cloudbase-init Documentation

Release 1.0

Cloudbase Solutions Srl

Apr 21, 2020

Contents

1 Intro **3**
- 1.1 Portable cloud initialization service . . . 3
- 1.2 Binaries . . . 3

2 Tutorial **5**
- 2.1 Sysprepping . . . 5
- 2.2 Configuration file . . . 5
- 2.3 File execution . . . 6

3 Services **7**
- 3.1 OpenStack (web API) . . . 7
- 3.2 OpenStack (configuration drive) . . . 8
- 3.3 Amazon EC2 . . . 9
- 3.4 CloudStack . . . 9
- 3.5 OpenNebula . . . 9
- 3.6 Ubuntu MaaS . . . 10
- 3.7 Configuring available services . . . 10

4 Plugins **11**
- 4.1 Setting host name *(main)* . . . 11
- 4.2 Creating user *(main)* . . . 11
- 4.3 Setting password *(main)* . . . 12
- 4.4 Static networking *(main)* . . . 12
- 4.5 Saving public keys *(main)* . . . 12
- 4.6 Volume expanding *(main)* . . . 13
- 4.7 WinRM listener *(main)* . . . 13
- 4.8 WinRM certificate *(main)* . . . 13
- 4.9 Scripts execution *(main)* . . . 13
- 4.10 Licensing *(main)* . . . 14
- 4.11 Clock synchronization *(pre-networking)* . . . 14
- 4.12 MTU customization *(pre-metadata-discovery)* . . . 14
- 4.13 User data *(main)* . . . 15
- 4.14 Configuring selected plugins . . . 15

5 Userdata **17**
- 5.1 PEM certificate . . . 17
- 5.2 Batch . . . 17

5.3 PowerShell . . . 17
5.4 Bash . . . 18
5.5 Python . . . 18
5.6 EC2 format . . . 18
5.7 Cloud config . . . 18
5.8 Multi-part content . . . 19
5.9 Sysnativeness . . . 19

6 Indices and tables **21**

Index **23**

Contents:

CHAPTER 1

Intro

The open source project **cloudbase-init** is a service conceived and maintained by Cloudbase Solutions Srl, currently working on NT systems. It was designed to initialize and configure guest operating systems under OpenStack, OpenNebula, CloudStack, MaaS and many others. Under Cloudbase page, beta and stable installers can be found and the service itself is very easy to configure through configuration files. It can also customize instances based on user input like local scripts and data.

More details on how you can use this can be found under *Tutorial*.

1.1 Portable cloud initialization service

The main goal of this project is to provide guest cloud initialization for *Windows* and other operating systems. The architecture of the project is highly flexible and allows extensions for additional clouds and plugins.

There's no limitation in the type of supported hypervisors. This service can be used on instances running on Hyper-V, KVM, Xen, ESXi etc.

1.2 Binaries

The following x64 and x86 builds are automatically generated by a Jenkins job at every commit:

- https://www.cloudbase.it/downloads/CloudbaseInitSetup_x64.msi
- https://www.cloudbase.it/downloads/CloudbaseInitSetup_x86.msi

CHAPTER 2

Tutorial

First, download your desired type of installer from *here*, then install it and fill in configuration options which suits you best. Based on the current selected *cloudbase-init* installer architecture, it'll be available under *C:\Program Files* or *C:\Program Files (x86)* as **Cloudbase Solutions\Cloudbase-Init** directory. There, are located some folders of interest like:

- bin - Executable files and other binaries.
- conf - Configuration files holding miscellaneous options.
- log - Here are the cloudbase-init logs.
- LocalScripts - User supplied *scripts*.
- Python - Bundle of executable and library files to support Python scripts and core execution.

After install, cloudbase-init acts like a 2-step service which will read metadata using *Services* and will pass that to the executing *Plugins*, this way configuring all the supported things. Depending on the platform, some plugins may request reboots.

2.1 Sysprepping

The System Preparation (Sysprep) tool prepares an installation of Windows for duplication, auditing, and customer delivery. Duplication, also called imaging, enables you to capture a customized Windows image that you can reuse throughout an organization. The Sysprep phase uses the "Unattend.xml" which implies the service to run using the "cloudbase-init-unattend.conf" configuration file.

