DevOpsDays Taipei

2025

別再用 Mock 騙自己了

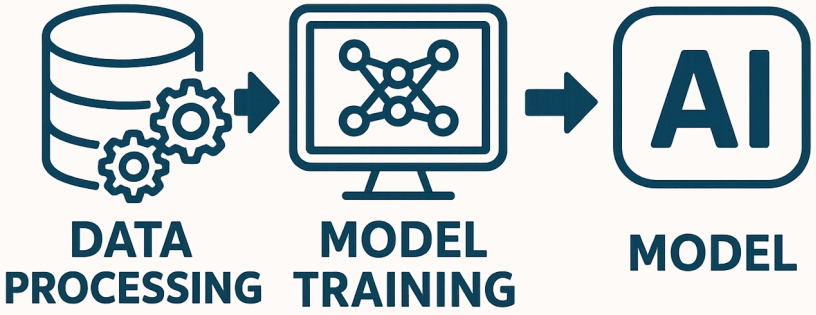
Testcontainers 在 CI/CD 的實戰攻略

陳柏妤 (Farah Chen)

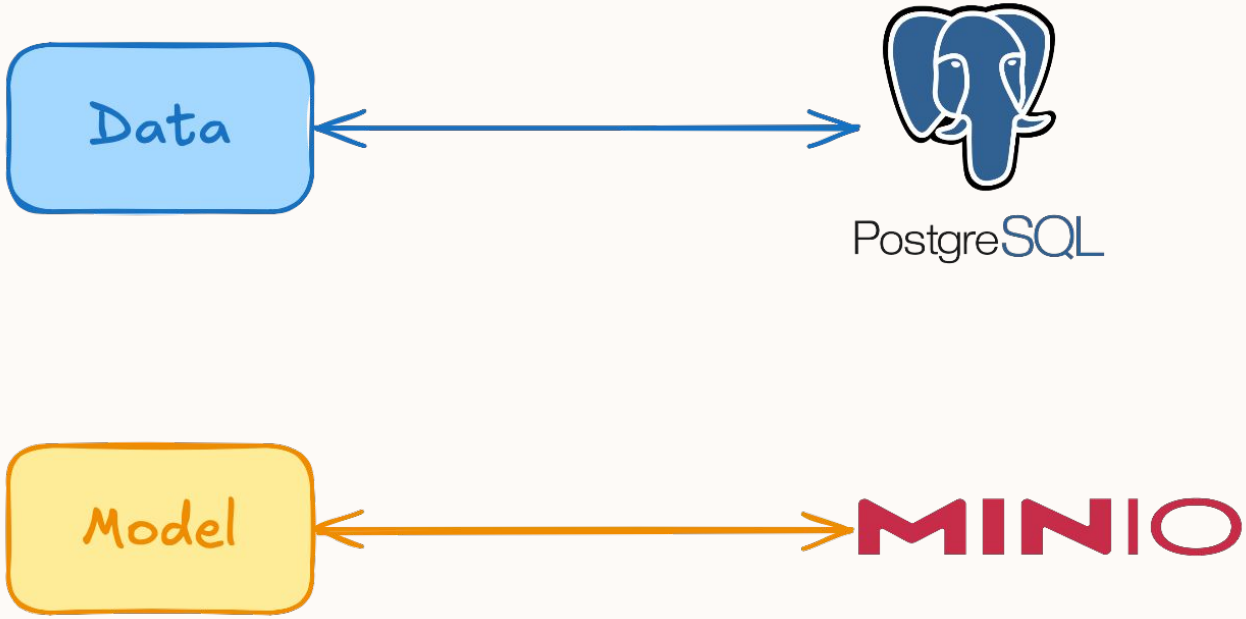
- 玉山銀行 智能金融處
- DevOps Engineer

關於玉山銀行智能金融處

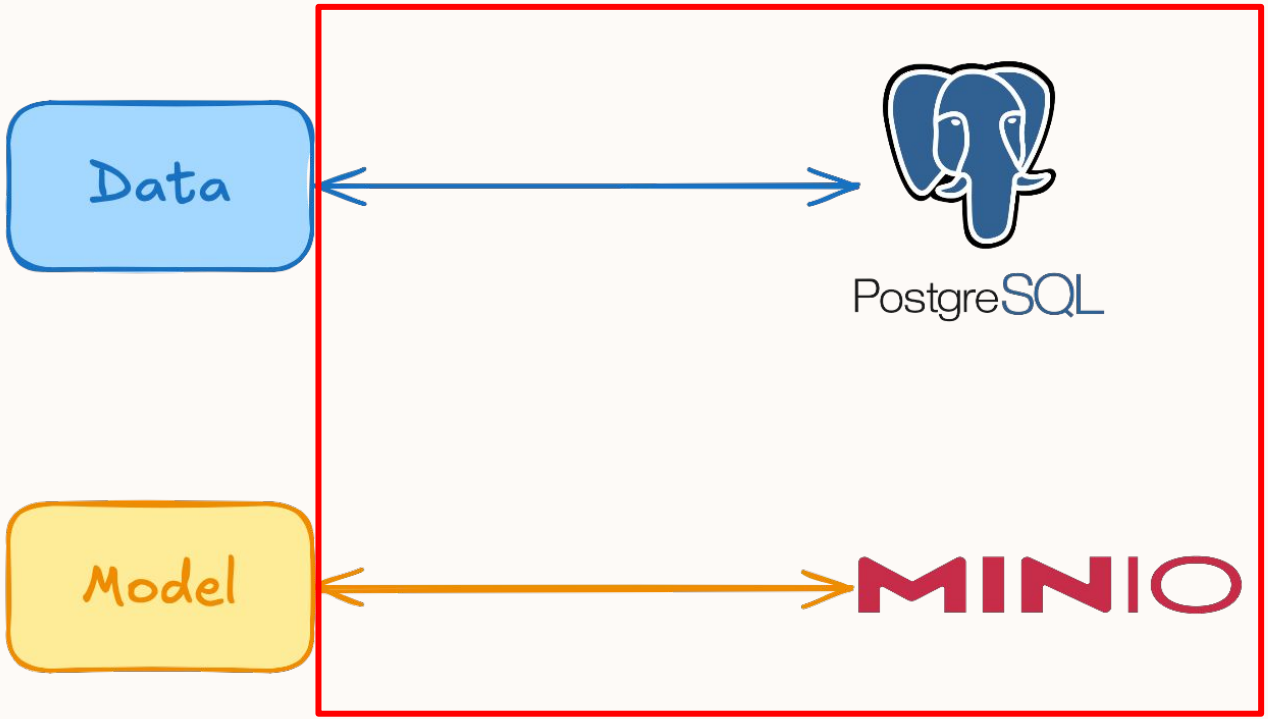
- 玉山銀行的 AI 中樞
- 以機器學習方法, 推動行內數位轉型
- 票據影像辨識、貸款評分模型、信用卡冒用偵測、GENIE



關於 Model 的產生



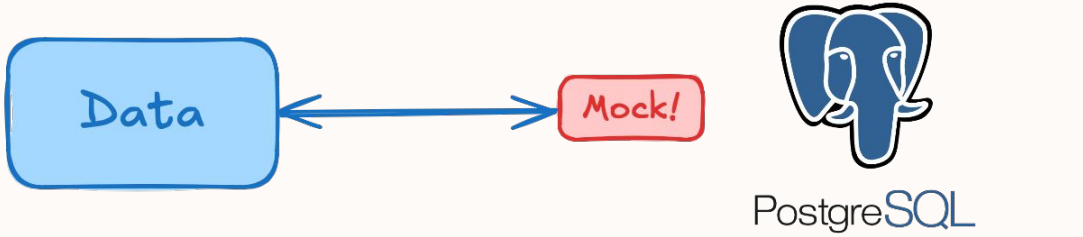
關於 Model 的產生

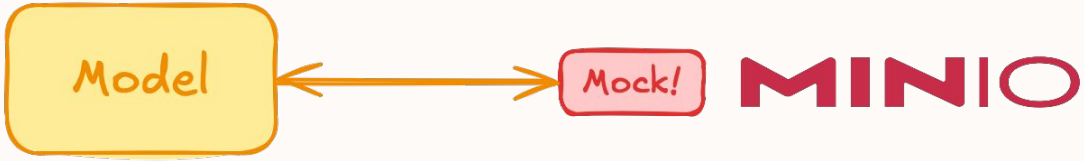


怎麼做 pytest？

Mock

用 **Mock 物件** 取代 **對外部元件的連線**





單元測試只用 Mock，足夠嗎？

情境：測試上傳 MinIO

```
mock_client.put_object.assert_called_with(...)
```

✅Mock 可以驗證的情境

- 方法有被正確調用
- 傳入的參數格式符合預期

Mock 的不足

❎ Mock 不能驗證的情境

- Bucket 不存在
- 資料長度錯誤導致上傳失敗
- MinIO Client 連線錯誤
- MinIO API 格式變動

Mock 的不足

❌ Mock 不能驗證的情境

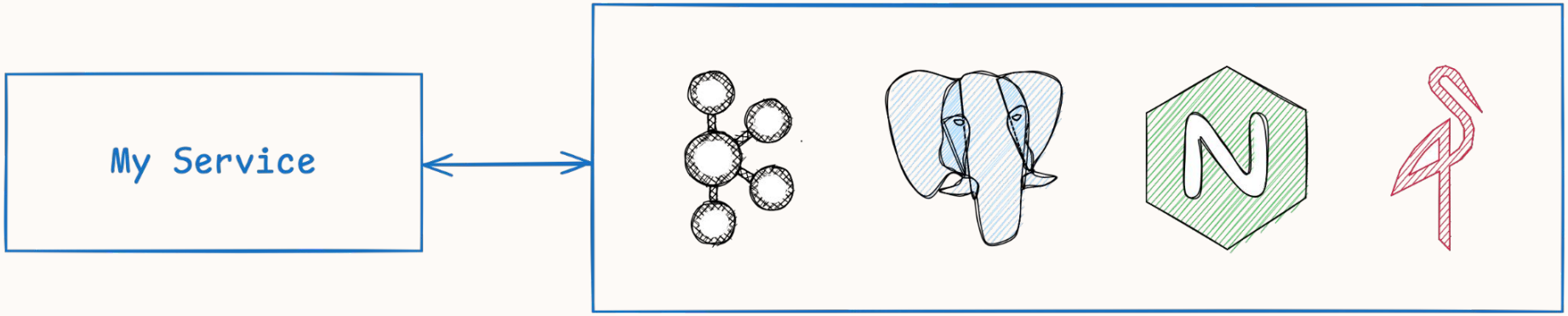
- Bucket 不存在
- 資料長度錯誤導致上傳失敗
- MinIO Client 連線錯誤
- MinIO API 格式變動

