Seed User Guide

Emily Smith, Jonathan Meyer, Matt Anderson

Table of Contents

1. Purpose

Scale and Seed optimize algorithm development from proof of concept to operational execution within distributed processing clusters. This guide introduces data scientists and algorithm developers to these technologies developed at the Research directorate of the National Geospatial-Intelligence Agency (NGA).

Scale is a distributed processing framework for near real-time processing of large data sets. It frees data scientists and algorithm developers from the complexity of distributed system concepts and offers an integration platform for highly parallel processing of data. Scale also ensures decision makers and system administrators are able to visually monitor data flows and ensure reliable data processing. It enables a true DevOps environment for algorithm development and cloud transition.

Seed is a general standard to aid in the discovery and consumption of a discrete unit of work contained within a Docker image. It is a JSON-defined manifest that is embedded into the Docker image as a label. Seed allows data scientists to focus their expertise on solving domain problems and provides tools to ensure their algorithm is packaged properly for operational use. Seed can be used throughout the development lifecycle to build, test and package within a simulated environment that closely emulates targeted operational environments. This reduces risk and allows the algorithm requirements to be explicitly defined by the developers who best understand their processing needs. Compliance with the Seed specification will be a requirement for algorithms to run in future versions of Scale.

SILO (Seed Images Location Operation) is a REST API provided by the Seed team for discovering Seed images. The API allows users to scan one or more repositories for Seed images and then search the resulting images and their manifests by keywords. Frameworks such as Scale can use these endpoints to assist users in finding algorithm images and creating jobs out of them.

The Seed command-line interface (CLI) is a standalone tool for working with Seed jobs locally. This document will focus primarily on using the Seed CLI to build, test and publish your algorithm as a Seed-compliant Docker image. The Seed toolchain consists of the CLI, SILO, and a processing system that can consume Seed images, such as Scale. Our recommendation is to familiarize yourself with using Seed by working through this tutorial. Details on more advanced topics are provided in the Reference section.

> Commands that can be run from the terminal are indicated in monospace font or as callouts; e.g. `mkdir test-dir && cd test-dir` or
>
> ```
> mkdir test-dir && cd test-dir
> ```

2. Expectations

2.1. Docker and Linux Fundamentals

Before using Seed, you should have a working familiarity with developing code using modern tools; at minimum knowledge of Docker and Linux fundamentals. This guide is written with the assumption that you are using a Linux workstation. The Seed CLI is heavily reliant on Docker to build and package Seed images (Docker image with Seed metadata), so it is highly recommended that you have superuser (i.e. `sudo`) access.

2.2. Algorithm Requirements

It is beneficial to identify and understand the following characteristics of each algorithm you intend to package with Seed:

- Software requirements (e.g. libraries, frameworks, and/or software dependencies needed)
- Hardware requirements (e.g. CPU and memory requirements, GPUs, etc.)
- Reference material or other support data
- Licenses required

3. Docker Installation

Refer to the Docker documentation for instructions on how to install Docker for your platform. Depending on your IT environment, you may need to request an administrator install Docker for you. The current version of Seed (1.2.1) requires Docker version 18.09 or newer. To verify that installation was successful and confirm the Docker version, run the command `docker --version` from the terminal.

> Depending on your environment, you may need superuser privileges to execute the `docker` command; e.g. `sudo docker`.

Output should be similar to the following:

```
docker version 19.03.8, build afacb8b
```

If this command fails, verify the Docker daemon is running using the `docker info` command. You can also use operating system utilities, such as `sudo systemctl is-active docker`, `sudo status docker` or `sudo service docker status`, or checking the service status using Windows utilities.

4. Seed CLI Installation

The Seed CLI can be installed by downloading a binary from the releases page on GitHub. Under each release, there is an "Assets" section with downloadable binaries for various operating systems. Download the appropriate version for your system and rename the downloaded binary file `seed` to match the usage in the rest of this guide. Move the binary to your desired location on your filesystem:

```
mkdir /path/to/seed-dir  # e.g. mkdir /opt/seed
mv ./seed-darwin-amd64 /path/to/seed-dir/seed  # e.g. mv ./seed-darwin-amd64
/opt/seed/seed
```

In order to run the `seed` command without needing to specify the full path to the binary, add it to your `$PATH` variable. The method for doing this will depend on the shell you are using but typically involves adding an entry to a config file. Common config files include `~/.bash_profile`, `~/.bashrc`, `~/.profile`, or `~/.tcsh_profile`. Using `~/.bash_profile` as an example, if it does not exist, create it using the following command:

```
touch ~/.bash_profile
```

Open it in your desired text editor, and add the following line:

```
export PATH=$PATH:/path/to/seed-dir  # e.g. export PATH=$PATH:/opt/seed
```

