Bộ môn An toàn Thông tin – Khoa Mạng máy tính và Truyền thông
Trường Đại học Công nghệ Thông tin – Đại học Quốc gia thành phố Hồ Chí Minh

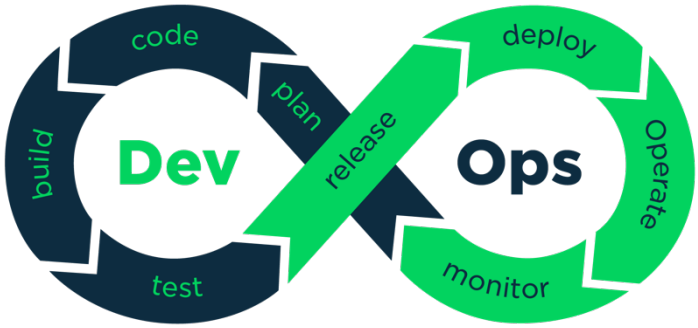
1

Lab

AUTOMATING EVERYTHING AS CODE

DevOps/CI-CD/Git

Thực hành môn Lập trình An toàn & Khai thác lỗ hổng phần mềm



Tháng 09/2021

Lưu hành nội bộ

<Nghiêm cấm đăng tải trên internet dưới mọi hình thức>

Mọi góp ý về tài liệu, vui lòng gửi về email inseclab@uit.edu.vn

INSECLAB

A. TỔNG QUAN

1. Mục tiêu

- Giới thiệu

2. Thời gian thực hành

- Thực hành tại lớp: **5** tiết tại phòng thực hành.
- Hoàn thành báo cáo kết quả thực hành: tối đa **13** ngày.

3. Môi trường thực hành

Sinh viên cần chuẩn bị trước máy tính với môi trường thực hành như sau:

> - 1 PC cá nhân với hệ điều hành tự chọn
> - Virtual Box hoặc **VMWare** + máy ảo **Linux** có Docker (hoặc Windows Subsystem Linux verision2)

B. THỰC HÀNH

1. Kiểm soát phiên bản phần mềm với Git

a) Thiết lập Git Repository

B1. Cấu hình thông tin người sử dụng và liên kết tài khoản trong local repository. Sử dụng tên bạn thay cho "Khoa Nghi", thay thế email bạn cho "hoangkhoa95@live.com"

```
┌──(kali㉿phaphajian)-[~]
└─$ git config --global user.name "Khoa Nghi"

┌──(kali㉿phaphajian)-[~]
└─$ git config --global user.email hoangkhoa95@live.com
```

B2. Bất cứ lúc nào, bạn có thể xem lại thiết lập này bằng lệnh git config –list

```
┌──(kali㉿phaphajian)-[~]
└─$ git config --list
user.name=Khoa Nghi
user.email=hoangkhoa95@live.com
```

B3. Tạo thư mục labs và di chuyển đến nó

```
┌──(kali㉿phaphajian)-[~]
└─$ mkdir labs && cd labs
```

B4. Tiếp tục tạo thư mục git-intro và di chuyển vào

```
┌──(kali㉿phaphajian)-[~/labs]
└─$ mkdir git-intro

┌──(kali㉿phaphajian)-[~/labs]
└─$ cd git-intro
```

B5. Sử dụng lệnh git init để khởi tạo thư mục hiện tại (git-intro) dưới dạng Git repository. Thông điệp được hiển thị cho biết rằng ta đã tạo một local repository trong dự án được chứa trong. thư mục .git. Đây là nơi chứa tất cả lịch sử thay đổi của code. Có thể thấy nó bằng lệnh ls -a.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git init
hint: Using 'master' as the name for the initial branch. This default
branch name
hint: is subject to change. To configure the initial branch name to use
in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:      git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this
command:
hint:
hint:      git branch -m <name>
Initialized empty Git repository in /home/kali/labs/git-intro/.git/

┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ ls -a
.  ..  .git
```

B6. Khi ta làm việc trog dự á, ta sẽ muốn kiểm tra xem tệp nào đã thay đổi. Điều này hữu ít khi ta chỉ commit một vài tập tin chứ không phải toàn bộ. Lệnh git status hiển thị các tập ti đã sửa đổi trong thư mục, được sắp đặt cho lần commit sắp tới, thông tin mang đến bao gồm:

- Đang ở nhánh (brach) master
- Commit này là Initial commit (lần đầu)
- Không có gì thay đổi tronnf commit

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

b) Staging và Committing một tập tin Repository

• Bước 1: Tạo tập tin

B1. Tại git-intro repository, sử dụng echo để tạo tập tin README.MD với thông tin trog giấu ngoặc kép

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ echo "I am on my way to passing the exam" > READNE.MD
```

B2. Dùng lệnh ls -la để kiểm tra tập tin vừa tạo, cũng nghư thư mục .git. Sau đó sử dụng lệnh cat để hiển thị nội dung README.MD

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ ls -la
total 16
drwxr-xr-x 3 kali kali 4096 Sep 16 11:38 .
drwxr-xr-x 3 kali kali 4096 Sep 16 11:22 ..
drwxr-xr-x 7 kali kali 4096 Sep 16 11:34 .git
-rw-r--r-- 1 kali kali   35 Sep 16 11:38 README.MD

┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat README.MD
I am on my way to passing the exam
```

• Bước 2: Kiểm tra trạng thái Repository

Kiểm tra trạng thái Repository bằng lệnh git status. Lúc này Git đã tìm thấy tệp mới trong thư mục.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
      README.MD

nothing added to commit but untracked files present (use "git add" to
track)
```

• Bước 3: Staging tập tin

B1. Tiếp theo sử dụng lệnh git add để "stage" thêm tệp README.MD. Giai đoạn này là trung gian trước khi commit. Lệnh này tạo một snapshot cho tệp. Mọi thay đổi của tệp này sẽ được thêm vào trường hợp git add khác trước khi commit.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git add README.MD
```

B2. Sử dụng lại lệnh git status, nhận thấy các thay đổi theo "stage" được hiển thị dưới dạng " new file: README.MD".

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
      new file:   README.MD
```

• Bước 4: Commit tập tin

Bây giờ khi đã "staged" sắp đặt các thay đổi, cần commit để Git theo dõi những thay đổi đó, commit nội dung theo "staged" dưới dạng commit snapshot bằng lệnh git commit. Tuỳ chọn -m cho phép thêm thông điệp giải thích những thay đổi mà ta làm ra. Lưu ý số và chữ trong hightlight là commit ID. Mọi commit được xác đihj bằng một hàm băm SHA1 duy nhất. Commit ID là 7 ký đầu tiên trong commit hash.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git commit -m "Committing README.MD to begin tracking changes"
[master (root-commit) c96b33f] Committing README.MD to begin tracking
changes
 1 file changed, 1 insertion(+)
 create mode 100644 README.MD
```

• Bước 5: Xem lịch sử commit

Sử dụng lệnh git log để hiển thị tất cả các commit trong lịch sử của nhánh hiện tại Theo mặc định, tất cả các commit là được thực hiện cho nhánh master.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git log
commit c96b33fc464081b8723e621a7a91af0ad1f4992c (HEAD -> master)
Author: Khoa Nghi <hoangkhoa95@live.com>
Date:   Thu Sep 16 12:01:52 2021 -0400

    Committing README.MD to begin tracking changes
```

c) Sửa đổi tập tin và theo dõi các thay đổi

• Bước 1: Sửa đổi tập tin

B1. Thực hiện thay đổi đối với README.MD bằng lệnh echo. Sử dụng ">>" sửa tập tin hiện có. Dấu ">" sẽ ghi đè lên tệp hiện có.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ echo "I am beginning to understand Git" >> README.MD
```

B2. Sử dụng lệnh cat để xem tệp đã sửa đổi.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat README.MD
I am on my way to passing the exam
I am beginning to understand Git
```

• Bước 2: Xác minh thay đổi đối với repository

Xác minh sự thay đổi trong repository bằng lệnh git status.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
      modified:   README.MD

no changes added to commit (use "git add" and/or "git commit -a")
```

• Bước 3: Stage tập tin thay đổi

Tập ti đã sửa đổi sẽ cần được stage lại trước khi nó có thể được commit bằng cách sử dụng lệnh git add một lần nữa.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git add README.MD
```

• Bước 4: Commit tập tin đã stage

Commit tập tin đã stage bằng cách sử dụng git commit, sẽ có ID commit mới.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git commit -m "Added additional line to file"
[master 892df0d] Added additional line to file
 1 file changed, 1 insertion(+)
```

• Bước 5: Xác minh các thay đổi trong repository

B1. Sử dụng lệnh git log một lần nữa để hiển thị tất cả các commit.

® **Bài tập (yêu cầu làm)**

1. *Cho biết thông tin về lần **commit** vừa thực hiện:*
   - *Commit ID*
   - *Ngày giờ commnit*
   - *Thông điệp commit*

B2. Khi bạn có quá nhiều commit trong log. Bạn có thể so sánh hai commit bằng lệnh git diff. Dấu "+" thể hiện sự thay được thêm vào.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git diff c96b33f 892df0d
diff --git a/README.MD b/README.MD
index bc8fbcc..8bae625 100644
--- a/README.MD
+++ b/README.MD
@@ -1 +1,2 @@
 I am on my way to passing the exam
+I am beginning to understand Git
```

d) Branches và Merging (nhánh và hợp nhất)

Khi một repository được tạo ra, tất cả các tập tin sẽ được đưa vào một nhánh master/ Phân nhánh sẽ giúp bạn kiểm soát và thay đổi trong một khu vực mà không hưởng nhánh chính, điều này giúp code không bị ghi đè không mong muốn.

• Bước 1: Tạo branch mới

Tạo branch mới feature với câu lệnh git branch <branch-name>

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git branch feature
```

• Bước 2: Kiểm tra branch hiện tại

Sử dụng lệnh git branch mà không có tên branch để hiển thị tất cả các branch cho repository này. Dấu "*"

bên cạnh brnnch master chỉ ra rằng đây là branch hiện tại - branch hiện được "checked out"

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch
  feature
* master
```

• Bước 3: Checkout branch mới

Sử dụng lệnh git checkout <branch-name> để chuyển qua branch feature

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git checkout feature
Switched to branch 'feature'
```

• Bước 4: Kiểm tra brach hiện tại

B1. Kiểm tra rằng đã chuyển sang nhánh mới

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch
* feature
  master
```

B2. Thêm một đoạn text mới vào tập tin README.MD

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  echo "This text was added originally while in the feature branch"
>> README.MD
```

B3. Kiểm tra đoạn text vừa được thêm vào

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat README.MD
I am on my way to passing the exam
I am beginning to understand Git
This text was added originally while in the feature branch
```

• Bước 5: Stage tập tin đã sửa đổi trong branch feature

B1. Stage tập tin cập nhật vào brach hiện tại

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git add README.MD
```

B2. Sử dụng lệnh git status và xem thông báo sửa đổi tập tin README.MD được stage trong branch feature

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
      modified:   README.MD
```

• Bước 6: Commit tập tin stage tronng brach feature

B1. Sử dụng git commit

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git commit -m "Added a third line in feature branch"
[feature 6d5b9f2] Added a third line in feature branch
 1 file changed, 1 insertion(+)
```

B2. Sử dụng lệnh git log để hiển thị tất cả các commit bao gồm cả ccommit vừa thực hiện với nhánh tính feature.

