bootstrap-vz Documentation

Release 0.9.11

Anders Ingemann

Aug 11, 2017

Contents

CHAPTER 1

Official EC2 manifests

The official Debian images for EC2 are built with bootstrap-vz. In the folder `manifests/official/ec2` you will find the various manifests that are used to create the different flavors of Debian AMIs for EC2.

You can read more about those official images in the Debian wiki.

The official images can be found on the AWS marketplace.

CHAPTER 2

Official GCE manifests

These are the official manifests used to build [Google Compute Engine (GCE) Debian images](https://cloud.google.com/compute/docs/images).

The included packages and configuration changes are necessary for Debian to run on GCE as a first class citizen of the platform. Included GCE software is published on github: [Google Compute Engine guest environment](https://github.com/GoogleCloudPlatform/compute-image-packages)

Debian 8 Jessie Package Notes:

- python-crcmod is pulled in from backports as it provides a compiled crcmod required for the Google Cloud Storage CLI (gsutil).
- cloud-utils and cloud-guest-utils are pulled in from backports as they provide a fixed version of growpart to safely grow the root partition on disks >2TB.
- google-cloud-sdk is pulled from a Google Cloud repository.
- google-compute-engine is pulled from a Google Cloud repository.
- google-compute-engine-init is pulled from a Google Cloud repository.
- google-config is pulled from a Google Cloud repository.

jessie-minimal:

The only additions are the necessary google-compute-engine, google-compute-engine-init, and google-config packages. This image is not published on GCE however the manifest is provided here for those wishing a minimal GCE Debian image.

stretch and stretch-minimal:

These manifests are provided for testing. Debian 9 Stretch is not yet stable.

Deprecated manifests:

Debian 7 Wheezy and Backports Debian 7 Wheezy are deprecated images on GCE and are no longer supported. These manifests are provided here for historic purposes.

The manifest file is the primary way to interact with bootstrap-vz. Every configuration and customization of a Debian installation is specified in this file.

The manifest format is YAML or JSON. It is near impossible to run the bootstrapper with an invalid configuration, since every part of the framework supplies a json-schema that specifies exactly which configuration settings are valid in different situations.

CHAPTER 3

Manifest variables

Many of the settings in the example manifests use strings like `debian-{system.release}-{system.architecture}-{{"{%y"}}}{{"{%m"}}}{{"{%d"}}}`. These strings make use of manifest variables, which can cross reference other settings in the manifest or specific values supplied by the bootstrapper (e.g. all python date formatting variables are available).

Any reference uses dots to specify a path to the desired manifest setting. Not all settings support this though, to see whether embedding a manifest variable in a setting is possible, look for the `manifest vars` label.

To insert a literal `{foo}` use double braces, that is `{{foo}}`. For example in a shell command where you may want to use the expression `${foo}`, use `${{foo}}` instead.

CHAPTER 4

Sections

The manifest is split into 7 sections.

Name

Single string property that specifies the name of the image.

- `name`: The name of the resulting image. When bootstrapping cloud images, this would be the name visible in the interface when booting up new instances. When bootstrapping for VirtualBox or kvm, it's the filename of the image. `required` `manifest vars`

Example:

```
---
name: debian-{system.release}-{system.architecture}-{%Y}-{%m}-{%d}-ebs
```

Provider

The provider section contains all provider specific settings and the name of the provider itself.

- `name`: target virtualization platform of the installation `required`

Consult the providers section of the documentation for a list of valid values.

Example:

```
---
provider:
  name: ec2
```

Bootstrapper

This section concerns the bootstrapper itself and its behavior. There are 4 possible settings:

- `workspace`: Path to where the bootstrapper should place images and intermediate files. Any volumes will be mounted under that path. `required`
- `tarball`: debootstrap has the option to download all the software and pack it up in a tarball. When starting the actual bootstrapping process, debootstrap can then be pointed at that tarball and use it instead of downloading anything from the internet. If you plan on running the bootstrapper multiple times, this option can save you a lot of bandwidth and time. This option just specifies whether it should create a new tarball or not. It will search for and use an available tarball if it already exists, regardless of this setting. `optional` Valid values: `true, false` Default: `false`
- `mirror`: The mirror debootstrap should download software from. It is advisable to specify a mirror close to your location (or the location of the host you are bootstrapping on), to decrease latency and improve bandwidth. If not specified, *the configured aptitude mirror URL* is used. `optional`
- `include_packages`: Extra packages to be installed during bootstrap. Accepts a list of package names. `optional`
- `exclude_packages`: Packages to exclude during bootstrap phase. Accepts a list of package names. `optional`
- `variant`: Debian variant to install. The only supported value is `minbase` and should only be used in conjunction with the Docker provider. Not specifying this option will result in a normal Debian variant being bootstrapped.

Example:

```
---
bootstrapper:
  workspace: /target
  tarball: true
  mirror: http://deb.debian.org/debian/
  include_packages:
     - whois
     - psmisc
  exclude_packages:
     - isc-dhcp-client
     - isc-dhcp-common
  variant: minbase
```

System

This section defines anything that pertains directly to the bootstrapped system and does not fit under any other section.

- `architecture`: The architecture of the system. Valid values: `i386, amd64` `required`
- `bootloader`: The bootloader for the system. Depending on the bootmethod of the virtualization platform, the options may be restricted. Valid values: `grub, extlinux, pv-grub` `required`
- `charmap`: The default charmap of the system. Valid values: Any valid charmap like `UTF-8`, `ISO-8859-` or `GBK`. `required`
- `hostname`: hostname to preconfigure the system with. `optional`

- `locale`: The default locale of the system. Valid values: Any locale mentioned in `/etc/locale.gen` `required`
- `release`: Defines which debian release should be bootstrapped. Valid values: `squeeze`, `wheezy`, `jessie`, `sid`, `oldstable`, `stable`, `testing`, `unstable` `required`
- `timezone`: Timezone of the system. Valid values: Any filename from `/usr/share/zoneinfo` `required`

Example:

```
---
system:
  release: jessie
  architecture: amd64
  bootloader: extlinux
  charmap: UTF-8
  hostname: jessie x86_64
  locale: en_US
  timezone: UTC
```

Packages

The packages section allows you to install custom packages from a variety of sources.

- `install`: A list of strings that specify which packages should be installed. Valid values: Package names optionally followed by a `/target` or paths to local `.deb` files. Note that packages are installed in the order they are listed. The installer invocations are bundled by package type (remote or local), meaning if you install two local packages, then two remote packages and then another local package, there will be two calls to `dpkg -i ...` and a single call to `apt-get install ...`.
- `install_standard`: Defines if the packages of the `"Standard System Utilities"` option of the Debian installer, provided by tasksel, should be installed or not. The problem is that with just `debootstrap`, the system ends up with very basic commands. This is not a problem for a machine that will not be used interactively, but otherwise it is nice to have at hand tools like `bash-completion`, `less`, `locate`, etc. `optional` Valid values: `true`, `false` Default: `false`
- `mirror`: The default aptitude mirror. `optional` Default: `http://deb.debian.org/debian/`
- `sources`: A map of additional sources that should be added to the aptitude sources list. The key becomes the filename in `/etc/apt/sources.list.d/` (with `.list` appended to it), except for `main`, which designates `/etc/apt/sources.list`. The value is an array with each entry being a line. `optional`
- `components`: A list of components that should be added to the default apt sources. For example `contrib` or `non-free` `optional` Default: `['main']`
- `trusted-keys`: List of paths (relative to the manifest) to `.gpg` keyrings that should be added to the aptitude keyring of trusted signatures for repositories. `optional`
- `apt.conf.d`: A map of `apt.conf(5)` configuration snippets. The key become the filename in `/etc/apt/apt.conf.d`, except `main` which designates `/etc/apt/apt.conf`. The value is a string in the `apt.conf(5)` syntax. `optional`
- `preferences`: Allows you to pin packages through apt preferences. The setting is an object where the key is the preference filename in `/etc/apt/preferences.d/`. The key `main` is special and refers to the file `/etc/apt/preferences`, which will be overwritten if specified. `optional` The values are objects with three keys:
  - `package`: The package to pin (wildcards allowed)

  - `pin`: The release to pin the package to.
  - `pin-priority`: The priority of this pin.

Example:

```
---
packages:
  install:
    - /root/packages/custom_app.deb
    - puppet
  install_standard: true
  mirror: http://cloudfront.debian.net/debian
  sources:
    puppet:
      - deb http://apt.puppetlabs.com wheezy main dependencies
  components:
    - contrib
    - non-free
  trusted-keys:
    - /root/keys/puppet.gpg
  apt.conf.d:
    00InstallRecommends: >-
      APT::Install-Recommends "false";
      APT::Install-Suggests   "false";
    00IPv4: 'Acquire::ForceIPv4 "false";'
  preferences:
    main:
      - package: *
        pin: release o=Debian, n=wheezy
        pin-priority: 800
      - package: *
        pin: release o=Debian Backports, a=wheezy-backports, n=wheezy-backports
        pin-priority: 760
      - package: puppet puppet-common
        pin: version 2.7.25-1puppetlabs1
        pin-priority: 840
```

Volume

bootstrap-vz allows a wide range of options for configuring the disk layout of the system. It can create unpartitioned as well as partitioned volumes using either the gpt or msdos scheme. At most, there are only three partitions with predefined roles configurable though. They are boot, root and swap.

- `backing`: Specifies the volume backing. This setting is very provider specific. Valid values: `ebs`, `s3`, `vmdk`, `vdi`, `raw`, `qcow2`, `lvm` `required`
- `partitions`: A map of the partitions that should be created on the volume.
- `type`: The partitioning scheme to use. When using `none`, only root can be specified as a partition. Valid values: `none`, `gpt`, `msdos` `required`
- `root`: Configuration of the root partition. `required`
  - `size`: The size of the partition. Valid values: Any datasize specification up to TB (e.g. 5KiB, 1MB, 6TB). `required`
  - `filesystem`: The filesystem of the partition. When choosing `xfs`, the `xfsprogs` package will need to be installed. Valid values: `ext2`, `ext3`, `ext4`, `xfs` `required`

- `format_command`: Command to format the partition with. This optional setting overrides the command bootstrap-vz would normally use to format the partition. The command is specified as a string array where each option/argument is an item in that array (much like the commands plugin). `optional` The following variables are available:
- `{fs}`: The filesystem of the partition.
- `{device_path}`: The device path of the partition.
- `{size}`: The size of the partition.
- `{mount_opts}`: Options to mount the partition with. This optional setting overwrites the default option list bootstrap-vz would normally use to mount the partiton (defaults). The List is specified as a string array where each option/argument is an item in that array. `optional` Here some examples:
- `nodev`
- `nosuid`
- `noexec`
- `journal_ioprio=3`

The default command used by bootstrap-vz is `['mkfs.{fs}', '{device_path}']`.

- `boot`: Configuration of the boot partition. All settings equal those of the root partition. `optional`
- `swap`: Configuration of the swap partition. Since the swap partition has its own filesystem you can only specify the size for this partition. `optional`
- `additional_path`: Configuration of additional partitions. (e.g. /var/tmp) All settings equal those of the root partition.

