ANVIL

Release 2015-dev

July 19, 2016

Contents

Everything about ANVIL, a set of python scripts and utilities to forge raw openstack into a productive tool!

CHAPTER 1

Summary

Anvil is a forging tool to help build OpenStack components and their dependencies into a complete package-oriented system.

It automates the git checkouts of the OpenStack components, analyzes & builds their dependencies and the components themselves into packages.

It allows a developer to setup an environment using the automatically created packages (and dependencies, ex. `RPMs`) with the help of anvil configuring the components to work correctly for the developer's needs.

The distinguishing part from devstack (besides being written in Python and not shell), is that after building those packages (currently `RPMs`) the same packages can be used later (or at the same time) to actually deploy at a larger scale using tools such as chef, salt, or puppet (to name a few).

1.1 Features

1.1.1 Configurations

A set of configuration files (in yaml format) that is used for common, component, distribution, code origins configuration...

All the yaml configuration files could be found in:

- `conf/templates/keystone/`
- `conf/components/`
- `conf/distros/`
- `conf/origins/`
- subdirectories of `conf/personas/`

1.1.2 Packaging

- Automatically downloading source from git and performing tag/branch checkouts.
- Automatically verifying and translating requirement files to known pypi/rpm packages.
- Automatically installing and building missing dependencies (pypi and rpm) for you.
- Automatically configuring the needed files, symlinks, adjustments, and any patches.

1.1.3 Pythonic

Written in **python** so it matches the style of other OpenStack components.

1.1.4 Code decoupling

- Components & actions are isolated as individual classes.
- Supports installation personas that define what is to be installed, thus decoupling the 'what' from the 'how'.

Note: This encouraging re-use by others...

1.1.5 Extensive logging

- All commands executed are logged in standard output, all configuration files read/written (and so on).

Note: Debug mode can be activated with `-v` option...

1.1.6 Package tracking and building

- Creation of a single `RPM` set for your installation.
  - This *freezes* what is needed for that release to a known set of packages and dependencies.
- Automatically building and/or including all needed dependencies.
- Includes your distributions *existing* native/pre-built packages (when and where applicable).
  - For example uncommenting the following in the bootstrap file will allow anvil to find dependencies in the epel repository.

CHAPTER 2

How anvil is architected

This little HOWTO can be used by those who wish to understand how anvil does things and why some of its architectural decisions were made.

2.1 History

Once upon a time there was a idea of replacing the then existing devstack with a more robust, more error-tolerant and more user/developer friendly OpenStack setup toolkit. Since the existing devstack did (and still does not support very well) complex intercomponent (and interpackage management system) dependencies and installing/packaging/starting/stopping/uninstalling of OpenStack components.

To solve this problem it was thought that there could be a toolkit that could handle this better. It would also be in Python the language of choice for the rest of the OpenStack components thus making it easier to understand for programmers who are already working in OpenStack code. Thus *devstack2* was born which was later renamed *devstack.py* and after a little while once again got renamed to *anvil*.

2.2 Structure

Anvil is designed to have the following set of software components:

- **Actions:** an action is a sequence of function calls on a set of implementing classes which follows a logically flow from one step to the next. At the end of each step an action may choose to negate a step of another action.
  - Preparing
    - Downloading source code
    - Post-download patching of the source code
    - Deep dependency & requirement analysis
    - Downloading and packaging of missing python dependencies
    - Packaging downloaded source code into SRPMs (aka source RPMs) that is placed into a SRPM repository.
  - Building
    - Creation of a binary RPM repository with all built packages and dependencies (converting the prepared source RPMs into binary RPMs).

