Simple Azure Documentation

Release 0.2

Hyungro Lee

Aug 23, 2017

Contents

Simple Azure is a Python library for Microsoft Azure Services including Virtual Machine (VM) to provision resources in a simple way. Infrastructure provisioning is supported now with the new Azure Resource Manager (ARM) Templates, therefore you can describe and share your application with infrastructure in a JSON format template to reproduce same environments. Launching classic virtual machines is supported using the Azure Service Management (ASM) API which is now called classic or legacy mode.

Simple Azure includes 407 community templates[1] from Azure QuickStart Templates to deploy software and infrastructure ranging from a simple linux VM deployment (i.e. 101-vm-simple-linux) to Azure Container Service cluster with a DC/OS orchestrator (i.e. 101-acs-dcos). You can import, export, search, modify, review and deploy these templates using Simple Azure and get information about deployed services in resource groups. Initial scripts or automation tools can be triggered after a completion of deployements therefore your software stacks and applications are installed and configured to run your jobs or start your services.

A classic virtual machine service is supported with the azure-sdk-for-python legacy package to create a single virtual machine (VM) and multiple VMs.

Simple Azure is currently in a development stage therefore new features will be added from time to time and issues and bugs might be easily found while you use the tool. Check out the latest version from the github repository. Documentation is also actively updated.

[1] as of 10/13/2016 from https://github.com/Azure/azure-quickstart-templates

CHAPTER 1

Deploying a Template in Simple Azure ARM Mode

Starting a single Linux VM with SSH key from Azure QuickStart Template is:

```
>>> from simpleazure import SimpleAzure
>>> saz = SimpleAzure()

# aqst is for Azure QuickStart Templates
>>> vm_sshkey_template = saz.aqst.get_template('101-vm-sshkey')

# arm is for Azure Resource Manager
>>> saz.arm.set_template(vm_sshkey_template)
>>> saz.arm.set_parameter("sshKeyData", "ssh-rsa AAAB... hrlee@quickstart")
>>> saz.arm.deploy()
```

Starting a sample VM from a custom template URL is:

```
>>> url = "https://raw.githubusercontent.com/Azure-Samples/resource-manager-python-
↪template-deployment/master/templates/template.json"
>>> saz.arm.deploy(template = url, param = { "sshKeyData": "ssh-rsa AAAB3Nza...",
↪'dnsLabelPrefix':"simpleazure", 'vmName':'simpleazure-first-vm'}) })
```

Note: For more about using ARM? check out *Quick Setup for Azure Resource Manager Mode*

Note: For more about deploying a custom Template? check out *Deploying Azure Virtual Machine in Simple Azure ARM Mode*

Note: For more about deploying Azure QuickStart Templates? check out *Deploying Azure QuickStart Templates*

CHAPTER 2

Docker Image

Simple Azure is available in a Docker image to run.

- With IPython Notebook:

```
docker run -d -p 8888:8888 lee212/simpleazure_with_ipython
```

Open a browser with the port number **8888**.

- Simple Azure only:

```
docker run -i -t lee212/simpleazure
```

CHAPTER 3

Installation

From github.com:

```
git clone https://github.com/lee212/simpleazure.git
cd simpleazure
pip install -r requirements.txt
python setup.py install
```

from Pypi:

```
pip install simpleazure
```

CHAPTER 4

Deploying Azure Virtual Machines (classic mode)

Three lines are required to deploy Window Azure Virtual Machine in Simple Azure.

```
from simpleazure.simpleazure import SimpleAzure as saz

azure = saz()
azure.create_vm()
```

CHAPTER 5

Caveats

Simple Azure was started in 2013 but wasn't consistently updated which means that some dated features may not work as expected. Relax, I am trying to get Simple Azure back on track after these abandoned moments, so please report any issues that you may encounter. I will try to fix or sort it out as quickly as possible I can.

Not supported features:

- Python 3 is NOT supported

Obsolete features (might be revived later):

- virtual cluster
- IPython cluster with the plugin
- Access to the open VM image repository (VM Depot)

If you are looking for a classic mode launching a virtual machine, you can get started with *Quickstart* and then learn more through *Tutorial* that shows how to deploy and utilize Azure Virtual Machines with Simple Azure. *Installation* and *Configuration* helps you get Simple Azure installed on your machine and *Command* describes how to use Simple Azure on the python shell. You can find resources here.

CHAPTER 6

Contribution

- issues

CHAPTER 7

Contents

Quick Setup for Azure Resource Manager Mode

Azure Resource Template (JSON based Infrastructure as Code) runs with Azure Resource Manager which Azure Python SDK 2.0.0+ supports with new packages and functions. Simple Azure uses the new version of Azure Python SDK to deploy software and infrastructure with Azure Resource Templates.

Previous development is now called 'legacy' or 'classic' mode of Azure Python SDK with limited features (although it still works to start or terminate Azure Virtual Machines).

This document explains a few changes of using ARM mode and describes how to setup account credentials differently compared to the classic mode. It is also worth to mention that guidelines and instructions from Azure official document or other online articles are insufficient to follow, this is understandable because ARM supports with Azure Python SDK is fairly new (as of September 2016) and some Azure services are in 'preview' mode.

Installation of Azure Python SDK

From Pypi:

```
pip install --pre azure
```

If you already have the azure package but need to upgrade then add `-U` option:

```
pip install --pre azure -U
```

If you are looking for the latest development, probably downloading code from github.com would be best:

```
git clone git://github.com/Azure/azure-sdk-for-python.git
cd azure-sdk-for-python
python setup.py install
```

Additional Packages

From Pypi:

```
pip install msrest
pip install msrestazure
```

From github.com repository:

```
pip install -r requirements.txt
```

You may encounter some errors like this, if you don't install additional packages:

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "/usr/local/lib/python2.7/site-packages/azure/common/credentials.py", line 25,␣
↪in <module>
raise ImportError("You need to install 'msrest' to use this feature")
ImportError: You need to install 'msrest' to use this feature
```

Authentication with Service Principal Credentials (ServicePrincipalCredentials)

Similar to AWS IAM service, Azure allows users to have resource access through Active Directory and Service Principal credentials which only require (encrypted) key strings such as clientID, secretKey or tenantID instead of certificate files generated by openssl e.g. .pem. Let's walk through SDK functions to see how it works.

`ServicePrincipalCredentials()` from `azure.common.credentials` requires three arguments: `client_id`, `secret`, and `tenant` to authenticate.

`client_id` is an unique application id, `secret` is a encrypted key string registered to the application and `tenant` is an unique user id.

Getting these values is explained from here: https://azure.microsoft.com/en-us/documentation/articles/resource-group-create-service-principal-portal/ but app registrations are not described entirely because Admin consent needs to be done additionally. Otherwise, registered apps are not visible in the subscriptions page to add access with Roles.

Note: Remember `client_id`, `secret` and `tenant` values including subscription id because these values are required to authenticate in Simple Azure ARM mode.

Reconsent Step

Follow the steps below:

- Go to the classic portal
- Select 'Active Directory' and find 'applications' tab at the top of the page
- Search apps by selecting 'Applications my company owns' in the search box
- Select your application and find 'Users and Groups' tab at the top of the page
- `Reconsent` if the page asks like `Admin consent is required prior to assigning users and groups. You can consent via the application by clicking here:`

With Azure CLI

It is easier to create a new app and a service principal with access to your subscriptions via Azure CLI. The official documentation is here: https://azure.microsoft.com/en-us/documentation/articles/resource-group-authenticate-service-principal-cli/

The two commands complete this step like:

```
$ azure ad sp create -n <app name> -p <password> --home-page <http or https url> --
↪identifier-uris <http or https url>
$ azure role assignment create --objectId <uuid returned from previous command> -o
↪<Role e.g. Owner or Reader> -c /subscriptions/<subscription ID>/
```

ServicePrincipalCredentials()

Try to authenticate with the `client_id`, `secret` and `tenant` in Python like :

```
from azure.common.credentials import ServicePrincipalCredentials as spc
cred = spc(client_id = 'abcdefghi-1234-4555-8173-jklmnopqrstu',secret='abcdEFGHIJ//
↪klmnopqrSTU/',tenant='1234567-abcd-7890-ABCD-1234567890')
```

If your credentials are invalid, you may see errors like this:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/site-packages/msrestazure/azure_active_directory.py",
↪ line 403, in __init__
    self.set_token()
  File "/usr/local/lib/python2.7/site-packages/msrestazure/azure_active_directory.py",
↪ line 434, in set_token
    raise_with_traceback(AuthenticationError, "", err)
  File "/usr/local/lib/python2.7/site-packages/msrest/exceptions.py", line 50, in␣
↪raise_with_traceback
    raise error
msrest.exceptions.AuthenticationError: , InvalidClientIdError: (invalid_request)␣
↪AADSTS90002: No service namespace named '<wrong id>' was found in the data store.
Trace ID: <UUID>
Correlation ID: <UUID>
Timestamp: 2016-10-04 15:41:24Z
```

or :

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/site-packages/msrestazure/azure_active_directory.py",
↪ line 403, in __init__
    self.set_token()
  File "/usr/local/lib/python2.7/site-packages/msrestazure/azure_active_directory.py",
↪ line 434, in set_token
    raise_with_traceback(AuthenticationError, "", err)
  File "/usr/local/lib/python2.7/site-packages/msrest/exceptions.py", line 50, in␣
↪raise_with_traceback
    raise error
 msrest.exceptions.AuthenticationError: , InvalidClientError: (invalid_client)␣
↪AADSTS70002: Error validating credentials. AADSTS50012: Invalid client secret is␣
↪provided.
  Trace ID: <UUID>
```

```
  Correlation ID: <UUID>
  Timestamp: 2016-10-04 15:41:33Z
```

This may occur because your secret is not registered properly or client_id or tenant is not found.

Create a new Resource Group

The first step prior to any deployment would be creating a new resource group and it can be done via `ResourceManagmentClient()` from `azure.mgmt.resource`

Let's try to create a sample group named 'quickstart-rg-1' by the following code:

```
from azure.mgmt.resource import ResourceManagementClient as rmc
client = rmc(cred, 'subscription_id')
client.resource_groups.create_or_update(
          'quickstart-rg-1',
          {
                    'location':'eastus'
          }
     )
```

Replace the 'subscription_id' with a real value.

If you do not have proper permissions, error message looks like:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/site-packages/azure/mgmt/resource/resources/
↪operations/resource_groups_operations.py", line 223, in create_or_update
    raise exp
msrestazure.azure_exceptions.CloudError: The client '<uuid>' with object id '<uuid>'␣
↪does not have authorization to perform action 'Microsoft.Resources/subscriptions/
↪resourcegroups/write' over scope '/subscriptions/<subscription_id>/resourcegroups/
↪quickstart-rg-1'.
```

If your subscription principal is not consent:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/site-packages/azure/mgmt/resource/resources/
↪operations/resource_groups_operations.py", line 223, in create_or_update
    raise exp
  msrestazure.azure_exceptions.CloudError: The received access token is not valid: at␣
↪least one of the claims 'puid' or 'altsecid' or 'oid' should be present. If you are␣
↪accessing as application please make sure service principal is properly created in␣
↪the tenant.
```

Authentication in Simple Azure

Simple Azure requires the following information to authenticate:

- subscription id (identication to your account, e.g. `azure account show` shows ID)
- client id (equal to `client_id`)
- tenant id (equal to `tenant`)

- client secret key (equal to `secret`)

With Environment Variables

It is recommmend to store the credentials using environment variables instead passing through as Python parameters in code. Use the following environment variable names to store:

- subscription id: `AZURE_SUBSCRIPTION_ID`
- client id: `AZURE_CLIENT_ID`
- tenant id: `AZURE_TENANT_ID`
- client secret key: `AZURE_CLIENT_SECRET`

In a simple form, save these in a file and load it before using Simple Azure in a shell. For example:

```
$ cat <<EOF > ~/.saz/cred
export AZURE_SUBSCRIPTION_ID=5s3ag2s5-2aa1-4828-xxxx-9g8sw72w5w5g
export AZURE_CLIENT_ID=5c5a3ea3-ap34-4pd0-xxxx-2p38ac00aap1
export AZURE_TENANT_ID=5e39a20e-c55a-53de-xxxx-2503a55et6ta
export AZURE_CLIENT_SECRET=xxxx
EOF
```

Then source it like:

```
$ source ~/.saz/cred
```

`env` command displays environment variables exposed, e.g.:

```
$ env|grep AZURE
AZURE_SUBSCRIPTION_ID=5s3ag2s5-2aa1-4828-xxxx-9g8sw72w5w5g
AZURE_CLIENT_ID=5c5a3ea3-ap34-4pd0-xxxx-2p38ac00aap1
AZURE_TENANT_ID=5e39a20e-c55a-53de-xxxx-2503a55et6ta
AZURE_CLIENT_SECRET=xxxx
```

Tips on Getting Credential via Azure CLI

Subscription id and tenant id are found by, for example:

