
Kingpin Documentation

Release 0.3.0

Matt Wise, Mikhail Simin

May 11, 2016

1	Installation	3
1.1	Github Checkout/Install	3
1.2	Direct PIP Install	3
1.3	Zip File Packaging	4
2	Basic Use	5
2.1	Credentials	5
2.2	JSON-based DSL	6
2.2.1	Validation	6
2.2.2	The Script	6
3	Actors	11
3.1	Amazon Web Services	11
3.1.1	Documentation	11
3.1.2	CloudFormation	11
3.1.3	Elastic Load Balancing (ELB)	12
3.1.4	Identity and Access Management (IAM)	14
3.1.5	Simple Queue Service (SQS)	15
3.2	Grouping Actors	17
3.2.1	Async	17
3.2.2	Sync	18
3.3	Hipchat	19
3.3.1	Message	19
3.3.2	Topic	19
3.4	Librato	20
3.4.1	Annotation	20
3.5	Miscellaneous	20
3.5.1	Macro	20
3.5.2	Sleep	21
3.5.3	GenericHTTP	22
3.6	PackageCloud	22
3.6.1	Documentation	22
3.6.2	Delete	23
3.6.3	DeleteByDate	23
3.6.4	WaitForPackage	24
3.7	Pingdom	24
3.7.1	Pause	24
3.7.2	Unpause	24

3.8	RightScale	25
3.8.1	Documentation	25
3.8.2	Deployment	25
3.8.3	Alert Specs	26
3.8.4	Server Arrays	27
3.8.5	Multi Cloud Images	34
3.9	Rollbar	35
3.9.1	Deploy	35
3.10	Slack	36
3.10.1	Message	36
4	Security	39
4.1	URLLIB3 Warnings Disabled	39
5	Development	41
5.1	Setting up your Environment	41
5.1.1	Create your VirtualEnvironment	41
5.1.2	Check out the code	41
5.1.3	Install the test-specific dependencies	41
5.2	Testing	41
5.2.1	Unit Tests	41
5.2.2	Integration Tests	42
5.3	Class/Object Architecture	43
5.4	Setup	43
5.5	Actor Design	44
5.5.1	Hello World Actor Example	44
5.5.2	Required Options	45
5.5.3	Required Methods	46
5.5.4	Helper Methods/Objects	47
5.5.5	self.option	47
5.5.6	Exception Handling	47
5.6	Simple API Access Objects	47
5.6.1	HTTPBin Actor with the RestConsumer	47
5.6.2	Exception Handling in HTTP Requests	48
6	Full Module Docs	51
6.1	kingpin.actors.aws.base	51
6.2	kingpin.actors.aws.cloudformation	51
6.3	kingpin.actors.aws.elb	53
6.4	kingpin.actors.aws.iam	55
6.5	kingpin.actors.aws.settings	56
6.6	kingpin.actors.aws.sqs	56
6.7	kingpin.actors.base	57
6.8	kingpin.actors.exceptions	58
6.9	kingpin.actors.group	59
6.10	kingpin.actors.hipchat	61
6.11	kingpin.actors.librato	62
6.12	kingpin.actors.misc	63
6.13	kingpin.actors.packagecloud	65
6.14	kingpin.actors.pingdom	66
6.15	kingpin.actors.rightscale.api	67
6.16	kingpin.actors.rightscale.base	68
6.17	kingpin.actors.rightscale.server_array	68
6.18	kingpin.actors.rollbar	75

6.19	<code>kingpin.actors.slack</code>	76
6.20	<code>kingpin.actors.utils</code>	78
6.21	<code>kingpin.utils</code>	78
Python Module Index		81

Kingpin: the chief element of any system, plan, or the like.

Kingpin provides 3 main functions:

- **API Abstraction** - Job instructions are provided to Kingpin via a JSON based DSL (read below). The schema is strict and consistent from one action to another.
- **Automation Engine** - Kingpin leverages python's [tornado](#) engine.
- **Parallel Execution** - Aside from non-blocking network IO, Kingpin can execute any action in parallel with another. (Read [group.Async](#) below)

Installation

The simplest installation method is via [PyPI](#).

```
$ pip install --process-dependency-links kingpin
```

Note, we *strongly* recommend running the code inside a Python virtual environment. All of our examples below will show how to do this.

1.1 Github Checkout/Install

```
$ virtualenv .venv --no-site-packages
New python executable in .venv/bin/python
Installing setuptools, pip...done.
$ source .venv/bin/activate
(.venv) $ git clone https://github.com/Nextdoor/kingpin
Cloning into 'kingpin'...
remote: Counting objects: 1824, done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 1824 (delta 4), reused 0 (delta 0)
Receiving objects: 100% (1824/1824), 283.35 KiB, done.
Resolving deltas: 100% (1330/1330), done.
(.venv)$ cd kingpin/
(.venv)$ python setup.py install
zip_safe flag not set; analyzing archive contents...
...
```

1.2 Direct PIP Install

```
$ virtualenv .venv --no-site-packages
New python executable in .venv/bin/python
Installing setuptools, pip...done.
$ source .venv/bin/activate
(.venv) $ git clone https://github.com/Nextdoor/kingpin
(.venv)$ pip install --process-dependency-links git+https://github.com/Nextdoor/kingpin.git
Downloading/unpacking git+https://github.com/Nextdoor/kingpin.git
  Cloning https://github.com/Nextdoor/kingpin.git (to master) to /var/folders/j6/qyd2dp6n3f156h6xknn...
...
```

1.3 Zip File Packaging

For the purpose of highly reliable and fast installations, you can also execute `make package` to generate a Python-executable `.zip` file. This file is built with all of the dependencies installed inside of it, and can be executed on the command line very simply:

```
$ virtualenv .venv --no-site-packages
New python executable in .venv/bin/python
Installing setuptools, pip...done.
$ source .venv/bin/activate
$ make kingpin.zip
$ python kingpin.zip --version
0.2.5
```

VirtualEnv Note

Its not strictly necessary to set up the virtual environment like we did in the example above – but it helps prevent any confusion during the build process around what packages are available or are not.

Basic Use

```
$ kingpin --help
Usage: kingpin [json file] <options>

Options:
  --version            show program's version number and exit
  -h, --help           show this help message and exit
  -j JSON, --json=JSON Path to JSON Deployment File
  -d, --dry            Executes a dry run only.
  -l LEVEL, --level=LEVEL
                        Set logging level (INFO|WARN|DEBUG|ERROR)
  --debug              Equivalent to --level=DEBUG
  -c, --color          Colorize the log output
```

The simplest use cases of this code can be better understood by looking at the `simple.json` file. Executing it is as simple as this:

```
$ export RIGHTSCALE_TOKEN=xyz
$ export RIGHTSCALE_ENDPOINT=https://us-3.rightscale.com
$ (.venv)$ kingpin -j examples/simple.json -d
2014-09-01 21:18:09,022 INFO      [main stage (DRY Mode)] Beginning
2014-09-01 21:18:09,022 INFO      [stage 1 (DRY Mode)] Beginning
2014-09-01 21:18:09,022 INFO      [copy serverA (DRY Mode)] Beginning
2014-09-01 21:18:09,023 INFO      [copy serverB (DRY Mode)] Beginning
2014-09-01 21:18:09,027 INFO      [copy serverC (DRY Mode)] Beginning
2014-09-01 21:18:09,954 INFO      [copy serverA (DRY Mode)] Verifying that array "kingpin-integration
...
2014-09-01 21:18:14,533 INFO      [stage 3 (DRY Mode)] Finished, success? True
2014-09-01 21:18:14,533 INFO      [main stage (DRY Mode)] Finished, success? True
```

Kingpin always executes a dry run before executing. Each actor specifies their own definition of a dry run. Actors are designed to do as much checking in the dry run as possible to assure that everything will work during real execution.

It's possible, with extreme discouragement to skip the default dry run by setting `SKIP_DRY` environment variable.

2.1 Credentials

In an effort to keep the commandline interface of Kingpin simple, the majority of the configuration settings used at runtime are actually set as environment variables. Individual Kingpin Actors have their credential requirements documented in their specific documentation (*see below*).

2.2 JSON-based DSL

The entire model for the configuration is based on the concept of a JSON dictionary that contains at least one *actor* configuration. This JSON format is highly structured and must rigidly conform to the `kingpin.schema`.

2.2.1 Validation

The JSON file will be validated for schema-conformity as one of the first things that happens at load-time when the app starts up. If it fails, you will be notified immediately. This is performed in `misc.Macro` actor.

2.2.2 The Script

Definition: *The blueprint or roadmap that outlines a movie story through visual descriptions, actions of characters and their dialogue. The term “script” also applies to stageplays as well.*

Every Kingpin *script* is a chunk of JSON-encoded data that contains *actors*. Each *actor* configuration includes the same three parameters: *actor*, *desc* and *options*.

The simplest script will have a single configuration that executes a single *actor*. More complex scripts can be created with our `group.Sync` and `group.Async` actors which can be used to group together multiple *actors* and execute them in a predictable order.

Schema Description

The JSON schema is simple. We take a single JSON object that has a few fields:

- `actor` - A text-string describing the name of the Actor package and class. For example, `kingpin.actors.rightscale.server_array.Clone`, or `misc.Sleep`.
- `condition` - A bool or string that indicates whether or not to execute this actor.
- `desc` - A text-string describing the name of the stage or action. Meant to ensure that the logs are very human readable.
- `warn_on_failure` - True/False whether or not to ignore an Actors failure and return True anyways. Defaults to False, but if True a warning message is logged.
- `timeout` - Maximum time (in *seconds*) for the actor to execute before raising an `ActorTimedOut` exception is raised.
- `options` - A dictionary of key/value pairs that are required for the specific *actor* that you’re instantiating. See individual Actor documentation below for these options.

The simplest JSON file could look like this:

```
{ "desc": "Hipchat: Notify Oncall Room",
  "actor": "hipchat.Message",
  "condition": "true",
  "warn_on_failure": true,
  "timeout": 30,
  "options": {
    "message": "Beginning release %RELEASE%", "room": "Oncall"
  }
}
```

However, much more complex configurations can be created by using the `group.Sync` and `group.Async` actors to describe massively more complex deployments.

Conditional Execution

The `base.BaseActor` definition supports a `condition` parameter that can be used to enable or disable execution of an actor in a given Kingpin run. The field defaults to enabled, but takes many different values which allow you to choose whether or not to execute portions of your script.

