img-proof Documentation

Release 7.26.0

SUSE

Apr 05, 2024

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Installation

1.1.1 SUSE package

SLES 15

Ensure you have properly registered SLES then perform the following commands as root for SLES 15:

```
$ SUSEConnect -p sle-module-public-cloud/15.#/x86_64
$ zypper ar https://download.opensuse.org/repositories/Cloud:Tools:CI/SLE_15_SP#/
↪Cloud:Tools:CI.repo
$ zypper refresh
$ zypper in python3-img-proof
```

Replace # with the service pack you are using. Currently support exists for SP2+.

openSUSE Leap 15

Perform the following commands as root for Leap 15:

```
$ zypper ar https://download.opensuse.org/repositories/Cloud:Tools:CI/openSUSE_Leap_15.#/
↪Cloud:Tools:CI.repo
$ zypper refresh
$ zypper in python3-img-proof
```

Currently Leap 15.3+ is supported.

openSUSE Tumbleweed

Perform the following commands as root for Tumbleweed:

```
$ zypper ar https://download.opensuse.org/repositories/Cloud:Tools:CI/openSUSE_
↪Tumbleweed/Cloud:Tools:CI.repo
$ zypper refresh
$ zypper in python3-img-proof
```

Note: An openSUSE and SLES test suite is shipped alongside the SUSE package as python3-img-proof-tests.

1.1.2 SUSE test suite

To install the SLES test suite alongside the package use the following command:

```
$ zypper in python3-img-proof-tests
```

1.1.3 PyPI

```
$ pip install img-proof
```

1.1.4 Development

Install the latest development version from GitHub:

```
$ pip install git+https://github.com/SUSE-Enceladus/img-proof.git
```

1.1.5 Branch

Install a specific branch from GitHub:

```
$ pip install git+https://github.com/SUSE-Enceladus/img-proof.git@{branch/release}
```

See PyPI docs for more information on vcs support.

1.2 Configuration

1.2.1 img-proof Config

The **img-proof** configuration file is ini format ~/.config/img_proof/config. This can be used for any configuration value including cloud framework specific values.

To override the default configuration location the CLI option `-C` or `--config` is available.

The config file can have multiple sections. The default section is [img_proof] and each cloud framework can have its own section such as [{cloud_framework}]. A config file with an [ec2] section may look like the following:

```
[img_proof]
test_dirs = /custom/tests/path/
results_dir = /custom/results/dir/

[ec2]
region = us-west-1
ssh_private_key_file = ~/.ssh/id_rsa
```

There are multiple ways to provide configuration values when using **img-proof**. All options are available via command line and the configuration file. Also, for certain clouds **img-proof** will read cloud specific config files.

All command line options which have a format such as `--ssh-user` can be placed in config with underscores. E.g. `--ssh-user` would be `ssh_user` in the config file.

The precedence for values is as follows:

command line -> cloud config -> img-proof config -> defaults

The command line arguments if provided will be used over all other values.

1.2.2 Azure Config

The Azure provider class has no additional config file. Options should be placed into the **img-proof** config file.

1.2.3 EC2 Config

For testing EC2 instances **img-proof** will look for the ec2utils configuration file located at ~/.ec2utils.conf.

See ec2utils for an example configuration file.

To override the EC2 config location the CLI option, `--cloud-config` is available. In order for **img-proof** to use the ec2imgutils config file the `--account-name` is required.

1.2.4 GCE Config

The GCE cloud class has no additional config file. Options should be placed into the **img-proof** config file.

1.2.5 OCI Config

For testing OCI instances **img-proof** will look for the Oracle configuration file located at ~/.oci/config.

See OCI docs for more info on the Oracle configuration file.

To override the OCI config location the CLI option, `--cloud-config` is available.

The OCI config file is optional as **img-proof** will also look for configuration arguments in the **img-proof** config file and these can be overridden by CLI values.

1.2.6 SSH Config

The SSH cloud class has no additional config file. Options should be placed into the **img-proof** config file.

1.2.7 Aliyun Config

The Aliyun cloud class has no additional config file. Options should be placed into the **img-proof** config file.