```
$ azure account show
info:    Executing command account show
data:    Name                        : Simple-Azure
data:    ID                          : 5s3ag2s5-2aa1-4828-xxxx-9g8sw72w5w5g
data:    State                       : Enabled
data:    Tenant ID                   : 5e39a20e-c55a-53de-xxxx-2503a55et6ta
data:    Is Default                  : true
data:    Environment                 : AzureCloud
data:    Has Certificate             : Yes
data:    Has Access Token            : Yes
data:    User name                   : hroe.lee@gmail.com
data:
info:    account show command OK
```

- `ID` represents `AZURE_SUBSCRIPTION_ID`.
- `Tenant ID` represents `AZURE_TENANT_ID`.

Client id is found by, for example:

```
$ azure ad app list
info:    Executing command ad app list
+ Listing applications
data:    AppId:                    5c5a3ea3-ap34-4pd0-xxxx-2p38ac00aap1
dqtq:    ObjectId:                 dc25d100-1234-4567-bf11-1234e1234dbq
data:    DisplayName:              simpleazure
data:    IdentifierUris:           0=https://simplezure.com/login
data:    ReplyUrls:
data:    AvailableToOtherTenants: False
data:    HomePage:                 http://simpleazure.com
data:
info:    ad app list command OK
```

`AppId` represents `AZURE_CLIENT_ID`.

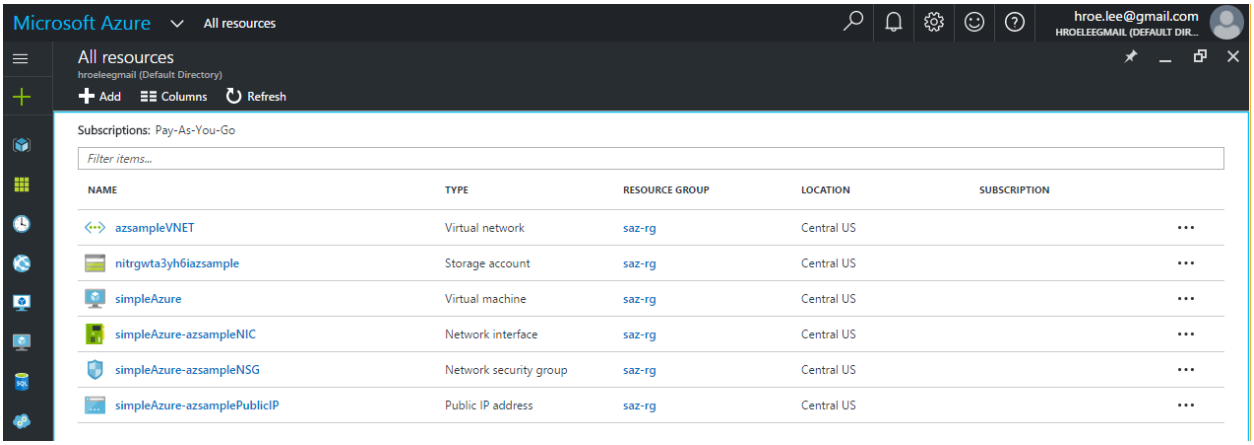
`AZURE_CLIENT_SECRET` is not visible because it is one-time displayed value from the portal. It is also same as the `<password>` used in the service principal credential in Azure CLI.

Deploying Azure Virtual Machine in Simple Azure ARM Mode

Simpla Azure deploys a Ubuntu 16.04 VM using the sample template from Azure-Samples like this:

```
>>> from simpleazure import SimpleAzure
>>> saz = SimpleAzure()
>>> url = "https://raw.githubusercontent.com/Azure-Samples/resource-manager-python-
↪template-deployment/master/templates/template.json"
>>> saz.arm.deploy(template = url, param = { "sshKeyData": "ssh-rsa
    AAAB3Nza..." })
```

A new deployment is completed in a resource group like:



Deleting a deployment is:

```
>>> saz.arm.terminate_deployment()
```

Or removing a resource group is:

```
>>> saz.arm.remove_resource_group()
```

Note: Use 'remove_resource_group()' if you force to stop and remove all running services. 'terminate_deployment()' does not remove services in running state.

Overview

Azure Virtual Machine is used to start via the servicemanagement API in Python (which is now legacy or classic mode) with limited access to resources but new Azure Resource Manager (ARM) supports launching a virtual machine with its template for a service deployment. This page demonstrates how to start a virtual machine in Simple Azure ARM mode with the template which contains information of resoures to be deployed e.g. Virtual Machine and Virtual Network with Resource Groups. Simple Azure is able to load custom templates from a file or a web and use the official community templates Azure-QuickStart-Templates.

Note: ARM does not support the classic version of virtual machines and cloud services which are only available via ServiceManagementAPI. VMs launched via ARM do not appear on ASM listing and vice versa.

ARM JSON Template

Azure Resource Template uses JSON format to describe its parameters, variables, resources and outputs. For example, the blank template looks like:

```
{
   "$schema": "http://schema.management.azure.com/schemas/2015-01-01/
↪deploymentTemplate.json#",
   "contentVersion": "1.0.0.0",
   "parameters": {  },
   "resources": [  ]
}
```

- This basic template is obtained from the azure portal: https://portal.azure.com/#create/Microsoft.MyGallery
- `resources` contains definition of azure services to be deployed e.g. Virtual Machine. Also, this entity is mandatory.
- `parameters` contains input values which allow you to provide when template is deployed.

Note: For more information about data structure of resources and parameters, see the 'authoring templates' here: https://azure.microsoft.com/en-us/documentation/articles/resource-group-authoring-templates/

In addition, there are variables and outputs which are recommended to add according to the official templates.

Note: Need to setup credentials for ARM? see the Azure Resource Manager page here *Quick Setup for Azure Resource Manager Mode*

Starting a VM with Simple Azure (step-by-step)

`arm` sub package is added under `simpleazure`. Try:

```
>>> from simpleazure import SimpleAzure
>>> saz = SimpleAzure()
>>> saz.arm
```

Credentials

The following Azure credentials are required to use ARM template on SimpleAzure. Credentials for ASM (Azure Service Management API) are not valid for ARM.

- subscription id (equal to env name `AZURE_SUBSCRIPTION_ID`)
- client id (equal to env name `AZURE_CLIENT_ID`)
- tenant id (equal to env name `AZURE_TENANT_ID`)
- client secret key (equal to env name `AZURE_CLIENT_SECRET`)

You may not be familiar with client id and client secret key, see the page here '*Quick Setup for Azure Resource Manager Mode*'. Client id and secret key can be obtained via Azure CLI or the new portal.

You can deliver credential values as parameters in Python Shell like:

