Ironic Python Agent Builder Documentation

Release 4.1.1.dev3

OpenStack Developers

Jan 18, 2024

CONTENTS

Tools and scripts to build a deployment, cleaning or inspection ramdisk based on Ironic Python Agent.

- Free software: Apache license
- Documentation: https://docs.openstack.org/ironic-python-agent-builder
- Source: https://opendev.org/openstack/ironic-python-agent-builder
- Bugs: https://storyboard.openstack.org/#!/project/948
- Release Notes: https://docs.openstack.org/releasenotes/ironic-python-agent-builder/

Contents:

CHAPTER

ONE

INSTALLING IRONIC PYTHON AGENT BUILDER

Download the `ironic-python-agent-builder` archive from tarballs.openstack.org, install it from your distributions repositories or use pip:

```
pip install --user diskimage-builder ironic-python-agent-builder
```

In RDO, the package is available since Train under a slightly different name:

```
sudo yum install -y openstack-ironic-python-agent-builder
```

CHAPTER

TWO

ADMINISTRATORS GUIDE

This guide describes how to build and use an Ironic Python Agent-based image using the builders provided in the **ironic-python-agent-builder** project.

2.1 diskimage-builder images

Images built using diskimage-builder are recommended for production use on real hardware. The recommended distributions are:

- CentOS Stream 8
- Debian Bullseye (`debian-minimal` element)

The following should work but receive only limited testing and support:

- CentOS 7 (using Python 3.6)
- openSUSE Leap 15.1
- Ubuntu 20.04 Focal

2.1.1 Building

with the helper script

To build an image using `ironic-python-agent-builder`, run:

```
ironic-python-agent-builder <distribution, e.g. ubuntu>
```

You can add other diskimage-builder elements via the `-e` flag:

```
ironic-python-agent-builder -e <extra-element> --release 8-stream centos
```

You can specify the base name of the target images:

```
ironic-python-agent-builder -o my-ipa --release 8-stream centos
```

You can specify the arch of the target image by setting `ARCH` environment variable (default is amd64):

```
export ARCH=aarch64
ironic-python-agent-builder -o my-ipa --release 8-stream centos
```

with diskimage-builder

You can also use diskimage-builder directly. First you need to set the `ELEMENTS_PATH` variable to the correct location:

- If installed with `pip install --user`, use:

```
export ELEMENTS_PATH=$HOME/.local/share/ironic-python-agent-builder/dib
```

- On Fedora/CentOS/RHEL (installed via `sudo pip install` or from packages):

```
export ELEMENTS_PATH=/usr/share/ironic-python-agent-builder/dib
```

- On Debian and its derivatives, if installed with `sudo pip install`:

```
export ELEMENTS_PATH=/usr/local/share/ironic-python-agent-builder/dib
```

Now you can build an image adding the `ironic-python-agent-ramdisk` element, for example:

```
export DIB_RELEASE=8-stream
disk-image-create -o ironic-python-agent \
    ironic-python-agent-ramdisk centos
```

To use a specific branch of ironic-python-agent, use:

```
export DIB_REPOREF_ironic_python_agent=origin/stable/queens
export DIB_REPOREF_requirements=origin/stable/queens
```

To build ironic-lib from source, do:

```
export IRONIC_LIB_FROM_SOURCE=true
# Optionally:
#export DIB_REPOREF_ironic_lib=<branch>
```

To build image for architectures other than amd64, you can either set the `ARCH` environment variable or use `-a` to specify the target architecture:

```
disk-image-create -a arm64 -o ironic-python-agent \
    ironic-python-agent-ramdisk fedora
```

ISO Images

Additionally, the IPA ramdisk can be packaged inside of an ISO for use with some virtual media drivers. Use the `iso-image-create` script in `ironic-python-agent-builder` repository, passing it the initrd and the kernel, for example:

```
./tools/iso-image-create -o /path/to/output.iso -i /path/to/ipa.initrd -k /
↪path/to/ipa.kernel
```

This is a generic tool that can be used to combine any initrd and kernel into a suitable ISO for booting, and so should work against any IPA ramdisk.

2.1.2 Advanced options

Disabling rescue

By default rescue mode is enabled in the images. Since it allows to set root password on the ramdisk by anyone on the network, you may disable it if the rescue feature is not supported. Set the following before building the image:

```
export DIB_IPA_ENABLE_RESCUE=false
```

SSH access

SSH access can be added to DIB built IPA images with the dynamic-login or the devuser element.

The *dynamic-login* element allows the operator to inject an SSH key at boot time via the kernel command line parameters:

- Add `sshkey="ssh-rsa <your public key here>"` to `pxe_append_params` setting in the `ironic.conf` file.