```
──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git log
commit 6d5b9f2a6ebca327f7071c5aefd8499e1be4dae3 (HEAD -> feature)
Author: Khoa Nghi <hoangkhoa95@live.com>
Date:   Fri Sep 17 00:28:55 2021 -0400

    Added a third line in feature branch

commit 892df0d26eb839212ddfad16fb885bf0a1391757 (master)
Author: Khoa Nghi <hoangkhoa95@live.com>
Date:   Thu Sep 16 12:20:35 2021 -0400

    Added additional line to file

commit c96b33fc464081b8723e621a7a91af0ad1f4992c
Author: Khoa Nghi <hoangkhoa95@live.com>
Date:   Thu Sep 16 12:01:52 2021 -0400

    Committing README.MD to begin tracking changes
```

• Bước 7: Checkout branch master

Chuyển sang branch master bằng lệnh git checkout master và xác minh branch làm việc hiện tại bằng cách sử dụng lệnh git branch.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git checkout master
M	README.MD
Switched to branch 'master'
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch
  feature
* master
```

• Bước 8: Merge tất cả các nội dung tập tin từ branch feature sang master

B1. Các branch thường được sử dụng khi triển khai các tính năng mới hoặc sửa lỗi. Họ có thể gửi các xem xét cho các thành viên trong nhóm và sau khi được xác minh có thể pull về brach master.

Merge nội dung từ branch feature về master có thể sử dụng lệnh git merge <branch-name>. branch-name là brach muốn pull về branch hiện tại.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git merge feature
Updating 892df0d..6d5b9f2
Fast-forward
 README.MD | 1 +
 1 file changed, 1 insertion(+)
```

B2. Kiểm tra nội dung sao khi merge

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat README.MD                                                    130
×
I am on my way to passing the exam
I am beginning to understand Git
This text was added originally while in the feature branch
```

• Bước 9: Xoá branch

B1. Kiểm tra brach feature vẫn còn tồn tại bằng lệnh git branch

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch
  feature
* master
```

B2. Xoá branch feature bằng lệnh git branch -d <branch-name>

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch -d feature
Deleted branch feature (was 6d5b9f2).
```

B3. Xác nhận lại branch feature còn tồn tại không bằng git branch

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch
* master
```

e) Xử lý xung đột khi merge

Đôi khi bạn sẽ gặp conflict (xung đột) khi merge. Khi ta thực hiện các thay đổi chồng chéo đối với tập tin và Git sẽ không tự động merge những thứ như thế.

• Bước 1: Tạo branch test mới

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch test
```

• Bước 2: Checkout branch test

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git checkout test
Switched to branch 'test'

┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git branch
  master
* test
```

• Bước 3: Kiểm tra nội dung tập tin README.MD hiện tại

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat README.MD
I am on my way to passing the exam
I am beginning to understand Git
This text was added originally while in the feature branch
```

• Bước 4: Sửa đổi nội dung tập tin

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ sed -i 's/feature/test/' README.MD
```

• Bước 5: Kiểm tra lại nội dung chỉnh sửa tập tin README.MD

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat README.MD
I am on my way to passing the exam
I am beginning to understand Git
This text was added originally while in the test branch
```

• Bước 6: Stage, commit branch test và checkout. branch master

Tùy chọn git commit -a chỉ ảnh hưởng đến các tập tin đã được sửa đổi và xóa. Nó không ảnh hưởng đến các tập tin mới.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git commit -a -m "Change feature to test"
[test 1068e44] Change feature to test
 1 file changed, 1 insertion(+), 1 deletion(-)

┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git checkout master
Switched to branch 'master'

┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git branch
* master
  test
```

• Bước 7: Tiếp tục sửa đổi và kiểm tra nội dung tập tin README.MD ở branch master

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ sed -i 's/feature/master/' README.MD

┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat README.MD
I am on my way to passing the exam
I am beginning to understand Git
This text was added originally while in the master branch
```

• Bước 8: Stage và commit branch master

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git commit -a -m "Changed feature to master"
[master 649442e] Changed feature to master
 1 file changed, 1 insertion(+), 1 deletion(-)
```

• Bước 9: Merge hai branch test và master

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git merge test
Auto-merging README.MD
CONFLICT (content): Merge conflict in README.MD
Automatic merge failed; fix conflicts and then commit the result
```

• Bước 10: Tìm kiếm xung đột

B1. Sử dụng lệnh git log để xem các commit. Lưu ý rằng phiên bản HEAD là branch master. Cái này sẽ hữu ích trong bước tiếp theo.

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git log                                                          1
×
commit 649442e58672154ec45cd60d40def94ac48c9700 (HEAD -> master)
Author: Khoa Nghi <hoangkhoa95@live.com>
Date:   Fri Sep 17 09:46:26 2021 -0400

    Changed feature to master
```

```
commit 6d5b9f2a6ebca327f7071c5aefd8499e1be4dae3
…
```

B2. Tập README.MD sẽ chứa thông tin tìm ra xung đột. Phiên HEAD (branch master) cho chứa từ "master" đang xung đột với phiên bản branch test là từ "test".

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat READNE.MD
I am on my way to passing the exam
I am beginning to understand Git
<<<<<<< HEAD
This text was added originally while in the master branch
=======
This text was added originally while in the test branch
>>>>>>> test
```

• Bước 11: Chỉnh sửa tập tin README.MD để xoá đoạn xung đột

B1. Dùng vim/nano để sửa tập ti

B2. Xoá đoạn highlight như sau, lưu lại và kiểm tra kết quả

```
I am on my way to passing the exam
I am beginning to understand Git
<<<<<<< HEAD
This text was added originally while in the master branch
=======
This text was added originally while in the test branch
>>>>>>> test
```

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ cat READNE.MD
I am on my way to passing the exam
I am beginning to understand Git
This text was added originally while in the master branch
```

• Bước 12: Stage, commit và kiểm tra commit branch master

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git add README.MD
```

```
┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$ git commit -a -m "Manually merged from test branch"
[master 22fe012] Manually merged from test branch

┌──(kali㉿phaphajian)-[~/labs/git-intro]
└─$  git log
commit 22fe012001cc1c9952cfa7f6955e5a0cae9dccec (HEAD -> master)
Merge: 649442e 1068e44
Author: Khoa Nghi <hoangkhoa95@live.com>
Date:   Fri Sep 17 10:04:32 2021 -0400

    Manually merged from test branch

commit 649442e58672154ec45cd60d40def94ac48c9700
….
```

f) Tích hợp Git với Github

Hiện tại, những thay đổi chỉ được lưu trữ tại máy. Git chạy cục bộ và không yêu cầu bất kỳ máy chủ hoặc dịch vụ lưu trữ đám mây nào. Git cho phép người dùng lưu trữ cục b và quản lý tập tin.

Việc sử dụng Github sẽ giúp làm việc nhóm tốt hơn và tránh tình trạng mất dữ liệu. Có một số dịch vụ Git khá phổ biến bao gồm GitHub, Stash từ Atlassian và GitLab.

- Bước 1: Tạo tài khoản Github
- Bước 2: Đăng nhập tài khoản Github
- Bước 3: Tạo Repository
- Bước 4: Tạo devops-study-team
- Bước 5: Sao chép tập tin README.MD vào thư mục vừa tạo
- Bước 6: Khởi tạo Git repository
- Bước 7: Trỏ Git repository đến GitHub repository
- Bước 8: Stage và commit tập tin README.MD
- Bước 9: Kiểm tra commit
- Bước 10: Gửi (push) tập tin từ Git lên Github
- Bước 11: Kiểm tra tập tin trênn. Github

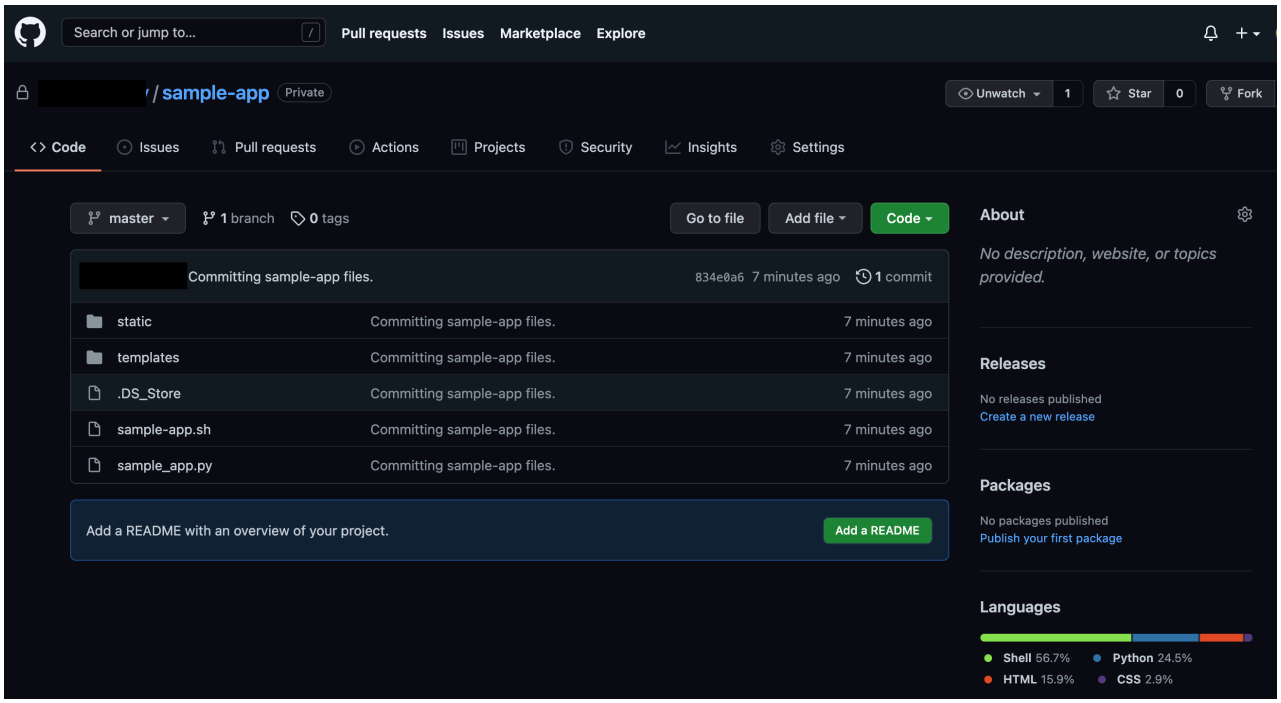
® Bài tập (yêu cầu làm)

2. *Sinh viên hoàn thiện các bước 3 đến 11 trong tích hợp **Git với Github**, trình bày step-by-step có minh chứng*

2. Xây CI/CD Pipeline bằng Jenkins

a) Commit Sample App lên Github

Sinh viên giải nén sample-app.zip, sau đó tạo GitHub repository để commit sample-app.



b) Sửa đổi Sample App và push thay đổi lên Git

Vì lý do đụng port Jenkins Docker nên ta sẽ thay đổi port trong mã nguồn của Sample App.