```
>>> sid = "5s3ag2s5-2aa1-4828-xxxx-9g8sw72w5w5g"
>>> cid = "5c5a3ea3-ap34-4pd0-xxxx-2p38ac00aap1"
>>> secret = "xxxxxxxxxxxxxxxx"
>>> tid = "5e39a20e-c55a-53de-xxxx-2503a55et6ta"
>>> saz.arm.set_credential(subscription = sid, client_id = cid, secret =
    secret, tenant = tid)
```

It is actually recommended to use environment variables. Create a file for credentials like:

```
$ cat <<EOF > ~/.saz/cred
export AZURE_SUBSCRIPTION_ID=5s3ag2s5-2aa1-4828-xxxx-9g8sw72w5w5g
export AZURE_CLIENT_ID=5c5a3ea3-ap34-4pd0-xxxx-2p38ac00aap1
export AZURE_TENANT_ID=5e39a20e-c55a-53de-xxxx-2503a55et6ta
export AZURE_CLIENT_SECRET=xxxx
EOF
```

And then source it before running Python like:

```
$ source ~/.saz/cred
```

Now. no parameters are necessary. Simple Azure loads credentials from environment variables:

```
>>> saz.arm.set_credential()
```

Load Template

We want to use 101-vm-sshkey template from the *azure-quickstart-templates* which deploys a Ubuntu 14.04.4-LTS Virtual Machine with a SSH key injection. `deploy()` accepts template from URL or a local file as long as it is a JSON format.

From URL:

```
>>> template_url = 'https://raw.githubusercontent.com/Azure/azure-quickstart-
↪templates/master/101-vm-sshkey/azuredeploy.json'
>>> saz.arm.set_template(template_url)
```

From FILE:

```
>>> template_path = "~/101-vm-sshkey/azuredeploy.json"
>>> saz.arm.set_template(template_path)
```

Set Parameters

`101-vm-sshkey` template requires ssh public key parameter to deploy a VM. Simple Azure loads a public key string from the base ssh directory ($HOME/.ssh).

We assume that you already have a SSH key pair generated with a default filename (`~/.ssh/id_rsa.pub` and `id_rsa`) in your home directory. `sshkey` object contains public key string like:

```
>>> saz.arm.sshkey.pubkey
ssh-rsa AAAAB3... hrlee@quickstart
```

We provide this as a parameter like:

```
>>> saz.arm.set_parameter({"sshKeyData": arm.sshkey.pubkey})
```

Note: sshKeyData is a parameter name defined in the template

Deployment

The `101-vm-sshkey` template contains six (6) resources: 1 Compute, 4 Network and 1 Storage to deploy a Ubuntu VM on Azure. Exact resource names are:

- Microsoft.Compute/virtualMachines
- Microsoft.Network/networkInterfaces
- Microsoft.Network/networkSecurityGroups
- Microsoft.Network/publicIPAddresses
- Microsoft.Network/virtualNetworks
- Microsoft.Storage/storageAccounts

The relations of these services are visualized via armvis.io here

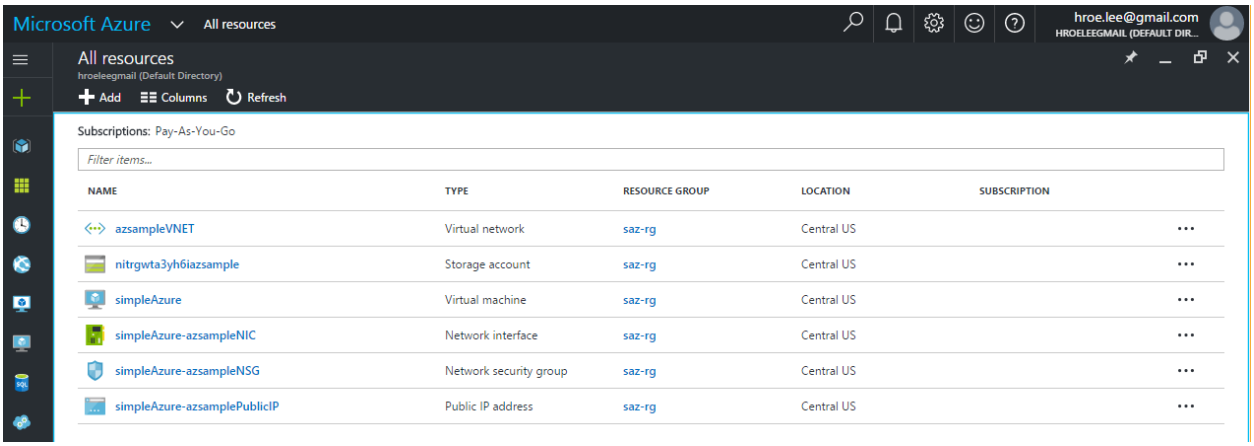
In Simple Azure, `deploy()` function creates a new deployment for these six resources by:

```
>>> saz.arm.deploy()
```

You can directly call `deploy()` function without setting template (set_template()) and parameters (set_parameter()) but sending them as function parameters like (Both ways work same):

```
>>> saz.arm.deploy(template_url, parameters)
```

The status of a deployment is visible on the Azure Portal like:



It may take several minutes to get the VM ready to access via SSH your your key.

Termination

When Simple Azure terminates VMs in a classic mode (which is using ServiceManagement API), each service needs to be deleted seperately, e.g. storage, cloud services and virtual machines. In ARM mode, however, a simple function call deletes resources in a same unit (a sample resource group or deployment).

Deleting a deployment is:

```
>>> saz.arm.terminate_deployment()
```

Removing a resource group is :

```
>>> saz.arm.remove_resource_group()
```

Deployment name or resource group name can be specified as a parameter, if you want to clean up other resources as well.

The following sections are for further readings about defining resources in a template.

Further Reading: Virtual Machine in Resources

Starting a new virtual machine (*"Microsoft.Compute/virtualMachines"*) requires Storage account and Network resources to store image file (.vhd) and configure a network interface with a public ip address. (This is probably different for Windows machines) Therefore, additional resources are expected in the `resources` entity to complete vm deployment.

It might be helpful to review virtual machine service from one of the existing templates. There is a template starting a VM with ssh public key: 101-vm-ssh-key template , and the virtual machine service is defined like this in `resources`:

```
{
  "apiVersion": "2015-08-01",
  "type": "Microsoft.Compute/virtualMachines",
  "name": "simpleazure",
  "location": "centralus",
  "properties": {
    "hardwareProfile": {
    "vmSize": "Standard_DS2"
    },
```

```
    "osProfile": {
      "computerName": "simpleazure",
      "adminUsername": "ubuntu",
      "linuxConfiguration": {
        "disablePasswordAuthentication": "true",
        "ssh": {
          "publicKeys": [
            {
              "keyData": "GEN-SSH-PUB-KEY"
            }
          ]
        }
      }
    },
    "storageProfile": {
      "imageReference": {
        "publisher": "Canonical",
        "offer": "UbuntuServer",
        "sku": "14.04-LTS",
        "version": "latest"
      },
      "osDisk": {
        "name": "osdisk",
        "vhd": {
          "uri": "[variables('storage_uri')]"
        },
        "createOption": "FromImage"
      }
    },
    "networkProfile": {
      {
        "id": "[resourceId('Microsoft.Network/networkInterfaces', variables('nicName
↪'))]"
      }
    }
  }
}
```

There are other elements available but only required ones are demonstrated in this example according to the ARM schemas

Deploying Azure QuickStart Templates

```
>>> from simpleazure import SimpleAzure
>>> saz = SimpleAzure()
>>> vm_sshkey_template = saz.aqst.get_template('101-vm-sshkey')
>>> saz.arm.load_template(vm_sshkey_template)
>>> saz.arm.add_parameter({"sshKeyData": "ssh-rsa AAAB... hrlee@quickstart"})
>>> saz.arm.deploy()
```

Azure offers Power Shell and CLI tool to deploy community templates[1] from starting a single virtual machine (e.g. 101-vm-sshkey) to building hadoop clusters with Apache Spark (e.g. hdinsight-apache-spark) with limited helper functions. Simple Azure supports deploying these templates in Python with powerful functions: import, export, edit, store, review, compare(diff), deploy and search.

[1] as of 10/13/2016 from https://github.com/Azure/azure-quickstart-templates

The example above shows that Simple Azure loads `101-vm-sshkey` template (which creates a VM with ssh access) from the *azure-quickstart-templates* github repository (which is included in Simple Azure) and deploys a virtual machine with a required parameter, ssh public key string (*sshKeyData*).

Overview

This page describes basic use of Azure QuickStart Templates with Simple Azure Python library which supports - template search, import, export, edit, store, review, compare(diff), and deploy functions.

QuickStart Directory Structure

A template in the azure quickstart is served in a single directory with required json files to describe resource deployments.

```
100-blank-template   (directory name)
|
\- azuredeploy.json  (main template to deploy)
\- azuredeploy.parameters.json       (required parameter definitions)
\- metadata.json     (template description)
```

Note that the directory name here (i.e. 100-blank-template) is an index of a template that Simple Azure uses.

Template Information

Azure QuickStart Templates are written by community therefore descriptions are necessary to understand resource deployments with properties and required parameters. Simple Azure reads template information based on the directory name and files in the directory. Metadata, for example, is retrieved by:

```
>>> vm_sshkey_template.metadata()
dateUpdated                                          2015-06-05
description        This template allows you to create a Virtual M...
githubUsername                                         squillace
itemDisplayName     Deploy a Virtual Machine with SSH rsa public key
summary             Deploy a Virtual Machine with SSH rsa public key
```

We can find this template is about a virtual machine deployment with ssh key from summary and itemDisplayName. Other information such as written date, author, and long description is also provided. According to the description, SSH public key will be required as a parameter because ssh key string should be injected when a virtual machine is booted. Parameter options can be retrieved by:

```
>>> vm_sshkey_template.parameters()
adminUsername          azureuser
sshKeyData       GEN-SSH-PUB-KEY
```

`101-vm-sshkey` template requires `sshKeyData` parameter to obtain ssh public key string from users otherwise this template won't deploy a virtual machine with a ssh access.

Template List

`get_templates()` lists all templates from Azure QuickStart Templates github repository and `ten()` pages its listing with 10 counts. `twenty()` is also provided with 20 counts.

```
>>> templates = saz.aqst.get_templates()
>>> templates.ten()
100-blank-template                                                              ␣
↪Blank Template
101-acs-dcos                                                    Azure Container␣
↪Service - DC/OS
101-acs-mesos                                                   Azure Container␣
↪Service - DC/OS
101-acs-swarm                                                   Azure Container␣
↪Service - Swarm
101-app-service-certificate-standard            Create and assign a standard App␣
↪Service Certi...
101-app-service-certificate-wildcard            Create and assign a wildcard App␣
↪Service Certi...
101-application-gateway-create                                    Create an␣
↪Application Gateway
101-application-gateway-public-ip                    Create an Application Gateway␣
↪with Public IP
101-application-gateway-public-ip-ssl-offload        Create an Application Gateway␣
↪with Public IP
101-automation-runbook-getvms                   Create Azure Automation Runbook to␣
↪retrieve Az...
```

Choose one of the templates with its directory name, for example, `101-acs-dcos` template (2nd template in the listing) is selected by:

```
>>> templates['101-acs-dcos'].metadata()
dateUpdated                                              2016-02-18
description        Deploy an Azure Container Service instance for...
githubUsername                                             rgardler
itemDisplayName                     Azure Container Service - DC/OS
summary            Azure Container Service optimizes the configur...

>>> templates['101-acs-dcos'].resources()
Microsoft.ContainerService/containerServices    {u'properties': {u'masterProfile': {u
↪'count': ...
```

We find that `101-acs-dcos` template is a Azure Container Service from its description and resource definition.

More options are available to search, load and deploy templates via Simple Azure and the following sections demonstrate these options with examples.

Searching Template

Try a template search with a keyword(s) to find an interesting template. For example, search 'rhel' keyword to find Red Hat Enterprise Linux templates.

