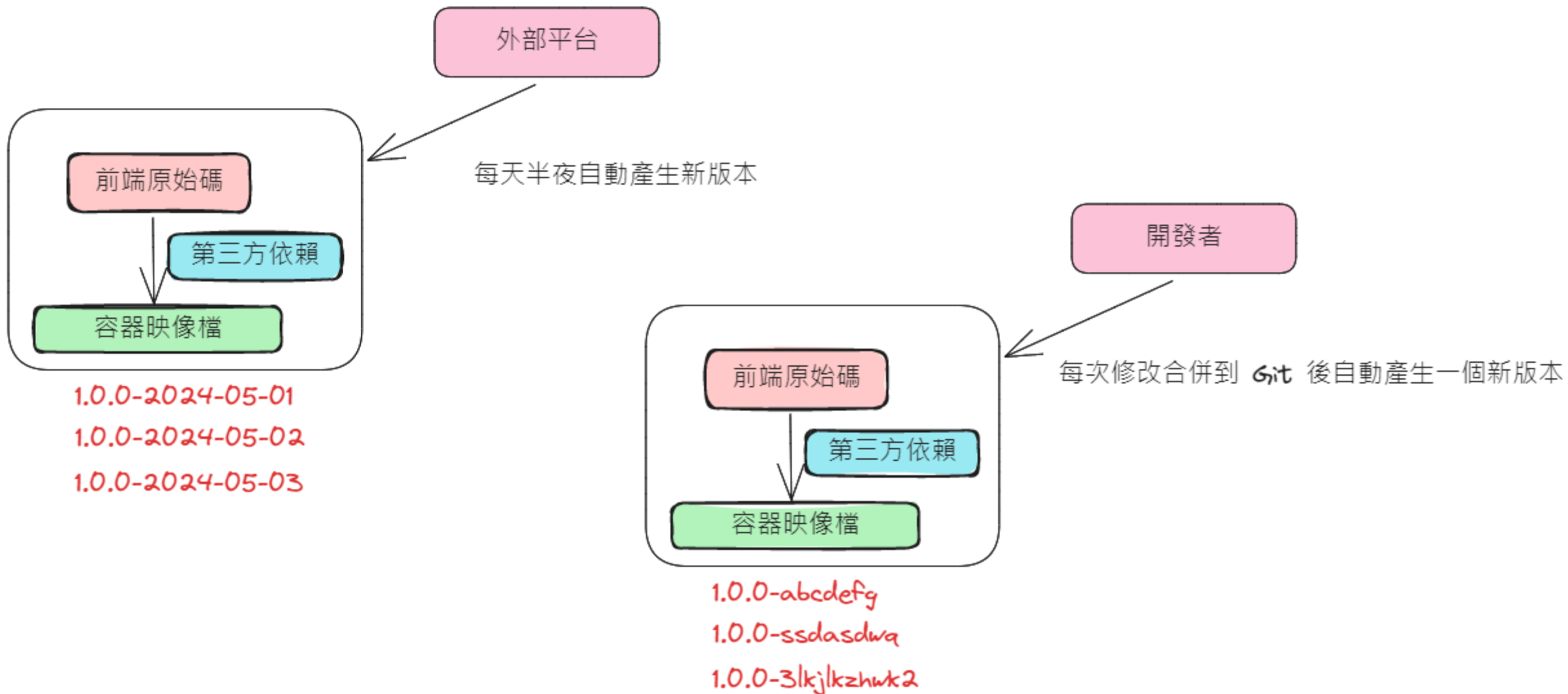
MANIFEST

<input type="checkbox"/>	20240507_e8a1d6226-a...	9 hours ago	 Passed	147.2 MiB	Never	SHA256 358edf3582fd	
<input type="checkbox"/>	20240507_e8a1d6226	9 hours ago	 Passed	153.8 MiB	Never	SHA256 aa7ee51b2cca	
<input type="checkbox"/>	20240506_3763f9f2a-ar...	a day ago	 Passed	147.2 MiB	Never	SHA256 f451e87d6d34	
<input type="checkbox"/>	20240506_3763f9f2a	a day ago	 Passed	153.8 MiB	Never	SHA256 37fb3c43e3b2	
<input type="checkbox"/>	20240505_a72353209-a...	2 days ago	 Passed	147.2 MiB	Never	SHA256 115b0d1fd565	
<input type="checkbox"/>	20240505_a72353209	2 days ago	 Passed	153.8 MiB	Never	SHA256 5c27456777c6	
<input type="checkbox"/>	20240504_2616b731c-a...	3 days ago	 Passed	147.2 MiB	Never	SHA256 2747c82850c1	

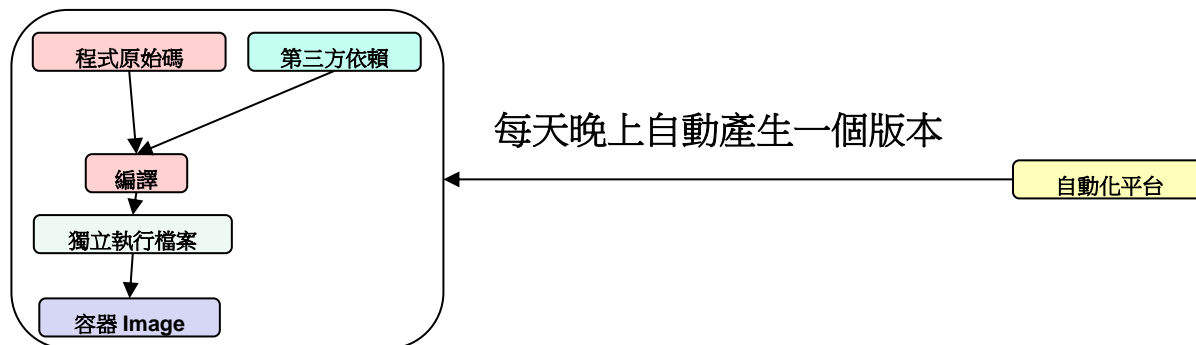
容器化

- 整合一些自動化流程，可以做到下列操作
- 每天晚上自動建置一個版本
 - 這樣每天都會有一個當前最新版本供團隊測試
- 每次程式碼修改，就自動建置一個版本
 - 開發人員可以快速針對自己的修改去測試
- 正式上線時，大家說好採用某個版本
 - 有共識採用哪個版本
 - 透過 **docker** 指令可以快速檢視運行版本

容器化



一些用法



myapp:2025-04-01
myapp:2025-04-02
myapp:2025-04-03
myapp:2025-04-04



myapp:abcdefg
myapp:123acdef
myapp:54721ujaks
myapp:18931hj2

要怎麼區分版本? 用什麼當作版本的名稱?

問題

- 解決問題
 - 前後端人員都可以利用 **Docker** 來建置完整的測試環境，不需要懂太多技術細節
 - 會操作 **Docker** 加上會設定即可
 - 不同機器之間的版本可以一致，同時可以透過 **docker** 快速檢視每個版本的差異
- 尚未解決問題
 - 多機器情況下，如何透過 **Docker** 去佈署這些服務
 - 人工 **SSH** 到每台機器
 - 透過 **Ansible** 這種工具，將 **SSH** 與後續的操作給程式化，讓你可以減少人為操作與可能的人為錯誤