  Warning: Quotation marks around the public key are important!

- Restart the ironic-conductor.

Note: This element is added to the published images by default.

The *devuser* element allows creating a user at build time, for example:

```
export DIB_DEV_USER_USERNAME=username
export DIB_DEV_USER_PWDLESS_SUDO=yes
export DIB_DEV_USER_AUTHORIZED_KEYS=$HOME/.ssh/id_rsa.pub
disk-image-create debian ironic-python-agent-ramdisk devuser
```

Consistent Network Interface Naming

Base cloud images normally disable consistent network interface naming by inserting an empty udev rule. Include `stable-interface-names` element if you want to have consistent network interface naming whenever it is required for instance image or deploy image.

```
ironic-python-agent-builder -e stable-interface-names --release 8-stream␣
↪centos
```

Firmware Removal

By default the element removes some firmware blobs to reduce the image size. The list can be found below this paragraph. The majority of these firmware images are used by SoCs, WI-FI chips, some GPUs and Smartnics which are unlikely to be encountered. If you want to override this, change the `IPA_REMOVE_FIRMWARE` environment variable to a comma-separated list of directories or files under `/usr/lib/firmware`. Set it to an empty string to disable firmware removal.

Fimrware removed:

- `amdgpu`
- `netronome`
- `qcom`
- `ti-communication`
- `ti-keystone`
- `ueagle-atm`
- `rsi`
- `mrvl`
- `brcm`
- `mediatek`
- `ath10k`
- `rtlwifi`

2.1.3 Available Elements

Ironic Python Agent (IPA) Extra Hardware

This element adds the hardware python package to the Ironic Python Agent (IPA) ramdisk. It also installs several package dependencies of the `hardware` module.

The `hardware` package provides improves hardware introspection capabilities and supports benchmarking. This functionality may be enabled by adding the `extra-hardware` collector in the `[DEFAULT] inspection_collectors` option or the `ipa-inspection-collectors` kernel command line argument.

The following environment variables may be set to configure the element when doing a source-based installation:

- `DIB_IPA_HARDWARE_PACKAGE` the full `hardware` Python package descriptor to use. If unset, `DIB_IPA_HARDWARE_VERSION` will be used.
- `DIB_IPA_HARDWARE_VERSION` the version of the `hardware` package to install when `DIB_IPA_HARDWARE_PACKAGE` is unset. If unset, the latest version will be installed.

ironic-python-agent-ramdisk

Builds a ramdisk with ironic-python-agent.

More information can be found at: https://docs.openstack.org/ironic-python-agent/latest/

Beyond installing the ironic-python-agent, this element does the following:

- Installs the `dhcp-all-interfaces` so the node, upon booting, attempts to obtain an IP address on all available network interfaces.
- Disables the `iptables` service on SysV and systemd based systems.
- Disables the `ufw` service on Upstart based systems.
- Installs packages required for the operation of the ironic-python-agent:: `qemu-utils` `parted` `hdparm` `util-linux`
- When installing from source, `python-dev` and `gcc` are also installed in order to support source based installation of ironic-python-agent and its dependencies.
- Install the certificate if any, which is set to the environment variable `DIB_IPA_CERT` for validating the authenticity by ironic-python-agent. The certificate can be self-signed certificate or CA certificate.
- Compresses initramfs with command specified in environment variable `DIB_IPA_COMPRESS_CMD`, which is gzip by default. This command should listen for raw data from stdin and write compressed data to stdout. Command can be with arguments.
- Configures rescue mode if `DIB_IPA_ENABLE_RESCUE` is not set to `false`.

This element outputs two files:

- `$IMAGE-NAME.initramfs`: The deploy ramdisk file containing the ironic-python-agent (IPA) service.
- `$IMAGE-NAME.kernel`: The kernel binary file.

Note: The package based install currently only enables the service when using the systemd init system. This can easily be changed if there is an agent package which includes upstart or sysv packaging.

Note: Using the ramdisk will require at least 1.5GB of ram

---

ironic-python-agent-tls

Adds TLS support to ironic-python-agent-ramdisk.

By default this element will enable TLS API support in IPA with a self-signed certificate and key created at build time.

Optionally, you can provide your own SSL certifiate and key, and optionally CA, via the following environment variables. They should be set to an accessible path on the build systems filesystem. If set, they will be copied into the built ramdisk, and IPA will be configured to use them.

The environment variables are:

- DIB_IPA_CERT_FILE should point to the TLS certificate for ramdisk use.
- DIB_IPA_KEY_FILE should point to the private key matching DIB_IPA_CERT_FILE.

