
LeadingAgile Code Analysis Gather CLI Docker Image

- 1. Using the Gather CLI Container
  - 1.1. Pull the Image
  - 1.2. Using Gather CLI
    - 1.2.1. Configuration File
      - 1.2.1.1. `statistics` configuration
      - 1.2.1.2. `answers` configuration
      - 1.2.1.3. Configuration File Example
- 2. Gather CLI as an Example
  - 2.1. Create a Dockerfile
  - 2.2. Create Entrypoint to Do the Work
  - 2.3. Build the Image
  - 2.4. Details

1. Using the Gather CLI Container

1.1. Pull the Image

To begin to use the Gather CLI image, you must pull it from the LeadingAgile Studios Docker Registry.

```
docker pull
    leadingagilestudios.azurecr.io/analysis/gather-cli:<version-number>
```

Adjust the version tag to match the current version of the Gather CLI image.

You must be authorized to access the LeadingAgile Studios Docker Registry, logged in to the LeadingAgile Azure instance, and logged in to the LeadingAgile Studios Docker Registry.

Using the Azure CLI, the steps to authenticate for pulling the image are:

```
az login
az acr login --name leadingagilestudios
```

1.2. Using Gather CLI

Use `docker run` to execute the Gather CLI. The options up to the image name are processed by Docker Run. The options after the image name are processed by the Gather CLI tool itself.

```
docker run -it --rm \
    -v ~/folder/with/repos/:/opt/repos/ \
    -v ~/folder/for/output/:/opt/output/ \
    leadingagilestudios.azurecr.io/analysis/gather-cli:<version-number>
        \
    -s 2021-03-31 \
    -r RUN_Q1_2021 \
    -t 52 \
    -c /opt/repos/config.yaml
    -g
```

Gather and Analyze data from a collection of Team repositories in a common folder.

Parameters before the image name are for the docker run command.

Parameters after the image name are passed to the internal processing script in the container.

Option	Description
--start-date	Start Date (for the --start-date option to metrics.gather). Optional, defaults to curren
--run-name	Run Name (for the --run-name option to metrics.gather and metrics.answers team). C
-t, --steps	Weeks (steps) to processed (for the --steps option to metrics.gather). Optional, defaul
--team-config	Configuration YAML File to be used while processing repositories
-g, --disable-graphs	Disable Graph Generation (default is to generate all graphs). Optional, defaults to N
-a, --disable-answers	Disable Answer Generation (default is to generate all answers for repositories and the
-s, --disable-statistics	Disable Statistics Generation (default is to generate all statistics for repositories and t
-m, --disable-metrics	Disable Metrics Generation (default is to gather the basic metrics data on which every
-v	Enable verbose output
--debug	Enable verbose logging
-V, --version	Version information

The Disable options are used to reduce run time (Graphs), regenerate parts of the analysis (Answers and Metrics).

The container will load, process the actions specfied, and the results will be stored in the host folder mounted as /opt/output/.

For the Gather CLI docker the above command will:

- Process all the repositories found in ~/folder/with/repos (which is mounted in the container as /opt/repos/)
- Write the gather data to the repository- and run-folders based at ~/folder/for/output (which is mounted in the container as /opt/output/)
- Use a START_DATE of 2021-03031
- Use a RUN_NAME of RUN_Q1_20201
- USe a config file inside ~/folder/with/repos named config.yaml

- Suppress creation of graphs

Be sure that the folder you mount for output exists BEFORE running the container, else the docker mount will fail and NO DATA will be visible on the host. The Gather CLI will appear to run, but it will be writing to a folder on a layer in the container which will disappear when the container exits.

1.2.1. Configuration File

A configuration file can be included in various portions of the assessment tool to help tailor the results. In particular, two areas are when calculating statistics and answering questions for a team(s). This file is in YAML format.