Container

- 前述簡述了如何透過 **Container** 來簡化應用程式的安裝與部屬
- 這邊來看一些實際的範例



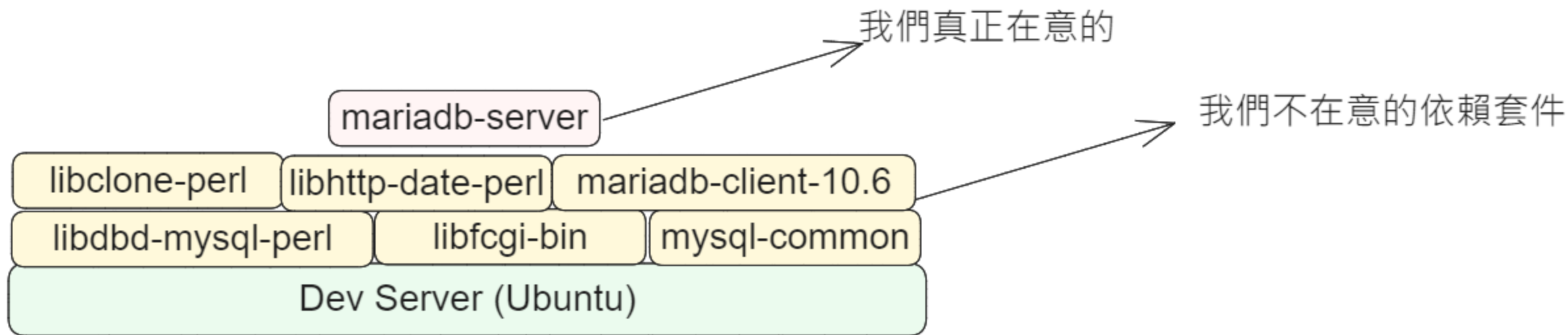
MariaDB

- 前述簡述了如何透過 **Container** 來簡化應用程式的安裝與部屬
- 這邊來看一些實際的範例
- 透過 **apt** 套件管理去安裝 **mariadb-server**，有一堆東西要安裝
 - 大家通常都是無腦按下Y，就是給它安裝

```
root@d15ee-s29-tvm3-ap-0001:~# apt install mariadb-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  galera-4 libcgi-fast-perl libcgi-pm-perl libclone-perl libconfig-inifiles-perl libdbd-mysql-perl libdbi-perl libencode-locale-perl libfcgi-bin libfcgi-perl
  libfcgi0ldbl libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl libhttp-date-perl libhttp-message-perl libio-html-perl liblwp-mediatypes-perl libmariadb3
  libmysqlclient21 libtimedate-perl liburi-perl mariadb-client-10.6 mariadb-client-core-10.6 mariadb-common mariadb-server-10.6 mariadb-server-core-10.6 mysql-common
  socat
Suggested packages:
  libmldbm-perl libnet-daemon-perl libsql-statement-perl libdata-dump-perl libipc-sharedcache-perl libbusiness-isbn-perl libwww-perl mailx mariadb-test
The following NEW packages will be installed:
  galera-4 libcgi-fast-perl libcgi-pm-perl libclone-perl libconfig-inifiles-perl libdbd-mysql-perl libdbi-perl libencode-locale-perl libfcgi-bin libfcgi-perl
  libfcgi0ldbl libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl libhttp-date-perl libhttp-message-perl libio-html-perl liblwp-mediatypes-perl libmariadb3
  libmysqlclient21 libtimedate-perl liburi-perl mariadb-client-10.6 mariadb-client-core-10.6 mariadb-common mariadb-server mariadb-server-10.6 mariadb-server-core-10.6
  mysql-common socat
0 upgraded, 30 newly installed, 0 to remove and 43 not upgraded.
Need to get 18.0 MB/18.4 MB of archives.
After this operation, 164 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

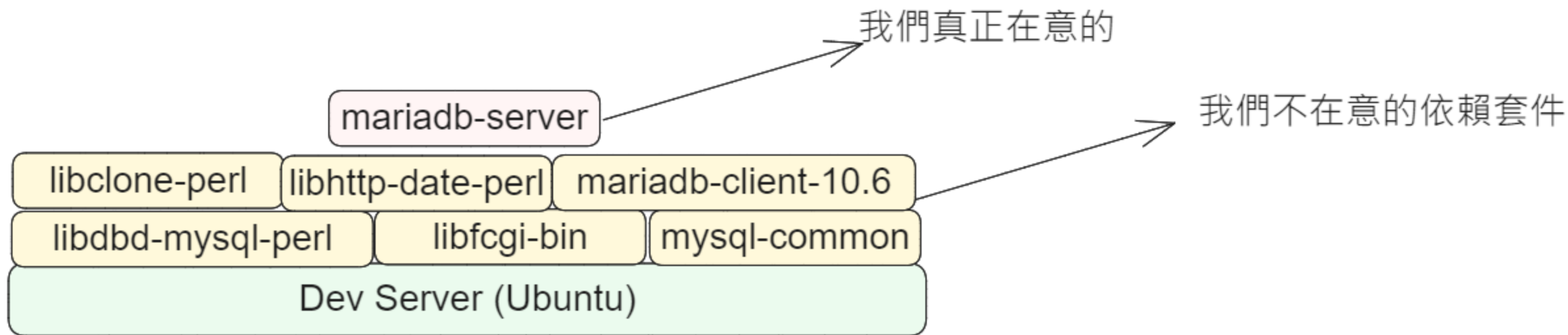
MariaDB

- 安裝完畢後，整個開發環境終於完成，這時候整個系統架構如下
- 我們只在意 **Mariadb-server**，其餘黃色的都是他的依賴套件



MariaDB

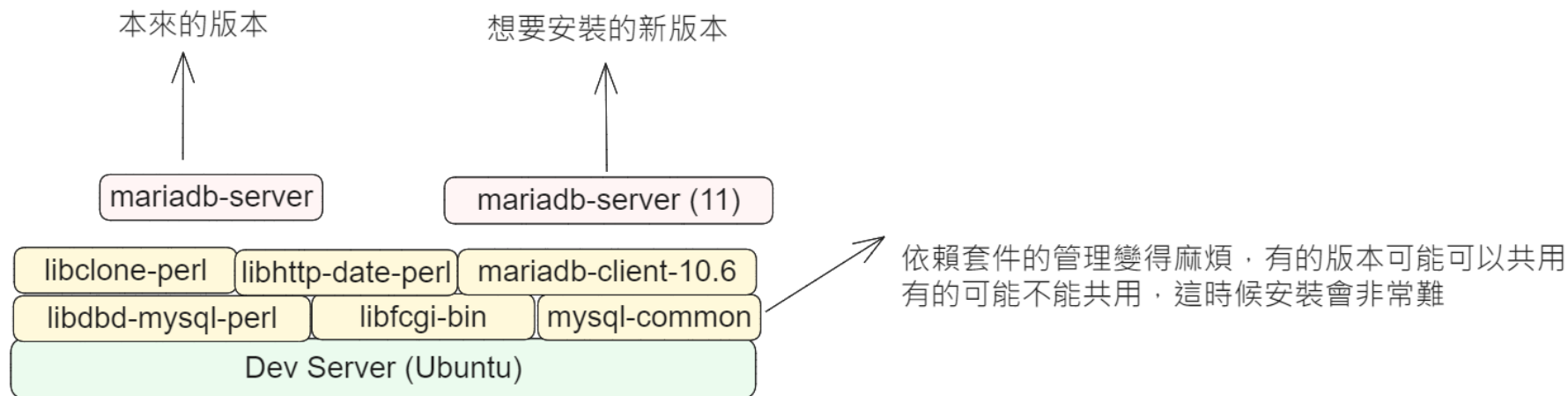
- 安裝完畢後，整個開發環境終於完成，這時候整個系統架構如下
- 我們只在意 **Mariadb-server**，其餘黃色的都是他的依賴套件



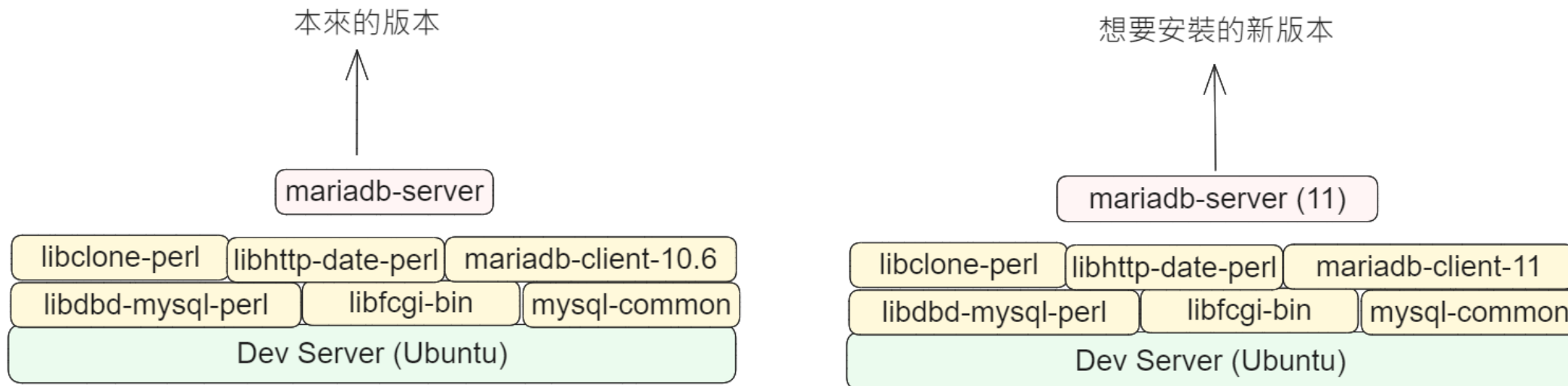
Why Container

- 結果某一天因為下列理由被要求測試 MariaDB 11.6
 - 新功能/更好的效能/Bug 修復/版本 sunset
- 所以需要重新安裝 MariaDB 11.6 進行測試
 - 結果安裝就會遇到下列問題
 - Ubuntu 22.04 預設沒有 11.6，要升級到 Ubuntu 24.04
 - 很多依賴性的套件版本也都要一起升級
- 如果想要針對 MariaDB 10.6 與 MariaDB 11.6 交替測試則會更麻煩
 - 除非準備兩台獨立機器

同機器多版本，升級管理複雜



不同機器多版本，管理相對輕鬆，但是成本高



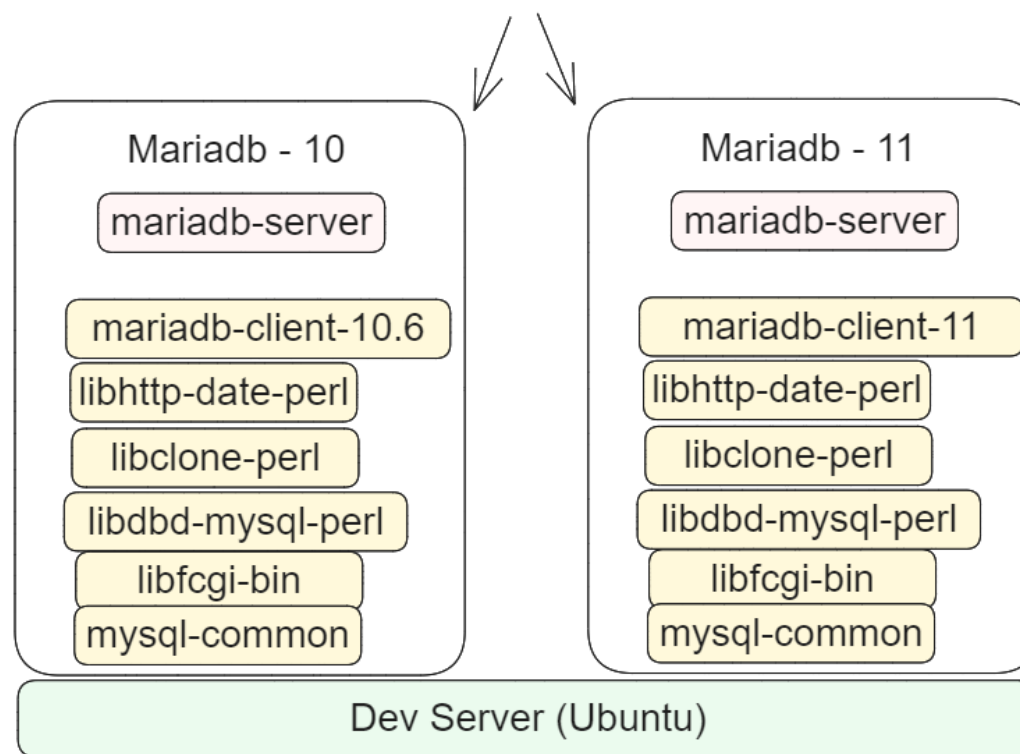
Why Container

- 一個常見的解法就是透過 **Container** 的概念，簡化 **MariaDB** 的安裝與部署
- **Container** 會將目標應用程式與所有的依賴套件都打包放到一起
 - 彼此之間獨立不影響彼此，所以可以同時安裝多套 **MariaDB** 互相測試
- 安裝方式
 - `docker run mariadb:10.6`
 - `docker run mariadb:11.4`
 - `docker run mariadb:11.5`
 - `docker run mariadb:11.6`

Why Container

- 一個常見的解法就是透過 **Container** 的概念，簡化 MariaDB 的安裝與部署
- **Container** 會將目標應用程式與所有的依賴套件都打包放到一起
 - 彼此之間獨立不影響彼此，所以可以同時安裝多套 **MariaDB** 互相測試

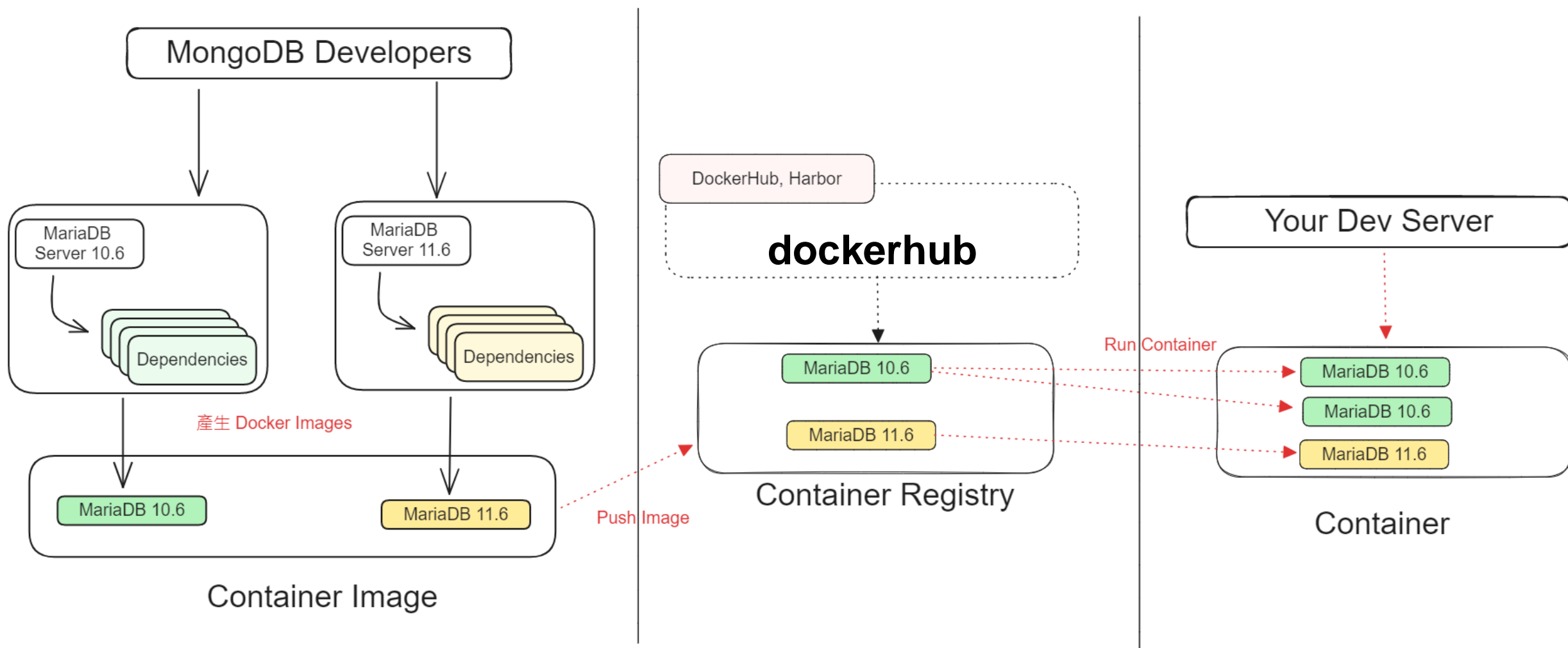
需要有一個方式能夠讓這些套件範圍縮小，彼此互相不干擾
然後將所有需要的檔案放一起，常見的一種手法稱為容器化 (Containerize)



Container

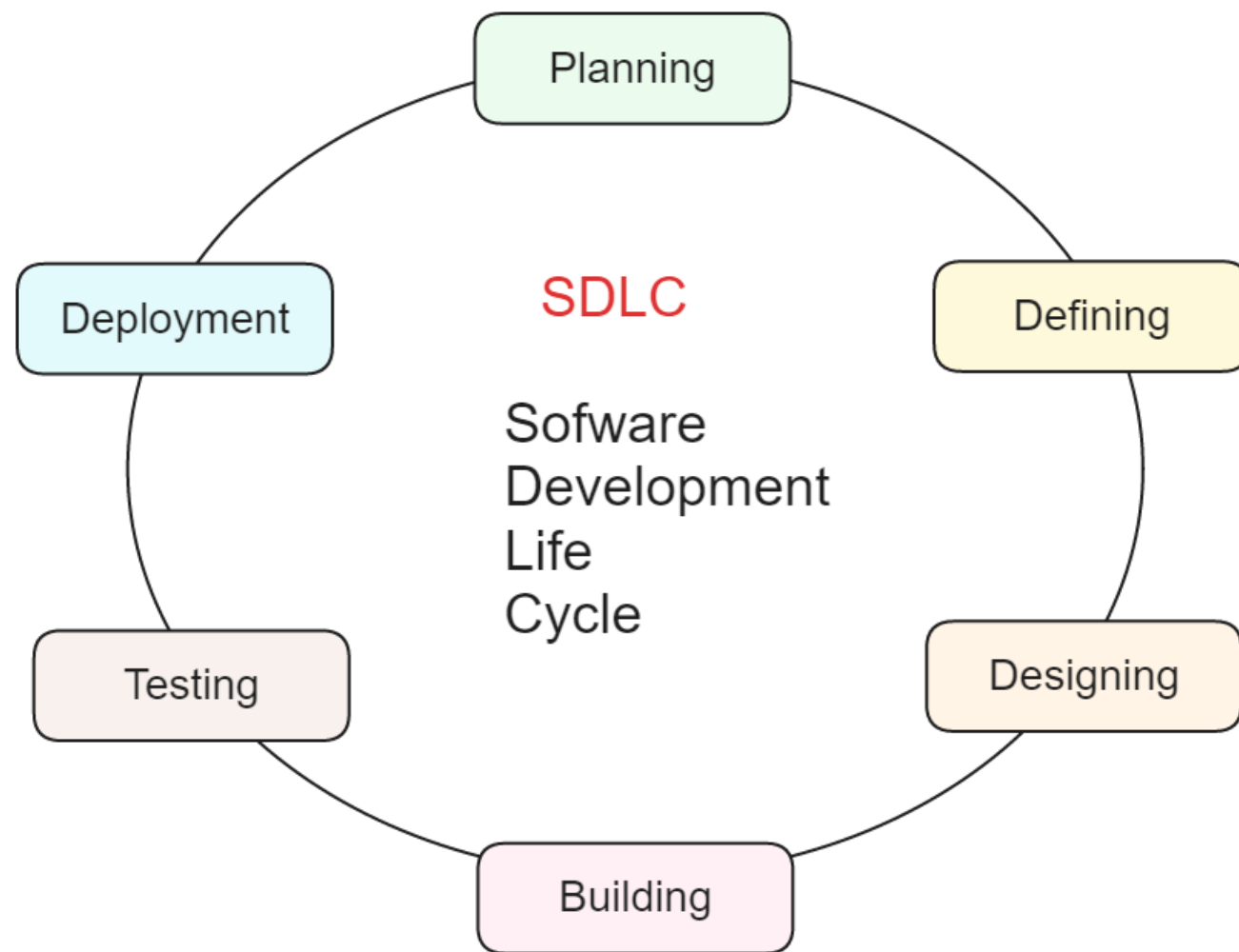
- 相對於 Virtual Machine，Container 提供一個輕量級的虛擬化概念，再共享 OS 的前提下，將應用程式所需要的依賴套件全部打包成一個檔案，以達到 “Build Once, Run Everywhere” 的概念。
- 三大元件
 - 被打包後的整包檔案(應用程式+依賴套件) -> 稱為 Container Image (容器 Image)
 - 打包包裹
 - 存放並且透過網路共享的地方 -> 稱為 Container Registry
 - 轉送包裹
 - 將打包後的應用程式給運行起來 -> 稱為 Container
 - 解開包裹

Container



Container

- 以軟體開發生命週期來看
Container 可以幫忙
 - Building
 - Testing
 - Deployment



Container 的開源專案

- 目前最知名支援 Container 流程的開源專案就是 Docker
 - Linux/Ubuntu/Mac 都有相關軟體可以安裝
- 搭建了 dockerhub 這個最知名的 Container Registry，讓全世界的開發者可以分享各種打包好的 Container Image
 - 更容易的去運行別人開發好的應用程式
 - 更快速且更方便的利用巨人來加速開發

Mariadb 為範例

- 當 Docker 安裝完畢後，如何於本地環境建立與創建？
- 先瀏覽有哪些版本與檔案
 - https://hub.docker.com/_/mariadb


mariadb

Docker Official Image • 1B+ • 6.0K

MariaDB Server is a high performing open source relational database, forked from MySQL.

DATABASES & STORAGE

Overview

Tags

Quick reference

- Maintained by: [MariaDB Foundation](#), [MariaDB plc](#), with contributions from our [community](#).
- Where to get help: [Database Administrators \(Stack Exchange\)](#), [MariaDB Knowledge Base](#) ([Ask a Question here](#)).

 Also see the ["Getting Help with MariaDB" article on the MariaDB Knowledge Base](#).

Supported tags and respective Dockerfile links

- [11.8.1-ubi9-rc](#) [11.8-ubi9-rc](#) [11.8.1-ubi-rc](#) [11.8-ubi-rc](#)
- [11.8.1-noble-rc](#) [11.8-noble-rc](#) [11.8.1-rc](#) [11.8-rc](#)
- [11.7.2-ubi9](#) [11.7-ubi9](#) [11-ubi9](#) [11.7.2-ubi](#) [11.7-ubi](#) [11-ubi](#)
- [11.7.2-noble](#) [11.7-noble](#) [11-noble](#) [noble](#) [11.7.2](#) [11.7](#) [11](#) [latest](#)
- [11.4.5-ubi9](#) [11.4-ubi9](#) [lts-ubi9](#) [11.4.5-ubi](#) [11.4-ubi](#) [lts-ubi](#)
- [11.4.5-noble](#) [11.4-noble](#) [lts-noble](#) [11.4.5](#) [11.4](#) [lts](#)
- [10.11.11-ubi9](#) [10.11-ubi9](#) [10-ubi9](#) [10.11.11-ubi](#) [10.11-ubi](#) [10-ubi](#)
- [10.11.11-jammy](#) [10.11-jammy](#) [10-jammy](#) [10.11.11](#) [10.11](#) [10](#)
- [10.6.21-ubi9](#) [10.6-ubi9](#) [10.6.21-ubi](#) [10.6-ubi](#)

Mariadb 為範例

- 通常都會有詳細文件告訴你如何執行
 - `docker run --name xxxx $位置:$版本`

How to use this image

The mariadb has a number of tags, and of note is `latest`, as the latest stable version, and `lts`, as the last long term support release.

Running the container

Starting using a minimal configuration

The environment variables required to use this image involves the setting of the root user password:

