Production Docker Image for Apache Airflow

Polidea

Airflow Summit 2020 - 14.07.2020
Production Container Image for Apache Airflow

Polidea

Airflow Summit 2020 - 14.07.2020
Hi!

Jarek Potiuk

Apache Airflow:
PMC Member and Committer

Polidea:
Principal Software Engineer (ex-CTO)

Airflow Summit:
Co-Organizer: Content (Lead)

@higrys
Intro
What questions will be answered?

- Context
  - What container images are and why they are important?

- Status
  - How it looked like so far?
  - How it is going to look like now?

- Internals
  - What is in the image?
  - How we test the image?

- Usage
  - How to extend Airflow Image?
  - How to customize Airflow Image?
  - How you can use the Image?

- Future
  - What’s next?
What this talk is NOT about?

- Basic container image knowledge
  - [https://docker-curriculum.com/](https://docker-curriculum.com/)

- Details of CI container image of Airflow
  - [https://github.com/apache/airflow/blob/master/IMAGES.rst](https://github.com/apache/airflow/blob/master/IMAGES.rst)

- Details of how Kubernetes Airflow integrate
  - "Airflow on Kubernetes" by Michael Hewitt
    [https://www.crowdcast.io/e/airflowsummit/6](https://www.crowdcast.io/e/airflowsummit/6)

- Details on deploying Airflow with the image
Who is the talk for?

- You want to deploy Airflow using container images
- You want to contribute to Airflow in DevOps area
- You want to learn about best practices of using Airflow Containers
- You are a curious person that want to learn something new
What is a container?

- **Standard** unit of software.
  - OCI: https://opencontainers.org/
- Packages code and its dependencies
- Lightweight execution package of software
- Container images - binary packages
Container ≠ Docker

- Docker is a command line tool
  - Building, Running, Sharing containers
- Docker Engine runs containers
- Alternatives: rkt, containerd, runc, podman, lxc, …
- DockerHub.com is popular container registry
- Alternatives: GitHub, GCR, ECR, ACR
Context: What is Container file

- Specify base image
- Run commands
- Copy files
- Set working directory
- Define entrypoint
- Define default command

```
FROM ubuntu:18.04
COPY . /app
RUN make /app && make install
WORKDIR /bin/project
ENTRYPOINT ["/bin/project"]
CMD ["--help"]
```
Context: Container Lifecycle: Build

Container Image file (Dockerfile) → Build → Container image → Container registry

Container execution engine
Context: Container Lifecycle: Run

- Container Image file (Dockerfile)
- Container execution engine
  - Run
- Container image
- Container registry
Context: Container Lifecycle: Push

- Container Image file (Dockerfile)
- Container execution engine
- Container image
- Push
- Container registry

Cloud
Context: Container Lifecycle: Pull

Container Image file (Dockerfile) → Container execution engine → Pull → Container registry
Why containers are important?

- Predictable, consistent development & test environment
- Predictable, consistent execution environment
- Lightweight but isolated: sandboxed view of the OS isolated from others
- Build once: run anywhere
- Kubernetes runs containers natively
- Bridge: “Development -> Operations”
History of Containers in Airflow: CI

- Used for CI for > 2 years: Gerardo Curiel
- Optimized and incorporated by Breeze 1.5 years ago or so
- Docker Compose as execution engine
- Slimmed down recently (Thanks Ash!)
- Optimized for development use
Status

**History of Containers in Airflow: Prod**

- Puckel image created by Matthieu "Puckel_" Roisil (Thanks Matthieu!)
  - Used by many users in production
  - Used by the publicly available Helm Chart (not managed by community)

- Official Production Image (managed by community)
  - Alpha Quality community image in 1.10.10
  - Beta Quality community image in 1.10.11 (now!)
State of the Official Production image

- Beta Quality - usable for production
- Most important feedback incorporated
- Already used in production
- Public Helm Chart switched to the Official Production Image
- Community Helm Chart (donated by Astronomer!) uses it for testing
- Stable version in v1-10-stable, development in master
Internals: DockerHub releases

Released image
- ~210 MB compressed size
- Python: 2.7, 3.5, 3.6, 3.7, 3.8
- 1.10.11 = Python 3.6
- manually released
- using “1.10.11” tag
- latest = 1.10.11
- docker pull apache/airflow
Internals: Releasing the image

Container Image or Container File?

