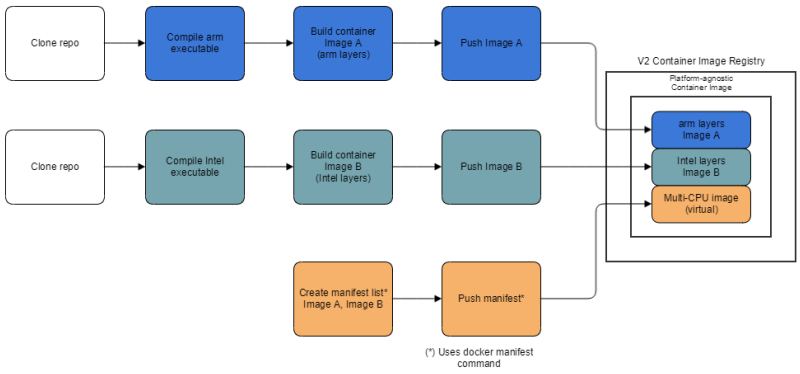
Building a Platform-Agnostic Container Image

Process

In general, the process to build a platform-agnostic container image follows the flow depicted on the following figure.

The commands needed to implement the flow are described, using an example, in the next section.



Example using a Python Micro-Service

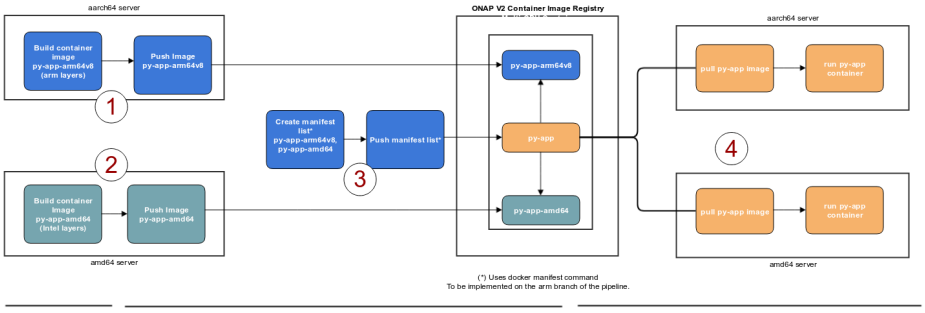
Main steps

The following diagram captures the main steps we need to take to enable platform-agnostic containers:

(1) and (2) Build and push container images for each platform.

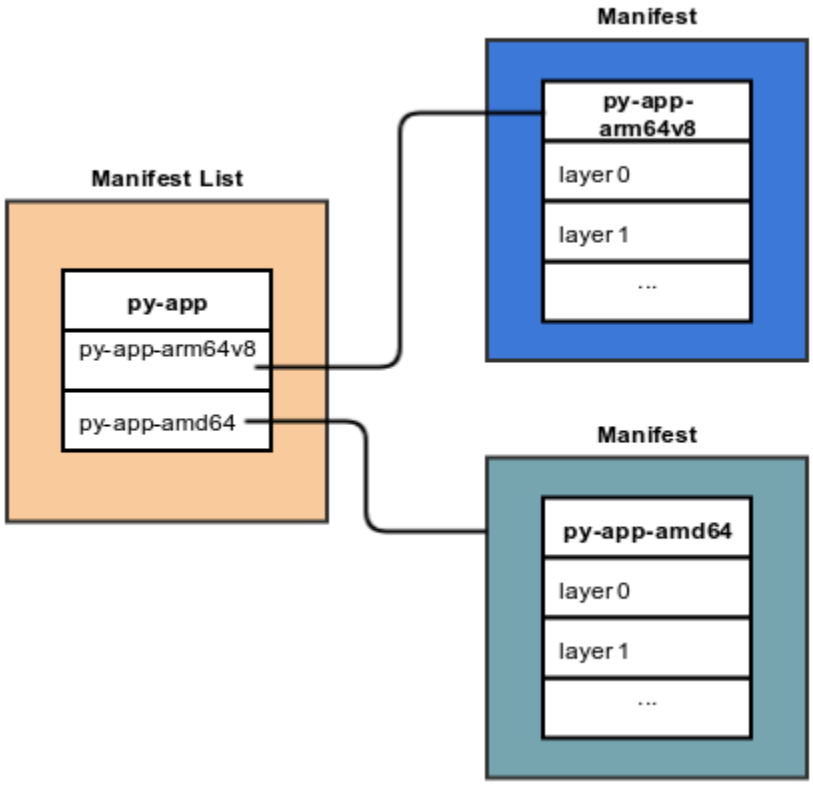
(3) Create and push a manifest list for the images above

(4) Pull and run the exact same image/tag on different platforms.