```
$ docker run --detach --name some-mariadb --env MARIADB_ROOT_PASSWORD=my-secret-pw mariadb:latest
```

or:

```
$ docker run --detach --name some-mariadb --env MARIADB_ALLOW_EMPTY_ROOT_PASSWORD=1 mariadb:latest
```

or:

```
$ docker run --detach --name some-mariadb --env MARIADB_RANDOM_ROOT_PASSWORD=1 mariadb:latest
```

... where the container logs will contain the generated root password.

Mariadb 為範例

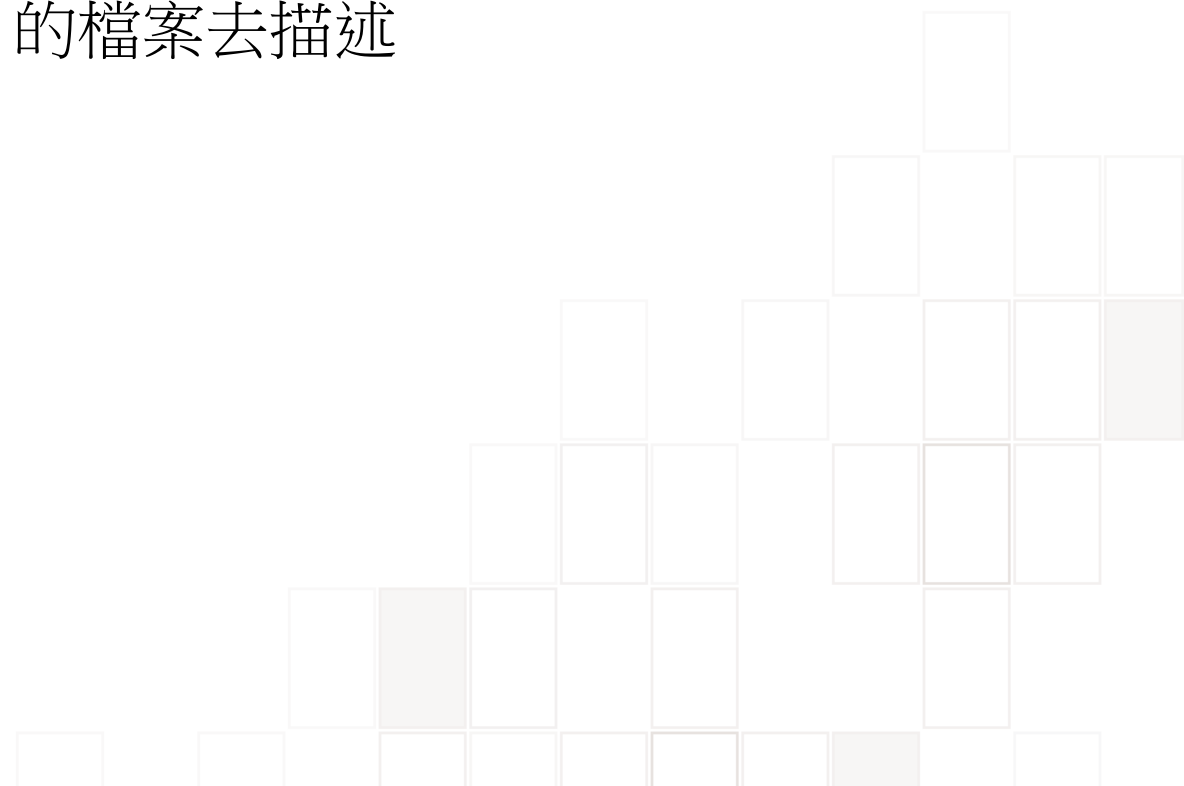
- 對使用者來說，我們在意的是如何抓取 Image 以及運行
- 常見用法
 - docker pull
 - 抓取 Image 但是不要跑，就是抓下來放
 - docker run
 - 幫忙運行，如果 image 不存在則順便抓取
 - docker rm
 - 移除正在運行的 container
 - docker logs
 - 檢視運行 container 的 logs

Mariadb 為範例

- 對開發者來說，我們在意的是如何打包 Image 並且推到遠方環境
- 常見用法
 - docker build
 - 將你的應用程式打包
 - docker push
 - 將本地打包好的 Image 給推到遠方的 Dockerhub

打包 Image

- 打包 **Container Image** 需要精準的描述這些 **Container** 到底本質是什麼？
 - 要安裝那些第三方依賴？
 - 啟動流程是什麼？
- 目前的準則都是透過一個名為 **Dockerfile** 的檔案去描述
 - 搭配 **docker build** 就可以產生



打包 Image

