Predictive Routing Deployment and Operations Guide

AI Core Services Monitoring and Logging

5/6/2025

Contents

AI Core Services Monitoring and Logging

AICS uses a number of logs to track the status of the various containers: Tango, Gunicorn worker containers, MongoDB, Minio (in release 9.0.013.01 and higher), and NGINX (in releases prior to 9.0.013.01). This topic provides the following information on monitoring the AI Core Services containers:

- Configure AICS Log Settings
- Access the Logs for AICS
- Using Logs to Troubleshoot Common Errors
- Checking the Logs for AICS Containers in HA Environments

The 200 response code indicates that a service is running normally. If you receive a different response code that indicates an issue, check additional logs.

Configure AICS Log Settings

To configure logging for AICS, set the following:

1. Configure the LOG_LEVEL environment variable.
2. If necessary, change the maximum log size (set to 100m by default).

Access the Logs for AICS

> **Important**
> To access the logs conveniently, you must add your username (PR_USER, by default) to the following Linux group: `sudo usermod -aG systemd-journal pm`. Otherwise, you must use the **sudo** command to see the logs for the various containers. The following steps assume that you are already connected to the shell of the hosts where AICS is deployed. You can issue these commands from any machine that is part of the AICS deployment cluster.

Tango Logs

To access Tango logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=tango -o cat
2019-07-12 12:09:26,755 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:26 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:09:36,986 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:36 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:09:47,217 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:47 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:09:57,446 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:57 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:10:07,678 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:10:07 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:10:17,909 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:10:17 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:10:28,140 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:10:28 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
```

MongoDB Logs

To access MongoDB logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=mongo -o cat
2019-07-12T12:07:58.662+0000 I NETWORK  [conn4] received client metadata from
172.18.0.4:56298 conn4: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:07:58.684+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56300
#5 (3 connections now open)
2019-07-12T12:07:58.695+0000 I NETWORK  [conn5] received client metadata from
172.18.0.4:56300 conn5: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:07:58.697+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56302
#6 (4 connections now open)
2019-07-12T12:07:58.707+0000 I NETWORK  [conn6] received client metadata from
172.18.0.4:56302 conn6: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:10:01.959+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56388
#7 (5 connections now open)
2019-07-12T12:10:01.969+0000 I NETWORK  [conn7] received client metadata from
172.18.0.4:56388 conn7: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:10:01.970+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56390
#8 (6 connections now open)
2019-07-12T12:10:01.991+0000 I NETWORK  [conn8] received client metadata from
172.18.0.4:56390 conn8: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:10:02.488+0000 I NETWORK  [conn7] end connection 172.18.0.4:56388 (5
connections now open)
2019-07-12T12:10:02.488+0000 I NETWORK  [conn8] end connection 172.18.0.4:56390 (4
connections now open)
```

Model Training Workers Logs

- By default, a cluster contains two running model training worker containers. As a result, the container name changes to reflect which container logs are to be reviewed.

To access Model Training Worker logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_model_training_1 -o cat
2019-07-12 12:14:39 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:39 [INFO] [] Using gunicorn keepalive value = [2]
```

```
2019-07-12 12:14:39 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:39 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:39 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:39 [INFO] [] Starting executor - topic model_training, timeout 2, sleeping
timeout 30
2019-07-12 12:14:40,197 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
127.0.0.1 - - [12/Jul/2019 12:14:49] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:14:59] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:20] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:30] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:40] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:50] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:00] "GET /status HTTP/1.1" 200 -

$ journalctl CONTAINER_NAME=workers_model_training_2 -o cat
2019-07-12 12:14:39 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:39 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:39 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:39 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:39 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:39 [INFO] [] Starting executor - topic model_training, timeout 2, sleeping
timeout 30
2019-07-12 12:14:40,476 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
127.0.0.1 - - [12/Jul/2019 12:14:49] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:00] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:20] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:30] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:40] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:50] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:00] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:20] "GET /status HTTP/1.1" 200 -
```

Analysis Workers Logs

To access Analysis Workers logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_analysis_1 -o cat
2019-07-12 12:14:40 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:40 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:40 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:40 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
```

```
not found in /data/ssl
2019-07-12 12:14:40 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:40 [INFO] [] Starting executor - topic analysis, timeout 2, sleeping timeout
30
2019-07-12 12:14:41,140 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 12:14:50] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:00] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:20] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:30] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:41] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:51] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:01] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:11] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:21] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:31] "GET /status HTTP/1.1" 200 -
```