1.2.1.1. `statistics` configuration When there is a need to generate statistics for a team that shares a repository with other teams (i.e. In a Mono-Repo scenario), the configuration file allows the ability to state just *how* the breakdown of the repository needs to occur. In YAML, under the key of `statistics`, each team that is sharing a repository can be listed along with a list of files and/or directories that the team is responsible for (or not responsible for). Only these files will count toward the statistics generated for that team.

If, for instance, you have team **akyula** working inside the missle-guidance-system repository alongside team **boryei**, where each team handled a set of subdirectories within the repository, the statistics section would look like this:

```
statistics:
  boryei:
    missile-guidance-system:
      includes:
        - guidance
        - targeting
  akyula:
    missile-guidance-system:
      includes:
        - firing-system
        - evasion
      excludes:
        - evasion/targeting
        - "*.js"
```

This example shows how to both include and exclude files and directories when computing statistics for a team.

1.2.1.2. `answers` configuration There are times when generating answers that the assessor will want to run all of the repositories for all of the teams

involved in the Expidition at once and report on the various teams individually. Using the answers section of the configuration file allows for this very scenario.

Simply list each team and their associated repositories and only those repositories will be included when generating the team-level report for how each team is doing when asked the various questions the tool attempts to answer.

Using the same two teams as above, an example of the answers section would look something like:

```
answers:
  boryei:
    - missle-guidance-system
    - sonar
    - ballast-control
  akyula:
    - missle-guidance-system
    - engine-control
    - communications
```

1.2.1.3. Configuration File Example

```
statistics:
  team-name-one:
    repository-main-directory-name:
      includes:
        - path/to/folder
        - another/path/to/folder
        - some/path/to/file.ext
    repository-two-directory-name:
      includes:
        - example/path
        - other/example/path

answers:
  team-name-one:
    - repository-main-directory-name
    - repository-two-directory-name
    - repository-three-directory-name
  team-name-two:
    - repository-main-directory-name
    - repository-two-directory-name
    - repository-four-directory-name
```

2. Gather CLI as an Example

The way the Gather CLI image is built is an example of how to do costume processing with the metrics.gather tool inside a docker image. This section explains the how-to for creating this image.

If you only need the existing Gather CLI images, it is better to pull them from the registry or build them using the build_gather_dockers. sh script.

2.1. Create a Dockerfile

Create a Dockerfile that uses gather as the base image. gather contains everything it needs.

Put this Dockerfile (end the entrypoint described below) in a folder with nothing else in the folder. Anything in the folder becomes available to the image during the build.

```
FROM gather:latest

ADD ./gather-cli_entrypoint.sh /usr/local/bin/gather-cli_entrypoint.sh
RUN chmod a+x /usr/local/bin/gather-cli_entrypoint.sh

ENTRYPOINT ["/usr/local/bin/gather-cli_entrypoint.sh"]
```

In this case the image is referenced as gather:latest. You should pull the current version of gather and tag it as gather:latest or use the full image identifier instead. See the gather docker image documentation for more details.

Dockerfile Reference

2.2. Create Entrypoint to Do the Work

Create an entrypoint script to call Gather and do the work. This entrypoint script will be run when the container is run.

Put this enrypoint script in the folder with the Dcokerfile (described above) in a folder with nothing else in the folder. Anything in the folder becomes available to the image during the build.

The example below is set up so that it assumes:

- There is a folder in the container called /opt/repos/ which has a collection of git repositories to process
- There is a folder in the container called /opt/output/ will be the output destination for the results

Both of these folders are mounted using the docker -v <host_folder>:<container_folder> option.