- Bước 1: Mở cả 2 tập tin **sample_app.py** và **sample-app.sh** chỉnh sửa port 8080 thành 5050 như sau:

```
┌──(kali㉿phaphajian)-[~/sample-app]
└─$ cat  sample_app.py
# Add to this file for the sample app lab
from flask import Flask
from flask import request
from flask import render_template

sample = Flask(__name__)
```

```
@sample.route("/")
def main():
    return render_template("index.html")

if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=5050)
```

```
┌──(kali㉿phaphajian)-[~/sample-app]
└─$ cat sample-app.sh
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/Dockerfile
echo "COPY  ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY  ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY  sample_app.py /home/myapp/" >> tempdir/Dockerfile
echo "EXPOSE 5050" >> tempdir/Dockerfile
echo "CMD python /home/myapp/sample_app.py" >> tempdir/Dockerfile

cd tempdir
docker build -t sampleapp .
docker run -t -d -p 5050:5050 --name samplerunning sampleapp
docker ps -a
```

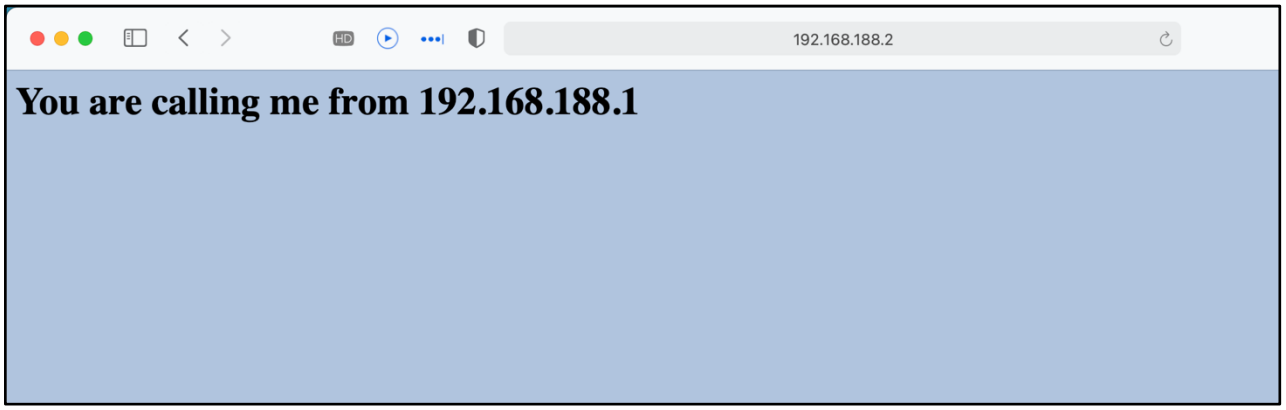
• Bước 2: Build và kiểm tra sample-app

B1. Thực thi bash để chạy ứng dụng với port mới 5050

```
┌──(kali㉿phaphajian)-[~/sample-app]
└─$ bash ./sample-app.sh
Sending build context to Docker daemon  6.144kB
Step 1/7 : FROM python
 ---> a5210955ee89
Step 2/7 : RUN pip install flask
 ---> Running in 2cb050e7cfad
---> 541beecb7f81
Step 3/7 : COPY  ./static /home/myapp/static/
 ---> b6d0cc6dd575
Step 4/7 : COPY  ./templates /home/myapp/templates/
 ---> bd34ad380e9d
Step 5/7 : COPY  sample_app.py /home/myapp/
 ---> f87bb7ea2f1d
Step 6/7 : EXPOSE 5050
 ---> Running in 7eabd272c88a
Removing intermediate container 7eabd272c88a
 ---> 554362d0fcc6
Step 7/7 : CMD python /home/myapp/sample_app.py
 ---> Running in f81da42beda7
Removing intermediate container f81da42beda7
 ---> f8c3e91bebfb
Successfully built f8c3e91bebfb
Successfully tagged sampleapp:latest
318d903b0c668d4a67259661cc816920f700e79a7ed7026c29ae42253405c42d
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS
PORTS                     NAMES
318d903b0c66   sampleapp   "/bin/sh -c 'python …"   2 seconds ago   Up
Less than a second   0.0.0.0:5050->5050/tcp   samplerunning
```

B2. Mở trình duyệt hoặc terminal kiểm tra đường dẫn localhost:5050

```
┌──(kali㉿phaphajian)-[~/sample-app]
└─$ curl localhost:5050
<html>
<head>
    <title>Sample app</title>
    <link rel="stylesheet" href="/static/style.css" />
</head>
<body>
    <h1>You are calling me from 172.17.0.1</h1>
</body>
</html>
```



• Bước 3: Push sự thay đổi lên GitHub

> ® **Bài tập (yêu cầu làm)**
>
> ***3.*** *Sinh viên thực hiện commit và push code với thông điệp* ***"Changed port from 8080 to 5050",*** *trình bày step-by-step có minh chứng*

c) Tải và thiết lập chạy Jenkins Docker Image

• Bước 1: Tải Jenkins Docker image

Jenkins Docker image được lưu trữ tại đây https://hub.docker.com/r/jenkins/jenkins
Tại thời điểm soạn bài lab này, thì hướng dẫn tại trang web trên để tải xuống Jenkins là docker pull jenkins/jenkins.

```
┌──(kali㉿phaphajian)-[~/sample-app]
└─$ docker pull jenkins/jenkins:lts
130 ×
lts: Pulling from jenkins/jenkins
4c25b3090c26: Pull complete
```

```
750d566fdd60: Pull complete
2718cc36ca02: Pull complete
5678b027ee14: Pull complete
c839cd2df78d: Pull complete
50861a5addda: Pull complete
ff2b028e5cf5: Pull complete
ee710b58f452: Pull complete
2625c929bb0e: Pull complete
6a6bf9181c04: Pull complete
bee5e6792ac4: Pull complete
6cc5edd2133e: Pull complete
c07b16426ded: Pull complete
e9ac42647ae3: Pull complete
fa925738a490: Pull complete
4a08c3886279: Pull complete
2d43fec22b7e: Pull complete
Digest:
sha256:a942c30fc3bcf269a1c32ba27eb4a470148eff9aba086911320031a3c3943e6
c
Status: Downloaded newer image for jenkins/jenkins:lts
docker.io/jenkins/jenkins:lts
```

• Bước 2: Khởi chạy Jenkins Docker container

Lệnh sau sẽ khởi chạy Jenkins Docker container và sau đó cho phép các lệnh Docker được thực thi bên trong máy chủ Jenkins.

```
┌──(kali㉿phaphajian)-[~/sample-app]
└─$ docker run --rm -u root -p 8080:8080 -v jenkins-
data:/var/jenkins_home -v $(which docker):/usr/bin/docker -v
/var/run/docker.sock:/var/run/docker.sock -v "$HOME":/home --name
jenkins_server jenkins/jenkins:lts
```

Các tùy chọn được sử dụng trong lệnh chạy docker này như sau:

- **--rm:** Tùy chọn này tự động xóa Docker container khi bạn ngừng chạy nó.
- **-u:** Tùy chọn này chỉ định người dùng. Ta muốn Docker cotainner này chạy dưới dạng root để tất cả các lệnh Docker được nhập bên trong máy chủ Jenkins được cho phép thực thi
- **-p:** Tùy chọn này chỉ định port mà máy chủ Jenkins sẽ chạy cục bộ (local).

® **Bài tập (yêu cầu làm)**

4. *Sinh viên tự tìm hiểu và giải thích tuỳ chọn* ***-v*** *ở lệnh chạy Docker trên.*

• Bước 3: Kiểm tra máy chủ Jenkins đang chạy

Sao chép mật khẩu admin ở output.

```
************************************************************
************************************************************
************************************************************

Jenkins initial setup is required. An admin user has been created and a
password generated.
Please use the following password to proceed to installation:

fa19d3e3d96f48eea85a84ea88b517ba

This may also be found at:
/var/jenkins_home/secrets/initialAdminPassword

************************************************************
************************************************************
2021-09-18 04:51:10.892+0000 [id=23]  INFO  hudson.WebAppMain$3#run:
Jenkins is fully up and running
```