Conditions that behave as `False`:

```
0, '0', 'False', 'FALse', 'FALSE'
```

Conditions that behave as `True`:

```
'any string', 'true', 'TRUE', '1', 1
```

Example usage:

```
{ "desc": "Hipchat: Notify Oncall Room",
  "actor": "hipchat.Message",
  "condition": "%SEND_MESSAGE%",
  "warn_on_failure": true,
  "options": {
    "message": "Beginning release %RELEASE%", "room": "Oncall"
  }
}
```

JSON Commenting

Because these JSON scripts can get quite large, Kingpin leverages the `demjson` package to parse your script. This package is slightly more graceful when handling syntax issues (extra commas, for example), and allows for JavaScript style commenting inside of the script.

Take this example:

```
{ "actor": "misc.Sleep",

  /* Cool description */
  "desc": 'This is funny',

  /* This shouldn't end with a comma, but does */
  "options": { "time": 30 }, }
```

The above example would fail to parse in most JSON parsers, but in `demjson` it works just fine.

Timeouts

By *default*, Kingpin actors are set to timeout after 3600s (1 hour). Each individual actor will raise an `ActorTimedOut` exception after this timeout has been reached. The `ActorTimedOut` exception is considered a `RecoverableActorFailure`, so the `warn_on_failure` setting applies here and thus the failure can be ignored if you choose to.

Additionally, you can override the *global default* setting on the commandline with an environment variable:

- `DEFAULT_TIMEOUT` - Time (in seconds) to use as the default actor timeout.

Here is an example log output when the timer is exceeded:

```
$ DEFAULT_TIMEOUT=1 SLEEP=10 kingpin -j examples/sleep.json
11:55:16 INFO Rehearsing... Break a leg!
11:55:16 INFO [DRY: Kingpin] Preparing actors from examples/sleep.json
11:55:16 INFO Rehearsal OK! Performing!
11:55:16 INFO Lights, camera ... action!
11:55:16 INFO [Kingpin] Preparing actors from examples/sleep.json
11:55:17 ERROR [Kingpin] kingpin.actors.misc.Macro._execute() execution exceeded deadline: 1s
11:55:17 ERROR [Sleep for some amount of time] kingpin.actors.misc.Sleep._execute() execution e
11:55:17 CRITICAL [Kingpin] kingpin.actors.misc.Macro._execute() execution exceeded deadline: 1s
11:55:17 CRITICAL [Sleep for some amount of time] kingpin.actors.misc.Sleep._execute() execution e
11:55:17 ERROR Kingpin encountered mistakes during the play.
11:55:17 ERROR kingpin.actors.misc.Macro._execute() execution exceeded deadline: 1s
```

Disabling the Timeout

You can disable the timeout on any actor by setting `timeout: 0` in your JSON.

Group Actor Timeouts

Group actors are special – as they do nothing but execute other actors. Although they support the `timeout: x` setting, they default to disabling the timeout (`timeout: 0`). This is done because the individual timeouts are generally owned by the individual actors. A single actor that fails will propagate its exception up the chain and through the Group actor just like any other actor failure.

As an example... If you take the following example code:

```
{ "desc": "Outer group",
  "actor": "group.Sync",
  "options": {
    "acts": [
      { "desc": "Sleep 10 seconds, but fail",
        "actor": "misc.Sleep",
        "timeout": 1,
        "warn_on_failure": true,
        "options": {
          "sleep": 10
        }
      },
      { "desc": "Sleep 2 seconds, but don't fail",
        "actor": "misc.Sleep",
        "options": {
          "sleep": 2
        }
      }
    ]
  }
}
```

The first `misc.Sleep` actor will fail, but only warn (`warn_on_failure=True`) about the failure. The parent `group.Sync` actor will continue on and allow the second `misc.Sleep` actor to continue.

Token-replacement

Environmental Tokens

In an effort to allow for more re-usable JSON files, *tokens* can be inserted into the raw JSON file like this `%TOKEN_NAME%`. These will then be dynamically swapped with environment variables found at execution time. Any missing environment variables will cause the JSON parsing to fail and will notify you immediately.

For an example, take a look at the `complex.json` file, and these examples of execution.

```
# Here we forget to set any environment variables
$ kingpin -j examples/complex.json -d
2014-09-01 21:29:47,373 ERROR      Invalid Configuration Detected: Found un-matched tokens in JSON st

# Here we set one variable, but miss the other one
$ RELEASE=0001a kingpin -j examples/complex.json -d
2014-09-01 21:29:56,027 ERROR      Invalid Configuration Detected: Found un-matched tokens in JSON st

# Finally we set both variables and the code begins...
$ OLD_RELEASE=0000a RELEASE=0001a kingpin -j examples/complex.json -d
2014-09-01 21:30:03,886 INFO       [Main (DRY Mode)] Beginning
2014-09-01 21:30:03,886 INFO       [Hipchat: Notify Oncall Room (DRY Mode)] Beginning
2014-09-01 21:30:03,886 INFO       [Hipchat: Notify Oncall Room (DRY Mode)] Sending message "Beginning
...

```

Contextual Tokens

Once the initial JSON files have been loaded up, we have a second layer of *tokens* that can be referenced. These tokens are known as *contextual tokens*. These *contextual tokens* are used during-runtime to swap out *strings* with *variables*. Currently only the `group.Sync` and `group.Async` actors have the ability to define usable tokens, but any actor can then reference these tokens.

Contextual tokens for simple variable behavior

```
{ "desc": "Send out hipchat notifications",
  "actor": "group.Sync",
  "options": {
    "contexts": [ { "ROOM": "Systems" } ],
    "acts": [
      { "desc": "Notify {ROOM}",
        "actor": "hipchat.Message",
        "options": {
          "room": "{ROOM}",
          "message": "Hey room .. I'm done with something"
        }
      }
    ]
  }
}
```

```
2015-01-14 15:03:16,840 INFO      [DRY: Send out hipchat notifications] Beginning 1 actions
2015-01-14 15:03:16,840 INFO      [DRY: Notify Systems] Sending message "Hey room .. I'm done with so

```

Contextual tokens used for iteration

```
{ "desc": "Send ending notifications...", "actor": "group.Async",
  "options": {
    "contexts": [
      { "ROOM": "Engineering", "WISDOM": "Get back to work" },
      { "ROOM": "Cust Service", "WISDOM": "Have a nice day" }
    ],
    "acts": [
      { "desc": "Notify {ROOM}",
        "actor": "hipchat.Message",
        "options": {
          "room": "{ROOM}",
          "message": "Hey room .. I'm done with the release. {WISDOM}"
        }
      }
    ]
  }
}
```

```
    }  
  ]  
}  
}
```

```
2015-01-14 15:02:22,165 INFO      [DRY: Send ending notifications...] Beginning 2 actions  
2015-01-14 15:02:22,165 INFO      [DRY: Notify Engineering] Sending message "Hey room .. I'm done wit  
2015-01-14 15:02:22,239 INFO      [DRY: Notify Cust Service] Sending message "Hey room .. I'm done w
```

Early Actor Instantiation

Again, in an effort to prevent mid-run errors, we pre-instantiate all Actor objects all at once before we ever begin executing code. This ensures that major typos or misconfigurations in the JSON will be caught early on.

Definition: *a participant in an action or process.*

3.1 Amazon Web Services

3.1.1 Documentation

`kingpin.actors.aws.base`

The AWS Actors allow you to interact with the resources (such as SQS and ELB) inside your Amazon AWS account. These actors all support dry runs properly, but each actor has its own caveats with `dry=True`. Please read the instructions below for using each actor.

Required Environment Variables

AWS_ACCESS_KEY_ID Your AWS access key

AWS_SECRET_ACCESS_KEY Your AWS secret

3.1.2 CloudFormation

`kingpin.actors.aws.cloudformation`

class `kingpin.actors.aws.cloudformation.Create(*args, **kwargs)`

Creates a CloudFormation stack.

Creates a CloudFormation stack from scratch and waits until the stack is fully built before exiting the actor.

Options

Capabilities A list of CF capabilities to add to the stack.

Disable_rollback Set to True to disable rollback of the stack if creation failed.

Name The name of the queue to create

Parameters A dictionary of key/value pairs used to fill in the parameters for the CloudFormation template.

Region AWS region (or zone) string, like 'us-west-2'

Template String of path to CloudFormation template. Can either be in the form of a local file path (ie, /my_template.json) or a URI (ie https://my_site.com/cf.json).

Timeout_in_minutes The amount of time that can pass before the stack status becomes CREATE_FAILED.

Examples

```
{ "desc": "Create production backend stack",
  "actor": "aws.cloudformation.Create",
  "options": {
    "capabilities": [ "CAPABILITY_IAM" ],
    "disable_rollback": true,
    "name": "%CF_NAME%",
    "parameters": {
      "test_param": "%TEST_PARAM_NAME%",
    },
    "region": "us-west-1",
    "template": "/examples/cloudformation_test.json",
    "timeout_in_minutes": 45,
  }
}
```

Dry Mode

Validates the template, verifies that an existing stack with that name does not exist. Does not create the stack.

class kingpin.actors.aws.cloudformation.**Delete** (*args, **kwargs)
Deletes a CloudFormation stack

Options

Name The name of the queue to create

Region AWS region (or zone) string, like ‘us-west-2’

Examples

```
{ "desc": "Create production backend stack",
  "actor": "aws.cloudformation.Create",
  "options" {
    "region": "us-west-1",
    "name": "%CF_NAME%",
  }
}
```

Dry Mode

Validates that the CF stack exists, but does not delete it.

3.1.3 Elastic Load Balancing (ELB)

kingpin.actors.aws.elb

class kingpin.actors.aws.elb.**WaitUntilHealthy** (*args, **kwargs)
Wait indefinitely until a specified ELB is considered “healthy”.

This actor will loop infinitely until a healthy threshold of the ELB is met. The threshold can be reached when the count as specified in the options is less than or equal to the number of InService instances in the ELB.

Another situation is for count to be a string specifying a percentage (see examples). In this case the percent of InService instances has to be greater than the count percentage.