Example:

```
---
volume:
  backing: vdi
  partitions:
    type: msdos
    boot:
      filesystem: ext2
      size: 32MiB
    root:
      filesystem: ext4
      size: 864MiB
    swap:
      size: 128MiB
```

Plugins

The plugins section is a map of plugin names to whatever configuration a plugin requires. Go to the plugin section of the documentation, to see the configuration for a specific plugin.

Example:

```
---
plugins:
  minimize_size:
```

```
    zerofree: true
    shrink: true
```

CHAPTER 5

Providers

Azure

This provider generates raw images for Microsoft Azure computing platform.

Manifest settings

Provider

- `waagent`: Waagent specific settings. `required`
  - `conf`: Path to `waagent.conf` that should override the default `optional`
  - `version`: Version of waagent to install. Waagent versions are available at: https://github.com/Azure/WALinuxAgent/releases `required`

Example:

```
---
provider:
  name: azure
  waagent:
    conf: /root/waagent.conf
    version: 2.0.4
```

The Windows Azure Linux Agent can automatically configure swap space using the local resource disk that is attached to the VM after provisioning on Azure. Modify the following parameters in /etc/waagent.conf appropriately:

```
ResourceDisk.Format=y
ResourceDisk.Filesystem=ext4
ResourceDisk.MountPoint=/mnt/resource
ResourceDisk.EnableSwap=y
ResourceDisk.SwapSizeMB=2048    ## NOTE: set this to whatever you need it to be.
```

Docker

The Docker provider creates a docker image from scratch, creates a Dockerfile for it and imports the image to a repo specified in the manifest.

In order to reduce the size of the image, it is highly recommend to make use of the minimize_size plugin. With optimal settings a 64-bit jessie image can be whittled down to 81.95 MB (built on Dec 13th 2015 with `manifests/examples/docker/jessie-minimized.yml`).

Manifest settings

Name

- `name`: The image name is the repository and tag to where an image should be imported. `required manifest vars`

Provider

- `dockerfile`: List of Dockerfile instructions that should be appended to the ones created by the bootstrapper. `optional`
- `labels`: Labels that should be added to the dockerfile. The image name specified at the top of the manifest will be added as the label `name`. Check out the docker docs for more information about custom labels. Project atomic also has some useful recommendations for generic container labels. `optional manifest vars`

Example:

```
---
name: bootstrap-vz:latest
provider:
  name: docker
  dockerfile:
    - CMD /bin/bash
  labels:
    name: debian-{system.release}-{system.architecture}-{%y}{%m}{%d}
    description: Debian {system.release} {system.architecture}
```

EC2

The EC2 provider automatically creates a volume for bootstrapping (be it EBS or S3), makes a snapshot of it once it is done and registers it as an AMI. EBS volume backing only works on an EC2 host while S3 backed volumes *should* work locally (at this time however they do not, a fix is in the works).

Unless the cloud-init plugin is used, special startup scripts will be installed that automatically fetch the configured authorized_key from the instance metadata and save or run any userdata supplied (if the userdata begins with `#!` it will be run). Set the variable `install_init_scripts` to `False` in order to disable this behaviour.

Manifest settings

Credentials

The AWS credentials can be configured in two ways: Via the manifest or through environment variables. To bootstrap S3 backed instances you will need a user certificate and a private key in addition to the access key and secret key, which are needed for bootstraping EBS backed instances.

The settings describes below should be placed in the `credentials` key under the `provider` section.

- `access-key`: AWS access-key. May also be supplied via the environment variable `$AWS_ACCESS_KEY` `required for EBS & S3 backing`
- `secret-key`: AWS secret-key. May also be supplied via the environment variable `$AWS_SECRET_KEY` `required for EBS & S3 backing`
- `certificate`: Path to the AWS user certificate. Used for uploading the image to an S3 bucket. May also be supplied via the environment variable `$AWS_CERTIFICATE` `required for S3 backing`
- `private-key`: Path to the AWS private key. Used for uploading the image to an S3 bucket. May also be supplied via the environment variable `$AWS_PRIVATE_KEY` `required for S3 backing`
- `user-id`: AWS user ID. Used for uploading the image to an S3 bucket. May also be supplied via the environment variable `$AWS_USER_ID` `required for S3 backing`

Example:

```
---
provider:
  name: ec2
  credentials:
    access-key: AFAKEACCESSKEYFORAWS
    secret-key: thes3cr3tkeyf0ryourawsaccount/FS4d8Qdva
```

Virtualization

EC2 supports both paravirtual and hardware virtual machines. The virtualization type determines various factors about the virtual machine performance (read more about this in the EC2 docs).

- `virtualization`: The virtualization type Valid values: `pvm`, `hvm` `required`

Example:

```
---
provider:
  name: ec2
  virtualization: hvm
```

Enhanced networking

Install enhanced networking drivers to take advantage of SR-IOV capabilities on hardware virtual machines. Read more about this in the EC2 docs.

Example:

```
---
provider:
  name: ec2
  virtualization: hvm
  enhanced_networking: simple
```

Image

- `description`: Description of the AMI. `manifest vars`
- `bucket`: When bootstrapping an S3 backed image, this will be the bucket where the image is uploaded to. `required for S3 backing`
- `region`: Region in which the AMI should be registered. `required for S3 backing`

Example:

```
---
provider:
  name: ec2
  description: Debian {system.release} {system.architecture}
  bucket: debian-amis
  region: us-west-1
```

Dependencies

To communicate with the AWS API boto3 is required you can install boto with `pip install boto3` (on wheezy, the packaged version is too low). S3 images are chopped up and uploaded using euca2ools (install with `apt-get install euca2ools`).

Google Compute Engine

The GCE provider can creates image as expected by GCE - i.e. raw disk image in *.tar.gz file. It can upload created images to Google Cloud Storage (to a URI provided in the manifest by `gcs_destination`) and can register images to be used by Google Compute Engine to a project provided in the manifest by `gce_project`. Both of those functionalities are not fully tested yet.

Note that to register an image, it must first be uploaded to GCS, so you must specify `gcs_destination` (upload to GCS) to use `gce_project` (register with GCE)

Manifest settings

Provider

- `description`: Description of the image.
- `gcs_destination`: Image destination in GCS.
- `gce_project`: GCE project in which to register the image.

Example:

```
---
provider:
  name: gce
  description: Debian {system.release} {system.architecture}
  gcs_destination: gs://my-bucket
  gce_project: my-project
```

KVM

The KVM provider creates virtual images for Linux Kernel-based Virtual Machines. It supports the installation of virtio kernel modules (paravirtualized drivers for IO operations). It also supports creating an image with LVM and qcow2 as a disk backend.

Manifest settings

Provider

- `virtio`: Specifies which virtio kernel modules to install. `optional`
- `logicalvolume`: Specifies the logical volume where the disk image will be built. `volumegroup`: Specifies the volume group where the logical volume will be stored. These options should only be used if `lvm` was given as a disk backend.

Example:

```
---
provider:
  name: kvm
  virtio:
    - virtio_blk
    - virtio_net
volume:
  backing: lvm
  logicalvolume: lvtest
  volumegroup: vgtest
```

Oracle

The Oracle provider creates RAW images compressed in a `.tar.gz` tarball. Those images can be uploaded using the web interface of the Oracle Compute Cloud Service dashboard or configured to be automatically sent by our Oracle Storage Cloud Service API embedded client.

Manifest settings

Credentials

The settings described below should be placed in the `credentials` key under the `provider` section, if the image is intended to be uploaded after generation. They will be used to authenticate the API client.

- `username`: the same login used to access the Oracle Compute Cloud dashboard. `required`
- `password`: password for the username specified above. `required`
- `identity-domain`: this is auto-generated by Oracle and available in the "New Account Information" e-mail message they send after registration. `required`

Example:

```
---
provider:
  name: oracle
  credentials:
    username: user@example.com
    password: qwerty123456
    identity-domain: usoracle9999
```

Provider

If the `credentials` have been specified, the following settings are available to customize the process of uploading and verifying an image.

- `container`: the container (folder) to which the image will be uploaded. `required`
- `verify`: specifies if the image should be downloaded again and have its checksum compared against the local one. Valid values: `true`, `false`. Default: `false`. `optional`

```
---
provider:
  name: oracle
  container: compute_images
  verify: true
```

VirtualBox

The VirtualBox provider can bootstrap to both .vdi and .vmdk images (raw images are also supported but do not run in VirtualBox). It's advisable to always use vmdk images for interoperability (e.g. OVF files *should* support vdi files, but since they have no identifier URL not even VirtualBox itself can import them).

VirtualBox Guest Additions can be installed automatically if the ISO is provided in the manifest. VirtualBox Additions iso can be installed from main Debian repo by running: *apt install virtualbox-guest-additions-iso*

Manifest settings

Provider

- `guest_additions`: Specifies the path to the VirtualBox Guest Additions ISO, which, when specified, will be mounted and used to install the VirtualBox Guest Additions. `optional`

Example:

```
---
provider:
  name: virtualbox
  guest_additions: /usr/share/virtualbox/VBoxGuestAdditions.iso
```

Providers in bootstrap-vz represent various cloud providers and virtual machines.

bootstrap-vz is an extensible platform with loose coupling and a significant amount of tooling, which allows for painless implementation of new providers.

The virtualbox provider for example is implemented in only 89 lines of python, since most of the building blocks are a part of the common task library. Only the kernel and guest additions installation are specific to that provider.

CHAPTER 6

Plugins

Admin user

This plugin creates a user with passwordless sudo privileges. It also disables the SSH root login. There are three ways to grant access to the admin user:

- Use the EC2 public key (EC2 machines only)
- Set a password for the user
- Provide a SSH public key to allow remote SSH login

If the EC2 init scripts are installed, the script for fetching the SSH authorized keys will be adjusted to match the username specified in `username`.

If a password is provided (the `password` setting), this plugin sets the admin password, which also re-enables SSH password login (off by default in Jessie or newer).

If the optional setting `pubkey` is present (it should be a full path to a SSH public key), you will be able to log in to the admin user account using the corresponding private key (this disables the EC2 public key injection mechanism).

The `password` and `pubkey` settings can be used at the same time.

Settings

- `username`: The username of the account to create. `required`
- `password`: An optional password for the account to create. `optional`
- `pubkey`: The full path to an SSH public key to allow remote access into the admin account. `optional`

Example:

```
---
plugins:
  admin_user:
    username: admin
```

```
    password: s3cr3t
    pubkey: /home/bootstrap-vz/.ssh/id_rsa
```

APT Proxy

This plugin creates a proxy configuration file for APT, so you could enjoy the benefits of using cached packages instead of downloading them from the mirror every time. You could just install `apt-cacher-ng` on the host machine and then add `"address": "127.0.0.1"` and `"port": 3142` to the manifest file.

Settings

- `address`: The IP or host of the proxy server. `required`
- `port`: The port (integer) of the proxy server. `required`
- `username`: The username for authentication against the proxy server. This is ignored if `password` is not also set. `optional`
- `password`: The password for authentication against the proxy server. This is ignored if `username` is not also set. `optional`
- `persistent`: Whether the proxy configuration file should remain on the machine or not. Valid values: `true`, `false` Default: `false`. `optional`

cloud-init

This plugin installs and configures cloud-init on the system. Depending on the release it installs it from either backports or the main repository.

cloud-init is only compatible with Debian wheezy and upwards.

