ODL Integration/Packaging

Release master

Daniel Farrell

Mar 19, 2020

TABLE OF CONTENTS

This guide provides details on how Packaging and Deployment of OpenDaylight is supported. Including packaging (RPMs), configuration management tools (Ansible, Puppet) and pre-built images (containers, Vagrant base boxes).

Contents:

CHAPTER

ONE

ANSIBLE ROLE

Ansible role for the OpenDaylight SDN controller.

1.1 Installing Ansible-OpenDaylight

The Ansible Galaxy tool that ships with Ansible can be used to install ansible-opendaylight.

To install the latest version of Ansible on Red Hat-based OSs:

```
$ sudo yum install -y ansible
```

To install the latest version of Ansible on Debian-based OSs:

```
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install -y ansible
```

After you install **ansible-galaxy**, install ansible-opendaylight:

```
$ ansible-galaxy install git+ssh://<LF ID>@git.opendaylight.org:29418/integration/
↪packaging/ansible-opendaylight.git
```

The OpenDaylight Ansible role doesn't depend on any other Ansible roles.

1.2 Role Variables

1.2.1 Karaf Features

To set extra Karaf features to be installed at OpenDaylight start time, pass them in a list to the **extra_features** variable. The extra features you pass will typically be driven by the requirements of your use case.

OpenDaylight normally installs a default set of Karaf features at boot. They are recommended, so the ODL Ansible role defaults to installing them. This can be customized by overriding the **default_features** variable. You shouldn't normally need to do so.

1.2.2 REST API Port

To change OpenDaylight's northbound REST API port from the default of 8181, use the **odl_rest_port** variable.

For example, in an Openstack deployment, the Swift project uses 8181 and conflicts with OpenDaylight.

The Ansible role will handle opening this port in FirewallD if it's active.

1.2.3 Install Method

OpenDaylight supports RPM and deb-based installs, either from a repository or directly from a URL to a package. Use the **instal_method** var to configure which deployment scenario is used.

Valid options: rpm_repo: Install ODL using its Yum repo config rpm_path: Install ODL RPM from a local path or remote URL dep_repo: Install ODL using a Debian repository deb_path: Install ODL .deb from a local path or remote URL

1.3 Installing OpenDaylight

To install OpenDaylight via ansible-opendaylight, use **ansible-playbook**.

```
sudo ansible-playbook -i "localhost," -c local examples/<playbook>
```

Example playbooks are provided for various deployments.

1.4 Example Playbooks

The playbook below would install and configure OpenDaylight using all defaults.

```
---
- hosts: example_host
    sudo: yes
roles:
    - opendaylight
```

To override default settings, pass variables to the **opendaylight** role.

```
---
- hosts: all
  sudo: yes
  roles:
    - role: opendaylight
      extra_features: ['odl-netvirt-openstack']
```

Results in:

```
opendaylight-user@root>feature:list | grep odl-netvirt-openstack
odl-netvirt-openstack | <odl-release> | x | odl-netvirt-<odl-release> | OpenDaylight␣
↪:: NetVirt :: OpenStack
```

1.5 License

OpenDaylight is Open Source. Contributions encouraged!

1.6 Author Information

The OpenDaylight Integration/Packaging project maintains this role.

CHAPTER

TWO

AUTORELEASE BUILDS

OpenDaylight's primary build pipeline is called "autorelease". It is managed by the RelEng/Autorelease project, and primarily takes the form of Autorelease's Jenkins jobs.

Autorelease builds every project from source. Artifact versions are rewritten from the -SNAPSHOT suffixes in version control to release versions, like -Carbon-SR1 or -Nitrogen. This contrasts with distribution jobs, which build only a few projects from source and use -SNAPSHOT artifact versions. This makes autorelesae builds slow, but identical to actual releases, whereas distribution builds are fast but slightly less similar to official releases.

2.1 Daily Releases

Autorelease's Jenkins jobs run daily for every active branch, including master.

- Carbon autorelease job
- Nitrogen autorelease job
- Oxygen autorelease job
- Fluorine autorelease job

Each of those jobs, when the build is successful, produces build artifacts that include an OpenDaylight distribution.

To download the distribution

1. Pick an autorelease job that completed successfully (yellow or blue dot)
2. Access its logged console output

   Logs are hosted on logs.opendaylight.org, at URLs like *https://logs.opendaylight.org/releng/vex-yul-odl-jenkins-1/autorelease-release-<stream>/ <build_number>/*, where stream could be "Fluorine" build_number "52".