Purging Workers Logs

To access Purging Worker logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_purging_1 -o cat
2019-07-12 12:14:41 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:41 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:41 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:41 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:41 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:41 [INFO] [] Starting executor - topic purging, timeout 2, sleeping timeout
30
2019-07-12 12:14:41,871 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 12:14:51] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:01] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:11] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:21] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:31] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:41] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:51] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:01] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:11] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:21] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:32] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:42] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:52] "GET /status HTTP/1.1" 200 -
```

Dataset Upload Workers Logs

To access Dataset Upload Worker logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_dataset_upload_1 -o cat
2019-07-12 12:14:42 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:42 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:42 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:42 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:42 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:42 [INFO] [] Starting executor - topic dataset_upload, timeout 2, sleeping
timeout 30
2019-07-12 12:14:42,563 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 12:14:52] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:02] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:12] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:22] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:32] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:42] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:52] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:02] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:12] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:22] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:32] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:42] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:52] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:17:02] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:17:12] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:17:23] "GET /status HTTP/1.1" 200 -
```

Example Commands to Locate Logs

To get last 100 lines of the Tango log, run:

```
$ journalctl CONTAINER_NAME=tango -n 100 -o cat
```

To get last 60 minutes of the Tango log, run:

```
$ journalctl CONTAINER_NAME=tango --since="1 hour ago" -o cat
```

To get the last ten hours of the MongoDB log, run:

```
$ journalctl CONTAINER_NAME=mongo --since="10 hour ago" -o cat
```

To tail Tango logs, run:

```
$ journalctl CONTAINER_NAME=tango -f -o cat
```

Common Errors with AICS

This section provides information about how to diagnose and troubleshoot problems when running the AICS application deployed in Docker Containers. This information applies to both HA and single node installations.

Volume

The default permissions on shared volumes are not configurable. If you are working with applications that require permissions different from the shared volume defaults at container runtime, you need to either use non-host-mounted volumes or find a way to make the applications work with the default file permissions.

The only volume that is required to run the AICS application mounted locally on the hosts is **/datadir**. This could be a mount from the network or locally available on the filesystem of the hosts.

If there is an existing mount already on the system with the similar name or there is an existing data in the folder, rename the mount or move the files to a different location within the system. During the installation of the AICS stack, it is assumed that the path exists and there is no existing data on the volume.

Permissions Errors

The current user might not be added to the Docker group or does not have enough permissions to perform Docker related operations.

```
$ docker ps
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/
docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.30/containers/json: dial unix /var/run/
docker.sock: connect: permission denied
```

During installation of the AICS application, ad the configured user (by default, PR_USER) to the `docker` and `log` groups. These permissions are necessary for administration tasks related to Docker and to identify any application-related issues.

If your user can access system logs, you should be able to discover common application errors related to functionality of the AICS stack.

Stopped Containers

There are scenarios in which the containers can fail and move from the running state to the stopped state. To check the status of all the running containers on the hosts, use the following command:

```
$ docker ps
CONTAINER ID        IMAGE                                   COMMAND
CREATED             STATUS                      PORTS               NAMES
a9ed052f99d6        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   27
minutes ago      Up 27 minutes (healthy)                         workers_dataset_upload_1
3656dae9e4cf        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   27
minutes ago      Up 27 minutes (healthy)                         workers_purging_1
de9640523ad4        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                         workers_analysis_1
9f461a9dbe0f        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   28
```

```
minutes ago      Up 28 minutes (healthy)                          workers_model_training_2
a380bc06279b        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                          workers_model_training_1
871e6f2f1f7f        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)   0.0.0.0:443->3031/tcp  tango
72cd51047081        minio/minio:RELEASE.2018-07-23T18-34-49Z   "sh -c 'mkdir -p /..."   29
minutes ago      Up 29 minutes (healthy)   0.0.0.0:9000->9000/tcp scripts_minio_1
97122976b30d        mongo:3.6                                  "docker-entrypoint..."   29
minutes ago      Up 29 minutes             27017/tcp              mongo
```