Settings

- `username`: The username of the account to create. `required`
- `groups`: A list of strings specifying which additional groups the account should be added to. `optional`
- `disable_modules`: A list of strings specifying which cloud-init modules should be disabled. `optional`
- `metadata_sources`: A string that sets the datasources that cloud-init should try fetching metadata from (corresponds to debconf-set-selections values). The source is automatically set when using the ec2 provider. `optional`

Commands

This plugin allows you to run arbitrary commands during the bootstrap process. The commands are run at an indeterminate point *after* packages have been installed, but *before* the volume has been unmounted.

Settings

- `commands`: A list of lists containing strings. Each top-level item is a single command, while the strings inside each list comprise parts of a command. This allows for proper shell argument escaping. To circumvent escaping, simply put the entire command in a single string, the command will additionally be evaluated in a shell (e.g. globbing will work). In addition to the manifest variables `{root}` is also available. It points at the root of the image volume. `required manifest vars`

Example

Create an empty *index.html* in */var/www* and delete all locales except english.

```
commands:
  commands:
    - [touch, '{root}/var/www/index.html']
    - ['rm -rf /usr/share/locale/[^en]*']
```

debconf

`debconf(7)` is the configuration system for Debian packages. It enables you to preconfigure packages before their installation.

This plugin lets you specify debconf answers directly in the manifest. You should only specify answers for packages that will be installed; the plugin does not check that this is the case.

Settings

The `debconf` plugin directly takes an inline string::

```
plugins:
  debconf: >-
    d-i pkgsel/install-language-support boolean false
    popularity-contest popularity-contest/participate boolean false
```

Consult `debconf-set-selections(1)` for a description of the data format.

Docker daemon

Install docker daemon in the image. Uses init scripts for the official repository.

This plugin can only be used if the distribution being bootstrapped is at least `wheezy`, as Docker needs a kernel version `3.8` or higher, which is available at the `wheezy-backports` repository. There's also an architecture requirement, as it runs only on `amd64`.

Settings

- `version`: Selects the docker version to install. To select the latest version simply omit this setting. Default: `latest optional`

ec2-launch

This plugin is spinning up **AWS classic instance** from the AMI created with the template from which this plugin is invoked.

Settings

- `security_group_ids`: A list of security groups (not VPC) to attach to the instance `required`
- `instance_type`: A string with AWS Classic capable instance to run (default: m3.medium) `optional`
- `ssh_key`: A string with the ssh key name to apply to the instance. `required`
- `print_public_ip`: A string with the path to write instance external IP to `optional`
- `tags`: `optional`
- `deregister_ami`: A boolean value describing if AMI should be kept after sinning up instance or not (default: false) `optional`

EC2 publish

This plugin lets you publish an EC2 AMI to multiple regions, make AMIs public, and output the AMIs generated in each file.

Settings

- `regions`: EC2 regions to copy the final image to. `optional`
- `public`: Whether the AMIs should be made public (i.e. available by ALL users). Valid values: `true`, `false` Default: `false`. `optional`
- `manifest_url`: URL to publish generated AMIs. Can be a path on the local filesystem, or a URL to S3 (https://bucket.s3-region.amazonaws.com/amis.json) `optional`

Expand Root

This plugin adds support to expand the root partition and filesystem dynamically on boot. It adds a shell script to call growpart and the proper filesystem expansion tool for a given device, partition, and filesystem. The growpart script is part of the cloud-guest-utils package in stretch and jessie-backports. The version of this script in jessie is broken in several ways and so this plugin installs the version from jessie-backports which works correctly. This plugin should not be used in conjunction with common.tasks.initd.AddExpandRoot and common.tasks.initd.AdjustExpandRootScript. It is meant to replace the existing internal common version of expand-root.

Settings

- `filesystem_type`: The type of filesystem to grow, one of ext2, ext3, ext4, of xfs.
- `root_device`: The root device we are growing, /dev/sda as an example.

- `root_partition`: The root partition ID we are growing, 1 (which becomes /dev/sda1). This is specified so you could grow a different partition on the root_device if you have a multi partition setup and because growpart takes the partition number as a separate argument.

File copy

This plugin lets you copy files from the host to the VM under construction, create directories, and set permissions and ownership.

Note that this necessarily violates the first development guideline.

Settings

The `file_copy` plugin takes a (non-empty) `files` list, and optionally a `mkdirs` list.

Files (items in the `files` list) must be objects with the following properties:

- `src` and `dst` (required) are the source and destination paths. `src` is relative to the manifest, whereas `dst` is a path in the VM.
- `permissions` (optional) is a permission string in a format appropriate for `chmod(1)`.
- `owner` and `group` (optional) are respectively a user and group specification, in a format appropriate for `chown(1)` and `chgrp(1)`.

Folders (items in the `mkdirs` list) must be objects with the following properties: - `dir` (required) is the path of the directory. - `permissions`, `owner` and `group` are the same as for files.

Google Cloud Repo

This plugin adds support to use Google Cloud apt repositories for Debian. It adds the public repo key and optionally will add an apt source list file and install a package containing the key in order to maintain the key over time.

Settings

- `cleanup_bootstrap_key`: Deletes the bootstrap key by removing /etc/apt/trusted.gpg in favor of the package maintained version. This is only to avoid having multiple keys around in the apt-key list. This should only be used with enable_keyring_repo.
- `enable_keyring_repo`: Add a repository and package to maintain the repo public key over time.

minimize size

This plugin can be used to reduce the size of the resulting image. Often virtual volumes are much smaller than their reported size until any data is written to them. During the bootstrapping process temporary data like the aptitude cache is written to the volume only to be removed again.

The minimize size plugin employs various strategies to keep a low volume footprint:

- Mount folders from the host into key locations of the image volume to avoid any unnecessary disk writes.

- Use zerofree to deallocate unused sectors on the volume. On an unpartitioned volume this will be done for the entire volume, while it will only happen on the root partition for partitioned volumes.
- Use vmware-vdiskmanager to shrink the real volume size (only applicable when using vmdk backing). The tool is part of the VMWare Workstation package.
- Tell apt to only download specific language files. See the apt.conf manpage for more details ("Languages" in the "Acquire group" section).
- Configure debootstrap and dpkg to filter out specific paths when installing packages

Settings

- `zerofree`: Specifies if it should mark unallocated blocks as zeroes, so the volume could be better shrunk after this. Valid values: true, false Default: false `optional`
- `shrink`: Whether the volume should be shrunk. This setting works best in conjunction with the zerofree tool. Valid values: true, false Default: false `optional`
- `apt`: Apt specific configurations. `optional`
  - `autoclean`: Configure apt to clean out the archive and cache after every run. Valid values: true, false Default: false `optional`
  - `languages`: List of languages apt should download. Use `[none]` to not download any languages at all. `optional`
  - `gzip_indexes`: Gzip apt package indexes. Valid values: true, false Default: false `optional`
  - `autoremove_suggests`: Suggested packages are removed when running. `apt-get purge --auto-remove` Valid values: true, false Default: false `optional`
- `dpkg`: dpkg (and debootstrap) specific configurations. These settings not only affect the behavior of dpkg when installing packages after the image has been created, but also during the bootstrapping process. This includes the behavior of debootstrap. `optional`
  - `locales`: List of locales that should be kept. When this option is used, all locales (and the manpages in those locales) are excluded from installation excepting the ones in this list. Specify an empty list to not install any locales at all. `optional`
  - `exclude_docs`: Exclude additional package documentation located in `/usr/share/doc` Valid values: true, false Default: false `optional`

NTP

This plugins installs the Network Time Protocol daemon and optionally defines which time servers it should use.

Settings

- `servers`: A list of strings specifying which servers should be used to synchronize the machine clock. `optional`

Open Nebula

This plugin adds OpenNebula contextualization to the image, which sets up the network configuration and SSH keys.

The virtual machine context should be configured as follows:

```
ETH0_DNS       $NETWORK[DNS, NETWORK_ID=2]
ETH0_GATEWAY   $NETWORK[GATEWAY, NETWORK_ID=2]
ETH0_IP        $NIC[IP, NETWORK_ID=2]
ETH0_MASK      $NETWORK[MASK, NETWORK_ID=2]
ETH0_NETWORK   $NETWORK[NETWORK, NETWORK_ID=2]
FILES          path_to_my_ssh_public_key.pub
```

The plugin will install all *.pub* files in the root authorized_keys file. When using the ec2 provider, the USER_EC2_DATA will be executed if present.

Settings

This plugin has no settings. To enable it add `"opennebula":{}` to the plugin section of the manifest.

Pip install

Install packages from the Python Package Index via pip.

Installs `build-essential` and `python-dev` debian packages, so Python extension modules can be built.

Settings

- `packages`: Python packages to install, a list of strings. The list can contain anything that `pip install` would accept as an argument, for example `awscli==1.3.13`.

prebootstrapped

When developing for bootstrap-vz, testing can be quite tedious since the bootstrapping process can take a while. The prebootstrapped plugin solves that problem by creating a snapshot of your volume right after all the software has been installed. The next time bootstrap-vz is run, the plugin replaces all volume preparation and bootstrapping tasks and recreates the volume from the snapshot instead.

The plugin assumes that the users knows what he is doing (e.g. it doesn't check whether bootstrap-vz is being run with a partitioned volume configuration, while the snapshot is unpartitioned).

When no snapshot or image is specified the plugin creates one and outputs its ID/path. Specifying an ID/path enables the second mode of operation which recreates the volume from the specified snapshot instead of creating it from scratch.

Settings

- `snapshot`: ID of the EBS snapshot to use. This setting only works with the volume backing `ebs`.
- `image`: Path to the loopbackvolume snapshot. This setting works with the volume backings `raw`, `s3`, `vdi`, `vmdk`

- `folder`: Path to the folder copy. This setting works with the volume backing `folder`

Puppet

Installs *puppet version 4 <http://puppetlabs.com/>* PC1 From the site repository *<http://apt.puppetlabs.com/>* and optionally applies a manifest inside the chroot. You can also have it copy your puppet configuration into the image so it is readily available once the image is booted.

Rationale and use case

You want to use this plugin when you wish to create an image and to be able to manage that image with Puppet. You have a Puppet 4 setup in mind and thus you want the image to contain the puppet agent software from the puppetlabs repo. You want it to almost contain everything you need to get it up and running This plugin does just that! While you're at it, throw in some modules from the forge as well! Want to include your own modules? Include them as assets!

For now this plugin is only compatible with Debian versions Wheezy and Jessie. These are also the distributions supported by puppetlabs.

Settings

- `manifest`: Path to the puppet manifest that should be applied. `optional`
- `assets`: Path to puppet assets. The contents will be copied into `/etc/puppetlabs` on the image. Any existing files will be overwritten. `optional`
- `install_modules`: A list of modules you wish to install available from *<https://forge.puppetlabs.com/>* inside the chroot. It will assume a FORCED install of the modules. This list is a list of tuples. Every tuple must at least contain the module name. A version is optional, when no version is given, it will take the latest version available from the forge. Format: [module_name (required), version (optional)]
- `enable_agent`: Whether the puppet agent daemon should be enabled. `optional - not recommended`. disabled by default. UNTESTED

An example bootstrap-vz manifest is included in the `KVM` folder of the manifests examples directory.