2.2 Configuration file

In the chosen installation path, under the *conf* directory, are present two config files named "cloudbase-init.conf" and "cloudbase-init-unattend.conf". These can hold various config options for picking up the desired available services and plugins ready for execution and also customizing user experience.

Explained example of configuration file:

```
[DEFAULT]
# What user to create and in which group(s) to be put.
username=Admin
groups=Administrators
inject_user_password=true  # Use password from the metadata (not random).
# Which devices to inspect for a possible configuration drive (metadata).
config_drive_raw_hhd=true
config_drive_cdrom=true
# Path to tar implementation from Ubuntu.
bsdtar_path=C:\Program Files (x86)\Cloudbase Solutions\Cloudbase-Init\bin\bsdtar.exe
# Logging debugging level.
verbose=true
debug=true
# Where to store logs.
logdir=C:\Program Files (x86)\Cloudbase Solutions\Cloudbase-Init\log\
logfile=cloudbase-init-unattend.log
default_log_levels=comtypes=INFO,suds=INFO,iso8601=WARN
logging_serial_port_settings=
# Enable MTU and NTP plugins.
mtu_use_dhcp_config=true
ntp_use_dhcp_config=true
# Where are located the user supplied scripts for execution.
local_scripts_path=C:\Program Files (x86)\Cloudbase Solutions\Cloudbase-
↪Init\LocalScripts\
# Services that will be tested for loading until one of them succeeds.
metadata_services=cloudbaseinit.metadata.services.configdrive.ConfigDriveService,
                  cloudbaseinit.metadata.services.httpservice.HttpService,
                  cloudbaseinit.metadata.services.ec2service.EC2Service,
                  cloudbaseinit.metadata.services.maasservice.MaaSHttpService
# What plugins to execute.
plugins=cloudbaseinit.plugins.common.mtu.MTUPlugin,
        cloudbaseinit.plugins.common.sethostname.SetHostNamePlugin
# Miscellaneous.
allow_reboot=false    # allow the service to reboot the system
stop_service_on_exit=false
```

The "cloudbase-init-unattend.conf" configuration file is similar to the default one and is used by the Sysprepping phase. It was designed for the scenario where the minimum user intervention is required and it only runs the MTU and host name plugins, leaving the image ready for further initialization cases.

More of these explained options are available under the *Services*, *Plugins* and *Userdata* documentation.

2.3 File execution

Cloudbase-init has the ability to execute user provided scripts, usually found in the default path *C:\Program Files (x86)\Cloudbase Solutions\Cloudbase-Init\LocalScripts*, through a specific *plugin* for doing this stuff. Depending on the platform used, the files should be valid MZPEs, PowerShell, Python, Batch or Bash scripts, containing the actual code. The user data plugin is also capable of executing various script types and return code value handling.

Based on their return codes, you can instruct the system to reboot or even re-execute the plugin on the next boot:

- 1001 - reboot and don't run the plugin again on next boot
- 1002 - don't reboot now and run the plugin again on next boot
- 1003 - reboot and run the plugin again on next boot

CHAPTER 3

Services

A **metadata service** has the role of pulling the guest provided data (configuration information) and exposing it to the *Plugins* for a general and basic initialization of the instance. These sub-services can change their behavior according to custom configuration options, if they are specified, which are documented below.

Supported metadata services (cloud specific):

3.1 OpenStack (web API)

class cloudbaseinit.metadata.services.httpservice.**HttpService**

A complete service which also supports password related capabilities and can be usually accessed with http://169.254.169.254/ magic address, which can also be changed using *metadata_base_url* option under the config file. A default value of *True* for *add_metadata_private_ip_route* option is used to add a route for the IP address to the gateway. This is needed for supplying a bridge between different VLANs in order to get access to the web server.

Capabilities:

- instance ID
- host name
- public keys
- authentication certificates (metadata + user data)
- static network configuration addresses
- admin password
- user data
- user content (additional files)
- ability to post passwords

Config options:

- metadata_base_url (string: "http://169.254.169.254/")
- add_metadata_private_ip_route (bool: True)

3.2 OpenStack (configuration drive)

class cloudbaseinit.metadata.services.configdrive.**ConfigDriveService**

This is similar to the web API, but it "serves" its files locally without requiring network access. The data is generally retrieved from a cdrom, vfat or *raw* disks/partitions by enabling selective lookup across different devices. Use the *config_drive_types* option to specify which types of config drive content the service will search for and also on which devices using the *config_drive_locations* option.

