Base pipeline configuration with no security

Let's start with a base pipeline configuration so that you can see a pipeline without a security integration. I've been working a lot with Python applications these days, so below is an example pipeline configuration for a Python app that doesn't include any security actions.

```
version: 2.1
jobs:
  build_test:
    docker:
      - image: circleci/python:3.7.4
    steps:
      - checkout
      - run:
          name: Install Python Dependencies
          command: |
            echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV && source $BASH_ENV
            pip install --user -r requirements.txt
      - run:
          name: Run Unit Tests
          command: |
            pytest
  build_push_image:
    docker:
      - image: circleci/python:3.7.4
    steps:
      - checkout
      - setup_remote_docker:
          docker_layer_caching: false
      - run:
          name: Build and Push Docker image to Docker Hub
          command: |
            echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV
            echo 'export TAG=${CIRCLE_SHA1}' >> $BASH_ENV
            echo 'export IMAGE_NAME=orb-snyk' >> $BASH_ENV && source $BASH_ENV
            pip install --user -r requirements.txt
            pyinstaller -F hello_world.py
            docker build -t $DOCKER_LOGIN/$IMAGE_NAME -t
$DOCKER_LOGIN/$IMAGE_NAME:$TAG .
            echo $DOCKER_PWD | docker login -u $DOCKER_LOGIN --password-stdin
            docker push $DOCKER_LOGIN/$IMAGE_NAME
```

```
workflows:
  build_test_deploy:
    jobs:
      - build_test
      - build_push_image:
          requires:
            - build_test
```

This pipeline configuration accomplishes the following:

- Installs application dependencies defined in the requirements.txt manifest file
- Executes the application's unit tests
- Creates/builds a new Docker image for the application
- Publishes the newly created Docker image to the Docker Hub registry for later use

The goal of this pipeline is to build, test, and deploy the code via a Docker image. At no point in this pipeline are there any security or vulnerability scans.

Security enabled pipeline - the Snyk app scan

Now you've seen a glimpse of an "insecure" pipeline, and it should make your skin crawl. Next, I'm going to show you an example of a security-enabled pipeline configuration.

```
version: 2.1
orbs:
  snyk: snyk/snyk@0.0.8
jobs:
  build_test:
    docker:
      - image: circleci/python:3.7.4
    steps:
      - checkout
      - run:
          name: Install Python Dependencies
          command: |
            echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV && source $BASH_ENV
            pip install --user -r requirements.txt
      - snyk/scan
      - run:
          name: Run Unit Tests
          command: |
            pytest
```

```
  build_push_image:
    docker:
      - image: circleci/python:3.7.4
    steps:
      - checkout
      - setup_remote_docker:
          docker_layer_caching: false
      - run:
          name: Build and Push Docker image to Docker Hub
          command: |
            echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV
            echo 'export TAG=${CIRCLE_SHA1}' >> $BASH_ENV
            echo 'export IMAGE_NAME=orb-snyk' >> $BASH_ENV && source $BASH_ENV
            pip install --user -r requirements.txt
            pyinstaller -F hello_world.py
            docker build -t $DOCKER_LOGIN/$IMAGE_NAME -t
$DOCKER_LOGIN/$IMAGE_NAME:$TAG .
            echo $DOCKER_PWD | docker login -u $DOCKER_LOGIN --password-stdin
            docker push $DOCKER_LOGIN/$IMAGE_NAME
workflows:
  build_test_deploy:
    jobs:
      - build_test
      - build_push_image:
          requires:
            - build_test
```

Before I explain what's going on in this pipeline, I want to mention that Snyk has many capabilities, but for the purposes of this post, I'm only going to cover the application and Docker image scans. The above example demonstrates the application scan. Now, I'll explain the new parts that I included to make the pipeline more secure.

