ansible-runner Documentation

Red Hat Ansible

Nov 17, 2021

CONTENTS:

Ansible Runner is a tool and python library that helps when interfacing with Ansible directly or as part of another system whether that be through a container image interface, as a standalone tool, or as a Python module that can be imported. The goal is to provide a stable and consistent interface abstraction to Ansible. This allows **Ansible** to be embedded into other systems that don't want to manage the complexities of the interface on their own (such as CI/CD platforms, Jenkins, or other automated tooling).

Ansible Runner represents the modularization of the part of Ansible Tower/AWX that is responsible for running `ansible` and `ansible-playbook` tasks and gathers the output from it. It does this by presenting a common interface that doesn't change, even as **Ansible** itself grows and evolves.

Part of what makes this tooling useful is that it can gather its inputs in a flexible way (See *Introduction to Ansible Runner*:). It also has a system for storing the output (stdout) and artifacts (host-level event data, fact data, etc) of the playbook run.

There are 3 primary ways of interacting with **Runner**

- A standalone command line tool (`ansible-runner`) that can be started in the foreground or run in the background asynchronously
- A reference container image that can be used as a base for your own images and will work as a standalone container or running in Openshift or Kubernetes
- A python module - library interface

Ansible Runner can also be configured to send status and event data to other systems using a plugin interface, see *Sending Runner Status and Events to External Systems*.

Examples of this could include:

- Sending status to Ansible Tower/AWX
- Sending events to an external logging service

CHAPTER

ONE

INTRODUCTION TO ANSIBLE RUNNER

Runner is intended to be most useful as part of automation and tooling that needs to invoke Ansible and consume its results. Most of the parameterization of the **Ansible** command line is also available on the **Runner** command line but **Runner** also can rely on an input interface that is mapped onto a directory structure, an example of which can be seen in the source tree.

Further sections in this document refer to the configuration and layout of that hierarchy. This isn't the only way to interface with **Runner** itself. The Python module interface allows supplying these details as direct module parameters in many forms, and the command line interface allows supplying them directly as arguments, mimicking the behavior of `ansible-playbook`. Having the directory structure **does** allow gathering the inputs from elsewhere and preparing them for consumption by **Runner**, then the tooling can come along and inspect the results after the run.

This is best seen in the way Ansible **AWX** uses **Runner** where most of the content comes from the database (and other content-management components) but ultimately needs to be brought together in a single place when launching the **Ansible** task.

1.1 Runner Input Directory Hierarchy

This directory contains all necessary inputs. Here's a view of the demo directory showing an active configuration.

Note that not everything is required. Defaults will be used or values will be omitted if they are not provided.

```
.
├── env
│   ├── envvars
│   ├── extravars
│   ├── passwords
│   ├── cmdline
│   ├── settings
│   └── ssh_key
├── inventory
│   └── hosts
└── project
    ├── test.yml
    └── roles
        └── testrole
            ├── defaults
            ├── handlers
            ├── meta
            ├── README.md
            ├── tasks
```

(continues on next page)

(continued from previous page)

```
            ├── tests
            └── vars
```

1.2 The env directory

The **env** directory contains settings and sensitive files that inform certain aspects of the invocation of the **Ansible** process, an example of which can be found in the demo env directory. Each of these files can also be represented by a named pipe providing a bit of an extra layer of security. The formatting and expectation of these files differs slightly depending on what they are representing.

1.3 env/envvars

Note: For an example see the demo envvars.

Ansible Runner will inherit the environment of the launching shell (or container, or system itself). This file (which can be in json or yaml format) represents the environment variables that will be added to the environment at run-time:

```
---
TESTVAR: exampleval
```

1.4 env/extravars

Note: For an example see the demo extravars.

Ansible Runner gathers the extra vars provided here and supplies them to the **Ansible Process** itself. This file can be in either json or yaml format:

```
---
ansible_connection: local
test: val
```

1.5 env/passwords

Note: For an example see the demo passwords.

Warning: We expect this interface to change/simplify in the future but will guarantee backwards compatibility. The goal is for the user of **Runner** to not have to worry about the format of certain prompts emitted from **Ansible** itself. In particular, vault passwords need to become more flexible.

Ansible itself is set up to emit passwords to certain prompts, these prompts can be requested (`-k` for example to prompt for the connection password). Likewise, prompts can be emitted via vars_prompt and also Ansible Vault.

In order for **Runner** to respond with the correct password, it needs to be able to match the prompt and provide the correct password. This is currently supported by providing a yaml or json formatted file with a regular expression and a value to emit, for example:

```
---
"^SSH password:\\s*?$": "some_password"
"^BECOME password.*:\\s*?$": "become_password"
```

1.6 env/cmdline

Warning: Current **Ansible Runner** does not validate the command line arguments passed using this method so it is up to the playbook writer to provide a valid set of options The command line options provided by this method are lower priority than the ones set by **Ansible Runner**. For instance, this will not override *inventory* or *limit* values.

Ansible Runner gathers command line options provided here as a string and supplies them to the **Ansible Process** itself. This file should contain the arguments to be added, for example:

```
--tags one,two --skip-tags three -u ansible --become
```

1.7 env/ssh_key

Note: Currently only a single ssh key can be provided via this mechanism but this is set to change soon.

This file should contain the ssh private key used to connect to the host(s). **Runner** detects when a private key is provided and will wrap the call to **Ansible** in ssh-agent.

1.8 env/settings - Settings for Runner itself

The **settings** file is a little different than the other files provided in this section in that its contents are meant to control **Runner** directly.

- `idle_timeout`: `600` If no output is detected from ansible in this number of seconds the execution will be terminated.
- `job_timeout`: `3600` The maximum amount of time to allow the job to run for, exceeding this and the execution will be terminated.
- `pexpect_timeout`: `10` Number of seconds for the internal pexpect command to wait to block on input before continuing
- `pexpect_use_poll`: `True` Use `poll()` function for communication with child processes instead of `select()`. `select()` is used when the value is set to `False`. `select()` has a known limitation of using only up to 1024 file descriptors.
- `suppress_ansible_output`: `False` Allow output from ansible to not be printed to the screen

- `fact_cache`: `'fact_cache'` The directory relative to `artifacts` where `jsonfile` fact caching will be stored. Defaults to `fact_cache`. This is ignored if `fact_cache_type` is different than `jsonfile`.
- `fact_cache_type`: `'jsonfile'` The type of fact cache to use. Defaults to `jsonfile`.

1.8.1 Process Isolation Settings for Runner

The process isolation settings are meant to control the process isolation feature of **Runner**.

- `process_isolation`: `False` Enable limiting what directories on the filesystem the playbook run has access to.
- `process_isolation_executable`: `bwrap` Path to the executable that will be used to provide filesystem isolation.
- `process_isolation_path`: `/tmp` Path that an isolated playbook run will use for staging.
- `process_isolation_hide_paths`: `None` Path or list of paths on the system that should be hidden from the playbook run.
- `process_isolation_show_paths`: `None` Path or list of paths on the system that should be exposed to the playbook run.
- `process_isolation_ro_paths`: `None` Path or list of paths on the system that should be exposed to the playbook run as read-only.

1.8.2 Container-based Execution

The `--containerized` setting instructs **Ansible Runner** to execute **Ansible** tasks inside a container environment. A default execution environment is provided on Quay.io at ansible/ansible-runner. Users also have the option of building their own container for executing playbooks, however.

To build an execution environment locally, run:

```
docker build --rm=true -t custom-container -f Dockerfile .
```

or, using podman:

```
podman build --rm=true -t custom-container -f Dockerfile .
```

To run Ansible Runner with your custom container:

```
ansible-runner run --container-image custom-container -p playbook.yml
```

See `ansible-runner -h` for other container-related options.

1.8.3 Performance Data Collection Settings for Runner

Runner is capable of collecting performance data (namely cpu usage, memory usage, and pid count) during the execution of a playbook run.

Resource profiling is made possible by the use of control groups (often referred to simply as cgroups). When a process runs inside of a cgroup, the resources used by that specific process can be measured.

Before enabling Runner's resource profiling feature, users must create a cgroup that **Runner** can use. It is worth noting that only privileged users can create cgroups. The new cgroup should be associated with the same user (and related group) that will be invoking **Runner**. The following command accomplishes this on a RHEL system:

```
sudo yum install libcgroup-tools
sudo cgcreate -a `whoami` -t `whoami` -g cpuacct,memory,pids:ansible-runner
```

In the above command, `cpuacct`, `memory`, and `pids` refer to kernel resource controllers, while `ansible-runner` refers to the name of the cgroup being created. More detailed information on the structure of cgroups can be found in the RHEL guide on Managing, monitoring, and updating the kernel

After a cgroup has been created, the following settings can be used to configure resource profiling. Note that `resource_profiling_base_cgroup` must match the name of the cgroup you create.

- `resource_profiling`: `False` Enable performance data collection.
- `resource_profiling_base_cgroup`: `ansible-runner` Top-level cgroup used to measure playbook resource utilization.
- `resource_profiling_cpu_poll_interval`: `0.25` Polling interval in seconds for collecting cpu usage.
- `resource_profiling_memory_poll_interval`: `0.25` Polling interval in seconds for collecting memory usage.
- `resource_profiling_pid_poll_interval`: `0.25` Polling interval in seconds for measuring PID count.
- `resource_profiling_results_dir`: `None` Directory where resource utilization data will be written (if not specified, will be placed in the `profiling_data` folder under the private data directory).

1.9 Inventory

The **Runner** `inventory` location under the private data dir has the same expectations as inventory provided directly to ansible itself. It can be either a single file or script or a directory containing static inventory files or scripts. This inventory is automatically loaded and provided to **Ansible** when invoked and can be further overridden on the command line or via the `ANSIBLE_INVENTORY` environment variable to specify the hosts directly. Giving an absolute path for the inventory location is best practice, because relative paths are interpreted relative to the `current working directory` which defaults to the `project` directory.

1.10 Project

The **Runner** `project` directory is the playbook root containing playbooks and roles that those playbooks can consume directly. This is also the directory that will be set as the `current working directory` when launching the **Ansible** process.

1.11 Modules

Runner has the ability to execute modules directly using Ansible ad-hoc mode.

1.12 Roles

Runner has the ability to execute Roles directly without first needing a playbook to reference them. This directory holds roles used for that. Behind the scenes, **Runner** will generate a playbook and invoke the `Role`.