Options

Name The name of the ELB to operate on

Count Number, or percentage of InService instance to consider this ELB healthy

Region AWS region (or zone) name, such as us-east-1 or us-west-2

Examples

```
{ "actor": "aws.elb.WaitUntilHealthy",
  "desc": "Wait until production-frontend has 16 hosts",
  "options": {
    "name": "production-frontend",
    "count": 16,
    "region": "us-west-2"
  }
}
```

```
{ "actor": "aws.elb.WaitUntilHealthy",
  "desc": "Wait until production-frontend has 85% of hosts in-service",
  "options": {
    "name": "production-frontend",
    "count": "85%",
    "region": "us-west-2"
  }
}
```

Dry Mode

This actor performs the finding of the ELB as well as calculating its health at all times. The only difference in dry mode is that it will not re-count the instances if the ELB is not healthy. A log message will be printed indicating that the run is dry, and the actor will exit with success.

class kingpin.actors.aws.elb.**SetCert** (*args, **kwargs)

Find a server cert in IAM and use it for a specified ELB.

Options

Region (str) AWS region (or zone) name, like us-west-2

Name (str) Name of the ELB

Cert_name (str) Unique IAM certificate name, or ARN

Port (int) Port associated with the cert. (default: 443)

Example

```
{ "actor": "aws.elb.SetCert",
  "desc": "Run SetCert",
  "options": {
    "cert_name": "new-cert",
    "name": "some-elb",
    "region": "us-west-2"
  }
}
```

Dry run

Will check that ELB and Cert names are existent, and will also check that the credentials provided for AWS have access to the new cert for ssl.

class kingpin.actors.aws.elb.**RegisterInstance** (*args, **kwargs)

Add an EC2 instance to a load balancer.

Options

Elb (str) Name of the ELB

Instances (str, list) Instance id, or list of ids. Default “self” id.

Region (str) AWS region (or zone) name, like us-west-2

Enable_zones (bool) add all available AZ to the elb. Default: True

Example

```
{ "actor": "aws.elb.RegisterInstance",
  "desc": "Run RegisterInstance",
  "options": {
    "elb": "prod-loadbalancer",
    "instances": "i-123456",
    "region": "us-east-1",
  }
}
```

Dry run

Will find the specified ELB, but not take any actions regarding instances.

class kingpin.actors.aws.elb.**DeregisterInstance** (*args, **kwargs)
Remove EC2 instance(s) from an ELB.

Options

Elb (str) Name of the ELB

Instances (str, list) Instance id, or list of ids

Region (str) AWS region (or zone) name, like us-west-2

Example

```
{ "actor": "aws.elb.DeregisterInstance",
  "desc": "Run DeregisterInstance",
  "options": {
    "elb": "fill-in",
    "instances": "fill-in",
    "region": "fill-in"
  }
}
```

Dry run

Will find the ELB but not take any actions regarding the instances.

3.1.4 Identity and Access Management (IAM)

kingpin.actors.aws.iam

class kingpin.actors.aws.iam.**UploadCert** (*args, **kwargs)
Uploads a new SSL Cert to AWS IAM.

Options

Private_key_path (str) Path to the private key.

Path (str) The AWS “path” for the server certificate. Default: “/”

Public_key_path (str) Path to the public key certificate.

Name (str) The name for the server certificate.

Cert_chain_path (str) Path to the certificate chain. Optional.

Example

```
{ "actor": "aws.iam.UploadCert",
  "desc": "Upload a new cert",
  "options": {
    "name": "new-cert",
    "private_key_path": "/cert.key",
    "public_key_path": "/cert.pem",
    "cert_chain_path": "/cert-chain.pem"
  }
}
```

Dry run

Checks that the passed file paths are valid. In the future will also validate that the files are of correct format and content.

class kingpin.actors.aws.iam.**DeleteCert** (*args, **kwargs)
Delete an existing SSL Cert in AWS IAM.

Options

Name (str) The name for the server certificate.

Example

```
{ "actor": "aws.iam.DeleteCert",
  "desc": "Run DeleteCert",
  "options": {
    "name": "fill-in"
  }
}
```

Dry run

Will find the cert by name or raise an exception if it's not found.

3.1.5 Simple Queue Service (SQS)

kingpin.actors.aws.sqs

class kingpin.actors.aws.sqs.**Create** (*args, **kwargs)
Creates a new SQS queue with the specified name

Options

Name (str) The name of the queue to create

Region (str) AWS region (or zone) string, like 'us-west-2'

Examples

```
{ "actor": "aws.sqs.Create",
  "desc": "Create queue named async-tasks",
  "options": {
    "name": "async-tasks",

```

```
    "region": "us-east-1",  
  }  
}
```

Dry Mode

Will not create any queue, or even contact SQS. Will create a mock.Mock object and exit with success.

class kingpin.actors.aws.sqs.**Delete** (*args, **kwargs)

Deletes the SQS queues

Note: **even if it's not empty**

Options

Name (str) The name of the queue to destroy

Region (str) AWS region (or zone) string, like 'us-west-2'

Idempotent (bool) Will not raise errors if no matching queues are found. (default: False)

Examples

```
{ "actor": "aws.sqs.Delete",  
  "desc": "Delete queue async-tasks",  
  "options": {  
    "name": "async-tasks",  
    "region": "us-east-1"  
  }  
}
```

```
{ "actor": "aws.sqs.Delete",  
  "desc": "Delete queues with 1234 in the name",  
  "options": {  
    "name": "1234",  
    "region": "us-east-1"  
  }  
}
```

Dry Mode

Will find the specified queue, but will have a noop regarding its deletion. Dry mode will fail if no queues are found, and idempotent flag is set to False.

class kingpin.actors.aws.sqs.**WaitUntilEmpty** (*args, **kwargs)

Wait indefinitely until for SQS queues to become empty

This actor will loop infinitely as long as the count of messages in at least one queue is greater than zero. SQS does not guarantee exact count, so this can return a stale value if the number of messages in the queue changes rapidly.

Options

Name (str) The name or regex pattern of the queues to operate on

Region (str) AWS region (or zone) string, like 'us-west-2'

Required (bool) Fail if no matching queues are found. (default: False)

Examples

```
{ "actor": "aws.sqs.WaitUntilEmpty",  
  "desc": "Wait until release-0025a* queues are empty",  
  "options": {
```

```

    "name": "release-0025a",
    "region": "us-east-1",
    "required": true
  }
}

```

Dry Mode

This actor performs the finding of the queue, but will pretend that the count is 0 and return success. Will fail even in dry mode if `required` option is set to True and no queues with the name pattern are found.

3.2 Grouping Actors

3.2.1 Async

class `kingpin.actors.group.Async(*args, **kwargs)`

Execute several `kingpin.actors.base.BaseActor` objects asynchronously.

Groups together a series of Actors and executes them asynchronously - waiting until all of them finish before returning.

Options

Concurrency Max number of concurrent executions. This will fire off N executions in parallel, and continue with the remained as soon as the first execution is done. This is faster than creating N Sync executions.

Acts An array of individual Actor definitions.

Contexts A list of dictionaries with *contextual tokens* to pass into the actors at instantiation time. If the list has more than one element, then every actor defined in `acts` will be instantiated once for each item in the `contexts` list.

Timeouts

Timeouts are disabled specifically in this actor. The sub-actors can still raise their own `kingpin.actors.exceptions.ActorTimedOut` exceptions, but since the group actors run an arbitrary number of sub actors, we have chosen to not have this actor specifically raise its own `kingpin.actors.exceptions.ActorTimedOut` exception unless the user sets the `timeout` setting.

Examples

Clone two arrays quickly.

```

{ "desc": "Clone two arrays",
  "actor": "group.Async",
  "options": {
    "contexts": [
      { "ARRAY": "NewArray1" },
      { "ARRAY": "NewArray2" }
    ],
    "acts": [
      { "desc": "do something",
        "actor": "server_array.Clone",
        "options": {
          "source": "template",
          "dest": "{ARRAY}"
        }
      }
    ]
  }
}

```

```
    }  
  }  
]  
}  
}
```

Dry Mode

Passes on the Dry mode setting to the sub-actors that are called.

Failure

In the event that one or more `acts` fail in this group, the entire group acts will return a failure to Kingpin. Because multiple actors are executing all at the same time, the all of these actors will be allowed to finish before the failure is returned.

3.2.2 Sync

class `kingpin.actors.group.Sync(*args, **kwargs)`

Execute a series of `kingpin.actors.base.BaseActor` synchronously.

Groups together a series of Actors and executes them synchronously in the order that they were defined.

Options

Acts An array of individual Actor definitions.

Contexts A list of dictionaries with *contextual tokens* to pass into the actors at instantiation time. If the list has more than one element, then every actor defined in `acts` will be instantiated once for each item in the `contexts` list.

Timeouts

Timeouts are disabled specifically in this actor. The sub-actors can still raise their own `kingpin.actors.exceptions.ActorTimedOut` exceptions, but since the group actors run an arbitrary number of sub actors, we have chosen to not have this actor specifically raise its own `kingpin.actors.exceptions.ActorTimedOut` exception unless the user sets the `timeout` setting.

Examples

Creates two arrays ... but sleeps 60 seconds between the two, then does not sleep at all after the last one:

```
{ "desc": "Clone, then sleep ... then clone, then sleep shorter...",  
  "actor": "group.Sync",  
  "options": {  
    "contexts": [  
      { "ARRAY": "First", "SLEEP": "60", },  
      { "ARRAY": "Second", "SLEEP": "0", }  
    ],  
    "acts": [  
      { "desc": "do something",  
        "actor": "server_array.Clone",  
        "options": {  
          "source": "template",  
          "dest": "{ARRAY}"  
        }  
      },  
      { "desc": "sleep",  
        "actor": "misc.Sleep",
```



```

        "options": {
            "sleep": "{SLEEP}",
        }
    ]
}

```

Dry Mode

Passes on the Dry mode setting to the acts that are called. Does **not** stop execution when one of the acts fails. Instead Group actor will finish all acts with warnings, and raise an error at the end of execution.

This provides the user with an insight to all the errors that are possible to encounter, rather than abort and quit on the first one.

Failure

In the event that an act fails, this actor will return the failure immediately. Because the acts are executed in-order of definition, the failure will prevent any further acts from executing.

The behavior is different in the dry run (read above.)

3.3 Hipchat

3.3.1 Message

class kingpin.actors.hipchat.**Message** (*args, **kwargs)

Sends a message to a room in HipChat.

Options

Room (str) The string-name (or ID) of the room to send a message to

Message (str) Message to send

Examples

```

{ "actor": "hipchat.Message",
  "desc": "Send a message!",
  "options": {
    "room": "Operations",
    "message": "Beginning Deploy: v1.2"
  }
}

```

Dry Mode

Fully supported – does not actually send messages to a room, but validates that the API credentials would have access to send the message using the HipChat `auth_test` optional API argument.

3.3.2 Topic

class kingpin.actors.hipchat.**Topic** (*args, **kwargs)

Sets a HipChat room topic.