1.3 Credentials

1.3.1 Azure

Azure uses service principals for authentication. A service principal (service account) json file is required to use the Azure cloud via file based authentication. It is critical the json file is generated with the endpoint URLs for SDK authentication.

To create the file you will need the Azure CLI.

The following command will generate the necessary json file:

```
$ az ad sp create-for-rbac --sdk-auth --role Contributor --scopes /subscriptions/
↪{subscription_id} --name "{name}" > mycredentials.json
```

Once a json credential file is generated for a service principal it can be used to test images/instances in Azure. The `--service-account-file` option should point to the path to this file.

See Azure docs for more info on creating a service principal json file.

1.3.2 EC2

The EC2 credentials are a `--secret-access-key` and `--access-key-id`. These can be from a root account but it's suggested to use IAM accounts to control role based access.

Once you have generated secret key values these can be configured with the `--secret-access-key` and `--access-key-id` options.

See EC2 docs for more information on setting up IAM accounts.

1.3.3 GCE

GCE uses service accounts for file based authentication. The service account is required to have the following roles:

- Compute Instance Admin (v1) Role (roles/compute.instanceAdmin.v1)
- Service Account User Role (roles/iam.serviceAccountUser)

Additionally the file must be JSON format and contain a private key.

The following steps will create a service account with gcloud and gsutil:

```
$ gcloud --project={project-id} iam service-accounts create {service-account-id}
$ gcloud --project={project-id} iam service-accounts keys create {service_account-id}-
↪key.json --iam-account {service-account-id}@{project-id}.iam.gserviceaccount.com
$ gcloud projects add-iam-policy-binding {project-id} --member serviceAccount:{service-
↪account-id}@{project-id}.iam.gserviceaccount.com --role roles/compute.instanceAdmin.v1
$ gcloud projects add-iam-policy-binding {project-id} --member serviceAccount:{service-
↪account-id}@{project-id}.iam.gserviceaccount.com --role roles/iam.serviceAccountUser
```

The json file generated by the second command "{service_account-id}-key.json" is used for GCE authentication.

```
$ img-proof test gce ... --service-account-file {service_account-id}-key.json
```

Or you can follow the Libcloud docs or Google docs.

Once a json credential file is generated for a service account it can be used to test images/instances in GCE. The `--service-account-file` option should point to the path to this file.

For more information on updating an existing service account:

- Create a new JSON private key: creating-managing-service-account-keys
- Granting roles: granting-roles-to-service-accounts

1.3.4 SSH

Requires no cloud credentials to test instances. SSH user, SSH private key can be placed in SSH section of config. The instance to be tested must be running.

1.3.5 OCI

To use OCI a new compartment, a new user, a new group and an api signing key are required. The user will require access to the compartment via a policy.

The first step is to create an API signing key which will be used by the user for running commands via the OCI SDK. The following doc provides info on creating a key and getting the public key and fingerprint.

Once you have the API signing key you will now create a user, group, compartment and a policy for the new user. The following doc provides all the steps necessary to set these artifacts up. The group will require the following policy for the new compartment:

```
Allow group {group_name} to manage all-resources in compartment {compartment_name}
```

With this setup you can now add the API key to your user. The steps to upload your public key are in the following doc: doc

All of this info can be added as arguments to the OCI config, **img-proof** config or as command line arguments when testing images in OCI. The required options are:

- `--availability-domain`
- `--compartment-id`
- `--oci-user-id`
- `--signing-key-fingerprint`
- `--signing-key-file`
- `--tenancy`

1.3.6 Aliyun

The Aliyun credentials are a `--access-secret` and `--access-key`. These can be from a root account but it's suggested to use RAM accounts to control role based access.

See Aliyun docs for more information on setting up RAM accounts.

CHAPTER

TWO

USAGE

img-proof provides two entry points, a command line interface and a controller class that can be used directly from Python code.

2.1 CLI

The command line interface is written using the Click package. The API documentation can be found at *API*.

2.1.1 Verbosity

As seen in the example the CLI output verbosity can be controlled via options:

--debug
: Display debug level logging to console. Including full stack trace if there is an exception.

--verbose
: (Default) Display logging info to console.