```
# syntax=docker/dockerfile:1
FROM ubuntu:22.04

# install app dependencies
RUN apt-get update && apt-get install -y python3 python3-pip
RUN pip install flask==3.0.*

# install app
COPY hello.py /

# final configuration
ENV FLASK_APP=hello
EXPOSE 8000
CMD ["flask", "run", "--host", "0.0.0.0", "--port", "8000"]
```

From: <https://docs.docker.com/build/concepts/dockerfile/>

打包 Image

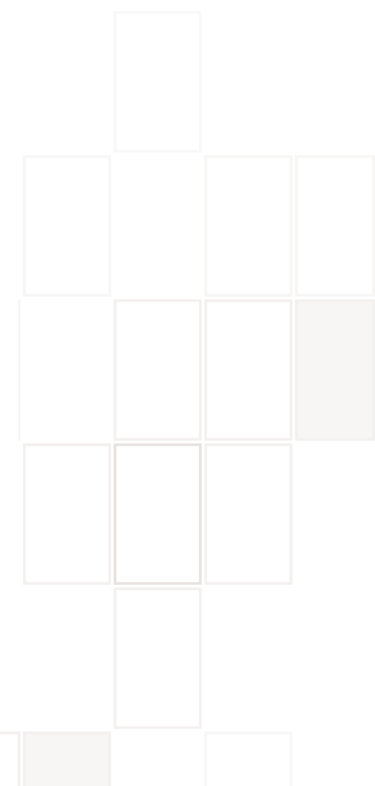
```
FROM python:3.12
WORKDIR /usr/local/app

# Install the application dependencies
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

# Copy in the source code
COPY src ./src
EXPOSE 5000

# Setup an app user so the container doesn't run as the root user
RUN useradd app
USER app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```



From: <https://docs.docker.com/get-started/docker-concepts/building-images/writing-a-dockerfile/>

打包應用程式

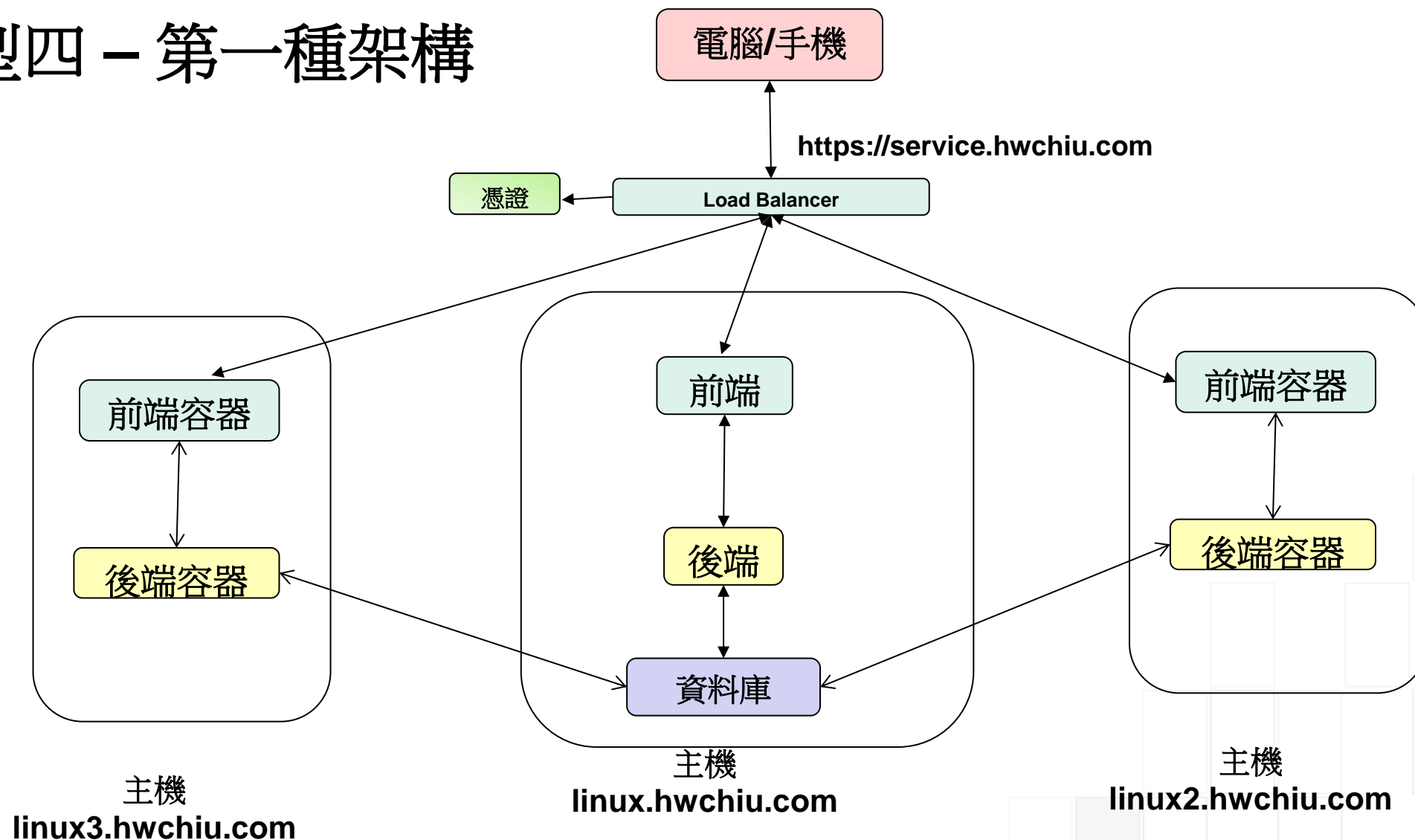
- 網路上資源很多，請嘗試
- 打包自己的應用程式
 - 撰寫 Dockerfile
 - 推到 Dockerhub
- 於不同台電腦，嘗試下載並且使用你所上傳的 **Docker Image**
 - 嘗試看看透過 **Container** 的機制來簡化應用程式的佈署與安裝

模型四

- 當前端多副本後，接下來嘗試將後端也多副本佈署看看
 - 所有的應用程式都採用容器化的方式去建置編譯與運行
- 這個架構下的問題就會變成
- 多副本架構下，請問前端要怎麼存取這些後端？
 - 多個前端
 - 多個後端



模型四 – 第一種架構

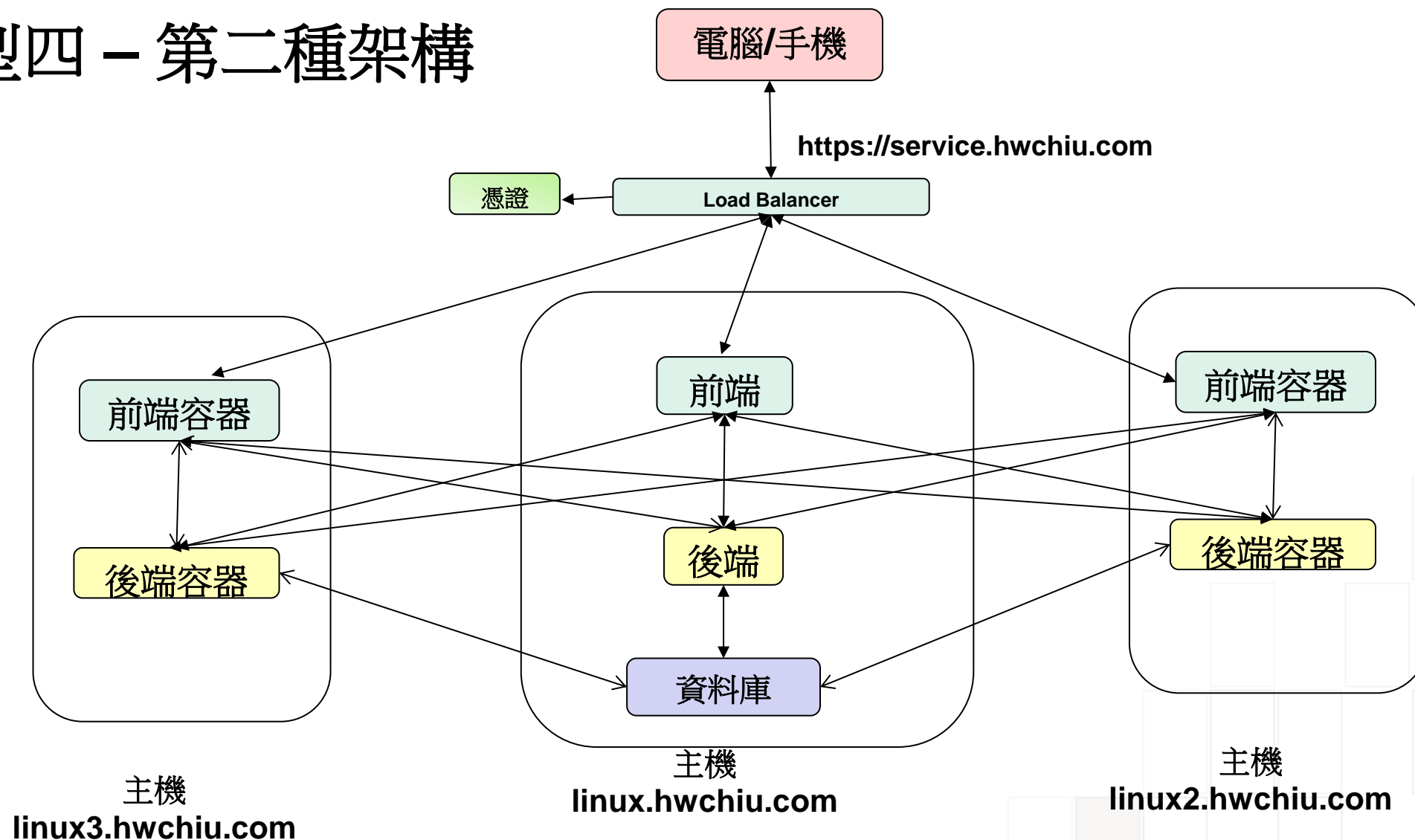


模型四 – 第一種架構

- 每台機器上，都佈署一個前端+一個後端
 - 每個前端都會連接到自己本地的後端
- 所以架構就是
 - 一個負載平衡器，連接到每台機器上的前端
 - 前端直接轉發到本地的後端
- 有任何問題？