實際與元件的互動

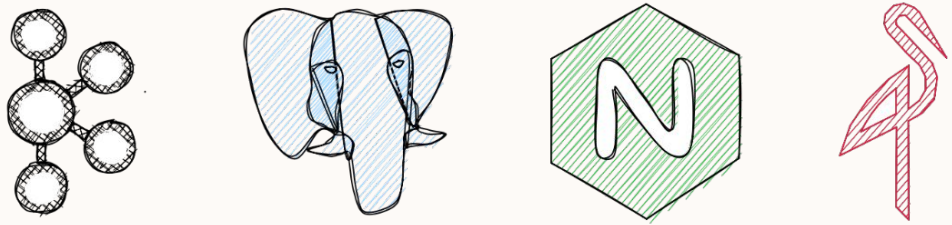
Problem

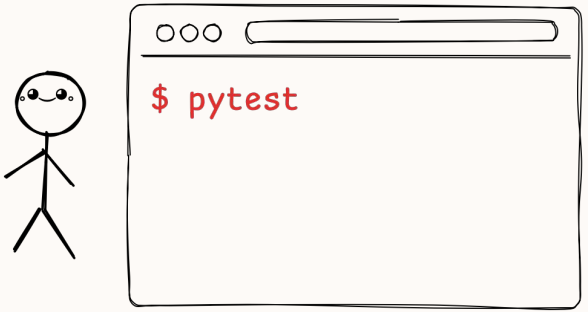
資料分析團隊需測試與外部元件的互動

被 Mock 隔離後難以正確測試

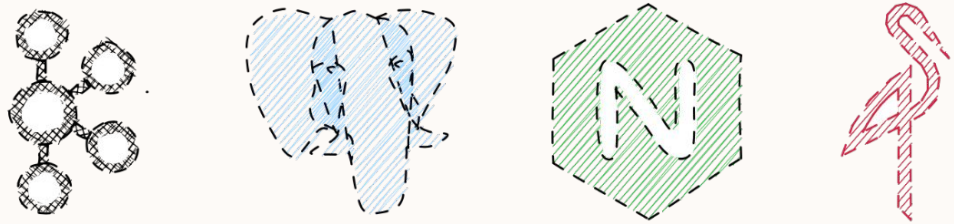

My Service

My Service

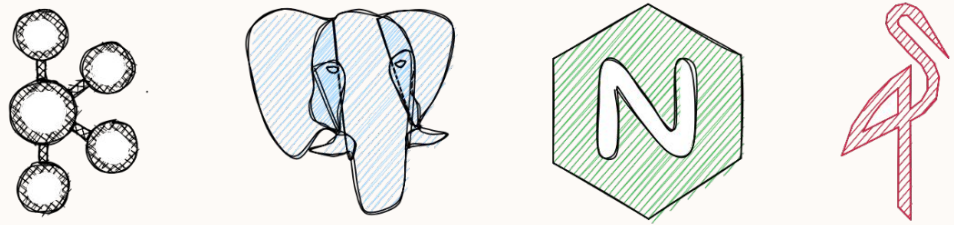




Test

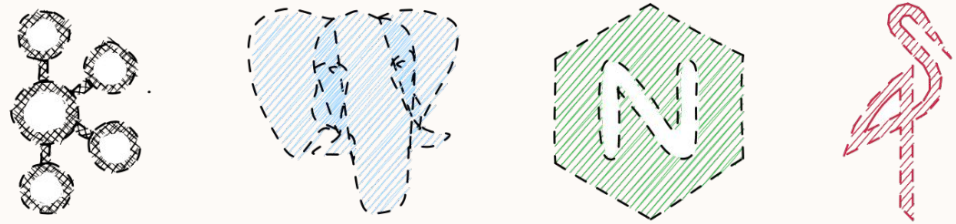


My Service



Destroy

Test



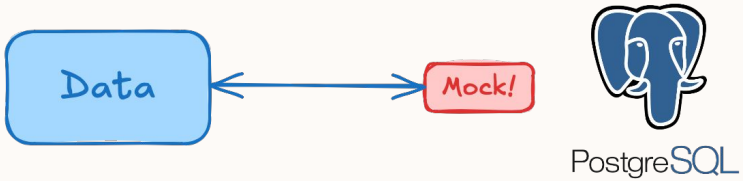
Solution : **Testcontainers**

What’s Testcontainers

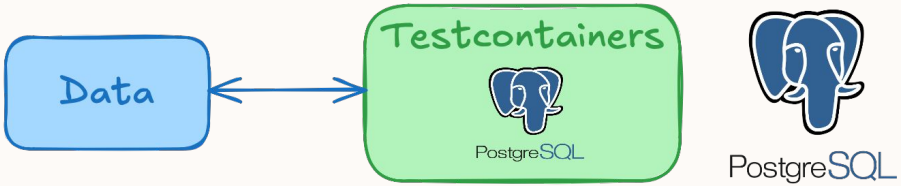
- 一個 library, 提供簡單輕量的 API
- 在測試時**把程式相依的外部元件 啟在容器中**，
  讓測試程式與那些元件互動