--quiet
: Silence logging information on test run.

2.1.2 Instance Options

The **instance-option** arguments provide a way to enable instance options that will be activated when launching instances. This is a multi-option value. To provide multiple options in a single command split each option into a separate argument. An example for tests in Google:

```
img-proof test gcp ... \
  --instance-option SEV_SNP_CAPABLE \
  --instance-option GVNIC
```

The Google instance options are the guest os feature flags. See https://cloud.google.com/compute/docs/images/create-custom#guest-os-features for more details. As seen above an example for Google looks like:

```
img-proof test gcp ... \
  --instance-option SEV_SNP_CAPABLE
```

The Amazon options are the different options available when running the run instances command. These can be found at https://docs.aws.amazon.com/cli/latest/reference/ec2/run-instances.html.

To provide an instance option for testing in Amazon the type of option and the key/val are provided in the following format: "OptType=key.val". Example usage to enable SEV SNP looks like:

```
img-proof test ec2 ... \
  --instance-option CpuOptions=AmdSevSnp.enabled
```

Where the key is derived from the CLI reference page provided above. In this case the AWS CLI option is –cpu-options which becomes "CpuOptions". "AmdSevSnp" is the key and the value is "enabled".

2.1.3 Cleanup

By default the instance will be terminated if all tests pass. If a test fails the instance will remain running for debugging purposes. This behavior can be configured with the `--cleanup` and `--no-cleanup` flags.

--cleanup
Instance will always be terminated.

--no-cleanup
Instance will always remain running.

2.1.4 ANSI Style

By default the command line output will be colored. To disable color output use the `--no-color` option.

2.1.5 Early Exit

The early exit option will stop the test run on the first failure. `--early-exit` is passed to Pytest as `-x`.

See Pytest docs for more info.

2.1.6 Requirements and external test injection

Using the `--inject` option; packages, archives and files can be injected on the test instance. This also provides the ability to install packages in an existing repository and run commands on the test instance.

The following sections may be provided in a YAML style config file. Each section can be a single item or a list of items. All files are copied and extracted to the default SSH location for the test instance. This is generally the user's home directory.

inject_packages
an rpm path or list of rpm paths which will be copied and installed on the test instance.

inject_archives
an archive or list of archives which will be copied and extracted on the test instance.

inject_files
a file path or list of file paths which will be copied to the test instance.

execute
a command or list of commands to run on the test instance.

install
a package name or list of package names to install from an existing repo on the test instance.

The order of processing for the sections is as follows:

1. inject_packages
2. inject_archives
3. inject_files
4. execute
5. install

Example

testing_injection.yaml.

```
inject_packages: /home/user/test.noarch.rpm
inject_archives: /home/user/test.tar.xz
inject_files: /home/user/test.py
install:
  - python3
  - python3-Django
execute: python test.py
```

```
> img-proof test ... --inject testing_injection.yaml
```

2.2 Code

img-proof can also be imported and invoked directly in Python 3 code through the controller class. It is installed as a Python site package and can be imported as follows:

```
from img_proof.ipa_controller import test_image

status, results = test_image(
    cloud_framework,
    access_key_id,
    ...
    storage_container,
    tests
)
```

See *img_proof.ipa_controller module* for specific methods that can be invoked.

CHAPTER

THREE

EXAMPLES

The following are a few use case examples for **img-proof**. The test suite in use is provided by the Open Build Service *python3-img-proof-test*. For more information on installing the test suite see the *Tests* documentation.

3.1 Launch & Test a new instance in Azure

The first step to testing an image is determining the image ID. The image ID will look different for all cloud frameworks. See the examples below for the three supported clouds:

- Azure: SUSE:SLES:12-SP3:latest
- EC2: ami-0f7c9a39e20a9adea
- GCE: sles-12-sp3-v20180814

To launch and test a new instance of a given image the *–image-id* or *-i* option is required.