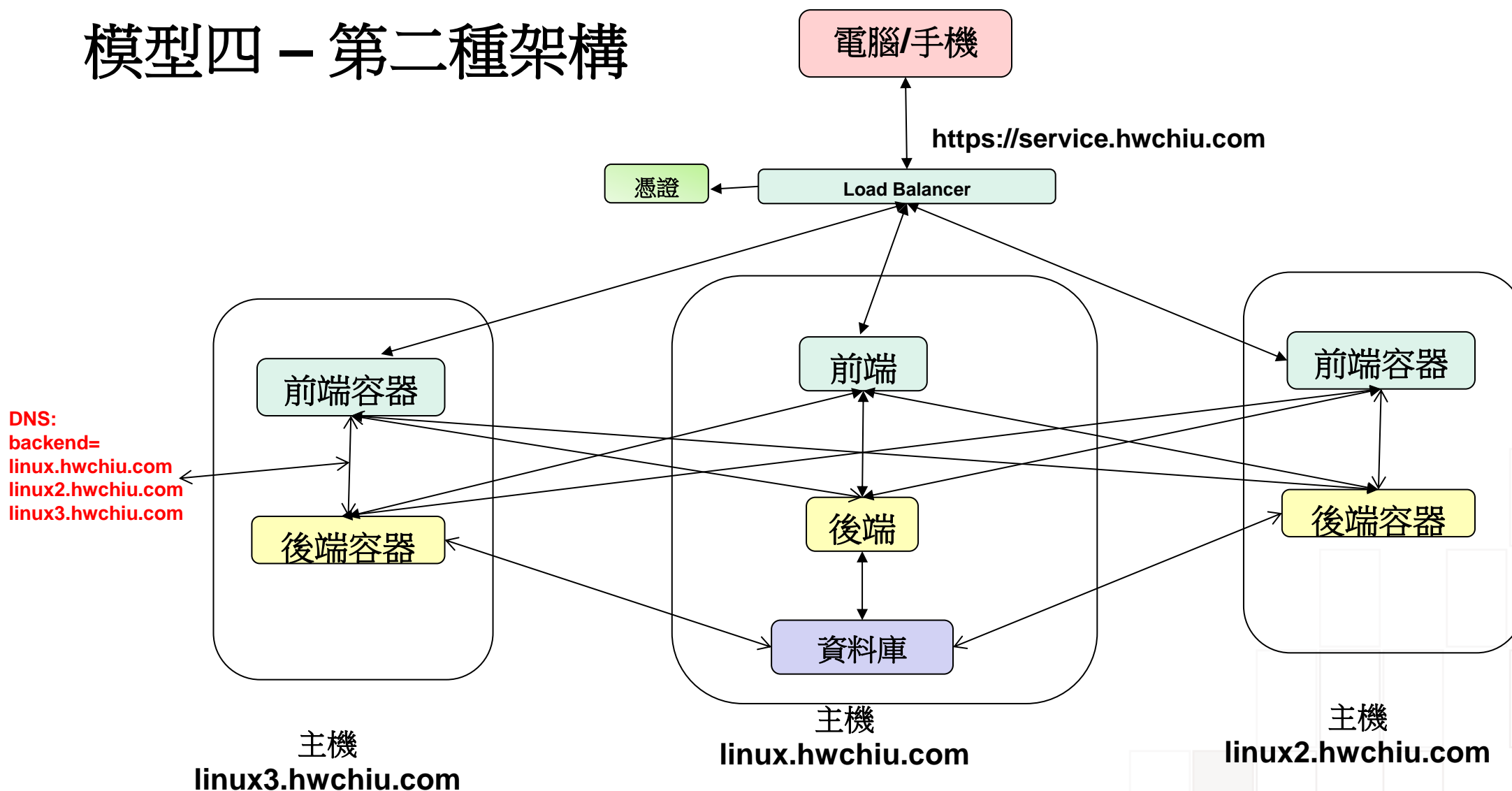
模型四 – 第二種架構



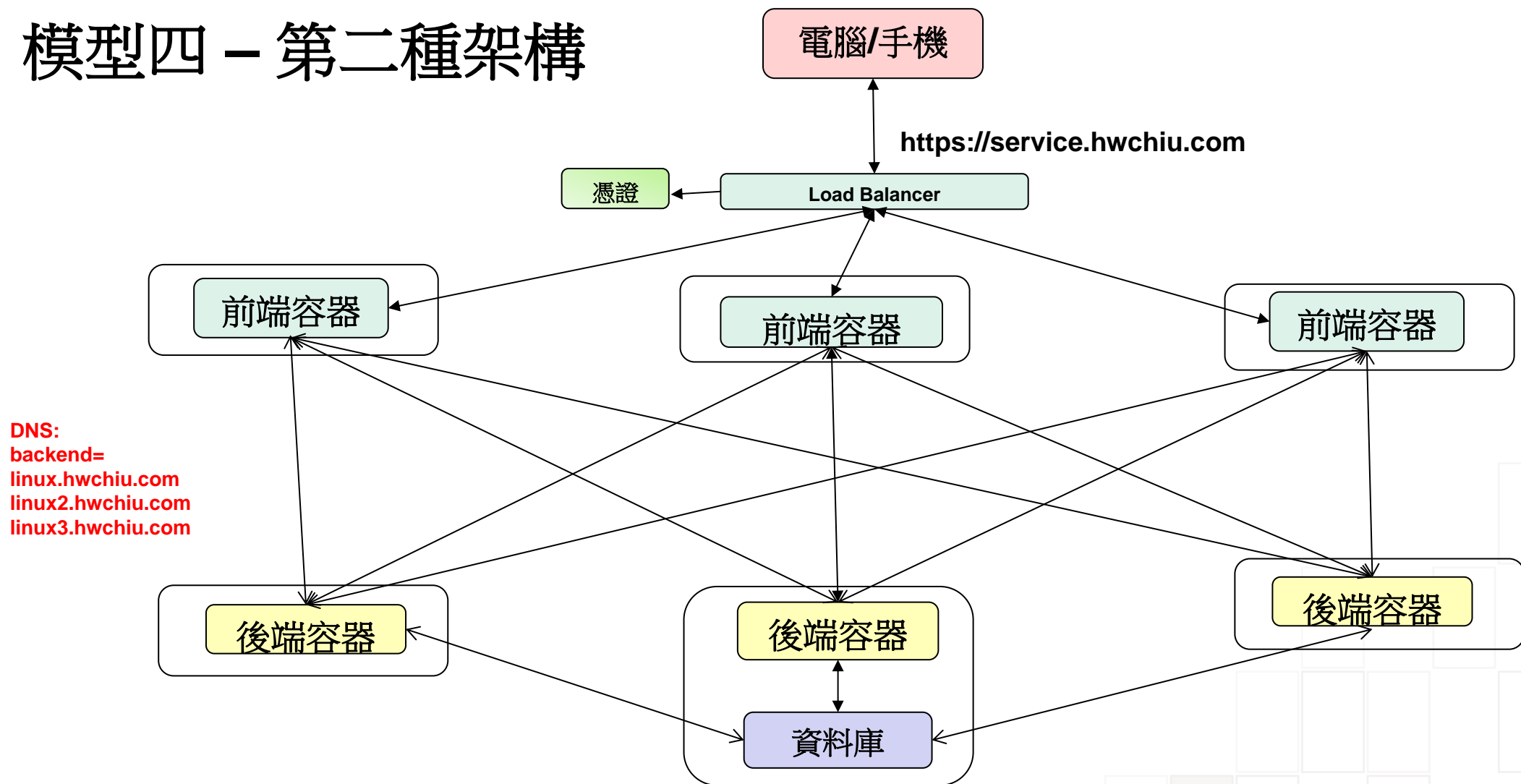
模型四 – 第二種架構

- 為了達到真正的分散，每個前端與後端應該要物理性分離
 - 彼此數量上也有可能不一樣
 - 譬如 三個前端副本 + 四個後端副本
- 理論上前端可能也要有一個方式存取到一個對應的後端
 - 後端的 IP 很多，全部都要記錄
 - 後端的存取也要有負載平衡處理
- 要如何把前端之前的處理方式導入？
 - 再次導入 DNS 去解析
 - 再一個額外的負載平衡器(Load Balancer)

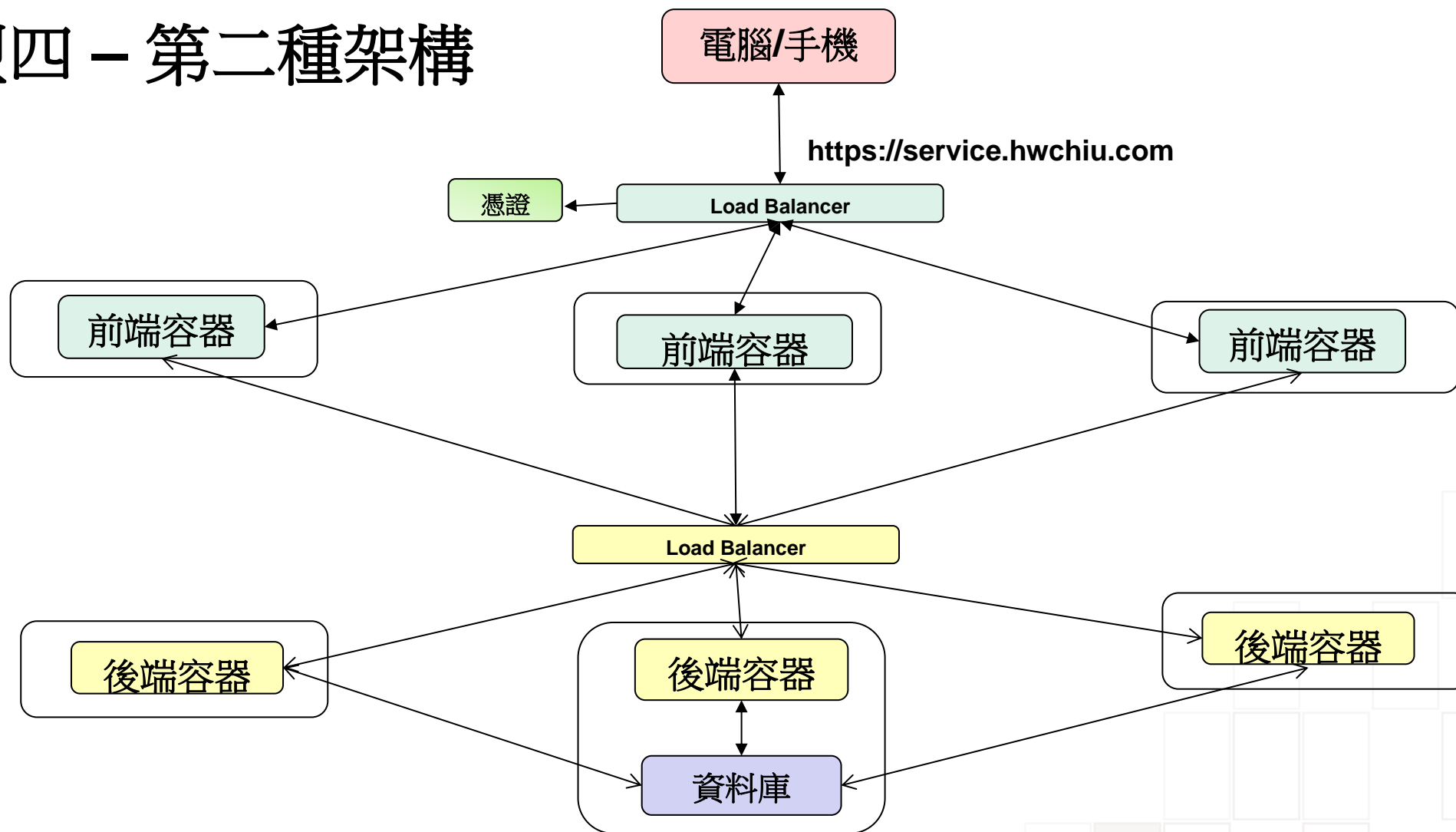
模型四 – 第二種架構



模型四 – 第二種架構



模型四 – 第二種架構



模型四 – 第二種架構

- 所有的開發跟佈署還要考慮到一個問題
- 動態加入一個新機器的時候該怎麼辦？
 - 修改設定檔案，重啟服務？
- 假設新增加一個新的後端服務
 - 架構一
 - 前端要怎麼知道？
 - ◆ 重新改寫設定檔案嗎？
 - ◆ 需要重啟嗎？
 - 架構二
 - DNS
 - 負載平衡器