Change when doing pytest

Before：用 **Mock 物件** 取代 **對外部元件的連線**



After：用 **對測試容器的連線** 取代 **對外部元件的連線**



對使用者而言

一個「友善的」測試方法，應該以什麼流程運作

容易設定

每次測試遵循相同的初始化配置
確保測試結果可重現

啟動可靠

不只是「起得來」，還要確認
service 已經 ready，可以進行測試

測試一致

無論環境，都能執行一致的測試流程

資源清理

環境自動清理，不會留下髒資料、殘留容器，
影響後續測試

容易設定

每次測試遵循相同的初始化配置
確保測試結果可重現

啟動可靠

不只是「起得來」，還要確認
service 已經 ready，可以進行測試

測試一致

無論環境，都能執行一致的測試流程

資源清理

環境自動清理，不會留下髒資料、殘留容器，
影響後續測試

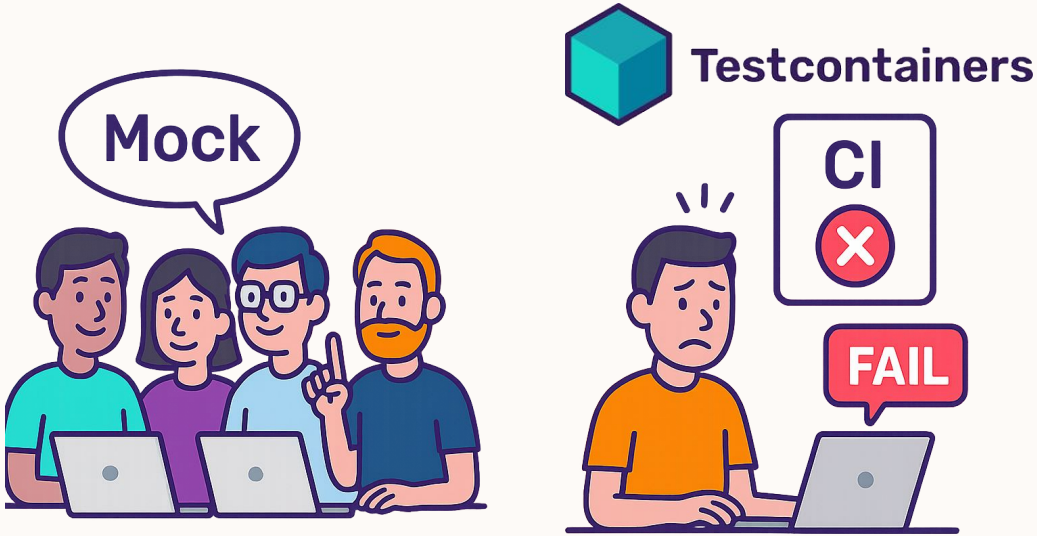
開發與 **CI** 環境皆能執行

是推廣測試工具的第一步

以 CI 為基準，建立團隊一致的驗證流程

如果測試工具無法在 CI 環境執行，就無法成為整個團隊共同依賴的測試基礎。

它會淪為**「個人實驗工具」**，無法成為**「團隊的品質保證機制」**。

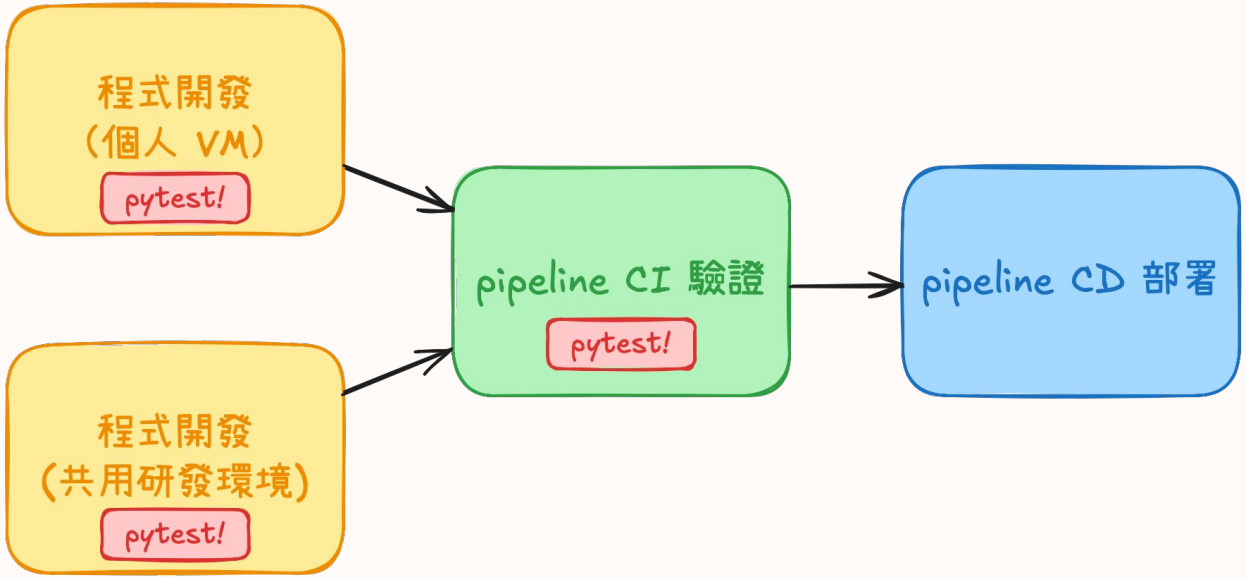


能不能在 CI pipeline 執行

不只是技術選項，而是價 值能否普及的關鍵

從開發到部署，

Testcontainers 落地的三個場景

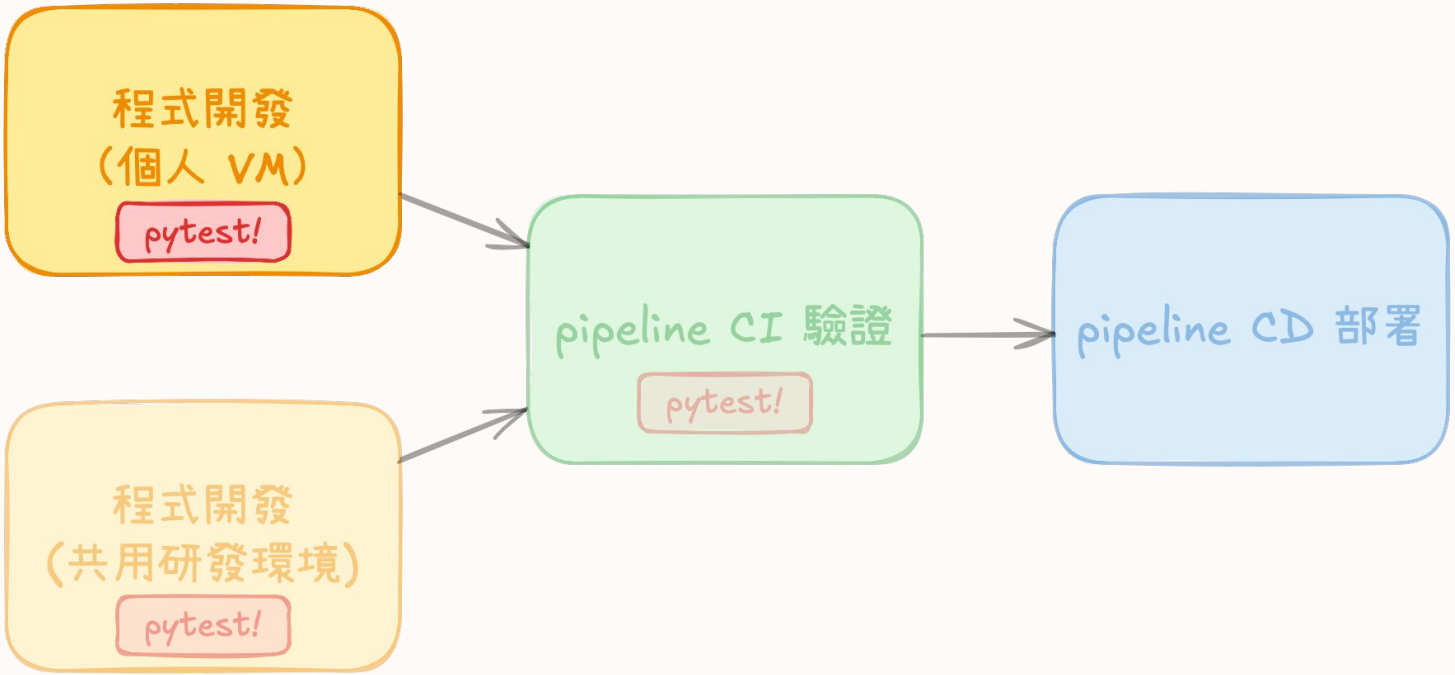


目標

在需要執行單元測試的環境

都能執行 Testcontainers

新手村:VM


程式開發
(個人 VM)
pytest!
程式開發
(共用研發環境)
pytest!
pipeline CI 驗證
pytest!
pipeline CD 部署

新手村:VM

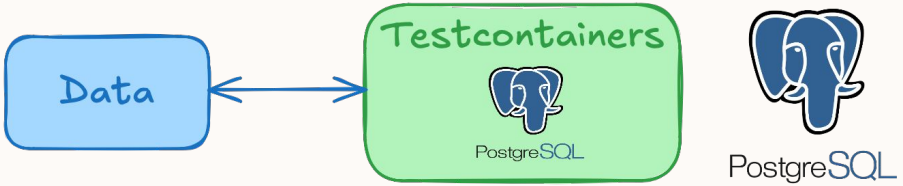
- 安裝 docker 與 Testcontianers 套件
- 於 fixture 定義 Testcontainers DB 連線

```
@pytest.fixture()
def testcontainer_db_conn():
    postgres_container = PostgresContainer(image='postgres:14')
    postgres_container.with_volume_mapping(os.path.abspath('init.sql'), <path_in_testcontainers>)
    with postgres_container as postgres:
        db_params = postgres.<conn_info>
        conn = psycopg2.connect(**db_params)
        yield conn
```

- 對 Testcontainers DB 測試資料寫入

```
def test_insert_db_success(testcontainer_db_conn):
    assert insert_db(testcontainer_db_conn) == "insert_success"
```

新手村:VM



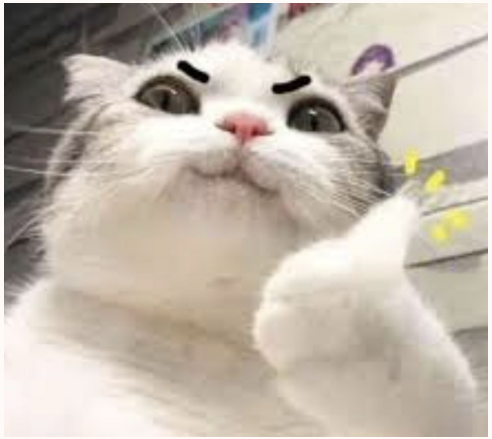
- 安裝 docker 與 Testcontianers 套件
- 於 fixture 定義 Testcontainers DB 連線

```
@pytest.fixture()
def testcontainer_db_conn():
    postgres_container = PostgresContainer(image='postgres:14')
    postgres_container.with_volume_mapping(os.path.abspath('init.sql'), <path_in_testcontainers>)
    with postgres_container as postgres:
        db_params = postgres.<conn_info>
        conn = psycopg2.connect(**db_params)
        yield conn
```

- 對 Testcontainers DB 測試資料寫入

```
def test_insert_db_success(testcontainer_db_conn):
    assert insert_db(testcontainer_db_conn) == "insert_success"
```