The next step is to determine what tests you want to run against the instance. To see what test modules are available there is an *img-proof list* command. You can invoke the command with *-v* option to see a verbose list of all tests within each module.

```
$ img-proof list
test_sles_guestregister
test_sles_haveged
test_sles_hostname
test_sles_install_migration
test_sles_lscpu
test_sles_motd
test_sles_repos
test_sles
```

By default img-proof looks in two directories for test modules:

- ~/img_proof/tests/
- /usr/share/lib/img_proof/tests/

This can be overridden with the *–test-dirs* option. The option is expected to be a comma separated list of absolute test directory paths.

Once you have a set of tests installed and chosen you can run img-proof against an image. For this example we will test the Azure image and only run the base SLES tests:

```
$ img-proof test azure -i suse:sles-15-sp2-byos:gen2:Latest \
  --distro sles test_sles

Starting instance
Testing soft reboot
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_motd.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_license.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_root_pass.py
Testing hard reboot
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_hostname.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_haveged.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_lscpu.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_kernel_version.py
Running test /share/lib/img_proof/tests/SLES/test_sles_multipath_off.py
PASSED tests=10|pass=10|fail=0|error=0
```

You can see that test_sles is a "test description". It's a YAML file that contains an ordered list of test modules to run. To find out more info on tests and test structure see the *Tests* documentation.

By default img-proof will launch the instance with a small instance size. For Azure this is *Standard_B1ms*. Also, if the tests pass the instance will be terminated and resource group will be cleaned up in Azure. This behavior can be modified with the *–cleanup* and *–no-cleanup* options.

There are many options available when running an img-proof test which can be listed via the help command:

```
$ img-proof test --help
```

3.2 Test an existing instance in Azure

If you want to run tests on an existing instance you can provide the *–running-instance-id* or *-r* option. All options and tests that are available for a new instance can be run against an existing one. When testing a running instance the instance will not be terminated if the tests pass. To terminate an already running instance the *–cleanup* option is required.

The running instance ID is different based on cloud provider. It can either be an ID or a name. For Azure the instance "ID" is an instance name.

```
$ img-proof test azure --running-instance-id img-proof-zephl \
  --distro sles test_sles

Testing soft reboot
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_motd.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_license.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_root_pass.py
Testing hard reboot
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_hostname.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_haveged.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_lscpu.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_kernel_version.py
Running test /share/lib/img_proof/tests/SLES/test_sles_multipath_off.py
PASSED tests=10|pass=10|fail=0|error=0
```

After running a test you can view the results using the results command:

```
$ img-proof results show
PASSED tests=10|pass=10|skip=0|fail=0|error=0
```

More information can be displayed by providing the verbose option *-v*:

```
$ img-proof results show -v
FAILED tests=10|pass=10|skip=0|fail=0|error=0

platform: azure
distro: sles
image: 10.0.0.1
timestamp: 20201118151743
log_file: /home/{user}/img_proof/results/azure/suse:sles-15-sp2-byos:gen2:Latest/img-
↪proof-zephl/20201118151743.log
results_file: /home/{user}/img_proof/results/azure/suse:sles-15-sp2-byos:gen2:Latest/img-
↪proof-zephl/20201118151743.results
region: southcentralus
instance: img-proof-zephl

test_soft_reboot PASSED
test_sles_motd::test_sles_motd[paramiko://10.0.0.1] PASSED
test_sles_license::test_sles_license[paramiko://10.0.0.1] PASSED
test_sles_root_pass::test_sles_root_pass[paramiko://10.0.0.1] PASSED
test_hard_reboot PASSED
test_sles_hostname::test_sles_hostname[paramiko://10.0.0.1] PASSED
test_sles_haveged::test_sles_haveged[paramiko://10.0.0.1] PASSED
test_sles_lscpu::test_sles_lscpu[paramiko://10.0.0.1] PASSED
test_sles_kernel_version::test_sles_kernel_version[paramiko://10.0.0.1] PASSED
test_sles_multipath_off::test_sles_multipath_off[paramiko://10.0.0.1] PASSED
```

3.3 Testing with SSH only

If you have a running instance that has an accessible IP address you can run img-proof tests without the use of a cloud provider framework. This means the instance must have an SSH key pair setup. Without cloud framework credentials the instance cannot be terminated after tests and must be running. There is also no way to do a framework reboot test.