The pipeline block below uses the Snyk orb to easily integrate the Snyk tool into the pipeline. This block is equivalent to an import or include statement in a scripting or programming language. In this block, you're also declaring the version of the Snyk orb you'd like to use.

```
version: 2.1
orbs:
  snyk: snyk/snyk@0.0.8
```

The next pipeline block defines the Docker image used to run the build. It then does a checkout or "git clone" of your source code into the container. Following that, the run: block will install the

dependencies listed in the requirements.txt file. This file lists all of your application libraries and dependencies which can be considered a [Software Bill of Materials (SBOM)] specific to the Python aspects of the project. It also feeds the list of software to Snyk, so that it knows what to scan and test.

```
jobs:
  build_test:
    docker:
      - image: circleci/python:3.7.4
    steps:
      - checkout
      - run:
          name: Install Python Dependencies
          command: |
            echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV && source $BASH_ENV
            pip install --user -r requirements.txt
```

Below you'll find the contents of the requirements.txt file that was previously discussed.

```
requirements.txt
1  flask == 0.12.4
2  pyinstaller == 3.5
3  pytest == 5.2.1
4  pylint == 2.4.2
5
```

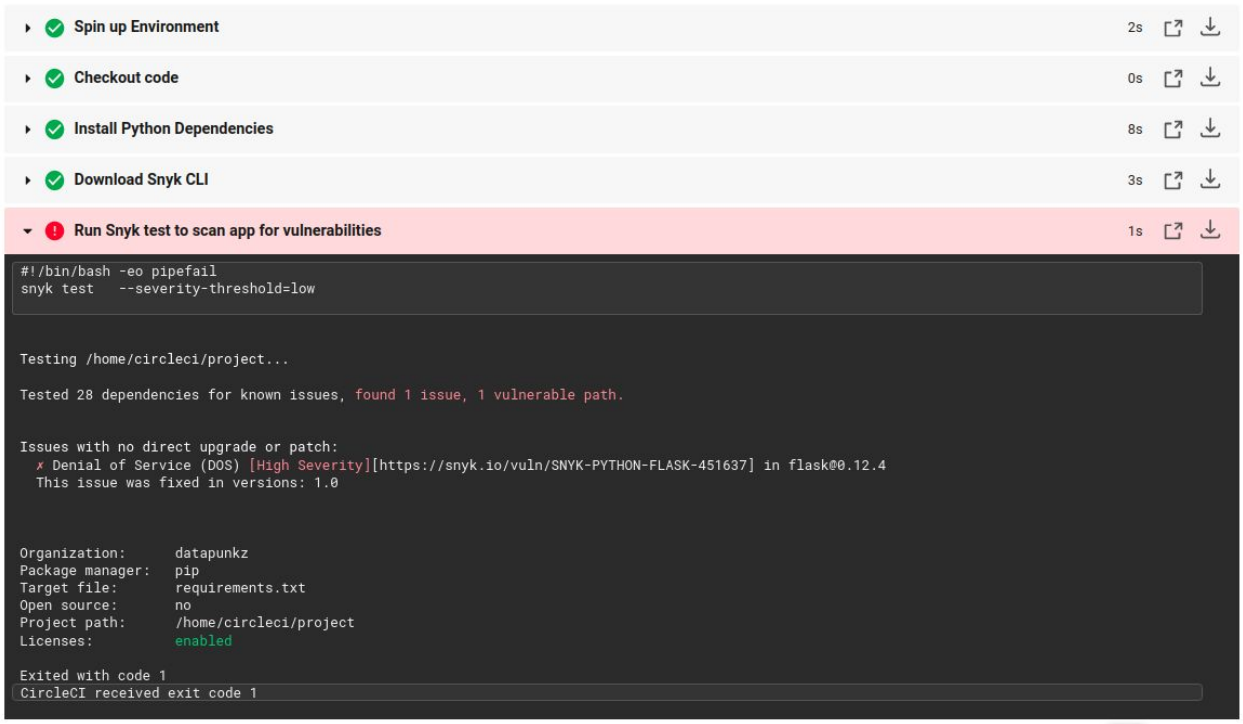
The next block is where we execute some DevSecOps action within the pipeline. - snyk/scan calls the scan command from the Snyk orb. It will read the requirements.txt file, and then compare that list of software against the Snyk vulnerability databases to look for any matches. If there are any matches, Snyk will flag it and fail this segment of the pipeline. The goal here is to alert teams to security issues as early as possible so that they can be quickly mitigated and the CI/CD process can securely continue.

```
      - snyk/scan
      - run:
          name: Run Unit Tests
          command: |
            pytest
```

The remainder of the example pipeline configuration deals with the Docker image build segments.

After the Snyk application security scan is complete, your build will pass if there are no vulnerabilities detected. If the scan detects a vulnerability, the build will fail. This is the Snyk orb's default behavior (see the Snyk orb parameters for more details on these parameters). Along with the pipeline failing after the scan, Snyk will provide a detailed report on why the build failed.