- **Phases:** a phase is a step of an action which can be tracked as an individual unit and can be marked as being completed. In the above install action for each component that installed when each step occurs for that component it will be recorded into a file so that if `ctrl-c` aborts anvil and later the install is restarted anvil can notice that the previous phases have already been completed and those phases can be skipped.
  - This is how anvil does action and step resuming.
- **Components:** a component is a class which implements the above steps (which are literally methods on an instance) and is registered with the persona and configuration to be activated. To aid in making it easier to add in new components a set of *generic* base classes exist that provide common functionality that should work in most simplistic installs. These can be found in `anvil/components/`. All current components that exist either use these base classes directly or inherit from them and override functions to provide additional capabilities needed to perform the specified action.
- **Distributions:** a distribution is a yaml file that is tied to a operating system distribution and provides references for components to use in a generic manner. Some of these references include how to map a components `pip-requires` file dependencies to distribution specific dependencies (possibly using `yum` or `apt`) or what non-specified dependencies are useful in getting the component up and running (such as `guestfish` for image mounting and manipulation). Other helpful references include allowing for components to specify standard identifiers for configuration such as `pip`. This allows the underlying yaml file to map the `pip` command to a distribution-centric command (in RHEL it it's really named `pip-python`), see the *commands* key in the yaml files for examples of these settings. Note that each distribution yaml file that exists in `conf/distros` provides this set of references for each component and gets selected based on the yaml key in that file named *platform_pattern*.
- **Configuration:** central to how anvil operates is the ability to be largely configuration driven (code when you need it but avoid it if you can). Distributions as seen by the `conf/distros` folder specify distribution-specific *configuration* that can be referenced by standard keys by a given component. Each component also receives additional configuration (accessible via a components `get_option` function) via the yaml files specified in `conf/components` which provides for a way to have configuration that is not distribution specific but instead is component specific (say for configuring *nova* to use kvm instead of qemu).
  - This configuration drive approach (as much as can be possible) was a key design goal that drives how anvil was and is developed. It has even seemed to be ahead of its time due to how anvil has a distribution yaml file that has specified component dependencies long before the OpenStack community even recognized such a dependency list was useful.
- **Personas:** a persona is a way for anvil to know what components (and possibly subsystems of those components) you wish to have the given action applied to. Since not everyone can agree on what is an install of OpenStack this concept allows for those who wish to have a different set to do so. It is as all other configuration another yaml file and can be examined by looking into the `conf/personas` folders.
  - Each yaml file contains the list of components to be performed for the given action, a simple set of options for those components (for options that may not be applicable to be in the component configuration yaml) and which subsystems a given component will have enabled (if the component supports this concept) as well as which distribution the persona supports (if there is a desire to restrict a given persona to a given distribution this field can be used to accomplish that goal).

CHAPTER 3

Getting started

Made to be as simple as possible, but not too simple...

3.1 Prerequisites

3.1.1 RTFM

Read the great documentation for developers/admins at

- http://docs.openstack.org/developer/
- http://docs.openstack.org/

This will vastly help you understand what the configurations and options do when ANVIL configures them.

3.1.2 Linux

One of the tested distributions.

- RHEL 6.2+
- CentOS 6.2+
- Oracle Enteprise Linux 6.2+

You can get CentOS 6.2+ (**64-bit** is preferred) from https://www.centos.org/

3.2 Installation

3.2.1 Pre-setup

Since RHEL requires a tty to perform `sudo` commands we need to disable this so `sudo` can run without a tty. This seems needed since nova and other components attempt to do `sudo` commands. This isn't possible in RHEL unless you disable this (since those instances won't have a tty).

```
$ sudo visudo
```

Then comment out line

```
Default requiretty
```

Also disable selinux:

```
$ sudo vi /etc/sysconfig/selinux
```

Change *SELINUX=enforcing* to *SELINUX=disabled* then reboot.

```
$ sudo reboot
```

Create specifc user to isolate all the Anvil processes from root user

```
$ sudo useradd <username>
$ sudo passwd <username>
```

Set user as sudoer

```
$ sudo visudo
```

Add *<username> ALL=(ALL) ALL*

Make all the rest of actions as <username> user

```
$ sudo su - <username>
```

3.2.2 Get git!

```
$ sudo yum install git -y
```

3.2.3 Download

We'll grab the latest version of ANVIL via git:

```
$ git clone https://git.openstack.org/openstack/anvil.git
$ cd anvil
```

3.2.4 Configuration

Any configuration to be updated should now be done.

Please edit the corresponding yaml files in `conf/components/` or `conf/components/personas` to fit your desired configuration of nova/glance and the other OpenStack components.

Note: You can use `-p <conf/components/required_file.yaml>` to specify a different persona.

To specify which versions of OpenStack components you want to install select or edit an origins configuration file from `<conf/origins/>`.

Note: You can use `-o <conf/origins/origins_file.yaml>` to specify this different origins file.

Respository notes for those with RedHat subscriptions

To enable the needed repositories for various requirements please also run:

```
sudo subscription-manager repos --enable rhel-6-server-optional-rpms
```

You can also include the RDO repositories (which has even more of the needed requirements). This will ensure that anvil has to build less dependencies overall.