Options

• **room** - The string-name (or ID) of the room to set the topic of

- `topic` - String of the topic to send

Examples

```
{ "actor": "hipchat.Topic",
  "desc": "set the room topic",
  "options": {
    "room": "Operations",
    "topic": "Latest Deployment: v1.2"
  }
}
```

Dry Mode

Fully supported – does not actually set a room topic, but validates that the API credentials would have access to set the topic of the room requested.

3.4 Librato

3.4.1 Annotation

class `kingpin.actors.librato.Annotation(*args, **kwargs)`

Librato Annotation Actor

Posts an Annotation to Librato.

Options

Title The title of the annotation

Description The description of the annotation

Name Name of the metric to annotate

Examples

```
{ "actor": "librato.Annotation",
  "desc": "Mark our deployment",
  "options": {
    "title": "Deploy",
    "description": "Version: 0001a",
    "name": "production_releases"
  }
}
```

Dry Mode

Currently does not actually do anything, just logs dry mode.

3.5 Miscellaneous

3.5.1 Macro

class `kingpin.actors.misc.Macro(*args, **kwargs)`

Parses a kingpin JSON file, instantiates and executes it.

Parse JSON

Kingpin JSON has 2 passes at its validity. JSON syntax must be valid, with the exception of a few useful deviations allowed by `demjson` parser. Main one being the permission of inline comments via `/* this */` syntax.

The second pass is validating the Schema. The JSON file will be validated for schema-conformity as one of the first things that happens at load-time when the app starts up. If it fails, you will be notified immediately.

Lastly after JSON is established to be valid, all the tokens are replaced with their specified value. Any key/value pair passed in the `tokens` option will be available inside of the JSON file as `%KEY%` and replaced with the value at this time.

In a situation where nested Macro executions are invoked the tokens *do not* propagate from outer macro into the inner. This allows to reuse token names, but forces the user to specify every token needed. Similarly, if environment variables are used for token replacement in the main file, these tokens are not available in the subsequent macros.

Pre-Instantiation

In an effort to prevent mid-run errors, we pre-instantiate all Actor objects all at once before we ever begin executing code. This ensures that major typos or misconfigurations in the JSON will be caught early on.

Execution

`misc.Macro` actor simply calls the `execute()` method of the most-outer actor; be it a single action, or a group actor.

Options

File String of local path to a JSON file.

Tokens Dictionary to search/replace within the file.

Examples

```
{ "desc": "Stage 1",
  "actor": "misc.Macro",
  "options": {
    "file": "deployment/stage-1.json",
    "tokens": {
      "TIMEOUT": 360,
      "RELEASE": "%RELEASE%"
    }
  }
}
```

Dry Mode

Fully supported – instantiates the actor inside of JSON with `dry=True`. The behavior of the consecutive actor is unique to each; read their description for more information on dry mode.

3.5.2 Sleep

`class kingpin.actors.misc.Sleep(desc, options, dry=False, warn_on_failure=False, condition=True, init_context={}, timeout=None)`

Sleeps for an arbitrary number of seconds.

Options

Sleep Integer of seconds to sleep.

Examples

```
{ "actor": "misc.Sleep",
  "desc": "Sleep for 60 seconds",
  "options": {
    "sleep": 60
  }
}
```

Dry Mode

Fully supported – does not actually sleep, just pretends to.

3.5.3 GenericHTTP

class kingpin.actors.misc.**GenericHTTP** (*desc, options, dry=False, warn_on_failure=False, condition=True, init_context={}, timeout=None*)

A very simple actor that allows GET/POST methods over HTTP.

Does a GET or a POST to a specified URL.

Options

Url Destination URL

Data Optional POST data as a `dict`

Username Optional for HTTPAuth.

Password Optional for HTTPAuth.

Examples

```
{ "actor": "misc.GenericHTTP",
  "desc": "Make a simple web call",
  "options": {
    "url": "http://example.com/rest/api/v1?id=123&action=doit",
    "username": "secret",
    "password": "%SECRET_PASSWORD%"
  }
}
```

Dry Mode

Will not do anything in dry mode except print a log statement.

3.6 PackageCloud

3.6.1 Documentation

`kingpin.actors.packagecloud`

The packagecloud actor allows you to perform maintenance operations on repositories hosted by packagecloud.io using their API:

<https://packagecloud.io/docs/api>

Required Environment Variables

PACKAGECLOUD_ACCOUNT	packagecloud	account	name,	i.e.
	https://packagecloud.io/PACKAGECLOUD_ACCOUNT			

PACKAGECLOUD_TOKEN packagecloud API Token

3.6.2 Delete

class kingpin.actors.packagecloud.**Delete** (*args, **kwargs)

Deletes packages from a PackageCloud repo.

Searches for packages that match the `packages_to_delete` regex pattern and deletes them. If `number_to_keep` is set, we always at least this number of versions of the given package intact in the repo. Also if `number_to_keep` is set, the older versions of a package (based on upload time) packages will be deleted first effectively leaving newer packages in the repo.

Options

Number_to_keep Keep at least this number of each package (defaults to 0)

Packages_to_delete Regex of packages to delete, e.g. `pkg1|pkg2`

Repo Which packagecloud repo to delete from

Examples

```
{ "desc": "packagecloud Delete example",
  "actor": "packagecloud.Delete",
  "options": {
    "number_to_keep": 10,
    "packages_to_delete": "deleteme",
    "repo": "test"
  }
}
```

3.6.3 DeleteByDate

class kingpin.actors.packagecloud.**DeleteByDate** (*args, **kwargs)

Deletes packages from a PackageCloud repo older than X.

Adds additional functionality to the `Delete` class with a `older_than` option. Only packages older than that number of seconds will be deleted.

Options

Number_to_keep Keep at least this number of each package (defaults to 0)

Older_than Delete packages created before this number of seconds

Packages_to_delete Regex of packages to delete, e.g. `pkg1|pkg2`

Repo Which packagecloud repo to delete from

Examples

```
{ "desc": "packagecloud DeleteByDate example",
  "actor": "packagecloud.DeleteByDate",
  "options": {
    "number_to_keep": 10,
    "older_than": 600,
    "packages_to_delete": "deleteme",
    "repo": "test"
  }
}
```

3.6.4 WaitForPackage

class kingpin.actors.packagecloud.**WaitForPackage** (*args, **kwargs)
Searches for a package that matches name and version until found or a timeout occurs.

Options

- Name** Name of the package to search for as a regex
- Version** Version of the package to search for as a regex
- Repo** Which packagecloud repo to delete from
- Sleep** Number of seconds to sleep for between each search

Examples

```
{ "desc": "packagecloud WaitForPackage example",  
  "actor": "packagecloud.WaitForPackage",  
  "options": {  
    "name": "findme",  
    "version": "0.1",  
    "repo": "test",  
    "sleep": 10,  
  }  
}
```

3.7 Pingdom

3.7.1 Pause

class kingpin.actors.pingdom.**Pause** (*args, **kwargs)
Start Pingdom Maintenance.

Pause a particular “check” on Pingdom.

Options

- Name** (Str) Name of the check

Example

```
{ "actor": "pingdom.Pause",  
  "desc": "Run Pause",  
  "options": {  
    "name": "fill-in"  
  }  
}
```

Dry run

Will assert that the check name exists, but not take any action on it.

3.7.2 Unpause

class kingpin.actors.pingdom.**Unpause** (*args, **kwargs)
Stop Pingdom Maintenance.

Unpause a particular “check” on Pingdom.

Options**Name** (Str) Name of the check**Example**

```
{ "actor": "pingdom.Unpause",
  "desc": "Run unpause",
  "options": {
    "name": "fill-in"
  }
}
```

Dry run

Will assert that the check name exists, but not take any action on it.

3.8 RightScale

3.8.1 Documentation

`kingpin.actors.rightscale.base`

The RightScale Actors allow you to interact with resources inside your Rightscale account. These actors all support dry runs properly, but each actor has its own caveats with `dry=True`. Please read the instructions below for using each actor.

Required Environment Variables

RIGHTSCALE_TOKEN RightScale API Refresh Token (from the *Account Settings/API Credentials* page)

RIGHTSCALE_ENDPOINT Your account-specific API Endpoint (defaults to <https://my.rightscale.com>)

exception `kingpin.actors.rightscale.base.ArrayNotFound`
Raised when a ServerArray could not be found.

exception `kingpin.actors.rightscale.base.ArrayAlreadyExists`
Raised when a ServerArray already exists by a given name.

3.8.2 Deployment

`kingpin.actors.rightscale.deployment`

class `kingpin.actors.rightscale.deployment.Create(*args, **kwargs)`
Creates a RightScale deployment.

Options match the documentation in RightScale: <http://reference.rightscale.com/api1.5/resources/ResourceDeployments.html>

Options

Name The name of the deployment to be created.

Description The description of the deployment to be created. (*optional*)

Server_tag_scope The routing scope for tags for servers in the deployment. Can be 'deployment' or 'account' (*optional*, default: deployment)

```
class kingpin.actors.rightscale.deployment.Destroy(*args, **kwargs)
```

Deletes a RightScale deployment.

Options match the documentation in RightScale: <http://reference.rightscale.com/api1.5/resources/ResourceDeployments.html>

Options

Name The name of the deployment to be deleted.

3.8.3 Alert Specs

```
kingpin.actors.rightscale.alerts
```

```
class kingpin.actors.rightscale.alerts.Create(*args, **kwargs)
```

Create a RightScale Alert Spec

Options match the documentation in RightScale: <http://reference.rightscale.com/api1.5/resources/ResourceAlertSpecs.html#create>

Options

Array The name of the Server or ServerArray to create the AlertSpec on.

Strict_array Whether or not to fail if the Server/ServerArray does not exist. (default: False)

Condition The condition (operator) in the condition sentence. (>, >=, <, <=, ==, !=)

Description The description of the AlertSpec. (*optional*)

Duration The duration in minutes of the condition sentence. (^d+\$)

Escalation_name Escalate to the named alert escalation when the alert is triggered. (*optional*)

File The RRD path/file_name of the condition sentence.

Name The name of the AlertSpec.

Threshold The threshold of the condition sentence.

Variable The RRD variable of the condition sentence

Vote_tag Should correspond to a vote tag on a ServerArray if vote to grow or shrink.

