argus Documentation

Release 0.4

Cloudbase Solutions

October 31, 2016
# Contents

## 1 Getting started

1.1 Concepts ................................................. 3
1.2 Writing integration tests ................................. 3

## 2 argus’s API

2.1 The argus.backends.base Module .......................... 7
2.2 The argus.backends.windows Module .................... 8
2.3 The argus.backends.tempest.cloud Module .............. 8
2.4 The argus.backends.tempest.manager Module .......... 9
2.5 The argus.backends.tempest.tempest_backend Module .... 10
2.6 The argus.backends.heat.client Module ............... 11
2.7 The argus.backends.heat.heat_backend Module .......... 11
2.8 The argus.recipes.base Module ......................... 11
2.9 The argus.recipes.cloud.base Module .................. 12
2.10 The argus.recipes.cloud.windows Module .............. 13
2.11 The argus.scenarios.base Module ...................... 15
2.12 The argus.scenarios.cloud.base Module ............... 16
2.13 The argus.scenarios.service_mock Module ............. 16
2.14 The argus.scenarios.windows Module .................. 18
2.15 The argus.client.base Module .......................... 18
2.16 The argus.client.windows Module ...................... 19
2.17 The argus.util Module ................................... 20
2.18 The argus.introspection.base Module .................. 21
2.19 The argus.introspection.cloud.base Module .......... 21
2.20 The argus.introspection.cloud.windows Module ........ 22

## 3 How to create images for argus - a short recipe ................. 25

## 4 Microsoft Nano Server .................................. 27

## 5 Indices and tables ..................................... 29

Python Module Index ......................................... 31
**argus** is a framework for writing complex integration tests, for code that needs to run under various different operating systems. The tests themselves are running on a host, while the code that is tested runs into virtual machine instances. The underlying technology for spinning up new machines can be anything, as long as a wrapper is written for it. We’re using for now OpenStack based backends.

The project is actually used to test **cloudbase-init**, a portable multi-cloud initialization service, targeted to Windows platforms.

Contents:
CHAPTER 1

Getting started

Before explaining how to use argus, first we must describe a couple of concepts that argus uses.

1.1 Concepts

First, *argus* is based on a concept of a **scenario**, which describes what and how something should be tested. As a parallel with the unittest module, you can view a scenario as a test case, composed of multiple test methods. The scenario itself is composed of a couple of other concepts, as well:

- first, we have the concept of a **backend**, the underlying component which provides virtual machine instances to the scenario. An instance in this case can be anything: it can be an OpenStack instance, created with nova, it can be a docker container or a virtual machine created with Vagrant.

- the second concept is the **introspection**, which provides a way for interacting with the instances created by the backend. This usually means establishing a communication channel from the host to the instance.

- the instance can be prepared before running the tests by using a **recipe**, which describes the steps that are executed before running the actual tests.

- finally, the scenario is composed of the tests themselves, which verifies that the expected things to test actually occurred into the instance.

In the next section, we’ll see how one can write a scenario with a couple of tests.

1.2 Writing integration tests

Before writing our integration tests, we need to think a little about what technologies we’re planning to use. The current stack that argus provides is tailored for interacting with Windows platforms, but since the code is decoupled, other platforms can be easily added by writing a couple of classes. The same can be said about the backends that argus is currently exported, they can be easily replaced with anything else, as long as the established API is used for the new ones.

1.2.1 The backend

We were saying in the previous section that a backend is used in order to spin up new VMs, which will be used to run our application code and the tests were writing will be verifying that what happened in them was actually what we were expecting to be happening. In other words, the tests will run on the host, while the application code will run in the VM, which means that we can do almost anything in those VMs.
The base backend class in argus in `argus.backends.base.BaseBackend`, which provides only three public methods, `argus.backends.base.BaseBackend.setup_instance()` for creating an instance, `argus.backends.base.BaseBackend.cleanup()` for destroying it and `argus.backends.base.BaseBackend.get_remote_client()` for obtaining a client that can talk with our instances.

There is no requirement about how the client needs to look, only that it needs to implement a method through which a command can be executed on the instance. This means that we can use any protocol we want as long as it is supported by the instance’s OS (SSH for instance can work great for Unix based systems, while WinRM can be used for Windows).

Let’s take an example where we’re using OpenStack and its `nova` component as our VMs provider.

Finding what images we have available is as easy as running this command:

```bash
$ nova image-list
+--------------------------------------+-----------+--------+--------+
| ID | Name   | Status | Server |
+--------------------------------------+-----------+--------+--------+
| 17a34b8e-c573-48d6-920c-b4b450172b41 | Windows 8 | ACTIVE |       |
+--------------------------------------+-----------+--------+--------+
```

Creating a new instance will look like this:

```bash
$ nova boot --flavor 2 --image 17a34b8e-c573-48d6-920c-b4b450172b41 windows-10
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0e4011a4-3128-4674-ab16-dd1b7ecc126e</td>
</tr>
<tr>
<td>...</td>
<td>.....</td>
</tr>
</tbody>
</table>

While destroying a new instance usually is as simple as:

```bash
$ nova delete 0e4011a4-3128-4674-ab16-dd1b7ecc126e
```

Knowing this, our nova backend can look like this:

```python
import subprocess

from argus.backends.base import BaseBackend
from argus.client.windows import WinRemoteClient

class NovaBackend(BaseBackend):
    def prepare_instance(self):
        flavor_id = self._conf.openstack.flavor_ref
        image_id = self._conf.openstack.image_ref
        command = ["nova", "boot", "--flavor", flavor_ref,
                   "--image", image_ref, self._name]
        popen = subprocess.call(command, out=subprocess.PIPE)
        self._instance_id = _get_the_instance_id(popen)

    def cleanup(self):
        subprocess.call(["nova", "delete", self._instance_id])

    def get_remote_client(self, username, password, **kwargs):
        """Get a client to the underlying instance using username and password."""
```

Chapter 1. Getting started
1.2.2 The recipe

Now that our backend is capable of spinning up new VMs, let’s see how can we prepare them in order to test our application code.

Preparing an instance is the duty of a recipe, argus providing a base class for them in `argus.recipes.base.BaseRecipe`. This class provides only one public method, `argus.recipes.base.BaseRecipe.prepare()`, which will be called when the scenario wants to prepare an instance. The recipe itself will be initialized with a configuration object (where argus will hold all its settings) and the backend we specified in the scenario.

The recipe will use backend’s underlying remote client in order to talk with the instance.

Let’s see how can we write a very simple recipe, that does nothing but to create a file before running our application code.