Run `source ~/.bash_profile` to reload the profile. To verify that installation was successful, run the command `seed` from the terminal. The Seed logo, usage instructions, and a list of commands should be displayed:

```
       .',,;;;;;,,''''.
      .::::::::::::;,,'.
    ,c:;;;;;;::. ....
., .c;,;;::::. .,,,,,'.
'c;,;,;:::::. .,'''''',.      ';;;;;.   .;;;;;;;;.   ;;;;;;;,   ,;;;;;,.
;:::,;::::;.  ,''''''''     ::.   .    'c,        .::         ;c.   ;c'
,:;;,,'..    .,''''''','     ;:,.       'c'        .:;         ;:.   .:,
 ,:::,.     .,'''''''',.      .,;::,    ':;,,,     .::;,;.     ;:.   .c,
   ....''..',''''''''',.          '::   'c'        .:;         ;:.   .:,
         .,'''''''',''.      .;.   .c;   'c,        .c:.        ;c.  .;c.
         ,,,,,,,,,'.         ';;;;;'    .;;;;;;;;.   ;;;;;;;,   ';;;;;,.
         ..'''...

Usage:  seed COMMAND

A tool for assisting in creating seed spec compliant algorithms

Commands:
  build     Builds Seed compliant Docker image
  batch     Executes Seed compliant docker image over multiple iterations
  init      Initialize new project with example seed.manifest.json file
  list      Lists all Seed compliant images residing on the local system
  publish   Publishes Seed compliant images to remote Docker registry
  pull      Pulls images from remote Docker registry
  run       Executes Seed compliant Docker image
  search    Allows for discovery of Seed compliant images hosted within a Docker
registry (default is docker.io)
  spec      Displays the specification for the current Seed version
  unpublish Removes images from remote Docker registry
  validate  Validates a Seed spec
  version   Prints the version of Seed spec

Run 'seed COMMAND --help' for more information on a command.
```

5. Seed Tutorial

Following this step-by-step tutorial will quickly get you up and running with Seed and demonstrate the steps needed to arrive at a complete Seed image. By the end of this guide, you will be able to:

- Build a Seed compliant algorithm
- Leverage the Seed CLI to ensure Seed compliance and build a Seed compliant Docker image
- Leverage the Seed CLI to test your Seed algorithm
- Leverage the Seed CLI to publish your Seed algorithm
- Understand next steps to automated distributed execution

5.1. Seed Compliance

For your algorithm to be compatible with Seed, it must satisfy the following criteria:

- **Run on Linux.** There is no language limitation other than it must be able to execute on Linux. You can use the Docker base image of your choice. Alpine and CentOS are the preferred Linux flavors.
- **Command-line Invocation.** Seed provides input via either arguments or environment variables. If your algorithm is prompting for input from a user, the job will continue to wait until it times out. There are no display popups such as error dialogs, file selection menus, splash screens, etc. In the event where a display device is required for rendering data, a pseudo device must be used.
- **Configurable.** Your algorithm will be run in a standalone container, therefore absolute file paths must not be embedded in the source code for your development environment. Necessary file paths should be passable into the algorithm either via an environment variable or from the command line.
- **Reporting.** While this isn't required, it is ideal if your algorithm outputs its progress and errors to the console and returns an appropriate exit code. Unique exit codes should be used for failures that can be anticipated. If failures are not captured appropriately, Seed will only be able to identify a general algorithm error, which may make debugging issues more difficult.

5.2. Sample Algorithm

The foundation of a Seed image is the algorithm that it contains. Everything that follows is informed by the requirements of your unique algorithm: the inputs it requires, the outputs it generates and the resources that are required to perform the computations. For this guide, we are going to use a very simple algorithm, one which takes a single image file as input and rotates the image a specified number of degrees. We are going to output metadata about the algorithm both to the console and to a file. This example illustrates how to accomplish the following:

- Accept a file input
- Accept an integer type input
- Write to the console
- Write to an output file

5.2.1. Running in Python

This example assumes Python 3 is installed and uses the Pillow library for image processing. For help installing Python, see the Python Beginners Guide. For help installing Pillow, see its installation instructions.

> Using tools such as pipenv or venv will help simplify environment and dependency management for Python programs.

We are going to write our algorithm using basic Linux commands. Use your favorite text editor or IDE to create an `image_rotate.py` file:

```
import json
import os
import shutil
import sys
from PIL import Image

def rotate_image(source_image_path, rotate_degrees, output_dir):
    """Rotates an image a specified number of degrees"""
    product_image_name =
f'ROTATED_{rotate_degrees}_{os.path.basename(source_image_path)}'

    # Open source image
    original_image = Image.open(source_image_path)

    # Rotate it specified number of degrees
    rotated_image = original_image.rotate(int(rotate_degrees), 0, 1)
    rotated_image.save(product_image_name)

    # Move saved image to output directory
    product_image_path = os.path.join(output_dir, product_image_name)
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    shutil.move(product_image_name, product_image_path)

    return product_image_name

if __name__ == "__main__":

    source_image_path = sys.argv[1]
    rotate_degrees = sys.argv[2]
    output_dir = sys.argv[3]

    product_image_path = rotate_image(source_image_path, rotate_degrees, output_dir)
```

Run the algorithm using the following command:

```
python image_rotate.py seed.png 180 output
```

A new image file named `ROTATED_180_seed.png` will be created in the `output` directory.