• Bước 4: Hình dung hệ thống đang chạy

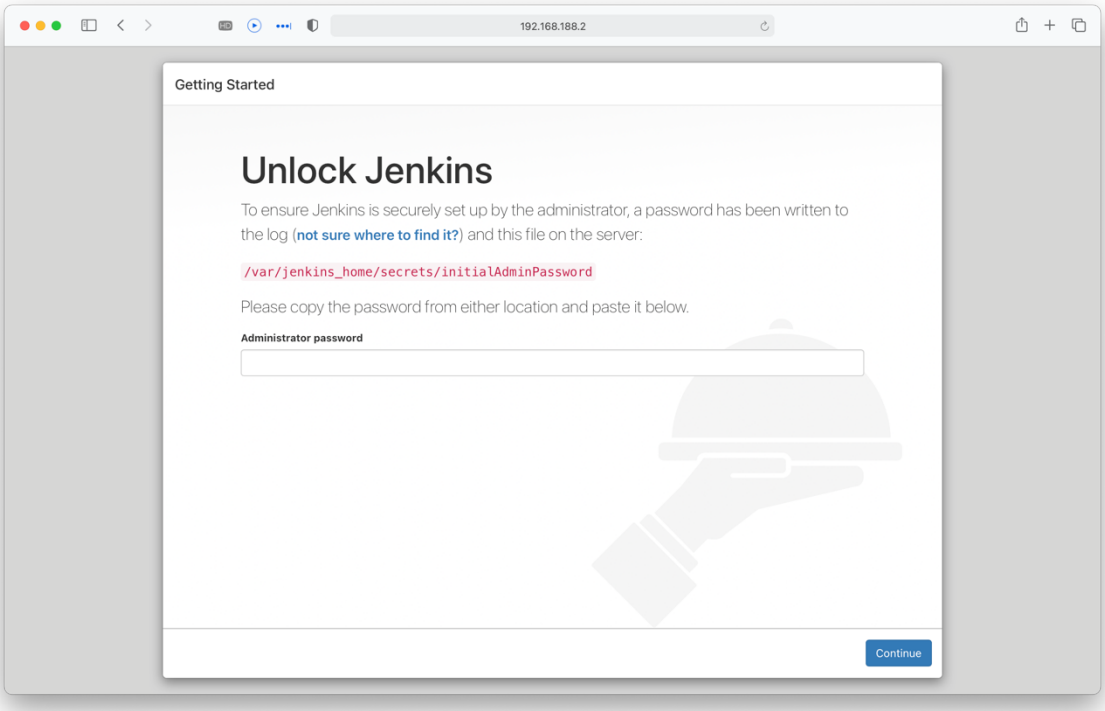
Biểu đồ sau cho bạn có cái nhìn các mực độ hệ thống Docker-inside-Docker.

```
+--------------------------------------+
|Your Computer's Operating System      |
| +----------------------------------+ |
| |Linux VM                          | |
| | +--------------------------+     | |
| | |Docker container          |     | |
| | | +----------------------+ |     | |
| | | | Jenkins server       | |     | |
| | | | +----------------+   | |     | |
| | | | |Docker container|   | |     | |
| | | | +----------------+   | |     | |
| | | +----------------------+ |     | |
| | +--------------------------+     | |
| +----------------------------------+ |
+--------------------------------------+
```

d) Cấu hình Jenkins

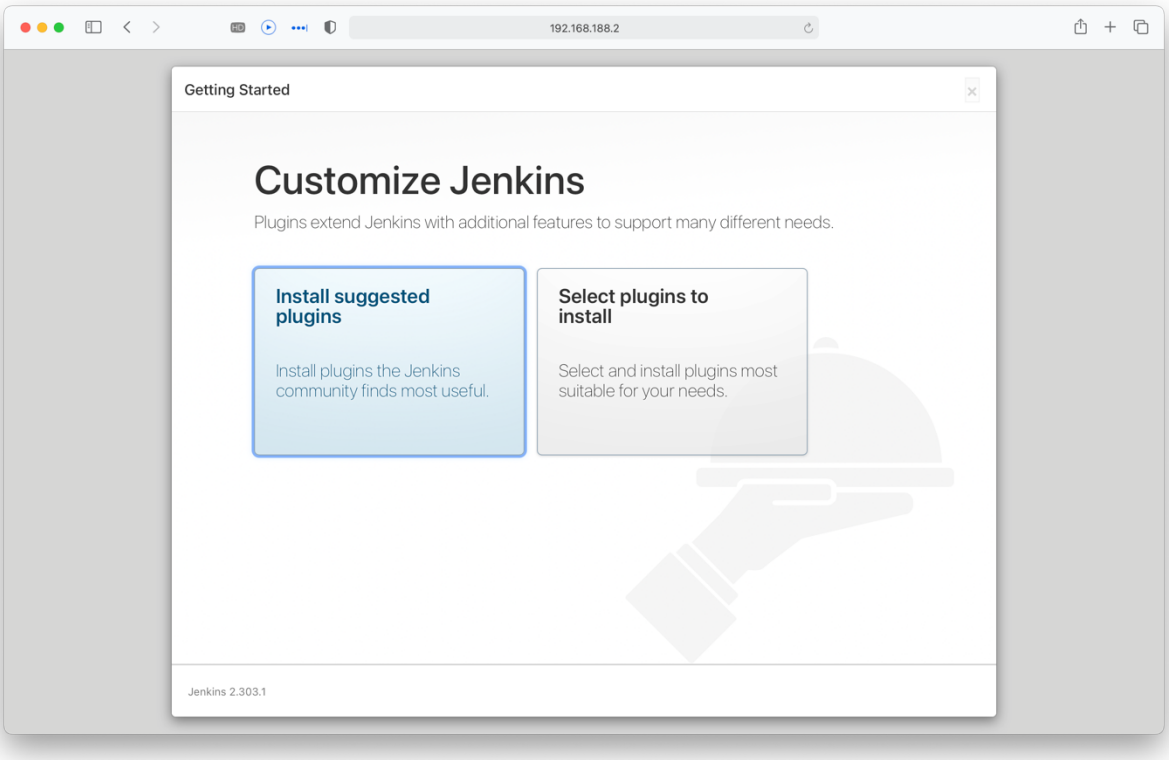
• Bước 1: Mở trình duyệt

Truy cập http://localhost:8080/ hoặc http://<ip linux vm>:8080/ và nhập mật khẩu admin.

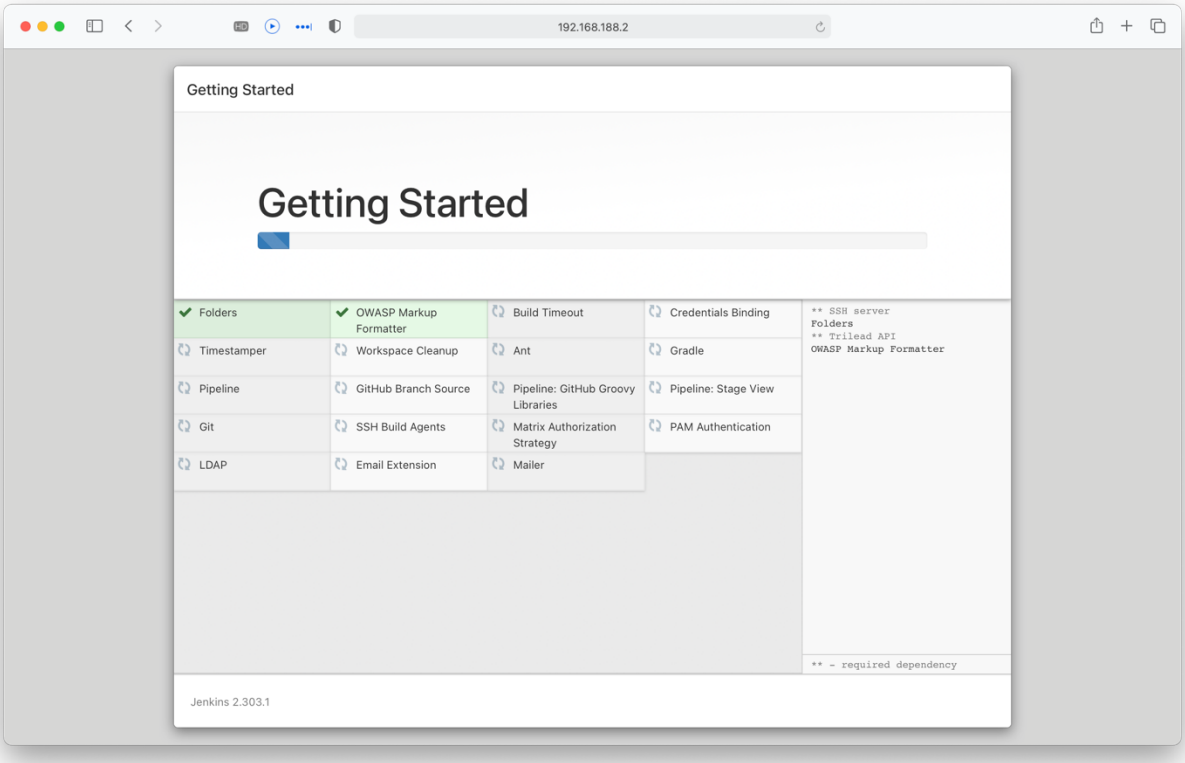


• Bước 2: Cài đặt các Jenkins plugins khuyến nghị

Chọn vào Cài đặt các plugin được đề xuất và đợi Jenkins tải xuống và cài đặt các plugin.



Trong cửa sổ termial, ta sẽ thấy các thông báo log khi quá trình cài đặt được tiến hành.



• Bước 3: Bỏ qua tạo tài khoản admin

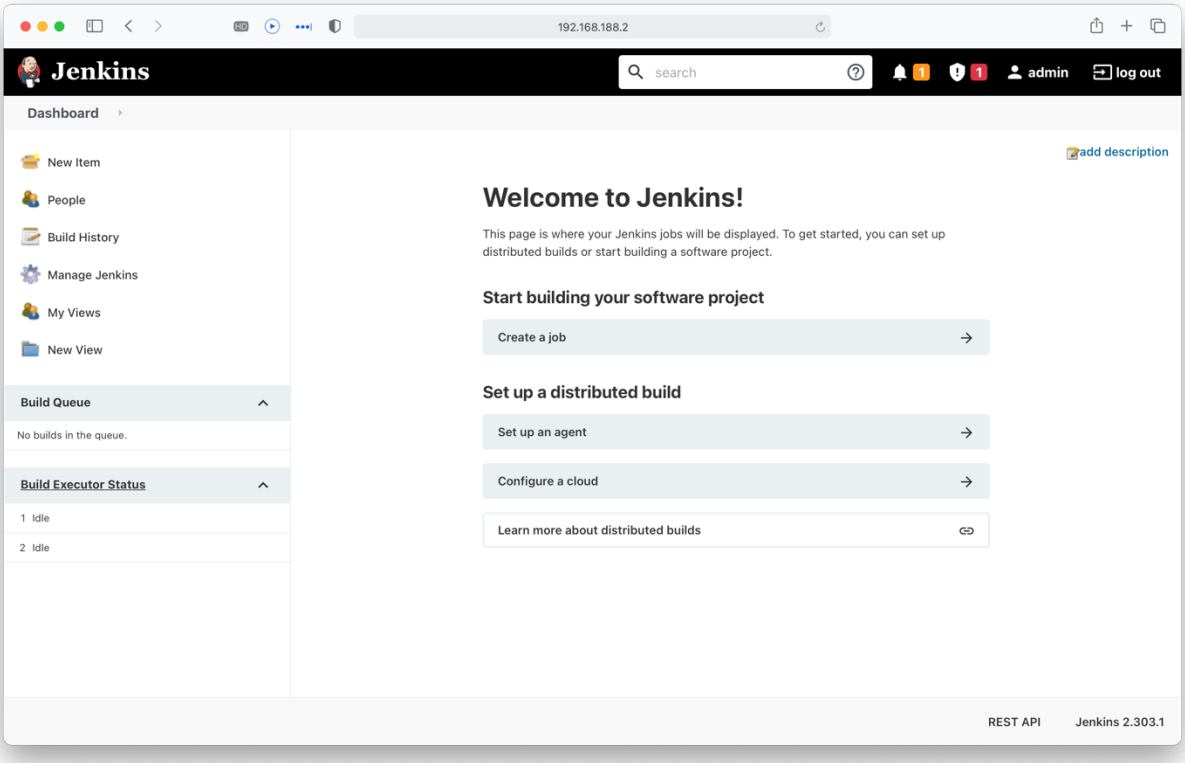
Chọn *Skip and continue as admin*

• Bước 4: *Bỏ. qua* instance configuration

Trong cửa sổ *Instance Configuration*, không thay đổi bất kỳ thứ gì. Chọn *Save and Finish*.