Limitations

(Help is always welcome, feel free to chip in!) General:

- This plugin only installs the PC1 package for now, needs to be extended to be able to install the package of choice
- The puppetlabs agent is only available to i386 and amd64 architectures. ARM is not available from repository at this time. If you need to have the agent on ARM, you need to build the agent yourself, and install them through the `packages` section of the bootstrap-vz manifest

Manifests:

- Running puppet manifests is not recommended and untested, see below

Assets:

- The assets path must be ABSOLUTE to your manifest file.

install_modules:

- It assumes installing the given list of tuples of modules with the following command: "... install –force $module_name (–version $version_number)" The module name is mandatory, the version is optional. When no version is given, it will pick the master version of the module from <*https://forge.puppetlabs.com/*>
- It assumes the modules are installed into the "production" environment. Installing into another environment e.g. develop, is currently not implemented.
- You cannot include local modules this way, to include you homebrewn modules, You need to inject them through the assets directive.

UNTESTED:

- **Enabling the agent and applying the manifest inside the chrooted environment.** Keep in mind that when applying a manifest when enabling the agent option, the system is in a chrooted environment. This can prevent daemons from running properly (e.g. listening to ports), they will also need to be shut down gracefully (which bootstrap-vz cannot do) before unmounting the volume. It is advisable to avoid starting any daemons inside the chroot at all.

root password

Sets the root password. This plugin removes the task that disables the SSH password authentication.

Settings

- `password`: The password for the root user. `required`

Salt

Install salt minion in the image. Uses salt-bootstrap script to install.

Settings

- `install_source`: Source to install salt codebase from. `stable` for current stable, `daily` for installing the daily build, and `git` to install from git repository. `required`
- `version`: Only needed if you are installing from `git`. `develop` to install current development head, or provide any tag name or commit hash from salt repo `optional`
- `master`: Salt master FQDN or IP `optional`
- `grains`: Set salt grains for this minion. Accepts a map with grain name as key and the grain data as value. `optional`

Unattended upgrades

Enables the unattended update/upgrade feature in aptitude. Enable it to have your system automatically download and install security updates automatically with a set interval.

Settings

- `update_interval`: Days between running `apt-get update`. `required`
- `download_interval`: Days between running `apt-get upgrade --download-only` `required`
- `upgrade_interval`: Days between installing any security upgrades. `required`

Vagrant

Vagrant is a tool to quickly create virtualized environments. It uses "boxes" to make downloading and sharing those environments easier. A box is a tarball containing a virtual volumes accompanied by an OVF specification of the virtual machine.

This plugin creates a vagrant box that is ready to be shared or deployed. At the moment it is only compatible with the VirtualBox provider and doesn't requires any additional settings.

Plugins are a key feature of bootstrap-vz. Despite their small size (most plugins do not exceed 100 source lines of code) they can modify the behavior of bootstrapped systems to a great extent.

Below you will find documentation for all plugins available for bootstrap-vz. If you cannot find what you are looking for, consider developing it yourself and contribute to this list!

CHAPTER 7

Supported builds

The following is a list of supported manifest combinations.

Bootloaders and partitions

Note that grub cannot boot from unpartitioned volumes.

Azure

TODO

EC2

EBS

Bootloader / Partitioning	none	msdos	gpt
pvgrub (paravirtualized)	supported	supported	supported
extlinux (hvm)	supported	supported	supported
grub (hvm)	*not supported*	supported	supported

S3

Bootloader / Partitioning	none	msdos	gpt
pvgrub (paravirtualized)	supported	*not implemented*	*not implemented*
extlinux (hvm)	*not implemented*	*not implemented*	*not implemented*
grub (hvm)	*not supported*	*not implemented*	*not implemented*

GCE

TODO

KVM

TODO

Oracle

TODO

VirtualBox

Bootloader / Partitioning	none	msdos	gpt
extlinux	supported	supported	supported
grub	*not supported*	supported	supported

Known working builds

The following is a list of supported releases, providers and architectures combination. We know that they are working because there's someone working on them.

Release	Provider	Architecture	Person
Jessie	EC2	`amd64`	James Bromberger
Jessie	GCE	`amd64`	Zach Marano (and GCE Team)
Jessie	KVM	`arm64`	Clark Laughlin
Jessie	Oracle	`amd64`	Tiago Ilieve

CHAPTER 8

Logfile

Every run creates a new logfile in the `logs/` directory. The filename for each run consists of a timestamp (`%Y%m%d%H%M%S`) and the basename of the manifest used. The log also contains debugging statements regardless of whether the `--debug` switch was used.

CHAPTER 9

Remote bootstrapping

bootstrap-vz is able to bootstrap images not only on the machine on which it is invoked, but also on remote machines that have bootstrap-vz installed.

This is helpful when you create manifests on your own workstation, but have a beefed up remote build server which can create images quickly. There may also be situations where you want to build multiple manifests that have different providers and require the host machines to be running on that provider (e.g. EBS backed AMIs can only be created on EC2 instances), when doing this multiple times SSHing into the machines and copying the manifests can be a hassle.

Lastly, the main motivation for supporting remote bootstrapping is the automation of system testing. As you will see *further down*, bootstrap-vz is able to select which build server is required for a specific test and run the bootstrapping procedure on said server.

bootstrap-vz-remote

Normally you'd use `bootstrap-vz` to start a bootstrapping process. When bootstrapping remotely simply use `bootstrap-vz-remote` instead, it takes the same arguments plus a few additional ones:

- `--servers <path>`: Path to a list of build-servers (see *build-servers.yml* for more info)
- `--name <name>`: Selects a specific build-server from the list of build-servers
- `--release <release>`: Restricts the autoselection of build-servers to the ones with the specified release

Much like when bootstrapping directly, you can press `Ctrl+C` at any time to abort the bootstrapping process. The remote process will receive the keyboard interrupt signal and begin cleaning up - pressing `Ctrl+C` a second time will abort that as well and kill the connection immediately.

Note that there is also a `bootstrap-vz-server`, this file is not meant to be invoked directly by the user, but is instead launched by bootstrap-vz on the remote server when connecting to it.

Dependencies

For the remote bootstrapping procedure to work, you will need to install bootstrap-vz as well as the `sudo` command on the remote machine. Also make sure that all the needed dependencies for bootstrapping your image are installed.

Locally the pip package Pyro4 is needed.

build-servers.yml

The file `build-servers.yml` informs bootstrap-vz about the different build servers you have at your disposal. In its simplest form you can just add your own machine like this:

```
local:
  type: local
  can_bootstrap: [virtualbox]
  release: jessie
  build_settings: {}
```

`type` specifies how bootstrap-vz should connect to the build-server. `local` simply means that it will call the bootstrapping procedure directly, no new process is spawned.

`can_bootstrap` tells bootstrap-vz for which providers this machine is capable of building images. With the exception of the EC2 provider, the accepted values match the accepted provider names in the manifest. For EC2 you can specify `ec2-s3` and/or `ec2-ebs`. `ec2-ebs` specifies that the machine in question can bootstrap EBS backed images and should only be used when the it is located on EC2. `ec2-s3` signifies that the machine is capable of bootstrapping S3 backed images.

Beyond being a string, the value of `release` is not enforced in any way. It's only current use is for `bootstrap-vz-remote` where you can restrict which build-server should be autoselected.

Remote settings

The other (and more interesting) setting for `type` is `ssh`, which requires a few more configuration settings:

```
local_vm:
  type: ssh
  can_bootstrap:
    - virtualbox
    - ec2-s3
  release: wheezy
  # remote settings below here
  address: 127.0.0.1
  port: 2222
  username: admin
  keyfile: path_to_private_key_file
  server_bin: /root/bootstrap/bootstrap-vz-server
```

The last 5 settings specify how bootstrap-vz can connect to the remote build-server. While the initial handshake is achieved through SSH, bootstrap-vz mainly communicates with its counterpart through RPC (the communication port is automatically forwarded through an SSH tunnel). `address`, `port`, `username` and `keyfile` are hopefully self explanatory (remote machine address, SSH port, login name and path to private SSH key file).

`server_bin` refers to the *abovementioned* bootstrap-vz-server executable. This is the command bootstrap-vz executes on the remote machine to start the RPC server.

Be aware that there are a few limitations as to what bootstrap-vz is able to deal with, regarding the remote machine setup (in time they may be fixed by a benevolent contributor):

- The login user must be able to execute sudo without a password
- The private key file must be added to the ssh-agent before invocation (alternatively it may not be password protected)
- The server must already be part of the known_hosts list (bootstrap-vz uses `ssh` directly and cannot handle interactive prompts)

Build settings

The build settings allow you to override specific manifest properties. This is useful when for example the VirtualBox guest additions ISO is located at `/root/guest_additions.iso` on server 1, while server 2 has it at `/root/images/vbox.iso`.

```
local:
  type: local
  can_bootstrap:
    - virtualbox
    - ec2-s3
  release: jessie
  build_settings:
    guest_additions: /root/images/VBoxGuestAdditions.iso
    apt_proxy:
      address: 127.0.0.1
      port: 3142
    ec2-credentials:
      access-key: AFAKEACCESSKEYFORAWS
      secret-key: thes3cr3tkeyf0ryourawsaccount/FS4d8Qdva
      certificate: /root/manifests/cert.pem
      private-key: /root/manifests/pk.pem
      user-id: 1234-1234-1234
    s3-region: eu-west-1
```

- `guest_additions` specifies the path to the VirtualBox guest additions ISO on the remote machine.
- `apt_proxy` sets the configuration for the *apt_proxy plugin* <*../plugins/apt_proxy*>.
- `ec2-credentials` contains all the settings you know from EC2 manifests.
- `s3-region` overrides the s3 bucket region when bootstrapping S3 backed images.

Run settings

The run settings hold information about how to start a bootstrapped image. This is useful only when running system tests.

```
local:
  type: local
  can_bootstrap:
    - ec2-s3
  release: jessie
  run_settings:
    ec2-credentials:
      access-key: AFAKEACCESSKEYFORAWS
      secret-key: thes3cr3tkeyf0ryourawsaccount/FS4d8Qdva
```

```
docker:
  machine: default
```

- `ec2-credentials` contains the access key and secret key used to boot an EC2 AMI.
- `docker.machine` The docker machine on which an image built for docker should run.

CHAPTER 10

Changelog

2017-02-20

Hugo Antoniio Sepulveda Manriquez:

- **Updated puppet plugin module:**
  - Installs Puppetlabs 4 PC1 agent software from apt.puppetlabs.com
  - Enables you to install modules from forge.puppetlabs.com in the image
  - **Important limitations**
    * Only works for Wheezy and Jessie for now.
    * If you need puppet 3, just add 'puppet' packages provider list.
    * modules: When installing from forge, it assumes 'install –force'
    * modules: When installing from forge, It assumes master version on forge

2016-06-04

Anders Ingemann

- Disable persistent network interface names for >=stretch (by @apolloclark)
- grub defaults and linux boot options are now easier to configure
- Source ixgbevf driver from intel, not sourceforge (by @justinsb)
- Use systemd on jessie (by @JamesBromberger)
- Tune ec2 images (sysctl settings, module blacklisting, nofail in fstab) (by @JamesBromberger)
- Add enable_modules option for cloud-init (by @JamesBromberger)

2016-06-02

Peter Wagner

- Added ec2_publish plugin

2016-06-02

Zach Marano:

- Fix expand-root script to work with newer version of growpart (in jessie-backports and beyond).
- **Overhaul Google Compute Engine image build.**
  - Add support for Google Cloud repositories.
  - Google Cloud SDK install uses a deb package from a Google Cloud repository.
  - Google Compute Engine guest software is installed from a Google Cloud repository.
  - Google Compute Engine guest software for Debian 8 is updated to new refactor.
  - Google Compute Engine wheezy and wheezy-backports manifests are deprecated.

