

Utilising cloud-init on Microsoft Azure

June 2020

Introduction

Cloud-init, available in Ubuntu images by default, is utilised for the initialisation of cloud instances. It enables the automated provisioning of an operating system image into a fully running state. Typical tasks handled by cloud-init include networking and storage initialisation via metadata from the cloud provider and optional package installation and configuration via data from the user.

Cloud-init can complete these tasks within the first boot of an instance. This prevents users from needing to spend time producing and maintaining images that are out-of-date, requiring repeated building, and maintenance of images.

This guide will demonstrate how to take advantage of cloud-init on Microsoft's Azure public cloud with Ubuntu.

Microsoft Azure

Microsoft Azure is an ever-expanding set of cloud services to help organisations meet their business challenges. It's the freedom to build, manage and deploy applications on a massive, global network using your favourite tools and frameworks.

Azure is actively working with endorsed Linux distro partners to have cloud-init enabled images by default in the Azure marketplace. For more information, see [Azure's support for cloud-init](#) page.

Metadata

To help initialise new instances, Azure provides metadata that describes how to setup and configure the instance based on inputs from the user. The metadata is generated and provided by the [Azure Instance Metadata Service](#) (IMDS) to the instance. Cloud-init reads that metadata on boot and then configures the system as appropriate.

Cloud-init has a query subcommand that allows the user to see what metadata was passed to the system. Below is an example from an Azure instance running Ubuntu 18.04 LTS:

```
{
  "azure_data": {
    "configurationsettype": "LinuxProvisioningConfiguration"
  },
  "imds": {
    "compute": {
      "location": "westus2",
      "name": "cloud-init-test",
      "offer": "UbuntuServer",
      "osType": "Linux",
      "placementGroupId": "",
      "platformFaultDomain": "0",
      "platformUpdateDomain": "0",
      "publisher": "Canonical",
      "resourceGroupName": "my-resrc-grp",
      "sku": "18.04-LTS",
      "subscriptionId": "7f84....6d754",
      "tags": "",
      "version": "18.04.201908131",
      "vmId": "76926b28-f7c4-4951-937f-307baae8b3d6",
      "vmScaleSetName": "",
      "vmSize": "Standard_D2s_v3",
      "zone": ""
    },
    "network": {
      "interface": [
        {
          "ipv4": {
            "ipAddress": [
              {
                "privateIpAddress": "10.0.0.5",
                "publicIpAddress": "52.183.119.125"
              }
            ],
            "subnet": [
              {
                "address": "10.0.0.0",
                "prefix": "24"
              }
            ]
          },
          "ipv6": {
            "ipAddress": []
          },
          "macAddress": "000D3AFD40C0"
        }
      ]
    },
    "instance_id": "286b9276-c4f7-5149-937f-307baae8b3d6",
    "local_hostname": "cloud-init-test",
    "random_seed": "T0VNMGQAAAAB...sqIKE8pzEefDQ4iHSVA=="
  }
}
```

The above is a simple system with only a single public and private IP address pair. More advanced network configuration can be used on Azure and cloud-init can prepare the system as requested.

Having the ability to query the data is very helpful when debugging or development of a deployment. The next section will demonstrate how a user can take advantage of values found in metadata with cloud-init to further customise an instance.

User data

Data provided by users to initialise the instance is called user data. Providing user data allows for automation, customisation, and repeated deployments of cloud instances. This data can exist in numerous formats such as a shell script or cloud-config, a declarative syntax for common configuration operations. However, user data is completely optional and not required for an instance initialisation.

Similar to metadata, a user can query for the user data that was passed to an instance. This does require privileged access as the contents of the user data could contain passwords or other sensitive data:

```
$ sudo cloud-init query userdata
```

Cloud-config

Users can provide cloud-config to an instance and have an array of declarative syntax configuration options available to them. Options exist for installing packages, setting up software archives, creating users and groups, configuring the system, and many other common operations. The options simplify configuration and reduce the amount of development by the user.