```python
from argus.recipe.base import BaseRecipe

class MyBasicRecipe(BaseRecipe):
    def prepare(self, **kwargs):
        install_python = "...
        self._backend.remote_client.run_remote_cmd(install_python)
```

That’s all really. Now of course you can do almost anything with the recipe, by running commands into the underlying instance. This really means that when our application code runs, our environment can be fully prepared to accomodate its needs.

1.2.3 The tests

This will be the simplest part when writing integration tests using argus. As mentioned earlier, the tests are nothing more than unittest-like tests on steroids, so they should be pretty familiar to anyone who wrote unittests in their career. As a note, the tests themselves will run on the host where argus is installed, not in the VM instances spinned up by the backend and they are used for testing that something occurred, as expected, in those instances.

Let’s write a test which verifies for instance that our application created a bunch of expected files on the root drive of the operating system.

```python
from argus.tests import base

class MyTest(base.BaseTestCase):
    def test_files_created(self):
        list_files_cmd = "...
        files = self._backend.remote_client.run_remote_cmd(list_files_cmd)
        self.assertIn(some_file, files)
```

Just with this code, we now have all the required components that can be used to form a very simple argus integration tests workflow.

1.2. Writing integration tests
1.2.4 The scenario

The final step is to write the scenario, which will use all the other components we mentioned so far.

```python
from argus.scenarios.base import BaseScenario
class MyScenario(BaseScenario):
    backend_type = NovaBackend
    recipe_type = MyBasicRecipe
    test_classes = (MyTest, )
```

That’s it! Point your favorite Python test runner to a file which contains this setup for your integration tests to run.
2.1 The argus.backends.base Module

class argus.backends.base.BaseBackend(name=None, userdata=None, metadata=None, availability_zone=None)

Class for managing instances

The backend is used for building and managing an underlying instance, being it an OpenStack instance, OpenNebula instance or a containerized OS.

Parameters

- **name** – The name of the instance that will be created.
- **userdata** – If any, the userdata which will be available in the instance to the corresponding cloud initialization service.
- **metadata** – If any, the metadata which should be available in the instance to the corresponding cloud initialization service.

cleanup()

Destroy and cleanup the relevant resources.

Cleanup the resources created by setup_instance(), such as the keypairs, floating ips and credentials.

get_remote_client(**kwargs)

Get a remote client to the underlying instance.

remote_client

An abstract property which should return the default client.

setup_instance()

Setup an underlying instance.

class argus.backends.base.CloudBackend(name=None, userdata=None, metadata=None, availability_zone=None)

Base backend for cloud related tasks.

floating_ip()

Get the floating ip that was attached to the underlying instance.

get_remote_client(username=None, password=None, **kwargs)

Get a remote client

This is different than remote_client, because that will always return a client with predefined credentials, while this method allows for a fine-grained control over this aspect. password can be omitted if
authentication by SSH key is used. The **kwargs parameter can be used for additional options (currently none).

```python
instance_output (limit=None)
```
Get the underlying’s instance output, if any.

**Parameters**

- **limit** – Number of lines to fetch from the end of console log.

```python
instance_password()
```
Get the underlying instance password, if any.

```python
internal_instance_id()
```
Get the underlying’s instance id.

Gets the id depending on the internals of the backend.

```python
private_key()
```
Get the underlying private key.

```python
public_key()
```
Get the underlying public key.

```python
reboot_instance()
```
Reboot the underlying instance.

```python
save_instance_output (suffix=None)
```
Retrieve and save all data written through the COM port.

If a suffix is provided, then the log name is preceded by it.

### 2.2 The `argus.backends.windows` Module

```python
class argus.backends.windows.WindowsBackendMixin
```
Mixin backend tailored for interacting with Windows.

```python
get_remote_client (username=None, password=None, protocol=’http’, **kwargs)
```
Uses :class:`argus.util.WinRemoteClient` as underlying client.

```python
remote_client
```
A property which caches the result on access.

### 2.3 The `argus.backends.tempest.cloud` Module

```python
class argus.backends.tempest.cloud.NetworkWindowsBackend (name, userdata, metadata, availability_zone)
```
Backend for providing static network configuration.

Creates an additional internal network which will be bound explicitly with the new created instance.

```python
get_network_interfaces ()
```
Retrieve and parse network details from the compute node.

```python
setup_instance ()
```

```python
class argus.backends.tempest.cloud.RescueWindowsBackend (name, userdata, metadata, availability_zone)
```
Instance rescue Windows-based backend.
rescue_server()  
Rescue the underlying instance.

unrescue_server()  
Unrescue the underlying instance.

2.4 The argus.backends.tempest.manager Module

class argus.backends.tempest.manager.APIManager  
The APIManager for interacting between modules.
Manager which uses tempest modules for interacting with the OpenStack API.

cleanup_credentials()  
Cleanup any credentials created during the initialization.

create_keypair(name)  
Create a new keypair with the given name
This will return a new Keypair object, which provides access to the public, private key pair, as well as a method for destroying the keypair if needed.

get_mtu()  
Get the MTU value, from the backend.

instance_output (instance_id, limit)  
Get the console output, sent from the instance.

Parameters

• instance_id – The id of the instance for which the output will be retrieved.
• limit – Number of lines to fetch from the end of console log.

instance_password (instance_id, keypair)  
Get the password posted by the given instance.

Parameters

• instance_id – The id of the instance for which the password will be returned.
• keypair – A keypair whose private key can be used to decrypt the password.

instance_server (instance_id)  
Get more details about the given instance id.

primary_credentials()  
Get the primary credentials.

Get the underlying: class: tempest.common.isolated_creds.IsolatedCreds.

reboot_instance (instance_id)  
Reboot the instance with the given id.

class argus.backends.tempest.manager.Keypair(name, public_key, private_key, manager)  
A keypair container.

destroy()  
Destroy the current keypair.
2.5 The `argus.backends.tempest.tempest_backend` Module

```python
class argus.backends.tempest.tempest_backend.BaseTempestBackend(name, userdata, metadata, availability_zone):
    Base class for backends built on top of Tempest.

    Parameters
    • `name` – The name will be used for creating instances with this backend.
    • `userdata` – The userdata which will be available in the instance to the corresponding cloud initialization service.
    • `metadata` – The metadata which will be available in the instance to the corresponding cloud initialization service. It will be the content of the `meta` key in OpenStack’s metadata for instance.
    • `availability_zone` – The availability zone in which the underlying instance will be available.

    cleanup()
    Cleanup the underlying instance.