1.13 Runner Artifacts Directory Hierarchy

This directory will contain the results of **Runner** invocation grouped under an `identifier` directory. This identifier can be supplied to **Runner** directly and if not given, an identifier will be generated as a UUID. This is how the directory structure looks from the top level:

```
.
├── artifacts
│   └── identifier
├── env
├── inventory
├── profiling_data
├── project
└── roles
```

The artifact directory itself contains a particular structure that provides a lot of extra detail from a running or previously-run invocation of Ansible/Runner:

```
.
├── artifacts
│   └── 37f639a3-1f4f-4acb-abee-ea1898013a25
│       ├── fact_cache
│       │   └── localhost
│       ├── job_events
│       │   ├── 1-34437b34-addd-45ae-819a-4d8c9711e191.json
│       │   ├── 2-8c164553-8573-b1e0-76e1-000000000006.json
│       │   ├── 3-8c164553-8573-b1e0-76e1-00000000000d.json
│       │   ├── 4-f16be0cd-99e1-4568-a599-546ab80b2799.json
│       │   ├── 5-8c164553-8573-b1e0-76e1-000000000008.json
│       │   ├── 6-981fd563-ec25-45cb-84f6-e9dc4e6449cb.json
│       │   └── 7-01c7090a-e202-4fb4-9ac7-079965729c86.json
│       ├── rc
│       ├── status
│       └── stdout
```

The **rc** file contains the actual return code from the **Ansible** process.

The **status** file contains one of three statuses suitable for displaying:

- success: The **Ansible** process finished successfully
- failed: The **Ansible** process failed
- timeout: The **Runner** timeout (see *env/settings - Settings for Runner itself*)

The **stdout** file contains the actual stdout as it appears at that moment.

1.14 Runner Artifact Job Events (Host and Playbook Events)

Runner gathers the individual task and playbook events that are emitted as part of the **Ansible** run. This is extremely helpful if you don't want to process or read the stdout returned from **Ansible** as it contains much more detail and status than just the plain stdout. It does some of the heavy lifting of assigning order to the events and stores them in json format under the `job_events` artifact directory. It also takes it a step further than normal **Ansible** callback plugins in that it will store the `stdout` associated with the event alongside the raw event data (along with stdout line numbers). It also generates dummy events for stdout that didn't have corresponding host event data:


```
{
  "uuid": "8c164553-8573-b1e0-76e1-000000000008",
  "parent_uuid": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "counter": 5,
  "stdout": "\r\nTASK [debug]␣
↪*********************************************************************",
  "start_line": 5,
  "end_line": 7,
  "event": "playbook_on_task_start",
  "event_data": {
    "playbook": "test.yml",
    "playbook_uuid": "34437b34-addd-45ae-819a-4d8c9711e191",
    "play": "all",
    "play_uuid": "8c164553-8573-b1e0-76e1-000000000006",
    "play_pattern": "all",
    "task": "debug",
    "task_uuid": "8c164553-8573-b1e0-76e1-000000000008",
    "task_action": "debug",
    "task_path": "\/home\/mjones\/ansible\/ansible-runner\/demo\/project\/test.yml:3",
    "task_args": "msg=Test!",
    "name": "debug",
    "is_conditional": false,
    "pid": 10640
  },
  "pid": 10640,
  "created": "2018-06-07T14:54:58.410605"
}
```


If the playbook runs to completion without getting killed, the last event will always be the `stats` event:


```
{
  "uuid": "01c7090a-e202-4fb4-9ac7-079965729c86",
  "counter": 7,
  "stdout": "\r\nPLAY RECAP␣
↪*********************************************************************\r\n\u001b[0;
↪32mlocalhost,\u001b[0m                  : \u001b[0;32mok=2    \u001b[0m changed=0    ␣
↪unreachable=0    failed=0   \r\n",
  "start_line": 10,
  "end_line": 14,
  "event": "playbook_on_stats",
  "event_data": {
    "playbook": "test.yml",
    "playbook_uuid": "34437b34-addd-45ae-819a-4d8c9711e191",
    "changed": {
```


(continues on next page)

(continued from previous page)


```
    },
    "dark": {

    },
    "failures": {

    },
    "ok": {
      "localhost,": 2
    },
    "processed": {
      "localhost,": 1
    },
    "skipped": {

    },
    "artifact_data": {

    },
    "pid": 10640
  },
  "pid": 10640,
  "created": "2018-06-07T14:54:58.424603"
}
```


Note: The **Runner module interface** presents a programmatic interface to these events that allow getting the final status and performing host filtering of task events.

1.15 Runner Profiling Data Directory

If resource profiling is enabled for **Runner** the `profiling_data` directory will be populated with a set of files containing the profiling data:

```
.
├── profiling_data
│   ├── 0-34437b34-addd-45ae-819a-4d8c9711e191-cpu.json
│   ├── 0-34437b34-addd-45ae-819a-4d8c9711e191-memory.json
│   ├── 0-34437b34-addd-45ae-819a-4d8c9711e191-pids.json
│   ├── 1-8c164553-8573-b1e0-76e1-000000000006-cpu.json
│   ├── 1-8c164553-8573-b1e0-76e1-000000000006-memory.json
│   └── 1-8c164553-8573-b1e0-76e1-000000000006-pids.json
```

Each file is in JSON text format. Each line of the file will begin with a record separator (RS), continue with a JSON dictionary, and conclude with a line feed (LF) character. The following provides an example of what the resource files may look like. Note that that since the RS and LF are control characters, they are not actually printed below:


```
==> 0-525400c9-c704-29a6-4107-00000000000c-cpu.json <==
{"timestamp": 1568977988.6844425, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 97.12799768097156}
```


(continues on next page)

(continued from previous page)

```
{"timestamp": 1568977988.9394386, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 94.17538298892688}
{"timestamp": 1568977989.1901696, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 64.38272588006255}
{"timestamp": 1568977989.4594045, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 83.77387744259856}

==> 0-525400c9-c704-29a6-4107-00000000000c-memory.json <==
{"timestamp": 1568977988.4281094, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 36.21484375}
{"timestamp": 1568977988.6842303, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 57.87109375}
{"timestamp": 1568977988.939303, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 66.60546875}
{"timestamp": 1568977989.1900482, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 71.4609375}
{"timestamp": 1568977989.4592078, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 38.25390625}

==> 0-525400c9-c704-29a6-4107-00000000000c-pids.json <==
{"timestamp": 1568977988.4284189, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 5}
{"timestamp": 1568977988.6845856, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 6}
{"timestamp": 1568977988.939547, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 8}
{"timestamp": 1568977989.1902773, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 13}
{"timestamp": 1568977989.4593227, "task_name": "Gathering Facts", "task_uuid": "525400c9-
↪c704-29a6-4107-00000000000c", "value": 6}
```

- Resource profiling data is grouped by playbook task.
- For each task, there will be three files, corresponding to cpu, memory and pid count data.
- Each file contains a set of data points collected over the course of a playbook task.
- If a task executes quickly and the polling rate for a given metric is large enough, it is possible that no profiling data may be collected during the task's execution. If this is the case, no data file will be created.

CHAPTER

TWO

INSTALLING ANSIBLE RUNNER

Ansible Runner requires Python >= 3.8 and is provided from several different locations depending on how you want to use it.

2.1 Using pip

To install the latest version from the Python Package Index:

```
$ pip install ansible-runner
```

2.2 Fedora

To install from the latest Fedora sources:

```
$ dnf install python-ansible-runner
```

2.3 Debian

Add an ansible-runner repository:

```
$ apt-get update
$ echo 'deb https://releases.ansible.com/ansible-runner/deb/ <trusty|xenial|stretch> main
↪' > /etc/apt/sources.list.d/ansible.list
```

Add a key:

```
$ apt-key adv --keyserver keyserver.ubuntu.com --recv 3DD29021
```

Install the package:

```
$ apt-get update
$ apt-get install ansible-runner
```

2.4 From source

Check out the source code from github:

```
$ git clone git://github.com/ansible/ansible-runner
```

Or download from the releases page

Then install:

```
$ pip install virtualenvwrapper
$ mkvirtualenv ansible-runner
$ pip install -e .
```

2.5 Build the distribution

To produce both wheel and sdist:

make dist

To produce an installable `wheel`:

```
make wheel
```

To produce a distribution tarball:

```
make sdist
```

2.6 Building the base container image

Make sure the `wheel` distribution is built (see *Build the distribution*) and run:

```
make image
```

2.7 Building the RPM

The RPM build uses a container image to bootstrap the environment in order to produce the RPM. Make sure you have docker installed and proceed with:

```
make rpm
```

CHAPTER

THREE

SENDING RUNNER STATUS AND EVENTS TO EXTERNAL SYSTEMS

Runner can store event and status data locally for retrieval, it can also emit this information via callbacks provided to the module interface.

Alternatively **Runner** can be configured to send events to an external system via installable plugins, there are currently two available

3.1 Event Structure

There are two types of events that are emitted via plugins:

- status events:

  These are sent whenever Runner's status changes (see *Runner.status_handler*) for example:

```
{"status": "running", "runner_ident": "XXXX" }
```

- ansible events:

  These are sent during playbook execution for every event received from **Ansible** (see *Playbook and Host Events*) for example:

```
{"runner_ident": "XXXX", <rest of event structure }
```

3.2 HTTP Status/Event Emitter Plugin

This sends status and event data to a URL in the form of json encoded POST requests.

This plugin is available from the ansible-runner-http github repo and is also available to be installed from pip:

```
$ pip install ansible-runner-http
```

In order to configure it, you can provide details in the Runner Settings file (see *env/settings - Settings for Runner itself*):

- *runner_http_url*: The url to receive the `POST`
- *runner_http_headers*: Headers to send along with the request.

The plugin also supports unix file-based sockets with:

- *runner_http_url*: The path to the unix socket
- *runner_http_path*: The path that will be included as part of the request to the socket

Some of these settings are also available as environment variables:

- RUNNER_HTTP_URL
- RUNNER_HTTP_PATH

3.3 ZeroMQ Status/Event Emitter Plugin

TODO

3.4 Writing your own Plugin

In order to write your own plugin interface and have it be picked up and used by **Runner** there are a few things that you'll need to do.

- Declare the module as a Runner entrypoint in your setup file (ansible-runner-http has a good example of this):

```
entry_points={'ansible_runner.plugins': 'modname = your_python_package_name'},
```

- Implement the `status_handler()` and `event_handler()` functions at the top of your package, for example see ansible-runner-http events.py and the `__init__` import at the top of the module package

After installing this, **Runner** will see the plugin and invoke the functions when status and events are sent. If there are any errors in your plugin they will be raised immediately and **Runner** will fail.

CHAPTER

FOUR

USING RUNNER AS A STANDALONE COMMAND LINE TOOL

The **Ansible Runner** command line tool can be used as a standard command line interface to **Ansible** itself but is primarily intended to fit into automation and pipeline workflows. Because of this, it has a bit of a different workflow than **Ansible** itself because you can select between a few different modes to launch the command.

While you can launch **Runner** and provide it all of the inputs as arguments to the command line (as you do with **Ansible** itself), there is another interface where inputs are gathered into a single location referred to in the command line parameters as `private_data_dir`. (see *Runner Input Directory Hierarchy*)

To view the parameters accepted by `ansible-runner`:

```
$ ansible-runner --help
```

An example invocation of the standalone `ansible-runner` utility:

```
$ ansible-runner run /tmp/private -p playbook.yml
```

Where playbook.yml is the playbook from the `/tmp/private/projects` directory, and `run` is the command mode you want to invoke **Runner** with

The different **commands** that runner accepts are:

- `run` starts `ansible-runner` in the foreground and waits until the underlying **Ansible** process completes before returning
- `start` starts `ansible-runner` as a background daemon process and generates a pid file
- `stop` terminates an `ansible-runner` process that was launched in the background with `start`
- `is-alive` checks the status of an `ansible-runner` process that was started in the background with `start`

While **Runner** is running it creates an `artifacts` directory (see *Runner Artifacts Directory Hierarchy*) regardless of what mode it was started in. The resulting output and status from **Ansible** will be located here. You can control the exact location underneath the `artifacts` directory with the `-i IDENT` argument to `ansible-runner`, otherwise a random UUID will be generated.

