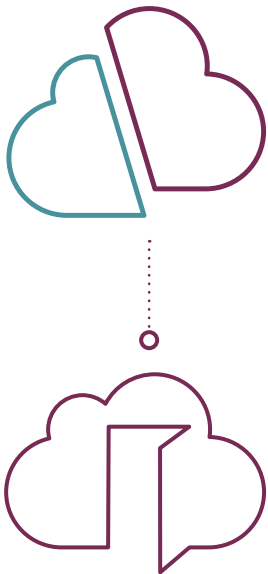


Cloud Instance Initialisation with cloud-init

March 2019



Executive Summary

As a technology business decision maker you have a great deal of choice today when it comes to both open source operating systems and cloud execution environments (or just clouds) to run those operating systems on. We are using the term 'cloud' here in its widest possible sense: from private cloud (i.e. an automated data centre), through hybrid to public cloud.

A common desire is to select the operating system which best supports your application stack and then be able to use that operating system as an abstraction layer between different clouds.

However, in order to function together an operating system and its cloud must share some critical configuration data which tells the operating system how to 'behave' in the particular environment. By separating configuration data, one can avoid the tight coupling of operating system and cloud, or to use the more common industry term 'locked-in'.

Cloud-init provides a mechanism for separating out configuration data from both the operating system and the execution environment so that you maintain the ability to change either at any time. It serves as a useful abstraction which ensures that the investments you make in your application stack on a specific operating system can be leveraged across multiple clouds.



Introduction

Cloud-init is the industry standard method for cross-platform cloud instance initialisation. It is supported by all major public cloud providers, provisioning systems for private cloud infrastructure, and bare-metal installation. It enables the automated provisioning of an operating system image (which is simply the collection of packages) into a fully running state, complete with required access and applications. Typical tasks handled by cloud-init include networking and storage initialisation, optional package installation and configuration, user account creation and security key provisioning for remote access. cloud-init can perform these tasks within the first boot of an instance. This prevents end users from needing to spend time producing and maintaining images that are out-of-date, require building, and maintenance. Instead cloud-init does run in most operating systems each time a new instance is launched, independently from image source.

This level of configuration is provided by cloud-init across all major Linux distributions and FreeBSD. With current support provided in the following.

- Ubuntu
- SLES/openSUSE
- RHEL/CentOS
- Fedora
- Gentoo Linux
- Debian
- ArchLinux
- FreeBSD

Cloud-init provides support across a wide ranging list of execution environments:

 Public Cloud Infrastructure (IaaS)	 Private Cloud Infrastructure
Amazon Web Services	Bare metal install
Microsoft Azure	OpenStack
Google Cloud Platform	LXD
Oracle Cloud Infrastructure	KVM
Softlayer	Metal-as-a-Service (MAAS)
Rackspace Public Cloud	
IBM Cloud	
Digital Ocean	
Bigstep	
Hetzner	
Joyent	
CloudSigma	
Alibaba Cloud	
OVH	

Every cloud is different and each platform provides information in a different format. Cloud-init takes the work out by standardising the platform interactions and abstracting the cloud's implementation away from the user. Cloud-init has hooks to various init systems to perform a four-stage initialisation, described later. These stages are specific to an instance running in a 'cloud-like' environment and during this time, items such as storage, networking, DNS names, user accounts, access keys, and software packages can be setup.

Cloud-init is written in Python and as mentioned is packaged for all major distributions of Linux. It is dual licenced under the GPLv3 and Apache 2.0 licenses.