• Bước 5: Bắt đầu sử dụng Jenkins

Ở cửa số kế tiếp, chọn *Start using Jenkins.*



e) Sử dụng Jenkins để chạy ứng dụng đã dựng

Đơn vị cơ bản của Jenkins là job (hay project). Ta có thể tạo nhiều job với các tác vụ sau:

- Lấy mã nguồn từ repository Github
- Dựng lại ứng dụng và script hoặc build tool.
- Đóng gói ứng dụng và chạy nó trên một máy chủ.

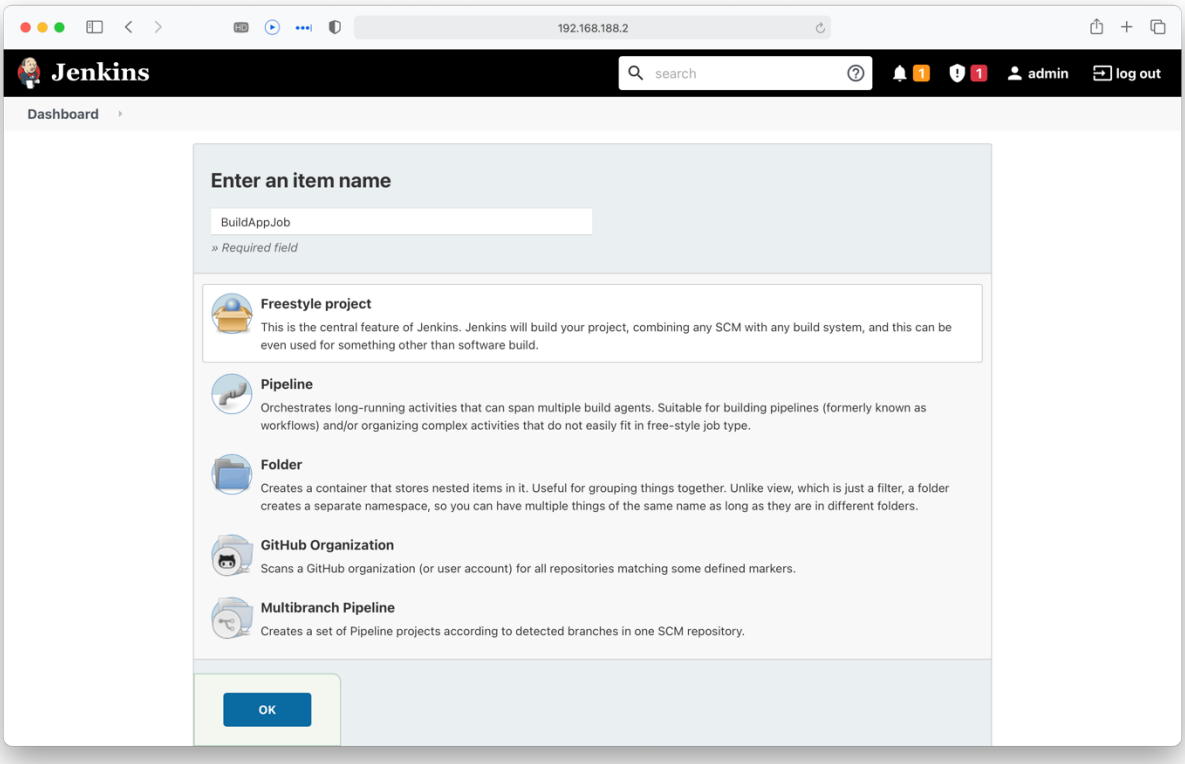
• Bước 1: Tạo job mới

B1. Chọn *Create a job*

B2. Trong trường *Enter an item name,* điền *BuildAppJob*

B3. Chọn thể loại *Freestyle project*

B4. Chọn *OK*



• Bước 2: Cấu hình Jenkins BuildAppJob

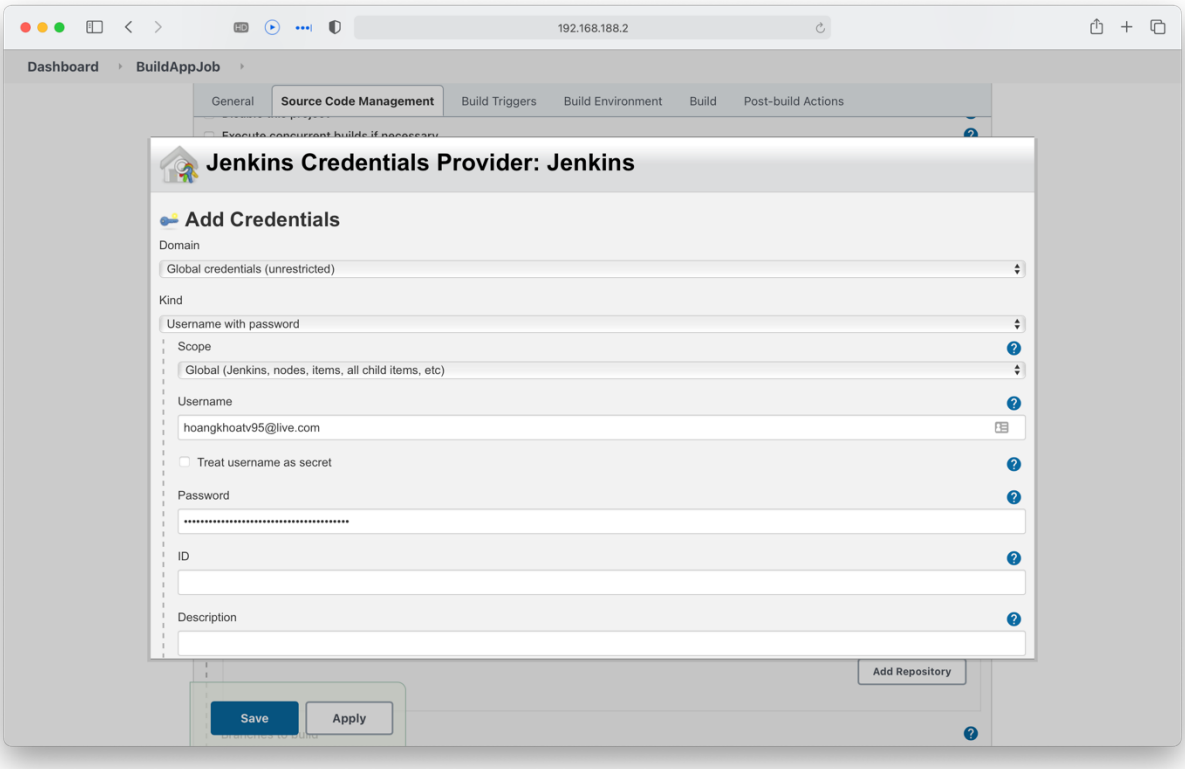
Bây giờ ta đang ở cửa sổ cấu hình, nơi ta có thể nhập chi tiết cho job của mình. Các tab trên cùng là phím tắt cho các tuỳ chọn bên dưới.

B1. Chọn *General tab*, thêm description cho job. Ví dụ: "My first Jenkins job."

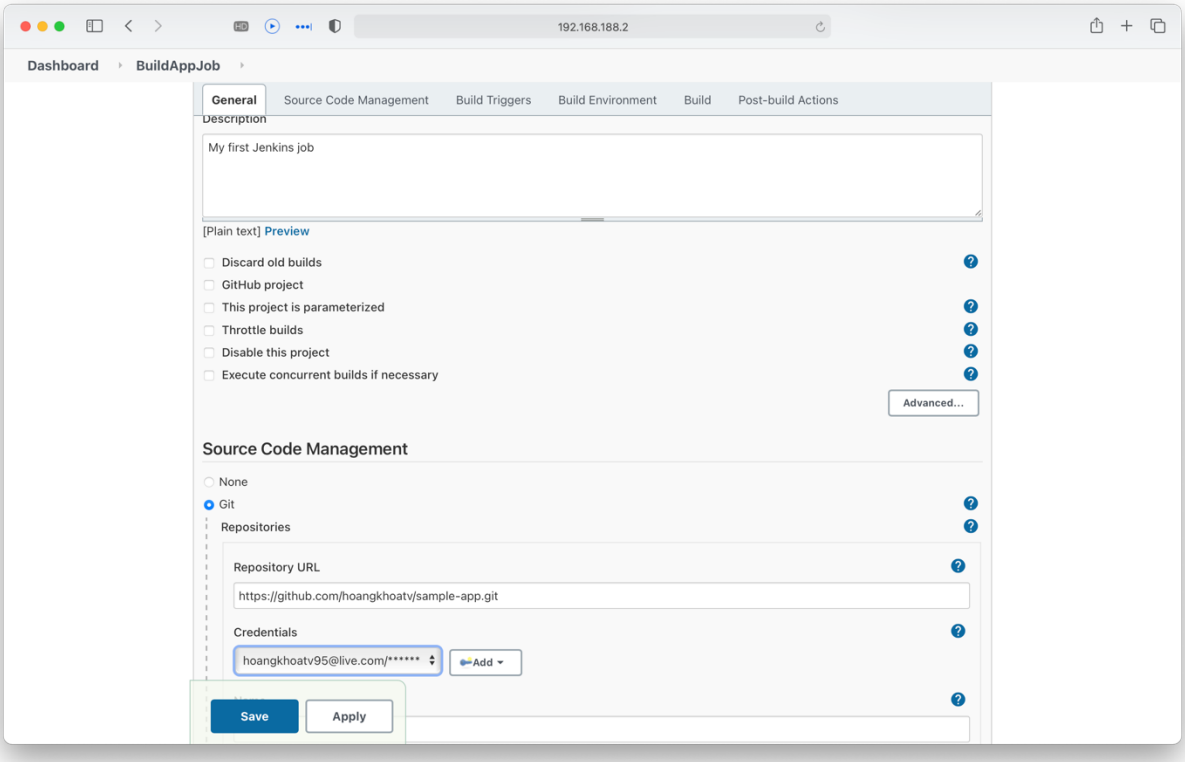
B2. Chọn *Source Code Management tab* và chọn nút radio *Git.* Thêm liên kết repository Github của ứng dụng. Ví dụ: *https://github.com/username/sample-app.git*

B3. Tại *Credentials,* chọn nút *Add* và chọn Jenskins

B4. Ở *Add Credentials* dialog box, điền GitHub username và password (token) và chọn *Add*.