```
>>> rhel_templates = saz.aqst.search("rhel")

>>> rhel_templates.count()
13
```

It found 13 templates and the first ten items are:

```
>>> rhel_templates.ten()
101-vm-full-disk-encrypted-rhel       Red Hat Enterprise Linux 7.2 VM (Fully␣
↪Encrypted)
```

```
101-vm-simple-rhel                        Red Hat Enterprise Linux VM (RHEL 7.2 or RHEL ..
↪.
201-encrypt-running-linux-vm                    Enable encryption on a running Linux␣
↪VM.
create-hpc-cluster-linux-cn                Create an HPC cluster with Linux compute␣
↪nodes
intel-lustre-client-server/scripts
intel-lustre-clients-on-centos            Intel Lustre clients using CentOS gallery␣
↪image
intel-lustre-clients-vmss-centos         Azure VM Scale Set as clients of Intel Lustre
openshift-origin-rhel                   OpenShift Origin on RHEL (On Demand image) or ..
↪.
openshift-origin-rhel/nested
sap-2-tier-marketplace-image              2-tier configuration for use with SAP␣
↪NetWeaver
```

Next items are displayed by calling `ten()` again:

```
>>> rhel_templates.ten()
== End of page ! ==
sap-3-tier-marketplace-image          3-tier configuration for use with SAP NetWeaver
vsts-tomcat-redhat-vm                  Red Hat Tomcat server for use with Team Servic..
↪.
zabbix-monitoring-cluster/scripts
```

Resource types can be used to search, for example, if `virtualMachines` and `publicipaddresses` are given:

```
>>> vms_with_public_ips = saz.aqst.search('virtualMachines publicipaddresses')

>>> vms_with_public_ips.ten()
201-customscript-extension-azure-storage-on-ubuntu                Custom Script␣
↪extension on a Ubuntu VM
201-customscript-extension-public-storage-on-ubuntu               Custom Script␣
↪extension on a Ubuntu VM
201-dependency-between-scripts-using-extensions        Use script extensions to␣
↪install Mongo DB on U...
201-oms-extension-ubuntu-vm                                  Deploy a Ubuntu VM␣
↪with the OMS extension
201-traffic-manager-vm
201-vm-winrm-windows                                   Deploy a Windows VM and␣
↪configures WinRM https...
anti-malware-extension-windows-vm                      Create a Windows VM with Anti-
↪Malware extensio...
apache2-on-ubuntu-vm                                                    Apache␣
↪Webserver on Ubuntu VM
azure-jenkins                                          Deploy instance of Jenkins␣
↪targeting Azure Pla...
bitcore-centos-vm                                      Bitcore Node and Utilities for␣
↪Bitcoin on Cent...
dtype: object
```

Let's select the first template.

```
>>> vms_with_public_ips.ten['201-customscript-extension-azure-storage-on-ubuntu'].
↪resources()
Microsoft.Compute/virtualMachines              {u'name': u'[variables('vmName')]', u
↪'apiVersi...
Microsoft.Compute/virtualMachines/extensions    {u'name': u'[concat(variables('vmName
↪'),'/', v...
```

```
Microsoft.Network/networkInterfaces             {u'name': u'[variables('nicName')]', u
↪'apiVers...
Microsoft.Network/publicIPAddresses             {u'properties': {u
↪'publicIPAllocationMethod': ...
Microsoft.Network/virtualNetworks               {u'properties': {u'subnets': [{u'name
↪': u"[var...
Microsoft.Storage/storageAccounts               {u'properties': {u'accountType': u
↪'[variables(...
```

Indeed, it has virtualMachines and publicIPAddresses resource types.

Template Details

Template consists of key elements: metadata, parameters, resources, and dependson (dependencies) to describe resource deployments. Simple Azure Template() object functions offer to review these template elements and visualize dependencies. The available functions are:

- [template object].metadata()
- [template object].parameters()
- [template object].resources()
- [template object].dependson()
- [template object].dependson_print()

Metadata

See metadata of the template `101-vm-simple-rhel` from the search results above:

```
>>> rhel_templates['101-vm-simple-rhel'].metadata()
dateUpdated                                              2016-02-23
description        This template will deploy a Red Hat Enterprise...
githubUsername                                           BorisB2015
itemDisplayName    Red Hat Enterprise Linux VM (RHEL 7.2 or RHEL ...
summary            This template will deploy RedHat (RHEL) VM, us...

>>> rhel_templates['101-vm-simple-rhel'].metadata().description
u'This template will deploy a Red Hat Enterprise Linux VM (RHEL 7.2 or
RHEL 6.7), using the Pay-As-You-Go RHEL VM image for the selected
version on Standard D1 VM in the location of your chosen resource group
with an additional 100 GiB data disk attached to the VM. Additional
charges apply to this image - consult Azure VM Pricing page for
details.'
```

Here, `metadata()` returns `101-vm-simple-rhel` template description in Pandas Series format and full description text is visible like python class variable (metadata().description).

This information is from `matadata.json` and returned by Pandas Series

```
[template object].metadata()            # pandas Series
```

Parameters

We may want to know what parameters are necessary to deploy for this template:

```
>>> rhel_templates['101-vm-simple-rhel'].parameters()
adminPassword
adminUsername
vmName
```

These three parameters need to be set before deploying the template and we will find out how to set parameters using Simple Azure later in this page.

This information is from `azuredeploy.parameters.json` and returned by Pandas Series:

```
[template object].parameters()          # pandas Series
```

Resources

According to the metadata earlier, we know that `101-vm-simple-rhel` deploys a virtual machine with Standard D1 but it isn't clear what resources are used.

```
>>> rhel_templates['101-vm-simple-rhel'].resources()
Microsoft.Compute/virtualMachines      {u'name': u'[parameters('vmName')]', u'apiVers.
↪..
Microsoft.Network/networkInterfaces    {u'name': u'[variables('nicName')]', u'apiVers.
↪..
Microsoft.Network/publicIPAddresses    {u'properties': {u'publicIPAllocationMethod': .
↪..
Microsoft.Network/virtualNetworks      {u'properties': {u'subnets': [{u'name': u"[var.
↪..
Microsoft.Storage/storageAccounts      {u'properties': {u'accountType': u'[variables(.
↪..
```

There are five services (including `virtualMachines` in Compute service) are described in the template to deploy RHEL image on Microsoft Azure.

This information is from `azuredeploy.json` and returned by Pandas Series:

```
[template object].resources()          # pandas Series
```

Service Dependency

Services can be related to other services when it deploys, for example, `publicIPAddresses` and `virtualNetworks` services are depended on `networkInterfaces` resource in the `101-vm-simple-rhel` template. Dependencies are not visible in `resources()` but in `dependson()` which returns its relation in python dict data type using pprint():

```
>>> rhel_templates['101-vm-simple-rhel'].dependson_print()
{u'Microsoft.Compute/virtualMachines': {u'Microsoft.Network/networkInterfaces': {u
↪'Microsoft.Network/publicIPAddresses': {u"[concat(uniquestring(parameters('vmName
↪')), 'publicip')]": {}},
                                                                              u
↪'Microsoft.Network/virtualNetworks': {u"[concat(uniquestring(parameters('vmName')),
↪'vnet')]": {}}},

↪                                    u'Microsoft.Storage/storageAccounts': {u
↪"[concat(uniquestring(parameters('vmName')), 'storage')]": {}}}}
```

Note: ARMVIZ.io depicts the service dependency on the web like Simple Azure. For example, `101-vm-simple-rhel`'s dependency is displayed here

The dependencies are retrieved from `dependsOn` section in `azuredeploy.json` in Python dictionary format (dependson()) and in Pretty Print format (dependson_print()):

```
[template object].dependson()           # dict type return
[template object].dependson_print()     # pprint
```

Template Deployment

Tip: Basic template deployment on Simple Azure is available, see *Deploying Azure Virtual Machine in Simple Azure ARM Mode*

Simple Azure has a sub module for Azure Resource Manager (ARM) which deploys a template on Azure.