5.2.1.1. Python Troubleshooting

If the following `ModuleNotFound` error occurs, it likely means the Pillow library is not installed correctly. Please refer to the Pillow installation instructions.

```
Traceback (most recent call last):
  File "image_rotate.py", line 5, in <module>
    from PIL import Image
ModuleNotFoundError: No module named 'PIL'
```

If the following `SyntaxError` occurs, it likely means you are using Python 2 instead of Python 3. Please refer to these instructions for installing Python.

```
File "image_rotate.py", line 9
  product_image_name =
    f'ROTATED_{rotate_degrees}_{os.path.basename(source_image_path)}'
                                                                    ^
SyntaxError: invalid syntax
```

5.2.2. Running in Docker

First, create a script called `image_rotate.sh` in the same directory as the Python file created above:

```
#!/bin/sh

INPUT=$1
DEGREES=$2
OUTPUT_DIR=$3

echo ''
echo '-----------------------------------------------------------------'
echo 'Rotating image with arguments '${INPUT} ${DEGREES} ${OUTPUT_DIR}

python ./image_rotate.py $INPUT $DEGREES $OUTPUT_DIR
rc=$?

echo 'Done rotating image'
echo '-----------------------------------------------------------------'
echo ''
exit ${rc}
```

We will use this script to invoke our image rotation algorithm.

Create a file named `Dockerfile` in the same directory:

```
FROM python:3.8-alpine

RUN apk add build-base python3-dev py-pip jpeg-dev zlib-dev
ENV LIBRARY_PATH=/lib:/usr/lib

RUN pip install --upgrade pip && \
    pip install --upgrade Pillow >= 7

WORKDIR /algo

ADD ./image_rotate.py .
ADD ./image_rotate.sh .
```

With these files, we can create our initial Docker containerized sample algorithm. Issue the following terminal commands to build the Docker image:

```
sudo docker build -t img-test .
```

The image may take a minute or two to build. The `-t` flag is used to specify the name and optional tag for the Docker image in the 'name:tag' format. In this case, we have used the name `img-test`, but we could use any name. Once it completes successfully, you should see output similar to the following:

```
Successfully built 84dd346586d1
Successfully tagged img-test:latest
```

Now run the Docker container using the following command:

```
sudo docker run --rm -v ${PWD}:/algo img-test  seed.png 45 output
```

Output should be similar to the following:

```
-----------------------------------------------------------------
Rotating image with arguments seed.png 45 output
Done rotating image
-----------------------------------------------------------------
```

Let's recap what we've done:

1. We wrote a simple Python algorithm that consumes three positional parameters: input file path, degrees of rotation, and output file directory.
2. Our algorithm uses the Pillow library to rotate the image a specified number of degrees.
3. The algorithm saves the rotated image as a new file in the specified output directory.
4. We wrote a simple shell script to launch our algorithm.
5. We wrote a basic Dockerfile that identified a base image, installed dependencies, and copied our

algorithm and launch script into it.

6. We built a Docker image of our own and called it `img-test`.
7. Finally, we ran a Docker container from our `img-test` image, mounted a volume, and passed it the command to run our algorithm with the required positional parameters.

There are some observations we should make about what we just accomplished.

1. We made our data accessible to the container in our `docker run` command by mounting a volume with `-v ${PWD}:/algo`. This maps our current working directory to the path `/algo` inside the container.
2. We used the `WORKDIR` command in our Dockerfile to set the working directory to `/algo`.
3. We prefixed our call to the script with `sh` so that we did not have to set the execute bit. Alternatively, we could set the execute bit in the Dockerfile with `RUN chmod +x ./image_rotate.sh`.
4. We used our `rotate_image.sh` script to call our Python code. This is a common pattern when writing Dockerfiles.
5. Because we mounted our current directory as a volume, our specified output directory `output` is created in our current directory, and the output of the algorithm is placed in this directory.

> Further discussion of volume mounts and getting data into a container can be found in the Reference Data section.

With the `img-test` Docker image created, we could share this with other people on our local machine. We could also tag it and push it to a remote registry (hub.docker.com, quay.io, etc.) and others would be able to run it. For a basic algorithm example, this is fairly simple, but what if we have a more complicated algorithm with specific resource requirements? What if our algorithm requires large supporting reference datasets? What if we need to leverage runtime licenses that must be carefully protected? What if we want all of these requirements to be explicitly documented and transparent to the consumers of your algorithm? Seed helps solve these problems.

5.3. Initialization

Building on our example algorithm, let's define the Seed manifest. A Seed manifest is a JSON document that defines an algorithm's purpose, who created it, the interface the algorithm provides, and its resource requirements. When you are building a Seed image, standard practice is to put the `seed.manifest.json` file in the same directory as the project's `Dockerfile`. To simplify the initial construction of this file you can use the `seed init` command from the same directory as the algorithm code:

```
Created Seed file: /Users/user/code/seed/guide/example/seed.manifest.json
```

The newly-created `seed.manifest.json` file includes all common sections of the manifest and can be revised to reflect your specific algorithm.

> The full Seed specification can be found at https://ngageoint.github.io/seed/seed.html.

sys::[ls -l *.txt]

Let's start by updating the manifest for our sample algorithm to match the following:

```
{
  "seedVersion": "1.0.0",
  "job": {
    "name": "image-rotate",
    "jobVersion": "1.0.0",
    "packageVersion": "1.0.0",
    "title": "Image Rotate",
    "description": "Rotates an image a specified number of degrees",
    "tags": [
      "jpg",
      "png",
      "image processing"
    ],
    "maintainer": {
      "name": "Matt Anderson",
      "organization": "AIS",
      "email": "matt.anderson@appliedis.com"
    },
    "timeout": 3600,
    "interface": {
      "command": "sh image_rotate.sh ${INPUT_FILE} ${DEGREES} ${OUTPUT_DIR}",
      "inputs": {
        "files": [
          {
            "name": "INPUT_FILE",
            "required": true,
            "mediaTypes": [
              "image/jpeg", "image/jpg", "image/png"
            ]
          }
        ],
        "json": [
          {
            "name": "DEGREES",
            "type": "integer",
            "required": true
          }
        ]
      },
      "outputs": {
        "files": [
          {
            "name": "ROTATED_IMAGE",
            "multiple": false,
            "pattern": "ROTATED_*"
          }
        ]
      }
    },
    "resources": {
      "scalar": [
        {
```

```
          "name": "cpus",
          "value": 0.5
        },
        {
          "name": "mem",
          "value": 512
        }
      ]
    }
  }
}
```

> In this example, we have named our output file `output.txt`. For an actual algorithm, we recommend using a more unique name or having the algorithm generate the name of the output file to help distinguish it from other outputs.