Warning: *deprecated options*

Using the option:

a. *config_drive_cdrom*

b. *config_drive_raw_hhd*

c. *config_drive_vfat*

It will search for metadata:

a. in mounted optical units

b. directly in the physical disk bytes

c. by exploring the physical disk as a vfat drive; which requires *mtools* (specified by the *mtools_path* option)

The interesting part with this service is the fact that is quite fast in comparison with the HTTP twin.

Capabilities:

- instance ID
- host name
- public keys (search in the entire metadata)
- authentication certificates (metadata + user data)
- static network configuration addresses
- admin password
- user data
- user content (additional files)

Config options:

- config_drive_types (list: ["vfat", "iso"])
- config_drive_locations (list: ["cdrom", "hdd", "partition"])
- mtools_path (string: None)

3.3 Amazon EC2

class cloudbaseinit.metadata.services.ec2service.**EC2Service**

This is similar to the OpenStack HTTP service but is using a different format for URLs and is having general capabilities.

Capabilities:

- instance ID
- host name
- public keys

Config options:

- ec2_metadata_base_url (string: "http://169.254.169.254/")
- ec2_add_metadata_private_ip_route (bool: True)

Note: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

3.4 CloudStack

class cloudbaseinit.metadata.services.cloudstack.**CloudStack**

Another web-based service which usually uses "10.1.1.1" or DHCP addresses for retrieving content.

Capabilities:

- instance ID
- host name
- public keys
- admin password (retrieval/deletion/polling)
- user data

Config options:

- cloudstack_metadata_ip (string: "10.1.1.1")

Note: By design, this service can update the password anytime, so it will cause the *setuserpassword* plugin to run at every boot and by security concerns, the password is deleted right after retrieval and no updating will occur until a new password is available on the server.

3.5 OpenNebula

class cloudbaseinit.metadata.services.opennebulaservice.**OpenNebulaService**

The *OpenNebula* provider is related to configuration drive and searches for a specific context file which holds all the available info. The provided details are exposed as bash variables gathered in a shell script.

Capabilities:

- instance ID (not present; usually a constant is returned)
- host name
- public keys
- static network configuration addresses
- user data

3.6 Ubuntu MaaS

class cloudbaseinit.metadata.services.maasservice.**MaaSHttpService**

This one works with instances on bare metal and uses web requests for retrieving the available exposed metadata. It uses OAuth to secure the requests.

Capabilities:

- instance ID
- host name
- public keys
- authentication certificates (x509)
- user data

3.7 Configuring available services

Some of these classes can be specified manually in the configuration file under *metadata_services* option. Based on this option, the service loader will search across these providers and try to load the most suitable one.

For more details on doing this, see *configuration* file in *Tutorial*.

CHAPTER 4

Plugins

Plugins execute actions based on the metadata obtained by the loaded service. They are intended to actually configure your instance using data provided by the cloud and even by the user. There are three stages for the plugins execution:

1. The *pre-networking* stage (for setting up the network, before doing any valid web request).

2. The *pre-metadata-discovery* one (additional configuration before the metadata service discovery).

3. The *main* set of plugins, which holds the rest of the plugins which are (re)executed according to their saved status.

Note that the first two stages (1,2) are executed each time the service starts.

Current list of supported plugins:

4.1 Setting host name *(main)*

class cloudbaseinit.plugins.common.sethostname.**SetHostNamePlugin**

Sets the instance's hostname. It may be truncated to 15 characters for Netbios compatibility reasons using the option below.

Config options:

- netbios_host_name_compatibility (bool: True)

Warning: This may require a system restart.

4.2 Creating user *(main)*

class cloudbaseinit.plugins.windows.createuser.**CreateUserPlugin**

Creates (or updates if existing) a new user and adds it to a set of provided local groups. By default, it creates the user "Admin" under "Administrators" group, but this can be changed under configuration file.