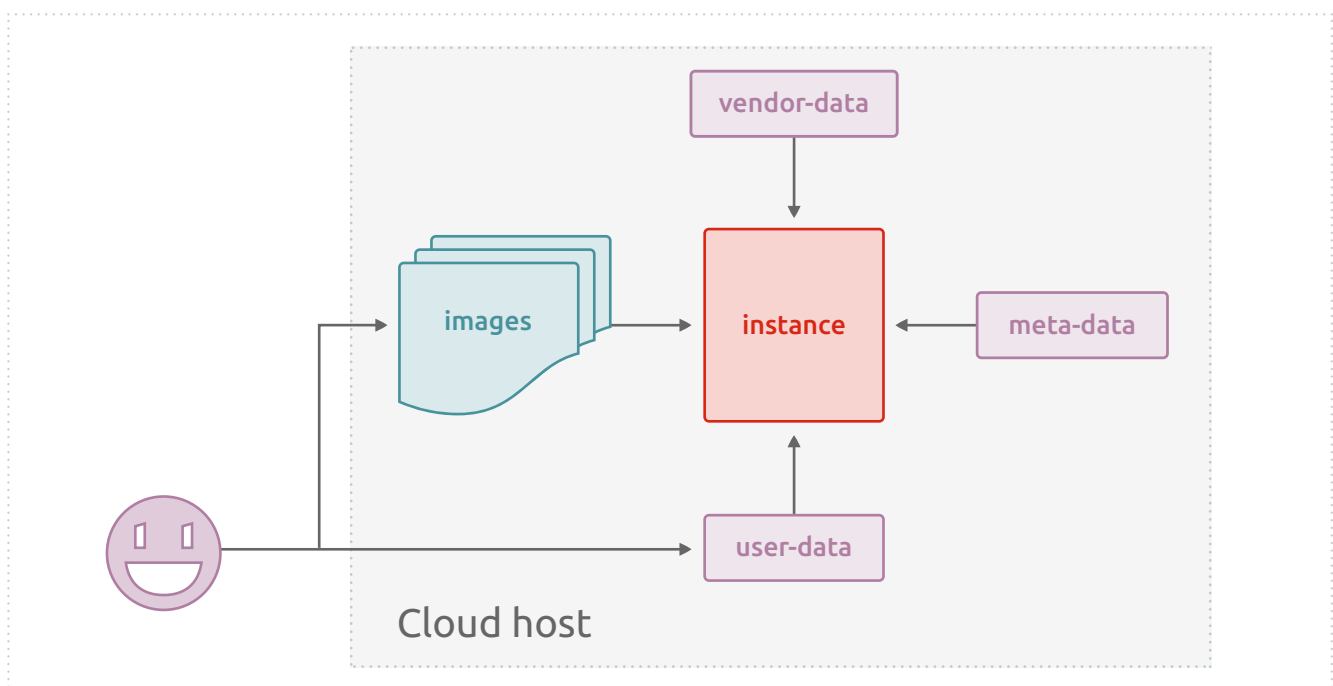
The following document will present end users with details of how a cloud instance is initialised from a disk image and the steps cloud-init takes on the image during boot.

Instance Initialisation

When a user launches a new instance on a cloud, how does the instance get provisioned and configured to enable remote access by the user? Cloud-init is the solution to allow the user and cloud to initialise the instance.

An individual running system (or instance) on a cloud is typically composed of the following parts:


- Disk Image
- Metadata
- User data (optional)
- Vendor data (optional)



Disk Image

A disk image contains the operating system for the instance. This allows the user to quickly and easily pick a necessary image and start deploying, rather than needing to create and upload an image before use. Many operating systems offer images on cloud platforms with cloud-init pre-installed.

Below is an example of an Ubuntu Server 18.04 LTS disk image found on Amazon Web Services.



Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0bbe6b35405ecebdb (64-bit x86) / ami-0db180c518750ee4f (64-bit Arm)

Free tier eligible

Ubuntu Server 18.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select

☒ 64-bit (x86)
☐ 64-bit (Arm)

Metadata

Each cloud provider produces metadata, which is information to aid in setting up the instance. Depending on what data the cloud exposes the output can contain information about the disk image, storage, networking, users, and other information.