- http://openstack.redhat.com/Repositories

3.2.5 Pre-installing

In order to ensure that anvil will have its correct dependencies you need to first run the bootstrapping code that will setup said dependencies for your operating system.

```
sudo ./smithy --bootstrap
```

3.2.6 Preparing

Now prepare *OpenStacks* components by running the following:

```
./smithy -a prepare
```

You should see a corresponding OpenStack repositories getting downloaded using git, python setups occurring and configuration files being written as well as source rpm packages being built and a repository setup from those components [1].

3.2.7 Building

Now build *OpenStacks* components by running the following:

```
sudo ./smithy -a build
```

You should see a corresponding OpenStack components and dependencies at this stage being packaged into rpm files and two repositories being setup for you [1]. One repository will be the dependencies that the OpenStack components need to run and th other will be the OpenStack components themselves.

3.3 Issues

Please report issues/bugs to https://launchpad.net/anvil. Much appreciated!

[1] If you desire more informational output add a `-v` or a `-vv` to the command.

CHAPTER 4

Documentation

For great documentation on all things OpenStack check out the following relevant links and webpages.

4.1 For admins/users

- http://docs.openstack.org/

4.2 For developers

4.2.1 Adding your own distribution

This little `HOWTO` can be used by those who wish to add-on to anvil to be able to support their own distribution or unsupported operating system (so that it can be supported).

Diving in!

First you have to have a little background on *anvil* and how it operates. So let's dive in and learn a little on how we can add in your own distribution support.

smithy The main shell script that bootstraps the needed dependencies for anvil to be able to start (including items such as termcolor, progressbar and netifaces). The code here is written in bash shell script so that it can execute in an environment without the needed prerequisites.

> **When to adjust:** Adjust the bootstrapping functions in this file to install any needed prerequisites for your operating system to run anvil. Look at how we are bootstrapping rhel (and how we are detecting rhel) for an example.

conf/distros This set of yaml files contains definitions for what packages, what pip to package mappings and what code entrypoints are used when setting up a given component. The critical key here is the `platform_pattern` key which is used as a regular expression to determine if the provided yaml file will work in the given running distribution. Other keys are used to identify which packaging class to use (ie `packager_name`) and how to map a component name to its action classes (i.e. `action_classes/install` will be constructed when an install action occurs). The `commands` section can be used to *house* arbitrary commands which may vary between operating systems (such as the `pip` executable name)

When to adjust the distro: If a suitable distribution already exists (which may be the case for many rhel variants), just go ahead and add-on to the regular expression your pattern. Ensure that your regular expression matches the output of the following command: `python -c "import platform; print(platform.platform())"` which is what anvil uses internally to match a given yaml file to a given distribution.

When to add a new file: If no suitable distribution exists (which may be the case for ubuntu), you will need to go ahead and create a new file for that distribution and include its dependencies and any variations in packaging and pip -> package mappings needed to setup that distribution with the openstack component software.

anvil/distros These are typically subclasses of components that may override generic functionality to correct for a given distribution doing or requiring something different to occur before/after or during an install or other action.

When to adjust: Feel free to add-on your own subclasses here as needed to handle any special actions that your new distribution may require and make sure you reference those classes/entrypoints in your **conf/distros** yaml file so that the correct subclass will be used. The rhel distro has a good set of examples that overload various key points so that rhel can work correctly.

anvil/packaging The modules in this folder will be referenced in your **conf/distros** yaml file and will control how to install packages (i.e. using yum and pip) and how to uninstall those same packages. This code will also get activated when a 'package' action occurs which currently will cause the necessary actions to occur to create a RPM `spec` file which can be used with the `rpmbuild` command.

When to adjust: If needed it should be simple to look at the packaging interface and add your own. After adding make sure you reference them in your **conf/distros** yaml file so that the correct subclass will be used. If you are going to want to create package files from the installed code then you will need to hook-in to a file similar to the RPM module that exists there.

CHAPTER 5

Questions and Answers

5.1 How do I cause the anvil dependencies to be reinstalled?

Anvil bootstraps itself via shell script (if you look at the code in the file `smithy` you will see that it is actually a bash script).

This bootstrapping occurs to ensure that anvils pypi/rpm/deb dependencies are installed before anvil can actually be used. To remove the files that are left behind to let the shell script know when this happens delete files located at `$HOME/.anvil_bootstrapped` and at `$PWD/.anvil_bootstrapped` to cause bootstrapping to occur again.