- Apache Software Foundation releases sources, not binaries
- Binaries can only be released for convenience of users
- Binaries must be rebuildable from released sources (PyPI, for example)
- Users should be able to build the software they need
- Should we release Container Image, Container File, or both?
Features of the production image

- Optimised for size (Compressed: ~230MB, ~800 MB on disk)
- Python 3.6, 3.7, 3.8 (2.0 and 1.10.*) , 2.7, 3.5 (1.10.*)
- Extras installed:
  - async, aws, azure, celery, dask, elasticsearch, gcp, kubernetes, mysql, postgres, redis, slack, ssh, statsd, virtualenv
- OpenShift compatible (dynamic uid allocation)
- Gunicorn using shared memory (optimised parallelism)
Features of the production image file

- Builds optimised image
- Highly customizable (ARGs)
- Multi segmented (build + main)
Internals: build image

Build image

- Pass arguments
- Define variables
- Install apt dependencies (with dev ones)
- Install airflow (sources, pip, github): `--user`
- Include constraints
- Transpile website (yarn)
- ~700 MB compressed, ~2GB on disk
- Root user

~ 730 modules
~ 360 MB

(side comment)

~700 MB compressed, ~2GB on disk

Root user
Internals: main image

Main image

- Pass arguments/ define variables
- Install apt dependencies (without dev!)
- Add user
- Uses root group (OpenShift)
- Copy Airflow
- Copy DAGs (optionally)
- Copy entrypoint and clean-logs
- Access to /etc/passwd
- Embed dags (for tests)
- Optimized Gunicorn parallelism
- Set working dir
- Exposes port
- Set user
- Entrypoint and command
- ~230 MB compressed, ~800 MB on disk
Internals: entrypoint

- Creates user dynamically if missing (OpenShift)
- Fallbacks to sqlite metadata
- Waits until metadata DB is up
- Waits until broker DB is up
- If “bash” or “python” -> runs command
- Else execute airflow command
Internals: .dockerignore

- Ignores everything by default
- You must explicitly include what you want by “!”
- You can further exclude specific subdirectories/patterns
- We generate a lot of stuff in airflow sources
- Sending big context to Docker engine takes time
- You avoid accidental inclusion of unneeded artifacts
How we test the image?

- The image and chart are part of Apache Airflow monorepo
- We build the image with every PR (dependencies)
- We use it in the Kubernetes tests for master (Helm Chart integration)
- We will use released images in the Helm Chart (backward compatibility)
- We will add more tests for various Helm configurations
Container Images Usage
Usage: Extending Airflow image - use released image

docker build . -t yourcompany/airflow:1.10.11-BUILD_ID

FROM apache/airflow:1.10.11

# change to root user temporarily
USER root

# Optionally install your own apt dependencies
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
    emacs \
    && apt-get autoremove -yqq --purge \
    && apt-get clean \
    && rm -rf "/var/lib/apt/lists/*"

# Change back to the airflow user
USER airflow

# Add extra dependencies
RUN pip install --user numpy

# Embed DAGs (Optionally) - DAGs can be baked in but also
# they can be git-synced or mounted from shared volume
COPY --chown=airflow:root dags-folder ${AIRFLOW_HOME}/dags/
### Usage