Instead of providing the image *–image-id* or instance *–running-instance-id* you are required to provide an IP address *–ip-address*.

```
$ img-proof test ssh --ip-address 10.0.0.1 \
  --distro sles test_sles

Testing soft reboot
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_motd.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_license.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_root_pass.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_hostname.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_haveged.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_lscpu.py
Running test /usr/share/lib/img_proof/tests/SLES/test_sles_kernel_version.py
Running test /share/lib/img_proof/tests/SLES/test_sles_multipath_off.py
PASSED tests=10|pass=10|fail=0|error=0
```

CHAPTER

FOUR

TESTS

Tests are developed using the Testinfra package. The package extends Pytest and provides a framework for writing Python tests to verify the actual state of systems.

4.1 SLES Test Suite

There is a suite of tests for SLES and openSUSE_Leap. It can be found in the GitHub repository.

They are also packaged in the Open Build Service for openSUSE:

```
$ zypper ar http://download.opensuse.org/repositories/Cloud:/Tools/<distribution>
$ zypper refresh
$ zypper in python3-img-proof-tests
```

4.2 Test directories

The default locations for test files are locally in ~/img_proof/tests/ and centralized in /usr/share/img_proof/tests. These locations can be overridden in the config and/or command line arguments as *–test-dirs*.

4.3 Test organization

Tests can be organized in a directory structure:

```
~/img_proof/tests/:
  conftest.py
  test_image.py
  openSUSE:
    test_leap.py
    EC2:
      test_leap_ec2.py
    GCE:
    ...
  SLES:
    test_sles.py
    test_sles_sap.py
    EC2:
```

(continues on next page)

(continued from previous page)

```
    test_sles_ec2.py
  ...
```

Additionally, test descriptions in YAML format can be used to organize tests:

test_leap_423.yaml.

```
tests:
  - test_image
  - test_leap
```

Adding tests to command line args you simply drop the extension:

```
$ img-proof test ... test_leap_423
```

This means there cannot be a name overlap with test files and/or test descriptions.

Test descriptions can also include other descriptions:

test_leap_423.yaml.

```
tests:
  - test_image
  - test_leap
include:
  - test_another_description
```

4.4 Test invocation

To invoke a specific test the Pytest conventions can be used:

```
$ img-proof test ... test_leap_ec2::test-services-running-enabled
```

To run only one parameterized test append ids and use [ID]:

```
@pytest.mark.parametrize("name", [
    ("cloud-init"),
    ("amazon-ssm-agent"),
], ids=['ci', 'ssm'])
def test_leap_ec2():
  ...
```

```
$ img-proof test ... test_leap_ec2::test-services-running-enabled[ssm]
```

4.4.1 Failures

By default all tests will run even with failure. Using the `--early-exit` option will halt test invocation at first failure.

Incremental test classes can be used to cause all subsequent tests to fail if the prev fails. To prevent expected failures.

4.5 Custom Test Modules

Modules are provided for checking standard things such as packages, services, files, etc.

Modules can be easily written or extended using Pytest fixtures. Any custom modules reside in the conftest.py file inside the test directory:

```
import pytest

@pytest.fixture()
def Echo(Command):
    def f(arg):
        return Command.check-output("echo %s", arg)
    return f


@pytest.fixture()
def CheckRepo(File):
    def f(repo, name):
        repo = File('/etc/zypp/repos.d/' + repo + '.repo')
        tests = [repo.exists,
                 repo.contains('enabled=1'),
                 repo.contains('name=%s' % name)]
        return all(tests)
    return f
```

To use a Pytest fixture in a test it is simply included as an arg by name. For example the test below includes the CheckRepo fixture which is then used to validate a specific repo exists and is enabled.

```
def test_some_repo(CheckRepo):
    assert CheckRepo('openSUSE-20200325-0', 'openSUSE-20200325-0')
```

In order for Pytest to discover the fixture used in a test module it is required to be in a conftest.py file in the same directory or a parent directory.

4.6 Useful Links

For more info on writing tests see the Testinfra and Pytest documentation.