Another way to make this happen temporarily is to use the following:

```
sudo BOOT_FILES=/dev/null ./smithy
```

This will make anvil think those files are coming from `/dev/null` which will always return nothing. Using the same variable also allows you to retarget the locations where the `smithy` shell script will look for the 'marker' files if you so choose (say in a continuous integration environment).

5.2 How do I run a specific OpenStack milestone?

Anvil has the same tag names as OpenStack releases so to run against a specific milestone of OpenStack just checkout the same tag in anvil and run the same actions as you would have ran previously.

An example of this, lets adjust `nova` to use the `stable/essex` branch.

- Open `conf/origins/master.yaml` file in your favorite editor
- Locate lines that describe the `nova` component
- Change branch parameter to the desired one

```
nova:
    repo: https://github.com/openstack/nova.git
    branch: stable/essex
```

- **Component origin parameters are:**
  - `repo: <repo_url>` - required
  - `branch: <branch>` - optional
  - `tag: <tag>` - optional

If no branch nor tag parameters were specified then `branch: master` is used by default.

Note: tag overrides branch (so you can't really include both)

CHAPTER 6

Bugs & Hugs & Code

ANVIL is an open-source tool released under the apache version 2.0 license. It *depends* on its **community** to keep it alive.

6.1 IRC

You can also usually find us on `#openstack-anvil` on freenode.

6.2 Source code

The source code is on github located at:

http://git.openstack.org/cgit/openstack/anvil (mirrored @ http://github.com/openstack/anvil/).

Feel free to fork it and contribute to it.

6.3 Bugs

http://bugs.launchpad.net/anvil

6.4 Branches

Anvil tries to work across different OpenStack releases as of the havana release...

If it does not work across the *majority* of OpenStack releases please file a bug.

6.5 Hacking

Feel free to hack but please try to follow the hacking guidelines.

6.6 Links

Please visit as often as you want at the following urls:

- http://launchpad.net/anvil (blueprints for features, bugs, q/a...)
- http://launchpad.net/~anvil-dev (talk to the devs directly)

Help and developer work/time is always much appreciated!

CHAPTER 7

Examples

7.1 Bootstrapping

This is needed to get ready for the rest of anvils stages by installing anvils python dependencies so that anvil can correctly run using said dependencies.

```
$ sudo ./smithy --bootstrap
```

Terminal recording: http://showterm.io/effa75ea631777a2e74a0/

7.2 Preparing

This stage does the download of the source repositories, analysis of dependencies, download of missing dependencies and building of source repositories and missing dependencies into source rpms.

```
$ ./smithy -a prepare
```

Terminal recording: http://showterm.io/12c29e87094f128d945fa/

7.3 Building

This is the stage responsible for translating the previously prepared source rpms into installable rpms (of the non-source type). The output of this phase is two repositories, one with the dependencies and one with the rpms for the openstack components themselves.

```
$ sudo ./smithy -a build
```