新手村:VM



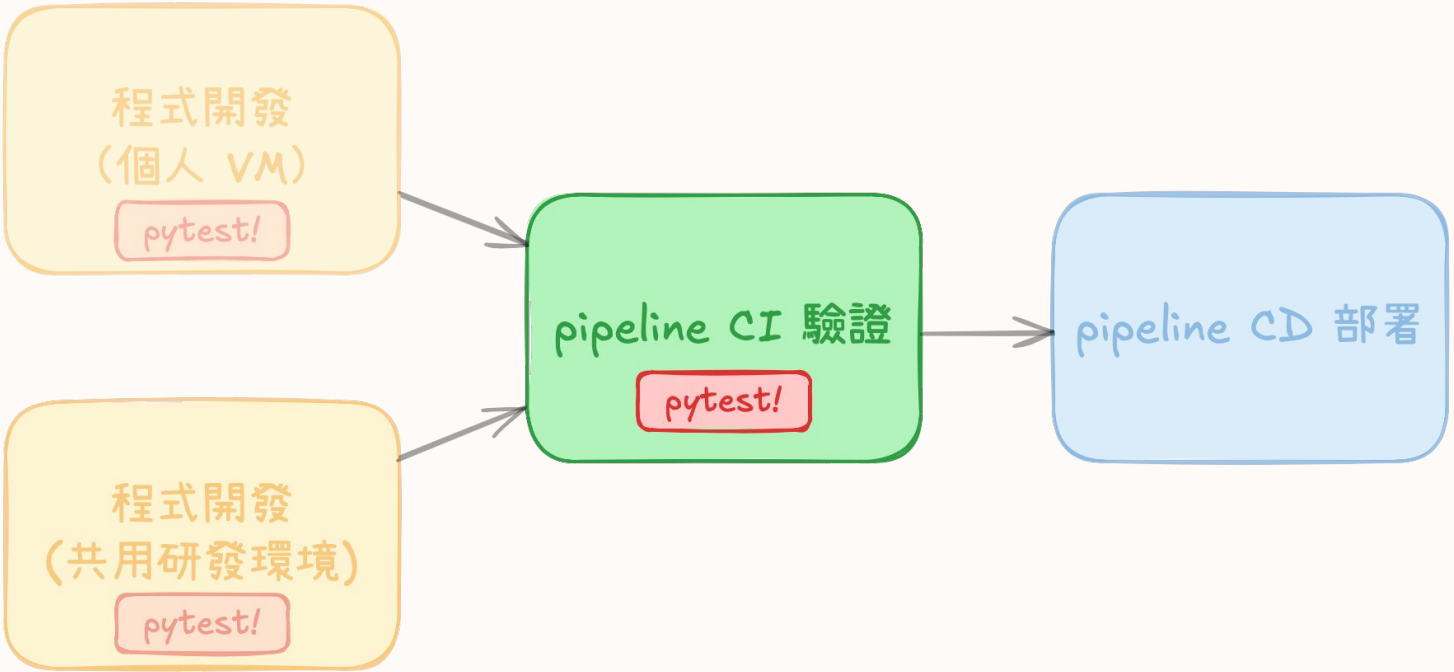
- 安裝 docker 與 Testcontianers 套件
- 於 fixture 定義 Testcontainers DB 連線

```
@pytest.fixture()
def testcontainer_db_conn():
    postgres_container = PostgresContainer(image='postgres:14')
    postgres_container.with_volume_mapping(os.path.abspath('init.sql'), <path_in_testcontainers>)
    with postgres_container as postgres:
        db_params = postgres.<conn_info>
        conn = psycopg2.connect(**db_params)
        yield conn
```

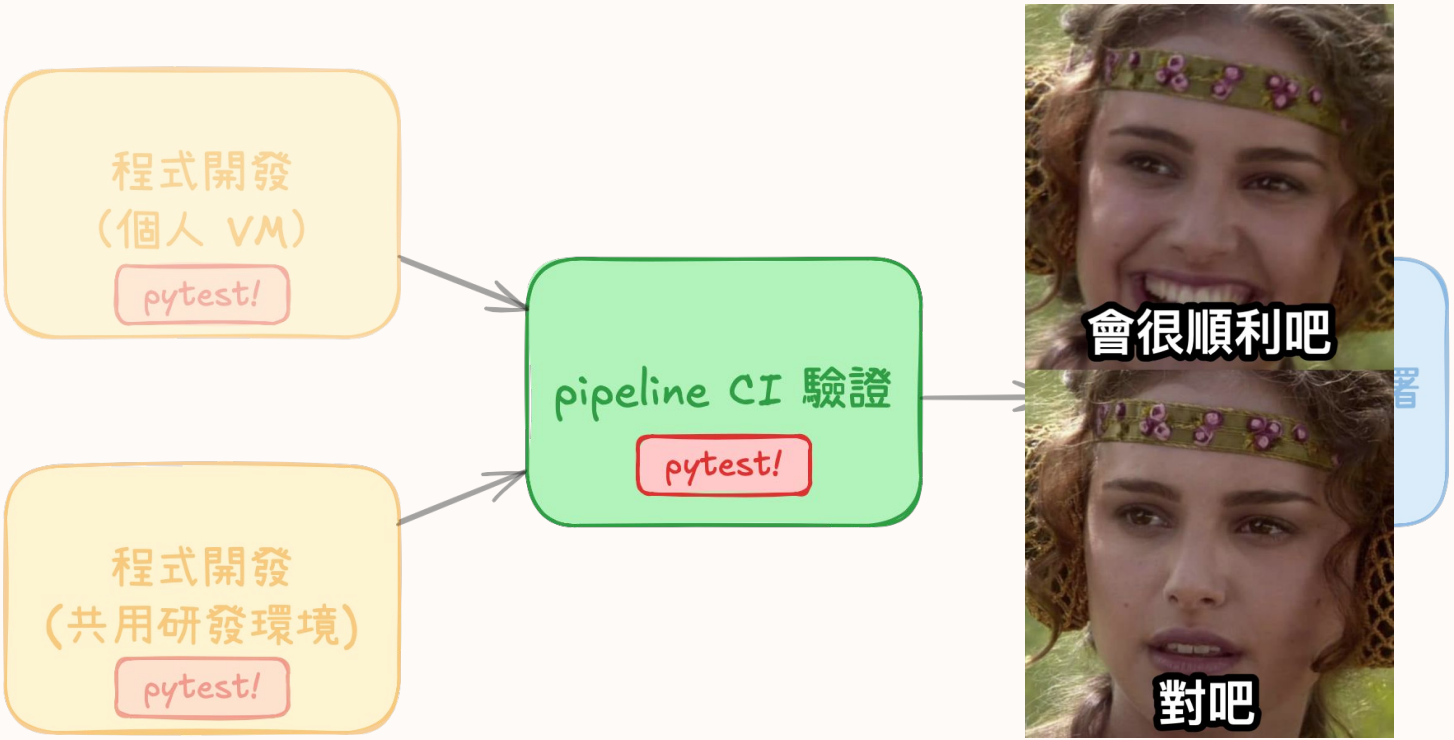
- 對 Testcontainers DB 測試資料寫入

```
def test_insert_db_success(testcontainer_db_conn):
    assert insert_db(testcontainer_db_conn) == "insert_success"
```

Level 2：pipeline 執行環境



Level 2：pipeline 執行環境



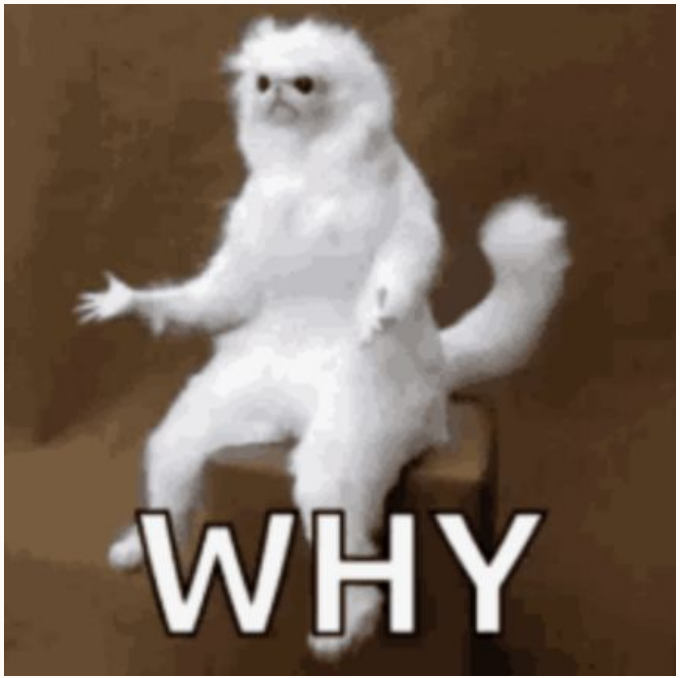
出事了！

```
wrapped = <bound method DockerContainer.get_exposed_port of <testcontainers.postgres.Postgre
instance = <testcontainers.postgres.PostgresContainer object at 0x7f43c1d24580>
args = (5432,), kwargs = {}

    @wrapt.decorator
    def wrapper(wrapped, instance, args, kwargs):
        exception = None
        logger.info("Waiting to be ready...")
        for attempt_no in range(config.MAX_TRIES):
            try:
                return wrapped(*args, **kwargs)
            except transient_exceptions as e:
                logger.debug(f"Connection attempt '{attempt_no + 1}' of '{config.MAX_TRIES + 1}' "
                             f"failed: {traceback.format_exc()}")
                time.sleep(config.SLEEP_TIME)
                exception = e
>       raise TimeoutException(
            f'Wait time ({config.MAX_TRIES * config.SLEEP_TIME}s) exceeded for {wrapped.__name__}'
            f'(args: {args}, kwargs {kwargs}). Exception: {exception}'
        )
```

出事了！

```
wrapped = <bound method DockerContainer.get_exposed_port of <testcontainers.postgres.Postgre
instance = <testcontainers.postgres.PostgresContainer object at 0x7f43c1d24580>
args = (5432,), kwargs = {}

    @wrapt.decorator
    def wrapper(wrapped, instance, args, kwargs):
        exception = None
        logger.info("Waiting to be ready...")
        for attempt_no in range(config.MAX_TRIES):
            try:
                return wrapped(*args, **kwargs)
            except transient_exceptions as e:
                logger.debug(f"Connection attempt '{attempt_no + 1}' of '{config.MAX_TRIES + 1}' "
                             f"fail
                time.sleep(config.S
                exception = e
>       raise TimeoutException(
            f'Wait time ({config.MA
            f'(args: {args}, kwargs
        )
```