Vote_type Vote to grow or shrink a ServerArray when the alert is triggered. Must either escalate or vote. (grow or shrink)

Examples

Create a high network activity alert on my-array:

```
{ "desc": "Create high network rx alert",
  "actor": "rightscale.alerts.Create",
  "options": {
    "array": "my-array",
    "strict_array": true,
    "condition": ">",
    "description": "Alert if amount of network data received is high",
    "duration": 180,
    "escalation_name": "Email Engineering",
    "file": "interface/if_octets-eth0",
    "name": "high network rx activity",
    "threshold": "50000000",
    "variable": "rx"
  }
}
```


Dry Mode

In Dry mode this actor *does* validate that the array exists. If it does not, a `kingpin.actors.rightscale.api.ServerArrayException` is thrown. Once that has been validated, the dry mode execution simply logs the Alert Spec that it would have created.

Example *dry* output:

```
TODO: Fill this in
```

```
class kingpin.actors.rightscale.alerts.Destroy(*args, **kwargs)
```

Destroy existing RightScale Alert Specs

This actor searches RightScale for any Alert Specs that match the name and array that you supplied, then deletes all of them. RightScale lets you have multiple alert specs with the same name, so if this actor finds multiple specs, it will delete them all.

Options

Array The name of the Server or ServerArray to delete the AlertSpec from.

Name The name of the AlertSpec.

Examples

Destroy a high network activity alert on my-array:

```
{ "desc": "Destroy high network rx alert",
  "actor": "rightscale.alerts.Destroy",
  "options": {
    "array": "my-array",
    "name": "high network rx activity",
  }
}
```

Dry Mode

In Dry mode this actor *does* validate that the array exists, and that the AlertSpec exists on that array so that it can be deleted. A `RecoverableActorFailure` error is thrown if it does not exist.

Example *dry* output:

```
14:31:49 INFO Rehearsing... Break a leg!
14:31:49 INFO [DRY: Kingpin] Preparing actors from delete.json
14:31:53 INFO [DRY: Destroy high network rx alert] Found
my-array (/api/server_arrays/329142003) to delete alert spec from
14:31:54 INFO [DRY: Destroy high network rx alert] Would have
deleted the alert spec "high network rx activity" on my-array
```

3.8.4 Server Arrays

kingpin.actors.rightscale.server_array

```
class kingpin.actors.rightscale.server_array.Clone(*args, **kwargs)
```

Clones a RightScale Server Array.

Clones a ServerArray in RightScale and renames it to the newly supplied name. By default, this actor is extremely strict about validating that the source array already exists, and that the dest array does not yet exist. This behavior can be overridden though if your Kingpin script creates the source, or destroys an existing dest ServerArray sometime before this actor executes.

Options

Source The name of the ServerArray to clone

Strict_source Whether or not to fail if the source ServerArray does not exist. (default: True)

Dest The new name for your cloned ServerArray

Strict_dest Whether or not to fail if the destination ServerArray already exists. (default: True)

Examples

Clone my-template-array to my-new-array:

```
{ "desc": "Clone my array",
  "actor": "rightscale.server_array.Clone",
  "options": {
    "source": "my-template-array",
    "dest": "my-new-array"
  }
}
```

Clone an array that was created sometime earlier in the Kingpin JSON, and thus does not exist yet during the dry run:

```
{ "desc": "Clone that array we created earlier",
  "actor": "rightscale.server_array.Clone",
  "options": {
    "source": "my-template-array",
    "strict_source": false,
    "dest": "my-new-array"
  }
}
```

Clone an array into a destination name that was destroyed sometime earlier in the Kingpin JSON:

```
{ "desc": "Clone that array we created earlier",
  "actor": "rightscale.server_array.Clone",
  "options": {
    "source": "my-template-array",
    "dest": "my-new-array",
    "strict_dest": false,
  }
}
```

Dry Mode

In Dry mode this actor *does* validate that the source array exists. If it does not, a `kingpin.actors.rightscale.api.ServerArrayException` is thrown. Once that has been validated, the dry mode execution pretends to copy the array by creating a mocked cloned array resource. This mocked resource is then operated on during the rest of the execution of the actor, guaranteeing that no live resources are modified.

Example *dry* output:

```
[Copy Test (DRY Mode)] Verifying that array "temp" exists
[Copy Test (DRY Mode)] Verifying that array "new" does not exist
[Copy Test (DRY Mode)] Cloning array "temp"
[Copy Test (DRY Mode)] Renaming array "<mocked clone of temp>" to "new"
```

class kingpin.actors.rightscale.server_array.**Update** (*args, **kwargs)

Update ServerArray Settings

Updates an existing ServerArray in RightScale with the supplied parameters. Can update any parameter that is described in the RightScale API docs [here](#):

Parameters are passed into the actor in the form of a dictionary, and are then converted into the RightScale format. See below for examples.

Options

Array (str) The name of the ServerArray to update

Exact (bool) whether or not to search for the exact array name. (default: `true`)

Params (dict) Dictionary of parameters to update

Inputs (dict) Dictionary of next-instance server array inputs to update

Examples

```
{ "desc": "Update my array",
  "actor": "rightscale.server_array.Update",
  "options": {
    "array": "my-new-array",
    "params": {
      "elasticity_params": {
        "bounds": {
          "min_count": 4
        },
        "schedule": [
          { "day": "Sunday", "max_count": 2,
            "min_count": 1, "time": "07:00" },
          { "day": "Sunday", "max_count": 2,
            "min_count": 2, "time": "09:00" }
        ]
      },
      "name": "my-really-new-name"
    }
  }
}
```

```
{ "desc": "Update my array inputs",
  "actor": "rightscale.server_array.Update",
  "options": {
    "array": "my-new-array",
    "inputs": {
      "ELB_NAME": "text:foobar"
    }
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the array, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

This means that the dry mode cannot validate that the supplied inputs will work.

Example *dry* output:

```
[Update Test (DRY Mode)] Verifying that array "new" exists
[Update Test (DRY Mode)] Array "new" not found -- creating a mock.
```

```
[Update Test (DRY Mode)] Would have updated "<mocked array new>" with
params: {'server_array[name]': 'my-really-new-name',
        'server_array[elasticity_params][bounds][min_count]': '4'}
```

class kingpin.actors.rightscale.server_array.**UpdateNextInstance** (*args, **kwargs)
Update the Next Instance parameters for a Server Array

Updates an existing ServerArray in RightScale with the supplied parameters. Can update any parameter that is described in the RightScale [ResourceInstances](#) docs.

Note about the image_href parameter

If you pass in the string default to the image_href key in your params dictionary, we will search and find the default image that your ServerArray's Multi Cloud Image refers to. This helper is useful if you update your ServerArrays to use custom AMIs, and then occasionally want to go back to using a stock AMI. For example, if you boot up your instances occasionally off a stock AMI, customize the host, and then bake that host into a custom AMI.

Parameters are passed into the actor in the form of a dictionary, and are then converted into the RightScale format. See below for examples.

Options

Array (str) The name of the ServerArray to update

Exact (bool) whether or not to search for the exact array name. (default: true)

Params (dict) Dictionary of parameters to update

Examples

```
{ "desc": "Update my array",
  "actor": "rightscale.server_array.UpdateNextInstance",
  "options": {
    "array": "my-new-array",
    "params": {
      "associate_public_ip_address": true,
      "image_href": "/image/href/123",
    }
  }
}
```

```
{ "desc": "Reset the AMI image to the MCI default",
  "actor": "rightscale.server_array.UpdateNextInstance",
  "options": {
    "array": "my-new-array",
    "params": {
      "image_href": "default",
    }
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the array, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

This means that the dry mode cannot validate that the supplied params will work.

Example dry output:

```
[Update my array (DRY Mode)] Verifying that array "new" exists
[Update my array (DRY Mode)] Array "new" not found -- creating a mock.
[Update my array (DRY Mode)] Would have updated "<mocked array new>"
with params: {'server_array[associate_public_ip_address]': true,
              'server_array[image_href]': '/image/href/'}
```

class kingpin.actors.rightscale.server_array.**Terminate**(*args, **kwargs)
 Terminate all instances in a ServerArray

Terminates all instances for a ServerArray in RightScale marking the array disabled.

Options

Array (str) The name of the ServerArray to destroy

Exact (bool) Whether or not to search for the exact array name. (default: true)

Strict (bool) Whether or not to fail if the ServerArray does not exist. (default: true)

Examples

```
{ "desc": "Terminate my array",
  "actor": "rightscale.server_array.Terminate",
  "options": {
    "array": "my-array"
  }
}
```

```
{ "desc": "Terminate many arrays",
  "actor": "rightscale.server_array.Terminate",
  "options": {
    "array": "array-prefix",
    "exact": false,
  }
}
```

Dry Mode

Dry mode still validates that the server array you want to terminate is actually gone. If you want to bypass this check, then set the warn_on_failure flag for the actor.

class kingpin.actors.rightscale.server_array.**Destroy**(*args, **kwargs)
 Destroy a ServerArray in RightScale

Destroys a ServerArray in RightScale by first invoking the Terminate actor, and then deleting the array as soon as all of the running instances have been terminated.