Below is an example of a failed application security scan reported in the CircleCI dashboard.



The scan results provide useful details regarding the security scan. They show that the application is using a vulnerable version of the Flask library, 0.12.4. The results also suggest a fix. The fix requires upgrading Flask to the newer 1.0 version.

You could also see results for this failed scan from the Snyk dashboard. That dashboard has an even greater level of detail. The image below shows an example of the Snyk dashboard.

datapunkz/orb-snyk:requirements.txt

Overview History Settings

Snapshot taken by snyk.io 20 hours ago.

View most recent snapshot

Vulnerabilities:	2 via 2 paths	Dependencies:	28	Source:	GitHub
Taken by:	Web	Python::	Python 3.7.4	Repository:	orb-snyk
Branch:	master	Manifest:	requirements.txt	Imported by:	Angel angel@circleci.com
Project owner:	Add a project owner				

Issues Dependencies

Search issues...

Severity

- High 2
- Medium 0
- Low 0

Status

- Open 2
- Patched 0
- Ignored 0

HIGH SEVERITY

Denial of Service (DOS)

Vulnerable module: flask

Introduced through: flask@0.12

Detailed paths

- **Introduced through:** project@0.0.0 › flask@0.12

Overview

Flask is a lightweight WSGI web application framework

Affected versions of this package are vulnerable to Denial of Service (DOS). The package allows for unsafe encoded JSON data to be decoded.

More about this issue

The security scan provides insight into the application's vulnerabilities and useful suggestions on how to mitigate the issue so that teams can fix them and move on to the next task. Some fixes are more involved than others, but being alerted to the issues is a very powerful feature.

Security enabled pipeline - the Snyk Docker image scan

I've discussed the Snyk application scanning capabilities above and now I want to discuss the Docker image scanning capabilities that are also easily integrated into CI/CD pipelines. The build_push_image: block shown below is from the previous **Base pipeline configuration with no security** example.

```
 build_push_image:
   docker:
     - image: circleci/python:3.7.4
   steps:
     - checkout
     - setup_remote_docker:
         docker_layer_caching: false
     - run:
         name: Build and Push Docker image to Docker Hub
         command: |
           echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV
```

```
          echo 'export TAG=${CIRCLE_SHA1}' >> $BASH_ENV
          echo 'export IMAGE_NAME=orb-snyk' >> $BASH_ENV && source $BASH_ENV
          pip install --user -r requirements.txt
          pyinstaller -F hello_world.py
          docker build -t $DOCKER_LOGIN/$IMAGE_NAME -t
$DOCKER_LOGIN/$IMAGE_NAME:$TAG .
          echo $DOCKER_PWD | docker login -u $DOCKER_LOGIN --password-stdin
          docker push $DOCKER_LOGIN/$IMAGE_NAME
```

The run block above has no security scans integrated. This increases the risk of exposure to severe vulnerabilities within the Docker image. Synk's Docker image scan is an essential feature for checking your Docker images for vulnerabilities. Below is an example of how to integrate the Snyk Docker image scans into your pipeline configurations.

```
 build_push_image:
   docker:
     - image: circleci/python:3.7.4
   steps:
     - checkout
     - setup_remote_docker:
         docker_layer_caching: false
     - run:
         name: Build and Scan Docker image
         command: |
           echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV
           echo 'export TAG=${CIRCLE_SHA1}' >> $BASH_ENV
           echo 'export IMAGE_NAME=orb-snyk' >> $BASH_ENV && source $BASH_ENV
           pip install --user -r requirements.txt
           pyinstaller -F hello_world.py
           docker build -t $DOCKER_LOGIN/$IMAGE_NAME -t
$DOCKER_LOGIN/$IMAGE_NAME:$TAG .
     - snyk/scan:
         fail-on-issues: true
         monitor-on-build: true
         docker-image-name: $DOCKER_LOGIN/$IMAGE_NAME:$TAG
         target-file: "Dockerfile"
         project: ${CIRCLE_PROJECT_REPONAME}/${CIRCLE_BRANCH}-app
         organization: ${SNYK_CICD_ORGANIZATION}
     - run:
         name: Push Docker image to Docker Hub
         command: |
           echo $DOCKER_PWD | docker login -u $DOCKER_LOGIN --password-stdin
```

```
        docker push $DOCKER_LOGIN/$IMAGE_NAME
```

The pipeline configuration block example above is significantly different than the original and I'll discuss those differences now.