There are a number of specific settings we made here that are worth highlighting:

1. `job.interface.command`. This setting defines exactly what command is issued on container launch. It mirrors the Docker command we ran in the previous section. The primary difference here is the use of environment variables.
2. `job.interface.command` environment variables. These variable names correspond to the `name` values within the `job.interface.inputs` and `job.interface.outputs` objects. If an `ENTRYPOINT` is specified in the Dockerfile, the contents of this setting will be passed to it.
3. `${INPUT_FILE}`. The Seed specification contract ensures that this variable will be populated with an absolute path to the input since we have marked it as a required input.
4. `${DEGREES}`. The Seed specification contract ensures that this variable will be populated with an integer value to the input since we have given it the explicit type "integer" and marked it as a required input.
5. `${OUTPUT_DIR}`. Seed provides some contextual values that ensure there are consistent locations for output capture. The `OUTPUT_DIR` environment variable is provided to all jobs and any file products must be placed under this location.
6. `OUTPUT_FILE` pattern expression. The expression is rooted at `OUTPUT_DIR` and all patterns defined are relative to that location. This is why we tell our job to write to `${OUTPUT_DIR}/output.txt` and our `pattern` is defined as `*.txt`.
7. `job.resources.scalar`. We provided a fractional CPU requirement and an amount of memory. We can use the `inputMultiplier` setting to inform Seed to allocate memory (MiB) in proportion to the total size of input files (MiB). In other words, if our `INPUT_FILE` is 25 MiB the allocated memory will be: 256 MiB + (4.0 * 25 MiB) = 356 MiB.

> A significant advantage of using Seed CLI is that it can emulate the resource constraints that will be placed on your algorithm in a production cluster environment.

5.4. Validate & Build

Validation and injection of the Seed manifest into the final package ensures adherence to the specification. The CLI allows you to validate a standalone manifest file, as well as apply validation as part of the build process. Let's build our job to see this in action:

```
seed build
```

The `seed build` command will automatically detect and use a Dockerfile and Seed manifest file in the current directory. Alternatively, the `-D` or `-dockerfile` and `-M` or `-manifest` options can be used to specify a path to a Dockerfile and/or manifest file.

```
INFO: Validating seed manifest file
/Users/user/code/seed/guide/example/seed.manifest.json against schema...
INFO: Checking for variable name collisions...
WARNING: /Users/user/code/seed/guide/example/seed.manifest.json does not specify some
recommended resources
Specifying cpu, memory and disk requirements are highly recommended
The following resources are not defined: [disk]
SUCCESS: No errors found. /Users/user/code/seed/guide/example/seed.manifest.json is
valid.
```