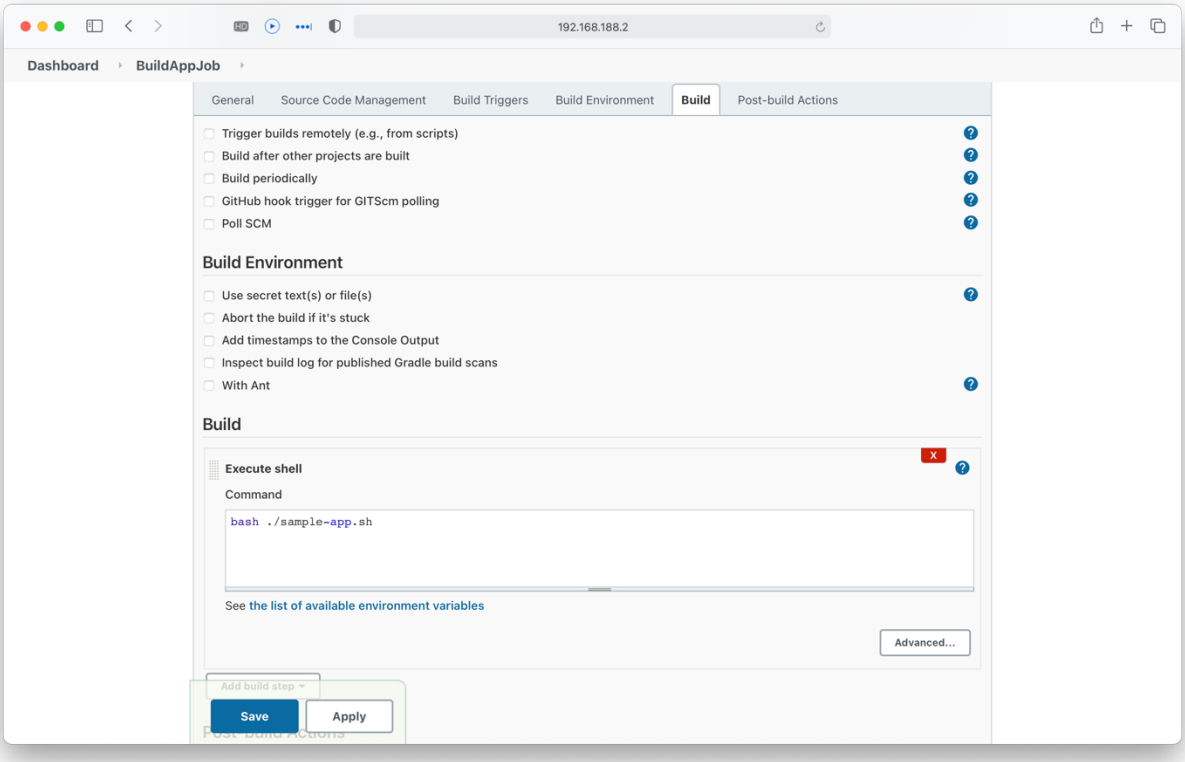


B5. Trong dropdown *Credentials,* hiện tại đang *None,* giờ chọn qua cấu hình vừa tạo



B6. Sau khi ta đã thêm đúng URL và thông tin đăng nhập, Jenkins kiểm tra quyền truy cập vào repository. Nếu có thông báo lỗi thì kiểm tra lại bước trên.

B7. Chọn tab *Build*
B8. Ở dropdown *Add build step,* chọn *Execute shell*
B9. Trong trường *Command,* nhập lệnh ta sử dụng để dựng ứng dụng *sample-app.sh*



B10. Lưu ý nhớ xoá docker container trước đó. Chọn *Save*. Ta quay trở lại Jenkins dashboard với BuildAppJob đã được chọn.

• Bước 3: Bắt Jenkins dựng ứng dụng

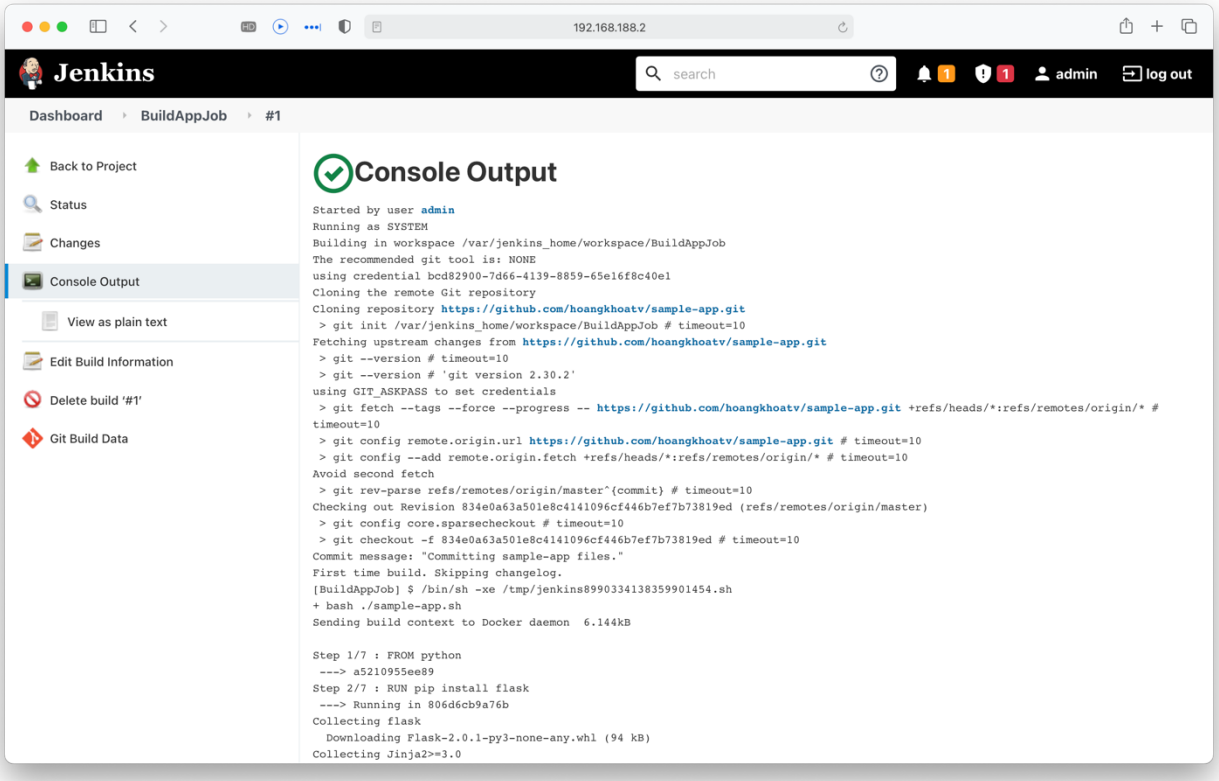
Ở bên trái, chọn *Build Now* để bắt đầu job. Jenkins sẽ tải xuống repository Git của ta và thực thi lệnh dựng *bash ./sample-app.sh*.

• Bước 4: Truy cập chi tiết bản dựng

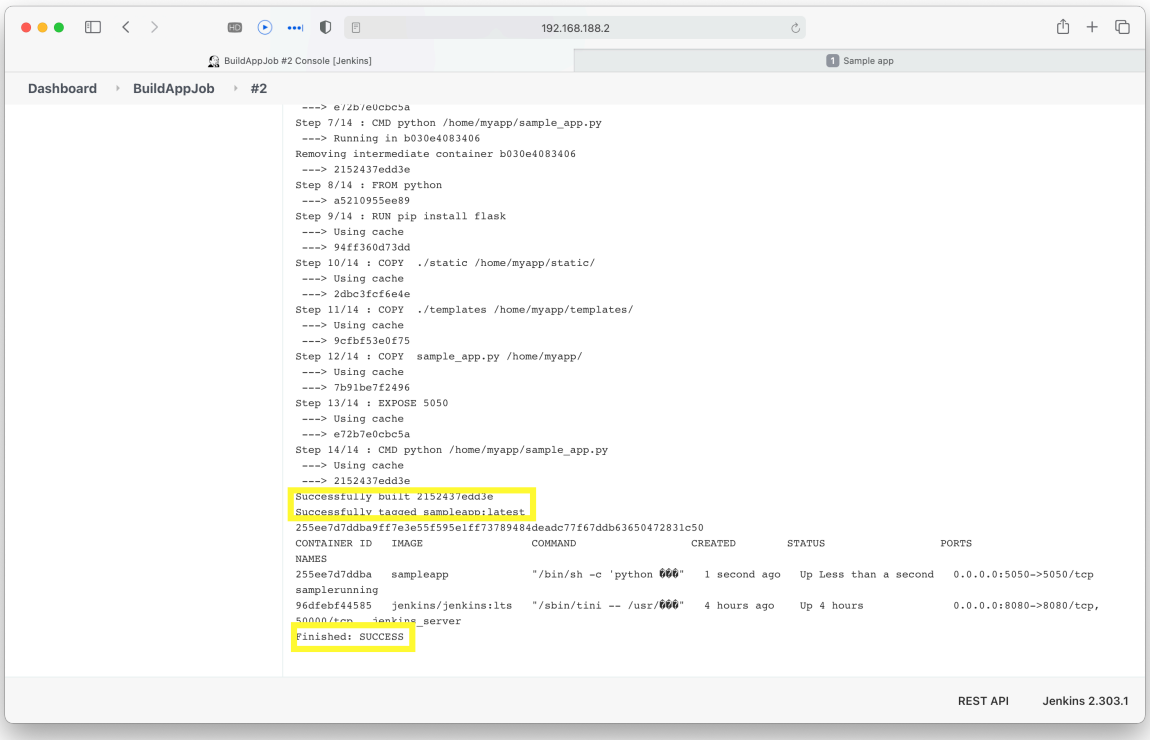
Trong phần *Build History,* chọn *build number.*

• Bước 5: Xem console output

Chọn *Console Output*.