    In order for the backend to be useful again, call `setup_instance()` method for preparing another underlying instance.

    floating_ip()
    get_image_by_ref()
    get_mtu()

    instance_output(limit=128)
    Get the console output, sent from the instance.

    instance_password()

    instance_server()
    Get the instance server object.

    internal_instance_id()

    private_key()

    public_key()

    reboot_instance()

    setup_instance()
```

```python
class argus.backends.tempest.tempest_backend.BaseWindowsTempestBackend(name, userdata, metadata, availability_zone):
    Base Tempest backend for testing Windows.
```

Base Tempest backend for testing Windows.
2.6 The `argus.backends.heat.client` Module

`argus.backends.heat.client.heat_client(credentials, api_version=1)`

Get a new Heat client using the given credentials.

2.7 The `argus.backends.heat.heat_backend` Module

```python
class argus.backends.heat.heat_backend.BaseHeatBackend(name=None, userdata=None, metadata=None, availability_zone=None)
```

A backend which uses Heat as the driving core.

- `cleanup()`
  - Get the underlying floating ip.

- `floating_ip()`
  - Get the image object by its reference id.

- `get_image_by_ref()`
  - Get the image object by its reference id.

- `get_mtu()`
  - Get the console output, sent from the instance.

- `instance_output(limit=128)`
  - Get the console output, sent from the instance.

- `instance_password()`
  - Get the underlying instance password, if any.

- `instance_server()`
  - Get the instance server object.

- `internal_instance_id()`
  - Get the underlying’s instance id.
    - Gets the instance id depending on the internals of the backend.

- `private_key()`
  - Get the underlying private key.

- `public_key()`
  - Get the underlying public key.

- `reboot_instance()`
  - Reboot the underlying instance.

- `setup_instance()`

```python
class argus.backends.heat.heat_backend.WindowsHeatBackend(name=None, userdata=None, metadata=None, availability_zone=None)
```

Heat backend tailored to work with Windows platforms.

2.8 The `argus.recipes.base` Module

Contains base recipes functionality.
A recipe is a class which knows how to provision an instance, by installing and configuring it with what it’s necessary.

```python
class argus.recipes.base.BaseRecipe(backend)
    Base class for a recipe.
    A recipe is a way in which an instance can be provisioned with some easy steps.
    prepare(**kwargs)
        Call this method to provision an instance.
        Parameters kwarg – If the recipe requires, pass additional information in the kwarg parameter.
```

### 2.9 The `argus.recipes.cloud.base` Module

Base recipe for preparing instances for cloudbaseinit testing.