```
>>> from simpleazure import SimpleAzure
>>> saz = SimpleAzure() # Azure Resource Manager object
```

Next step is loading a template with a parameter.

Load Template

arm object needs to know which template will be used to deploy and we tell:

```
>>> saz.arm.load_template(rhel['101-vm-simple-rhel'])
```

Set Parameter

In our example of RHEL, three parameters need to be set before its deployment, `adminPassword`, `adminUsername` and `vmName`:

```
>>> saz.arm.set_parameters(
               {"adminPassword":"xxxxx",
                "adminUsername":"azureuser",
                "vmName":"simpleazure-quickstart"}
              )

{'adminPassword': {'value': 'xxxxx'},
 'adminUsername': {'value': 'azureuser'},
 'vmName': {'value': 'saz-quickstart'}}
```

Python dict data type has updated with *value* key name like `{ '[parameter name]' : { 'value': '[parameter value'] }}` and these parameter settings will be used when the template is deployed.

Note: Use `add_parameter()`, if you have additional parameter to add in existing parameters, e.g. add_parameter({"dnsName":"azure-preview"})

Deployment

`deploy()` function runs a template with a parameter if they are already set.

```
>>> saz.arm.deploy()
```

Or you can directly deploy a template with parameters.

```
>>> saz.arm.deploy(rhel['101-vm-simple-rhel'], {"adminPassword":"xxxxx",
"adminUsername":"azureuser", "vmName":"saz-quickstart"})
```

It may take a few minutes to complete a deployment and give access to a virtual machine.

Access

If a template is deployed with an access to virtual machines i.e. SSH via public IP addresses, `view_info()` returns an ip address in a same resource group. `Microsoft.Network/PublicIPAddresses` service is fetched in this example.

```
>>> saz.arm.view_info()
[u'40.77.103.150']
```

Use the same login user name and password from the parameters defined earlier:

```
$ ssh 40.77.103.150 -l azureuser
  The authenticity of host '40.77.103.150 (40.77.103.150)' can't be established.
  ECDSA key fingerprint is 64:fc:dd:7c:98:8c:ed:93:63:61:56:31:81:ad:cf:69.
  Are you sure you want to continue connecting (yes/no)? yes
  Warning: Permanently added '40.77.103.150' (ECDSA) to the list of known hosts.
  azureuser@40.77.103.150's password:
  [azureuser@simpleazure-quickstart-rhel ~]$
```

We confirm that the virutual machine is RHEL 7.2 by:

```
[azureuser@simpleazure-quickstart-rhel ~]$ cat /etc/redhat-release
Red Hat Enterprise Linux Server release 7.2 (Maipo)
```

Termination

Deleting a resource group where deployment is made terminates all services in the resource group.

```
>>> saz.arm.remove_resource_group()
```

Tutorial (ARM - New)

The recently added tutorials are available in IPython Notebook below. These are based on Azure Resource Manager mode (ARM).

- Tutorial - Account Setup for Azure Resource Manager (ARM) (Mandatory for the FIRST user)
- Tutorial - Modifying a Template in Python
- Tutorial - Deploying Software Stacks after Provisiong

Tutorial (classic mode)

The tutorial in this page assists you in learning how to use Simple Azure on Python. In this tutorial, we creates Windows Azure Virtual Machines for Azure Data Science Core and BioLinux with IPython. It shows usage in pre-configured environments but it explains main features that you need to know. For more detail of function APIs, module descriptions might be helpful. There are several tools and libraries that we have used such as StarCluster, IPython, azure-sdk-for-python, Bioblend, etc.

You can also have an interactive python tutorial via IPython Notebook to learn Simple Azure.

Tutorial by IPython Notebook

IPython Notebook Viewer supports sharing and viewing ipython notebook files. Simple Azure's tutorial can be viewed through the ipynb viewer.

Simple Azure

- Simplified Windows Azure SDK for Python (Simple Azure)
- Deploying a community image, Azure Data Science Core
- Enabling IPython Cluster with Simple Azure

azure-sdk-for-python (legacy)

azure-sdk-for-python is the official python library for azure:

- Deploying Windows Azure Virtual Machines with Python SDK (azure-sdk-for-python)
- Deploying Windows and Linux VMs through azure-sdk-for-python
- Deploying multiple VMs through azure-sdk-for-python

Galaxy workflow toolkit

Galaxy project provides a web-based platform and a python library for scientific research. Here, the tutorials below introduce use cases of Galaxy on IPython Notebook.

- BioBlend (Galaxy Python library) with IPython
- Finding coding exon which has the highest number of single nucleotide polymorphisms on chromosome 22

Installation

Simple Azure Installation is available via github, Pypi and docker image.

> **Warning:** Python 3.0+ is not supported.

Docker Installation

Simple Azure is available in a Docker image to run.

- With IPython Notebook:

```
docker run -d -p 8888:8888 lee212/simpleazure_with_ipython
```

Open a browser with the port number **8888**.

- Simple Azure only:

```
docker run -i -t lee212/simpleazure
```

Python Pypi Installation

```
pip install simpleazure
```

Github Installation

```
git clone https://github.com/lee212/simpleazure.git
cd simpleazure
pip install -r requirements.txt
python setup.py install
```

Virtualenv and virtualenvwrapper

Virtualenv enables project-based development for python and virtualenvwrapper provides simple commands to switch different python environments. It is not required to install but would be useful when you need a user space installation without super-user privilege.

Pypi with virtualenv

```
$ mkvirtualenv simpleazure
(simpleazure)$ pip install simpleazure
```

pypi - system wide installation with sudo

```
$ sudo pip install simpleazure
```

pypi on Windows

On Windows, *easy_install* help install Simple Azure. distribute_setup.py file do an installation of easy_install.

`;C:\Python27\Scripts`

Once you have easy_install, you can install simple azure:

```
> easy_install pip
> pip install simpleazure
```

Quickstart (classic mode)

This page provides a good introduction to Simple Azure in classic mode.

Deploying Azure Virtual Machine

```
from simpleazure import SimpleAzure

azure = SimpleAzure()
azure.asm.create_vm()
```

You can change an operating system image by the `set_image()` function. For example, *Ubuntu 12.04 distribution* can be selected like as follows:

```
azure.asm.set_image(label="Ubuntu Server 12.04.2 LTS")
```

This allows you to create a vm with the selected image. Note that `set_image()` must be used before calling the `create_vm()` function.

Deploying several machines

`create_cluster()` function allows you to create multiple machines at once. `num=` parameter can be used to specify the number of nodes.