Cloud-config is specified by adding the `#cloud-config` header on the first line. In the following example of cloud-config, cloud-init would set the hostname, fully update the system, and install a pair of additional packages and any required dependencies:

```
#cloud-config
hostname: azure01
package_update: true
package_upgrade: true
packages:
  - certbot
  - tree
```

See the cloud-init [cloud-config documentation](#) for more details on the wide breadth of available options.

Templating

User data can utilise Jinja templating to customise actions or values when using cloud-init. By using templating, a user can specify conditions and replacements based on a particular scenario. A user can also take advantage of metadata values, like the cloud or region name and even make use of the assigned public and private IP addresses to customise software configurations.

In the example below, a shell script passed as user data would install custom software only if the instance is on the Azure uswest2 region. Note the `##` template: jinja is required as the first line to tell cloud-init to treat this as a template to run substitution:

```
## template: jinja
#!/bin/bash
{% if v1.region == 'uswest2' -%}
echo 'Installing custom config for {{ v1.region }}'
...
{%- endif %}
```

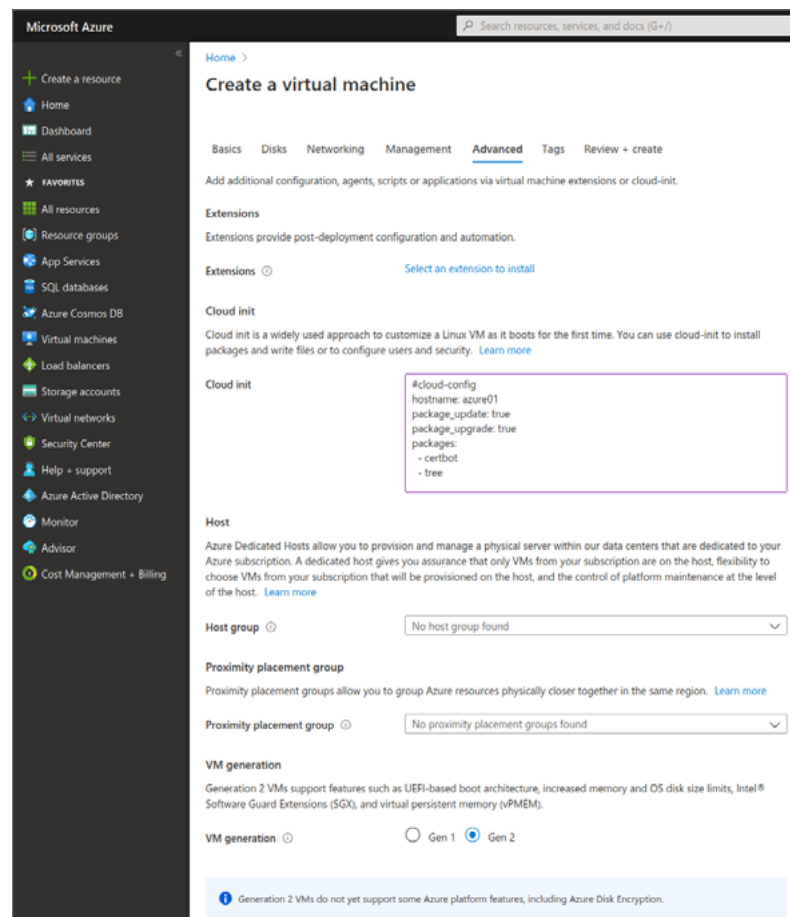
The cloud-init metadata documentation has further details on templating.

Azure + cloud-init

The following section demonstrates how Azure users can use cloud-init with the Azure marketplace, Azure CLI tool, and Azure SDK for Python. To pass a cloud-init config during a VM creation, use the customData VM Create property, this is an immutable, and has a maximum size of 64KB.