```python
class argus.recipes.cloud.base.BaseCloudbaseinitRecipe(backend)
    Base recipe for testing an instance with Cloudbaseinit.
    The method prepare() does all the necessary work for preparing a new instance. The executed steps are:
        • wait for boot completion.
        • get an install script for CloudbaseInit
        • installs CloudbaseInit
        • waits for the finalization of the installation.
    execution_prologue()
        Executed before any downloaded script.
        Do extra things to assure a successful remote powershell (and others) script execution.
    get_installation_script()
        Get the installation script for cloudbaseinit.
    inject_cbinit_config()
        Inject the conf in the instance.
    install_cbinit()
        Install the cloudbaseinit code.
    pre_sysprep()
        Run finalization code before sysprepping.
    prepare(service_type=None, **kwargs)
        Prepare the underlying instance.
        The following operations will be executed:
            • wait for boot completion
            • get an installation script for CloudbaseInit
            • install CloudbaseInit by running the previously downloaded file.
            • wait until the instance is up and running.
```
**prepare_cbinit_config** *(service_type)*

Prepare the config objects.

Prepare `self.cbinit_config` and `self.cbinit_unattend_conf` objects.

**replace_code** *

Do whatever is necessary to replace the code for cloudbaseinit.

**replace_install** *

Do whatever is necessary to replace the installation.

**set_mtu** *(interface=None, subinterface_name=None, mtu_value=None, store_type=None)*

Sets the MTU value for the underlying instance.

Sets the MTU value in order to avoid packet loss. More details about the parameters can be found below:

**sysprep** *

Do the final steps after installing cloudbaseinit.

This requires running sysprep on Windows, but on other platforms there might be no need for calling it.

**wait_cbinit_finalization** *

Wait for the finalization of cloudbaseinit.

**wait_for_boot_completion** *

Wait for the instance to finish up booting.

### 2.10 The **argus.recipes.cloud.windows** Module

Windows cloudbaseinit recipes.

**class argus.recipes.cloud.windows.AlwaysChangeLogonPasswordRecipe** *(backend)*

Always change the password at next logon.

**behaviour** = `always`

**class argus.recipes.cloud.windows.BaseNextLogonRecipe** *(backend)*

Useful for testing the next logon behaviour.

**behaviour** = None

**prepare_cbinit_config** *(service_type)*

**class argus.recipes.cloud.windows.ClearPasswordLogonRecipe** *(backend)*

Change the password at next logon if the password is from metadata.

**behaviour** = `clear_text_injected_only`

**class argus.recipes.cloud.windows.CloudbaseinitCloudstackRecipe** *(backend)*

Recipe for Cloudstack metadata service mocking.

**config_group** = `cloudstack`

**metadata_address** = `http://172.17.95.62/cloudstack`

**prepare_cbinit_config** *(service_type)*

**class argus.recipes.cloud.windows.CloudbaseinitCreateUserRecipe** *(backend)*

A recipe for creating the user created by cloudbaseinit.

The purpose is to use this recipe for testing that cloudbaseinit works, even when the user which should be created already exists.
class argus.recipes.cloud.windows.CloudbaseinitEC2Recipe(backend)
    Recipe for EC2 metadata service mocking.
    
    config_group = 'ec2'
    metadata_address = 'http://172.17.95.62'

class argus.recipes.cloud.windows.CloudbaseinitHTTPRecipe(backend)
    Recipe for http metadata service mocking.
    metadata_address = 'http://172.17.95.62/'

class argus.recipes.cloud.windows.CloudbaseinitImageRecipe(backend)
    Calibrate already sys-prepared cloudbase-init images.
    
    prepare(service_type=None, **kwargs)
    wait_cbinit_finalization()

class argus.recipes.cloud.windows.CloudbaseinitKeysRecipe(backend)
    Recipe that facilitates x509 certificates and public keys testing.
    
    prepare_cbinit_config(service_type)

class argus.recipes.cloud.windows.CloudbaseinitLocalScriptsRecipe(backend)
    Recipe for testing local scripts return codes.
    
    pre_sysprep()

class argus.recipes.cloud.windows.CloudbaseinitLongHostname(backend)
    Recipe for testing the netbios long hostname compatibility option.
    
    prepare_cbinit_config(service_type)

class argus.recipes.cloud.windows.CloudbaseinitMaasRecipe(backend)
    Recipe for Maas metadata service mocking.
    
    config_group = 'maas'
    metadata_address = 'http://172.17.95.62/
    
    prepare_cbinit_config(service_type)

class argus.recipes.cloud.windows.CloudbaseinitMockServiceRecipe(backend)
    A recipe for patching the cloudbaseinit’s conf with a custom server.
    
    Attributes:
    config_group: The group name specific to the metadata provider.
    config_entry: Field containing the metadata url name for the specified provider.
    metadata_address: The address for where to access the metadata service.
    
    config_entry = 'metadata_base_url'
    config_group = 'DEFAULT'
    metadata_address = 'metadata_url_value'
    
    prepare_cbinit_config(service_type)

class argus.recipes.cloud.windows.CloudbaseinitRecipe(backend)
    Recipe for preparing a Windows instance.
    
    execution_prologue()
get_installation_script()
Get installation script for CloudbaseInit.

inject_cbinit_config()
Inject the cloudbase-init config in the right place.

install_cbinit()
Proceed on checking if cloudbase-init should be installed.

pre_sysprep()

prepare_cbinit_config(service_type)
Prepare the cloudbase-init config.

replace_code()
Replace the code of cloudbaseinit.

replace_install()
Replace the cb-init installed files with the downloaded ones.
For the same file names, there will be a replace. The new ones will just be added and the other files will be left there. So it’s more like an update.

set_mtu(interface='ipv4', subinterface_name='Ethernet', mtu_value=1400, store_type='active')
sysprep()
Prepare the instance for the actual tests, by running sysprep.

wait_cbinit_finalization()

wait_for_boot_completion()

class argus.recipes.cloud.windows.CloudbaseinitScriptRecipe(backend)
A recipe which adds support for testing .exe scripts.

pre_sysprep()

class argus.recipes.cloud.windows.CloudbaseinitWinrmRecipe(backend)
A recipe for testing the WinRM configuration plugin.

prepare_cbinit_config(service_type)

2.11 The argus.scenarios.base Module

class argus.scenarios.base.BaseScenario(methodName='runTest')
Scenario which sets up an instance and prepares it using a recipe.

availability_zone = None
backend = None

backend_type = None
The backend class which will be used.

introspection = None

introspection_type = None
The introspection class which will be used.

metadata = None
The metadata that will be available in the instance.
This can be anything as long as the underlying backend supports it.
classmethod prepare_instance()
    Prepare the underlying instance.

classmethod prepare_recipe()
    Call the prepare method of the underlying recipe
    This method can be overwritten in the case the recipe’s prepare method needs special arguments passed down.

recipe = None

recipe_type = None
    The recipe class which will be used.

classmethod setUpClass()
    Prepare the scenario for running
    This means that the backend will be instantiated and an instance will be created and prepared. After the preparation is finished, the tests can run and can introspect the instance to check what they are supposed to be checking.

classmethod tearDownClass()
    Cleanup this scenario
    This usually means that any resource that was created in setUpClass() needs to be destroyed here.

test_classes = None
    A tuple of test classes which will be merged into the scenario.

userdata = None
    The userdata that will be available in the instance
    This can be anything as long as the underlying backend supports it.