Config options:

- username (string: "Admin")
- groups (list of strings: ["Administrators"])

4.3 Setting password *(main)*

class cloudbaseinit.plugins.windows.setuserpassword.**SetUserPasswordPlugin**

Sets the cloud user's password. If a password has been provided in the metadata during boot it will be used, otherwise a random password will be generated, encrypted with the user's SSH public key and posted to the metadata provider (currently supported only by the OpenStack HTTP metadata provider). An option called *inject_user_password* is set *True* by default to make available the use of metadata password which is found under the "admin_pass" field or through an URL request. If the option is set to *False* or if the password isn't found in metadata, then an attempt of using an already set password is done (usually a random value by the *createuser* plugin). With *first_logon_behaviour* you can control what happens with the password at the next logon. If this option is set to "always", the user will be forced to change the password at the next logon. If it is set to "clear_text_injected_only", the user will be forced to change the password only if the password is a clear text password, coming from the metadata. The last option is "no", when the user is never forced.

Config options:

- inject_user_password (bool: True)
- first_logon_behaviour (string: "clear_text_injected_only")

Note: This plugin can run multiple times (for posting the password if the metadata service supports this).

4.4 Static networking *(main)*

class cloudbaseinit.plugins.common.networkconfig.**NetworkConfigPlugin**

Statically configures each network adapter for which corresponding details are found into metadata. The details/addresses association is done using MAC matching and if this fails, then name or interface index matching. The basic setting is based on IPv4 addresses, but it supports IPv6 addresses too if they are enabled and exposed to the metadata. The purpose of this plugin is to configure network adapters, for which the DHCP server is disabled, to have internet access and static IPs.

Warning: This may require a system restart.

4.5 Saving public keys *(main)*

class cloudbaseinit.plugins.common.sshpublickeys.**SetUserSSHPublicKeysPlugin**

Creates an **authorized_keys** file in the user's home directory containing the SSH keys provided in the metadata. It is needed by the plugin responsible for encrypting and setting passwords.

Config options:

- username (string: "Admin")

4.6 Volume expanding *(main)*

class cloudbaseinit.plugins.windows.extendvolumes.**ExtendVolumesPlugin**

Extends automatically a disk partition to its maximum size. This is useful when booting images with different flavors. By default, all the volumes are extended, but you can select specific ones by populating with their indexes the *volumes_to_extend* option.

Config options:

- volumes_to_extend (list of integers: None)

Note: This plugin will run at every boot.

4.7 WinRM listener *(main)*

class cloudbaseinit.plugins.windows.winrmlistener.**ConfigWinRMListenerPlugin**

Configures a WinRM HTTPS listener to allow remote management via WinRM or PowerShell.

Config options:

- winrm_enable_basic_auth (bool: True)

Note: This plugin will run until a full and proper configuration will take place.

4.8 WinRM certificate *(main)*

class cloudbaseinit.plugins.windows.winrmcertificateauth.**ConfigWinRMCertificateAuthPlugin**

Enables password-less authentication for remote management via WinRS or PowerShell. Usually uses x509 embedded with UPN certificates.

Config options:

- username (string: "Admin")

Note: http://www.cloudbase.it/windows-without-passwords-in-openstack/

4.9 Scripts execution *(main)*

class cloudbaseinit.plugins.common.localscripts.**LocalScriptsPlugin**

Executes any script (powershell, batch, python etc.) located in the following path indicated by *local_scripts_path* option. More details about the supported scripts and content can be found in *Tutorial* on *file execution* subject.

Config options:

- local_scripts_path (string: None)

Warning: This may require a system restart.

Note: This plugin may run multiple times (depending on the script(s) return code).

4.10 Licensing *(main)*

class cloudbaseinit.plugins.windows.licensing.**WindowsLicensingPlugin**

Activates the Windows instance if the *activate_windows* option is *True*.

Config options:

- activate_windows (bool: False)

4.11 Clock synchronization *(pre-networking)*

class cloudbaseinit.plugins.windows.ntpclient.**NTPClientPlugin**

Applies NTP client info based on the DHCP server options, if available. This behavior is enabled only when the *ntp_use_dhcp_config* option is set to *True* (which by default is *False*).

Config options:

- ntp_use_dhcp_config (bool: False)

Note: This plugin will run until the NTP client is configured.