Manifest List and Image Layers

Digging a little bit deeper into step (4), the following diagram shows the relationship between a manifest list and image manifests for our platform-agnostic image (tag).



The following sections describe the commands needed to create a multi-cpu architecture container image. Let's call the image onap/py-app.

Note that this flow could be used by ONAP developers during the development-test-debug process.

For the release process, the flow will implemented using CI/CD pipelines as shown in the next section.

Source code

Code structure

```
.
 app
    main.py
    requirements.txt
 Dockerfile
```

Python App

```
from flask import Flask
import platform
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello ONAP. I am a Python service running on " + platform.machine()

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True, port=5000)
```

Requirements

```
Flask==0.10.1
```

Dockerfile

```
FROM python:2.7-alpine

LABEL maintainer="adolfo@orangemonk.net"

# Keep it simple

COPY ./app /app
WORKDIR /app
RUN pip install -r requirements.txt

ENTRYPOINT ["python"]
CMD ["main.py"]
```

Build arm image (A)

Log into an arm server, copy the code into the structure depicted above.

cd to the root of the code tree above, then execute

```
docker build -t onap/py-app-arm64 .
```

Push arm image to the registry

Once the image has been successfully built, push it to the repository.

Note that if you are using a private repository, you might need to "docker tag" the image before executing the next command.

```
docker push onap/py-app-arm64v8:latest
```

Build Intel image (B)

Log into an intel server, setup the code structure as before.

Let's now repeat the process for the intel layers of the multi-cpu container image.

```
docker build -t onap/py-app-amd64 .
```

Push Intel image to the registry

```
docker push onap/py-app-amd64:latest
```

Create a manifest list for image A and image B

Now that we have built and pushed layers for each cpu architecture, we can create the manifest list to put the final container image together with

```
docker manifest create onap/py-app \
                       onap/py-app-arm64v8 \
                       onap/py-app-amd64
```

Verify that the manifest describes a platform-agnostic container image.

```
docker manifest inspect --verbose onap/py-app
```

Verify that the manifest actually represents a multi-cpu architecture by looking at the different "platform" entries.

Notice how, in this case, the manifest shows layers for both arm and Intel cpu architectures.


```
[
    {
        "Ref": "docker.io/onap/py-app-amd64:latest",
        "Descriptor": {
            "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
            "digest": "sha256:4b1b8361d47770668ff4c728dc388f376158d9e8d82c0b4cdb22147cc8ca1be6",
            "size": 1579,
            "platform": {
                "architecture": "amd64",
                "os": "linux"
            }
        },
        "SchemaV2Manifest": {
            "schemaVersion": 2,
            "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
            "config": {
                "mediaType": "application/vnd.docker.container.image.v1+json",
                "size": 6616,
                "digest": "sha256:f2947f4a0e9d5d0a4ccf74a7b3ad94611a8921ca022b154cabcad9e453ea6ec5"
            },
            "layers": [
                {
```

```
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 2206931,
                "digest": "sha256:4fe2ade4980c2dda4fc95858ebb981489baec8c1e4bd282ab1c3560be8ff9bde"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 308972,
                "digest": "sha256:1b23fa3ccba56eced7bd53d424b29fd05cd66ca0203d90165e988fdd8e71fed7"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 17711780,
                "digest": "sha256:b714494d7662fbb89174690cefce1051117ed524ec7995477b222b8d96fb8f0c"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 1776866,
                "digest": "sha256:1098418f3d2f83bcccdbb3af549d75d9a5e9c37420e5c0d474fd84b022f6c995"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 360,
                "digest": "sha256:ca727cee7c2469ab6edb7ca86378985b3747938a325ddb7d90f3b85a3d14b34f"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 4315553,
                "digest": "sha256:767bbe8ba063767093b0350e8d1b86b438c677e3884c2a21851c00970d88317c"
            }
        ]
    }
},
{
    "Ref": "docker.io/onap/py-app-arm64v8:latest",
    "Descriptor": {
        "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
        "digest": "sha256:8a06f997353177dae82d7e01bc3893b2f05c6ac27b317655df3ca2287f9b83a9",
        "size": 1786,
        "platform": {
            "architecture": "arm64",
            "os": "linux"
        }
    },
    "SchemaV2Manifest": {
        "schemaVersion": 2,
        "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
        "config": {
            "mediaType": "application/vnd.docker.container.image.v1+json",
            "size": 6864,
            "digest": "sha256:b097a21c92a9a0dde06f9b36bf10def56a028b3b9b1617d6d6c6a8559d14d9d7"
        },
        "layers": [
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 2099514,
                "digest": "sha256:47e04371c99027fae42871b720fdc6cdddcb65062bfa05f0c3bb0a594cb5bbbd"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 176,
                "digest": "sha256:b4103359e1ecd9a7253d8b8a041d4e81db1ff4a5e1950bc0e02305d221c9e6c2"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 308518,
                "digest": "sha256:92079a442932f09a622f4a0f863e5cc2f6e0471a98e5121fa719d2a276440386"
            },
            {
                "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                "size": 18605961,
                "digest": "sha256:f1fc35806f46347993a2cd1eb7f7dd7837b0bef0392c8e2c973b24c02ad874a9"
```

```
                },
                {
                    "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                    "size": 1786389,
                    "digest": "sha256:c2983ee3d71107a9a0bc1996fc3a58e050026995fad4aee9d72539153db1df3d"
                },
                {
                    "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                    "size": 391,
                    "digest": "sha256:44c3eae5ed66bb040727a64fd78573fe6cc4a94a9317d5cd6f39e53332c2ae21"
                },
                {
                    "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
                    "size": 4305558,
                    "digest": "sha256:8daa79c3024a565c320ff69990ad48273937cc3f6f0cdb324e086c268cf6245e"
                }
            ]
        }
    }
]
```