The helper scripts used here are described the Details section.

```
#!/usr/bin/env bash

source /usr/local/bin/gather_entrypoint_utils.sh

cd /opt/code-analysis || exit 1

prep_ssh_keys_folder
```

```
prep_git_authentication

repos_folder="/opt/repos"
output_folder="/opt/output"

for repo_dir in "${repos_folder}"/*; do
    if [ -d "${repo_dir}" ]; then
        repo_output_folder="${output_folder}/$(basename "${repo_dir}")"

        python -m metrics.gather -r "${repo_dir}" -t 52 -dg -o
            "${repo_output_folder}"

        # answer the questions on the repo for all the languages found
        for stats_file in
            "${repo_output_folder}"/*_data_statistics.json; do
            python -m metrics.answer -s "${stats_file}" -o
                "${repo_output_folder}"
        done    fi
done
```

2.3. Build the Image

Use the build_gather_dockers.sh to build all the docker images. These instructions are for cases where you need to build the Gather CLI Docker separately for some reason

This assumes you have built, or have a pulled from a registry, a gather image.

Tag the gather image as gather:latest because that is what the GatherCli.Dockerfile expects. (if using the build_gather_dockers.sh this is done for you.)

For example:

```
docker tag leadingagilestudios.azurecr.io/analysis/gather:0.1.1
    gather:latest
```

Build the Gather CLI image:

```
docker build --rm -t
    leadingagilestudios.azurecr.io/analysis/gather-cli:0.1.1 -f
    GatherCli.Dockerfile ./
```

This will generate your gather-cli image.

You should be able to see it in the current list of images (yours might be different in the tags and IDs):

```
docker image ls
REPOSITORY                                                    TAG
    IMAGE ID        CREATED         SIZE
leadingagilestudios.azurecr.io/analysis/gather-cli        0.1.1
    6d53cb6c9c6e    4 hours ago    3.89GB
```

```
gather                                                          latest
    5355a215cc2a    4 hours ago    3.89GB
leadingagilestudios.azurecr.io/analysis/gather                  0.1.1
    5355a215cc2a    4 hours ago    3.89GB
leadingagilestudios.azurecr.io/analysis/gather-dev              0.1.1
    a26bf4a4ef35    4 hours ago    4.06GB
swift                                                           5.3-focal
    8d02b9b0bbd0    5 weeks ago    1.7GB
```

2.4. Details

Helper scripts

The gather image includes some helper scripts to ease the developement of the entrypoint script. These helpers are installed into the container in the gather image. Any image derived from gather will be able to use them and not have to re-ADD them.

/usr/local/bin/gather_entrypoint_utils.sh

This script is intended to be sourced into an entrypoint script.

It contains these functions:

---

prep_ssh_keys_folder

Uses the /tmp/.ssh/ mounted folder to initialize the container ssh configuration using keys made available from the host.

- Checks for the existence of a folder /tmp/.ssh/. You need to mount this folder when running the container (see the description above) for it to exist in the container. This function does nothing if you do not mount it in exactly this container location.
- If /tmp/.ssh/ exists, the function:
  - Copies all public and private key files from /tmp/.ssh/ to ~/.ssh/
  - Ensures the ssh-agent is running
  - Registers all private keys with the agent using ssh-add

Public key files are identified as any file that has a line beginning with ssh-.

Private key files are identified as any file that contains -BEGIN.*PRIVATE KEY-.

You can limit the keys the container can use by creating a folder with only the keys needed.

If you do not need to do any ssh based activities, you do not need to using the ssh key folder. If you are only processing repositories in local folders or you never user git protocol to clone/update repositories, you will not need ssh.

---

prep_git_authentication

Uses the environment variables GITHUB_USR and GITHUB_PAT to initialize git/Github authentication so that authentication does not happen on each git access to a protected resource.

- Checks that both GITHUB_USR and GITHUB_PAT environment variables are set with docker run (i.e. −e GITHUB_USR=fred −e GITHUB_PAT=fd12ac)
- If they are set:
  - Sets the git credential timeout to 86400 seconds (24hours) in the git config
  - Authenticates the user with git credential approve

This is what it does, in case you would rather control that in your entrypoint instead:

```
git config --global credential.helper 'cache --timeout=86400'

(echo url=https://github.com; echo username="${GITHUB_USR}"; echo
    password="${GITHUB_PAT}"; echo ) | git credential approve
```

If you only access local repositories (i.e. do not need to clone or fetch from external repositories), you do not need to use the credentials when running the container.