This indicates all the containers are running correctly, with no issues. However, to identify any stopped containers, use the following command:

```
$ docker ps -a
CONTAINER ID        IMAGE                                      COMMAND
CREATED             STATUS                          PORTS                    NAMES
a9ed052f99d6        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)
workers_dataset_upload_1
3656dae9e4cf        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                                 workers_purging_1
de9640523ad4        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                                 workers_analysis_1
9f461a9dbe0f        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)
workers_model_training_2
a380bc06279b        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)
workers_model_training_1
871e6f2f1f7f        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   29
minutes ago      Exited (137) 8 seconds ago                              tango
72cd51047081        minio/minio:RELEASE.2018-07-23T18-34-49Z   "sh -c 'mkdir -p /..."   29
minutes ago      Up 29 minutes (healthy)      0.0.0.0:9000->9000/tcp   scripts_minio_1
97122976b30d        mongo:3.6                                  "docker-entrypoint..."   29
minutes ago      Up 29 minutes                27017/tcp               mongo
```

In the example above, the Tango container has failed and the state has transitioned from `running` to `exited`. To identify the cause of the container failure, use the following command:

```
$ docker inspect --format '{{ json .State.ExitCode }}' tango
137
```

This provides the exit code for the container. If the code is not familiar, check online resources, such as Exit Codes With Special Meanings in the Advanced Bash-Scripting Guide to identify the cause of the failure.

To determine whether the failure is caused by a memory issue or some other cause, use the following command:

```
$ docker inspect --format '{{ json .State.OOMKilled }}' tango
true
```

This indicates whether the container was able to allocate or utilize the amount of memory required to run the application. If there was insufficient memory, free up some memory from the system or kill unnecessary memory-intensive processes.

CPU Issues

CPU issues can be difficult to identify and debug because CPU is a compressible resource, unlike

memory. When memory requests exceed the limit, the kernel kills the process. When CPU exceeds the limit, the kernel simply allocates that process less CPU time, making it run slower. The healthcheck configured in the containers automatically detects unresponsive containers and then recreates them, making it more likely in this situation that the container recovers in good time.

The installation script ensures that minimum CPU and memory requirements are fulfilled during the setups of AICS, but these can change over time. If the resources are increased or you plan to change them, there is no need to run the installation script again. Instead, stop and then restart the application, following the procudure provided in Start and Stop AICS.

Restarting an Exited Container

The container can be brought up normally by issuing the following command. If it does not return to a normal running state, you will need to continue troubleshooting.

```
$ docker start tango
tango
```

Exit Codes With Special Meanings in the Advanced Bash-Scripting Guide lists some exit codes you might encounter.

The following links provide more information to help troubleshoot Docker containers:

- docker service ps
- docker inspect

Configure Maximum Log Size

GPR 9.0.013.01 and higher has a default maximum log file size of 100m. If you are running an earlier version of AICS or if your environment requires a different setting, use the instructions in this section to configure the log file size.

In a single-server environment, execute the commands in this section in your AICS server. In a high availability (HA) environment, review the output of the `docker service ls` command executed on node-1:

- Ensure that there are three tango instances.
- Note down how many worker containers you are running.

Perform the following steps to change the log file size setting:

1. Open the following files on your single server or node-1, depending on your environment:
   - **tango-swarm.yml**
   - **worker-swarm.yml**
2. Check that both files contain the following section at the same level as the `deploy:` section:

   ```
   logging:
   ```

```
        options:
         max-size: 100m
```

3. After you make changes, execute the `bash restart.sh` command on your single server/node-1. This executes a rolling restart of all containers.
4. Check the health of the system by executing the following command on your single server/node-1: `docker service ls`.
   Verify that the number of instances of tango and the workers containers is the same as when you started.
5. To clean up old log files that are not needed anymore, use the following Docker `prune` commands:

```
docker container prune -f
docker volume prune -f
docker network prune -f
```