模型四 – 第二種架構

- DNS:
 - 需要調整 DNS 名稱，有任何更動就是申請 域名的改變
 - DNS 改變要花多少時間？
 - 機器壞掉：申請域名更動了，結果流量還是繼續走向壞掉的機器？
 - 加入新機器：域名反應太慢，結果當前副本數量撐不住直接被打壞？
- 負載平衡
 - 修改負載平衡器的設定檔案
 - 人為修改還是程式化修改
 - 重啟會影響既有的服務？

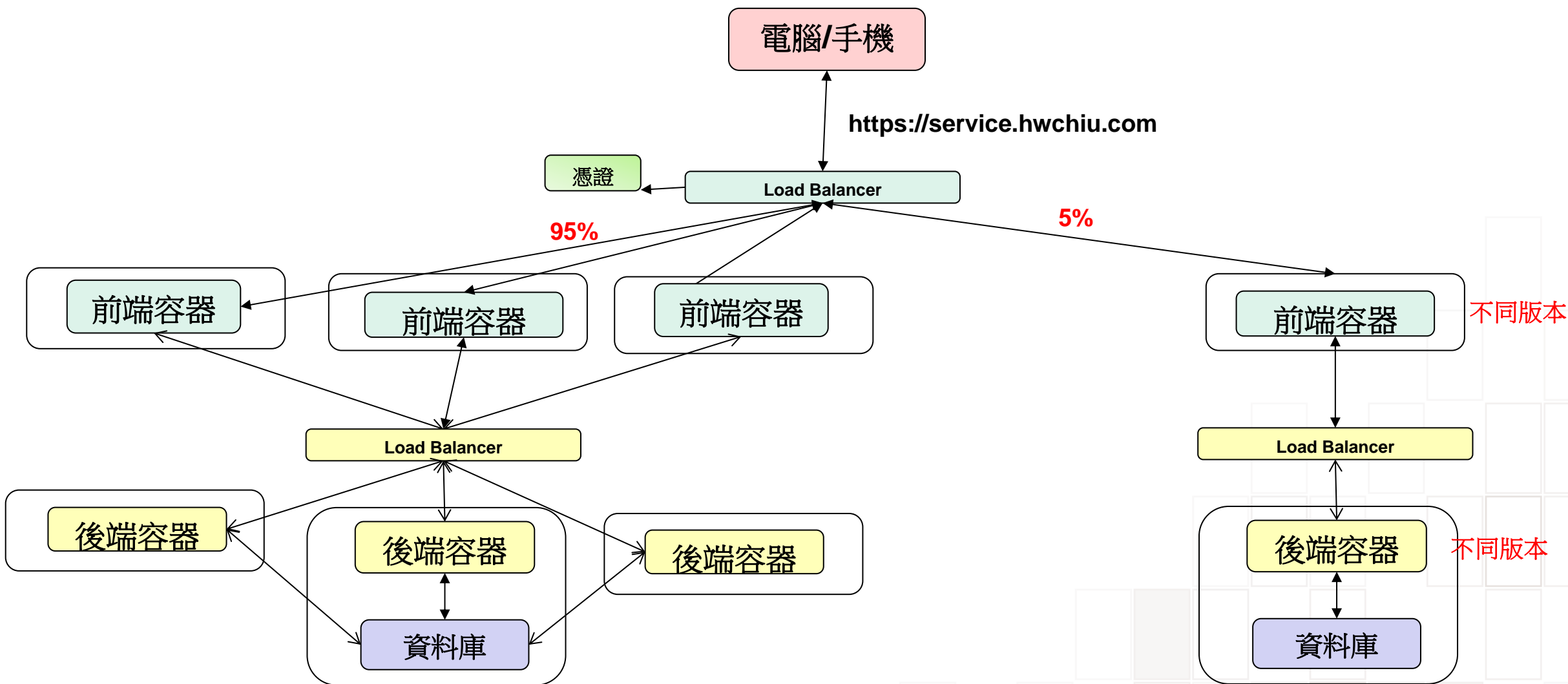
模型四 – 流量策略

- 前後端都多副本且多機器後，現在剩下的困難問題還是佈署
 - 如何同時佈署多個節點
- 更複雜的是如果要導入不同的佈署策略
 - 藍綠佈署
 - 金絲雀佈署
 - ...etc

模型四 – 流量策略

- 以臉書/IG 來說，如果我想要讓某些使用者先使用新版本，其餘的繼續使用舊版本，我該怎麼辦？
 - 譬如 5%:95% (5% 用新版，95%用舊版)
- 前後端都需要兩個版本
 - 前端要有能力將流量給分量到不同的後端版本
- 若採取 **DNS** 架構來說，似乎這邊又有點難處理
- 若採取 **LoadBalancer** 的架構，似乎可以透過某些演算法來幫忙處理

模型四 – 第二種架構



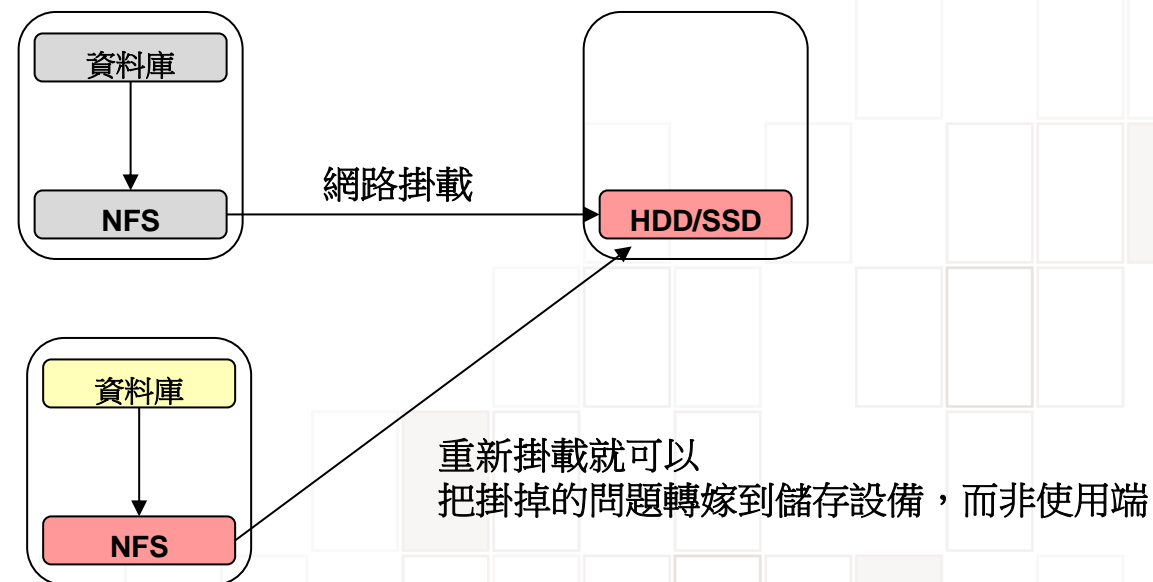
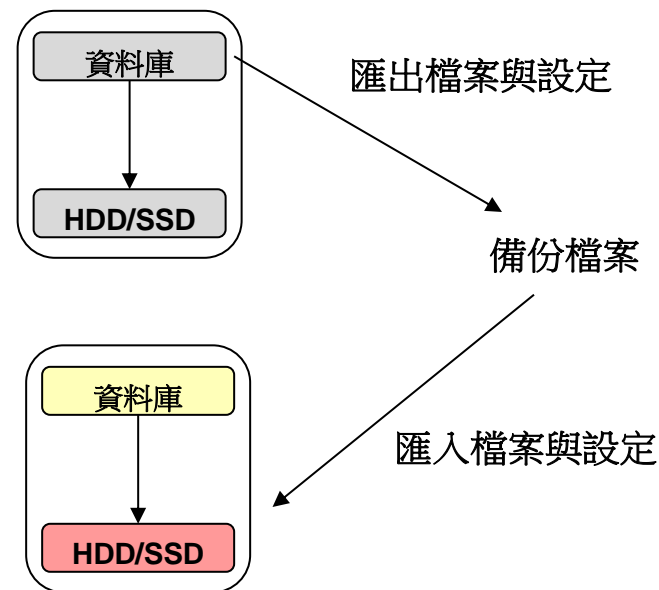
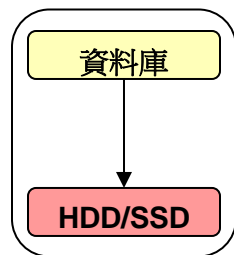
模型五

- 前後端都已經多副本了，目前唯一剩下的瓶頸就是資料庫本身
- 資料庫本身會牽扯到兩個部分
 - 資料放哪邊
 - 本地硬碟
 - 遠方硬碟
 - 資料庫本身的應用是否支援多副本
 - 多副本常見會分成
 - ◆ 多讀多寫
 - ◆ 多讀單寫
 - 想想 OS 課程中所提到的 Lock + Critical Section
 - 概念類似，很多東西想要達到多讀多寫是比想像中困難的

模型五

- 資料庫所需要的硬碟，分本地與遠端
- 本地硬碟
 - 若節點損毀，你也沒有辦法於其他節點上佈署，因為資料也遺失
 - 搭配 **SSD/NVME**，存取速度快
- 遠方硬碟
 - 要準備一台檔案伺服器
 - NFS, NetApp 等
 - 資料庫的應用可以換機器佈署，只要可以透過網路存取到即可
 - 依賴網路存取，因此網路也會影響存取效能