The first step of the build is to apply validation. We can see the file that is being validated against the schema. We are also informed that our resources section does not contain all the recommended resource objects. Our build completed successfully. We can see the `com.ngageoint.seed.manifest` LABEL that contains our serialized manifest in Step 3/3 of the Docker build process, as well as the `seed run` command in the final line of the output:

```
INFO: Building image-rotate-1.0.0-seed:1.0.0
dockerfile: .
INFO: Running Docker command:
docker build -t image-rotate-1.0.0-seed:1.0.0 /Users/matt/code/seed/guide/example
--label
com.ngageoint.seed.manifest="{\"seedVersion\":\"1.0.0\",\"job\":{\"name\":\"image-
rotate\",\"jobVersion\":\"1.0.0\",\"packageVersion\":\"1.0.0\",\"title\":\"Image
Rotate\",\"description\":\"Rotates an image a specified number of
degrees\",\"tags\":[\"jpg\",\"png\",\"image
processing\"],\"maintainer\":{\"name\":\"Matt
Anderson\",\"organization\":\"AIS\",\"email\":\"matt.anderson@appliedis.com\"},\"timeo
ut\":3600,\"interface\":{\"command\":\"sh image_rotate.sh \${INPUT_FILE} \${DEGREES}
\${OUTPUT_DIR}\",\"inputs\":{\"files\":[{\"name\":\"INPUT_FILE\",\"required\":true,\"m
ediaTypes\":[\"image\/jpeg\",\"image\/jpg\",\"image\/png\"]}],\"json\":[{\"name\":\"DE
GREES\",\"type\":\"integer\",\"required\":true}]},\"outputs\":{\"files\":[{\"name\":\"
ROTATED_IMAGE\",\"multiple\":false,\"pattern\":\"ROTATED_*\"}]}},\"resources\":{\"scal
ar\":[{\"name\":\"cpus\",\"value\":0.1},{\"name\":\"mem\",\"value\":256,\"inputMultipl
ier\":4.0},{\"name\":\"disk\",\"value\":128}]}}}"
Sending build context to Docker daemon  18.23MB
Step 1/8 : FROM python:3.8-alpine
 ---> db0e2316082c
Step 2/8 : RUN apk add build-base python-dev py-pip jpeg-dev zlib-dev
 ---> Using cache
 ---> 427251dce74d
Step 3/8 : ENV LIBRARY_PATH=/lib:/usr/lib
```

```
 ---> Using cache
 ---> 61111d74877b
Step 4/8 : RUN pip install --upgrade pip &&     pip install --upgrade Pillow >= 7
 ---> Using cache
 ---> af2fcca4e098
Step 5/8 : WORKDIR /algo
 ---> Using cache
 ---> 40fd0aa70d58
Step 6/8 : ADD ./image_rotate.py .
 ---> Using cache
 ---> fbdbd31a55e0
Step 7/8 : ADD ./image_rotate.sh .
 ---> Using cache
 ---> 5bbd749f6360
Step 8/8 : LABEL
com.ngageoint.seed.manifest="{\"seedVersion\":\"1.0.0\",\"job\":{\"name\":\"image-
rotate\",\"jobVersion\":\"1.0.0\",\"packageVersion\":\"1.0.0\",\"title\":\"Image
Rotate\",\"description\":\"Rotates an image a specified number of
degrees\",\"tags\":[\"jpg\",\"png\",\"image
processing\"],\"maintainer\":{\"name\":\"Matt
Anderson\",\"organization\":\"AIS\",\"email\":\"matt.anderson@appliedis.com\"},\"timeo
ut\":3600,\"interface\":{\"command\":\"sh image_rotate.sh \${INPUT_FILE} \${DEGREES}
\${OUTPUT_DIR}\",\"inputs\":{\"files\":[{\"name\":\"INPUT_FILE\",\"required\":true,\"m
ediaTypes\":[\"image\/jpeg\",\"image\/jpg\",\"image\/png\"]}],\"json\":[{\"name\":\"DE
GREES\",\"type\":\"integer\",\"required\":true}]},\"outputs\":{\"files\":[{\"name\":\"
ROTATED_IMAGE\",\"multiple\":false,\"pattern\":\"ROTATED_*\"}]}},\"resources\":{\"scal
ar\":[{\"name\":\"cpus\",\"value\":0.1},{\"name\":\"mem\",\"value\":256,\"inputMultipl
ier\":4.0},{\"name\":\"disk\",\"value\":128}]}}}"
 ---> Running in 6aa9a7b29fa1
Removing intermediate container 6aa9a7b29fa1
 ---> 7c20874e1ba9
Successfully built 7c20874e1ba9
Successfully tagged image-rotate-1.0.0-seed:1.0.0
INFO: Successfully built image. This image can be published with the following
command:
seed publish -in image-rotate-1.0.0-seed:1.0.0 -r my.registry.address
This image can be run with the following command:
seed run -rm -in image-rotate-1.0.0-seed:1.0.0 -i INPUT_FILE=<file> -j DEGREES=<json>
-o <outdir>
```

Let's address the warning regarding disk resource by updating our manifest with a third object in the `job.resources.scalar` array:

```
{
  "seedVersion": "1.0.0",
  "job": {
    ...<omitted for brevity>...
    "resources": {
      "scalar": [
        {
          "name": "cpus",
          "value": 0.1
        },
        {
          "name": "mem",
          "value": 256,
          "inputMultiplier": 4.0
        },
        {
          "name": "disk",
          "value": 16
        }
      ]
    }
  }
}
```

We added a disk requirement of 16 MiB to resolve the warning. This space is only to accommodate any temporary storage needed as part of the job execution. The storage required to write the input files to disk is already accounted for by Seed. Now that our manifest is updated, explicitly perform a validation to ensure the warning is resolved:

```
seed validate
```

```
INFO: Validating seed manifest file
/Users/user/code/seed/guide/example/seed.manifest.json against schema...
INFO: Checking for variable name collisions...
SUCCESS: No errors found. /Users/user/code/seed/guide/example/seed.manifest.json is
valid.
```

With the warnings corrected, let's create a new build:

```
seed build
```

With a complete Seed image now created, we can now run our job using the Seed image. The last line of the console output shows us how we can use the Seed CLI to run our Seed job:

```
seed run -rm -in image-rotate-1.0.0-seed:1.0.0 -i INPUT_FILE=<file> -j DEGREES=<json>
-o <outdir>
```

5.5. Run

The `seed run` command provides the bulk of the functionality within the CLI. This is how we ensure our job is ready to run in an operational environment. By leveraging `seed run` we can be confident that the job we publish behaves consistently with how we defined our interface and requirements in the `seed.manifest.json`. Let's run our example to demonstrate the information the command can provide:

```
$ seed run
INFO: Image name not specified. Attempting to use manifest: .
INFO: Found manifest: /Users/users/code/seed/guide/example/seed.manifest.json
INFO: Retrieving seed manifest from image-rotate-1.0.0-seed:1.0.0
LABEL=com.ngageoint.seed.manifest
normalName: INPUT_FILE
ERROR: Error occurred processing inputs arguments.
ERROR: Incorrect input data files key/values provided. -i arguments should be in the
form:
  seed run -i KEY1=path/to/file1 -i KEY2=path/to/file2 ...
The following input file keys are expected:
  INPUT_FILE

ERROR: Incorrect input data files key/values provided. -i arguments should be in the
form:
  seed run -i KEY1=path/to/file1 -i KEY2=path/to/file2 ...
The following input file keys are expected:
  INPUT_FILE
```