   There will be a link at the top of build's Jenkins page.
3. Open *deploy-staged-repository.log.gz* in browser

   Search for "staging repository with ID" to find the repository ID, which will be of the form "autorelease-1432".
4. Navigate to OpenDaylight's Nexus and find the staging repository with the same name
5. Drill down into one of these directories to find the build artifacts:
   - Carbon or older: org/opendaylight/integration/distribution-karaf/
   - Nitrogen or newer: org/opendaylight/integration/karaf/

Note: Autorelease build artifacts are persevered for 60 days.

Autorelease jobs trigger OpenDaylight's distribution tests when they complete.

To see the test results

1. Go to integration-distribution-test-<branch> job's Jenkins page
2. Find the job that started after the autorelease in question finished
3. Open it and explore the subprojects section for test results of all the jobs triggered.

   For example, in case of Nitrogen, you can find the list and the results of jobs triggered here.

The latest successful autorelease builds can also be easily found in Nexus at staging/org/opendaylight/integration/distribution-karaf/. Look for 0.5.4-Boron-SR4, 0.6.1-Carbon-SR1, 0.7.0-Nitrogen or similar staging repositories. Note that the artifacts in these repositories are not static - they are replaced each time new artifacts are generated. Use the "autorelease-XXXX" repositories described above for semi-persistent URLs.

2.2 Official Releases

As a part of the OpenDaylight community's efforts to move towards Continuous Delivery, there is very little mechanical difference between the automated daily releases documented above and official releases. The same autorelease job runs, builds artifacts and kicks off distribution tests against them. When doing official releases, the OpenDaylight community iterates through those builds (calling them Release Candidate 1, RC2, ...) until no blocking bugs are found. The OpenDaylight Technical Steering Committee then hears feedback from the Release Engineering and Integration/Test teams, and if all's well blesses the build as an official release. The build's Nexus staging repo is then promoted to a release repo and publicized (example: opendaylight.release/org /opendaylight/integration/distribution-karaf/0.6.0-Carbon). Official releases are persevered forever.

For more information about OpenDaylight releases, including timelines, see the Release Plans.

CHAPTER

THREE

CONFIGURATION MANAGEMENT

The Configuration Management Layer of the packaging and delivery stack provided by upstream OpenDaylight installs OpenDaylight via the Packaging Layer and then does any additional configuration required by the particular deployment's requirements. Examples include setting Karaf features to install at boot, remapping OpenDaylight ports, opening OpenDaylight ports in firewalld and managing OpenDaylight's systemd service. As additional knobs are required to configure deployments, upstream support should be added here.

CHAPTER

FOUR

DEBS

The build.py helper script is used for building OpenDaylight .debs. It can build a set of .debs based on provided version arguments. The dynamic aspects of builds, such as ODL and deb version info, have all been extracted to single YAML configuration file.

The variables available for configuration and instructions on how to install are documented here.

4.1 Build Deb Job

The Jenkins build_deb job builds the .deb package described by the given build description, using build.py inside the deb directory.

CHAPTER

FIVE

DISTRIBUTION JOB BUILDS

Unlike autorelease builds, which build every project from source, distribution jobs only build a few Karaf features. The other artifacts are pulled pre-built from OpenDaylight's Nexus repository and packaged into the Karaf distribution. This makes them much quicker (minutes instead of ~4 hours).

The other major difference between autorelease and distribution job builds is that distribution jobs use the -SNAPSHOT artifact version suffixes that are actually stored in version control, whereas autorelease builds rewrite versions to use the suffix for the next release, like -Carbon-SR1 or -Nitrogen. Because of this, distribution builds are sometimes called "snapshot builds".

For each active branch, builds created by distribution jobs can be found in the subdirectories at opendaylight.snapshot/org/opendaylight/integration /distribution-karaf/. Each build artifact is versioned with a timestamp and unique, incrementing build number.

5.1 Distribution Builds Triggered by Merge Jobs

Distribution job builds are typically kicked off when a patch is merged into a project. Projects define <*project*>-merge-<branch> Jenkins jobs, which are kicked off by Gerrit merge event. To find the merge job for a Gerrit, look for comments from the jenkins-releng user like "Build Started https://jenkins.opendaylight.org/releng/job/netvirt-merge-boron/216/".

Alternatively, browse a project's Jenkins tab and look at the recent runs. For example, go to https://jenkins.opendaylight.org/releng/, select Merge-Carbon and you'll find the list of all project merge jobs in the format <*project*>-merge-carbon. Click any to view the recent build job details and logs.