4.12 MTU customization *(pre-metadata-discovery)*

class cloudbaseinit.plugins.common.mtu.**MTUPlugin**

Sets the network interfaces MTU based on the value provided by the DHCP server options, if available and enabled (by default is *True*). This is particularly useful for cases in which a lower MTU value is required for networking (e.g. OpenStack GRE Neutron Open vSwitch configurations).

Config options:

- mtu_use_dhcp_config (bool: True)

Note: This plugin will run at every boot.

4.13 User data *(main)*

class cloudbaseinit.plugins.common.userdata.**UserDataPlugin**

Executes custom scripts provided by user data metadata as plain text or compressed with Gzip. More details, examples and possible formats here: *Userdata*.

4.14 Configuring selected plugins

By default, all plugins are executed, but a custom list of them can be specified through the *plugins* option in the configuration file.

For more details on doing this, see *configuration* file in *Tutorial*.

CHAPTER 5

Userdata

The *userdata* is the user custom content exposed to the guest instance by the currently deployed and running cloud infrastructure. Its purpose is to provide additional data for the instance to customize it as much as you need, if the cloud initialization service does support this feature.

Fortunately, *cloudbase-init* is able to interpret and use this kind of user specific data in multiple ways. In most of the cases, the thing that indicates of what type is the processed data is usually the **first line**.

Currently supported contents:

5.1 PEM certificate

—–BEGIN CERTIFICATE—–

This one should start with a PEM specific beginning header, which will be eventually parsed by the *configuration drive* and *web API* OpenStack services and used by the *WinRM certificate (main)* plugin for storing and using it.

5.2 Batch

rem cmd

The file is executed in a *cmd.exe* shell (can be changed with the *COMSPEC* environment variable).

5.3 PowerShell

#ps1_sysnative (system native)

#ps1_x86 (Windows On Windows 32bit)

Execute PowerShell scripts using the desired executable. For finding out more about the system nativeness thing, click *here*.

5.4 Bash

#!/bin/bash

A bash shell needs to be installed in the system and available in the *PATH* in order to use this feature.

5.5 Python

#!/usr/bin/env python

Python is available by default with the build itself, but also it must be in the system *PATH*.

5.6 EC2 format

There is no "first line" here, but the content should follow a XML pattern with valid Batch/PowerShell script contents under **script** or **powershell** enclosing tags like in this example:

```
<script>
set root=%SystemDrive%
echo ec2dir>%root%\ec2file.txt
</script>

<powershell>
$root = $env:SystemDrive
$dname = Get-Content "$root\ec2file.txt"
New-Item -path "$root\$dname" -type directory
</powershell>
```

Note: http://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/UsingConfig_WinAMI.html

5.7 Cloud config

#cloud-config

Cloud-config YAML configuration as supported by *cloud-init*, excluding Linux specific content. The following cloud-config directives are supported:

- write_files - Defines a set of files which will be created on the local filesystem. It can be a list of items or only one item, with the following attributes:
    1. path - Absolute path on disk where the content should be written.
    2. content - The content which will be written in the given file.
    3. permissions - Integer representing file permissions.
    4. encoding - The encoding of the data in content. Supported encodings are: b64, base64 for base64-encoded content, gz, gzip for gzip encoded content, gz+b64, gz+base64, gzip+b64, gzip+base64 for base64 encoded gzip content.

  Examples:

```
# One item
write_files:
   encoding: b64
   content: NDI=
   path: C:\test
   permissions: '0o466'
```

```
# Multiple items
write_files:
   -   encoding: b64
       content: NDI=
       path: C:\b64
       permissions: '0644'
   -   encoding: base64
       content: NDI=
       path: C:\b64_1
       permissions: '0644'
   -   encoding: gzip
       content: !!binary |
           H4sIAGUfoFQC/zMxAgCIsCQyAgAAAA==
       path: C:\gzip
       permissions: '0644'
```

- set_timezone - Change the underlying timezone.

  Example:

```
set_timezone: Asia/Tbilisi
```

- set_hostname - Override the already default set host name value. (metadata)

  Example:

```
set_hostname: newhostname
```

5.8 Multi-part content

MIME multi-part user data is supported. The content will ne handled based on the content type.