4.1 Executing Runner in the foreground

When launching **Runner** with the `run` command, as above, the program will stay in the foreground and you'll see output just as you expect from a normal **Ansible** process. **Runner** will still populate the `artifacts` directory, as mentioned in the previous section, to preserve the output and allow processing of the artifacts after exit.

4.2 Executing Runner in the background

When launching **Runner** with the `start` command, the program will generate a pid file and move to the background. You can check its status with the `is-alive` command, or terminate it with the `stop` command. You can find the stdout, status, and return code in the `artifacts` directory.

4.3 Running Playbooks

An example invocation using `demo` as private directory:

```
$ ansible-runner run demo --playbook test.yml
```

4.4 Running Modules Directly

An example invocating the `debug` module with `demo` as a private directory:

```
$ ansible-runner run demo -m debug --hosts localhost -a msg=hello
```

4.5 Running Roles Directly

An example invocation using `demo` as private directory and `localhost` as target:

```
$ ansible-runner run demo --role testrole --hosts localhost
```

Ansible roles directory can be provided with `--roles-path` option. Role variables can be passed with `--role-vars` at runtime.

4.6 Running with Process Isolation

Runner supports process isolation. Process isolation creates a new mount namespace where the root is on a tmpfs that is invisible from the host and is automatically cleaned up when the last process exits. You can enable process isolation by providing the `--process-isolation` argument on the command line. **Runner** as of version 2.0 defaults to using `podman` as the process isolation executable, but supports using any executable that is compatible with the `bubblewrap` CLI arguments by passing in the `--process-isolation-executable` argument:

```
$ ansible-runner --process-isolation ...
```

Runner supports various process isolation arguments that allow you to provide configuration details to the process isolation executable. To view the complete list of arguments accepted by `ansible-runner`:

```
$ ansible-runner --help
```

4.7 Running with Directory Isolation

If you need to be able to execute multiple tasks in parallel that might conflict with each other or if you want to make sure a single invocation of Ansible/Runner doesn't pollute or overwrite the playbook content you can give a base path:

```
$ ansible-runner --directory-isolation-base-path /tmp/runner
```

Runner will copy the project directory to a temporary directory created under that path, set it as the working directory, and execute from that location. After running that temp directory will be cleaned up and removed.

4.8 Outputting json (raw event data) to the console instead of normal output

Runner supports outputting json event data structure directly to the console (and stdout file) instead of the standard **Ansible** output, thus mimicking the behavior of the `json` output plugin. This is in addition to the event data that's already present in the artifact directory. All that is needed is to supply the `-j` argument on the command line:

```
$ ansible-runner ... -j ...
```

4.9 Cleaning up artifact directories

Using the command line argument `--rotate-artifacts` allows you to control the number of artifact directories that are present. Given a number as the parameter for this argument will cause **Runner** to clean up old artifact directories. The default value of `0` disables artifact directory cleanup.

CHAPTER

FIVE

USING RUNNER AS A PYTHON MODULE INTERFACE TO ANSIBLE

Ansible Runner is intended to provide a directly importable and usable API for interfacing with **Ansible** itself and exposes a few helper interfaces.

The modules center around the *Runner* object. The helper methods will either return an instance of this object which provides an interface to the results of executing the **Ansible** command or a tuple the actual output and error response based on the interface.

Ansible Runner itself is a wrapper around **Ansible** execution and so adds plugins and interfaces to the system in order to gather extra information and process/store it for use later.

5.1 Helper Interfaces

The helper *interfaces* provides a quick way of supplying the recommended inputs in order to launch a **Runner** process. These interfaces also allow overriding and providing inputs beyond the scope of what the standalone or container interfaces support. You can see a full list of the inputs in the linked module documentation.

5.2 `run()` helper function

ansible_runner.interface.run()

When called, this function will take the inputs (either provided as direct inputs to the function or from the *Runner Input Directory Hierarchy*), and execute **Ansible**. It will run in the foreground and return the *Runner* object when finished.

5.3 `run_async()` helper function

ansible_runner.interface.run_async()

Takes the same arguments as *ansible_runner.interface.run()* but will launch **Ansible** asynchronously and return a tuple containing the `thread` object and a *Runner* object. The **Runner** object can be inspected during execution.

5.4 run_command() helper function

ansible_runner.interface.run_command()

When called, this function will take the inputs (either provided as direct inputs to the function or from the *Runner Input Directory Hierarchy*), and execute the command passed either locally or within an container based on the parameters passed. It will run in the foreground and return a tuple of output and error response when finished. While running the within container image command the current local working diretory will be volume mounted within the container, in addition to this for any of ansible command line utilities the inventory, vault-password-file, private-key file path will be volume mounted if provided in the `cmdline_args` parameters.

5.5 run_command_async() helper function

ansible_runner.interface.run_command_async()

Takes the same arguments as *ansible_runner.interface.run_command()* but will launch asynchronously and return a tuple containing the `thread` object and a *Runner* object. The **Runner** object can be inspected during execution.

5.6 get_plugin_docs() helper function

ansible_runner.interface.get_plugin_docs()

When called, this function will take the inputs, and execute the ansible-doc command to return the either the plugin-docs or playbook snippet for the passed list of plugin names. The plugin docs can be fetched either from locally installed plugins or from within an container image based on the parameters passed. It will run in the foreground and return a tuple of output and error response when finished. While running the command within the container the current local working diretory will be volume mounted within the container.

5.7 get_plugin_docs_async() helper function

ansible_runner.interface.get_plugin_docs_async()

Takes the same arguments as *ansible_runner.interface.get_plugin_docs_async()* but will launch asynchronously and return a tuple containing the `thread` object and a *Runner* object. The **Runner** object can be inspected during execution.

5.8 get_plugin_list() helper function

ansible_runner.interface.get_plugin_list()

When called, this function will take the inputs, and execute the ansible-doc command to return the list of installed plugins. The installed plugin can be fetched either from local environment or from within an container image based on the parameters passed. It will run in the foreground and return a tuple of output and error response when finished. While running the command within the container the current local working diretory will be volume mounted within the container.

5.9 get_inventory() helper function

ansible_runner.interface.get_inventory()

When called, this function will take the inputs, and execute the ansible-inventory command to return the inventory releated information based on the action. If *action* is *list* it will return all the applicable configuration options for ansible, for *host* action it will return information of a single host andf for *graph* action it will return the inventory. The exectuin will be in the foreground and return a tuple of output and error response when finished. While running the command within the container the current local working diretory will be volume mounted within the container.

5.10 get_ansible_config() helper function

ansible_runner.interface.get_ansible_config()

When called, this function will take the inputs, and execute the ansible-config command to return the Ansible configuration releated information based on the action. If *action* is *list* it will return all the hosts related information including the host and group variables, for *dump* action it will return the enitre active configuration and it can be customized to return only the changed configuration value by settingg the *only_changed* boolean parameter to *True*. For *view* action it will return the view of the active configuration file. The exectuin will be in the foreground and return a tuple of output and error response when finished. While running the command within the container the current local working diretory will be volume mounted within the container.

5.11 The Runner object

The *Runner* object is returned as part of the execution of **Ansible** itself. Since it wraps both execution and output it has some helper methods for inspecting the results. Other than the methods and indirect properties, the instance of the object itself contains two direct properties:

- `rc` will represent the actual return code of the **Ansible** process
- **`status` will represent the state and can be one of:**
  - `unstarted`: This is a very brief state where the Runner task has been created but hasn't actually started yet.
  - `successful`: The `ansible` process finished successfully.
  - `failed`: The `ansible` process failed.

5.12 Runner.stdout

The *Runner* object contains a property *ansible_runner.runner.Runner.stdout* which will return an open file handle containing the `stdout` of the **Ansible** process.

5.13 Runner.stderr

When the `runner_mode` is set to `subprocess` the *Runner* object uses a property *ansible_runner.runner.Runner.stderr* which will return an open file handle containing the `stderr` of the **Ansible** process.

5.14 Runner.events

ansible_runner.runner.Runner.events is a `generator` that will return the *Playbook and Host Events* as Python `dict` objects.

5.15 Runner.stats

ansible_runner.runner.Runner.stats is a property that will return the final `playbook stats` event from **Ansible** in the form of a Python `dict`

5.16 Runner.host_events

ansible_runner.runner.Runner.host_events() is a method that, given a hostname, will return a list of only **Ansible** event data executed on that Host.

5.17 Runner.get_fact_cache

ansible_runner.runner.Runner.get_fact_cache() is a method that, given a hostname, will return a dictionary containing the Facts stored for that host during execution.

5.18 Runner.event_handler

A function passed to *__init__* of *Runner*, this is invoked every time an Ansible event is received. You can use this to inspect/process/handle events as they come out of Ansible. This function should return *True* to keep the event, otherwise it will be discarded.

5.19 Runner.cancel_callback

A function passed to `__init__` of *Runner*, and to the *ansible_runner.interface.run()* interface functions. This function will be called for every iteration of the *ansible_runner.interface.run()* event loop and should return *True* to inform **Runner** cancel and shutdown the **Ansible** process or *False* to allow it to continue.

5.20 Runner.finished_callback

A function passed to `__init__` of *Runner*, and to the *ansible_runner.interface.run()* interface functions. This function will be called immediately before the **Runner** event loop finishes once **Ansible** has been shut down.