5.2 Custom Distributions

Distributions can be built with an additional set of unmerged patches. The integration-multipatch-test-<branch> jobs allow users to specify a set of patches to cherry-pick onto a project's source code before building. This is very useful for testing complex changes that impact multiple projects.

To build a custom distribution that includes a set of unmerged patches, first make sure you have permission to trigger Jenkins jobs. Send an email to the OpenDaylight Helpdesk (helpdesk@opendaylight.org) to request access. Be sure to include your Linux Foundation user ID in the request.

Once you can trigger Jenkins jobs, navigate to the Jenkins web UI for the multipatch-test job of the branch you're interested in. Make sure you're logged in, then click on the "Build with Parameters" link in the sidebar. The only parameter that requires configuration is PATCHES_TO_BUILD. This is a CSV list of patches in project[=checkout][:cherry-pick]* format. For each given project, the job will checkout 0 or 1 specified patches, then cherry-pick 0 or more additional patches on top of that checkout. If no checkout is specified, cherry-picks will be done on top of the tip of the branch of the multipatch-test job you're using.

For example, to build with a single unmerged patch from NetVirt:

    netvirt:59/50259/47

Because of the colon, this would cherry-pick the change on top of the tip of the multipatch-test job branch.

To build with the same NetVirt patch, but by directly checking it out, use an equals sign.

    netvirt=59/50259/47

This will be the same thing if the patch has recently been rebased on top of the tip of the branch, but may be different if the patch is based on a different set of patches.

To build with checked-out patches from Genius and NetVirt:

    genius=32/53632/9,netvirt=59/50259/47

To checkout a patch from controller, then cherry-pick another on top of it:

    controller=61/29761/5:45/29645/6

The numbers in the changeset are the Gerrit change ID of the patch (middle number) and the patchset of the Gerrit (last number). The first number is just the last two digits of the Gerrit change ID (I'm not sure why this is necessary). I belive it's required that patches be listed in the order the projects are built (NetVirt depends on Genius, so Genius is listed first).

For the definitive explination of how the multipatch job works, see the JJB source that defines it.

CHAPTER

SIX

PACKAGES

Builds can be packaged as RPMs or .debs. To provide inputs into OpenDaylight's Continious Delivery pipelines to downstream projects, many builds are automatically packaged. Every succesful autorelease build is packaged as an RPM. Every day, the latest distribution snapshot build is packaged as an RPM. This keeps new artifacts flowing even if some projects are breaking autorelease. Custom packages can be built from custom distributions, for example with yet-to-be merged patches that need system testing.

6.1 RPMs

OpenDaylight has a mature RPM Continuous Delivery pipeline. Every autorelease build is automatically packaged as an RPM, and even if autorelease is broken a daily job builds the latest distribution snapshot build into an RPM.

RPMs can be passed to test jobs that install them, start OpenDaylight with its systemd service, connect to the Karaf shell and verify basic functionality.

RPMs are hosted on the CentOS Community Build system repositories. Some repos are updated very frequently with the latest builds, while others are permanent homes of official releases.

Developers can build custom RPMs with pre-merge patches for testing by first creating a custom distribution with the integration-multipatch-test job and then feeding the resulting artifact to the packaging-build-rpm job.

6.1.1 Build Jobs

OpenDaylight Integration/Packaging has added support for many variations of fully automated RPM builds.

6.1.1.1 packaging-build-rpm

The packaging-build-rpm job is the primary way to build an RPM from an OpenDaylight distribution (built by autorelease or the *snapshot distribution <distribution-job-builds.html>* job). It accepts a set of parameters that can be used to configure the build and passes them to the RPM build logic in Integration/Packaging's repo. The job produces both a noarch RPM and source RPM. The noarch RPM can be passed to test jobs for validation. The source RPM can be downloaded to a system with the required credentials and then pushed to the CentOS Community Build system to be built into a noarch RPM on their servers and hosted in their repos.

The RPM and SRPM artifacts of the job are handled differently depending on the Jenkins silo the job is executing in.

When running in production (releng silo), artifacts are hosted on Nexus. There are RPM repos for each active branch (oxygen-devel, fluorine-devel, neon-devel). New builds are automatically added to the appropriate devel for their branch.