Thông báo thành công và kết quả lệnh *docker ps -a*. Ở container, ứng dụng vừa dựng chạy ở port 5050 và Jenkins server là 8080.



- Bước 6: Mở trình duyệt và truy cập xác nhận ứng dụng đã chạy.

f) Dùng Jenkins để kiểm tra bản dựng

> ® Bài tập (yêu cầu làm)
>
> **5.** *Sinh viên hoàn thành job kiểm tra bản dựng ứng dụng tự động theo gợi ý bên dưới, trình bày step-by-step có minh chứng.*

Trong phần này ta sẽ tạo job thứ 2, kiểm tra bản dựng hoạt động bình thường.
Lưu ý bạn đã dừng và xoá docker container *samplerunning.*

• Bước 1: Tạo job mới để kiểm tra sample-app

Đặt tên: *TestAppJob_TênNhóm*

• Bước 2: Cấu hình Jenkins TestAppJob

B1. Đặt description: "My first TênNhóm's Jenkins test."

B2. *Source Code Management* chọn *None*

B3. Ở *Build Triggers tab,* chọn checkbox *Build after other projects are built* và điền *BuildAppJob*

• Bước 3: Viết tập lệnh thử nghiệm sẽ chạy sau khi dựng BuildAppJob

B1. Chọn *Build tab*

B2. Chọn *Add build step* và chọn *Execute shell*. Gợi ý viết bash shell nhận kết quả trả về từ việc truy cập đường dẫn ứng dụng và kiểm tra trong đó có tồn thông tin nhận biết thành công hay không. Nếu thành công thì exit code là 0 và ngược lại là 1.

```
//Có làm thì mới có ăn :3
```

B3. Chọn *Save*

• Bước 4: Yêu cầu Jenkins chạy lại job BuildAppJob

• Bước 5: Kiểm tra cả 2 job hoàn thành hay không

Nếu thành công ta sẽ thấy cột Last Success ở cả BuildAppJob và TestAppJob.

g) Tạo Pipeline trong Jenkins

Mặc dù có thể chạy hai job của mình bằng cách chỉ cần nhấp vào nút Build Now cho BuildAppJob, nhưng các dự án phát triển phần mềm thường phức tạp hơn nhiều. Những dự án này có thể được hưởng lợi rất nhiều từ tự động hóa các bản dựng để tích hợp liên tục các thay đổi đoạn mã và liên tục tạo các bản dựng phát triển đã sẵn sàng để triển khai. Đây là bản chất của CI/CD. Một pipeline có thể được tự động hóa điều này.

> ® Bài tập (yêu cầu làm)
>
> **6.** *Sinh viên hoàn thành pipeline của 2 ứng dụng Buid và Test theo gợi ý bên dưới, trình bày step-by-step có minh chứng.*

• Bước 1: Tạo job Pipeline

B1. Chọn *New Item*

B2. Ở trường *Enter an item name,* điền *SamplePipeline**TênNhóm***

B3. Chọn kiểu job Pipeline

B4. Chọn *OK*

• Bước 2: Cấu hình job SamplePipeline

Ở tab *Build trigger*, trong phần *Pipeline section* điền đoạn mã sau:

```
node {
    stage('Preparation') {
        catchError(buildResult: 'SUCCESS') {
            sh 'docker stop samplerunning'
            sh 'docker rm samplerunning'
        }
    }
    stage('Build') {
        build 'BuildAppJob'
    }
    stage('Results') {
        build 'TestAppJob'
    }
}
```

⇨ Sinh viên giải thích đoạn mã trên.

B5. Chọn Save và ta sẽ trờ về Jenkins dashboard của SamplePipeline

• Bước 3: Chạy SamplePipeline.

Chọn *Buid Now* để chạy job. Nếu code lỗi vào *Stage View* để xem log.

• Bước 4: Kiểm tra SamplePipeline output

Chọn liên kết *Permalinks* và tiếp tục chọn *Console Output*.

3. Test Python Function với unittest

Sử dụng unittest để kiểm tra function có chức năng tìm kiếm đệ quy JSON object. Hàm trả về giá trị được gắn thẻ bằng một khoá cố định. Lập trình viên thông thường thực hiện hành động trên JSON object trả về bởi lời gọi API.

Bài test này sẽ sử dụng 3 tập tin như sau:

Tập tin	Chú thích
recursive_json_search.py	Đoạn mã này gồm function json_search() mà ta muốn test
test_data.py	Đây là dữ liệu mà hàm json_search () đang tìm kiếm
test_json_search.py	Đây là tập ta sẽ tạo để kiểm tra hàm json_search () trong tập mã recursive_json_search.py.

• Bước 1: Đánh giá tập tin test_data.py

Mở *unittest/test_data.py* và kiểm tra nội dung của nó. Dữ liệu JSON này là điển hình của dữ liệu trả về. Dữ liệu mẫu đủ phức tạp để trở thành một bài test tốt. Ví dụ, nó có các loại dict và list xen kẽ.

```
┌──(kali㉿phaphajian)-[~/unittest]
└─$ more test_data.py
key1 = "issueSummary"
key2 = "XY&^$#*@!1234%^&"

data = {
  "id": "AWcvsjx864kVeDHDi2gB",
  "instanceId": "E-NETWORK-EVENT-AWcvsjx864kVeDHDi2gB-1542693469197",
  "category": "Warn",
  "status": "NEW",
  "timestamp": 1542693469197,
  "severity": "P1",
  "domain": "Availability",
  "source": "DNAC",
  "priority": "P1",
  "type": "Network",
  "title": "Device unreachable",
```

```
  "description": "This network device leaf2.abc.inc is unreachable from
controller. The device role is ACCESS.",
  "actualServiceId": "10.10.20.82",
  "assignedTo": "",
  "enrichmentInfo": {
    "issueDetails": {
      "issue": [
        {
          "issueId": "AWcvsjx864kVeDHDi2gB",
          "issueSource": "Cisco DNA",
          "issueCategory": "Availability",
          "issueName": "snmp_device_down",
          "issueDescription": "This network device leaf2.abc.inc is
unreachable from controller. The device role is ACCE
SS.",
          "issueEntity": "network_device",
          "issueEntityValue": "10.10.20.82",
          "issueSeverity": "HIGH",
          "issuePriority": "",
          "issueSummary": "Network Device 10.10.20.82 Is Unreachable
From Controller",
          "issueTimestamp": 1542693469197,
          "suggestedActions": [
            {
              "message": "From the controller, verify whether the last
hop is reachable.",
--More--(28%)
```


• Bước 2: Tạo hàm json_search() mà ta sẽ testing

Hàm của ta mong đợi một khoá và một JSON object làm tham số đầu vào và trả về list cặp key/value. Phiên bản hiện tại của function cần testing để xem chúng có hoạt động đúng như dự định. Mục đích của function này là nhập dữ liệu test trước. Sau đó, nó tìm kiếm dữ liệu phù hợp các biến key trong tập tin *test_data.py*. Nếu tìm ra một kết quả phù hợp, nó sẽ nối dữ liệu phù hợp vào list. Hàm *print()* ở cuối dùng để in nội dung list cho biến đầu tiền *key1 = "issueSummary.*

```
from test_data import *
def json_search(key,input_object):
    ret_val=[]
    if isinstance(input_object, dict): # Iterate dictionary
        for k, v in input_object.items(): # searching key in the dict
            if k == key:
                temp={k:v}
                ret_val.append(temp)
            if isinstance(v, dict): # the value is another dict so repeat
                json_search(key,v)
            elif isinstance(v, list): # it's a list
                for item in v:
                    if not isinstance(item, (str,int)): # if dict or
list repeat
                        json_search(key,item)
    else: # Iterate a list because some APIs return JSON object in a
list
        for val in input_object:
            if not isinstance(val, (str,int)):
                json_search(key,val)
    return ret_val
print(json_search("issueSummary",data))
```

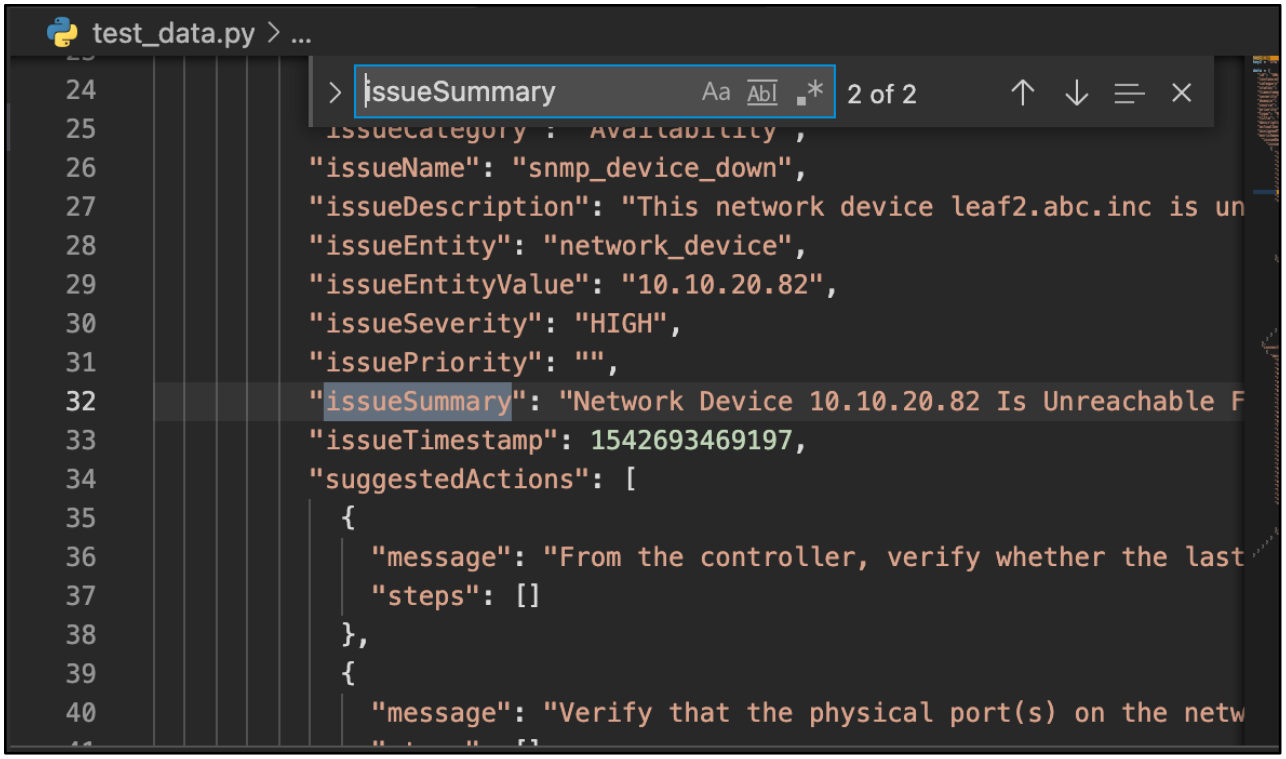
B1. Mở tập tin /unittest/recursive_json_search.py