2016-03-03

Anders Ingemann:

- Rename integration tests to system tests

2016-02-23

Nicolas Braud-Santoni:

- #282, #290: Added 'debconf' plugin
- #290: Relaxed requirements on plugins manifests

2016-02-10

Manoj Srivastava:

- #252: Added support for password and static pubkey auth

2016-02-06

Tiago Ilieve:

- Added Oracle Compute Cloud provider
- #280: Declared Squeeze as unsupported

2016-01-14

Jesse Szwedko:

- #269: EC2: Added growpart script extension

2016-01-10

Clark Laughlin:

- Enabled support for KVM on arm64

2015-12-19

Tim Sattarov:

- #263: Ignore loopback interface in udev rules (reduces startup of networking by a factor of 10)

2015-12-13

Anders Ingemann:

- Docker provider implemented (including integration testing harness & tests)
- minimize_size: Added various size reduction options for dpkg and apt
- Removed image section in manifest. Provider specific options have been moved to the provider section. The image name is now specified on the top level of the manifest with "name"
- Provider docs have been greatly improved. All now list their special options.
- All manifest option documentation is now accompanied by an example.
- Added documentation for the integration test providers

2015-11-13

Marcin Kulisz:

- Exclude docs from binary package

2015-10-20

Max Illfelder:

- Remove support for the GCE Debian mirror

2015-10-14

Anders Ingemann:

- Bootstrap azure images directly to VHD

2015-09-28

Rick Wright:

- Change GRUB_HIDDEN_TIMEOUT to 0 from true and set GRUB_HIDDEN_TIMEOUT_QUIET to true.

2015-09-24

Rick Wright:

- Fix a problem with Debian 8 on GCE with >2TB disks

2015-09-04

Emmanuel Kasper:

- Set Virtualbox memory to 512 MB

2015-08-07

Tiago Ilieve:

- Change default Debian mirror

2015-08-06

Stephen A. Zarkos:

- Azure: Change default shell in /etc/default/useradd for Azure images
- Azure: Add boot parameters to Azure config to ease local debugging
- Azure: Add apt import for backports
- Azure: Comment GRUB_HIDDEN_TIMEOUT so we can set GRUB_TIMEOUT
- Azure: Wheezy images use wheezy-backports kernel by default
- Azure: Change Wheezy image to use single partition
- Azure: Update WALinuxAgent to use 2.0.14
- Azure: Make sure we can override grub.ConfigureGrub for Azure images
- Azure: Add console=tty0 to see kernel/boot messages on local console

- Azure: Set serial port speed to 115200
- Azure: Fix error with applying azure/assets/udev.diff

2015-07-30

James Bromberger:

- AWS: Support multiple ENI
- AWS: PVGRUB AKIs for Frankfurt region

2015-06-29

Alex Adriaanse:

- Fix DKMS kernel version error
- Add support for Btrfs
- Add EC2 Jessie HVM manifest

2015-05-08

Alexandre Derumier:

- Fix #219: ^PermitRootLogin regex

2015-05-02

Anders Ingemann:

- Fix #32: Add image_commands example
- Fix #99: rename image_commands to commands
- Fix #139: Vagrant / Virtualbox provider should set ostype when 32 bits selected
- Fix #204: Create a new phase where user modification tasks can run

2015-04-29

Anders Ingemann:

- Fix #104: Don't verify default target when adding packages
- Fix #217: Implement get_version() function in common.tools

2015-04-28

Jonh Wendell:

- root_password: Enable SSH root login

2015-04-27

John Kristensen:

- Add authentication support to the apt proxy plugin

2015-04-25

Anders Ingemann (work started 2014-08-31, merged on 2015-04-25):

- Introduce remote bootstrapping
- Introduce integration testing (for VirtualBox and EC2)
- Merge the end-user documentation into the sphinx docs (plugin & provider docs are now located in their respective folders as READMEs)
- Include READMEs in sphinx docs and transform their links
- Docs for integration testing
- Document the remote bootstrapping procedure
- Add documentation about the documentation
- Add list of supported builds to the docs
- Add html output to integration tests
- Implement PR #201 by @jszwedko (bump required euca2ools version)
- grub now works on jessie
- extlinux is now running on jessie
- Issue warning when specifying pre/successors across phases (but still error out if it's a conflict)
- Add salt dependencies in the right phase
- extlinux now works with GPT on HVM instances
- Take @ssgelm's advice in #155 and copy the mount table – df warnings no more
- Generally deny installing grub on squeeze (too much of a hassle to get working, PRs welcome)
- Add 1 sector gap between partitions on GPT
- Add new task: DetermineKernelVersion, this can potentially fix a lot of small problems
- Disable getty processes on jessie through logind config
- Partition volumes by sectors instead of bytes This allows for finer grained control over the partition sizes and gaps Add new Sectors unit, enhance Bytes unit, add unit tests for both
- Don't require qemu for raw volumes, use *truncate* instead

- Fix #179: Disabling getty processes task fails half the time
- Split grub and extlinux installs into separate modules
- Fix extlinux config for squeeze
- Fix #136: Make extlinux output boot messages to the serial console
- Extend sed_i to raise Exceptions when the expected amount of replacements is not met

Jonas Bergler:

- Fixes #145: Fix installation of vbox guest additions.

Tiago Ilieve:

- Fixes #142: msdos partition type incorrect for swap partition (Linux)

2015-04-23

Tiago Ilieve:

- Fixes #212: Sparse file is created on the current directory

2014-11-23

Noah Fontes:

- Add support for enhanced networking on EC2 images

2014-07-12

Tiago Ilieve:

- Fixes #96: AddBackports is now a common task

2014-07-09

Anders Ingemann:

- Allow passing data into the manifest
- Refactor logging setup to be more modular
- Convert every JSON file to YAML
- Convert "provider" into provider specific section

2014-07-02

Vladimir Vitkov:

- Improve grub options to work better with virtual machines

2014-06-30

Tomasz Rybak:

- Return information about created image

2014-06-22

Victor Marmol:

- Enable the memory cgroup for the Docker plugin

2014-06-19

Tiago Ilieve:

- Fixes #94: allow stable/oldstable as release name on manifest

Vladimir Vitkov:

- Improve ami listing performance

2014-06-07

Tiago Ilieve:

- Download *gsutil* tarball to workspace instead of working directory
- Fixes #97: remove raw disk image created by GCE after build

2014-06-06

Ilya Margolin:

- pip_install plugin

2014-05-23

Tiago Ilieve:

- Fixes #95: check if the specified APT proxy server can be reached

2014-05-04

Dhananjay Balan:

- Salt minion installation & configuration plugin
- Expose debootstrap –include-packages and –exclude-packages options to manifest

2014-05-03

Anders Ingemann:

- Require hostname setting for vagrant plugin
- Fixes #14: S3 images can now be bootstrapped outside EC2.
- Added enable_agent option to puppet plugin

2014-05-02

Tomasz Rybak:

- Added Google Compute Engine Provider

CHAPTER 11

Developers

Contributing

Sending pull requests

Do you want to contribute to the bootstrap-vz project? Nice! Here is the basic workflow:

- Read the *development guidelines*
- Fork this repository.
- Make any changes you want/need.
- Check the coding style of your changes using tox by running *tox -e flake8* and fix any warnings that may appear. This check will be repeated by Travis CI once you send a pull request, so it's better if you check this beforehand.
- If the change is significant (e.g. a new plugin, manifest setting or security fix) add your name and contribution to the changelog.
- Commit your changes.
- Squash the commits if needed. For instance, it is fine if you have multiple commits describing atomic units of work, but there's no reason to have many little commits just because of corrected typos.
- Push to your fork, preferably on a topic branch.
- Send a pull request to the *master* branch.

Please try to be very descriptive about your changes when you write a pull request, stating what it does, why it is needed, which use cases this change covers, etc. You may be asked to rebase your work on the current branch state, so it can be merged cleanly. If you push a new commit to your pull request you will have to add a new comment to the PR, provided that you want us notified. Github will otherwise not send a notification.

Be aware that your modifications need to be properly documented. Please take a look at the *documentation section* to see how to do that.

Happy hacking! :-)

Development guidelines

The following guidelines should serve as general advice when developing providers or plugins for bootstrap-vz. Keep in mind that these guidelines are not rules , they are advice on how to better add value to the bootstrap-vz codebase.

The manifest should always fully describe the resulting image

The outcome of a bootstrapping process should never depend on settings specified elsewhere.

This allows others to easily reproduce any setup other people are running and makes it possible to share manifests. The official debian EC2 images for example can be reproduced using the manifests available in the manifest directory of bootstrap-vz.

The bootstrapper should always be able to run fully unattended

For end users, this guideline minimizes the risk of errors. Any required input would also be in direct conflict with the previous guideline that the manifest should always fully describe the resulting image.

Additionally developers may have to run the bootstrap process multiple times though, any prompts in the middle of that process may significantly slow down the development speed.

The bootstrapper should only need as much setup as the manifest requires

Having to shuffle specific paths on the host into place (e.g. `/target` has to be created manually) to get the bootstrapper running is going to increase the rate of errors made by users. Aim for minimal setup.

Exceptions are of course things such as the path to the VirtualBox Guest Additions ISO or tools like `parted` that need to be installed on the host.

Roll complexity into which tasks are added to the tasklist

If a `run()` function checks whether it should do any work or simply be skipped, consider doing that check in `resolve_tasks()` instead and avoid adding that task altogether. This allows people looking at the tasklist in the logfile to determine what work has been performed.

If a task says it will modify a file but then bails , a developer may get confused when looking at that file after bootstrapping. He could conclude that the file has either been overwritten or that the search & replace does not work correctly.

Control flow should be directed from the task graph

Avoid creating complicated `run()` functions. If necessary, split up a function into two semantically separate tasks.

This allows other tasks to interleave with the control-flow and add extended functionality (e.g. because volume creation and mounting are two separate tasks, the prebootstrapped plugin can replace the volume creation task with a task of its own that creates a volume from a snapshot instead, but still reuse the mount task).

Task classes should be treated as decorated run() functions

Tasks should not have any state, thats what the BootstrapInformation object is for.

Only add stuff to the BootstrapInformation object when really necessary

This is mainly to avoid clutter.

Use a json-schema to check for allowed settings

The json-schema may be verbose but it keeps the bulk of check work outside the python code, which is a big plus when it comes to readability. This only applies as long as the checks are simple. You can of course fall back to doing the check in python when that solution is considerably less complex.

When invoking external programs, use long options whenever possible

This makes the commands a lot easier to understand, since the option names usually hint at what they do.

When invoking external programs, don't use full paths, rely on `$PATH`

This increases robustness when executable locations change. Example: Use `log_call(['wget', ...])` instead of `log_call(['/usr/bin/wget', ...])`.