When running in the sandbox, artifacts are thrown away by default. To keep an artifact for further testing, either:

- Set the DEPLOY_TO_REPO parameter to opendaylight-epel-7-x86_64-devel. This is a scratch repo that sandbox packaging jobs have permission to push to. Packages will land in the scratch repo on Nexus.
- Add a path regex that matches it to the Archive Artifacts param of the job (*ARCHIVE_ARTIFACTS=/home/jenkins/rpmbuild/RPMS/ noarch/opendaylight*.rpm*). The files matched will be stored in OpenDaylight's log archive along with the other job logs.

6.1.1.2 packaging-build-rpm-snap

The packaging-build-rpm-snap job packages the most recent *snapshot distribution <distribution-job-builds.html>* build from a given branch as an RPM. This could be used by a developer to test code that was just merged, but which has not been included in an autorelease build yet. The job is also triggered daily, to ensure that OpenDaylight's Continuous Delivery pipeline is fed new builds even if autorelease is broken.

6.1.2 Test Jobs

6.1.2.1 packaging-test-rpm

The packaging-test-rpm job accepts a link to an RPM and validates it. It installs the package with the system's package manager, starts OpenDaylight's systemd service, verifies that it's reported as active, connects to the Karaf shell and checks that some key bundles are present.

6.1.3 Repositories

6.1.3.1 OpenDaylight Nexus

Packages resulting from build jobs running on OpenDaylight's infrastructure are automatically hosted on OpenDaylight's Nexus repositories.

6.1.3.1.1 Continious Delivery Repositories

OpenDaylight provides fully-automated Continuous Delivery pipelines for RPMs.

Every RPM built in the production RelEng Jenkins silo is pushed to the devel repo appropriate for its branch. Builds are triggered for every successful autorelase job, as well as daily using the latest available snapshot build.

Continuous Delivery repos for Oxygen, Fluorine and Neon:

- opendaylight-oxygen-epel-7-x86_64-devel
- opendaylight-fluorine-epel-7-x86_64-devel
- opendaylight-neon-epel-7-x86_64-devel

6.1.3.2 CentOS Community Build System

While most RPM builds are triggered automatically in OpenDaylight's Jenkins, some RPMs are promoted to be hosted in OpenDaylight's CentOS repositories. There are a series of repos that are updated at varying frequencies, from testing repos that are updated with pre-release versions very frequently to release repos that are the permanent home of official OpenDaylight releases.

6.1.3.2.1 Release Repositories

Repositories with the -release suffix host official OpenDaylight releases. They are updated infrequently to never, and will host their release artifacts forever. Release repos are subdivided into two groups based version numbers. Repositories with both a major and minor version number (80, 83) are pinned to a specific OpenDaylight release or service release (Oxygen 8.0.0, Oxygen SR3 8.3.0). Repositories with only a major version (8, 9) will always host the latest service release from that major release. If a new SR comes out, the repo will get the update (Oxygen SR4 will replace Oxygen SR3).

Release repo for the latest Oxygen, Fluorine and Neon service releases:

- nfv7-opendaylight-8-release
- nfv7-opendaylight-9-release
- nfv7-opendaylight-10-release

Release repos that will permanently host specific Oxygen, Fluorine and Neon releases:

- nfv7-opendaylight-80-release
- nfv7-opendaylight-81-release
- nfv7-opendaylight-82-release
- nfv7-opendaylight-83-release
- nfv7-opendaylight-84-release
- nfv7-opendaylight-90-release
- nfv7-opendaylight-91-release
- nfv7-opendaylight-92-release
- nfv7-opendaylight-93-release
- nfv7-opendaylight-100-release
- nfv7-opendaylight-101-release
- nfv7-opendaylight-102-release
- nfv7-opendaylight-103-release
- nfv7-opendaylight-110-release
- nfv7-opendaylight-111-release
- nfv7-opendaylight-112-release

6.1.3.3 Repository Configuration Files

While it's possible to install RPMs directly (*dnf install -y <URL>*), it's often easier to use a repository configuration file to install whatever the latest RPM is in a given repo.

The OpenDaylight Integration/Packaging project provides example repo config files for each official repository.

Package managers like Yum and DNF will automatically find repo configuration files placed in the /etc/yum.repos.d/ directory. Curl them into place with something like:

> **sudo curl -o /etc/yum.repos.d/opendaylight-10-devel.repo** "https://git.opendaylight.org/gerrit/gitweb?p=integration/packaging.git;a=blob_plain;f=packages/rpm/example_repo_configs/opendaylight-10-devel.repo"

Standard install commands will now find the repository as expected.

```
sudo dnf install -y opendaylight
```

The latest RPM in the repo will be installed.