TimeoutException：
Testcontainers 沒有正確啟動導致連線超時


WHY

痛點

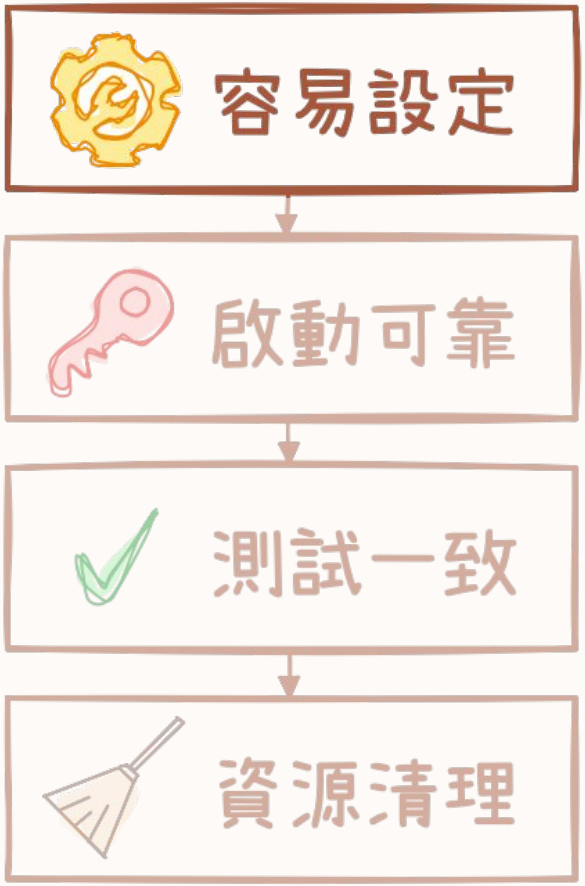
以容器運行的 CI pipeline 環境

都需要處理 Volume Mount 時的搬檔問題

痛點

以容器運行的 CI pipeline 環境

都需要處理 **Volume Mount** 時的搬檔問題



Volume Mount

Port

Env

Volume Mount

- 將檔案掛載到容器內的特定位置
- 使用 Volume Mount 的時機
  - 初始化 DB 的 init.sql
  - 配置 Nginx 的 config file

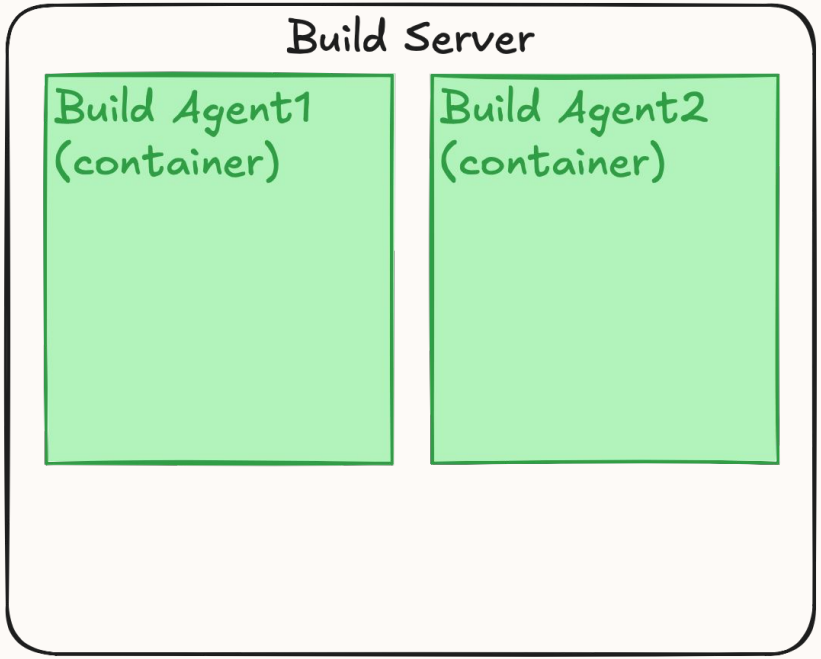
```
@pytest.fixture()
def testcontainer_db_conn():
    postgres_container = PostgresContainer(image='postgres:14')
    postgres_container.with_volume_mapping(os.path.abspath('init.sql'), <path_in_testcontainers>)
    with postgres_container as postgres:
        db_params = postgres.<conn_info>
        conn = psycopg2.connect(**db_params)
        yield conn
```