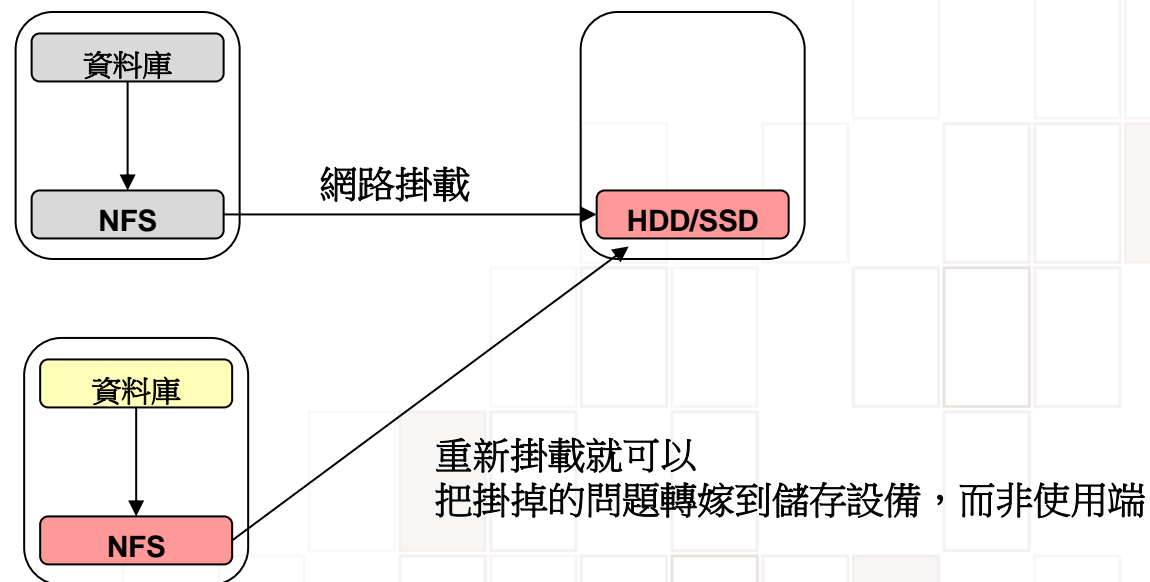
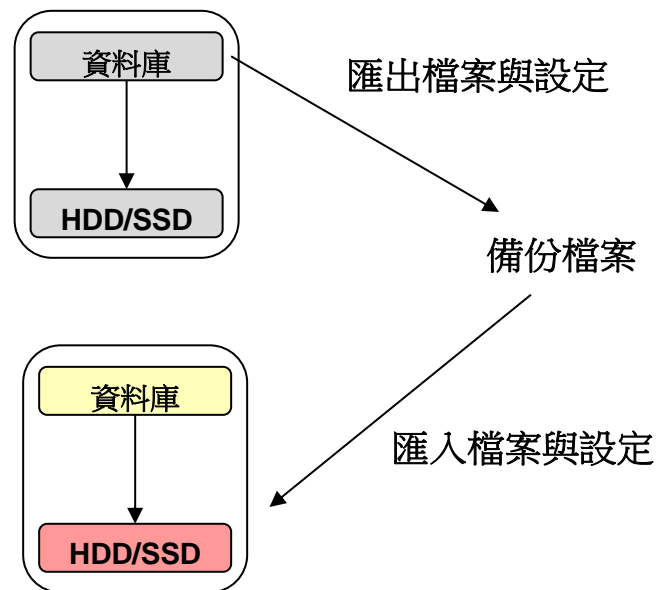
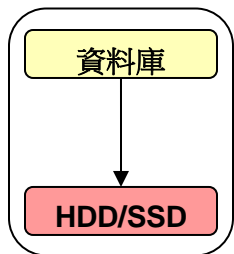
模型五 (節點損毀怎麼辦)



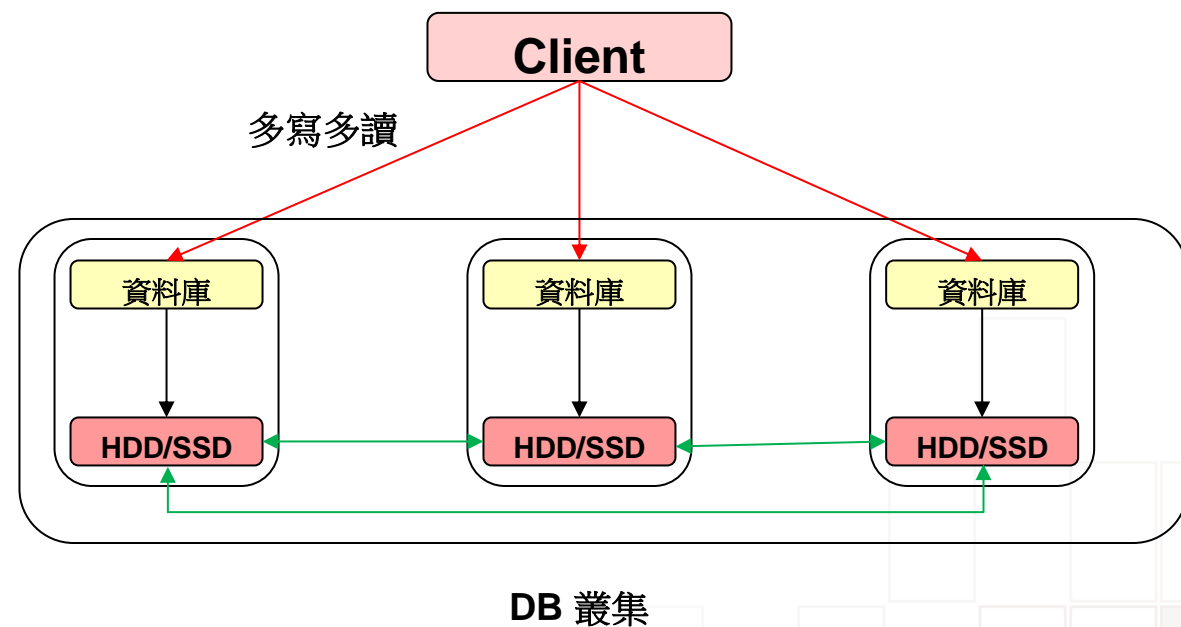
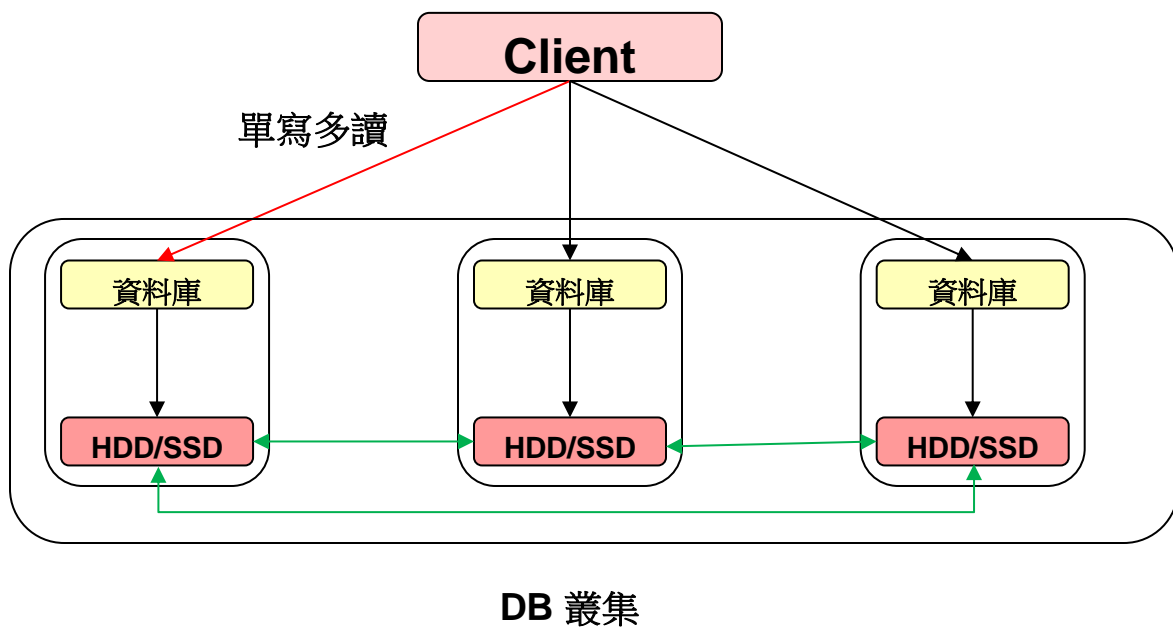
模型五

- 至於資料庫要的多副本模式，常見的有
 - 多讀多寫
 - 多讀單寫
- 如果走多讀單寫
 - 應用程式則是要知道，寫要去哪個節點寫
 - 如果寫的副本壞了，後續會怎麼運作？
- 多讀多寫
 - 應用程式隨便找人寫都好
- 聽起來多讀多寫很棒，但是實作上更難，一定會有更多的技術挑戰跟效能影響
 - 需要閱讀每個資料庫的架構文件

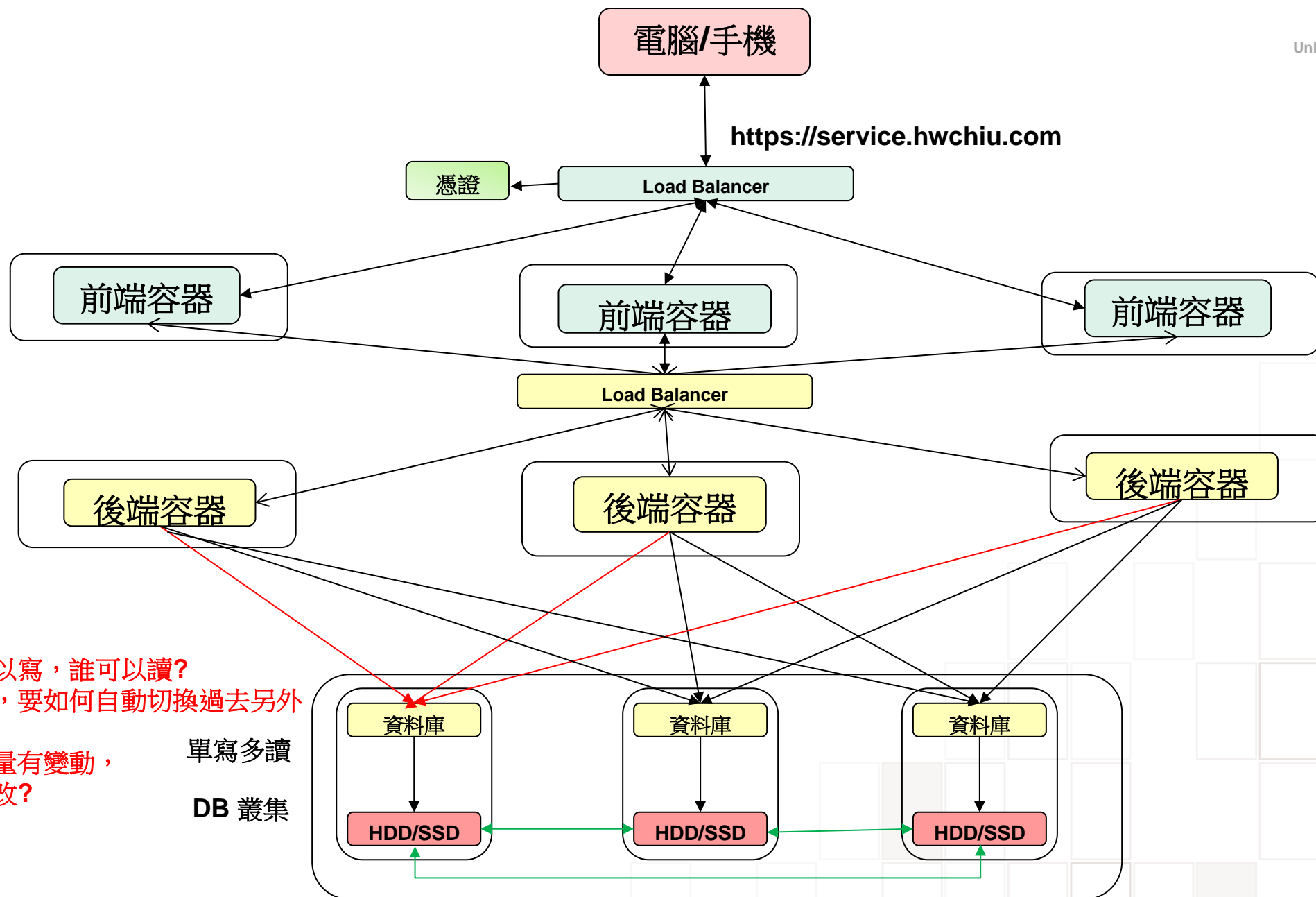
模型五 (節點損毀怎麼辦)



模型五



模型五



問題:

1. 後端怎麼知道誰可以寫，誰可以讀？
2. 如果寫的服務壞了，要如何自動切換過去另外一個**DB**？
3. 如果資料庫副本數量有變動，後端應用程式怎麼改？

單寫多讀

DB 叢集

模型回顧

- 到目前為止，透過 **Docker** 加上各種(負載平衡 or **DNS**) 的幫助，我們能夠完成多副本的前後端佈署
 - 透過 **SSH/Ansible** 等各種腳本的幫忙，我們能夠程式化佈署的操作，減少人為操作
- 然後上述流程實際上還有很多細節要討論，並沒有如前述講的這麼完美