Options

Array (str) The name of the ServerArray to destroy

Exact (bool) Whether or not to search for the exact array name. (default: true)

Strict (bool) Whether or not to fail if the ServerArray does not exist. (default: true)

Examples

```
{ "desc": "Destroy my array",
  "actor": "rightscale.server_array.Destroy",
  "options": {
    "array": "my-array"
  }
}
```

```
{ "desc": "Destroy many arrays",
  "actor": "rightscale.server_array.Destroy",
  "options": {
    "array": "array-prefix",
    "exact": false,
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the `array`, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

Example *dry* output:

```
[Destroy Test (DRY Mode)] Beginning
[Destroy Test (DRY Mode)] Terminating array before destroying it.
[Destroy Test (terminate) (DRY Mode)] Array "my-array" not found --
creating a mock.
[Destroy Test (terminate) (DRY Mode)] Disabling Array "my-array"
[Destroy Test (terminate) (DRY Mode)] Would have terminated all array
"<mocked array my-array>" instances.
[Destroy Test (terminate) (DRY Mode)] Pretending that array <mocked
array my-array> instances are terminated.
[Destroy Test (DRY Mode)] Pretending to destroy array "<mocked array
my-array>"
[Destroy Test (DRY Mode)] Finished successfully. Result: True
```

class kingpin.actors.rightscale.server_array.**Launch** (*args, **kwargs)

Launch instances in a ServerArray

Launches instances in an existing ServerArray and waits until that array has become healthy before returning. *Healthy* means that the array has at least the user-specified `count` or `min_count` number of instances running as defined by the array definition in RightScale.

Options

Array (str) The name of the ServerArray to launch

Count (str, int) Optional number of instance to launch. Defaults to `min_count` of the array.

Enable (bool) Should the autoscaling of the array be enabled? Settings this to `false`, or omitting the parameter will not disable an enabled array.

Exact (bool) Whether or not to search for the exact array name. (default: `true`)

Examples

```
{ "desc": "Enable array and launch it",
  "actor": "rightscale.server_array.Launch",
  "options": {
    "array": "my-array",
    "enable": true
  }
}
```

```
{ "desc": "Enable arrays starting with my-array and launch them",
  "actor": "rightscale.server_array.Launch",
  "options": {
```

```

    "array": "my-array",
    "enable": true,
    "exact": false
  }
}

```

```

{ "desc": "Enable array and launch 1 instance",
  "actor": "rightscale.server_array.Launch",
  "options": {
    "array": "my-array",
    "count": 1
  }
}

```

Dry Mode

In Dry mode this actor *does* search for the array, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

Example *dry* output:

```

[Launch Array Test #0 (DRY Mode)] Verifying that array "my-array" exists
[Launch Array Test #0 (DRY Mode)] Array "my-array" not found -- creating
a mock.
[Launch Array Test #0 (DRY Mode)] Enabling Array "my-array"
[Launch Array Test #0 (DRY Mode)] Launching Array "my-array" instances
[Launch Array Test #0 (DRY Mode)] Would have launched instances of array
<MagicMock name='my-array.self.show().soul.__getitem__()'
id='4420453200'>
[Launch Array Test #0 (DRY Mode)] Pretending that array <MagicMock
name='my-array.self.show().soul.__getitem__()' id='4420453200'>
instances are launched.

```

class kingpin.actors.rightscale.server_array.**Execute** (*args, **kwargs)

Executes a RightScale script/recipe on a ServerArray

Executes a RightScript or Recipe on a set of hosts in a ServerArray in RightScale using individual calls to the live running instances. These can be found in your RightScale account under *Design -> RightScript* or *Design -> Cookbooks*

The RightScale API offers a *multi_run_executable* method that can be used to run a single script on all servers in an array – but unfortunately this API method provides no way to monitor the progress of the individual jobs on the hosts. Furthermore, the method often executes on recently terminated or terminating hosts, which throws false-negative error results.

Our actor explicitly retrieves a list of the *operational* hosts in an array and kicks off individual execution tasks for every host. It then tracks the execution of those tasks from start to finish and returns the results.

Options

Array (str) The name of the ServerArray to operate on

Script (str) The name of the RightScript or Recipe to execute

Expected_runtime (str, int) Expected number of seconds to execute. (default: 5)

Concurrency Max number of concurrent executions. This will fire off N executions in parallel, and continue with the remained as soon as the first execution is done. This is faster than creating N

Sync executions. **Note:** When applied to multiple (M) arrays cumulative concurrency accross all arrays will remain at N. It will not be M x N.

Inputs (dict) Dictionary of Key/Value pairs to use as inputs for the script

Exact (str) Boolean whether or not to search for the exact array name. (default: `true`)

Examples

```
{ "desc": "Execute script on my-array",
  "actor": "rightscale.server_array.Execute",
  "options": {
    "array": "my-array",
    "script": "connect to elb",
    "expected_runtime": 3,
    "inputs": {
      "ELB_NAME": "text:my-elb"
    }
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the `array`, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

Example *dry* output:

```
[Destroy Test (DRY Mode)] Verifying that array "my-array" exists
[Execute Test (DRY Mode)]
  kingpin.actors.rightscale.server_array.Execute Initialized
[Execute Test (DRY Mode)] Beginning execution
[Execute Test (DRY Mode)] Verifying that array "my-array" exists
[Execute Test (DRY Mode)] Would have executed "Connect instance to ELB"
  with inputs '{"inputs[ELB_NAME]': 'text:my-elb'}' on "my-array".
[Execute Test (DRY Mode)] Returning result: True
```

3.8.5 Multi Cloud Images

`kingpin.actors.rightscale.mci`

class `kingpin.actors.rightscale.mci.Create(*args, **kwargs)`

Creates a RightScale Multi Cloud Image.

Options match the documentation in RightScale: <http://reference.rightscale.com/api1.5/resources/ResourceMultiCloudImages.htm>

Options

Name The name of the MCI to be created.

Description The description of the MCI to be created. (*optional*)

Images A list of dicts that each describe a single cloud and the image in that cloud to launch. See below for details.

Image Definitions

Each cloud image definition is a dictionary that takes a few keys.

Cloud The name of the cloud as found in RightScale. We use the cloud 'Name' which can be found in your Settings -> Account Settings -> Clouds -> insert_cloud_here page. For example AWS us-west-2.

Image The cloud-specific Image UID. For example ami-a1234abc.

Instance_type The default instance type to launch when this AMI is launched. For example, m1.small. (*optional*)

User_data The custom user data to pass to the instance on-bootup. (*optional*)

Examples

```
{ "actor": "rightscale.mci.Create",
  "desc": "Create an MCI",
  "options": {
    "name": "Ubuntu i386 14.04",
    "description": "this is our test mci",
    "images": [
      {
        "cloud": "EC2 us-west-2",
        "image": "ami-e29774d1",
        "instance_type": "m1.small",
        "user_data": "cd /bin/bash"
      },
      {
        "cloud": "EC2 us-west-1",
        "image": "ami-b58142f1",
        "instance_type": "m1.small",
        "user_data": "cd /bin/bash"
      }
    ]
  }
}
```

class kingpin.actors.rightscale.mci.**Destroy** (*args, **kwargs)

Deletes a RightScale MCI.

Options match the documentation in RightScale: <http://reference.rightscale.com/api1.5/resources/ResourceMultiCloudImages.htm>

Options

Name The name of the multi cloud image to be deleted.

Examples

```
{ "actor": "rightscale.mci.Destroy",
  "desc": "Create an MCI",
  "options": {
    "name": "Ubuntu i386 14.04",
  }
}
```

3.9 Rollbar

3.9.1 Deploy

class kingpin.actors.rollbar.**Deploy** (*args, **kwargs)

Posts a Deploy message to Rollbar.

https://rollbar.com/docs/deployments_other/

API Token

You must use an API token created in your *Project Access Tokens* account settings section. This token should have *post_server_item* permissions for the actual deploy, and *read* permissions for the Dry run.

Options

Environment The environment to deploy to

Revision The deployment revision

Local_username The user who initiated the deploy

Rollbar_username (*Optional*) The Rollbar Username to assign the deploy to

Comment (*Optional*) Comment describing the deploy

Examples

```
{ "actor": "rollbar.Deploy",
  "desc": "update rollbar deploy",
  "options": {
    "environment": "Prod",
    "revision": "%DEPLOY%",
    "local_username": "Kingpin",
    "rollbar_username": "Kingpin",
    "comment": "some comment %DEPLOY%"
  }
}
```

Dry Mode

Accesses the Rollbar API and validates that the token can access your project.

3.10 Slack

3.10.1 Message

class kingpin.actors.slack.**Message** (*args, **kwargs)

Sends a message to a channel in Slack.

Options

Channel The string-name of the channel to send a message to

Message String of the message to send

Examples

```
{ "desc": "Let the Engineers know things are happening",
  "actor": "slack.Message",
  "options": {
    "channel": "#operations",
    "message": "Beginning Deploy: %VER%"
  }
}
```

Dry Mode

Fully supported – does not actually send messages to a room, but validates that the API credentials would have access to send the message using the Slack `auth.test` API method.

4.1 URLLIB3 Warnings Disabled

Recently urllib3 library has started issuing `InsecurePlatformWarning`. We suppress urllib3 warnings to limit log output to Kingping's own.

Development

5.1 Setting up your Environment

5.1.1 Create your VirtualEnvironment

```
$ virtualenv .venv --no-site-packages
New python executable in .venv/bin/python
Installing setuptools, pip...done.
$ source .venv/bin/activate
```

5.1.2 Check out the code

```
(.venv) $ git clone git@github.com:Nextdoor/kingpin
Cloning into 'kingpin'...
Warning: Permanently added 'github.com,192.30.252.128' (RSA) to the list of known hosts.
remote: Counting objects: 1831, done.
remote: irangedCompressing objects: 100% (17/17), done.
remote: Total 1831 (delta 7), reused 0 (delta 0)
Receiving objects: 100% (1831/1831), 287.68 KiB, done.
Resolving deltas: 100% (1333/1333), done.
```

5.1.3 Install the test-specific dependencies

```
(.venv) $ pip install -r kingpin/requirements.test.txt
...
(.venv) $ cd kingpin
(.venv) $ python setup.py test
...
```

5.2 Testing

5.2.1 Unit Tests

The code is 100% unit test coverage complete, and no pull-requests will be accepted that do not maintain this level of coverage. That said, it's possible (*likely*) that we have not covered every possible scenario in our unit tests that could cause failures. We will strive to fill out every reasonable failure scenario.

5.2.2 Integration Tests

Because it's hard to predict cloud failures, we provide integration tests for most of our modules. These integration tests actually go off and execute real operations in your accounts, and rely on particular environments being setup in order to run. These tests are great to run though to validate that your credentials are all correct.

Specific integration test notes are below, describing what is required to run each set of tests.