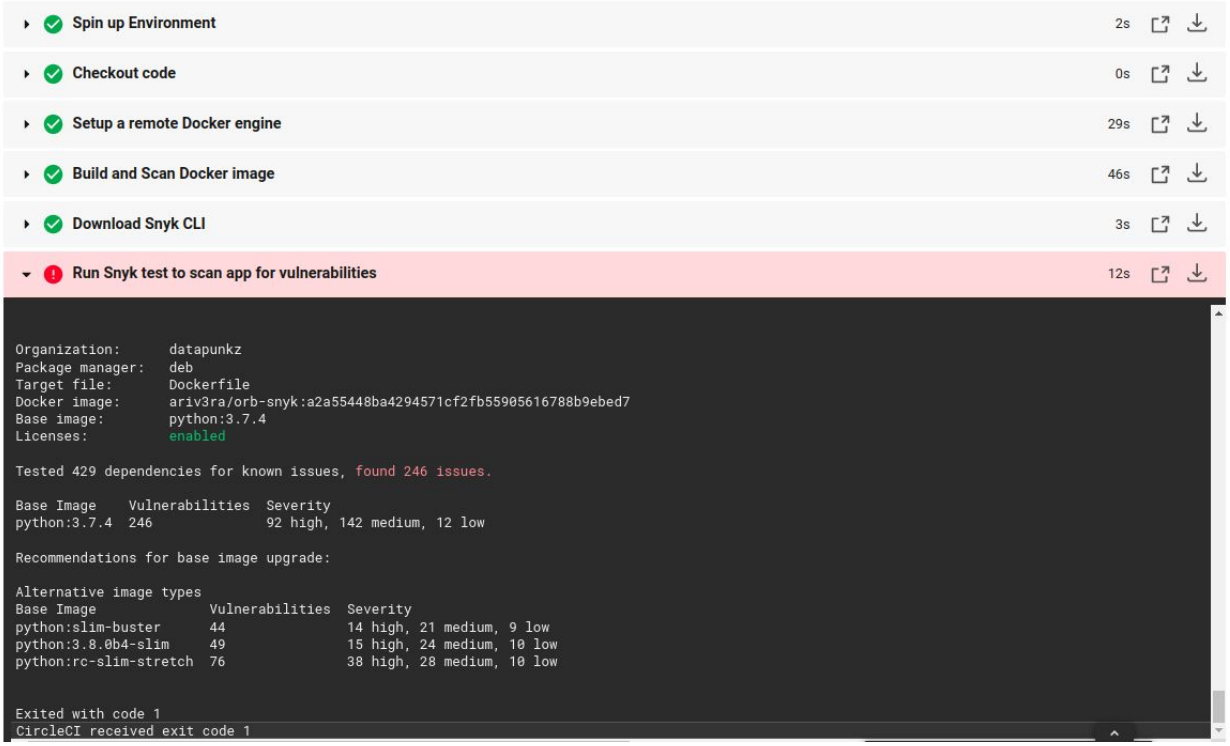
The block below is a new - run: block that sets up some environment variables that are used to name and version the Docker image being built. The last line in this block builds the image.

```
    - run:
        name: Build and Scan Docker image
        command: |
          echo 'export PATH=~$PATH:~/.local/bin' >> $BASH_ENV
          echo 'export TAG=${CIRCLE_SHA1}' >> $BASH_ENV
          echo 'export IMAGE_NAME=orb-snyk' >> $BASH_ENV && source $BASH_ENV
          pip install --user -r requirements.txt
          pyinstaller -F hello_world.py
          docker build -t $DOCKER_LOGIN/$IMAGE_NAME -t
$DOCKER_LOGIN/$IMAGE_NAME:$TAG .
```