6.1.4 Custom RPMs

It's possible for developers to build custom RPMs, typically with unmerged patches that need system testing.

Most developers will want to run these jobs in the ODL Jenkins sandbox instance, as only a few community members have permission to manually trigger jobs on the releng Jenkins instance. See the Jenkins sandbox docs for details about how to get permissions to trigger sandbox jobs, required configuration and normal usage.

To build an custom distribution with unmerged code, first use the integration-multipatch-test job to create distribution that includes the set of unmerged patches. See the Custom Distributions section for extensive docs.

Once you have the distribution you want to package as an RPM, pass it to the packaging-build-rpm job to do the build. Use the See the *packaging-build-rpm* section for docs.

CHAPTER

SEVEN

PACKAGING OPENDAYLIGHT RELEASES

These docs are for Integration/Packaging committers to reference while they package OpenDaylight releases and service releases. Users should not, and would not be able to due to missing permissions, follow this guide. This process is not needed for Continuous Delivery pipeline packages, just formal releases.

7.1 Building on CentOS Community Build System

OpenDaylight builds and hosts formal releases and service releases on the CentOS Community Build System (CBS). Building on the CBS requires human intervention, as the required credentials can't be stored on our build systems. Continuous Delivery builds are hosted on Nexus to remove this need for a human.

Find the release tarball. Make sure it's the one that has been promoted to the opendaylight.releases Nexus repository, not the same build as a Release Candidate before promotion. The packaging logic will only give release version numbers for builds of artifacts from this release repository.

https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/karaf/0.8.1/karaf-0.8.1.tar.gz

Use the packaging-build-rpm job, for the right stream, to package the tarball.

TODO: Document getting permission to run jobs on RelEng Jenkins.

https://jenkins.opendaylight.org/releng/job/packaging-build-rpm-oxygen/

If it builds and passes tests, download the resulting source RPM from Nexus.

https://nexus.opendaylight.org/content/repositories/opendaylight-oxygen-epel-7-x86_64-devel/

Build the SRPM on the CentOS CBS, using the build target for this release.

TODO: Document adding build targets to CBS, only needed for new major releases TODO: Document getting CBS permissions

```
cbs build nfv7-opendaylight-8-el7 opendaylight-8.1.0-1.el7.src.rpm
```

After the SRPM uploads and the noarch RPM builds, tag it to the appropriate build tags for this release. If this is the first time tag has been used, you'll also need to add the package to the tag.

Releases should typically be tagged to three related tags.

- Candidate tag for this major version

  ```
  cbs add-pkg nfv7-opendaylight-8-candidate opendaylight –owner=dfarrell07 cbs tag-build nfv7-opendaylight-8-candidate opendaylight-8.1.0-1.el7
  ```

- Release tag for this major.minor version

cbs add-pkg nfv7-opendaylight-81-release opendaylight –owner=dfarrell07 cbs tag-build nfv7-opendaylight-81-release opendaylight-8.1.0-1.el7

- Release tag for this major version

  cbs tag-build nfv7-opendaylight-8-release opendaylight-8.1.0-1.el7

It may be advisable to fully do the candidate tag first, and only once everything is verified working do the release tags.

Wait for the repository to regenerate and show the new package.

http://cbs.centos.org/repos/nfv7-opendaylight-8-candidate/x86_64/os/Packages/

Once the RPM is available on the CBS, test it with the test-rpm-master job.

https://jenkins.opendaylight.org/releng/job/packaging-test-rpm-master/

7.2 Updating Docs

Update the downloads page to point at the new RPM.

https://git.opendaylight.org/gerrit/gitweb?p=docs.git;a=blob;f=docs/downloads.rst

Update the Int/Pack repositories docs with any repo additions/removals.

Repositories

7.3 Adding Example Configuration Files

Add example configuration files for any new RPM repositories.

https://git.opendaylight.org/gerrit/gitweb?p=integration/packaging.git;a=tree;f=packages/rpm/example_repo_configs

As a change that depends on the repository configuration file change, add a Packer variables file for the ODL version and CBS repository URL.

https://git.opendaylight.org/gerrit/gitweb?p=integration/packaging.git;a=tree;f=packer/vars

TODO: Document building/pushing VMs/containers after INTPAK-12 automation

7.4 Updating Tests for Release Events

Various release-related events require changes in packaging test coverage.