class argus.scenarios.base.ScenarioMeta
    Metaclass for merging test methods from a given list of test cases.

    is_final()
        Check current class if is final.
        Checks if the class is final and if it has all the attributes set.

2.12 The argus.scenarios.cloud.base Module

class argus.scenarios.cloud.base.CloudScenario(methodName='runTest')
    Base scenario class for testing cloudbaseinit.

    classmethod prepare_recipe()
        Prepare the underlying recipe.
        Prepare the recipe using custom behavior tailored to cloudbaseinit.

        service_type = 'http'

2.13 The argus.scenarios.service_mock Module

class argus.scenarios.cloud.service_mock.BaseServiceApp(backend)
stop_me()
   Stop the current running cherrypy engine.

class argus.scenarios.cloud.service_mock.CloudstackMetadataServiceApp(backend)
   Metadata app for CloudStack service.
   latest(data_type, operation=None)
   meta_data(operation)
   service_offering()
   user_data(operation=None)

class argus.scenarios.cloud.service_mock.CloudstackPasswordManagerApp(backend)
   Metadata app for CloudStack password manager.
   index()
   password(password=None)
   saved_password()
   send_my_password()

class argus.scenarios.cloud.service_mock.EC2MetadataServiceApp(*args, **kwargs)
   Mock server for testing EC2 metadata service.
   default(*args)
   keydict
      Build a dictionary with all the public keys.
      Use as keys their indexes in increasing order starting with 0.
   public_keys(*remain)
      Mimic the behavior of EC2 metadata service.
      A first request to /public-keys will return all the available keys, each one per line in this form: “index=key-name”. Then, based on the number of keys and their indexes, will follow requests like /public-keys/<index>/openssh-key which returns the actual content.

class argus.scenarios.cloud.service_mock.HTTPKeysMetadataServiceApp(backend)
   Custom OpenStack http metadata.
   default(*args)

class argus.scenarios.cloud.service_mock.MaasMetadataServiceApp(backend)
   Metadata app for MaaS service.
   meta_data(operation=None)
   user_data()
   static x509()

class argus.scenarios.cloud.service_mock.MetadataServiceAppMixin
   Common metadata resources.
   instance_id()
   local_hostname()
   public_keys()

class argus.scenarios.cloud.service_mock.ServiceManager(services, backend)
   Creates the required mocked service processes.
2.14 The \texttt{argus.scenarios.windows} Module

\begin{verbatim}

  class argus.scenarios.cloud.windows.BaseServiceMockMixin
     Mixin class for mocking metadata services.

     In order to have support for mocked metadata services, set a list of \texttt{named()} entries in the class, as such:

     \begin{verbatim}
     class Test(BaseServiceMockMixin, CloudScenario):
         services = [
             named(application, script_name, host, port)
         ]
     \end{verbatim}

  classmethod prepare_instance()
  classmethod tearDownClass()

\end{verbatim}

\end{verbatim}

\begin{verbatim}

  class argus.scenarios.cloud.windows.CloudstackWindowsScenario
     Scenario for testing the Cloudstack metadata service.

     services = [service(application=<class 'argus.scenarios.cloud.service_mock.CloudstackMetadataServiceApp'>, script_name='', host='0.0.0.0', port=8080)]

  class argus.scenarios.cloud.windows.EC2WindowsScenario
     Scenario for testing the EC2 metadata service.

     services = [service(application=<class 'argus.scenarios.cloud.service_mock.EC2MetadataServiceApp'>, script_name='/2009-04-04/meta-data', host='0.0.0.0', port=80)]

  class argus.scenarios.cloud.windows.HTTPKeysWindowsScenario
     Scenario for testing custom OpenStack http metadata service.

     services = [service(application=<class 'argus.scenarios.cloud.service_mock.HTTPKeysMetadataServiceApp'>, script_name='/openstack', host='0.0.0.0', port=80)]

  class argus.scenarios.cloud.windows.MaaSWindowsScenario
     Scenario for testing the Maas metadata service.

     services = [service(application=<class 'argus.scenarios.cloud.service_mock.MaaSMetadataServiceApp'>, script_name='2012-03-01', host='0.0.0.0', port=80)]

  def argus.scenarios.cloud.windows.get_port_number(url)
     Gets the port number from a given url.

     Parameters  url -- String value of a URL.

     Return type  int

     Returns  The port value from the given url.

\end{verbatim}

\end{verbatim}

2.15 The \texttt{argus.client.base} Module

\begin{verbatim}

  class argus.client.base.BaseClient
     Get a remote client to a Windows instance.

     Parameters  hostname -- A hostname where the client should connect. This can be anything that
                             the client needs (an ip, a fully qualified domain name etc.).

\end{verbatim}

\end{verbatim}
run_remote_cmd(command, command_type=None)
Run the given remote command.

The command will be executed on the remote underlying server. It will return a tuple of three elements, stdout, stderr and the return code of the command.

2.16 The argus.client.windows Module

class argus.client.windows.WinRemoteClient (hostname, username, password, transport_protocol='http', cert_pem=None, cert_key=None)
Get a remote client to a Windows instance.

Parameters
• hostname – The ip where the client should be connected.
• username – The username of the client.
• password – The password of the remote client.
• transport_protocol – The transport for the WinRM protocol. Only http and https makes sense.
• cert_pem – Client authentication certificate file path in PEM format.
• cert_key – Client authentication certificate key file path in PEM format.

copy_file(filepath, remote_destination)
Copy the given filepath in the remote destination.

The remote destination is the file name where the content of filepath will be written.

read_file(filepath)
Get the content of the given file.

run_command(cmd, command_type='powershell')
Run the given command and return execution details.

Return type tuple
Returns stdout, stderr, exit_code

run_command_until_condition(cmd, retry_count=15, delay=10, command_type='powershell')
Run the given cmd until a condition cond occurs.

Parameters cond – A callable which receives the standard output returned by executing the command. It should return a boolean value, which tells to this function to stop execution.

Raises ArgusCLIError if there is output found in the standard error.

This method uses and behaves like run_command_with_retry but with an additional condition parameter.

run_command_verbose(cmd, command_type='powershell')
Run the given command and log anything it returns.

Do this with retrying support.

Return type string
Returns stdout
**run_command_with_retry** *(cmd, count=15, delay=10, command_type='powershell')*

Run the given *cmd* until succeeds.

**Parameters**

- **cmd** – A string, representing a command which needs to be executed on the underlying remote client.
- **count** – The number of retries which this function has. If the value is *None*, then the function will retry forever.
- **delay** – The number of seconds to sleep when retrying a command.

**Return type**  tuple

**Returns**  stdout, stderr, exit_code

**run_remote_cmd** *(cmd, command_type='powershell')*

Run the given remote command.

The command will be executed on the remote underlying server. It will return a tuple of three elements, stdout, stderr and the return code of the command.

**write_file** *(data, remote_destination)*

Copy the given data in the remote destination.

The remote destination is the file name where the content of filepath will be written.

## 2.17 The argus.util Module

**argus.util.decrypt_password** *(private_key, password)*

Decode password and unencrypts it with private key.

Requires openssl binary available in the path.

**argus.util.get_logger** *(name='argus', format_string='%(asctime)s - %(name)s - %(levelname)s - %(message)s', logging_file='argus.log')*

Obtain a new logger object.