Executing the tests

```

HIPCHAT_TOKEN=<xxx> RIGHTSCALE_TOKEN=<xxx> make integration

PYFLAKES_NODOCTEST=True python setup.py integration pep8 pyflakes
running integration
integration_01_clone_dry (integration_server_array.IntegrationServerArray) ... ok
integration_02a_clone (integration_server_array.IntegrationServerArray) ... ok
integration_02b_clone_with_duplicate_array (integration_server_array.IntegrationServerArray) ... ok
integration_03a_update_params (integration_server_array.IntegrationServerArray) ... ok
integration_03b_update_with_invalid_params (integration_server_array.IntegrationServerArray) ... ok
integration_04_launch (integration_server_array.IntegrationServerArray) ... ok
integration_05_destroy (integration_server_array.IntegrationServerArray) ... ok
integration_test_execute_real (integration_hipchat.IntegrationHipchatMessage) ... ok
integration_test_execute_with_invalid_creds (integration_hipchat.IntegrationHipchatMessage) ... ok
integration_test_init_without_environment_creds (integration_hipchat.IntegrationHipchatMessage) ... ok

```

Name	Stmts	Miss	Cover	Missing
-----	-----	-----	-----	-----
kingpin	0	0	100%	
kingpin.actors	0	0	100%	
kingpin.actors.base	62	5	92%	90, 95, 146, 215-216
kingpin.actors.exceptions	4	0	100%	
kingpin.actors.hipchat	58	5	91%	59, 111-118
kingpin.actors.misc	17	5	71%	47-49, 57-62
kingpin.actors.rightscale	0	0	100%	
kingpin.actors.rightscale.api	137	46	66%	153-164, 251-258, 343-346, 381-382, 4
kingpin.actors.rightscale.base	31	3	90%	36, 49, 79
kingpin.actors.rightscale.server_array	195	49	75%	59-62, 68-72, 79, 174, 190-196, 213-2
kingpin.utils	67	30	55%	57-69, 78, 93-120, 192-202
-----	-----	-----	-----	-----
TOTAL	571	143	75%	
-----	-----	-----	-----	-----

```

Ran 10 tests in 880.274s

OK
running pep8
running pyflakes

```

kingpin.actor.rightscale.server_array

These tests clone a ServerArray, modify it, launch it, and destroy it. They rely on an existing ServerArray template being available and launchable in your environment. For simple testing, I recommend just using a standard RightScale ServerTemplate.

Required RightScale Resources

- **ServerArray:** *kingpin-integration-testing* Any ServerArray that launches a server in your environment.
- **RightScript:** *kingpin-integration-testing-script* Should be a script that sleeps for a specified amount of time.
Requires “SLEEP” input

5.3 Class/Object Architecture

```
kingpin.rb
|
+-- deployment.Deployer
|   | Executes a deployment based on the supplied DSL.
|   |
+-- actors.rightscale
|   | RightScale Cloud Management Actor
|   |
|   +-- server_array
|       +-- Clone
|       +-- Destroy
|       +-- Execute
|       +-- Launch
|       +-- Update
|
+-- actors.aws
|   | Amazon Web Services Actor
|   |
|   +-- elb
|       +-- WaitUntilHealthy
|       |
|   +-- sqs
|       +-- Create
|       +-- Delete
|       +-- WaitUntilEmpty
|
+-- actors.email
|   | Email Actor
|   |
+-- actors.hipchat
|   | Hipchat Actor
|   |
|   +-- Message
|   |
+-- actors.librato
|   | Librato Metric Actor
|   |
+-- Annotation
```

5.4 Setup

```
# Create a dedicated Python virtual environment and source it
virtualenv --no-site-packages .venv
unset PYTHONPATH
source .venv/bin/activate

# Install the dependencies
make build

# Run the tests
make test
```

5.5 Actor Design

Kingpin Actors are self-contained python classes that execute operations asynchronously. Actors should follow a consistent structure (described below) and be written to be as fault tolerant as possible.

5.5.1 Hello World Actor Example

This is the basic structure for an actor class.

```
import os

from tornado import gen

from kingpin.actors import base
from kingpin.actors import exceptions

# All actors must have an __author__ tag. This is used actively
# by the Kingpin code, do not forget this!
__author__ = 'Billy Joe Armstrong <american_idiot@broadway.com>'

# Perhaps you need an API token?
TOKEN = os.getenv('HELLO_WORLD_TOKEN', None)

class HelloWorld(base.BaseActor):
    # Create an all_options dictionary that contains all of
    # the required and optional options that can be passed into
    # this actor.
    all_options = {
        'name': (str, None, 'Your name'),
        'world': (str, None, 'World we\'re saying hello to!'),
    }

    # Optionally, if you need to do any instantiation-level, non-blocking
    # validation checks (for example, looking for an API token) you can do
    # them in the __init__. Do *not* put blocking code in here.
    def __init__(self, *args, **kwargs):
        super(HelloWorld, self).__init__(*args, **kwargs)
        if not TOKEN:
            raise exceptions.InvalidCredentials(
                'Missing the "HELLO_WORLD_TOKEN" environment variable.')

        # Initialize our hello world sender object. This is non-blocking.
        self._hello_world = my.HelloWorldSender(token=TOKEN)

    # Its nice to wrap some of your logic into separate methods. This
    # method handles sending the message, or pretends to send the
    # message if we're in a dry run.
    @gen.coroutine
    def _send_message(self, name, world):
        # Attempt to log into the API to sanity check our credentials
        try:
            yield self._hello_world.login()
        except Shoplifter:
            msg = 'Could not log into the world!'
            raise exceptions.UnrecoverableActorFailure(msg)

    # Make sure to support DRY mode all the time!
```

```

if self._dry:
    self.log.info('Would have said Hi to %s' % world)
    raise gen.Return()

# Finally, send the message!
try:
    res = yield self._hello_world.send(
        from=name, to=world)
except WalkingAlone as e:
    # Lets say that this error is completely un-handleable exception,
    # there's no one to say hello to!
    self.log.critical('Some extra information about this error...')

    # Now, raise an exception that is will stop execution of Kingpin,
    # regardless of the warn_on_failure setting.
    raise exceptions.UnrecoverableActorException('Oh my: %s' % e)

# Return the value back to the execute method
raise gen.Return(res)

# The meat of the work happens in the _execute() method. This method
# is called by the BaseActor.execute() method. Your method must be
# wrapped in a gen.Coroutine wrapper. Note, the _execute() method takes
# no arguments, all arguments for the actor were passed in to the
# __init__() method.
@gen.coroutine
def _execute(self):
    self.log.debug('Warming up the HelloWorld Actor')

    # Fire off an async request to a our private method for sending
    # hello world messages. Get the response and evaluate
    res = yield self._send_message(
        self.option('name'), self.option('world'))

    # Got a response. Did our message really go through though?
    if not res:
        # The world refuses to hear our message... A shame, really, but
        # not entirely critical.
        self.log.error('We failed to get our message out ... just '
            'letting you know!')
        raise exceptions.RecoverableActorFailure(
            'A shame, but I suppose they can listen to what they want')

    # We've been heard!
    self.log.info('%s people have heard our message!' % res)

    # Indicate to Tornado that we're done with our execution.
    raise gen.Return()

```

5.5.2 Required Options

The following options are baked into our *BaseActor* model and must be supported by any actor that subclasses it. They are fundamentally critical to the behavior of Kingpin, and should not be bypassed or ignored.

`desc`

A string describing the stage or action that's occurring. Meant to be human readable and useful for logging. You do not need to do anything intentionally to support this option (it's handled in `kingpin.actors.base.BaseActor`).

`dry`

All Actors *must* support a `dry` run flag. The codepath that's executed when `_execute()` is yielded should be as wet as possible without actually making any changes. For example, if you have an actor that checks the state of an Amazon ELB (*hint see `aws.elb.WaitUntilHealthy`*), you would want the actor to actually search Amazon's API for the ELB, actually check the number of instances that are healthy in the ELB, and then fake a return value so that the rest of the script can be tested.

`options`

Your actor can take in custom options (ELB name, Route53 DNS entry name, etc) through a dictionary named `options` that's passed in to every actor and stored as `self._options`. The contents of this dictionary are entirely up to you.

`warn_on_failure (optional)`

If the user sets `warn_on_failure=True`, any raised exceptions that subclass `kingpin.actors.exceptions.RecoverableActorFailure` will be swallowed up and warned about, but will not cause the execution of the kingpin script to end.

Exceptions that subclass `kingpin.actors.exceptions.UnrecoverableActorFailure` (or uncaught third party exceptions) will cause the actor to fail and the script to be aborted **no matter what!**

5.5.3 Required Methods

`_execute()` method

Your actor can execute any code you would like in the `_execute()` method. This method should make sure that it's a tornado-style generator (thus, can be yielded), and that it never calls any blocking operations.

Actors must *not*:

- Call a blocking operation ever
- Call an async operation from inside the `init()` method
- Bypass normal logging methods
- `return` a result (should `raise gen.Return(...)`)

Actors must:

- Subclass `kingpin.actors.base.BaseActor`
- Include `__author__` attribute that's a single *string* with the owners listed in it.
- Implement a `*_execute()*` method
- Handle as many possible exceptions of third-party libraries as possible
- Return `None` when the actor has succeeded.

Actors can:

- Raise *kingpin.actors.exceptions.UnrecoverableActorFailure*. This is considered an unrecoverable exception and no Kingpin will not execute any further actors when this happens.
- Raise *kingpin.actors.exceptions.RecoverableActorFailure*. This is considered an error in execution, but is either expected or at least cleanly handled in the code. It allows the user to specify `warn_on_failure=True`, where they can then continue on in the script even if an actor fails.

****Super simple example Actor `_execute()` method****

```
@gen.coroutine
def _execute(self):
    self.log.info('Making that web call')
    res = yield self._post_web_call(URL)
    raise gen.Return(res)
```

5.5.4 Helper Methods/Objects

self.log

For consistency in logging, a custom Logger object is instantiated for every Actor. This logging object ensures that prefixes such as the `desc` of an Actor are included in the log messages. Usage examples:

```
self.log.error('Hey, something failed')
self.log.info('I am doing work')
self.log.warning('I do not think that should have happened')
```

5.5.5 self.option

Accessing options passed to the actor from the JSON file should be done via `self.option()` method. Accessing `self._options` parameter is not recommended, and the edge cases should be handled via the `all_options` class variable.

5.5.6 Exception Handling

5.6 Simple API Access Objects

Most of the APIs out there leverage basic REST with JSON or XML as the data encoding method. Since these APIs behave similarly, we have created a simple API access object that can be extended for creating actors quickly. The object is called a `RestConsumer` and is in the `kingpin.actors.support.api` package. This `RestConsumer` can be subclassed and filled in with a `dict` that describes the API in detail.

5.6.1 HTTPBin Actor with the RestConsumer

```
HTTPBIN = {
    'path': '/',
    'http_methods': {'get': {}},
    'attrs': {
        'get': {
            'path': '/get',
            'http_methods': {'get': {}},
```

```
    },
    'post': {
        'path': '/post',
        'http_methods': {'post': {}},
    },
    'put': {
        'path': '/put',
        'http_methods': {'put': {}},
    },
    'delete': {
        'path': '/delete',
        'http_methods': {'delete': {}},
    },
}
}

class HTTPBinRestClient(api.RestConsumer):

    _CONFIG = HTTPBIN
    _ENDPOINT = 'http://httpbin.org'

class HTTPBinGetThenPost(base.BaseActor):
    def __init__(self, *args, **kwargs):
        super(HTTPBinGetThenPost, self).__init__(*args, **kwargs)
        self._api = HTTPBinRestClient()

    @gen.coroutine
    def _execute(self):
        yield self._api.get().http_get()

        if self._dry:
            raise gen.Return()

        yield self._api.post().http_post(foo='bar')

        raise gen.Return()
```

5.6.2 Exception Handling in HTTP Requests

The `RestClient.fetch()` method has been wrapped in a `retry` decorator that allows you to define different behaviors based on the exceptions returned from the `fetch` method. For example, you may want to handle an `HTTPError` exception with a 401 error code differently than a 503 error code.