The first three lines show how the `seed run` command inferred the image from our current directory because it contained a Seed manifest, which was then used to find the Seed image built from it. The next two lines beginning with `ERROR` show how the CLI is also able to identify the inputs that are required, but we failed to provide (in this case, `INPUT_FILE`). Let's specify these inputs and try again:

```
$ seed run -i INPUT_FILE=seed.png -j DEGREES=180 -o output
INFO: Image name not specified. Attempting to use manifest: .
INFO: Found manifest: /Users/user/code/seed/guide/example/seed.manifest.json
INFO: Retrieving seed manifest from image-rotate-1.0.0-seed:1.0.0
LABEL=com.ngageoint.seed.manifest
normalName: INPUT_FILE
INFO: /Users/user/code/seed/guide/example/output-image-rotate-1.0.0-seed_1.0.0-2020-
05-19T15_10_22-04_00 not found; creating directory...
INFO: Running Docker command:
docker run -v
/Users/user/code/seed/guide/example/seed.png:/Users/user/code/seed/guide/example/seed.
png -e INPUT_FILE=/Users/user/code/seed/guide/example/seed.png -v
/Users/user/code/seed/guide/example/output-image-rotate-1.0.0-seed_1.0.0-2020-05-
19T15_10_22-04_00:/Users/user/code/seed/guide/example/output-image-rotate-1.0.0-
seed_1.0.0-2020-05-19T15_10_22-04_00 -e
OUTPUT_DIR=/Users/user/code/seed/guide/example/output-image-rotate-1.0.0-seed_1.0.0-
2020-05-19T15_10_22-04_00 -e DEGREES=180 -e ALLOCATED_CPUS=0.100000 -m 257m -e
ALLOCATED_MEM=257 -e ALLOCATED_DISK=128.000000 image-rotate-1.0.0-seed:1.0.0
/Users/user/code/seed/guide/example/seed.png 180
/Users/user/code/seed/guide/example/output-image-rotate-1.0.0-seed_1.0.0-2020-05-
19T15_10_22-04_00

-----------------------------------------------------------------
Rotating image with arguments /Users/user/code/seed/guide/example/seed.png 180
/Users/user/code/seed/guide/example/output-image-rotate-1.0.0-seed_1.0.0-2020-05-
19T15_10_22-04_00
Done rotating image
-----------------------------------------------------------------


INFO: image-rotate-1.0.0-seed:1.0.0 run took 531.131393ms
INFO: Validating output files found under /Users/user/code/seed/guide/example/output-
image-rotate-1.0.0-seed_1.0.0-2020-05-19T15_10_22-04_00...
SUCCESS: 1 files found for output ROTATED_IMAGE:
    /Users/user/code/seed/guide/example/output-image-rotate-1.0.0-seed_1.0.0-2020-05-
19T15_10_22-04_00/ROTATED_180_seed.png
```

A complete run of our job! Let's review what the CLI has accomplished for us:

- A directory for output data was created prior to launch. If no output directory is provided, the CLI creates a timestamped directory to avoid any name collision during subsequent executions. This is mounted into the container from the host when the Docker container is launched.
- `OUTPUT_DIR` environment variable is set on container launch to match the output volume that is being mounted at runtime to capture output.
- The input file `seed.png` we specified is explicitly mounted into the container at runtime. Unlike when we ran our algorithm using `docker run`, we do not have to explicitly mount the input file as a volume—Seed handles that for us.
- `INPUT_FILE` environment variable is set on container launch to inject the absolute file path relative to the container context. This is why we indicate inputs via environment variable syntax in our `job.interface.command` value of the `seed.manifest.json`.

- `DEGREES` environment variable is set on container launch to inject the value we specify for our JSON input type.
- Complete `docker run` statement is output to help identify the exact invocation command Seed CLI uses to launch THE Seed image. You can see the resource requirements identified as environment variables as well. Typically resource requirements are not needed, but JVM applications may benefit from explict understanding of their memory constraints.
- Following the output of our job, we can see the CLI validated that an output file was written in a location that matches the pattern we defined under `job.interface.outputs.files`.

Now that we understand the basics of running and testing our job, we can use more advanced features of the `seed run` command to further validate performance or exercise it against different test input datasets. See the Use Cases section for more examples.

5.6. Publish

After testing our job, we will typically want to share it so that it can used by others. Seed supports several registry backends commonly used in the Docker ecosystem. Docker Hub is a managed registry that makes it easy to publish your Seed image without configuring any additional services of your own. Before you can publish, you will need to register for an account at https://hub.docker.com.

> Executing `docker login` before running `seed publish` eliminates the need to specify a username/password in the `seed publish` statement if you are publishing to your own registry.