模型回顧

- 節點壞掉的話
 - 要怎麼重新佈署這些 前後端服務
 - **Load Balancer** 那邊的設定要不要更新? 會不會有連線跑到壞掉的節點?
- 流量負荷不了，要加入新節點該怎麼辦?
 - **Load Balancer** 那邊的設定要不要更新?
- 流量負荷不了，要加入更多副本(機器數維持一樣)該怎麼辦?
 - **Load Balancer** 改設定?
- 上述的這些問題，系統管理員要怎麼處理?
 - 人為觀察，手動處理
 - 程式化觀察，手動處理
 - 程式化觀察，程式化但手動處理
 - 程式化觀察，程式自動處理

模型回顧

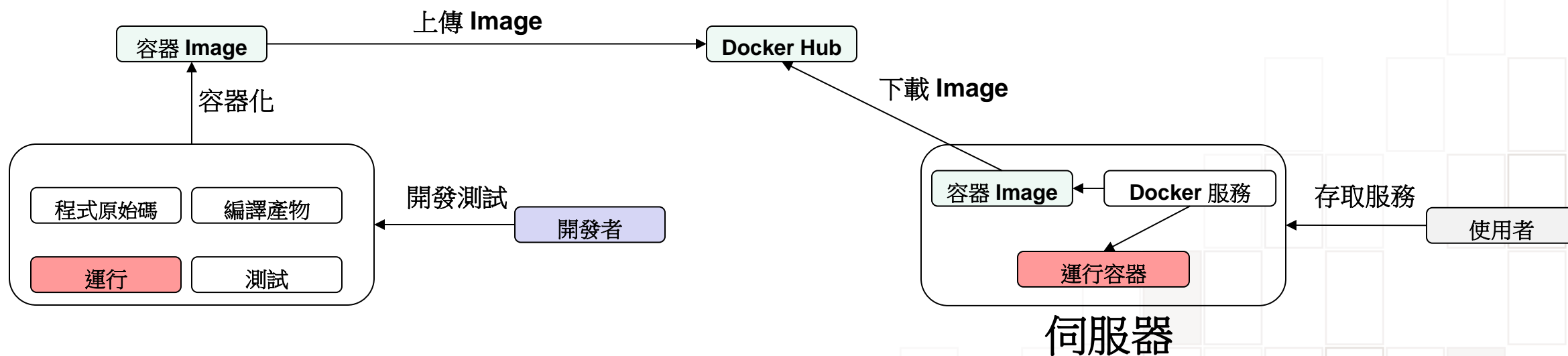
- 以開發者來說，目前的架構還有什麼痛點
- 平常修改的東西，要如何佈署到環境中
- 手動
 - Docker build -> 建置 Image
 - Ansible/SSH -> 佈署到遠方機器
 - 不是每個人都有每台機器的存取與執行權限
 - 如果要分不同版本佈署，設定要怎麼處理
 - 如果佈署失敗要怎麼辦

工作流程

- 以當前架構來說，我們有下列事務要處理
 - 開發前端
 - 開發後端
 - 處理 DNS/LoadBalancer
 - 處理 Nginx
 - 處理節點安裝
 - 部署應用程式

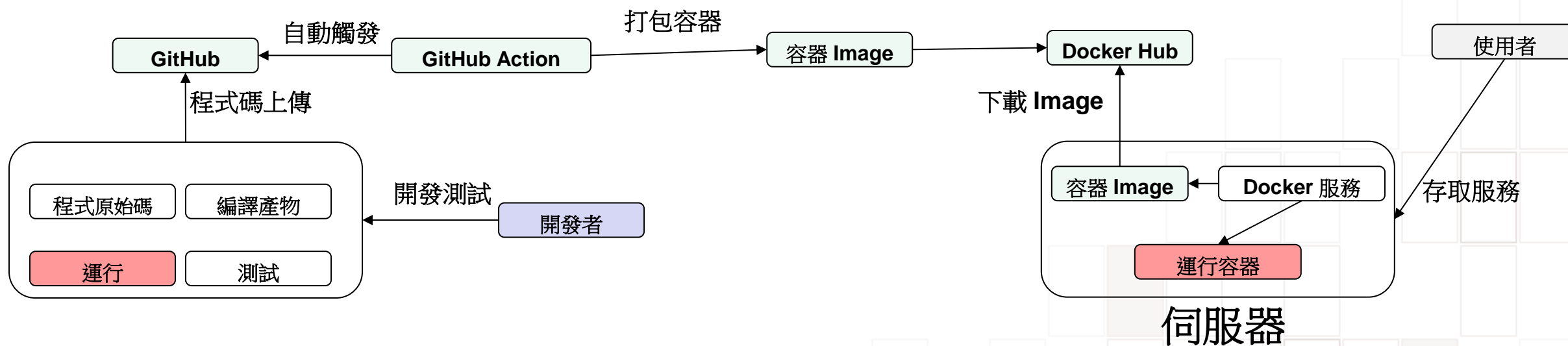
開發與佈署流程(人工操作)

- 手動進行
 - 開發/測試/建置容器/上傳容器/下載容器/運行容器
- 程式化操作
 - N/A



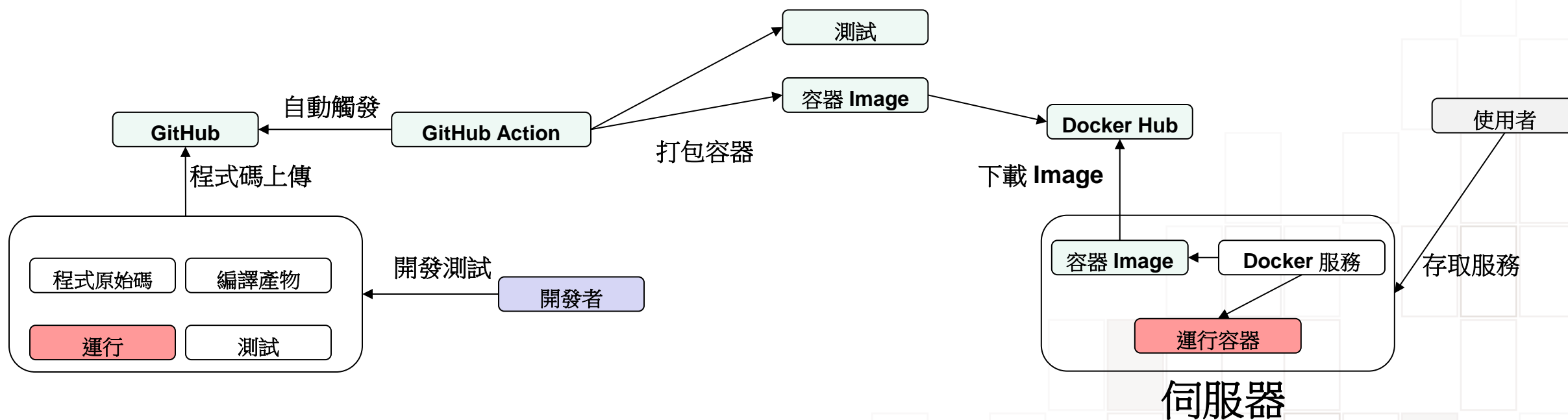
開發與佈署流程(嘗試改進)

- 手動進行
 - 開發/測試/下載容器/運行容器
- 程式化操作
 - 建置容器/上傳容器/



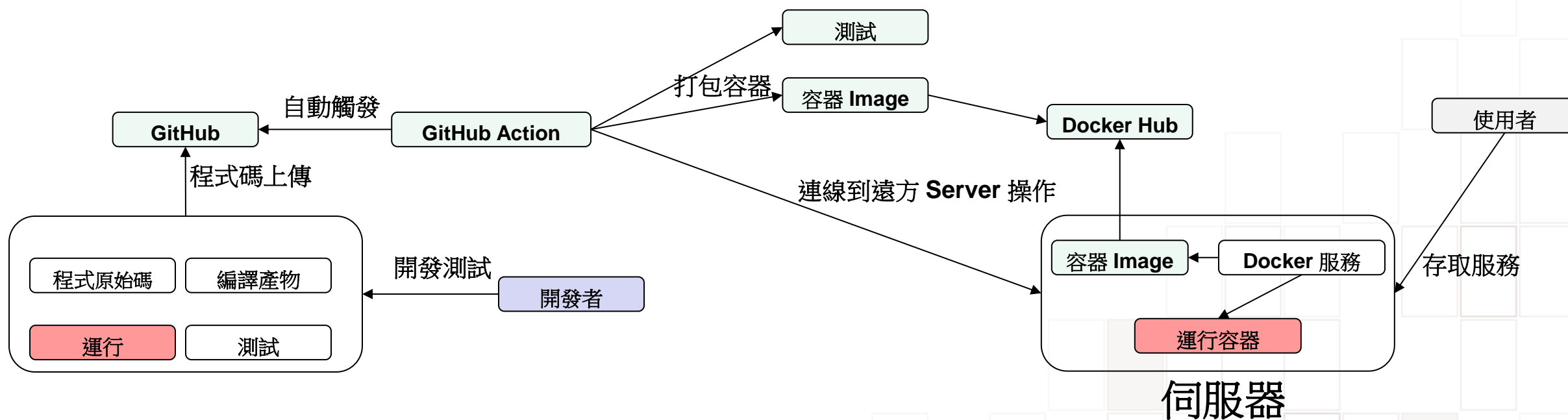
開發與佈署流程(嘗試改進)

- 手動進行
 - 開發//下載容器/運行容器
- 程式化操作
 - 測試/建置容器/上傳容器/



開發與佈署流程(嘗試改進)

- 手動進行
 - 開發/
- 程式化操作
 - 下載容器/運行容器/測試/建置容器/上傳容器/



回家作業

Catrgory	Topic	Score	Remark
Docker Hub (20)	創建一個 Docker Hub Repo，並且有一個名為 2025cloud 的專案	10	必須要是 Public，助教才可以點選 Repo 名稱必須是 2025cloud
	專案內有兩個以上 (≥ 2) 的 Container Image	10	Tag/大小 都不限定，有兩個檔案即可
Git Repo README & Dockerfile (30)	GitHub 專案內有 Dockerfile	10	有檔案即可，內容與否取決於你的應用程式
	README 有清楚描述如何透過 docker build 打包你的應用程式	10	創一個 issue，open 狀態
	README 有清楚描述如何透過 docker run 運行你Container Image	10	可以是直接放到 dockerhub 上的，也可以是前述 docker build 結果。總之就是要讓使用者有東西可以參考
GitHub Action 整合 (40)	GitHub Action 有辦法去自動執行 Docker Build	10	1. 不限定觸發條件，有相關 Action 即可 Image Tag 沒有規定 2. 遞交 GH Action 連結
	GitHub Action 有辦法去自動執行 Docker Push 將產生好的 Image 給推到前述的 2025cloud repo	10	1. 遞交 GH Action 連結，過程要可以看到是推到哪個 Dockerhub/Repo/Tag 2. 搭配前述 DockerHub 的連結，要確認 Tag 有在上面
	產生一個 Pull Request，裡面故意寫壞 Dockerfile 讓他失敗，GitHub Action 要有辦法偵測壞掉	20	1. 請遞交一個 Pull Request 的連結 2. PR 內敘述如何改壞 Dockerfile 3. GH 下要可以看到自動化執行失敗
文件詳述 (10)	README 以圖文的方式描述目前專案自動化產生 Container Image 的邏輯，以及 Tag 的選擇邏輯	10	1. 什麼樣的情況會產生 Docker Image/Tag 的設計流程 2. 重點是能夠講清楚自己的設計，沒有對錯