- text/x-shellscript - Any script to be executed: PowerShell, Batch, Bash or Python.
- text/part-handler - A script that can manage other content type parts. This is used in particular by Heat / CFN templates, although Linux specific.
- text/x-cfninitdata - Heat / CFN content. Written to the path provided by *heat_config_dir* option which defaults to "C:\cfn". (examples of Heat Windows templates)

5.9 Sysnativeness

When deciding which path to use for system executable files...

On 32bit OSes, the return value will be the *System32* directory, which contains 32bit programs. On 64bit OSes, the return value may be different, depending on the Python bits and the *sysnative* parameter. If the Python interpreter is

32bit, the return value will be *System32* (containing 32bit programs) if *sysnative* is set to False and *Sysnative* otherwise. But if the Python interpreter is 64bit and *sysnative* is False, the return value will be *SysWOW64* and *System32* for a True value of *sysnative*.

Why this behavior and what is the purpose of *sysnative* parameter?

On a 32bit OS the things are clear, there is one *System32* directory containing 32bit applications and that's all. On a 64bit OS, there's a *System32* directory containing 64bit applications and a compatibility one named *SysWOW64* (WindowsOnWindows) containing the 32bit version of them. Depending on the Python interpreter's bits, the *sysnative* flag will try to bring the appropriate version of the system directory, more exactly, the physical *System32* or *SysWOW64* found on disk. On a WOW case (32bit interpreter on 64bit OS), a return value of *System32* will point to the physical *SysWOW64* directory and a return value of *Sysnative*, which is consolidated by the existence of this alias, will point to the real physical *System32* directory found on disk. If the OS is still 64bit and there is no WOW case (that means the interpreter is 64bit), the system native concept is out of discussion and each return value will point to the physical location it intends to.

On a 32bit OS the *sysnative* parameter has no meaning, but on a 64bit one, based on its value, it will provide a real/alias path pointing to system native applications if set to True (64bit programs) and to system compatibility applications if set to False (32bit programs). Its purpose is to provide the correct system paths by taking into account the Python interpreter bits too, because on a 32bit interpreter version, *System32* is not the same with the *System32* on a 64bit interpreter. Also, using a 64bit interpreter, the *Sysnative* alias will not work, but the *sysnative* parameter will take care to return *SysWOW64* if you explicitly want 32bit applications, by setting it to False.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index

C

cloudbaseinit.metadata.services.cloudstack.CloudStack
(*built-in class*), 9
cloudbaseinit.metadata.services.configdrive.ConfigDriveService
(*built-in class*), 8
cloudbaseinit.metadata.services.ec2service.EC2Service
(*built-in class*), 9
cloudbaseinit.metadata.services.httpservice.HttpService
(*built-in class*), 7
cloudbaseinit.metadata.services.maasservice.MaaSHttpService
(*built-in class*), 10
cloudbaseinit.metadata.services.opennebulaservice.OpenNebulaService
(*built-in class*), 9
cloudbaseinit.plugins.common.localscripts.LocalScriptsPlugin
(*built-in class*), 13
cloudbaseinit.plugins.common.mtu.MTUPlugin
(*built-in class*), 14
cloudbaseinit.plugins.common.networkconfig.NetworkConfigPlugin
(*built-in class*), 12
cloudbaseinit.plugins.common.sethostname.SetHostNamePlugin
(*built-in class*), 11
cloudbaseinit.plugins.common.sshpublickeys.SetUserSSHPublicKeysPlugin
(*built-in class*), 12
cloudbaseinit.plugins.common.userdata.UserDataPlugin
(*built-in class*), 15
cloudbaseinit.plugins.windows.createuser.CreateUserPlugin
(*built-in class*), 11
cloudbaseinit.plugins.windows.extendvolumes.ExtendVolumesPlugin
(*built-in class*), 13
cloudbaseinit.plugins.windows.licensing.WindowsLicensingPlugin
(*built-in class*), 14
cloudbaseinit.plugins.windows.ntpclient.NTPClientPlugin
(*built-in class*), 14
cloudbaseinit.plugins.windows.setuserpassword.SetUserPasswordPlugin
(*built-in class*), 12
cloudbaseinit.plugins.windows.winrmcertificateauth.ConfigWinRMCertificateAuthPlugin
(*built-in class*), 13
cloudbaseinit.plugins.windows.winrmlistener.ConfigWinRMListenerPlugin
(*built-in class*), 13