痛點

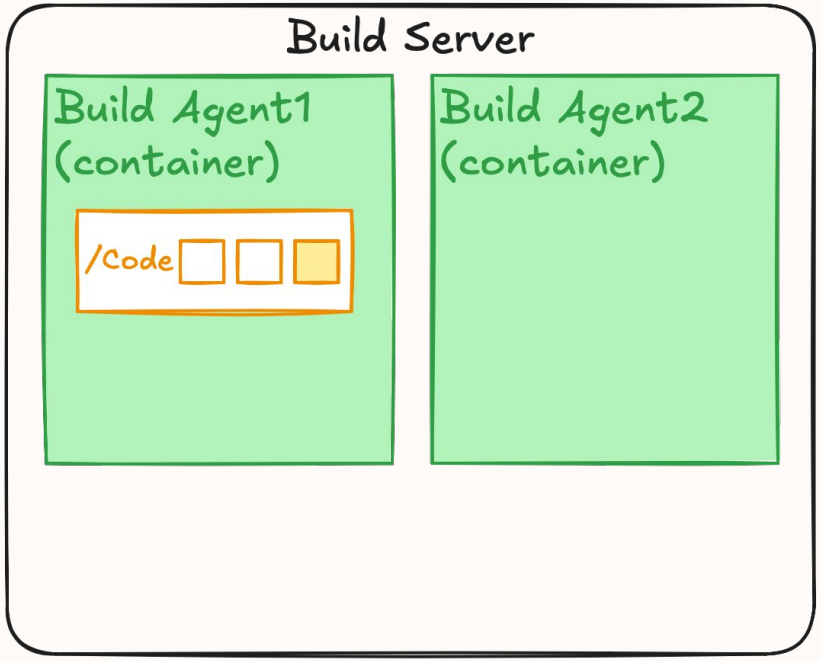
以容器運行的 CI pipeline 環境

都需要處理 Volume Mount 時的搬檔問題

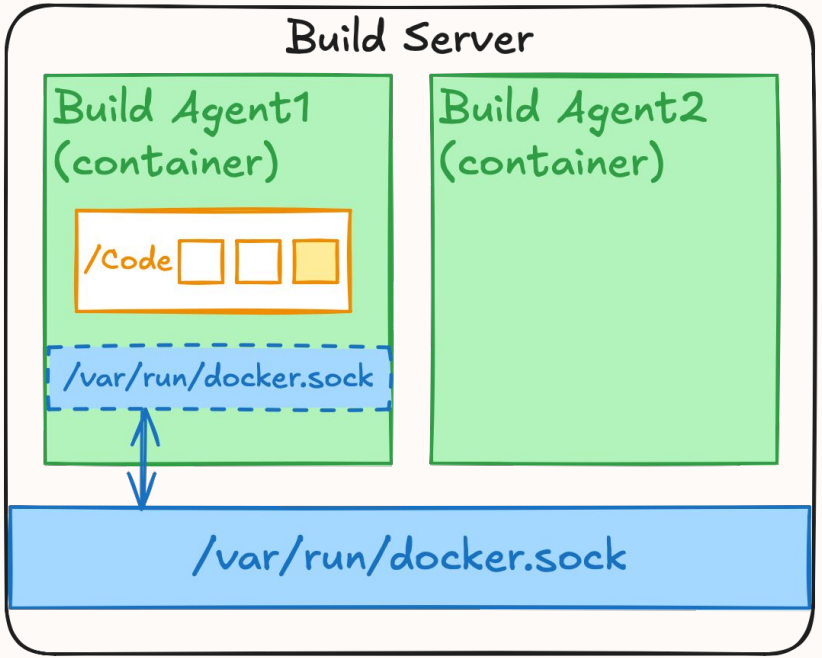
容器化 Pipeline 執行環境



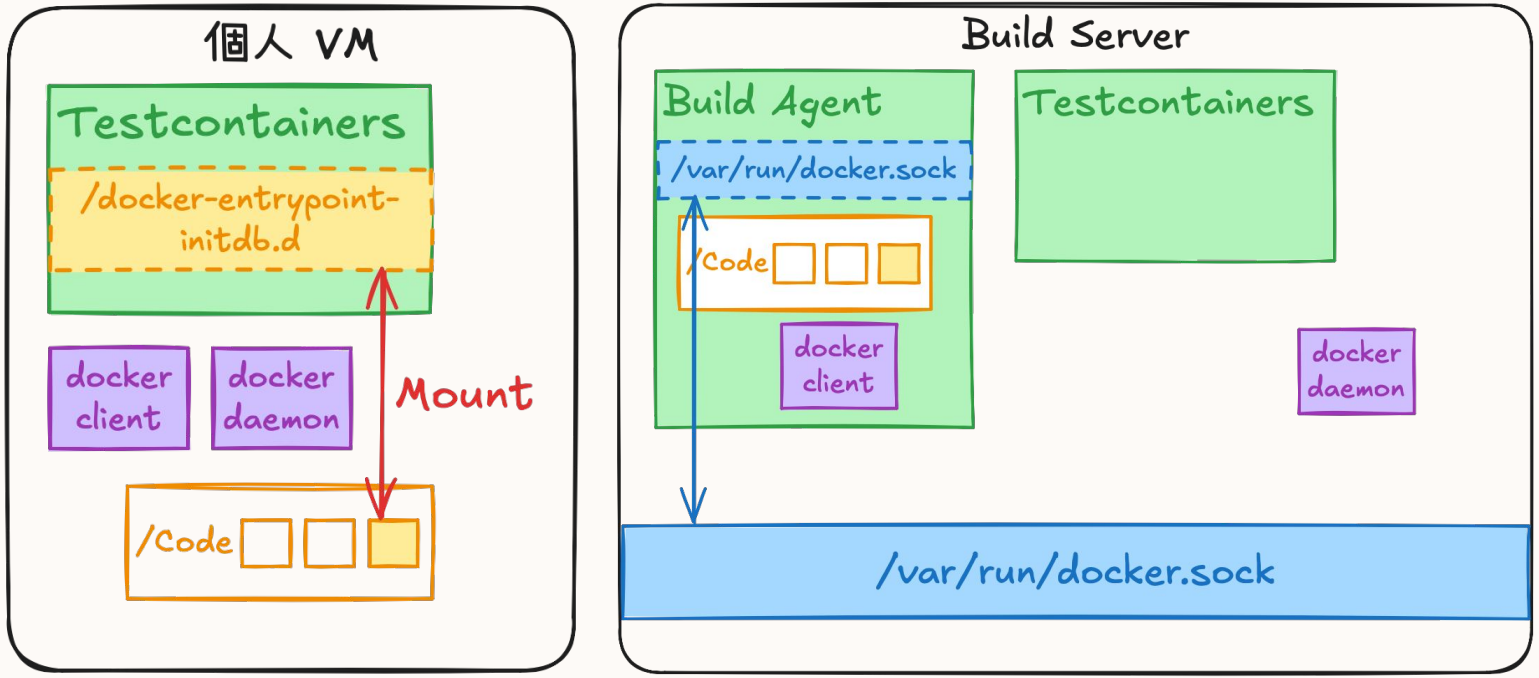
容器化 Pipeline 執行環境



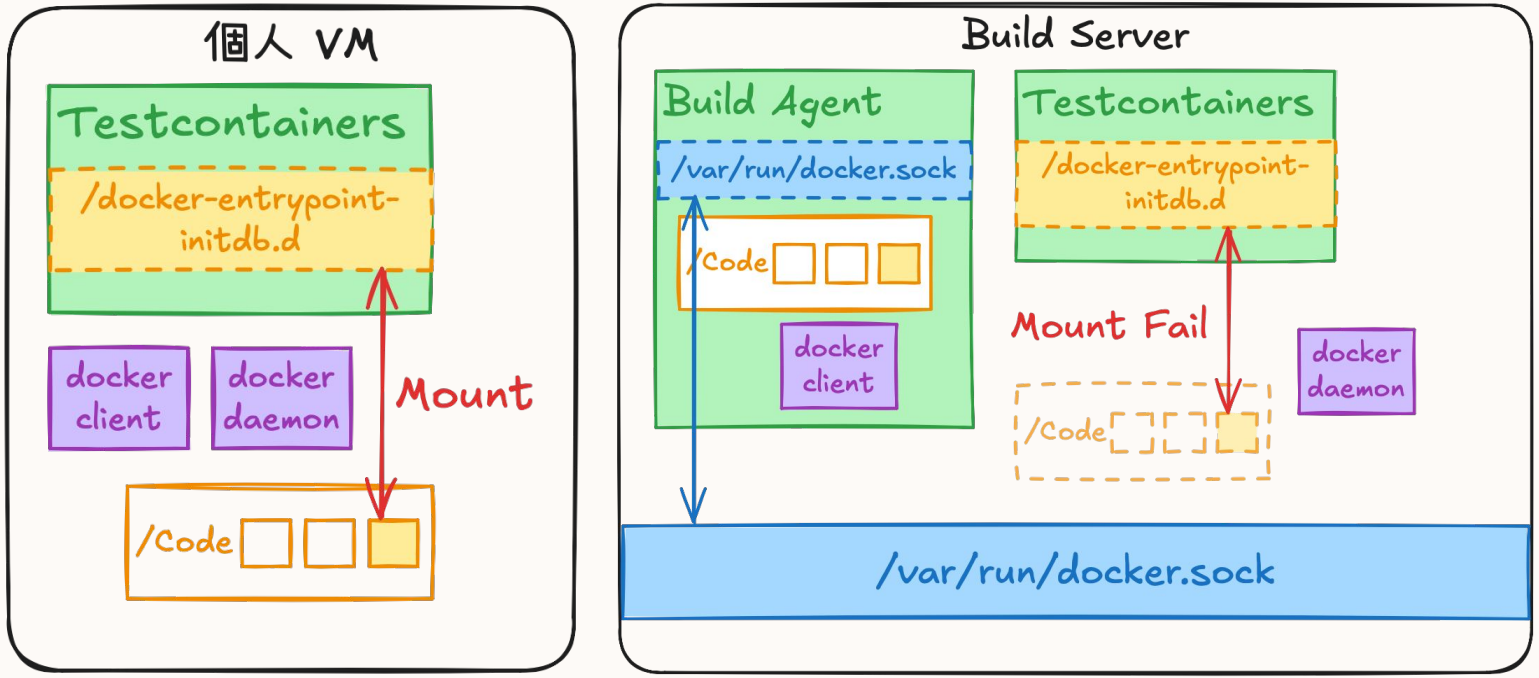
容器化 Pipeline 執行環境



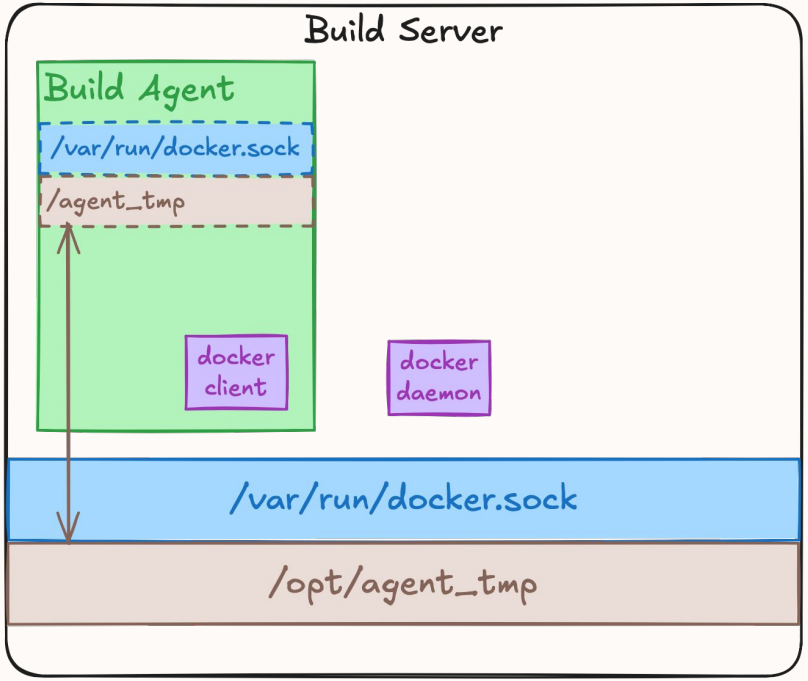
根因：Docker in Docker



根因：Docker in Docker



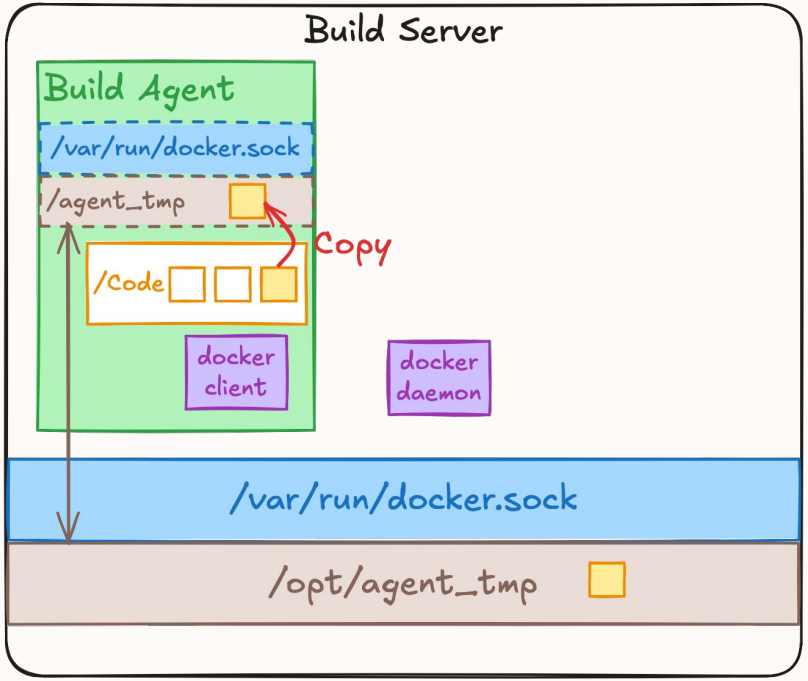
解法：Pipeline Runtime 搬檔



1. 初始化 Agent 時

   Mount 指定路徑到 host 上

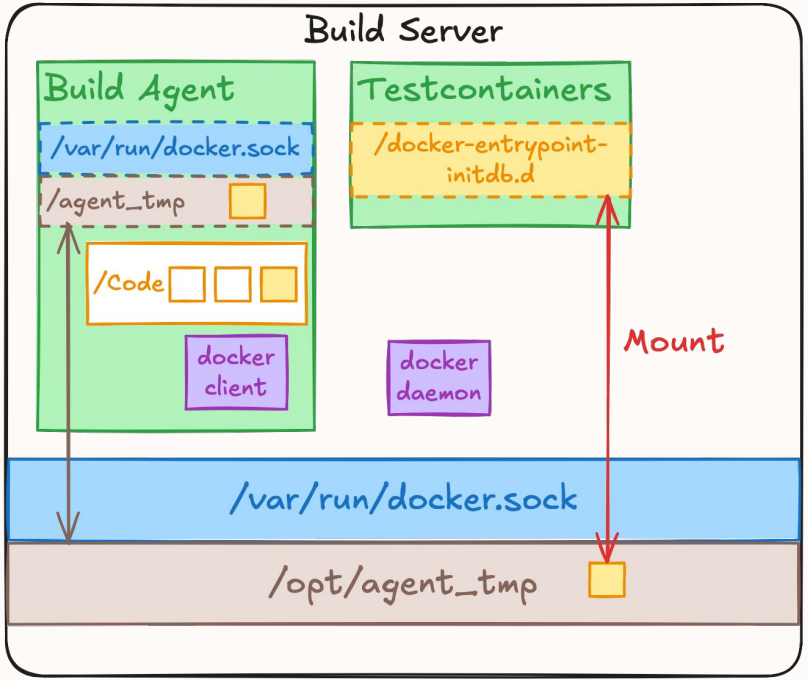
解法：Pipeline Runtime 搬檔



2. pipeline 執行 pytest 之前

   將 mounted file 複製到該路徑

解法：Pipeline Runtime 搬檔



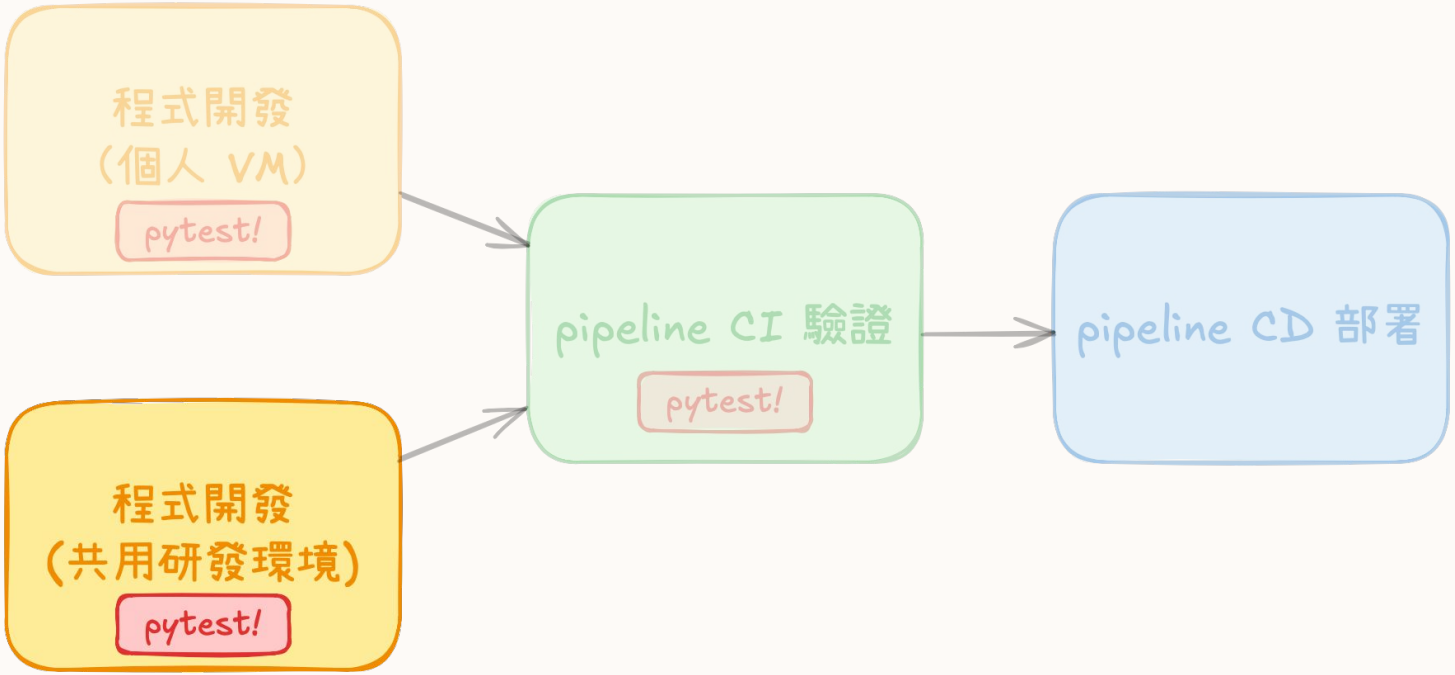
3. 成功將檔案 Mount 到測試容器內

結論

在容器內部執行 Testcontainers

- 配置 docker in docker（才能執行 docker 指令）
- 執行搬檔機制（避免 Volume Mapping 失敗）