```
azure = SimpleAzure()
azure.asm.create_cluster(num=4)

my-cluster-vm-0-87412
{'request_id': '88c94c00288d42acaf877783f09c4558'}
my-cluster-vm-1-61293
{'request_id': 'abfd563c2c4f4926872b6b1dba27a93b'}
my-cluster-vm-2-96085
{'request_id': '29b55f6cb5e94cfdbf244a7c848c854d'}
my-cluster-vm-3-46927
{'request_id': 'b1a3446ebafe47a295df4c9d1b7d743c'}
```

Deploying a Virtual Machine from the community images (VM DEPOT)

Personalized and preconfigured virtual machines images can be imported from the community repository (http://vmdepot.msopentech.com). This example explains as how to deploy a virtual machine with the community image, Azure Data Science Core.

```
azure = SimpleAzure()
q = azure.asm.get_registered_image(name="Azure-Data-Science-Core")
azure.asm.set_image(image=q)
azure.asm.create_vm()
```

Simple Azure on Command Line Interface (CLI) (TBD)

Simple Azure supports commands on a linux shell. For example, you can create clusters of virtual machines on Windows Azure like StarCluster like as follows: (StarCluster is a cluster-computing toolkit for Amazon EC2)

```
$ simpleazure-cluster start mycluster
```

Command (classic mode)

Simple Azure supports command line tools, so a user can create virtual machines on the shell.

Creating Clusters

The usage is based on StarCluster. We aim to provide an identical interface and command name to use clusters like StarCluster.

simpleazure-cluster

A user can create one or more clusters of virtual machines on Windows Azure:

```
$ simpleazure-cluster start mycluster
```

Note. `mycluster` is a profile name and the config file has been stored under the default directory `$HOME/.azure/cluster` as a yaml file.

The number of clusters can be changed in the config file (mycluster.yaml) like this:

```
...
num=5
```

sshmaster

This command allows to login to a master node via SSH.

```
$ simpleazure-cluster sshmaster mycluster
[myvm-81fd6840ae.cloudapp.net] run: bash
[myvm-81fd6840ae.cloudapp.net] out: azureuser@myvm-81fd6840ae:~$ ls -al
[myvm-81fd6840ae.cloudapp.net] out: total 52
[myvm-81fd6840ae.cloudapp.net] out: drwxr-xr-x 6 azureuser azureuser 4096 Jul 29␣
↪23:50 .
[myvm-81fd6840ae.cloudapp.net] out: drwxr-xr-x 3 root      root      4096 Jul 25␣
↪22:06 ..
[myvm-81fd6840ae.cloudapp.net] out: -rw------- 1 azureuser azureuser 4617 Aug  6␣
↪21:38 .bash_history
[myvm-81fd6840ae.cloudapp.net] out: -rw-r--r-- 1 azureuser azureuser  220 Apr  3 ␣
↪2012 .bash_logout
[myvm-81fd6840ae.cloudapp.net] out: -rw-r--r-- 1 azureuser azureuser 3486 Apr  3 ␣
↪2012 .bashrc
[myvm-81fd6840ae.cloudapp.net] out: drwx------ 2 azureuser azureuser 4096 Jul 25␣
↪22:22 .cache
[myvm-81fd6840ae.cloudapp.net] out: drwxrwxr-x 4 azureuser azureuser 4096 Jul 27␣
↪20:17 .ipython
```

```
[myvm-81fd6840ae.cloudapp.net] out: -rw-r--r-- 1 azureuser azureuser  675 Apr  3 ␣
↪2012 .profile
[myvm-81fd6840ae.cloudapp.net] out: drwx------ 2 azureuser azureuser 4096 Jul 26␣
↪19:22 .ssh
[myvm-81fd6840ae.cloudapp.net] out: -rw------- 1 azureuser azureuser 5658 Jul 26␣
↪22:44 .viminfo
[myvm-81fd6840ae.cloudapp.net] out: drwx------ 2 azureuser azureuser 4096 Jul 29␣
↪23:56 .w3m
[myvm-81fd6840ae.cloudapp.net] out: azureuser@myvm-81fd6840ae:~$ hostname
[myvm-81fd6840ae.cloudapp.net] out: myvm-81fd6840ae
[myvm-81fd6840ae.cloudapp.net] out: azureuser@myvm-81fd6840ae:~$ exit
[myvm-81fd6840ae.cloudapp.net] out: exit
[myvm-81fd6840ae.cloudapp.net] out:
```

Configuration (classic mode)

Simple Azure uses the default directory for the azure-cli tool, `$HOME/.azure`. It contains the `config.json` which includes an endpoint and a subscription of Windows Azure.

SSH Keys

`.ssh` directory contains certificates and thumbprints (fingerprints) of key pairs on the default directory `$HOME/.azure`.

pfx certificate

Personal Information Exchange (pfx) certificate is required to the Cloud (hosted) service.

`openssl` supports to convert your public and private keys to a pfx certificate.

private key files

For example, `myPrivateKey.key` which is a rsa 2048 private key is used to get access to virtual machines using SSH.

Cluster configuration

Simple Azure supports cluster computing and personal profiles enable individual settings when you launch clusters. The default directory for cluster conf file is `cluster` under the `$HOME/.azure` default directory.

Cluster profile

For example, `mycluster` has information in a yaml format in the cluster directory.

```
master: myvm-81fd6840ae
engines: [myvm-4406510ce8,myvm-7103520de1,myvm-5103520de4,myvm-5104120de1]
pkey: /home/azureuser/.azure/.ssh/myPrivateKey.key
num: 5
```