Coding style

bootstrap-vz is coded to comply closely with the PEP8 style guidelines. There however a few exceptions:

- Max line length is 110 chars, not 80.
- Multiple assignments may be aligned with spaces so that the = match vertically.
- Ignore `E221 & E241`: Alignment of assignments
- Ignore `E501`: The max line length is not 80 characters

The codebase can be checked for any violations quite easily, since those rules are already specified in the tox configuration file.

```
tox -e flake8
```

Documentation

When developing a provider or plugin, make sure to update/create the README.rst located in provider/plugin folder. Any links to other rst files should be relative and work, when viewed on github. For information on how to build the documentation and how the various parts fit together, refer to the documentation about the documentation :-)

Developing plugins

Developing a plugin for bootstrap-vz is a fairly straightforward process, since there is very little code overhead.

The process is the same whether you create an *internal* or an *external* plugin (though you need to add some code for package management when creating an external plugin)

Start by creating an `__init__.py` in your plugin folder. The only obligatory function you need to implement is `resolve_tasks()`. This function adds tasks to be run to the tasklist:

```
def resolve_tasks(taskset, manifest):
    taskset.add(tasks.DoSomething)
```

The manifest variable holds the manifest the user specified, with it you can determine settings for your plugin and e.g. check of which release of Debian bootstrap-vz will create an image.

A task is a class with a static `run()` function and some meta-information:

```
class DoSomething(Task):
    description = 'Doing something'
    phase = phases.volume_preparation
    predecessors = [PartitionVolume]
    successors = [filesystem.Format]

    @classmethod
    def run(cls, info):
        pass
```

To read more about tasks and their ordering, check out the section on how bootstrap-vz works.

Besides the `resolve_tasks()` function, there is also the `resolve_rollback_tasks()` function, which comes into play when something has gone awry while bootstrapping. It should be used to clean up anything that was created during the bootstrapping process. If you created temporary files for example, you can add a task to the rollback taskset that deletes those files, you might even already have it because you run it after an image has been successfully bootstrapped:

```
def resolve_rollback_tasks(taskset, manifest, completed, counter_task):
    counter_task(taskset, tasks.DoSomething, tasks.UndoSomething)
```

In `resolve_rollback_tasks()` you have access to the taskset (this time it contains tasks that will be run during rollback), the manifest, and the tasks that have already been run before the bootstrapping aborted (`completed`).

The last parameter is the `counter_task()` function, with it you can specify that a specific task (2nd param) has to be in the taskset (1st param) for the rollback task (3rd param) to be added. This saves code and makes it more readable than running through the completed tasklist and checking each completed task.

You can also specify a `validate_manifest()` function. Typically it looks like this:

```
def validate_manifest(data, validator, error):
    from bootstrapvz.common.tools import rel_path
    validator(data, rel_path(__file__, 'manifest-schema.yml'))
```

This code validates the manifest against a schema in your plugin folder. The schema is a JSON schema, since bootstrap-vz supports yaml, you can avoid a lot of curly braces quotes:

```
$schema: http://json-schema.org/draft-04/schema#
title: Example plugin manifest
type: object
properties:
  plugins:
    type: object
    properties:
      example:
        type: object
        properties:
          message: {type: string}
        required: [message]
        additionalProperties: false
```

In the schema above we check that the `example` plugin has a single property named `message` with a string value (setting `additionalProperties` to `false` makes sure that users don't misspell optional attributes).

Internal plugins

Internal plugins are part of the bootstrap-vz package and distributed with it. If you have developed a plugin that you think should be part of the package because a lot of people might use it you can send a pull request to get it included (just remember to read the guidelines first).

External plugins

External plugins are packages distributed separately from bootstrap-vz. Separate distribution makes sense when your plugin solves a narrow problem scope specific to your use-case or when the plugin contains proprietary code that you would not like to share. They integrate with bootstrap-vz by exposing an entry-point through `setup.py`:

```
setup(name='example-plugin',
      version=0.9.5,
      packages=find_packages(),
      include_package_data=True,
      entry_points={'bootstrapvz.plugins': ['plugin_name = package_name.module_name']}
↪,
      install_requires=['bootstrap-vz >= 0.9.5'],
      )
```

Beyond `setup.py` the package might need a `MANIFEST.in` so that assets like `manifest-schema.yml` are included when the package is built:

```
include example/manifest-schema.yml
include example/README.rst
```

To test your package from source you can run `python setup.py develop` to register the package so that bootstrap-vz can find the entry-point of your plugin.

An example plugin is available at https://github.com/andsens/bootstrap-vz-example-plugin, you can use it as a starting point for your own plugin.

Installing external plugins

Some plugins may not find their way to the python package index (especially if it's in a private repo). They can of course still be installed using pip:

```
pip install git+ssh://git@github.com/username/repo#egg=plugin_name
```

Documentation

Both the end-user and developer documentation is combined into a single sphinx build (the two were previously split between github pages and sphinx).

Building

To build the documentation, simply run `tox -e docs` in the project root. Serving the docs through http can be achieved by subsequently running `(cd docs/_build/html; python -m SimpleHTTPServer 8080)` and accessing them on `http://localhost:8080/`.

READMEs

Many of the folders in the project have a README.rst which describes the purpose of the contents in that folder. These files are automatically included when building the documentation, through use of the include directive.

Include files for the providers and plugins are autogenerated through the sphinx conf.py script.

Links

All links in rst files outside of `docs/` (but also `docs/README.rst`) that link to other rst files are relative and reference folder names when the link would point at a README.rst otherwise. This is done to take advantage of the github feature where README files are displayed when viewing its parent folder. When accessing the `manifests/` folder for example, the documentation for how manifests work is displayed at the bottom.

When sphinx generates the documentation, these relative links are automatically converted into relative links that work inside the generated html pages instead. If you are interested in how this works, take a look at the link transformation module in `docs/transform_github_links`.

Commandline switches

As a developer, there are commandline switches available which can make your life a lot easier.

- `--debug`: Enables debug output in the console. This includes output from all commands that are invoked during bootstrapping.
- `--pause-on-error`: Pauses the execution when an exception occurs before rolling back. This allows you to debug by inspecting the volume at the time the error occurred.
- `--dry-run`: Prevents the `run()` function from being called on all tasks. This is useful if you want to see whether the task order is correct.

Taskoverview

How bootstrap-vz works

Tasks

At its core bootstrap-vz is based on tasks that perform units of work. By keeping those tasks small and with a solid structure built around them a high degree of flexibility can be achieved. To ensure that tasks are executed in the right order, each task is placed in a dependency graph where directed edges dictate precedence. Each task is a simple class that defines its predecessor tasks and successor tasks via attributes. Here is an example:

```
class MapPartitions(Task):
    description = 'Mapping volume partitions'
    phase = phases.volume_preparation
    predecessors = [PartitionVolume]
    successors = [filesystem.Format]

    @classmethod
    def run(cls, info):
        info.volume.partition_map.map(info.volume)
```

In this case the attributes define that the task at hand should run after the `PartitionVolume` task — i.e. after volume has been partitioned (`predecessors`) — but before formatting each partition (`successors`). It is also placed in the `volume_preparation` phase. Phases are ordered and group tasks together. All tasks in a phase are run before proceeding with the tasks in the next phase. They are a way of avoiding the need to list 50 different tasks as predecessors and successors.

The final task list that will be executed is computed by enumerating all tasks in the package, placing them in the graph and sorting them topologically. Subsequently the list returned is filtered to contain only the tasks the provider and the plugins added to the taskset.

System abstractions

There are several abstractions in bootstrap-vz that make it possible to generalize things like volume creation, partitioning, mounting and package installation. As a rule these abstractions are located in the `base/` folder, where the manifest parsing and task ordering algorithm are placed as well.

CHAPTER 12

API

Base functionality

The base module represents concepts of the bootstrapping process that tasks can interact with and handles the gather, sorting and running of tasks.

Filesystem handling

Volume

class bootstrapvz.base.fs.volume.**Volume**(*partition_map*)

Represents an abstract volume. This class is a finite state machine and represents the state of the real volume.

_before_link_dm_node(*e*)

Links the volume using the device mapper This allows us to create a 'window' into the volume that acts like a volume in itself. Mainly it is used to fool grub into thinking that it is working with a real volume, rather than a loopback device or a network block device.

Parameters **e** (*_e_obj*) – Event object containing arguments to create()

Keyword arguments to link_dm_node() are:

Parameters

- **logical_start_sector** (*int*) – The sector the volume should start at in the new volume
- **start_sector** (*int*) – The offset at which the volume should begin to be mapped in the new volume
- **sectors** (*int*) – The number of sectors that should be mapped

Read more at: http://manpages.debian.org/cgi-bin/man.cgi?query=dmsetup&apropos=0&sektion=0&manpath=Debian+7.0+wheezy&format=html&locale=en

Raises ***VolumeError*** – When a free block device cannot be found.

_before_unlink_dm_node(*e*)
Unlinks the device mapping

_check_blocking(*e*)
Checks whether the volume is blocked

Raises ***VolumeError*** – When the volume is blocked from being detached

Partitionmaps

Abstract Partitionmap

class bootstrapvz.base.fs.partitionmaps.abstract.**AbstractPartitionMap**(*bootloader*)
Abstract representation of a partiton map This class is a finite state machine and represents the state of the real partition map

_before_map(*event*)

Raises ***PartitionError*** – In case a partition could not be mapped.

_before_unmap(*event*)

Raises ***PartitionError*** – If the a partition cannot be unmapped

create(*volume*)
Creates the partition map

Parameters **volume** (Volume) – The volume to create the partition map on

get_total_size()
Returns the total size the partitions occupy

Returns The size of all partitions

Return type Sectors

is_blocking()
Returns whether the partition map is blocking volume detach operations

Return type bool

map(*volume*)
Maps the partition map to device nodes

Parameters **volume** (Volume) – The volume the partition map resides on

unmap(*volume*)
Unmaps the partition

Parameters **volume** (Volume) – The volume to unmap the partition map from

GPT Partitionmap

class bootstrapvz.base.fs.partitionmaps.gpt.**GPTPartitionMap**(*data*, *sector_size*, *bootloader*)
Represents a GPT partition map

_before_create(*event*)
Creates the partition map

MS-DOS Partitionmap

class bootstrapvz.base.fs.partitionmaps.msdos.**MSDOSPartitionMap**(*data*, *sector_size*, *bootloader*)

Represents a MS-DOS partition map Sometimes also called MBR (but that confuses the hell out of me, so ms-dos it is)

No Partitionmap

class bootstrapvz.base.fs.partitionmaps.none.**NoPartitions**(*data*, *sector_size*, *bootloader*)

Represents a virtual 'NoPartitions' partitionmap. This virtual partition map exists because it is easier for tasks to simply always deal with partition maps and then let the base abstract that away.

get_total_size()

Returns the total size the partitions occupy

Returns The size of all the partitions

Return type Sectors

is_blocking()

Returns whether the partition map is blocking volume detach operations

Return type bool

Partitions

Abstract partition

class bootstrapvz.base.fs.partitions.abstract.**AbstractPartition**(*size*, *filesystem*, *format_command*, *mountopts*)

Abstract representation of a partiton This class is a finite state machine and represents the state of the real partition

_after_mount(*e*)

Mount any mounts associated with this partition

_before_format(*e*)