5.21 Runner.status_handler

A function passed to `__init__` of *Runner* and to the *ansible_runner.interface.run()* interface functions. This function will be called any time the `status` changes, expected values are:

- *starting*: Preparing to start but hasn't started running yet
- *running*: The **Ansible** task is running
- *canceled*: The task was manually canceled either via callback or the cli
- *timeout*: The timeout configured in Runner Settings was reached (see *env/settings - Settings for Runner itself*)
- *failed*: The **Ansible** process failed
- *successful*: The **Ansible** process succeeded

5.22 Usage examples

```
import ansible_runner
r = ansible_runner.run(private_data_dir='/tmp/demo', playbook='test.yml')
print("{}: {}".format(r.status, r.rc))
# successful: 0
for each_host_event in r.events:
    print(each_host_event['event'])
print("Final status:")
print(r.stats)
```

```
import ansible_runner
r = ansible_runner.run(private_data_dir='/tmp/demo', host_pattern='localhost', module=
↪'shell', module_args='whoami')
print("{}: {}".format(r.status, r.rc))
# successful: 0
for each_host_event in r.events:
    print(each_host_event['event'])
print("Final status:")
print(r.stats)
```

```
# run the role named 'myrole' contained in the '<private_data_dir>/project/roles' directory
r = ansible_runner.run(private_data_dir='/tmp/demo', role='myrole')
print("{}: {}".format(r.status, r.rc))
print(r.stats)
```

```
# run ansible/generic commands in interactive mode within container
out, err, rc = ansible_runner.run_command(
    executable_cmd='ansible-playbook',
```

(continues on next page)

(continued from previous page)

```
    cmdline_args=['gather.yaml', '-i', 'inventory', '-vvvv', '-k'],
    input_fd=sys.stdin,
    output_fd=sys.stdout,
    error_fd=sys.stderr,
    host_cwd='/home/demo',
    process_isolation=True,
    container_image='network-ee'
)
print("rc: {}".format(rc))
print("out: {}".format(out))
print("err: {}".format(err))
```

```
# run ansible/generic commands in interactive mode locally
out, err, rc = ansible_runner.run_command(
    executable_cmd='ansible-playbook',
    cmdline_args=['gather.yaml', '-i', 'inventory', '-vvvv', '-k'],
    input_fd=sys.stdin,
    output_fd=sys.stdout,
    error_fd=sys.stderr,
)
print("rc: {}".format(rc))
print("out: {}".format(out))
print("err: {}".format(err))
```

```
# get plugin docs from within container
out, err = ansible_runner.get_plugin_docs(
    plugin_names=['vyos.vyos.vyos_command'],
    plugin_type='module',
    response_format='json',
    process_isolation=True,
    container_image='network-ee'
)
print("out: {}".format(out))
print("err: {}".format(err))
```

```
# get plugin docs from within container in async mode
thread_obj, runner_obj = ansible_runner.get_plugin_docs_async(
    plugin_names=['ansible.netcommon.cli_config', 'ansible.netcommon.cli_command'],
    plugin_type='module',
    response_format='json',
    process_isolation=True,
    container_image='network-ee'
)
while runner_obj.status not in ['canceled', 'successful', 'timeout', 'failed']:
    time.sleep(0.01)
    continue

print("out: {}".format(runner_obj.stdout.read()))
print("err: {}".format(runner_obj.stderr.read()))
```

```
# get plugin list installed on local system
out, err = ansible_runner.get_plugin_list()
print("out: {}".format(out))
print("err: {}".format(err))
```

```
# get plugins with file list from within container
out, err = ansible_runner.get_plugin_list(list_files=True, process_isolation=True,␣
↪container_image='network-ee')
print("out: {}".format(out))
print("err: {}".format(err))
```

```
# get list of changed ansible configuration values
out, err = ansible_runner.get_ansible_config(action='dump',  config_file='/home/demo/
↪ansible.cfg', only_changed=True)
print("out: {}".format(out))
print("err: {}".format(err))

# get ansible inventory information
out, err = ansible_runner.get_inventory(
    action='list',
    inventories=['/home/demo/inventory1', '/home/demo/inventory2'],
    response_format='json',
    process_isolation=True,
    container_image='network-ee'
)
print("out: {}".format(out))
print("err: {}".format(err))
```

5.23 Providing custom behavior and inputs

TODO

The helper methods are just one possible entrypoint, extending the classes used by these helper methods can allow a lot more custom behavior and functionality.

Show:

- How `Runner Config` is used and how overriding the methods and behavior can work
- Show how custom cancel and status callbacks can be supplied.

CHAPTER

SIX

USING RUNNER WITH EXECUTION ENVIRONMENTS

Execution Environments are meant to be a consistent, reproducible, portable, and sharable method to run Ansible Automation jobs in the exact same way on your laptop as they are executed in Ansible AWX. This aids in the development of automation jobs and Ansible Content that is meant to be run in **Ansible AWX**, Ansible Tower, or via Red Hat Ansible Automation Platform in a predictable way.

More specifically, the term **Execution Environments** within the context of **Ansible Runner** refers to the container runtime execution of **Ansible** via **Ansible Runner** within an OCI Compliant Container Runtime using an OCI Compliant Container Image that appropriately bundles Ansible Base, Ansible Collection Content, and the runtime dependencies required to support these contents. The base image is the Red Hat Enterprise Linux Universal Base Image and the build tooling provided by Ansible Builder aids in the creation of these images.

All aspects of running **Ansible Runner** in standalone mode (see: *Using Runner as a standalone command line tool*) are true here with the exception that the process isolation is inherently a container runtime (podman by default).

6.1 Using Execution Environments from Protected Registries

When a job is run that uses an execution environment container image from a private/protected registry, you will first need to authenticate to the registry.

If you are running the job manually via *ansible-runner run*, logging in on the command line via *podman login* first is a method of authentication. Alternatively, creating a *container_auth_data* dictionary with the keys *host*, *username*, and *password* and putting that in the job's *env/settings* file is another way to ensure a successful pull of a protected execution environment container image. Note that this involves listing sensitive information in a file which will not automatically get cleaned up after the job run is complete.

When running a job remotely via AWX or Ansible Tower, Ansible Runner can pick up the authentication information from the Container Registry Credential that was provided by the user. The *host*, *username*, *password*, and *verify_ssl* inputs from the credential are passed into Ansible Runner via the *container_auth_data* dictionary as key word arguments into a *json* file which gets deleted at the end of the job run (even if the job was canceled/interrupted), enabling the bypassing of sensitive information from any potentially persistent job-related files.

6.2 Notes and Considerations

There are some differences between using Ansible Runner and running Ansible directly from the command line that have to do with configuration, content locality, and secret data.

6.2.1 Secrets

Typically with Ansible you are able to provide secret data via a series of mechanisms, many of which are pluggable and configurable. When using Ansible Runner, however, certain considerations need to be made; these are analogous to how Ansible AWX and Tower manage this information.

See *Runner Input Directory Hierarchy* for more information

6.2.2 Container Names

Like all ansible-runner jobs, each job has an identifier associated with it which is also the name of the artifacts subfolder where results are saved to. When a container for job isolation is launched, it will be given a name of `ansible_runner_<job identifier>`. Some characters from the job identifier may be replaced with underscores for compatibility with names that Podman and Docker allow.

This name is used internally if a command needs to be ran against the container at a later time (e.g., to stop the container when the job is canceled).

6.2.3 ~/.ssh/ symlinks

In order to make the `run` container execution of Ansible easier, Ansible Runner will automatically bind mount your local ssh agent UNIX-domain socket (`SSH_AUTH_SOCK`) into the container runtime. However, this does not work if files in your `~/.ssh/` directory happen to be symlinked to another directory that is also not mounted into the container runtime. The Ansible Runner `run` subcommand provides the `--container-volume-mount` option to address this, among other things.

Here is an example of an ssh config file that is a symlink:

```
$ $ ls -l ~/.ssh/config
lrwxrwxrwx. 1 myuser myuser 34 Jul 15 19:27 /home/myuser/.ssh/config -> /home/myuser/
↪dotfiles/ssh_config

$ ansible-runner run \
    --container-volume-mount /home/myuser/dotfiles/:/home/myuser/dotfiles/ \
    --process-isolation --process-isolation-executable podman \
    /tmp/private --playbook my_playbook.yml -i my_inventory.ini
```

CHAPTER

SEVEN

USING RUNNER AS A CONTAINER INTERFACE TO ANSIBLE

The design of **Ansible Runner** makes it especially suitable for controlling the execution of **Ansible** from within a container for single-purpose automation workflows. A reference container image definition is provided and is also published to Quay.io.

```
$ podman run --rm -e RUNNER_PLAYBOOK=test.yml -v $PWD/demo:/runner quay.io/ansible/
↪ansible-runner:latest
 PLAY [all] *********************************************************************

 TASK [Gathering Facts] *********************************************************
 ok: [localhost]

 TASK [debug] *******************************************************************
 ok: [localhost] => {
   "msg": "Test!"
 }

 PLAY RECAP *********************************************************************
 localhost                  : ok=2    changed=0    unreachable=0    failed=0
```

The reference container image is purposefully light-weight and only containing the dependencies necessary to run `ansible-runner` itself. It's intended to be overridden.

7.1 Overriding the reference container image

TODO

7.2 Gathering output from the reference container image

TODO

7.3 Changing the console output to emit raw events

This can be useful when directing task-level event data to an external system by means of the container's console output.

See *Running with Process Isolation*

CHAPTER

EIGHT

REMOTE JOB EXECUTION

Ansible Runner supports the concept that a job run may be requested on one host but executed on another. This capability is primarily intended to be used by Receptor.

Support for this in Runner involves a three phase process.

- **Transmit**: Convert the job to a binary format that can be sent to the worker node.
- **Worker**: Actually execute the job.
- **Process**: Receive job results and process them.

The following command illustrates how the three phases work together:

```
$ ansible-runner transmit ./demo -p test.yml | ansible-runner worker | ansible-runner␣
↪process ./demo
```

In this example, the *ansible-runner transmit* command is given a private data directory of *./demo* and told to select the *test.yml* playbook from it. Instead of executing the playbook as *ansible-runner run* would do, the data dir and command line parameters are converted to a compressed binary stream that is emitted as stdout. The *transmit* command generally takes the same command line parameters as the *run* command.

The *ansible-runner worker* command accepts this stream, runs the playbook, and generates a new compressed binary stream of the resulting job events and artifacts. This command optionally accepts the *–private-data-dir* option. If provided, it will extract the contents sent from *ansible-runner transmit* into that directory.

The *ansible-runner process* command accepts the result stream from the worker, and fires all the normal callbacks and does job event processing. In the command above, this results in printing the playbook output and saving artifacts to the data dir. The *process* command takes a data dir as a parameter, to know where to save artifacts.

8.1 Cleanup of Resources Used by Jobs

The transmit and process commands do not offer any automatic deletion of the private data directory or artifacts, because these are how the user interacts with runner.

When running *ansible-runner worker*, if no *–private-data-dir* is given, it will extract the contents to a temporary directory which is deleted at the end of execution. If the `--private-data-dir` option is given, then the directory will persist after the run finishes unless the `--delete` flag is also set. In that case, the private data directory will be deleted before execution if it exists and also removed after execution.

The following command offers out-of-band cleanup.

$ ansible-runner worker cleanup –file-pattern=/tmp/foo_*

This would assure that old directories that fit the file glob "/tmp/foo_*" are deleted, which would could be used to assure cleanup of paths created by commands like *ansible-runner worker –private_data_dir=/tmp/foo_3*, for example. NOTE: see the *–grace-period* option, which sets the time window.

This command also takes a *–remove-images* option to run the podman or docker *rmi* command. There is otherwise no automatic cleanup of images used by a run, even if *container_auth_data* is used to pull from a private container registry. To be sure that layers are deleted as well, the *–image-prune* flag is necessary.

8.2 Python API

Python code importing Ansible Runner can make use of these facilities by setting the *streamer* parameter to *ansible_runner.interface.run*. This parameter can be set to *transmit*, *worker* or *process* to invoke each of the three stages. Other parameters are as normal in the CLI.