Azure Marketplace

To specify user data when launching instances via the Azure marketplace, navigate to the Advanced tab when creating a virtual machine. On the new page, go down to the cloud-init section. There a user can paste in any valid user data, such as the aforementioned cloud-config or a shell script:



The screenshot shows the 'Create a virtual machine' page in the Azure portal, specifically the 'Advanced' tab. The 'Cloud init' section is highlighted with a purple box, showing a sample cloud-config script. The script sets the hostname to 'azure01', enables package updates and upgrades, and installs certbot and tree.

```
#cloud-config
hostname: azure01
package_update: true
package_upgrade: true
packages:
- certbot
- tree
```

The cloud-init config you pass in will be automatically converted to Base64, as required by the API.

Cloud-init will retrieve the user data from an Azure's metadata source during the instance launch. Cloud-init will then process and execute the user data as appropriate to initialise the system.

Note, if this option is deactivated, this means the image is not cloud-init enabled yet.

Azure CLI

Launching VMs via the CLI using the [Azure CLI](#) tool also accepts user data, the AZ CLI will automatically convert the input into Base64. In the example below, the user data is provided in a file called user-data.yaml. The file is passed to the vm create command using the --custom-data flag:

```
$ az group create --location westus2 --name my-resrc-grp
$ az vm create --resource-group=my-resrc-grp --name=cloud-init-test \
--admin-username=$USER --ssh-key-value @$HOME/.ssh/id_rsa.pub \
--image UbuntuLTS --custom-data user-data.yaml
```

Azure Resource Manager (ARM)

If you are deploying Azure resources using ARM deployments, you can pass user data via the customData parameter within the osProfile section of the VM resource:

```
“name”: “[parameters(‘virtualMachineName’)]”,
“type”: “Microsoft.Compute/virtualMachines”,
“apiVersion”: “2019-07-01”,
“location”: “[parameters(‘location’)]”,
“dependsOn”: [
..],
“properties”: {
..
  “osProfile”: {
    “computerName”: “[parameters(‘virtualMachineName’)]”,
    “adminUsername”: “[parameters(‘adminUsername’)]”,
    “adminPassword”: “[parameters(‘adminPassword’)]”,
    “customData”: “[parameters(‘customData’)]”
  },
```

The customData parameter expects a Base64 value, you can convert your cloud-init config to Base64, or you can use the ARM [Base64](#) function to convert a string to Base64. You should also adhere to [ARM security best practices](#) when passing in user-data.

Terraform

Terraform supports creating Azure VMs with passing user data via `custom_data` parameter in the OS profile section. Internally, Terraform will Base64 encode this value before sending it to the API. The maximum length of the binary array is 65535 bytes:

```
os_profile {
  computer_name = "myvm"
  admin_username = "azureuser"
  custom_data = file("user-data.yaml")
}

os_profile_linux_config {
  disable_password_authentication = true
  ssh_keys {
    path      = "/home/azureuser/.ssh/authorized_keys"
    key_data = "ssh-rsa AAAAB3Nz{snip}hwhqT9h"
  }
}
```

Azure SDK for Python

For even faster and more customisation, a user can write deployment scripts using the [Azure SDK for Python](#).

To pass user data the VM creation [parameters](#) need to include the `customData` key as [specified in the docs](#). The user data needs to be encoded as a Base64 string to properly pass through. While the web and CLI methods handle this for the user automatically, the user is required to this when using the Azure SDK:

```
resrc_grp_name = 'my-resrc-grp'
vm_name = 'cloud-init-test'
vm_parameters = { ... }

vm_parameters['customData'] = base64.b64encode(user_data.encode())
vm = compute_client.virtual_machines.create_or_update(
    resrc_grp_name, vm_name, vm_parameters
)
```

Summary

By using cloud-init to initialise cloud instances, deployments are done more quickly and efficiently. Cloud-init's ability to customise the initialisation through custom templates adds even more flexibility. Finally, Azure makes using cloud-init easy as numerous methods have the ability to make use of passed in user data.

Learn more:

- [Cloud-init website](#)
- [Cloud-init documentation](#)
- [Cloud-init support for virtual machines in Azure](#)