Formats the partition

_before_mount(*e*)

Mount the partition

_before_unmount(*e*)

Unmount any mounts associated with this partition

add_mount(*source*, *destination*, *opts=[]*)

Associate a mount with this partition Automatically mounts it

Parameters

- **source** (*str*, AbstractPartition) – The source of the mount
- **destination** (*str*) – The path to the mountpoint
- **opts** (*list*) – Any options that should be passed to the mount command

get_end()
Gets the end of the partition

Returns The end of the partition

Return type Sectors

get_uuid()
Gets the UUID of the partition

Returns The UUID of the partition

Return type str

remove_mount(*destination*)
Remove a mount from this partition Automatically unmounts it

Parameters **destination** (*str*) – The mountpoint path of the mount that should be removed

Base partition

class bootstrapvz.base.fs.partitions.base.**BasePartition**(*size*, *filesystem*, *format_command*, *mountopts*, *previous*)
Represents a partition that is actually a partition (and not a virtual one like 'Single')

_before_create(*e*)
Creates the partition

create(*volume*)
Creates the partition

Parameters **volume** (Volume) – The volume to create the partition on

get_index()
Gets the index of this partition in the partition map

Returns The index of the partition in the partition map

Return type int

get_start()
Gets the starting byte of this partition

Returns The starting byte of this partition

Return type Sectors

map(*device_path*)
Maps the partition to a device_path

Parameters **device_path** (*str*) – The device path this partition should be mapped to

GPT partition

class bootstrapvz.base.fs.partitions.gpt.**GPTPartition**(*size*, *filesystem*, *format_command*, *mountopts*, *name*, *previous*)
Represents a GPT partition

GPT swap partition

class bootstrapvz.base.fs.partitions.gpt_swap.**GPTSwapPartition**(*size*, *previous*)
Represents a GPT swap partition

MS-DOS partition

class bootstrapvz.base.fs.partitions.msdos.**MSDOSPartition**(*size*, *filesystem*, *format_command*, *mountopts*, *name*, *previous*)
Represents an MS-DOS partition

MS-DOS swap partition

class bootstrapvz.base.fs.partitions.msdos_swap.**MSDOSSwapPartition**(*size*, *previous*)
Represents a MS-DOS swap partition

Single

class bootstrapvz.base.fs.partitions.single.**SinglePartition**(*size*, *filesystem*, *format_command*, *mountopts*)
Represents a single virtual partition on an unpartitioned volume

get_start()
Gets the starting byte of this partition

Returns The starting byte of this partition

Return type Sectors

Unformatted partition

class bootstrapvz.base.fs.partitions.unformatted.**UnformattedPartition**(*size*, *previous*)
Represents an unformatted partition It cannot be mounted

Exceptions

exception bootstrapvz.base.fs.exceptions.**PartitionError**
Raised when an error occurs while interacting with the partitions on the volume

exception bootstrapvz.base.fs.exceptions.**VolumeError**
Raised when an error occurs while interacting with the volume

Package handling

Package list

class bootstrapvz.base.pkg.packagelist.**PackageList**(*manifest_vars*, *source_lists*)
Represents a list of packages

class **Local**(*path*)
A local package

class PackageList.**Remote**(*name*, *target*)
A remote package with an optional target

PackageList.**add**(*name*, *target=None*)
Adds a package to the install list

Parameters

- **name** (*str*) – The name of the package to install, may contain manifest vars references
- **target** (*str*) – The name of the target release for the package, may contain manifest vars references

Raises

- ***PackageError*** – When a package of the same name but with a different target has already been added.
- ***PackageError*** – When the specified target release could not be found.

PackageList.**add_local**(*package_path*)
Adds a local package to the installation list

Parameters **package_path** (*str*) – Path to the local package, may contain manifest vars references

Sources list

class bootstrapvz.base.pkg.sourceslist.**Source**(*line*)
Represents a single source line

class bootstrapvz.base.pkg.sourceslist.**SourceLists**(*manifest_vars*)
Represents a list of sources lists for apt

add(*name*, *line*)
Adds a source to the apt sources list

Parameters

- **name** (*str*) – Name of the file in sources.list.d, may contain manifest vars references
- **line** (*str*) – The line for the source file, may contain manifest vars references

target_exists(*target*)
Checks whether the target exists in the sources list

Parameters **target** (*str*) – Name of the target to check for, may contain manifest vars references

Returns Whether the target exists

Return type bool

Preferences list

class bootstrapvz.base.pkg.preferenceslist.**Preference**(*preference*)
Represents a single preference

class bootstrapvz.base.pkg.preferenceslist.**PreferenceLists**(*manifest_vars*)
Represents a list of preferences lists for apt

add(*name*, *preferences*)
Adds a preference to the apt preferences list

Parameters

- **name** (*str*) – Name of the file in preferences.list.d, may contain manifest vars references
- **preferences** (*object*) – The preferences

Exceptions

exception bootstrapvz.base.pkg.exceptions.**PackageError**
Raised when an error occurrs while handling the packageslist

exception bootstrapvz.base.pkg.exceptions.**SourceError**
Raised when an error occurs while handling the sourceslist

Bootstrap information

class bootstrapvz.base.bootstrapinfo.**BootstrapInformation**(*manifest=None*, *debug=False*)
The BootstrapInformation class holds all information about the bootstrapping process. The nature of the attributes of this class are rather diverse. Tasks may set their own attributes on this class for later retrieval by another task. Information that becomes invalid (e.g. a path to a file that has been deleted) must be removed.

_BootstrapInformation__create_manifest_vars(*manifest*, *additional_vars={}*)
Creates the manifest variables dictionary, based on the manifest contents and additional data.

Parameters

- **manifest** (Manifest) – The Manifest
- **additional_vars** (*dict*) – Additional values (they will take precedence and overwrite anything else)

Returns The manifest_vars dictionary

Return type dict

class bootstrapvz.base.bootstrapinfo.**DictClass**
Tiny extension of dict to allow setting and getting keys via attributes

Manifest

The Manifest module contains the manifest that providers and plugins use to determine which tasks should be added to the tasklist, what arguments various invocations should have etc..

class bootstrapvz.base.manifest.**Manifest**(*path=None*, *data=None*)
This class holds all the information that providers and plugins need to perform the bootstrapping process. All actions that are taken originate from here. The manifest shall not be modified after it has been loaded. Currently,

immutability is not enforced and it would require a fair amount of code to enforce it, instead we just rely on tasks behaving properly.

load_data(*data=None*)
Loads the manifest and performs a basic validation. This function reads the manifest and performs some basic validation of the manifest itself to ensure that the properties required for initalization are accessible (otherwise the user would be presented with some cryptic error messages).

load_modules()
Loads the provider and the plugins.

parse()
Parses the manifest. Well... "parsing" is a big word. The function really just sets up some convenient attributes so that tasks don't have to access information with info.manifest.data['section'] but can do it with info.manifest.section.

schema_validator(*data*, *schema_path*)
This convenience function is passed around to all the validation functions so that they may run a json-schema validation by giving it the data and a path to the schema.

Parameters

- **data** (*dict*) – Data to validate (normally the manifest data)
- **schema_path** (*str*) – Path to the json-schema to use for validation

validate()
Validates the manifest using the provider and plugin validation functions. Plugins are not required to have a validate_manifest function

validation_error(*message*, *data_path=None*)
This function is passed to all validation functions so that they may raise a validation error because a custom validation of the manifest failed.

Parameters

- **message** (*str*) – Message to user about the error
- **data_path** (*list*) – A path to the location in the manifest where the error occurred

Raises **ManifestError** – With absolute certainty

Tasklist

The tasklist module contains the TaskList class.

class bootstrapvz.base.tasklist.**TaskList**(*tasks*)
The tasklist class aggregates all tasks that should be run and orders them according to their dependencies.

run(*info*, *dry_run=False*)
Converts the taskgraph into a list and runs all tasks in that list

Parameters

- **info** (*dict*) – The bootstrap information object
- **dry_run** (*bool*) – Whether to actually run the tasks or simply step through them

bootstrapvz.base.tasklist.**check_ordering**(*task*)
Checks the ordering of a task in relation to other tasks and their phases.

This function checks for a subset of what the strongly connected components algorithm does, but can deliver a more precise error message, namely that there is a conflict between what a task has specified as its predecessors or successors and in which phase it is placed.

Parameters **task** (Task) – The task to check the ordering for

Raises **TaskListError** – If there is a conflict between task precedence and phase precedence

bootstrapvz.base.tasklist.**create_list**(*taskset*, *all_tasks*)
Creates a list of all the tasks that should be run.

bootstrapvz.base.tasklist.**get_all_classes**(*path=None*, *prefix=''*, *excludes=[]*)
Given a path to a package, this function retrieves all the classes in it

Parameters

- **path** (*str*) – Path to the package
- **prefix** (*str*) – Name of the package followed by a dot
- **excludes** (*list*) – List of str matching module names that should be ignored

Returns A generator that yields classes

Return type generator

Raises **Exception** – If a module cannot be inspected.

bootstrapvz.base.tasklist.**get_all_tasks**(*loaded_modules*)
Gets a list of all task classes in the package

Returns A list of all tasks in the package

Return type list

bootstrapvz.base.tasklist.**load_tasks**(*function*, *manifest*, **args*)
Calls `function` on the provider and all plugins that have been loaded by the manifest. Any additional arguments are passed directly to `function`. The function that is called shall accept the taskset as its first argument and the manifest as its second argument.

Parameters

- **function** (*str*) – Name of the function to call
- **manifest** (Manifest) – The manifest
- **args** (*list*) – Additional arguments that should be passed to the function that is called

bootstrapvz.base.tasklist.**strongly_connected_components**(*graph*)
Find the strongly connected components in a graph using Tarjan's algorithm.

Source: http://www.logarithmic.net/pfh-files/blog/01208083168/sort.py

Parameters **graph** (*dict*) – mapping of tasks to lists of successor tasks

Returns List of tuples that are strongly connected comoponents

Return type list

bootstrapvz.base.tasklist.**topological_sort**(*graph*)
Runs a topological sort on a graph.

Source: http://www.logarithmic.net/pfh-files/blog/01208083168/sort.py

Parameters **graph** (*dict*) – mapping of tasks to lists of successor tasks

Returns A list of all tasks in the graph sorted according to ther dependencies

Return type list

Logging

This module holds functions and classes responsible for formatting the log output both to a file and to the console.

class bootstrapvz.base.log.**ColorFormatter**(*fmt=None*, *datefmt=None*)
Colorizes log messages depending on the loglevel

class bootstrapvz.base.log.**FileFormatter**(*fmt=None*, *datefmt=None*)
Formats log statements for output to file Currently this is just a stub

class bootstrapvz.base.log.**SourceFormatter**(*fmt=None*, *datefmt=None*)
Adds a [source] tag to the log message if it exists The python docs suggest using a LoggingAdapter, but that would mean we'd have to use it everywhere we log something (and only when called remotely), which is not feasible.

bootstrapvz.base.log.**get_console_handler**(*debug*, *colorize*)
Returns a log handler for the console The handler color codes the different log levels

Params bool debug Whether to set the log level to DEBUG (otherwise INFO)

Params bool colorize Whether to colorize console output

Returns The console logging handler

bootstrapvz.base.log.**get_file_handler**(*path*, *debug*)
Returns a log handler for the given path If the parent directory of the logpath does not exist it will be created The handler outputs relative timestamps (to when it was created)

Params str path The full path to the logfile

Params bool debug Whether to set the log level to DEBUG (otherwise INFO)

Returns The file logging handler

bootstrapvz.base.log.**get_log_filename**(*manifest_path*)
Returns the path to a logfile given a manifest The logfile name is constructed from the current timestamp and the basename of the manifest

Parameters **manifest_path** (*str*) – The path to the manifest

Returns The path to the logfile

Return type str

Task

class bootstrapvz.base.task.**Task**
The task class represents a task that can be run. It is merely a wrapper for the run function and should never be instantiated.

classmethod **run**(*info*)
The run function, all work is done inside this function

Parameters **info** (BootstrapInformation) – The bootstrap info object.