CHAPTER

NINE

DEVELOPER DOCUMENTATION

9.1 ansible_runner package

9.1.1 Subpackages

ansible_runner.callbacks package

Submodules

ansible_runner.callbacks.awx_display module

class ansible_runner.callbacks.awx_display.**CallbackModule**
Bases: display_callback.module.AWXDefaultCallbackModule

ansible_runner.callbacks.minimal module

class ansible_runner.callbacks.minimal.**CallbackModule**
Bases: display_callback.module.AWXMinimalCallbackModule

ansible_runner.display_callback package

Submodules

ansible_runner.display_callback.display module

ansible_runner.display_callback.events module

ansible_runner.display_callback.minimal module

class ansible_runner.display_callback.minimal.**CallbackModule**(*display=None*, *options=None*)
Bases: ansible.plugins.callback.CallbackBase

This is the default callback interface, which simply prints messages to stdout when new callback events are received.

CALLBACK_NAME = 'minimal'

CALLBACK_TYPE = 'stdout'

CALLBACK_VERSION = 2.0

v2_on_file_diff(*result*)

v2_runner_on_failed(*result*, *ignore_errors=False*)

v2_runner_on_ok(*result*)

v2_runner_on_skipped(*result*)

v2_runner_on_unreachable(*result*)

ansible_runner.display_callback.module module

class ansible_runner.display_callback.module.**AWXDefaultCallbackModule**
Bases: *ansible_runner.display_callback.module.BaseCallbackModule*, ansible.plugins.callback.default.CallbackModule

CALLBACK_NAME = 'awx_display'

class ansible_runner.display_callback.module.**AWXMinimalCallbackModule**
Bases: *ansible_runner.display_callback.module.BaseCallbackModule*, *ansible_runner.display_callback.minimal.CallbackModule*

CALLBACK_NAME = 'minimal'

v2_playbook_on_play_start(*play*)

v2_playbook_on_task_start(*task*, *is_conditional*)

class ansible_runner.display_callback.module.**BaseCallbackModule**
Bases: ansible.plugins.callback.CallbackBase

Callback module for logging ansible/ansible-playbook events.

CALLBACK_TYPE = 'stdout'

CALLBACK_VERSION = 2.0

EVENTS_WITHOUT_PLAY = ['playbook_on_start', 'playbook_on_stats']

EVENTS_WITHOUT_TASK = ['playbook_on_start', 'playbook_on_stats', 'playbook_on_setup', 'playbook_on_notify', 'playbook_on_import_for_host', 'playbook_on_not_import_for_host', 'playbook_on_no_hosts_matched', 'playbook_on_no_hosts_remaining']

capture_event_data(*event*, ***event_data*)

clear_play()

clear_task(*local=False*)

set_play(*play*)

set_playbook(*playbook*)

set_task(*task*, *local=False*)

v2_on_file_diff(*result*)

v2_playbook_on_cleanup_task_start(*task*)

v2_playbook_on_handler_task_start(*task*)

v2_playbook_on_import_for_host(*result*, *imported_file*)

v2_playbook_on_include(*included_file*)

v2_playbook_on_no_hosts_matched()

v2_playbook_on_no_hosts_remaining()

v2_playbook_on_not_import_for_host(*result*, *missing_file*)

v2_playbook_on_notify(*handler*, *host*)

v2_playbook_on_play_start(*play*)

v2_playbook_on_setup()

v2_playbook_on_start(*playbook*)

v2_playbook_on_stats(*stats*)

v2_playbook_on_task_start(*task*, *is_conditional*)

v2_playbook_on_vars_prompt(*varname*, *private=True*, *prompt=None*, *encrypt=None*, *confirm=False*, *salt_size=None*, *salt=None*, *default=None*, *unsafe=None*)

v2_runner_item_on_failed(*result*)

v2_runner_item_on_ok(*result*)

v2_runner_item_on_skipped(*result*)

v2_runner_on_async_failed(*result*)

v2_runner_on_async_ok(*result*)

v2_runner_on_async_poll(*result*)

v2_runner_on_failed(*result*, *ignore_errors=False*)

v2_runner_on_file_diff(*result*, *diff*)

v2_runner_on_no_hosts(*task*)

v2_runner_on_ok(*result*)

v2_runner_on_skipped(*result*)

v2_runner_on_start(*host*, *task*)

Event used when host begins execution of a task

New in version 2.8.

v2_runner_on_unreachable(*result*)

v2_runner_retry(*result*)

ansible_runner.display_callback.module.**current_time**()

Module contents

class ansible_runner.display_callback.**AWXDefaultCallbackModule**

Bases: *ansible_runner.display_callback.module.BaseCallbackModule*, ansible.plugins.callback.default.CallbackModule

CALLBACK_NAME = 'awx_display'

class ansible_runner.display_callback.**AWXMinimalCallbackModule**

Bases: *ansible_runner.display_callback.module.BaseCallbackModule*, *ansible_runner.display_callback.minimal.CallbackModule*

CALLBACK_NAME = 'minimal'

v2_playbook_on_play_start(*play*)

v2_playbook_on_task_start(*task*, *is_conditional*)

9.1.2 Submodules

ansible_runner.exceptions module

exception ansible_runner.exceptions.**AnsibleRunnerException**
Bases: Exception

Generic Runner Error

exception ansible_runner.exceptions.**CallbackError**
Bases: *ansible_runner.exceptions.AnsibleRunnerException*

Exception occurred in Callback

exception ansible_runner.exceptions.**ConfigurationError**
Bases: *ansible_runner.exceptions.AnsibleRunnerException*

Misconfiguration of Runner

ansible_runner.interface module

ansible_runner.interface.**get_ansible_config**(*action*, *config_file=None*, *only_changed=None*, ***kwargs*)
Run an ansible-config command to get ansible configuration releated details.

Parameters

- **action** (*str*) – Valid values are one of `list`, `dump`, `view` `list` returns all config options, `dump` returns the active configuration and `view` returns the view of configuration file.
- **config_file** (*str*) – Path to configuration file, defaults to first file found in precedence. .
- **only_changed** (*bool*) – The boolean value when set to `True` returns only the configurations that have changed from the default. This parameter is applicable only when `action` is set to `dump`.
- **runner_mode** (*str*) – The applicable values are `pexpect` and `subprocess`. Default is set to `subprocess`.
- **host_cwd** (*str*) – The current working directory from which the command in executable_cmd should be be executed.
- **envvars** (*dict*) – Environment variables to be used when running Ansible. Environment variables will also be read from `env/envvars` in `private_data_dir`
- **passwords** (*dict*) – A dictionary containing password prompt patterns and response values used when processing output from Ansible. Passwords will also be read from `env/passwords` in `private_data_dir`.
- **settings** (*dict*) – A dictionary containing settings values for the `ansible-runner` runtime environment. These will also be read from `env/settings` in `private_data_dir`.
- **ssh_key** (*str*) – The ssh private key passed to `ssh-agent` as part of the ansible-playbook run.
- **quiet** (*bool*) – Disable all output

- **json_mode** – Store event data in place of stdout on the console and in the stdout file
- **artifact_dir** (*str*) – The path to the directory where artifacts should live, this defaults to 'artifacts' under the private data dir
- **project_dir** (*str*) – The path to the playbook content, this defaults to 'project' within the private data dir
- **rotate_artifacts** (*int*) – Keep at most n artifact directories, disable with a value of 0 which is the default
- **timeout** (*int*) – The timeout value in seconds that will be passed to either `pexpect` of `subprocess` invocation (based on `runner_mode` selected) while executing command. It the timeout is triggered it will force cancel the execution.
- **process_isolation** (*bool*) – Enable process isolation, using a container engine (e.g. podman).
- **process_isolation_executable** (*str*) – Process isolation executable or container engine used to isolate execution. (default: podman)
- **container_image** (*str*) – Container image to use when running an ansible task (default: quay.io/ansible/ansible-runner:devel)
- **container_volume_mounts** (*list*) – List of bind mounts in the form 'host_dir:/container_dir:labels. (default: None)
- **container_options** (*list*) – List of container options to pass to execution engine.
- **container_workdir** (*str*) – The working directory within the container.
- **fact_cache** (*str*) – A string that will be used as the name for the subdirectory of the fact cache in artifacts directory. This is only used for 'jsonfile' type fact caches.
- **fact_cache_type** (*str*) – A string of the type of fact cache to use. Defaults to 'jsonfile'.
- **private_data_dir** (*str*) – The directory containing all runner metadata needed to invoke the runner module. Output artifacts will also be stored here for later consumption.
- **ident** (*str*) – The run identifier for this invocation of Runner. Will be used to create and name the artifact directory holding the results of the invocation.
- **event_handler** (*function*) – An optional callback that will be invoked any time an event is received by Runner itself, return True to keep the event
- **cancel_callback** (*function*) – An optional callback that can inform runner to cancel (returning True) or not (returning False)
- **finished_callback** (*function*) – An optional callback that will be invoked at shutdown after process cleanup.
- **status_handler** (*function*) – An optional callback that will be invoked any time the status changes (e.g. . . . started, running, failed, successful, timeout)
- **artifacts_handler** (*function*) – An optional callback that will be invoked at the end of the run to deal with the artifacts from the run.
- **check_job_event_data** (*bool*) – Check if job events data is completely generated. If event data is not completely generated and if value is set to 'True' it will raise 'AnsibleRunnerException' exception, if set to 'False' it log a debug message and continue execution. Default value is 'False'

Type json_mode: bool

Returns Returns a tuple of response and error string. In case if `runner_mode` is set to `pexpect` the error value is empty as `pexpect` uses same output descriptor for stdout and stderr.

`ansible_runner.interface.`**`get_inventory`**(*action*, *inventories*, *response_format=None*, *host=None*, *playbook_dir=None*, *vault_ids=None*, *vault_password_file=None*, *output_file=None*, *export=None*, ***kwargs*)

Run an ansible-inventory command to get inventory related details.