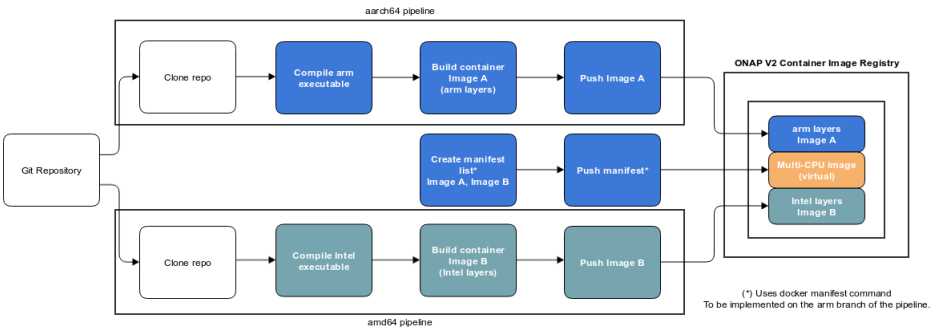
Push the manifest list to the registry

```
docker manifest push onap/py-app
```

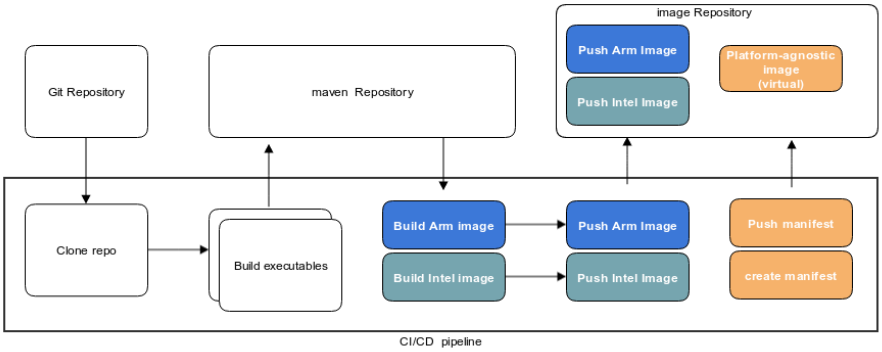
Building Multi-CPU container images using CI/CD pipelines

The following diagram depicts a CI/CD flow that implements the production of multi-cpu architecture container images.

Although the flow shows a pipeline for two branches: arm-linux and intel-linux, the model can be extended --given the hardware and software resources-- to any number of platforms.



The following view illustrates intermediate step of building and storing executable and sequence of image processing



Intermediate Image Naming Convention

As described above, platform-agnostic support requires platform specific images that are put together to expose a multi-platform image. Platform architectures include, among others, `arm`,`intel`, `mips`, `ppc64le`, and `s390x`

These platform-specific image names are typically used only by developers that build the images. ONAP end users should only use the aggregate tag but can still inspect the image using docker manifest.

The following is the recommended naming convention for ONAP platform-specific images that will be produced by the different pipelines (<onap-image-name>). This convention is aligned with existing industry standards and naming conventions (amd64, arm64v8).

Architecture	OS	Variant	Image Name
amd64	Linux		<onap-image-name>-amd64
arm64	Linux	v8	<onap-image-name>-arm64v8
mips	Linux		<onap-image-name>-mips

Note: This table does not contain a exhaustive list of options and must only be used as an image naming guide. Because ONAP is vendor-agnostic, the list is not a statement of what architectures or OSs ONAP must support.