Phase

class bootstrapvz.base.phase.**Phase**(*name*, *description*)

The Phase class represents a phase a task may be in. It has no function other than to act as an anchor in the task graph. All phases are instantiated in common.phases

pos()

Gets the position of the phase

Returns The positional index of the phase in relation to the other phases

Return type int

Common

The common module contains features that are common to multiple providers and plugins. It holds both a large set of shared tasks and also various tools that are used by both the base module and tasks.

Volume representations

Shared tasks

CHAPTER 13

Testing

Unit tests

System tests

System tests test bootstrap-vz in its entirety. This testing includes building images from manifests and creating/booting said images.

Since hardcoding manifests for each test, bootstrapping them and booting the resulting images is too much code for a single test, a testing harness has been developed that reduces each test to it's bare essentials:

- Combine available *manifest partials* into a single manifest
- Boot an instance from a manifest
- Run tests on the booted instance

In order for the system testing harness to be able to bootstrap it must know about your build-servers. Depending on the manifest that is bootstrapped, the harness chooses a fitting build-server, connects to it and starts the bootstrapping process.

When running system tests, the framework will look for `build-servers.yml` at the root of the repo and raise an error if it is not found.

Manifest combinations

The tests mainly focus on varying key parts of an image (e.g. partitioning, Debian release, bootloader, ec2 backing, ec2 virtualization method) that have been problem areas. Essentially the tests are the cartesian product of these key parts.

Aborting a test

You can press `Ctrl+C` at any time during the testing to abort - the harness will automatically clean up any temporary resources and shut down running instances. Pressing `Ctrl+C` a second time stops the cleanup and quits immediately.

Manifest partials

Instead of creating manifests from scratch for each single test, reusable parts are factored out into partials in the manifest folder. This allows code like this:

```
partials = {'vdi': '{provider: {name: virtualbox}, volume: {backing: vdi}}',
            'vmdk': '{provider: {name: virtualbox}, volume: {backing: vmdk}}',
            }

def test_unpartitioned_extlinux_oldstable():
    std_partials = ['base', 'stable64', 'extlinux', 'unpartitioned', 'root_password']
    custom_partials = [partials['vmdk']]
    manifest_data = merge_manifest_data(std_partials, custom_partials)
```

The code above produces a manifest for Debian stable 64-bit unpartitioned virtualbox VMDK image. `root_password` is a special partial in that the actual password is randomly generated on load.

Missing parts

The system testing harness is in no way complete.

- It still has no support for providers other than Virtualbox, EC2 and Docker.
- Creating an SSH connection to a booted instance is cumbersome and does not happen in any of the tests - this would be particularly useful when manifests are to be tested beyond whether they boot up.

System test providers

Docker

Dependencies

The host machine running the system tests must have docker installed.

EC2

Dependencies

The host machine running the system tests must have the python package `boto` installed (`>= 2.14.0`).

Virtualbox

Dependencies

VirtualBox itself is required on the machine that is running the system tests. The same machine also needs to have python package `vboxapi` (`>=1.0`) installed.

System testing providers are implemented on top of the abstraction that is the testing harness.

Implementation

At their most basic level all they need to implement is the `boot_image()` function, which, when called, boots the image that has been bootstrapped. It should yield something the test can use to ascertain whether the image has been successfully bootstrapped (i.e. a reference to the bootlog or an object with various functions to interact with the booted instance). How this is implemented is up to the individual provider.

A `prepare_bootstrap()` function may also be implemented, to ensure that the bootstrapping process can succeed (i.e. create the AWS S3 into which an image should be uploaded).

Both functions are generators that yield, so that they may clean up any created resources, once testing is done (or failed, so remember to wrap `yield` in a `try:..  finally:..`).

Debugging

When developing a system test provider, debugging through multiple invocations of `tox` can be cumbersome. A short test script, which sets up logging and invokes a specific test can be used instead:

Example:

```
#!/usr/bin/env python

from tests.system.docker_tests import test_stable
from bootstrapvz.base.main import setup_loggers

setup_loggers({'--log': '-', '--color': 'default', '--debug': True})
test_stable()
```

The testing framework consists of two parts: The unit tests and the integration tests.

The unit tests are responsible for testing individual parts of bootstrap-vz, while the integration tests test entire manifests by bootstrapping and booting them.

Selecting tests

To run one specific test suite simply append the module path to tox:

```
$ tox -e unit tests.unit.releases_tests
```

Specific tests can be selected by appending the function name with a colon to the modulepath – to run more than one tests, simply attach more arguments.

```
$ tox -e unit tests.unit.releases_tests:test_lt tests.unit.releases_tests:test_eq
```

CHAPTER 14

bootstrap-vz

bootstrap-vz is a bootstrapping framework for Debian that creates ready-to-boot images able to run on a number of cloud providers and virtual machines. bootstrap-vz runs without any user intervention and generates images for the following virtualization platforms:

- Amazon AWS EC2 (supports both HVM and PVM; S3 and EBS backed; used for official Debian images; *Quick start*)
- Docker (*Quick start*)
- Google Compute Engine (used by Google for official Debian images)
- KVM (Kernel-based Virtual Machine)
- Microsoft Azure
- Oracle Compute Cloud Service (used for official Debian images)
- Oracle VirtualBox (*with Vagrant support*)

Its aim is to provide a reproducible bootstrapping process using manifests as well as supporting a high degree of customizability through plugins.

Documentation

The documentation for bootstrap-vz is available at bootstrap-vz.readthedocs.org. There, you can discover *what the dependencies* for a specific cloud provider are, see a list of available plugins and learn how you create a manifest.

Note to developers: The shared documentation links on github and readthedocs are transformed in a rather peculiar and nifty way.

Installation

bootstrap-vz has a master branch into which stable feature branches are merged.

After checking out the branch of your choice you can install the python dependencies by running `python setup.py install`. However, depending on what kind of image you'd like to bootstrap, there are other debian package dependencies as well, at the very least you will need `debootstrap`. The documentation explains this in more detail.

Note that bootstrap-vz will tell you which tools it requires when they aren't present (the different packages are mentioned in the error message), so you can simply run bootstrap-vz once to get a list of the packages, install them, and then re-run.

Quick start

Here are a few quickstart tutorials for the most common images. If you plan on partitioning your volume, you will need the `parted` package and `kpartx`:

```
root@host:~# apt-get install parted kpartx
```

Note that you can always abort a bootstrapping process by pressing `Ctrl+C`, bootstrap-vz will then initiate a cleanup/rollback process, where volumes are detached/deleted and temporary files removed, pressing `Ctrl+C` a second time shortcuts that procedure, halts the cleanup and quits the process.

Docker

```
user@host:~$ sudo -i # become root
root@host:~# git clone https://github.com/andsens/bootstrap-vz.git # Clone the repo
root@host:~# apt-get install debootstrap python-pip docker.io # Install dependencies␣
↪from aptitude
root@host:~# pip install termcolor jsonschema fysom docopt pyyaml pyrfc3339 # Install␣
↪python dependencies
root@host:~# bootstrap-vz/bootstrap-vz bootstrap-vz/manifests/examples/docker/jessie-
↪minimized.yml
```

The resulting image should be no larger than 82 MB (81.95 MB to be exact). The manifest `jessie-minimized.yml` uses the minimize_size plugin to reduce the image size considerably. Rather than installing docker from the debian main repo it is recommended to install the latest docker version.

VirtualBox Vagrant

```
user@host:~$ sudo -i # become root
root@host:~# git clone https://github.com/andsens/bootstrap-vz.git # Clone the repo
root@host:~# apt-get install qemu-utils debootstrap python-pip # Install dependencies␣
↪from aptitude
root@host:~# pip install termcolor jsonschema fysom docopt pyyaml # Install python␣
↪dependencies
root@host:~# modprobe nbd max_part=16
root@host:~# bootstrap-vz/bootstrap-vz bootstrap-vz/manifests/examples/virtualbox/
↪jessie-vagrant.yml
```

(The *modprobe nbd max_part=16* part enables the network block device driver to support up to 16 partitions on a device)

If you want to use the minimize_size plugin, you will have to install the `zerofree` package and VMWare Workstation as well.

Amazon EC2 EBS backed AMI

```
user@host:~$ sudo -i # become root
root@host:~# git clone https://github.com/andsens/bootstrap-vz.git # Clone the repo
root@host:~# apt-get install debootstrap python-pip # Install dependencies from␣
↪aptitude
root@host:~# pip install termcolor jsonschema fysom docopt pyyaml boto3 # Install␣
↪python dependencies
root@host:~# bootstrap-vz/bootstrap-vz bootstrap-vz/manifests/official/ec2/ebs-jessie-
↪amd64-hvm.yml
```

To bootstrap S3 backed AMIs, bootstrap-vz will also need the `euca2ools` package. However, version 3.2.0 is required meaning you must install it directly from the eucalyptus repository like this:

```
apt-get install --no-install-recommends python-dev libxml2-dev libxslt-dev gcc zlib1g-
↪dev
pip install git+git://github.com/eucalyptus/euca2ools.git@v3.2.0
```

Cleanup

bootstrap-vz tries very hard to clean up after itself both if a run was successful but also if it failed. This ensures that you are not left with volumes still attached to the host which are useless. If an error occurred you can simply correct the problem that caused it and rerun everything, there will be no leftovers from the previous run (as always there are of course rare/unlikely exceptions to that rule). The error messages should always give you a strong hint at what is wrong, if that is not the case please consider opening an issue and attach both the error message and your manifest (preferably as a gist or similar).

Dependencies

bootstrap-vz has a number of dependencies depending on the target platform and the selected plugins. At a bare minimum the following python libraries are needed:

- termcolor
- fysom
- jsonschema
- docopt
- pyyaml

To bootstrap Debian itself debootstrap is needed as well.

Any other requirements are dependent upon the manifest configuration and are detailed in the corresponding sections of the documentation. Before the bootstrapping process begins however, bootstrap-vz will warn you if a requirement has not been met.

Developers

The API documentation, development guidelines and an explanation of bootstrap-vz internals can be found at bootstrap-vz.readthedocs.org.

Contributing

Contribution guidelines are described in the documentation under Contributing. There's also a topic regarding the coding style.

Before bootstrap-vz

bootstrap-vz was coded from scratch in python once the bash script architecture that was used in the build-debian-cloud bootstrapper reached its limits. The project has since grown well beyond its original goal, but has kept the focus on Debian images.

Python Module Index

b

Index

Symbols

A

B

T

U

V