Terminal recording: http://showterm.io/2fee38794dcf536ccd437/

7.4 Packaging

To see the packages built (after prepare has finished).

```
$ ls /home/harlowja/openstack/deps/rpmbuild/SPECS/ | cat
```

```
openstack-deps.spec
pylint.spec
pyparsing.spec
python-babel.spec
python-cheetah.spec
python-cinderclient.spec
python-cinder.spec
python-cliff.spec
python-cliff-tablib.spec
python-cmd2.spec
python-colorama.spec
python-coverage.spec
python-crypto.spec
python-decorator.spec
python-discover.spec
python-docutils.spec
python-extras.spec
python-fixtures.spec
python-glanceclient.spec
python-glance.spec
python-hp3parclient.spec
python-httplib2.spec
python-jinja2.spec
python-jsonpatch.spec
python-jsonpointer.spec
python-jsonschema.spec
python-keystoneclient.spec
python-keystone.spec
python-ldap.spec
python-logilab-astng.spec
python-logilab-common.spec
python-lxml.spec
python-markdown.spec
python-markupsafe.spec
python-mimeparse.spec
python-netaddr.spec
python-nose-exclude.spec
python-nosehtmloutput.spec
python-nose.spec
python-nosexcover.spec
python-novaclient.spec
python-nova.spec
python-openstack-nose-plugin.spec
python-oslo-config.spec
python-pam.spec
python-pastedeploy.spec
python-pep8.spec
python-prettytable.spec
python-pygments.spec
python-pysqlite.spec
python-neutronclient.spec
python-repoze-lru.spec
python-routes.spec
python-setuptools-git.spec
python-setuptools.spec
python-sphinx.spec
python-sqlalchemy-migrate.spec
python-sqlalchemy.spec
```

```
python-subunit.spec
python-tablib.spec
python-tempita.spec
python-termcolor.spec
python-testrepository.spec
python-testtools.spec
python-unittest2.spec
python-warlock.spec
python-webob.spec
python-wsgiref.spec
python-xattr.spec
```

```
$ cat openstack-deps.spec
```

```
Name: openstack-deps
Version: 2013.6.3
Release: 0
License: Apache 2.0
Summary: OpenStack dependencies
BuildArch: noarch

Requires: MySQL-python
Requires: avahi
Requires: coreutils
Requires: curl
Requires: dnsmasq
Requires: dnsmasq-utils
Requires: ebtables
Requires: fuse
Requires: gawk
Requires: git
Requires: guestfish
Requires: iptables
Requires: iputils
Requires: iscsi-initiator-utils
Requires: kpartx
Requires: libguestfs
Requires: libguestfs-mount
Requires: libguestfs-tools
Requires: libvirt
Requires: libvirt-client
Requires: libvirt-python
Requires: libxml2-devel
Requires: libxslt-devel
Requires: lsof
Requires: mlocate
Requires: mysql
Requires: mysql-server
Requires: openssh-server
Requires: parted
Requires: postgresql-devel
Requires: psmisc
Requires: python
Requires: python-devel
Requires: python-distutils-extra
Requires: python-setuptools
Requires: qemu-img
Requires: qemu-kvm
```

```
Requires: rabbitmq-server
Requires: sqlite
Requires: sqlite-devel
Requires: sudo
Requires: tcpdump
Requires: unzip
Requires: vconfig
Requires: wget

%description


%pre

# rabbitmq-server
service qpidd stop 2>/dev/null || true
chkconfig qpidd off 2>/dev/null || true


%files

```

```
$ cat python-nova.spec
```

```
%define pkg_name nova
%define version 2013.1
%define unmangled_version 2013.1
%define unmangled_version 2013.1
%define release 1

Summary: cloud computing fabric controller
Name: python-nova
Epoch: 2
Version: %{version}
Release: %{release}
Source0: %{pkg_name}-%{unmangled_version}.tar.gz
License: UNKNOWN
Group: Development/Libraries
BuildRoot: %{_tmppath}/%{pkg_name}-%{version}-%{release}-buildroot
Prefix: %{_prefix}
BuildArch: noarch
Vendor: OpenStack <nova@lists.launchpad.net>
Requires:  python-sqlalchemy >= 0.7.8 python-sqlalchemy < 0.7.99 python-cheetah >= 2.4.4 python-amqpl
Url: http://www.openstack.org/

%description
UNKNOWN

%prep
%setup -n %{pkg_name}-%{unmangled_version} -n %{pkg_name}-%{unmangled_version}

%build
python setup.py build

%install
python setup.py install --single-version-externally-managed -O1 --root=$RPM_BUILD_ROOT --record=INSTA
abspath_installed_files=$(readlink -f INSTALLED_FILES)
(
```

```
    cd $RPM_BUILD_ROOT
    for i in usr/*/python*/site-packages/* usr/bin/*; do
        if [ -e "$i" ]; then
            sed -i "s@/$i/@DELETE_ME@" "$abspath_installed_files"
            echo "/$i"
        fi
    done
    if [ -d usr/man ]; then
        rm -rf usr/share/man
        mkdir -p usr/share
        mv usr/man usr/share/
        sed -i "s@/usr/man/@DELETE_ME@" "$abspath_installed_files"
        for i in usr/share/man/*; do
            echo "/$i/*"
        done
    fi
) >> GATHERED_FILES
{ sed '/^DELETE_ME/d' INSTALLED_FILES; cat GATHERED_FILES; } | sort -u > INSTALLED_FILES.tmp
mv -f INSTALLED_FILES{.tmp,}


%clean
rm -rf $RPM_BUILD_ROOT

%files -f INSTALLED_FILES
%defattr(-,root,root)
```