The following command will publish our Seed image that we built and tested to Docker Hub:

```
$ seed publish -r index.docker.io -O dockerhub-username -u dockerhub-username -p
"dockerhub-password"
INFO: Image name not specified. Attempting to use manifest: .
INFO: Found manifest: /Users/user/code/seed/guide/example/seed.manifest.json
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Docker login warning: WARNING! Using --password via the CLI is insecure. Use
--password-stdin.

Login Succeeded
INFO: Tagging image image-rotate-1.0.0-seed:1.0.0 as index.docker.io/dockerhub-
username/image-rotate-1.0.0-seed:1.0.0
INFO: Running Docker command:
docker tag image-rotate-1.0.0-seed:1.0.0 index.docker.io/dockerhub-username/image-
rotate-1.0.0-seed:1.0.0
INFO: Performing docker push index.docker.io/dockerhub-username/image-rotate-1.0.0-
seed:1.0.0
INFO: Running Docker command:
docker push index.docker.io/dockerhub-username/image-rotate-1.0.0-seed:1.0.0
The push refers to repository [docker.io/dockerhub-username/image-rotate-1.0.0-seed]
156f2dc847e9: Pushed
a6d503001157: Mounted from library/busybox
1.0.0: digest: sha256:fc29e2201b87bc32a94c58afcebba45115bf930788c8d1b95f871a4de9356396
size: 734
INFO: Removing local image index.docker.io/dockerhub-username/image-rotate-1.0.0-
seed:1.0.0
INFO: Running Docker command:
docker rmi index.docker.io/dockerhub-username/image-rotate-1.0.0-seed:1.0.0
Untagged: dockerhub-username/image-rotate-1.0.0-seed:1.0.0
Untagged: dockerhub-username/image-rotate-1.0.0-
seed@sha256:fc29e2201b87bc32a94c58afcebba45115bf930788c8d1b95f871a4de9356396
```

As can be seen from the output, we are internally performing a series of operations to publish the image. We attach an appropriate tag to the Docker image to comply with the specification that reflects the remote registry `index.docker.io` and organization `dockerhub-username`. This is followed by a push of the image to the repository and cleanup of the remote tags. This leaves our local environment with only the image names we built for our use, which can be verified by running the `docker images` command.

> In the example above, we are pushing to our personal Docker hub account, where the organization is the same as the username. For other use cases, the organization may differ from the username; for example, where organization is `my-company` and username is `my-username`.

6. Reference

6.1. Use Cases

6.1.1. Reference Data

If a small amount of reference data (e.g. less than 100 MB) is needed for a Seed job, it can be included by adding the file to the Docker image. For larger amounts of reference data, avoid excessively increasing the size of the Docker image by passing the reference data to the job as a mountable directory. For more details about mounting directories, see the Docker documentation about bind mounts and using volumes.

6.1.2. Secret Data

Although it is generally preferable to use the Inputs object to pass in data, the Settings object can be used to specify sensitive information for the job, such as a database password.

> ⚠ Sensitive information, such as passwords, private keys, or other credentials should never been included as files or code in the Seed job. Always use the Settings object to inject these values.

Update the `job.interface` object in our manifest from the [Tutorial] with a `settings` object:

```
{
  "seedVersion": "1.0.0",
  "job": {
    ...<omitted for brevity>...
    "interface": {
      ...<omitted for brevity>...
      "settings": [
        {
          "name": "MY_SECRET_SETTING",
          "secret": true
        }
      ]
    }
  }
}
```

Re-build the Seed job using the `seed build` command. `MY_SECRET_SETTING` will be injected into the job as an environment variable at runtime. It can be specified as a environment variable when executing the job using `seed run`:

```
seed run -i INPUT_FILE=seed.png -j DEGREES=180 -e MY_SECRET_SETTING=<secret-value>
```

If the environment variable is ommitted from the `seed run` command, the Seed CLI will output an error:

```
$ seed run -i INPUT_FILE=seed.PNG  -j DEGREES=180
INFO: Image name not specified. Attempting to use manifest: .
INFO: Found manifest: /Users/user/code/seed/guide/example/seed.manifest.json
INFO: Retrieving seed manifest from image-rotate-1.0.0-seed:1.0.0
LABEL=com.ngageoint.seed.manifest
normalName: INPUT_FILE
INFO: /Users/user/code/seed/guide/example/output-image-rotate-1.0.0-seed_1.0.0-2020-
04-21T14_37_21-04_00 not found; creating directory...
ERROR: Error occurred processing settings arguments.
ERROR: Incorrect setting key/values provided. -e arguments should be in the form:
  seed run -e SETTING=somevalue ...
The following settings are expected:
  MY_SECRET_SETTING

ERROR: Incorrect setting key/values provided. -e arguments should be in the form:
  seed run -e SETTING=somevalue ...
The following settings are expected:
  MY_SECRET_SETTING
```

When running the Seed job on a Scale cluster, secret names and values are input as part of the JobType definition in Scale. Scale then stores the secrets in a {hashicorp-vault-url[HashiCorp Vault] secret store where they are only accessible by running jobs. The system will automatically retrieve any secrets defined in the Seed job and inject them at runtime. You can also contact the Scale team for assistance with adding JobTypes that contain secrets.

6.2. Optimization

6.2.1. Image Optimization

Use of Anaconda and other large libraries associated with data science and machine learning jobs is common. This makes sense for development on Windows or other environments that make compilation of complex scientific and math libraries challenging but should not be used in a Seed job. As a rule of thumb, in order to avoid excessive strain on Docker registry and image cache resources, Docker images in Seed jobs should be kept under 500 MB. Docker images should be based on minimal operating systems, such as Alpine or Busybox, and layer optimization techniques should be applied. Docker recommends a number of best practices for writing Dockerfiles.