The *name* parameter will be the name of the logger and *format_string* will be the format it will use for logging. *logging_file* is a file where the messages will be written.

**argus.util.get_resource** *(resource)*

Get the given resource from the list of known resources.

**class argus.util.cached_property** *(func, name=None)*

A property which caches the result on access.

**argus.util.run_once** *(func, state={}, errors={})*

A memoization decorator, whose purpose is to cache calls.

**argus.util.rand_name** *(name='')*

Generate a random name

If *name* is given, then it will be prepended to the generated string, separated by a minus sign.

**argus.util.get_public_keys** ()

Get the public_keys resource.

Used by the cloudbaseinit’s tests.
argus.util.get_certificate()
Get the certificate resource.
Used by the cloudbaseinit’s tests.

2.18 The argus.introspection.base Module

class argus.introspection.base.BaseInstanceIntrospection(remote_client)
Generic utility class for introspecting an instance.

Parameters

• conf – The configuration object used by argus.

• remote_client – A client which can be used by argus. This needs to be an instance of argus.remote_client.BaseClient.

2.19 The argus.introspection.cloud.base Module

class argus.introspection.cloud.base.CloudInstanceIntrospection(remote_client)
Introspection class for testing cloudbase-init.

get_cloudbaseinit_traceback()
Return the traceback, if any, from the cloudbaseinit’s logs.

get_cloudconfig_executed_plugins()
Get a dictionary of files, created by the cloud-config plugin.

The values are the actual file content.

get_disk_size()
Return the disk size from the instance.

get_group_members(group)
Get the members of the local group given.

get_instance_file_content(filepath)
Return the content of the given file from the instance.

get_instance_hostname()
Get the hostname of the instance.

get_instance_keys_path()
Return the authorized_keys file path from the instance.

get_instance_mtu()
Get the mtu value from the instance.

get_instance_ntp_peers()
Get the NTP peers from the instance.

get_network_interfaces()
Get IP available instance network adapters.

get_swap_status()
Get whether the swap memory is enabled or not.

Returns True if swap memory is enabled, False if not.

Return type bool
**get_timezone()**
Get the timezone of the instance.

**get_userdata_executed_plugins()**
Get the count of userdata executed plugins.

**list_location(location)**
Return the list of files and folder from the given location.

**username_exists(username)**
Check if the given username exists in the instance.

### 2.20 The *argus.introspection.cloud.windows* Module

**class** `argus.introspection.cloud.windows.Address(v4, v6)`

- **v4**
  Alias for field number 0

- **v6**
  Alias for field number 1

**class** `argus.introspection.cloud.windows.InstanceIntrospection(remote_client)`
Utilities for introspecting a Windows instance.

- **get_cloudbaseinit_traceback()**

- **get_cloudconfig_executed_plugins()**

- **get_disk_size()**

- **get_group_members(group)**

- **get_instance_file_content(filepath)**

- **get_instance_hostname()**

- **get_instance_keys_path()**

- **get_instance_mtu()**

- **get_instance_ntp_peers()**

- **get_instance_os_version()**
  Get the version of the underlying OS
  Return a tuple of two elements, the major and the minor version.

- **get_network_interfaces()**
  Get a list with dictionaries of network details.
  If a value is an empty string, then that value is missing.

- **get_service_triggers(service)**
  Get the triggers of the given service.
  Return a tuple of two elements, where the first is the start trigger and the second is the end trigger.

- **get_swap_status()**
  Get whether the swap memory is enabled or not.
  **Returns** True if swap memory is enabled, False if not.
Return type  bool

get_timezone()

get_user_flags(user)

getat_userdata_executed_plugins()

instance_exe_script_executed()

list_location(location)

username_exists(username)

class argus.introspection.cloud.windows.NICDetails(mac, address, gateway, netmask, dns, dhcp)

address
  Alias for field number 1

dhcp
  Alias for field number 5

dns
  Alias for field number 4

gateway
  Alias for field number 2

mac
  Alias for field number 0

netmask
  Alias for field number 3

argus.introspection.cloud.windows.escape_path(path)
  Escape the spaces in the given path in order to work with Powershell.

argus.introspection.cloud.windows.get_cbinit_dir(execute_function)
  Get the location of cloudbase-init from the instance.

argus.introspection.cloud.windows.get_cbinit_key(execute_function)
  Get the proper registry key for Cloudbase-init.

argus.introspection.cloud.windows.get_python_dir(execute_function)
  Find python directory from the cb-init installation.

argus.introspection.cloud.windows.set_config_option(option, value, execute_function)
  Set the value for the given option to value.
In order to work properly, argus needs specialized images, tailored for its needs. The current document tries to describe what steps are necessary for creating a proper image.

It assumes that the images are created using this repository https://github.com/PCManticore/windows-openstack-imaging-tools, instead of the original one, https://github.com/cloudbase/windows-openstack-imaging-tools, which I was unable to make it work for our scenarios.

For Microsoft Nano Server there are other steps, look at the bottom.

1. First, grab an ISO with the OS you want to create an image for.
2. You might want to configure Autounattend.xml to suit your needs and to be in a format acceptable for the operating system you’re working with. Make sure to use a proper product key in the <ProductKey> section.
3. Run `sudo ./create-autounattend-floppy.sh`, which will use the aforementioned Autounattend.xml in order to create an Autounattend.vfd, which will be used as a floppy device later on by another script.
4. Modify `create_img.sh` and add the ISO and the qcow2 paths.
5. Run `sudo ./create_img.sh`
6. If the Autounattend.xml was in a format known by the said OS, the installation should be automatic, requiring no manual input. Jump to the glance step. Otherwise, the following steps should be done manually or by customizing the installation scripts.
9. Create the CiAdmin user with the password Passw0rd and make it an Administrator.
10. Execute `Set-ExecutionPolicy Bypass` in an Administrator Powershell instance, in order to support running scripts from argus.
11. Activate RDP support:
    - run `SystemPropertiesAdvanced`
    - go to the Remote tab
    - Activate `Allow remote connections to this computer`
12. Run `.\install_git.ps1` from the cloned repository.
13. Run `.\Specialize.ps1` from the cloned repository.
14. Run `.\FirstLogon.ps1` from the cloned repository. Depending on how old your OS is, just doing this won’t be enough. This script will try to detect what virtio drivers are needed, but it might not support installing the latest version of virtio.

You might take a look at this compatibility table, taken from [https://pve.proxmox.com/wiki/Windows_VirtIO_Drivers](https://pve.proxmox.com/wiki/Windows_VirtIO_Drivers)

<table>
<thead>
<tr>
<th>OS</th>
<th>Numeric Version</th>
<th>dir for Storage / Balloon</th>
<th>dir for network</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2008 R2 / Windows 7</td>
<td>6.1</td>
<td>Win7 (32/64)</td>
<td>Win7 (32/64)</td>
</tr>
<tr>
<td>W2008 / Vista</td>
<td>6.0</td>
<td>Wlh (32/64)</td>
<td>Vista (32/64)</td>
</tr>
<tr>
<td>W2003</td>
<td>5.2</td>
<td>Wnet (32/64)</td>
<td>XP (32/64)</td>
</tr>
</tbody>
</table>

In some situations though, just installing these particular versions of virtio will still lead to BSODs when trying to boot the image through openstack and these combinations were found to be useful during our attempts to create new argus images:

<table>
<thead>
<tr>
<th>OS</th>
<th>virtio version for Storage / Balloon</th>
<th>virtio version for network</th>
</tr>
</thead>
<tbody>
<tr>
<td>W2008</td>
<td>0.54</td>
<td>0.102 or latest</td>
</tr>
<tr>
<td>W8.1</td>
<td>0.102 or latest</td>
<td>0.94</td>
</tr>
</tbody>
</table>

15. Run `.\Logon.ps1`. This script will configure WinRM and will finalize the image, by running sysprep.

16. After the creation script finished preparing the image, you can add it to glance and use its id in argus.conf.
Microsoft Nano Server

1. You need to create a new Nano Server image, you can follow these steps https://github.com/cloudbase/cloudbase-init-offline-install. Take care to specify that you don’t want to install cloudbase-init -AddCloudbaseInit:$false. You need to do this on a Windows host.

You also need to create an image for KVM (the script will install the proper VirtIO drivers) Platform KVM.

3. Allow ping, SMB and create the CiAdmin User. Mount the vhd and create this folder: C:\Windows\Setup\Scripts. Add there the PostInstall.ps1 and SetupComplet.cmd that you can find in this repo.

4. Install Git

You need to download in a windows machine Git for Windows Portable from https://git-scm.com/download/win, install it on that machine.

Copy PortableGit on the same vhd in C:\Program Files\PortableGit.

5. Now you can upload it using glance.
Indices and tables

- genindex
- modindex
- search
a

argus.backends.base, 7
argus.backends.heat.client, 11
argus.backends.heat.heat_backend, 11
argus.backends.tempest.cloud, 8
argus.backends.tempest.manager, 9
argus.backends.tempest.tempest_backend, 10
argus.backends.windows, 8
argus.client.base, 18
argus.client.windows, 19
argus.introspection.base, 21
argus.introspection.cloud.base, 21
argus.introspection.cloud.windows, 22
argus.recipes.base, 11
argus.recipes.cloud.base, 12
argus.recipes.cloud.windows, 13
argus.scenarios.base, 15
argus.scenarios.cloud.base, 16
argus.scenarios.cloud.service_mock, 16
argus.scenarios.cloud.windows, 18
argus.util, 20
CloudbaseinitCreateUserRecipe (class in argus.recipes.cloud.windows), 13
CloudbaseinitEC2Recipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitHTTPRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitImageRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitKeysRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitLocalScriptsRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitLongHostname (class in argus.recipes.cloud.windows), 14
CloudbaseinitMaasRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitMockServiceRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitScriptRecipe (class in argus.recipes.cloud.windows), 14
CloudbaseinitWinrmRecipe (class in argus.recipes.cloud.windows), 15
CloudInstanceIntrospection (class in argus.introspection.cloud.base), 21
CloudScenario (class in argus.scenarios.cloud.service_mock), 16
CloudstackMetadataServiceApp (class in argus.scenarios.cloud.service_mock), 17
CloudstackPasswordManagerApp (class in argus.scenarios.cloud.service_mock), 17
CloudstackWindowsScenario (class in argus.scenarios.cloud.windows), 18
config_entry (argus.recipes.cloud.windows.CloudbaseinitMockServiceRecipe
attribute), 14
config_group (argus.recipes.cloud.windows.CloudbaseinitCloudstackRecipe
attribute), 13
config_group (argus.recipes.cloud.windows.CloudbaseinitEC2Recipe
attribute), 14
config_group (argus.recipes.cloud.windows.CloudbaseinitMaasRecipe
attribute), 14
config_group (argus.recipes.cloud.windows.CloudbaseinitMockServiceRecipe
attribute), 14
copy_file() (argus.client.windows.WinRemoteClient
method), 19
create_keypair() (argus.backends.tempest.manager.APIManager
method), 9
decrypt_password() (in module argus.util), 20
default() (argus.scenarios.cloud.service_mock.EC2MetadataServiceApp
method), 17
default() (argus.scenarios.cloud.service_mock.HTTPKeysMetadataServiceApp
method), 17
destroy() (argus.backends.tempest.manager.Keypair
method), 9
dhcp (argus.introspection.cloud.windows.NICDetails
attribute), 23
dns (argus.introspection.cloud.windows.NICDetails
attribute), 23
desc() (argus.backends.tempest.manager.APIManager
method), 9
ed() (argus.backends.tempest.manager.APIManager
method), 9
EC2MetadataServiceApp (class in argus.scenarios.cloud.service_mock), 17
EC2WindowsScenario (class in argus.scenarios.cloud.windows), 18
escape_path() (in module argus.introspection.cloud.windows), 23
evaluation_prologue() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe
method), 12
evaluation_prologue() (argus.recipes.cloud.windows.CloudbaseinitRecipe
method), 14
floating_ip() (argus.backends.base.CloudBackend
method), 7
floating_ip() (argus.backends.heat.heat_backend.BaseHeatBackend
method), 11
floating_ip() (argus.backends.tempest.tempest_backend.BaseTempestBackend
method), 10
gateway (argus.introspection.cloud.windows.NICDetails
attribute), 23
get_cbinit_dir() (in module argus.introspection.cloud.windows.CloudbaseinitMockServiceRecipe
attribute), 14
get_certificate() (in module argus.util), 20
get_cloudconfig_executed_plugins() (argus.introspection.cloud.base.CloudInstanceIntrospection
method), 21
get_cloudconfig_executed_plugins() (argus.introspection.cloud.windows.InstanceIntrospection
method), 22
get_cloudbaseinit_traceback() (argus.introspection.cloud.base.CloudInstanceIntrospection
method), 21
get_cloudbaseinit_traceback() (argus.introspection.cloud.windows.InstanceIntrospection
method), 22
get_disk_size() (argus.introspection.cloud.base.CloudInstanceIntrospection
method), 21
get_disk_size() (argus.introspection.cloud.windows.InstanceIntrospection
method), 22
index (argus.introspection.cloud.windows.NICDetails
attribute), 23
index (argus.introspection.cloud.windows.NICDetails
attribute), 23

Index 35
index() (argus.scenarios.cloud.service_mock.CloudstackPasswordManagerApp method), 17
inject_cbinit_config() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 12
inject_cbinit_config() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
install_cbinit() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 12
install_cbinit() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
instance_exe_script_executed() (argus.introspection.cloud.windows.InstanceIntrospection method), 23
instance_id() (argus.scenarios.cloud.service_mock.MetadataserviceApp method), 17
instance_output() (argus.backends.base.CloudBackend method), 8
instance_output() (argus.backends.heat.heat_backend.BaseHeatBackend method), 11
instance_output() (argus.backends.tempest.manager.APIManager method), 9
instance_output() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 10
instance_password() (argus.backends.base.CloudBackend method), 8
instance_password() (argus.backends.heat.heat_backend.BaseHeatBackend method), 11
instance_password() (argus.backends.tempest.manager.APIManager method), 9
instance_password() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 10
instance_server() (argus.backends.heat.heat_backend.BaseHeatBackend method), 11
instance_server() (argus.backends.tempest.manager.APIManager method), 9
instance_server() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 10
InstanceIntrospection (class in argus.introspection.cloud.windows), 22
internal_instance_id() (argus.backends.base.CloudBackend method), 8
internal_instance_id() (argus.backends.heat.heat_backend.BaseHeatBackend method), 11
internal_instance_id() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 8
introspection() (argus.scenarios.base.BaseScenario attribute), 15
introspection_type (argus.scenarios.base.BaseScenario attribute), 15
is_final() (argus.scenarios.base.ScenarioMeta method), 16
KeyPair (class in argus.backends.tempest.manager), 9
MaasWindowsScenario (class in argus.scenarios.cloud.service_mock), 17
MaasMetadataServiceApp (class in argus.scenarios.cloud.service_mock), 17
meta_data() (argus.scenarios.cloud.service_mock.CloudstackMetadataServiceApp method), 17