Parameters

- **`action`** (*str*) – Valid values are one of `graph`, `host`, `list` `graph` create inventory graph, `host` returns specific host info and works as inventory script and `list` output all hosts info and also works as inventory script.
- **`inventories`** (*list*) – List of inventory host path.
- **`response_format`** (*str*) – The output format for response. Valid values can be one of `json`, `yaml`, `toml`. Default is `json`. If `action` is `graph` only allowed value is `json`.
- **`host`** (*str*) – When `action` is set to `host` this parameter is used to get the host specific information.
- **`playbook_dir`** (*str*) – This parameter is used to sets the relative path for the inventory.
- **`vault_ids`** (*str*) – The vault identity to use.
- **`vault_password_file`** (*str*) – The vault password files to use.
- **`output_file`** (*str*) – The file path in which inventory details should be sent to.
- **`export`** – The boolean value if set represent in a way that is optimized for export,not as an accurate representation of how Ansible has processed it.
- **`runner_mode`** (*str*) – The applicable values are `pexpect` and `subprocess`. Default is set to `subprocess`.
- **`host_cwd`** (*str*) – The host current working directory to be mounted within the container (if enabled) and will be the work directory within container.
- **`envvars`** (*dict*) – Environment variables to be used when running Ansible. Environment variables will also be read from `env/envvars` in `private_data_dir`
- **`passwords`** (*dict*) – A dictionary containing password prompt patterns and response values used when processing output from Ansible. Passwords will also be read from `env/passwords` in `private_data_dir`.
- **`settings`** (*dict*) – A dictionary containing settings values for the `ansible-runner` runtime environment. These will also be read from `env/settings` in `private_data_dir`.
- **`ssh_key`** (*str*) – The ssh private key passed to `ssh-agent` as part of the ansible-playbook run.
- **`quiet`** (*bool*) – Disable all output
- **`json_mode`** (*bool*) – Store event data in place of stdout on the console and in the stdout file
- **`artifact_dir`** (*str*) – The path to the directory where artifacts should live, this defaults to 'artifacts' under the private data dir
- **`project_dir`** (*str*) – The path to the playbook content, this defaults to 'project' within the private data dir
- **`rotate_artifacts`** (*int*) – Keep at most n artifact directories, disable with a value of 0 which is the default

- **timeout** (*int*) – The timeout value in seconds that will be passed to either `pexpect` of `subprocess` invocation (based on `runner_mode` selected) while executing command. It the timeout is triggered it will force cancel the execution.
- **process_isolation** (*bool*) – Enable process isolation, using a container engine (e.g. podman).
- **process_isolation_executable** (*str*) – Process isolation executable or container engine used to isolate execution. (default: podman)
- **container_image** (*str*) – Container image to use when running an ansible task (default: quay.io/ansible/ansible-runner:devel)
- **container_volume_mounts** (*list*) – List of bind mounts in the form 'host_dir:/container_dir:labels. (default: None)
- **container_options** (*list*) – List of container options to pass to execution engine.
- **container_workdir** (*str*) – The working directory within the container.
- **fact_cache** (*str*) – A string that will be used as the name for the subdirectory of the fact cache in artifacts directory. This is only used for 'jsonfile' type fact caches.
- **fact_cache_type** (*str*) – A string of the type of fact cache to use. Defaults to 'jsonfile'.
- **private_data_dir** (*str*) – The directory containing all runner metadata needed to invoke the runner module. Output artifacts will also be stored here for later consumption.
- **ident** (*str*) – The run identifier for this invocation of Runner. Will be used to create and name the artifact directory holding the results of the invocation.
- **event_handler** (*function*) – An optional callback that will be invoked any time an event is received by Runner itself, return True to keep the event
- **cancel_callback** (*function*) – An optional callback that can inform runner to cancel (returning True) or not (returning False)
- **finished_callback** (*function*) – An optional callback that will be invoked at shutdown after process cleanup.
- **status_handler** (*function*) – An optional callback that will be invoked any time the status changes (e.g. . . started, running, failed, successful, timeout)
- **artifacts_handler** (*function*) – An optional callback that will be invoked at the end of the run to deal with the artifacts from the run.
- **check_job_event_data** (*bool*) – Check if job events data is completely generated. If event data is not completely generated and if value is set to 'True' it will raise 'AnsibleRunnerException' exception, if set to 'False' it log a debug message and continue execution. Default value is 'False'

Type export: bool

Returns Returns a tuple of response and error string. In case if `runner_mode` is set to `pexpect` the error value is empty as `pexpect` uses same output descriptor for stdout and stderr. If the vaue of `response_format` is `json` it returns a python dictionary object.

ansible_runner.interface.**get_plugin_docs**(*plugin_names*, *plugin_type=None*, *response_format=None*, *snippet=False*, *playbook_dir=None*, *module_path=None*, ***kwargs*)

Run an ansible-doc command to get plugin docs in the foreground and return a Runner object when complete.

Parameters

- **plugin_names** (*list*) – The name of the plugins to get docs.
- **plugin_type** (*str*) – The type of the plugin mentioned in plugins_names. Valid values are become, cache, callback, cliconf, connection, httpapi, inventory, lookup, netconf, shell, vars, module, strategy. If the value is not provided it defaults to module.
- **response_format** (*str*) – The output format for response. Valid values can be one of json or human and the response is either json string or plain text in human readable foramt. Default value is json.
- **snippet** (*bool*) – Show playbook snippet for specified plugin(s).
- **playbook_dir** (*str*) – This parameter is used to sets the relative path to handle playbook adjacent installed plugins.
- **module_path** (*str*) – This parameter is prepend colon-separated path(s) to module library (default=~/.ansible/plugins/modules:/usr/share/ansible/plugins/modules).
- **runner_mode** (*str*) – The applicable values are pexpect and subprocess. Default is set to subprocess.
- **host_cwd** (*str*) – The host current working directory to be mounted within the container (if enabled) and will be the work directory within container.
- **envvars** (*dict*) – Environment variables to be used when running Ansible. Environment variables will also be read from env/envvars in private_data_dir
- **passwords** (*dict*) – A dictionary containing password prompt patterns and response values used when processing output from Ansible. Passwords will also be read from env/passwords in private_data_dir.
- **settings** (*dict*) – A dictionary containing settings values for the ansible-runner runtime environment. These will also be read from env/settings in private_data_dir.
- **ssh_key** (*str*) – The ssh private key passed to ssh-agent as part of the ansible-playbook run.
- **quiet** (*bool*) – Disable all output
- **json_mode** (*bool*) – Store event data in place of stdout on the console and in the stdout file
- **artifact_dir** (*str*) – The path to the directory where artifacts should live, this defaults to 'artifacts' under the private data dir
- **project_dir** (*str*) – The path to the playbook content, this defaults to 'project' within the private data dir
- **rotate_artifacts** (*int*) – Keep at most n artifact directories, disable with a value of 0 which is the default
- **timeout** (*int*) – The timeout value in seconds that will be passed to either pexpect of subprocess invocation (based on runner_mode selected) while executing command. It the timeout is triggered it will force cancel the execution.
- **process_isolation** (*bool*) – Enable process isolation, using a container engine (e.g. podman).
- **process_isolation_executable** (*str*) – Process isolation executable or container engine used to isolate execution. (default: podman)
- **container_image** (*str*) – Container image to use when running an ansible task (default: quay.io/ansible/ansible-runner:devel)

- **container_volume_mounts** (*list*) – List of bind mounts in the form 'host_dir:/container_dir:labels. (default: None)
- **container_options** (*list*) – List of container options to pass to execution engine.
- **container_workdir** (*str*) – The working directory within the container.
- **fact_cache** (*str*) – A string that will be used as the name for the subdirectory of the fact cache in artifacts directory. This is only used for 'jsonfile' type fact caches.
- **fact_cache_type** (*str*) – A string of the type of fact cache to use. Defaults to 'jsonfile'.
- **private_data_dir** (*str*) – The directory containing all runner metadata needed to invoke the runner module. Output artifacts will also be stored here for later consumption.
- **ident** (*str*) – The run identifier for this invocation of Runner. Will be used to create and name the artifact directory holding the results of the invocation.
- **event_handler** (*function*) – An optional callback that will be invoked any time an event is received by Runner itself, return True to keep the event
- **cancel_callback** (*function*) – An optional callback that can inform runner to cancel (returning True) or not (returning False)
- **finished_callback** (*function*) – An optional callback that will be invoked at shutdown after process cleanup.
- **status_handler** (*function*) – An optional callback that will be invoked any time the status changes (e.g...started, running, failed, successful, timeout)
- **artifacts_handler** (*function*) – An optional callback that will be invoked at the end of the run to deal with the artifacts from the run.
- **check_job_event_data** (*bool*) – Check if job events data is completely generated. If event data is not completely generated and if value is set to 'True' it will raise 'AnsibleRunnerException' exception, if set to 'False' it log a debug message and continue execution. Default value is 'False'

Returns Returns a tuple of response and error string. In case if `runner_mode` is set to `pexpect` the error value is empty as `pexpect` uses same output descriptor for stdout and stderr. If the value of `response_format` is `json` it returns a python dictionary object.

ansible_runner.interface.**get_plugin_docs_async**(*plugin_names*, *plugin_type=None*, *response_format=None*, *snippet=False*, *playbook_dir=None*, *module_path=None*, ***kwargs*)

Run an ansible-doc command in the background which will start immediately. Returns the thread object and a Runner object.

This uses the same parameters as *ansible_runner.interface.get_plugin_docs()*

Returns A tuple containing a `threading.Thread` object and a *ansible_runner.runner.Runner* object

ansible_runner.interface.**get_plugin_list**(*list_files=None*, *response_format=None*, *plugin_type=None*, *playbook_dir=None*, *module_path=None*, ***kwargs*)

Run an ansible-doc command to get list of installed Ansible plugins.

Parameters

- **list_files** (*bool*) – The boolean parameter is set to `True` returns file path of the plugin along with the plugin name.

- **response_format** (*str*) – The output format for response. Valid values can be one of `json` or `human` and the response is either json string or plain text in human readable foramt. Default value is `json`.
- **plugin_type** (*str*) – The type of the plugin mentioned in plugins_names. Valid values are `become`, `cache`, `callback`, `cliconf`, `connection`, `httpapi`, `inventory`, `lookup`, `netconf`, `shell`, `vars`, `module`, `strategy`. If the value is not provided it defaults to `module`.
- **playbook_dir** (*str*) – This parameter is used to sets the relative path to handle playbook adjacent installed plugins.
- **module_path** (*str*) – This parameter is prepend colon-separated path(s) to module library (default=~/.ansible/plugins/modules:/usr/share/ansible/plugins/modules).
- **runner_mode** (*str*) – The applicable values are `pexpect` and `subprocess`. Default is set to `subprocess`.
- **host_cwd** (*str*) – The host current working directory to be mounted within the container (if enabled) and will be the work directory within container.
- **envvars** (*dict*) – Environment variables to be used when running Ansible. Environment variables will also be read from `env/envvars` in `private_data_dir`
- **passwords** (*dict*) – A dictionary containing password prompt patterns and response values used when processing output from Ansible. Passwords will also be read from `env/passwords` in `private_data_dir`.
- **settings** (*dict*) – A dictionary containing settings values for the `ansible-runner` runtime environment. These will also be read from `env/settings` in `private_data_dir`.
- **ssh_key** (*str*) – The ssh private key passed to `ssh-agent` as part of the ansible-playbook run.
- **quiet** (*bool*) – Disable all output
- **json_mode** (*bool*) – Store event data in place of stdout on the console and in the stdout file
- **artifact_dir** (*str*) – The path to the directory where artifacts should live, this defaults to 'artifacts' under the private data dir
- **project_dir** (*str*) – The path to the playbook content, this defaults to 'project' within the private data dir
- **rotate_artifacts** (*int*) – Keep at most n artifact directories, disable with a value of 0 which is the default
- **timeout** (*int*) – The timeout value in seconds that will be passed to either `pexpect` of `subprocess` invocation (based on `runner_mode` selected) while executing command. It the timeout is triggered it will force cancel the execution.
- **process_isolation** (*bool*) – Enable process isolation, using a container engine (e.g. podman).
- **process_isolation_executable** (*str*) – Process isolation executable or container engine used to isolate execution. (default: podman)
- **container_image** (*str*) – Container image to use when running an ansible task (default: quay.io/ansible/ansible-runner:devel)
- **container_volume_mounts** (*list*) – List of bind mounts in the form 'host_dir:/container_dir:labels. (default: None)
- **container_options** (*list*) – List of container options to pass to execution engine.