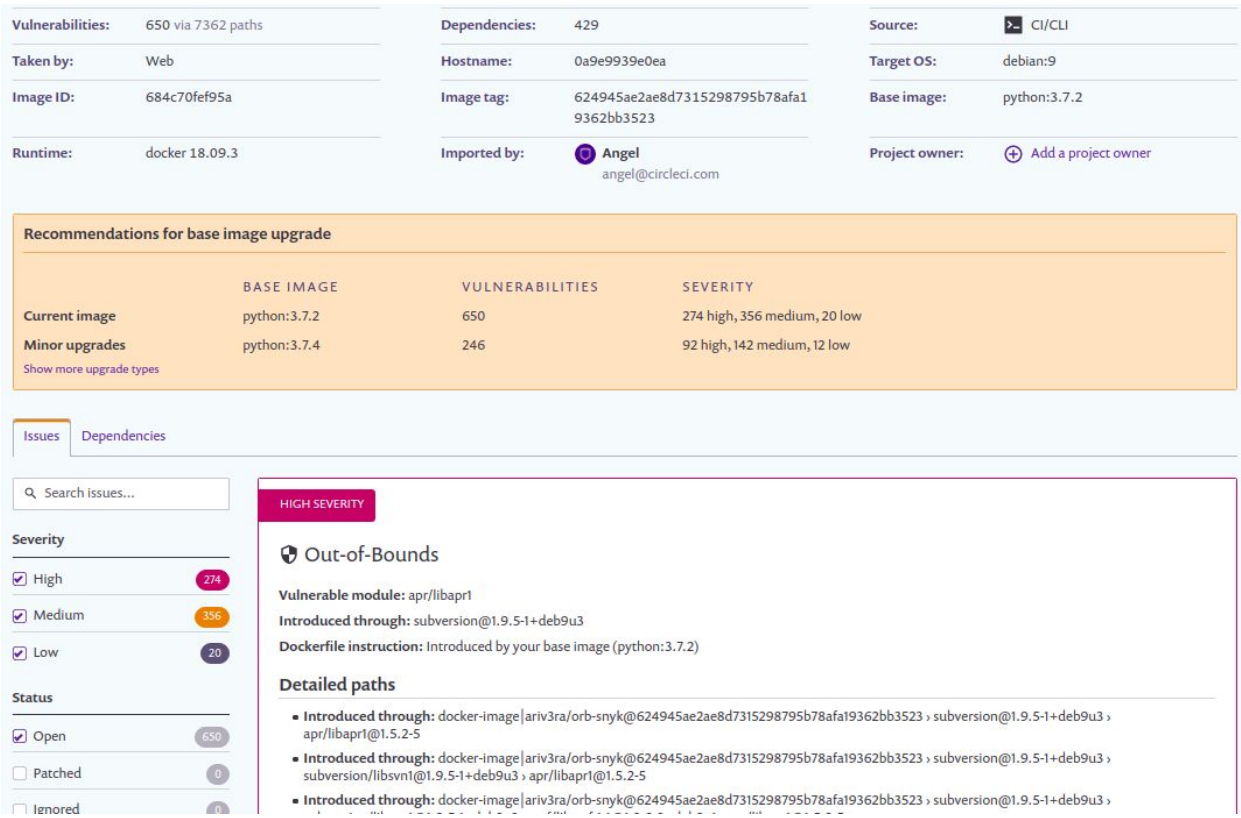
The next block below is where the Docker image security scan is declared and executed. - snyk/scan: is the same command used in the previous app security scan with some differences. In order to execute the Docker image scans from the - snyk/scan: command, you have to declare and set values for the docker-image-name: and target-file: parameters. Again, I suggest that you familiarize yourself with the [Snyk orb parameters] to understand the tool's capabilities.

```
    - snyk/scan:
        fail-on-issues: true
        monitor-on-build: true
        docker-image-name: $DOCKER_LOGIN/$IMAGE_NAME:$TAG
        target-file: "Dockerfile"
        project: ${CIRCLE_PROJECT_REPONAME}/${CIRCLE_BRANCH}-app
        organization: ${SNYK_CICD_ORGANIZATION}
```

Below is an example of a failed Docker scan within a pipeline as reported in the CircleCI dashboard. The details shown appear similar to the app scan but are definitely different. It shows that vulnerabilities in the Docker image stem from the base image which is python:3.7.4 This image is published by the language maintainers.



Like the Snyk app scan, more details regarding the failed Docker image scan can be found from the Snyk dashboard. The Snyk dashboard will have all the details needed to mitigate the detected vulnerabilities. An example of a failed Docker image scan from the Snyk dashboard is shown below.



The last pipeline block accesses Docker Hub and pushes and publishes the scanned Docker image to the Docker Hub registry. This is the last command executed in the build_push_image: block.

```
- run:
    name: Push Docker image to Docker Hub
    command: |
      echo $DOCKER_PWD | docker login -u $DOCKER_LOGIN --password-stdin
      docker push $DOCKER_LOGIN/$IMAGE_NAME
```