---

### Extending image - Pros & Cons

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Use released images</td>
<td>● Potentially bigger size</td>
</tr>
<tr>
<td>● Simple build command</td>
<td>● Predefined extras only</td>
</tr>
<tr>
<td>● Own Dockerfile</td>
<td>● Installs limited set of python</td>
</tr>
<tr>
<td>● No need for Airflow sources</td>
<td>dependencies</td>
</tr>
</tbody>
</table>
Usage: Customising Airflow image - default docker build

```
git clone git@github.com:apache/airflow.git
cd airflow
git checkout v1.10-stable
```

Same as apache/airflow:1.10.11
- Python 3.6
- Default extras
- No additional dependencies
Usage: Customising Airflow image - use build args

- Installs from PyPi ==1.10.11
- Additional airflow extras, dev, runtime deps ...
- Does not use local sources (can be run from master including entrypoint!)

```
docker build . \
    --build-arg PYTHON_BASE_IMAGE="python:3.7-slim-buster" \
    --build-arg PYTHON_MAJOR_MINOR_VERSION=3.7 \
    --build-arg AIRFLOW_INSTALL_SOURCES="apache-airflow" \
    --build-arg AIRFLOW_INSTALL_VERSION="==1.10.11" \
    --build-arg CONSTRAINT_REQUIREMENTS=\n    "https://raw.githubusercontent.com/apache/airflow/1.10.11/requirements/requirements-python3.7.txt" \
    --build-arg AIRFLOW_SOURCES_FROM="empty" \
    --build-arg AIRFLOW_SOURCES_TO="/empty" \
    --build-arg ADDITIONAL_AIRFLOW_EXTRAS="jdbc" \
    --build-arg ADDITIONAL_DEV_DEPS="gcc g++" \
    --build-arg ADDITIONAL_RUNTIME_DEPS="default-jre-headless"
```
Usage: Image Customization options

- Choose Base image (python)
- Install Airflow from PyPI
- Install from GitHub branch/tag
- Install additional extras
- Install additional python deps
- Install additional apt dev deps
- Install additional apt runtime deps
- Choose different UID/GID
- Choose different AIRFLOW_HOME
- Choose different HOME dir
- Build Cassandra driver concurrently

See IMAGES.rst in the Airflow repo.
Usage: It’s a Breeze to build images

- Breeze - development and test environment
- Supports building production image
- Auto-complete of options
- New Breeze video showing building production images:
  https://s.apache.org/airflow-breeze
- ./breeze build-image --help

See BREEZE.rst in the Airflow repo
Customising image - Pros & Cons

**Pros**
- Highly optimized for size
- Build image from sources (security reviews!)
- Can add any extras
- Can add any dependency
- Breeze build commands
- Works from master and 1.10.*

**Cons**
- Need access to airflow sources
- Complex build command
- Need to understand internals
Usage

Why not eat and have cake?

git clone git@github.com:apache/airflow.git

cd airflow

git checkout v1-10-stable

./breze build-image --production-image --additional-extras "jira" \
--install-airflow-version "1.10.11"

When dependencies change

FROM base-image-for-your-company:1.10.11-2020-07-14
COPY --chown airflow:root dags-folder "$AIRFLOW_HOME/dags"

When DAGs change

base-image-for-your-company:1.10.11-2020-07-14

Runtime Container image

Base Container image
Usage

How to deploy the images?

- Docker and Docker-Compose - not recommended for production
- Managed Container Services
  - Managed: Amazon ECS, Google Container on VMs, Azure Container Instances
- Kubernetes on-Prem:
  - Helm Chart
  - Airflow Operator (not recommended yet)
- Managed Kubernetes: Amazon EKS, Google GKE, Azure AKS
- OpenShift (also Kubernetes)
Future

What is the future for Airflow images?

- It won’t change too much!
- Better automated testing via Helm Chart
- Automated releases for 2.0
- ARM support might be the big one. (Apple Mac OS)
- Official Docker Compose
- Smaller features (depends on feedback and expectations):
  - ON BUILD support?
  - AIRFLOW__CORE__SQLALCHEMY_CONN_CMD, AIRFLOW__CELERY__BROKER_URL_CMD support?
  - Automated user creation?