- **container_workdir** (*str*) – The working directory within the container.
- **fact_cache** (*str*) – A string that will be used as the name for the subdirectory of the fact cache in artifacts directory. This is only used for 'jsonfile' type fact caches.
- **fact_cache_type** (*str*) – A string of the type of fact cache to use. Defaults to 'jsonfile'.
- **private_data_dir** (*str*) – The directory containing all runner metadata needed to invoke the runner module. Output artifacts will also be stored here for later consumption.
- **ident** (*str*) – The run identifier for this invocation of Runner. Will be used to create and name the artifact directory holding the results of the invocation.
- **event_handler** (*function*) – An optional callback that will be invoked any time an event is received by Runner itself, return True to keep the event
- **cancel_callback** (*function*) – An optional callback that can inform runner to cancel (returning True) or not (returning False)
- **finished_callback** (*function*) – An optional callback that will be invoked at shutdown after process cleanup.
- **status_handler** (*function*) – An optional callback that will be invoked any time the status changes (e.g...started, running, failed, successful, timeout)
- **artifacts_handler** (*function*) – An optional callback that will be invoked at the end of the run to deal with the artifacts from the run.
- **check_job_event_data** (*bool*) – Check if job events data is completely generated. If event data is not completely generated and if value is set to 'True' it will raise 'AnsibleRunnerException' exception, if set to 'False' it log a debug message and continue execution. Default value is 'False'

Returns Returns a tuple of response and error string. In case if `runner_mode` is set to `pexpect` the error value is empty as `pexpect` uses same output descriptor for stdout and stderr. If the vaue of `response_format` is `json` it returns a python dictionary object.

ansible_runner.interface.**init_command_config**(*executable_cmd*, *cmdline_args=None*, ***kwargs*)

Initialize the Runner() instance

This function will properly initialize both run_command() and run_command_async() functions in the same way and return a value instance of Runner.

See parameters given to *ansible_runner.interface.run_command()*

ansible_runner.interface.**init_plugin_docs_config**(*plugin_names*, *plugin_type=None*, *response_format=None*, *snippet=False*, *playbook_dir=None*, *module_path=None*, ***kwargs*)

Initialize the Runner() instance

This function will properly initialize both get_plugin_docs() and get_plugin_docs_async() functions in the same way and return a value instance of Runner.

See parameters given to *ansible_runner.interface.get_plugin_docs()*

ansible_runner.interface.**init_runner**(***kwargs*)

Initialize the Runner() instance

This function will properly initialize both run() and run_async() functions in the same way and return a value instance of Runner.

See parameters given to *ansible_runner.interface.run()*

ansible_runner.interface.**run**(***kwargs*)

Run an Ansible Runner task in the foreground and return a Runner object when complete.

Parameters

- **private_data_dir** (*str*) – The directory containing all runner metadata needed to invoke the runner module. Output artifacts will also be stored here for later consumption.
- **ident** (*str*) – The run identifier for this invocation of Runner. Will be used to create and name the artifact directory holding the results of the invocation.
- **json_mode** (*bool*) – Store event data in place of stdout on the console and in the stdout file
- **playbook** (*str or filename or list*) – The playbook (either a list or dictionary of plays, or as a path relative to `private_data_dir/project`) that will be invoked by runner when executing Ansible.
- **module** – The module that will be invoked in ad-hoc mode by runner when executing Ansible.
- **module_args** – The module arguments that will be supplied to ad-hoc mode.
- **host_pattern** – The host pattern to match when running in ad-hoc mode.
- **inventory** (*str or dict or list*) – Overrides the inventory directory/file (supplied at `private_data_dir/inventory`) with a specific host or list of hosts. This can take the form of:
  - Path to the inventory file in the `private_data_dir`
  - Native python dict supporting the YAML/json inventory structure
  - A text INI formatted string
  - A list of inventory sources, or an empty list to disable passing inventory
- **role** (*str*) – Name of the role to execute.
- **roles_path** (*dict or list*) – Directory or list of directories to assign to ANSIBLE_ROLES_PATH
- **envvars** (*dict*) – Environment variables to be used when running Ansible. Environment variables will also be read from `env/envvars` in `private_data_dir`
- **extravars** (*dict*) – Extra variables to be passed to Ansible at runtime using `-e`. Extra vars will also be read from `env/extravars` in `private_data_dir`.
- **passwords** (*dict*) – A dictionary containing password prompt patterns and response values used when processing output from Ansible. Passwords will also be read from `env/passwords` in `private_data_dir`.
- **settings** (*dict*) – A dictionary containing settings values for the `ansible-runner` runtime environment. These will also be read from `env/settings` in `private_data_dir`.
- **ssh_key** (*str*) – The ssh private key passed to `ssh-agent` as part of the ansible-playbook run.
- **cmdline** (*str*) – Command line options passed to Ansible read from `env/cmdline` in `private_data_dir`
- **limit** (*str*) – Matches ansible's `--limit` parameter to further constrain the inventory to be used
- **forks** (*int*) – Control Ansible parallel concurrency
- **verbosity** (*int*) – Control how verbose the output of ansible-playbook is

- **quiet** (*bool*) – Disable all output
- **artifact_dir** (*str*) – The path to the directory where artifacts should live, this defaults to 'artifacts' under the private data dir
- **project_dir** (*str*) – The path to the playbook content, this defaults to 'project' within the private data dir
- **rotate_artifacts** (*int*) – Keep at most n artifact directories, disable with a value of 0 which is the default
- **timeout** (*int*) – The timeout value in seconds that will be passed to either `pexpect` of `subprocess` invocation (based on `runner_mode` selected) while executing command. It the timeout is triggered it will force cancel the execution.
- **streamer** (*str*) – Optionally invoke ansible-runner as one of the steps in the streaming pipeline
- **_input** (*file*) – An optional file or file-like object for use as input in a streaming pipeline
- **_output** (*file*) – An optional file or file-like object for use as output in a streaming pipeline
- **event_handler** (*function*) – An optional callback that will be invoked any time an event is received by Runner itself, return True to keep the event
- **cancel_callback** (*function*) – An optional callback that can inform runner to cancel (returning True) or not (returning False)
- **finished_callback** (*function*) – An optional callback that will be invoked at shutdown after process cleanup.
- **status_handler** (*function*) – An optional callback that will be invoked any time the status changes (e.g...started, running, failed, successful, timeout)
- **artifacts_handler** (*function*) – An optional callback that will be invoked at the end of the run to deal with the artifacts from the run.
- **process_isolation** (*bool*) – Enable process isolation, using either a container engine (e.g. podman) or a sandbox (e.g. bwrap).
- **process_isolation_executable** (*str*) – Process isolation executable or container engine used to isolate execution. (default: podman)
- **process_isolation_path** (*str*) – Path that an isolated playbook run will use for staging. (default: /tmp)
- **process_isolation_hide_paths** (*str or list*) – A path or list of paths on the system that should be hidden from the playbook run.
- **process_isolation_show_paths** (*str or list*) – A path or list of paths on the system that should be exposed to the playbook run.
- **process_isolation_ro_paths** (*str or list*) – A path or list of paths on the system that should be exposed to the playbook run as read-only.
- **container_image** (*str*) – Container image to use when running an ansible task (default: quay.io/ansible/ansible-runner:devel)
- **container_volume_mounts** (*list*) – List of bind mounts in the form 'host_dir:/container_dir. (default: None)
- **container_options** (*list*) – List of container options to pass to execution engine.
- **resource_profiling** (*bool*) – Enable collection of resource utilization data during playbook execution.

- **resource_profiling_base_cgroup** (*str*) – Name of existing cgroup which will be sub-grouped in order to measure resource utilization (default: ansible-runner)
- **resource_profiling_cpu_poll_interval** (*float*) – Interval (in seconds) between CPU polling for determining CPU usage (default: 0.25)
- **resource_profiling_memory_poll_interval** (*float*) – Interval (in seconds) between memory polling for determining memory usage (default: 0.25)
- **resource_profiling_pid_poll_interval** (*float*) – Interval (in seconds) between polling PID count for determining number of processes used (default: 0.25)
- **resource_profiling_results_dir** (*str*) – Directory where profiling data files should be saved (defaults to profiling_data folder inside private data dir)
- **directory_isolation_base_path** (*str*) – An optional path will be used as the base path to create a temp directory, the project contents will be copied to this location which will then be used as the working directory during playbook execution.
- **fact_cache** (*str*) – A string that will be used as the name for the subdirectory of the fact cache in artifacts directory. This is only used for 'jsonfile' type fact caches.
- **fact_cache_type** (*str*) – A string of the type of fact cache to use. Defaults to 'jsonfile'.
- **omit_event_data** (*bool*) – Omits extra ansible event data from event payload (stdout and event still included)
- **only_failed_event_data** (*bool*) – Omits extra ansible event data unless it's a failed event (stdout and event still included)
- **check_job_event_data** (*bool*) – Check if job events data is completely generated. If event data is not completely generated and if value is set to 'True' it will raise 'AnsibleRunnerException' exception, if set to 'False' it log a debug message and continue execution. Default value is 'False'

Returns A *ansible_runner.runner.Runner* object, or a simple object containing *rc* if run remotely

ansible_runner.interface.**run_async**(***kwargs*)

Runs an Ansible Runner task in the background which will start immediately. Returns the thread object and a Runner object.

This uses the same parameters as *ansible_runner.interface.run()*

Returns A tuple containing a `threading.Thread` object and a *ansible_runner.runner.Runner* object

ansible_runner.interface.**run_command**(*executable_cmd*, *cmdline_args=None*, ***kwargs*)

Run an (Ansible) commands in the foreground and return a Runner object when complete.

Parameters

- **executable_cmd** (*str*) – The command to be executed.
- **cmdline_args** (*list*) – A list of arguments to be passed to the executable command.
- **input_fd** (*file descriptor*) – This parameter is applicable when `runner_mode` is set to `subprocess`, it provides the input file descrption to interact with the sub-process running the command.
- **output_fd** (*file descriptor*) – The output file descriptor to stream the output of command execution.