CHAPTER

FIVE

API

CHAPTER

SIX

IMG_PROOF PACKAGE

6.1 Subpackages

6.1.1 img_proof.scripts package

Submodules

img_proof.scripts.cli module

img_proof.scripts.cli_utils module

Module contents

6.2 Submodules

6.2.1 img_proof.collect_items module

6.2.2 img_proof.ipa_azure module

6.2.3 img_proof.ipa_cloud module

6.2.4 img_proof.ipa_constants module

6.2.5 img_proof.ipa_controller module

6.2.6 img_proof.ipa_distro module

6.2.7 img_proof.ipa_ec2 module

6.2.8 img_proof.ipa_exceptions module

6.2.9 img_proof.ipa_gce module

6.2.10 img_proof.ipa_opensuse_leap module

6.2.11 img_proof.ipa_sles module

6.2.12 img_proof.ipa_ssh module

6.2.13 img_proof.ipa_utils module

6.2.14 img_proof.results_plugin module

6.3 Module contents

img-proof (IPA) provides a command line utility to test images in the Public Cloud (AWS, Azure, GCE, OCI, Aliyun, etc.).

CHAPTER

SEVEN

ABOUT

With **img-proof** you can now test custom images in a cloud framework agnostic way with one tool and one API. **img-proof** supports the Fedora, openSUSE, RHEL, and SLES distributions. It also supports the three largest cloud frameworks (AWS, Azure and GCE). However, it is intended to be distribution agnostic and framework transparent so both are easily extensible.

CHAPTER

EIGHT

OVERVIEW

The goal of **img-proof** is to provide a unit test framework that can be used to verify the actual state of custom public cloud images. To do this it leverages two packages.

8.1 pytest

Tests are written using the pytest framework.

The pytest framework makes it easy to write small tests, yet scales to support complex functional testing for applications and libraries.

8.2 Testinfra

Testinfra is a plugin for pytest which provides connection backends and test modules such as File, Group, User, Package, Service, etc.

With Testinfra you can write unit tests in Python to test actual state of your servers configured by management tools like Salt, Ansible, Puppet, Chef and so on.

8.3 img-proof

img-proof leverages Testinfra as a unit test framework. It also provides distribution classes and cloud framework classes.

8.3.1 Distribution Classes

Classes can be used to provide distribution specific testing. This includes tests such as soft reboot (e.g. "shutdown -r now") and update.

The current supported distributions are:

- Fedora (fedora)
- openSUSE_Leap (opensuse_leap)
- RHEL (rhel)
- SLES (sles)

In addition to soft reboot, refresh and update there is a built in test for hard reboot (framework reboot).

These tests are "synchronization points". The tests are built into each distribution as the commands can be slightly different.

By default pytest does not guarantee the order of tests. However, there are cases where this is ideal when testing images. For example you can run an update then do a reboot to ensure an instance starts properly.

A test suite such as ['test_soft_reboot', 'test_sles', 'test_update', 'test_hard_reboot', 'test_sles_ec2'] will be broken into five pytest invocations:

Soft reboot, test_sles, update, hard reboot, test_sles_ec2

The order of tests is guaranteed and the results will be aggregated to determine the status of a test run. If a test fails it will be re-run up to the number of retries. The default is three times and this can be configured with the **–retry-count** option.

8.3.2 Cloud Framework Classes

The cloud framework classes contain methods necessary to interact with instances/images in a given cloud framework.

Some of the required methods for cloud framework classes include:

- Launch instances
- Terminate instances
- Get instance status
- Get instance info
- Stop/Start/Reboot instances

The implementations are dependent on each CSP API and require the CSP credentials to perform the necessary operations.

The current supported CSPs are:

- Azure
- EC2
- GCE
- OCI
- SSH
- Aliyun

The SSH class is generic and can be used for any accessible instance that is running. There are no credentials required except the instance needs the proper SSH User and SSH Key configured for access.

The SSH class cannot be used to test hard reboot (framework reboot) or to launch/start/stop/terminate instances.

CHAPTER

NINE

CONTRIBUTING

Contributions to **img-proof** are welcome and encouraged. See CONTRIBUTING for info on getting started.

CHAPTER

TEN

ISSUES/ENHANCEMENTS

Please submit issues and requests to Github.

CHAPTER

ELEVEN

LICENSE