B2. Copy code trên vào tập tin =)))

B3. Chạy code. Sẽ không xảy và nhậnn được list rỗng. Nếu hàm json_search() đúng, thì thì key "issueSummary" không có trong dữ liệu JSON được trả về từ việc gọi API.

```
┌──(kali㉿phaphajian)-[~/unittest]
└─$ python3 recursive_json_search.py
[]
```

B4. Vậy làm sao biết được hàm json_search() hoạt động đúng. Mở tập tin test_data.py và tìm kiếm “issueSummary”.



⇨ Đoạn code lỗi.

• Bước 3: Tạo một unit test để kiểm tra function có hoạt động đúng như dự kiến

B1. Mở tập tin test_json_search.py

B2. Dòng đầu tiên thêm thư viện unittest

```
import unittest
```

B3. Thêm các dòng import function đang testing cũng như dữ liệu JSON mà hàm sử dụng

```
from recursive_json_search import *
from test_data import *
```

B4. Bây giờ thêm đoạn mã vào class json_search_test. Mã tạo ra subclass TestCase của unittest framework. Class định nghĩa một số phương pháp kiểm tra được sử dụng trên function json_search() trong recursive_json_search.py. Lưu ý mỗi phương thức test bắt đầu với ***test_***, cho phép unittest framework tự động khám phá. Thêm các dòng sau vào cuối test_json_search.py

```
class json_search_test(unittest.TestCase):
    '''test module to test search function in
`recursive_json_search.py`'''
    def test_search_found(self):
        '''key should be found, return list should not be empty'''
        self.assertTrue([]!=json_search(key1,data))
    def test_search_not_found(self):
        '''key should not be found, should return an empty list'''
        self.assertTrue([]==json_search(key2,data))
    def test_is_a_list(self):
        '''Should return a list'''
        self.assertIsInstance(json_search(key1,data),list)
```

Trong code unittest, đang sử dụng ba phương pháp để test function tìm kiếm.

1. Đưa ra một key tồn tại sẵn trong JSON object, xem đoạn code testing có thể tìm thấy key như vậy không.
2. Đưa một key không tồn tại trong JSON object, xem liệu code testing có xác nhận rằng không có key nào tìm được.
3. Kiểm tra xem function có trả về một list, nó nên như vậy.

Để tạo các test, đoạn code sử dụng phương thức assert được tích hợp trong class unittest TestCase để kiểm tra các điều kiện. Phương thức assertTrue(x) sẽ kiểm tra một điều kiện có đúng không, assertIsInstance(a,b) kiểm tra xem a có phải là một thể hiện của kiểu b hay không. Thể hiện kiểu ở đây là list.

Lưu ý mỗi phương thức đều có chú thích trong nháy đôi (“). Đều này bắt buộc nếu kiểm tra thông tin output của phương pháp test khi chạy.

B5. Cuối tập tin, thêm phương thức unittest.main(), đều cho phép unittest chạy từ command line

```
if __name__ == '__main__':
     unittest.main()
```

• Bước 4: Chạy test để xem kết quả

B1. Chạy code test. Đầu tiên ta thấy list trống, thứ hai ta thấy ***.F.***. Dấu chấm có nghĩa test passed và điểm F có nghĩa là test failed. Do đó, lần kiểm tra thứ nhất đã test passed, lần kiểm tra thứ hai test failed, và lần kiểm tra thứ ba cũng test passed.

```
┌──(kali㉿phaphajian)-[~/unittest]
```

```
└─$  python3 test_json_search.py
[]
.F.
======================================================================
FAIL: test_search_found (__main__.json_search_test)
key should be found, return list should not be empty
----------------------------------------------------------------------
Traceback (most recent call last):
  File    "/home/kali/unittest/test_json_search.py",    line    9,    in
test_search_found
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=1)
```

B2. Để liệt từng bài test và kết quả của nó. Hãy thêm tuỳ chọn verbose (-v)

```
┌──(kali㉿phaphajian)-[~/unittest]
└─$       python3       -m       unittest       -v       test_json_search
1 ×
[]
test_is_a_list (test_json_search.json_search_test)
Should return a list ... ok
test_search_found (test_json_search.json_search_test)
key should be found, return list should not be empty ... FAIL
test_search_not_found (test_json_search.json_search_test)
key should not be found, should return an empty list ... ok

======================================================================
FAIL: test_search_found (test_json_search.json_search_test)
key should be found, return list should not be empty
----------------------------------------------------------------------
Traceback (most recent call last):
  File    "/home/kali/unittest/test_json_search.py",    line    9,    in
test_search_found
```

```
    self.assertTrue([]!=json_search(key1,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.010s

FAILED (failures=1)
```

• Bước 5: Điều tra và sửa lỗi đầu tiên trong đoạn mã recursive_json_search.py

key should be found, return list should not be empty ... FAIL => Khoá không tìm được. Tại sao? Ta nhìn vào đoạn code của hàm đệ quy thấy rằng *ret_val=[ ]* đang được. thực thi lặp đi lặp lại mỗi lần hàm được gọi. Điều này làm cho list luôn rỗng và kết quả tích luỹ từ lệnh *ret_val.append(temp)* mà đang thêm vào danh sách *ret_val=[ ]*

```
def json_search(key,input_object):
    ret_val=[]
    if isinstance(input_object, dict): # Iterate dictionary
        for k, v in input_object.items(): # searching key in the dict
            if k == key:
                temp={k:v}
                ret_val.append(temp)
```

® Bài tập (yêu cầu làm)

7. *Sinh viên sửa lại lỗi trên bằng cách di chuyển **ret_val=[]** và thực thi lại **recursive_json_search.py**, trình bày step-by-step có minh chứng.*

• Bước 6: Chạy test lại để xem tất cả các lỗi trong code đã được khắc phục chưa.

B1. Chạy lại unittest mà không có tùy chọn -v để xem liệu test_json_search có trả về lỗi không. Kết quả **..F,** nghĩa là lần 3 test failed.

```
┌──(kali㉿phaphajian)-[~/unittest]
└─$ python3 -m unittest test_json_search
[{'issueSummary': 'Network Device 10.10.20.82 Is Unreachable From
Controller'}]
..F
======================================================================
FAIL: test_search_not_found (test_json_search.json_search_test)
key should not be found, should return an empty list
```

```
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/home/kali/unittest/test_json_search.py", line 12, in
test_search_not_found
    self.assertTrue([]==json_search(key2,data))
AssertionError: False is not true

----------------------------------------------------------------------
Ran 3 tests in 0.002s

FAILED (failures=1)
```

B2. Mở tập ti *test_data.py* và tìm từ khoá *issueSummary* (key1). Ta sẽ thấy key1 xuất hiện trong dữ liệu 1 lần và key2 với giá trị *XY&^$#*@!1234%^&* không tìm thấy trong dữ liệu. Tức lần lần test 3 kiểm tra nó không tồn tại trong đó. Comment trong test 3 **key should not be found, should return an empty list**.. Tuỳ nhiên hàm trả về danh sách rỗng.

• Bước 7: Điều tra và sửa lỗi đầu hai trong đoạn mã recursive_json_search.py

Xem lại code *recursive_json_search.py* một lần nữa. Nếu bạn đã sửa lỗi câu trên thành công theo hướng gợi ý thì ta tiếp tục sửa tiếp

® **Bài tập (yêu cầu làm)**

8. *Sinh viên hãy tìm ra nguyên nhân, tìm cách sửa lỗi trên và thực thi lại* ***unittest****, trình bày step-by-step có minh chứng.*

C. YÊU CẦU & ĐÁNH GIÁ

1. Yêu cầu

- Sinh viên tìm hiểu và thực hành theo hướng dẫn.
- Sinh viên báo cáo kết quả thực hiện và nộp bài bằng **1 trong 2 hình thức**:

h) Cách 1: Báo cáo chi tiết:

Báo cáo cụ thể quá trình thực hành (có ảnh minh họa các bước) và giải thích các vấn đề kèm theo. Trình bày trong file PDF theo mẫu có sẵn tại website môn học.

i) Cách 2: Video trình bày chi tiết:

Quay lại quá trình thực hiện Lab của sinh viên kèm thuyết minh trực tiếp mô tả và giải thích quá trình thực hành. Upload lên **Youtube** và chèn link vào đầu báo cáo theo mẫu. **Lưu ý:** *Không chia sẻ ở chế độ Public trên Youtube.*

Đặt tên file báo cáo theo định dạng như mẫu:

[Mã lớp]-LabX_MSSV1-Tên SV1_MSSV2 –Tên SV2

Ví dụ*: [NT101.I11.1]-Lab1_14520000-Viet_14520999-Nam.*

- Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
- Nộp báo cáo trên theo thời gian đã thống nhất tại website môn học.

2. Đánh giá:

- Sinh viên hiểu và tự thực hiện được bài thực hành, đóng góp tích cực tại lớp.
- Báo cáo trình bày chi tiết, giải thích các bước thực hiện và chứng minh được do nhóm sinh viên thực hiện.
- Hoàn tất nội dung cơ bản và có thực hiện nội dung *mở rộng – cộng điểm* (với lớp **ANTN**).

Kết quả thực hành cũng được đánh giá bằng kiểm tra kết quả trực tiếp tại lớp vào cuối buổi thực hành hoặc vào buổi thực hành thứ 2.

Lưu ý: Bài sao chép, nộp trễ, "*gánh team*", … sẽ được xử lý tùy mức độ.

HẾT

Chúc các bạn hoàn thành tốt!