- **error_fd** (*file descriptor*) – This parameter is applicable when runner_mode is set to subprocess, it provides the error file descrption to read the error received while executing the command.
- **runner_mode** (*str*) – The applicable values are pexpect and subprocess. If the value of input_fd parameter is set or the executable command is one of ansible-config, ansible-doc or ansible-galaxy the default value is set to subprocess else in other cases it is set to pexpect.
- **host_cwd** (*str*) – The host current working directory to be mounted within the container (if enabled) and will be the work directory within container.
- **envvars** (*dict*) – Environment variables to be used when running Ansible. Environment variables will also be read from env/envvars in private_data_dir
- **passwords** (*dict*) – A dictionary containing password prompt patterns and response values used when processing output from Ansible. Passwords will also be read from env/passwords in private_data_dir.
- **settings** (*dict*) – A dictionary containing settings values for the ansible-runner runtime environment. These will also be read from env/settings in private_data_dir.
- **ssh_key** (*str*) – The ssh private key passed to ssh-agent as part of the ansible-playbook run.
- **quiet** (*bool*) – Disable all output
- **json_mode** (*bool*) – Store event data in place of stdout on the console and in the stdout file
- **artifact_dir** (*str*) – The path to the directory where artifacts should live, this defaults to 'artifacts' under the private data dir
- **project_dir** (*str*) – The path to the playbook content, this defaults to 'project' within the private data dir
- **rotate_artifacts** (*int*) – Keep at most n artifact directories, disable with a value of 0 which is the default
- **timeout** (*int*) – The timeout value in seconds that will be passed to either pexpect of subprocess invocation (based on runner_mode selected) while executing command. It the timeout is triggered it will force cancel the execution.
- **process_isolation** (*bool*) – Enable process isolation, using a container engine (e.g. podman).
- **process_isolation_executable** (*str*) – Process isolation executable or container engine used to isolate execution. (default: podman)
- **container_image** (*str*) – Container image to use when running an ansible task (default: quay.io/ansible/ansible-runner:devel)
- **container_volume_mounts** (*list*) – List of bind mounts in the form 'host_dir:/container_dir:labels. (default: None)
- **container_options** (*list*) – List of container options to pass to execution engine.
- **container_workdir** (*str*) – The working directory within the container.
- **fact_cache** (*str*) – A string that will be used as the name for the subdirectory of the fact cache in artifacts directory. This is only used for 'jsonfile' type fact caches.
- **fact_cache_type** (*str*) – A string of the type of fact cache to use. Defaults to 'jsonfile'.

- **private_data_dir** (*str*) – The directory containing all runner metadata needed to invoke the runner module. Output artifacts will also be stored here for later consumption.
- **ident** (*str*) – The run identifier for this invocation of Runner. Will be used to create and name the artifact directory holding the results of the invocation.
- **event_handler** (*function*) – An optional callback that will be invoked any time an event is received by Runner itself, return True to keep the event
- **cancel_callback** (*function*) – An optional callback that can inform runner to cancel (returning True) or not (returning False)
- **finished_callback** (*function*) – An optional callback that will be invoked at shutdown after process cleanup.
- **status_handler** (*function*) – An optional callback that will be invoked any time the status changes (e.g...started, running, failed, successful, timeout)
- **artifacts_handler** (*function*) – An optional callback that will be invoked at the end of the run to deal with the artifacts from the run.
- **check_job_event_data** (*bool*) – Check if job events data is completely generated. If event data is not completely generated and if value is set to 'True' it will raise 'AnsibleRunnerException' exception, if set to 'False' it log a debug message and continue execution. Default value is 'False'

Returns Returns a tuple of response, error string and return code. In case if `runner_mode` is set to `pexpect` the error value is empty as `pexpect` uses same output descriptor for stdout and stderr.

ansible_runner.interface.**run_command_async**(*executable_cmd*, *cmdline_args=None*, ***kwargs*)

Run an (Ansible) commands in the background which will start immediately. Returns the thread object and a Runner object.

This uses the same parameters as *ansible_runner.interface.run_command()*

Returns A tuple containing a `threading.Thread` object and a *ansible_runner.runner.Runner* object

ansible_runner.loader module

class ansible_runner.loader.**ArtifactLoader**(*base_path*)

Bases: `object`

Handles loading and caching file contents from disk

This class will load the file contents and attempt to deserialize the contents as either JSON or YAML. If the file contents cannot be deserialized, the contents will be returned to the caller as a string.

The deserialized file contents are stored as a cached object in the instance to avoid any additional reads from disk for subsequent calls to load the same file.

abspath(*path*)

Transform the path to an absolute path

Args: path (string): The path to transform to an absolute path

Returns: string: The absolute path to the file

get_contents(*path*)

Loads the contents of the file specified by path

Args:

path (string): The relative or absolute path to the file to be loaded. If the path is relative, then it is combined with the base_path to generate a full path string

Returns: string: The contents of the file as a string

Raises: ConfigurationError: If the file cannot be loaded

`isfile`(*path*)
Check if the path is a file

Params path The path to the file to check. If the path is relative it will be exanded to an absolute path

Returns boolean

`load_file`(*path*, *objtype=None*, *encoding='utf-8'*)
Load the file specified by path

This method will first try to load the file contents from cache and if there is a cache miss, it will load the contents from disk

Args: path (string): The full or relative path to the file to be loaded

encoding (string): The file contents text encoding

objtype (object): The object type of the file contents. This is used to type check the deserialized content against the contents loaded from disk. Ignore serializing if objtype is string_types

Returns:

object: The deserialized file contents which could be either a string object or a dict object

Raises: ConfigurationError:

ansible_runner.runner module

`class` ansible_runner.runner.`Runner`(*config*, *cancel_callback=None*, *remove_partials=True*, *event_handler=None*, *artifacts_handler=None*, *finished_callback=None*, *status_handler=None*)

Bases: `object`

`event_callback`(*event_data*)
Invoked for every Ansible event to collect stdout with the event data and store it for later use

`property events`
A generator that will return all ansible job events in the order that they were emitted from Ansible

Example


```
{
   "event": "runner_on_ok",
   "uuid": "00a50d9c-161a-4b74-b978-9f60becaf209",
   "stdout": "ok: [localhost] => {\r\n    \"   msg\":\"Test!\"\r\n}",
   "counter": 6,
   "pid": 740,
   "created": "2018-04-05T18:24:36.096725",
   "end_line": 10,
   "start_line": 7,
   "event_data": {
      "play_pattern": "all",
      "play": "all",
```


(continues on next page)

(continued from previous page)


```
        "task": "debug",
        "task_args": "msg=Test!",
        "remote_addr": "localhost",
        "res": {
            "msg": "Test!",
            "changed": false,
            "_ansible_verbose_always": true,
            "_ansible_no_log": false
        },
        "pid": 740,
        "play_uuid": "0242ac11-0002-443b-cdb1-000000000006",
        "task_uuid": "0242ac11-0002-443b-cdb1-000000000008",
        "event_loop": null,
        "playbook_uuid": "634edeee-3228-4c17-a1b4-f010fdd42eb2",
        "playbook": "test.yml",
        "task_action": "debug",
        "host": "localhost",
        "task_path": "/tmp/demo/project/test.yml:3"
    }
}
```


get_fact_cache(*host*)
Get the entire fact cache only if the fact_cache_type is 'jsonfile'

classmethod handle_termination(*pid*, *pidfile=None*, *is_cancel=True*)
Internal method to terminate a subprocess spawned by *pexpect* representing an invocation of runner.

Parameters

- **pid** – the process id of the running the job.
- **pidfile** – the daemon's PID file
- **is_cancel** – flag showing whether this termination is caused by instance's cancel_flag.

host_events(*host*)
Given a host name, this will return all task events executed on that host

kill_container()
Internal method to terminate a container being used for job isolation

run()
Launch the Ansible task configured in self.config (A RunnerConfig object), returns once the invocation is complete

set_fact_cache(*host*, *data*)
Set the entire fact cache data only if the fact_cache_type is 'jsonfile'

property stats
Returns the final high level stats from the Ansible run

Example: {'dark': {}, 'failures': {}, 'skipped': {}, 'ok': {u'localhost': 2}, 'processed': {u'localhost': 1}}

status_callback(*status*)

property stderr
Returns an open file handle to the stderr representing the Ansible run

property stdout
Returns an open file handle to the stdout representing the Ansible run

ansible_runner.runner_config module

ansible_runner.utils module

class ansible_runner.utils.**Bunch**(***kwargs*)
Bases: object

Collect a bunch of variables together in an object. This is a slight modification of Alex Martelli's and Doug Hudgeon's Bunch pattern.

get(*key*)

update(***kwargs*)

class ansible_runner.utils.**OutputEventFilter**(*handle*, *event_callback*, *suppress_ansible_output=False*, *output_json=False*)
Bases: object

File-like object that looks for encoded job events in stdout data.

EVENT_DATA_RE = re.compile('\\x1b\\[K((?:[A-Za-z0-9+/=]+\\x1b\\[\\d+D)+)\\x1b\\[K')

close()

flush()

write(*data*)

ansible_runner.utils.**args2cmdline**(**args*)

ansible_runner.utils.**check_isolation_executable_installed**(*isolation_executable*)
Check that process isolation executable (e.g. podman, docker, bwrap) is installed.

ansible_runner.utils.**cleanup_artifact_dir**(*path*, *num_keep=0*)

ansible_runner.utils.**cleanup_folder**(*folder*)
Deletes folder, returns True or False based on whether a change happened.

ansible_runner.utils.**cli_mounts**()

ansible_runner.utils.**collect_new_events**(*event_path*, *old_events*)
Collect new events for the 'events' generator property

ansible_runner.utils.**dump_artifact**(*obj*, *path*, *filename=None*)
Write the artifact to disk at the specified path

Args:

obj (string): The string object to be dumped to disk in the specified path. The artifact filename will be automatically created

path (string): The full path to the artifacts data directory.

filename (string, optional): The name of file to write the artifact to. If the filename is not provided, then one will be generated.

Returns: string: The full path filename for the artifact that was generated

ansible_runner.utils.**dump_artifacts**(*kwargs*)
Introspect the kwargs and dump objects to disk

ansible_runner.utils.**ensure_str**(*s*, *encoding='utf-8'*, *errors='strict'*)
Copied from six==1.12

Coerce *s* to *str*.

For Python 2:

- *unicode* -> encoded to *str*
- *str* -> *str*

For Python 3:

- *str* -> *str*
- *bytes* -> decoded to *str*

ansible_runner.utils.**get_executable_path**(*name*)

ansible_runner.utils.**isinventory**(*obj*)
Inspects the object and returns if it is an inventory

Args: obj (object): The object to be inspected by this function

Returns: boolean: True if the object is an inventory dict and False if it is not

ansible_runner.utils.**isplaybook**(*obj*)
Inspects the object and returns if it is a playbook

Args: obj (object): The object to be inspected by this function

Returns: boolean: True if the object is a list and False if it is not

ansible_runner.utils.**open_fifo_write**(*path*, *data*)
open_fifo_write opens the fifo named pipe in a new thread. This blocks the thread until an external process (such as ssh-agent) reads data from the pipe.

ansible_runner.utils.**register_for_cleanup**(*folder*)
Provide the path to a folder to make sure it is deleted when execution finishes. The folder need not exist at the time when this is called.

ansible_runner.utils.**sanitize_container_name**(*original_name*)
Docker and podman will only accept certain characters in container names This takes a given name from user-specified values and replaces the invalid characters so it can be used in docker/podman CLI commands

ansible_runner.utils.**santize_json_response**(*data*)
Removes warning message from response message emitted by ansible command line utilities. :param action: The string data to be santizied :type action: str

ansible_runner.utils.**signal_handler**()

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

R

S

U

V

W