meta_data() (argus.scenarios.cloud.service_mock.MaasMetadataServiceApp method), 17
metadata (argus.scenarios.base.BaseScenario attribute), 15
metadata_address (argus.recipes.cloud.windows.CloudbaseinitCloudstackRecipe attribute), 13
metadata_address (argus.recipes.cloud.windows.CloudbaseinitEC2Recipe attribute), 14
metadata_address (argus.recipes.cloud.windows.CloudbaseinitHTTPRecipe attribute), 14
metadata_address (argus.recipes.cloud.windows.CloudbaseinitMaasRecipe attribute), 14
MetaDataServiceAppMixin (class in argus.scenarios.cloud.service_mock), 17
named (class in argus.scenarios.cloud.windows), 18
netmask (argus.introspection.cloud.windows.NICDetails attribute), 23
NetworkWindowsBackend (class in argus.backends.tempest.cloud), 8
NICDetails (class in argus.introspection.cloud.windows), 23
password() (argus.scenarios.cloud.service_mock.CloudstackPasswordManagerApp method), 17
pre_sysprep() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 12
pre_sysprep() (argus.recipes.cloud.windows.CloudbaseinitCreateUserRecipe method), 11
pre_sysprep() (argus.recipes.cloud.windows.CloudbaseinitLocalScriptsRecipe method), 10
pre_sysprep() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 9
pre_sysprep() (argus.recipes.cloud.windows.CloudbaseinitScriptRecipe method), 9
prepare() (argus.recipes.base.BaseRecipe method), 12
prepare() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 12
prepare() (argus.recipes.cloud.windows.CloudbaseinitImageRecipe method), 14
prepare_ebninit_config() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 12
prepare_ebninit_config() (argus.recipes.cloud.windows.BaseNextLogonRecipe method), 13
prepare_ebninit_config() (argus.recipes.cloud.windows.CloudbaseinitCloudstackRecipe method), 13
prepare_ebninit_config() (argus.recipes.cloud.windows.CloudbaseinitKeysRecipe method), 14
prepare_ebninit_config() (argus.recipes.cloud.windows.CloudbaseinitLongHostname method), 14
prepare_ebninit_config() (argus.recipes.cloud.windows.CloudbaseinitMaasRecipe method), 14
prepare_ebninit_config() (argus.recipes.cloud.windows.CloudbaseinitMockServiceRecipe method), 14
prepare_ebninit_config() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
prepare_ebninit_config() (argus.recipes.cloud.windows.CloudbaseinitWinrmRecipe method), 15
prepare_instance() (argus.recipes.base.BaseRecipe method), 15
prepare_instance() (argus.recipes.cloud.base.CloudScenario class method), 16
prepare_recipe() (argus.scenarios.base.BaseScenario class method), 16
prepare_recipe() (argus.scenarios.cloud.base.CloudScenario class method), 16
primary_credentials() (argus.backends.tempest.manager.APIManager class method), 9
private_key() (argus.backends.base.CloudBackend method), 8
private_key() (argus.backends.heat.heat_backend.BaseHeatBackend method), 11
private_key() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 10
public_key() (argus.backends.base.CloudBackend method), 8
public_key() (argus.backends.heat.heat_backend.BaseHeatBackend method), 11
public_key() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 10
public_keys() (argus.scenarios.cloud.service_mock.EC2MetadataServiceApp method), 17
public_keys() (argus.scenarios.cloud.service_mock.MetadataServiceAppMixin method), 17
rand_name() (in module argus.util), 20
read_file() (argus.client.windows.WinRemoteClient method), 19
reboot_instance() (argus.backends.base.CloudBackend method), 8
reboot_instance() (argus.backends.heat.heat_backend.BaseHeatBackend method), 11
reboot_instance() (argus.backends.tempest.manager.APIManager method), 9
reboot_instance() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 10
recipe (argus.scenarios.base.BaseScenario attribute), 16
recipe_type (argus.scenarios.base.BaseScenario attribute), 16
remote_client (argus.backends.base.BaseBackend attribute), 7
remote_client (argus.backends.windows.WindowsBackendMixin attribute), 8
replace_code() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 13
replace_code() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
replace_install() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 13
replace_install() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
rescue_server() (argus.backends.tempest.cloud.RescueWindowsBackend method), 8
RescueWindowsBackend (class in argus.backends.tempest.cloud), 8
run_command() (argus.client.windows.WinRemoteClient method), 19
run_command_until_condition() (argus.client.windows.WinRemoteClient method), 19
run_command_verbose() (argus.client.windows.WinRemoteClient method), 19
run_command_with_retry() (argus.client.windows.WinRemoteClient method), 19
run_once() (in module argus.util), 20
run_remote_cmd() (argus.client.windows.WinRemoteClient method), 18
run_remote_cmd() (argus.client.windows.WinRemoteClient method), 20
save_instance_output() (argus.backends.base.CloudBackend method), 8
saved_password() (argus.scenarios.cloud.service_mock.CloudstackPasswordManagerApp method), 17
ScenarioMeta (class in argus.scenarios.base), 16
send_my_password() (argus.scenarios.cloud.service_mock.CloudstackPasswordManagerApp method), 17
service_offering() (argus.scenarios.cloud.service_mock.CloudstackMetadataServiceApp method), 17
service_type (argus.scenarios.cloud.base.CloudScenario attribute), 16
ServiceManager (class in argus.scenarios.cloud.service_mock), 17
services (argus.scenarios.cloud.windows.CloudstackWindowsScenario attribute), 18
services (argus.scenarios.cloud.windows.EC2WindowsScenario attribute), 18
services (argus.scenarios.cloud.windows.HTTPKeysWindowsScenario attribute), 18
services (argus.scenarios.cloud.windows.MaasWindowsScenario attribute), 18
set_config_option() (in module argus.introspection.cloud.windows), 23
set_mtu() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 13
set_mtu() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
setup_instance() (argus.backends.tempest.cloud.NetworkWindowsBackend method), 8
setup_instance() (argus.backends.tempest.tempest_backend.BaseTempestBackend method), 10
setUpClass() (argus.scenarios.base.BaseScenario class method), 16
stop_link (argus.scenarios.cloud.windows.named attribute), 18
stop_me() (argus.scenarios.cloud.service_mock.BaseServiceApp method), 16
sysprep() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 13
sysprep() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
tearDownClass() (argus.scenarios.base.BaseScenario class method), 16
tearDownClass() (argus.scenarios.cloud.windows.BaseServiceMockMixin class method), 16
terminate() (argus.scenarios.cloud.service_mock.ServiceManager method), 17
test_classes (argus.scenarios.base.BaseScenario attribute), 16
unrescue_server() (argus.backends.tempest.cloud.RescueWindowsBackend method), 9
user_data() (argus.scenarios.cloud.service_mock.CloudstackMetadataServiceApp method), 17
user_data() (argus.scenarios.cloud.service_mock.MaasMetadataServiceApp method), 17
userdata (argus.scenarios.base.BaseScenario attribute), 16
username_exists() (argus.introspection.cloud.base.CloudInstanceIntrospection method), 22
username_exists() (argus.introspection.cloud.windows.InstanceIntrospection method), 23
V
wait_cbinit_finalization() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 13
wait_cbinit_finalization() (argus.recipes.cloud.windows.CloudbaseinitImageRecipe method), 14
wait_cbinit_finalization() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
W
V
wait_for_boot_completion() (argus.recipes.cloud.base.BaseCloudbaseinitRecipe method), 13
wait_for_boot_completion() (argus.recipes.cloud.windows.CloudbaseinitRecipe method), 15
WindowsBackendMixin (class in argus.backends.windows), 8
WindowsHeatBackend (class in argus.backends.heat.heat_backend), 11
WinRemoteClient (class in argus.client.windows), 19
write_file() (argus.client.windows.WinRemoteClient method), 20

x
x509() (argus.scenarios.cloud.service_mock.MaasMetadataServiceApp static method), 17