You can configure the generated certificate with the following environment variables:

- DIB_IPA_CERT_HOSTNAME the CN for the generated certificate. Defaults to ipa-ramdisk.example.com.
- DIB_IPA_CERT_EXPIRATION expiration, in days, for the certificate. Defaults to 1095 (three years).

Note that the certificates generated by this element are self-signed, and any nodes using them will need to set agent_verify_ca=False in driver_info.

This element can also configure client certificate validation in IPA. If you wish to validate client certificates, set DIB_IPA_CA_FILE to a CA file you wish IPA client connections to be validated against. This CA file will be copied into the built ramdisk, and IPA will be configured to use it.

2.2 TinyIPA images

TinyIPA is an Ironic Python Agent image based on TinyCoreLinux. It is very lightweight and thus very suitable for CI use. It may lack necessary drivers and the build process uses insecure communication, thus these images are not recommended for production usage.

2.2.1 Requirements

You need to have a git clone of **ironic-python-agent-builder**:

```
git clone https://opendev.org/openstack/ironic-python-agent-builder
cd ironic-python-agent-builder/tinyipa
```

Then you need to install some utilities. For the main build script:

- wget
- pip
- unzip
- sudo
- awk
- mksquashfs

For building an ISO youll also need:

- mkisofs, genisoimage, or xorrisofs

2.2.2 Building

Building ramdisk

To create a new ramdisk, run:

```
make
```

or:

```
./build-tinyipa.sh && ./finalise-tinyipa.sh
```

This will create two new files once completed:

- `tinyipa.vmlinuz` - the kernel image
- `tinyipa.gz` - the initramfs image

Upload them to the Image service or another location where you want them to be hosted (an HTTP or FILE location in case of standalone ironic).

Building ISO

Once youve built tinyIPA it is possible to pack it into an ISO if required. To create a bootable ISO, run:

```
make iso
```

or:

```
./build-iso.sh
```

This will create one new file once completed:

- `tinyipa.iso`

Cleaning up

To clean up the whole build environment, run:

```
make clean
```

For cleaning up just the iso or just the ramdisk build:

```
make clean_iso
```

or:

```
make clean_build
```

2.2.3 Advanced options

Enabling/disabling SSH access to the ramdisk

By default tinyIPA will be built with OpenSSH server installed but no public SSH keys authorized to access it.

If you want to enable SSH access to the image, set `AUTHORIZE_SSH` variable in your shell before building tinyIPA:

```
export AUTHORIZE_SSH=true
```

By default it will use public RSA or DSA keys of the user running the build. To provide a different public SSH key, export path to it in your shell before building tinyIPA:

```
export SSH_PUBLIC_KEY=<full-path-to-public-key>
```

If you want to disable SSH altogether, set `INSTALL_SSH` variable in your shell to `false` before building tinyIPA:

```
export INSTALL_SSH=false
```

If you want to change the SSH access of a previously built tinyIPA image, use the make target `addssh`:

```
make addssh
```

This command will either use a local image specified by the `TINYIPA_RAMDISK_FILE` environment variable or download the version specified by the `BRANCH_PATH` environment variable (e.g. `master` or `stable-queens`) from tarballs.openstack.org. It will install and configure OpenSSH if needed and add public SSH keys for the user named `tc` using either the same `SSH_PUBLIC_KEY` shell variable or the public keys of the local user.

Enabling biosdevname in the ramdisk

If you want to collect BIOS given names of NICs in the inventory, set `TINYIPA_REQUIRE_BIOSDEVNAME` variable in your shell before building tinyIPA:

```
export TINYIPA_REQUIRE_BIOSDEVNAME=true
```

Using ironic-lib from source

ironic-lib contains important parts of the provisioning logic. If you would like to build an IPA image with your local checkout of ironic-lib, export the following variable:

```
export IRONIC_LIB_SOURCE=/absolute/path/to/ironic-lib/checkout
```

CHAPTER

THREE

CONTRIBUTOR DOCUMENTATION

If you would like to contribute to the development of OpenStack, you must follow the steps in this page:

> http://docs.openstack.org/infra/manual/developers.html

If you already have a good understanding of how the system works and your OpenStack accounts are set up, you can skip to the development workflow section of this documentation to learn how changes to OpenStack should be submitted for review via the Gerrit tool:

> http://docs.openstack.org/infra/manual/developers.html#development-workflow

Pull requests submitted through GitHub will be ignored.

Bugs should be filed on Storyboard, not GitHub:

> https://storyboard.openstack.org/#!/project/948