The source and format of the metadata varies from cloud to cloud. Cloud-init refers to the cloud's metadata source as the datasource. Some clouds utilise a drive attached to the instance (e.g. Microsoft Azure and LXD) and other times it is an API endpoint (e.g. Amazon Web Services, Google Compute Engine, OpenStack) reached over the network.

A user can view the exposed metadata from the datasource after deployment using cloud-init's query subcommand.

Amazon Web Services example on next page...

Metadata

Amazon Web Services
example of exposed
metadata.

```
$ cloud-init query ds.meta_data
{
  "ami_id": "ami-0fa829a9258e37841",
  "ami_launch_index": "0",
  "ami_manifest_path": "(unknown)",
  "block_device_mapping": {
    "ami": "sda1",
    "ephemeral0": "sdb",
    "ephemeral1": "sdc",
    "root": "/dev/sda1"
  },
  "hostname": "ip-172-31-35-4.us-west-2.compute.internal",
  "instance_action": "none",
  "instance_id": "i-02a8b6bf43af16ad7",
  "instance_type": "i3.metal",
  "local_hostname": "ip-172-31-35-4.us-west-2.compute.internal",
  "local_ipv4": "172.31.35.4",
  "mac": "06:d5:8f:f6:90:fe",
  "metrics": {
    "vhostmd": "<?xml version='1.0' encoding='UTF-8'>"
  },
  "network": {
    "interfaces": {
      "macs": {
        "06:d5:8f:f6:90:fe": {
          "device_number": "0",
          "interface_id": "eni-0f60aeebf0e09575d",
          "ipv4_associations": {
            "54.245.12.52": "172.31.35.4"
          }
        },
        "local_hostname": "ip-172-31-35-4.us-west-2.compute.internal",
        "local_ipv4s": "172.31.35.4",
        "mac": "06:d5:8f:f6:90:fe",
        "owner_id": "022588101030",
        "public_hostname": "ec2-54-245-12-52.us-west-2.compute.amazonaws.com",
        "public_ipv4s": "54.245.12.52",
        "security_group_ids": "sg-e6ef7d80",
        "security_groups": "default",
        "subnet_id": "subnet-71b23707",
        "subnet_ipv4_cidr_block": "172.31.32.0/20",
        "vpc_id": "vpc-9bf381ff",
        "vpc_ipv4_cidr_block": "172.31.0.0/16",
        "vpc_ipv4_cidr_blocks": "172.31.0.0/16"
      }
    }
  },
  "placement": {
    "availability_zone": "us-west-2a"
  },
  "profile": "default-hvm",
  "public_hostname": "ec2-54-245-12-52.us-west-2.compute.amazonaws.com",
  "public_ipv4": "54.245.12.52",
  "public_keys": {
    "powershell": [
      "ssh-rsa // Your public key"
    ]
  },
  "reservation_id": "r-00661ad53885c04b8",
  "security_groups": "default",
  "services": {
    "domain": "amazonaws.com",
    "partition": "aws"
  }
}
```

User Data

Data provided by end users to initialise the instance is called user data. This is the mechanism in which users interact with cloud-init. This data can range from simple shell scripts to more structured “cloud config” that utilise cloud-init’s built-in config modules. This type of data is completely optional and not required for an instance initialisation.

As an example, user data can be as simple as a basic shell script.

```
#!/bin/sh
sudo apt-get update
sudo apt-get install --yes pastebinit
cloud-init query userdata | pastebinit
```

For more advanced configuration, cloud-init also has built in config modules that enable setting up specific parts of system. These modules are used via a cloud-config based user data and includes special configuration options to configure the following and more.