Tests need to be updated when:

- New releases or service releases are cut
- Old releases or service releases go End-of-Life (EOL)
- New branches are added
- Branches go EOL
- Old temporary autorelease, snapshot or multipatch builds expire (every 30-60 days)

7.4.1 Updating Unit Tests

There are unit tests in Int/Pack that verify the Python build automation scripts.

https://git.opendaylight.org/gerrit/gitweb?p=integration/packaging.git;a=blob;f=packages/test_lib.py

Update the unit tests by grepping around and copying examples.

7.4.2 Updating Functional Tests

There are functional tests in RelEng/Builder that build and test packages to verify build jobs.

https://git.opendaylight.org/gerrit/gitweb?p=releng/builder.git;a=tree;f=jjb/packaging

Update the functional tests by grepping around and copying examples. Make sure to update both the test cases and the default parameters.

7.5 Updating Puppet

The puppet-opendaylight Rakefile, which drives our functional Beaker tests, needs to be updated when a new ODL major release comes out. It does not need to be updated for SRs because it pulls the latest from the <release>-devel Nexus repo (for a diffrent value for <release> on each puppet-opendaylight branch).

The default rpm_repo param in manafests/params.pp and rspec-puppet unit/acceptance tests throughout the repo also need to be updated. They track the latest CD pipelines, so they need to be updated when new branches are cut and CD repos initiated.

7.6 Updating Ansible

The default vars in vars/main.yml need to be updated for each major release and SR. Grep around to find the places to update.

New example playbooks in the ansible-opendaylight/examples directory need to be added for each new branch-cutting/CD pipeline and major release.

rpm_<new devel branch major version>_devel.yml

rpm_<just-released major version>_release.yml

Also update the playbook used in test-ansible-rpm script for each new CD repo.

https://git.opendaylight.org/gerrit/gitweb?p=releng/builder.git;a=blob;f=jjb/packaging/test-ansible-rpm.sh

CHAPTER

EIGHT

VERSIONING

Documentation about OpenDaylight's upstream versioning.

8.1 Overview

Opendaylight has a variety of types of version numbers. Internal ODL features have versions, but they are not visible to external consumers of OpenDaylight. OpenDaylight, built into a distribution of many features, has a version number. OpenDaylight is repackaged in a variety of formats (RPMs, .debs, Docker images, Vagrant base boxes, etc) and follows the guidelines for each. OpenDaylight packages are consumed by configuration management tooling (Ansible, Puppet), which also have their own types of versioning.

8.2 RPMs

The RPM versioning follows the Fedora Packaging Guidelines.

- Major Version numbers that increment with each Simultaneous Release (5=Boron, 6=Carbon, 7=Nitrogen, 8=Oxygen...).
- Minor Version numbers that increment with each Service Release (5.0=Boron, 5.1=Boron SR1, 5.2=Boron SR2...).
- Patch Version is currently unused.
- Package Version numbers that increment for each new package build of the same ODL build (5.0.0-1=Boron, 5.0.0-2=Boron with RPM update).
- Snapshot/autorelease versions with timestamps and incrementing build numbers for pre-release builds (8.0.0-0.1.20171020rel2011=Oxygen pre-release autorelease build, 8.0.0-0.1.20171101snap835=Oxygen pre-release snapshot build...).

See the OpenDaylight builds on the Nexus or the CentOS Community Build System for examples.

8.3 Debs

Mostly the same as RPMs, slightly different way of denoting pre-release builds.

8.4 Docker Images

Docker uses Major, Minor and Patch versions. It doesn't support pre-release version numbers, which is okay since we don't currently build Docker images for pre-release versions. See the tags of the opendaylight/odl image for examples.

8.5 Vagrant Base Boxes

Vagrant follows Rubygems versioning, which uses Major, Minor and Patch versions for semver. It doesn't support pre-release version numbers, which is okay since we don't currently build Vagrant base boxes for pre-release versions. See the versions of the opendaylight/odl base box for examples.

8.6 Ansible Role

The Ansible role follows Semantic Versioning. Version bumps are based on API changes. Backwards incompatible API changes cause Major Version bumps, backwards compatible API changes cause minor version bumps, bugfixes and minor updates can be batched into patch version bumps.

8.7 Puppet Module

The Puppet module follows Semantic Versioning. Version bumps are based on API changes. Backwards incompatible API changes cause Major Version bumps, backwards compatible API changes cause minor version bumps, bugfixes and minor updates can be batched into patch version bumps. See the changelog and metadata for examples of correctly bumping versions.