Checking the Logs for HA AICS Containers

To access AICS logs when the services are running in a HA architecture, execute the following commands on any node in the cluster:

For Tango Logs

```
$ docker service logs tango_tango
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:04,722 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:04 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:14,977 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:14 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:25,204 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:25 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:35,432 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:35 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:45,659 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:45 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:55,886 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:55 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
```

For MongoDB Logs

```
$ docker service logs mongo_mongo1
I NETWORK  [conn59] end connection 127.0.0.1:47132 (37 connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.733+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:33372
#60 (38 connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.744+0000 I NETWORK  [conn60] received client metadata from 10.0.0.5:33372
conn60: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
```

```
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.745+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:33376
#61 (39 connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.755+0000 I NETWORK  [conn61] received client metadata from 10.0.0.5:33376
conn61: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:02.263+0000 I NETWORK  [conn61] end connection 10.0.0.5:33376 (38
connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:02.263+0000 I NETWORK  [conn60] end connection 10.0.0.5:33372 (37
connections now open)

$ docker service logs mongo_mongo2
 NETWORK  [LogicalSessionCacheRefresh] Starting new replica set monitor for rs0/
mongo_mongo1:27017,mongo_mongo2:27017,mongo_mongo3:27017
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:01.634+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:40506
#27 (16 connections now open)
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:01.643+0000 I NETWORK  [conn27] received client metadata from
127.0.0.1:40506 conn27: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB
Internal Client", version: "3.6.13" }, os: { type: "Linux", name: "Ubuntu", architecture:
"x86_64", version: "16.04" } }
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:01.651+0000 I NETWORK  [conn27] end connection 127.0.0.1:40506 (15
connections now open)
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:02.000+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:44322
#28 (16 connections now open)
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:02.016+0000 I NETWORK  [conn28] received client metadata from 10.0.0.5:44322
conn28: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:02.534+0000 I NETWORK  [conn28] end connection 10.0.0.5:44322 (15
connections now open)

$ docker service logs mongo_mongo3
I NETWORK  [LogicalSessionCacheRefresh] Starting new replica set monitor for rs0/
mongo_mongo1:27017,mongo_mongo2:27017,mongo_mongo3:27017
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.925+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:50938
#28 (16 connections now open)
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.935+0000 I NETWORK  [conn28] received client metadata from
127.0.0.1:50938 conn28: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB
Internal Client", version: "3.6.13" }, os: { type: "Linux", name: "Ubuntu", architecture:
"x86_64", version: "16.04" } }
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.944+0000 I NETWORK  [conn28] end connection 127.0.0.1:50938 (15
connections now open)
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.999+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:47094
#29 (16 connections now open)
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:02.013+0000 I NETWORK  [conn29] received client metadata from 10.0.0.5:47094
conn29: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
```

```
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:02.533+0000 I NETWORK  [conn29] end connection 10.0.0.5:47094 (15
connections now open)
```

And so on, for however many MongoDB nodes you have configured.

For Workers Logs

```
$ docker service logs workers_analysis
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:09] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:19] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:29] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:39] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:49] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:42:00] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:42:10] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:42:20] "GET /status HTTP/1.1" 200 -

$ docker service logs workers_model_training
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:41:59] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:09] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:19] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:29] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:39] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:49] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:59] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:09] "GET /status HTTP/1.1" 200 -

$ docker service logs workers_purging
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:26] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:36] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:46] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:56] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:07] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:17] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:27] "GET /status HTTP/1.1" 200 -
```

```
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:37] "GET /status HTTP/1.1" 200 -

$ docker service logs workers_dataset_upload
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:47] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:57] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:07] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:18] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:28] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:38] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:48] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:58] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:44:08] "GET /status HTTP/1.1" 200 -
```

To return only the last *N* lines of a log file, use the same commands as above, appending the command `--tail` *N*, as in the following example:

```
$ docker service logs workers_analysis --tail 100
```

To continuously stream output of a log, use the same commands as above, appending the command `-f`, as in the following example:

```
$ docker service logs workers_analysis -f
```