Level 3 : 共用研發環境



痛點

當前環境**不足以**支援啟動 Testcontainers

共用研發環境

- 底層採用 K8s，上層提供 Jupyter Notebook 的開發環境
- 每個 user 是一個 pod
- 此環境提供 **80+** 使用者開發

如何讓使用者在開發環境啟動 Testcontainer？

在 Pod 內部啟 Testcontainers？

在 Pod 外部啟 Testcontainers？

如何讓使用者在開發環境啟動 Testcontainer？

在 Pod 內部啟 Testcontainers？

破壞 Pod 輕量化的設計

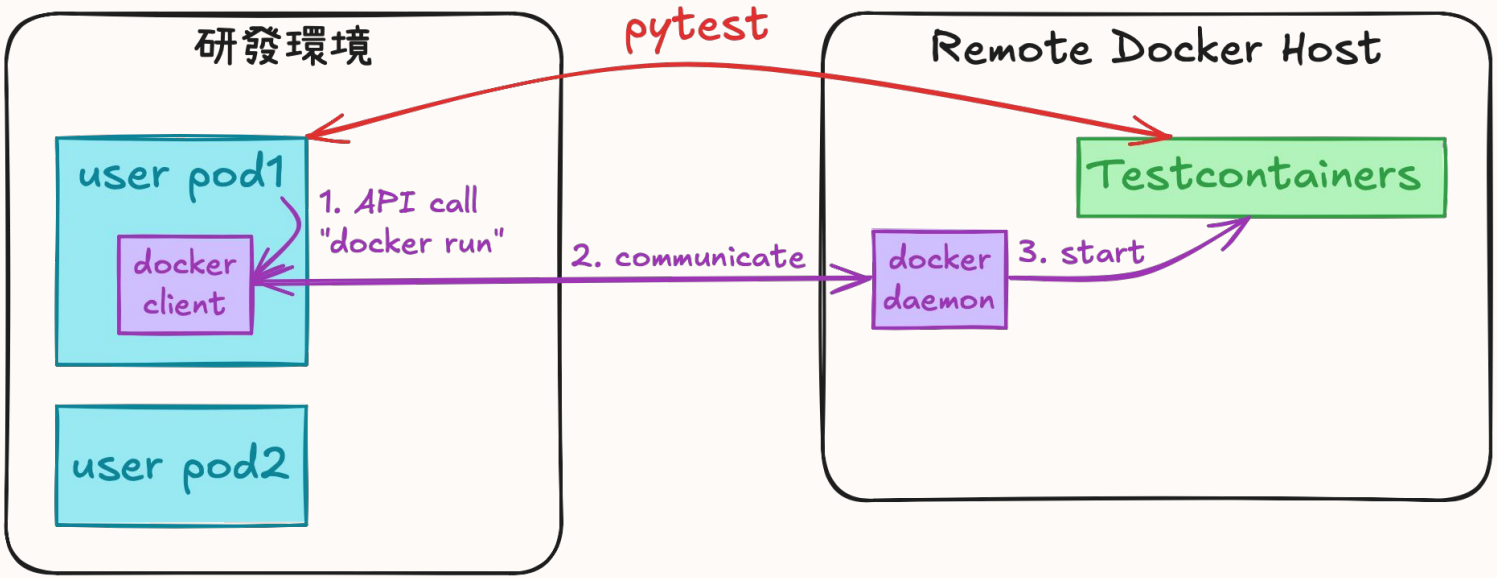
多個 user 就要多個 image 儲存空間

在 Pod 外部啟 Testcontainers？

會影響 k8s 的 container runtime

解法：Remote Docker Host

Remote Docker Host：將 container 啟在另一台 VM 上



使用 Remote Docker Host 的優點

1. **最小化當前環境需要的改動** 、減少推行成本
    a. 只需加入環境變數 $DOCKER_HOST
2. image 儲存於遠端 VM、可多位使用者共用
3. 啟動的 container 不會增加研發環境的負擔

使用 Remote Docker Host 的優點

1. **最小化當前環境需要的改動** 、減少推行成本
    a. 只需加入環境變數 $DOCKER_HOST
2. image 儲存於遠端 VM、可多位使用者共用
3. 啟動的 container 不會增加研發環境的負擔