6.2.2. Image Clean-up

Another strategy for optimizing the size of the Docker image associated with the Seed job is to add commands to clean up after any package installations. Consult the documentation for any package management system used by the operating system (e.g `apk` or `yum`) and/or language (e.g. `pip` or `mpm`), as the exact clean-up commands will vary depending on the environment and tools being used.

6.3. Best Practices

6.3.1. Workflow Decomposition

Breaking down larger processing chains into separate jobs will yield benefits throughout the algorithm development lifecycle. Problems can be discovered sooner and component algorithms can be built and tested independently, helping to minimize any single person or team being a bottleneck. The greatest benefits can be realized when extracting general purpose, reusable jobs that can be applied to multiple data types, such as a job for generating a tile pyramid within a GeoTIFF. This also helps ensure that commonly used jobs are both optimized and thoroughly proven by reuse. Of course, there are always exceptions, and some algorithms (e.g. those that are I/O bound) may benefit from combining several processing steps into a single job.

6.3.2. Log Everything

Not having direct access to the file system of your job means the only method for feedback on what is happening inside the container is through console output. Use standard output and/or standard error to indicate any progress or errors for your algorithm. Some languages (such as Python) may require specifying that output should not be buffered until the process exits. This will facilitate live viewing of output for long-running processes.

6.3.3. Meaningful Errors

Specific error conditions represented by unique process exit codes should be used to give clear feedback when a known error has occurred. These can be used to distinguish between data errors (where the data arrived but contained an error that could not be automatically addressed by the job) and errors that occurred within the job execution itself. See also Why should I define error codes? in the FAQ section.

6.3.4. Runtime Configuration

Most jobs have configuration values that are desirable to change at runtime. These could be anything from confidence values for feature detection, to a band number within a multi-band image. Ensure that your job can consume its configuration as either environment variables or command-line arguments. This makes it possible to define your job as a configurable Docker image and avoid requiring the creation of new Docker images for common changes.

6.3.5. Privilege Minimization

Docker images often are set to use the `root` user by default. Avoid this unless absolutely necessary. Instead of using the `root` user, add a non-privileged user to the Docker image and set the USER for the image to this non-privileged user.

6.3.6. Dockerfiles

Docker provides guidance on some additional best practices for writing Dockerfiles.

6.4. Frequently Asked Questions (FAQs)

6.4.1. What is this Seed image?

Seed is a general standard that was developed to aid in the discovery and running of self-contained algorithms. It is a JSON-defined manifest that is embedded into the Docker image as a label. It defines:

- Algorithm - name, version, description
- Developer - name, company, contact info
- Inputs/Outputs - how many, what type
- Hardware Requirements - CPUs, memory, disk
- Error handling - data vs. algorithm errors
- Environment variables
- Mounts

6.4.2. How do I use Docker?

Many resources for learning about Docker can be found on the Docker website. The primary goal of this guide is to explain how to integrate an algorithm with Seed and not focus on how to build a Docker container.

6.4.3. Why not just add all the files in my directory to the Docker image?

The image should be as small as possibly and only include the necessary files. It is pulled across the network each time it runs on a node, so smaller images result in less network traffic.

6.4.4. Why not just increase the hardware resources as much as possible? I want it to run fast.

It is critical to strike a balance between the minimum resources required to run the algorithm and acceptable runtimes. Scale depends on accurate requirements in order to efficiently schedule jobs on the cluster. If a job requests more resources than it uses, these resources are wasted as they cannot not be allocated to other jobs. In a cloud environment, wasted resources equate to wasted money. When job requirements depend on the input, the Seed manifest supports specifying a multiplicative value for the resource requests via the `inputMultiplier` property.

6.4.5. My algorithm requires some reference data, should I just embed it into the Docker image?

It depends. If the total size of the file(s) is sufficiently small (e.g. less than 100 MB), add it into the Docker image. If it has larger requirements, such as a lookup directory, please work with the Scale team to create a mountable directory.

6.4.6. How to secure sensitive information like passwords inside my Docker image?

The Seed manifest allows the developer to define "Settings", which will be defined as environment variables at runtime. These "Settings" can be marked as "secret" and will be stored in HashiCorp Vault when running the job in a processing system that consumes Seed images, such as Scale. The values for these environment variables are hidden and only injected at runtime. See Secret Data for more details.

6.4.7. Why should I define error codes?

The developer should properly capture and define errors in their code using exit codes. The Seed manifest allows the developer to define specific error codes with name, description, and type (job vs. data). A data error indicates that the input data was invalid and the job **will never** be able to run it. All other errors are job errors and could potentially be resolved on future executions (re-queues/re-process). A third type, "System" errors, are reserved for the Scale framework.

6.4.8. How do I use exit codes in Python/Matlab/Java/etc.?

Code should use proper try/catch or if/else statements to identify the type of error. The code can exit with any value between 0-255. **The zero value indicates successful run.** The exit code can be checked after a program finishes with "echo $?".

Common examples exiting with code 42:

Python: `sys.exit(42)`

Matlab: `exit(42)`

Java: `System.exit(42)`

C/C++: `exit(42)`

Shell: `exit 42`

6.4.9. My algorithm involves running Python code, followed by Matlab code, and finally C code. Should I put it all in one Docker image?

It depends. Does each step involve heavy processing or create time-consuming intermediate products? If so, they would be better as their own standalone jobs with its output products being passed to a subsequent step in a workflow. If the individual steps are relatively small (in runtime and resources), it can all run within a single container using a shell script or similar to execute each step.