Example configuration tasks	
Set hostname Add SSH keys	Add users and groups Partition Disks
Update and upgrade package software	Configure LXD
Run arbitrary code	Grow the root partition
Start Puppet or Chef	Phone home
Setup timezone and locale	Mirror selection

Customisation of these modules is available via “cloud config”. Cloud config is a YAML syntax file prefaced with ‘#cloud-config’ at the top of the file. The same shell script above written like cloud-config is shown below which demonstrates how instead of needing to provide numerous ‘apt’ or ‘dnf’ commands the user can provide a list of packages to install and cloud-init will install those packages.

```
#cloud-config
hostname: my_cloud_instance
packages:
  pastebinit
runcmd:
  cloud-init query userdata | pastebinit
```

Cloud-init's cloud-config also provides the ability to utilise Jinja templating. The user can make use of the previously mentioned metadata in their own user data to make decisions on how to setup an instance. This allows for custom user data settings depending on the cloud, region, or even storage and networking configurations.

```
## template: jinja
#cloud-config
{% set NAME='system-' ~ v1.platform ~ '-' ~ v1.region ~ '-' ~ range(9) | random %}
hostname: {{ NAME }}

{% if v1.cloud_name == 'aws' and v1.region == 'us-east-2' -%}
echo Installing custom packages
...
{%- endif %}
```

The mechanism to provide user data varies by cloud. However, most provide a method via the UI as well as via some API or CLI mechanism as well.

Below is an example found on Microsoft's Azure Cloud Computing Platform launch screen.

CLOUD INIT

Cloud init is a widely used approach to customize a Linux VM as it boots for the first time. You can use cloud-init to install packages and write files or to configure users and security. [Learn more](#)

Cloud init

Review + create

Previous

Next : Tags >

Cloud provided command-line tools to launch instances have user data flags to pass a user data file during instances creation.

```
aws ec2 run-instances --image-id ami-abc1234 --count 1 \  
  --instance-type m4.large --key-name keypair \  
  --subnet-id subnet-abcd1234 --security-group-ids sg-abcd1234 \  
  --user-data file:///user_data.yaml
```

Finally, a user can view the passed in user data with cloud-init's query subcommand.

```
$ cloud-init query userdata  
#cloud-config  
users:  
- name: powersj  
  ssh-import-id: lp:powersj  
  shell: /bin/bash  
  sudo: ['ALL=(ALL) NOPASSWD:ALL']
```

Vendor Data

Vendor data is the same as user data, however, it comes from the cloud platform vendor itself. User data directives always takes precedence over vendor data. The value of vendor data is it allows the cloud vendor to specify potentially platform or region specific values for instances. For example, setting up package mirrors, NTP servers, etc. Also note that vendor data is completely optional and not required for configuring an instance.

```
$ cloud-init query vendordata  
#cloud-config  
{}
```


The four stages of cloud-init

During the boot process cloud-init executes four stages. Each of these stages are setup to allow cloud-init to configure the instance from very early on to allow for network and storage configurations to occur before the rest of the system is configured.

The following will describe each stage and what configuration occurs during:

1. Init Local

The first stage runs as early as possible in order to allow cloud-init to generate network configuration and activate networking. As such, this is run as soon as the root filesystem is mounted read-write. Cloud-init will go and search for any local data source objects that may allow it to obtain metadata to configure the system. If any information is received cloud-init will proceed to setup networking devices, including bridges, bonds, vlans, etc.

2. Init

The second stage runs after networking is up and searches for any network data sources to find metadata for the system. This allows for custom storage configurations and expanding disk volumes to occur very early on. At this stage cloud-init will also make any changes it is requested to make to the block devices as well as setup file systems. Finally, the user-data for the system is obtained for use during the next stage.

3. Modules Config

The third stage follows immediately after the previous one to guaranteed networking and begins running the cloud-init config modules. Here SSH keys are imported, apt or yum is configured, and time related services are configured.

4. Modules Final

The final stage will then run at the very end of the boot process. This stage involves package install and configuration via user-scripts. Puppet and chef tools are setup and configured, and finally the final message module is run.

Summary

Cloud-init is an essential tool for cloud instance initialisation. It provides the ability to setup and configure instances across clouds and distributions. Instances can be configured with special storage and networking and with customised software configurations.