Testcontainers 長在另一台機器上 → 要處理搬檔

如何把檔案送到 Remote Docker Host

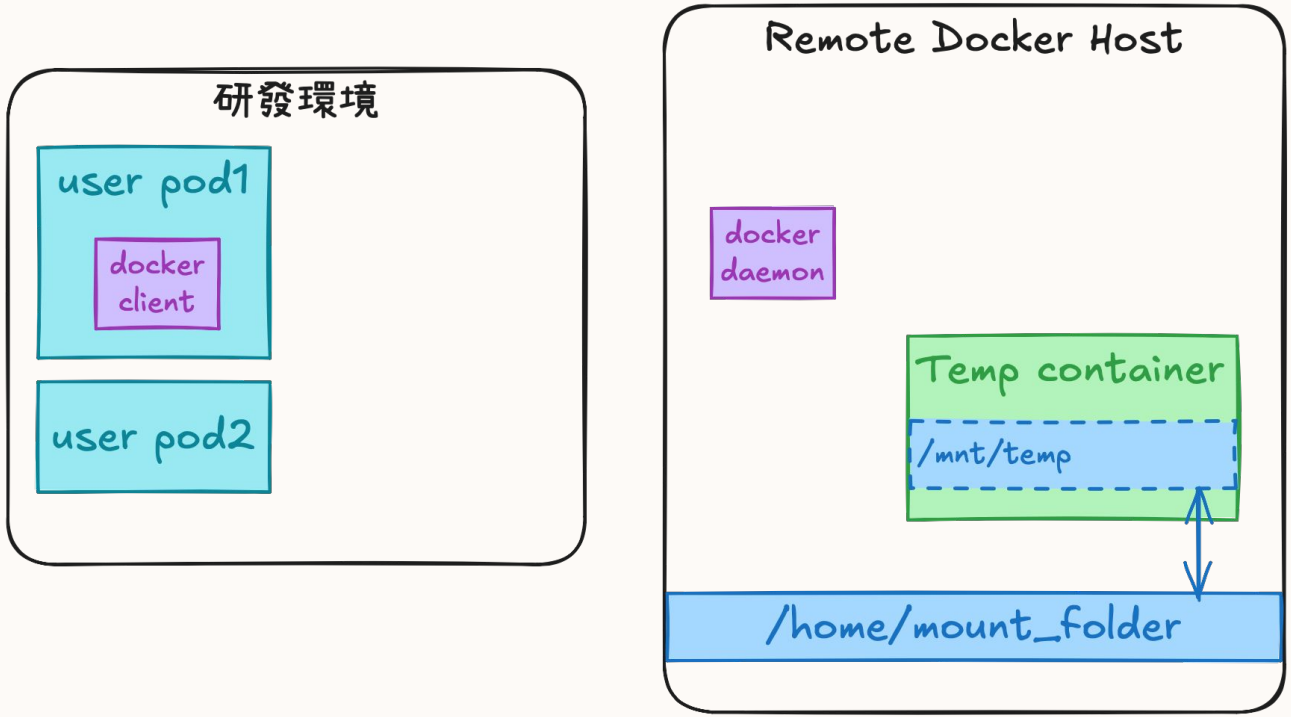
SCP

(Secure Copy Protocol)

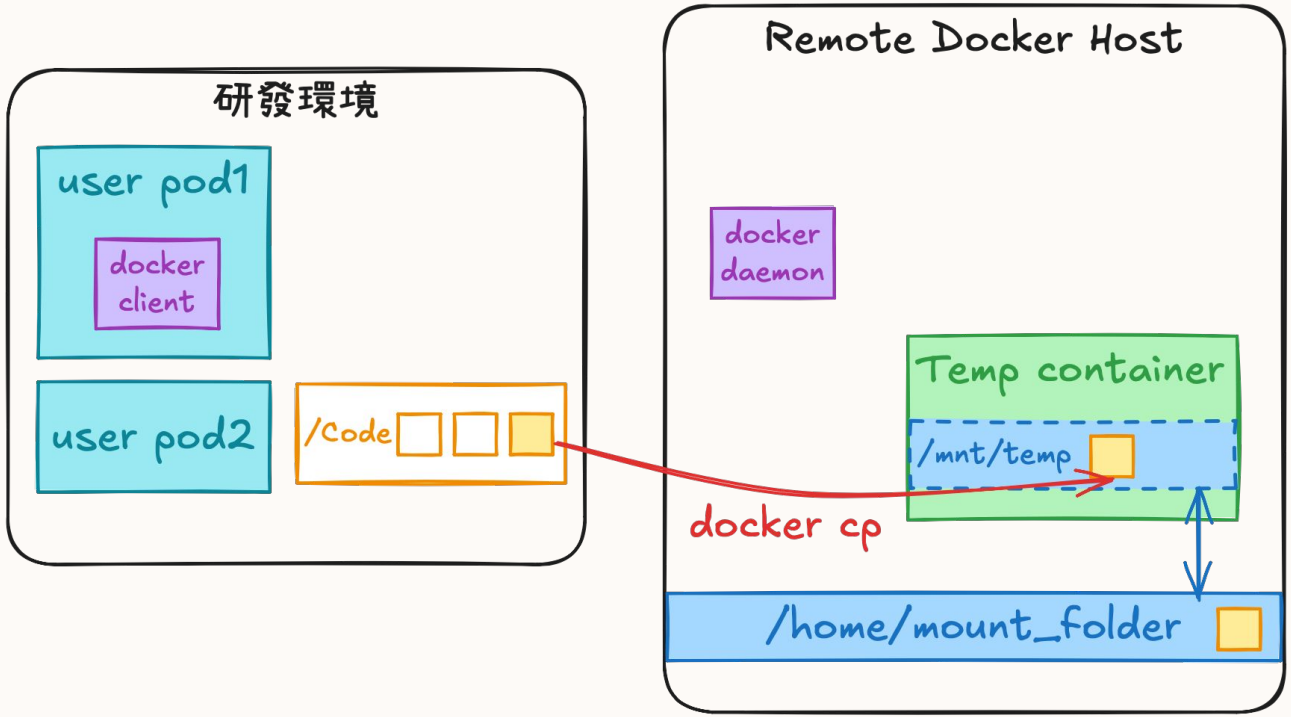
NFS

(Network File System)

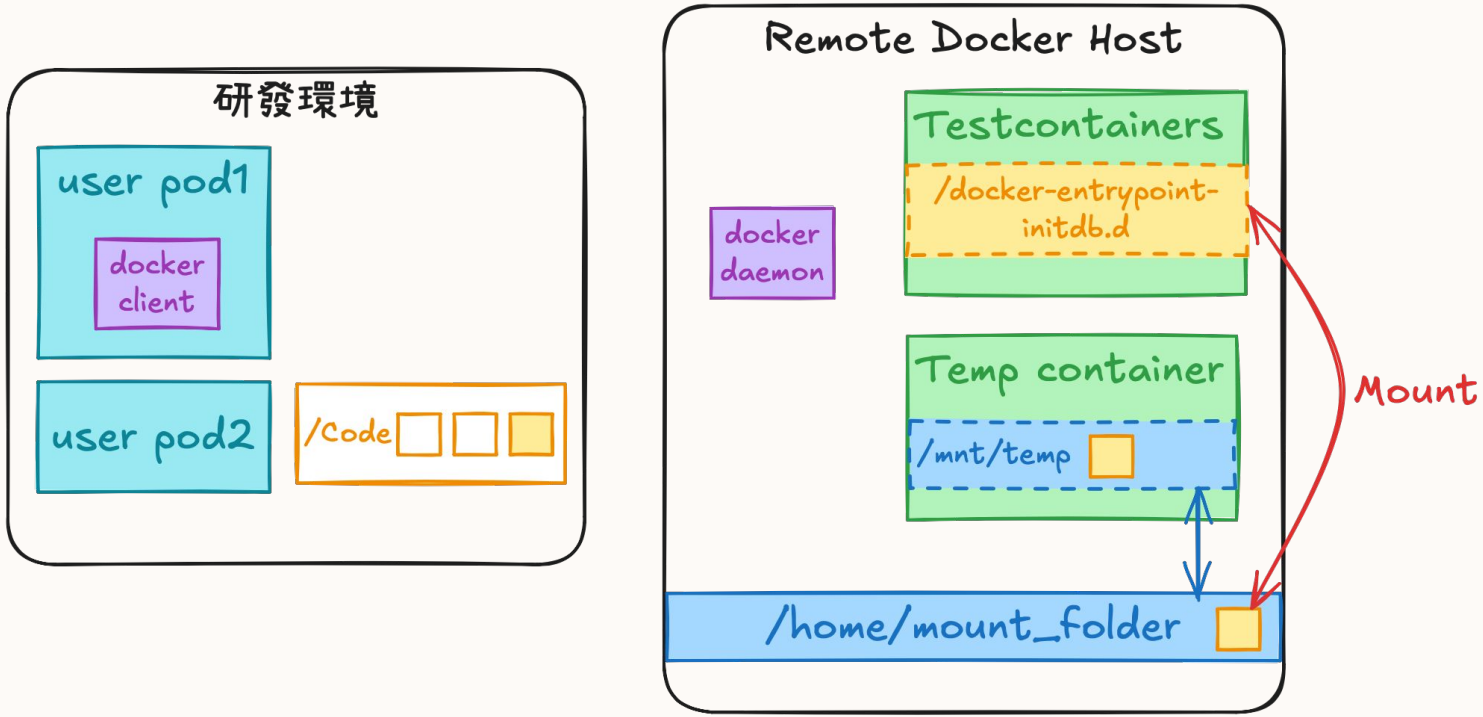
解法：Temp Container + docker cp



解法：Temp Container + docker cp



解法：Temp Container + docker cp



為何需要 Temp Container

- docker cp 只能複製到**已經存在的 container** 內
- volume mount 是在 container **啟動前** 的設定

結論

當前執行環境限制、難以啟動 Testcontainer

希望最小化對當前環境的改動

- 使用 Remote Docker Host 啟在另一台 VM 上
- temp container + docker cp 可解決搬檔問題

Takeaway

- 當程式關注「**與外部元件的互動**」時，Testcontainers 可以彌補 Mock 的不足、提升測試可靠性
- 開發機資源有限時，可透過 **Remote Docker Host** 解決
- 在跨機器或容器的環境下，**Volume Mapping 是導入過程的最大坑**

～感謝聆聽～