You can customize the exception handling by subclassing the `RestClient`:

```
class MyRestClient(api.RestClient):
    _EXCEPTIONS = {
        httpclient.HTTPError: {
            '401': my.CustomException(),
            '403': exceptions.InvalidCredentials,
            '500': my.UnretryableError(),
            '502': exceptions.InvalidOptions,

            # This acts as a catch-all
            '': exceptions.RecoverableActorFailure,
```

```
}  
}
```

Full Module Docs

6.1 kingpin.actors.aws.base

The AWS Actors allow you to interact with the resources (such as SQS and ELB) inside your Amazon AWS account. These actors all support dry runs properly, but each actor has its own caveats with `dry=True`. Please read the instructions below for using each actor.

Required Environment Variables

AWS_ACCESS_KEY_ID Your AWS access key

AWS_SECRET_ACCESS_KEY Your AWS secret

exception `kingpin.actors.aws.base.ELBNotFound`

Raised when an ELB is not found

exception `kingpin.actors.aws.base.InvalidMetaData`

Raised when fetching AWS metadata.

6.2 kingpin.actors.aws.cloudformation

exception `kingpin.actors.aws.cloudformation.CloudFormationError`

Raised on any generic CloudFormation error.

exception `kingpin.actors.aws.cloudformation.InvalidTemplate`

An invalid CloudFormation template was supplied.

exception `kingpin.actors.aws.cloudformation.StackAlreadyExists`

The requested CloudFormation stack already exists.

exception `kingpin.actors.aws.cloudformation.StackNotFound`

The requested CloudFormation stack does not exist.

class `kingpin.actors.aws.cloudformation.CloudFormationBaseActor(*args, **kwargs)`

Base Actor for CloudFormation tasks

class `kingpin.actors.aws.cloudformation.Create(*args, **kwargs)`

Creates a CloudFormation stack.

Creates a CloudFormation stack from scratch and waits until the stack is fully built before exiting the actor.

Options

Capabilities A list of CF capabilities to add to the stack.

Disable_rollback Set to True to disable rollback of the stack if creation failed.

Name The name of the queue to create

Parameters A dictionary of key/value pairs used to fill in the parameters for the CloudFormation template.

Region AWS region (or zone) string, like 'us-west-2'

Template String of path to CloudFormation template. Can either be in the form of a local file path (ie, /my_template.json) or a URI (ie https://my_site.com/cf.json).

Timeout_in_minutes The amount of time that can pass before the stack status becomes CREATE_FAILED.

Examples

```
{ "desc": "Create production backend stack",
  "actor": "aws.cloudformation.Create",
  "options": {
    "capabilities": [ "CAPABILITY_IAM" ],
    "disable_rollback": true,
    "name": "%CF_NAME%",
    "parameters": {
      "test_param": "%TEST_PARAM_NAME%",
    },
    "region": "us-west-1",
    "template": "/examples/cloudformation_test.json",
    "timeout_in_minutes": 45,
  }
}
```

Dry Mode

Validates the template, verifies that an existing stack with that name does not exist. Does not create the stack.

class kingpin.actors.aws.cloudformation.**Delete**(*args, **kwargs)

Deletes a CloudFormation stack

Options

Name The name of the queue to create

Region AWS region (or zone) string, like 'us-west-2'

Examples

```
{ "desc": "Create production backend stack",
  "actor": "aws.cloudformation.Create",
  "options" {
    "region": "us-west-1",
    "name": "%CF_NAME%",
  }
}
```

Dry Mode

Validates that the CF stack exists, but does not delete it.

6.3 kingpin.actors.aws.elb

exception kingpin.actors.aws.elb.CertNotFound

Raised when an ELB is not found

kingpin.actors.aws.elb.p2f(*string*)

Convert percentage string into float.

Converts string like '78.9%' into 0.789

class kingpin.actors.aws.elb.ELBBaseActor(*args, **kwargs)

Base class for ELB actors.

class kingpin.actors.aws.elb.WaitUntilHealthy(*args, **kwargs)

Wait indefinitely until a specified ELB is considered "healthy".

This actor will loop infinitely until a healthy threshold of the ELB is met. The threshold can be reached when the `count` as specified in the options is less than or equal to the number of InService instances in the ELB.

Another situation is for `count` to be a string specifying a percentage (see examples). In this case the percent of InService instances has to be greater than the `count` percentage.

Options

Name The name of the ELB to operate on

Count Number, or percentage of InService instance to consider this ELB healthy

Region AWS region (or zone) name, such as us-east-1 or us-west-2

Examples

```
{ "actor": "aws.elb.WaitUntilHealthy",
  "desc": "Wait until production-frontend has 16 hosts",
  "options": {
    "name": "production-frontend",
    "count": 16,
    "region": "us-west-2"
  }
}
```

```
{ "actor": "aws.elb.WaitUntilHealthy",
  "desc": "Wait until production-frontend has 85% of hosts in-service",
  "options": {
    "name": "production-frontend",
    "count": "85%",
    "region": "us-west-2"
  }
}
```

Dry Mode

This actor performs the finding of the ELB as well as calculating its health at all times. The only difference in dry mode is that it will not re-count the instances if the ELB is not healthy. A log message will be printed indicating that the run is dry, and the actor will exit with success.

class kingpin.actors.aws.elb.SetCert(*args, **kwargs)

Find a server cert in IAM and use it for a specified ELB.

Options

Region (str) AWS region (or zone) name, like us-west-2

Name (str) Name of the ELB

Cert_name (str) Unique IAM certificate name, or ARN

Port (int) Port associated with the cert. (default: 443)

Example

```
{ "actor": "aws.elb.SetCert",
  "desc": "Run SetCert",
  "options": {
    "cert_name": "new-cert",
    "name": "some-elb",
    "region": "us-west-2"
  }
}
```

Dry run

Will check that ELB and Cert names are existent, and will also check that the credentials provided for AWS have access to the new cert for ssl.

class kingpin.actors.aws.elb.**RegisterInstance** (*args, **kwargs)
Add an EC2 instance to a load balancer.

Options

Elb (str) Name of the ELB

Instances (str, list) Instance id, or list of ids. Default “self” id.

Region (str) AWS region (or zone) name, like us-west-2

Enable_zones (bool) add all available AZ to the elb. Default: True

Example

```
{ "actor": "aws.elb.RegisterInstance",
  "desc": "Run RegisterInstance",
  "options": {
    "elb": "prod-loadbalancer",
    "instances": "i-123456",
    "region": "us-east-1",
  }
}
```

Dry run

Will find the specified ELB, but not take any actions regarding instances.

class kingpin.actors.aws.elb.**DeregisterInstance** (*args, **kwargs)
Remove EC2 instance(s) from an ELB.

Options

Elb (str) Name of the ELB

Instances (str, list) Instance id, or list of ids

Region (str) AWS region (or zone) name, like us-west-2

Example

```
{ "actor": "aws.elb.DeregisterInstance",
  "desc": "Run DeregisterInstance",
  "options": {
```

```

    "elb": "fill-in",
    "instances": "fill-in",
    "region": "fill-in"
  }
}

```

Dry run

Will find the ELB but not take any actions regarding the instances.

6.4 kingpin.actors.aws.iam

class kingpin.actors.aws.iam.**IAMBaseActor** (*args, **kwargs)
Base class for IAM actors.

class kingpin.actors.aws.iam.**UploadCert** (*args, **kwargs)
Uploads a new SSL Cert to AWS IAM.

Options

Private_key_path (str) Path to the private key.

Path (str) The AWS “path” for the server certificate. Default: “/”

Public_key_path (str) Path to the public key certificate.

Name (str) The name for the server certificate.

Cert_chain_path (str) Path to the certificate chain. Optional.

Example

```

{ "actor": "aws.iam.UploadCert",
  "desc": "Upload a new cert",
  "options": {
    "name": "new-cert",
    "private_key_path": "/cert.key",
    "public_key_path": "/cert.pem",
    "cert_chain_path": "/cert-chain.pem"
  }
}

```

Dry run

Checks that the passed file paths are valid. In the future will also validate that the files are of correct format and content.

class kingpin.actors.aws.iam.**DeleteCert** (*args, **kwargs)
Delete an existing SSL Cert in AWS IAM.

Options

Name (str) The name for the server certificate.

Example

```

{ "actor": "aws.iam.DeleteCert",
  "desc": "Run DeleteCert",
  "options": {
    "name": "fill-in"
  }
}

```

```
}  
}
```

Dry run

Will find the cert by name or raise an exception if it's not found.

6.5 kingpin.actors.aws.settings

Common settings used by many of the kingpin.actors.aws modules.

kingpin.actors.aws.settings.is_retriable_exception(exception)

Return true if this AWS exception is transient and should be retried.

Example:

```
>>> @retry(retry_on_exception=is_retriable_exception)
```

6.6 kingpin.actors.aws.sqs

exception kingpin.actors.aws.sqs.QueueNotFound

Raised by SQS Actor when a needed queue is not found.

exception kingpin.actors.aws.sqs.QueueDeletionFailed

Raised if Boto fails to delete an SQS queue.

http://boto.readthedocs.org/en/latest/ref/sqs.html#boto.sqs.connection.SQSConnection.delete_queue

class kingpin.actors.aws.sqs.Create(*args, **kwargs)

Creates a new SQS queue with the specified name

Options

Name (str) The name of the queue to create

Region (str) AWS region (or zone) string, like 'us-west-2'

Examples

```
{ "actor": "aws.sqs.Create",  
  "desc": "Create queue named async-tasks",  
  "options": {  
    "name": "async-tasks",  
    "region": "us-east-1",  
  }  
}
```

Dry Mode

Will not create any queue, or even contact SQS. Will create a mock.Mock object and exit with success.

class kingpin.actors.aws.sqs.Delete(*args, **kwargs)

Deletes the SQS queues

Note: even if it's not empty

Options

Name (str) The name of the queue to destroy

Region (str) AWS region (or zone) string, like 'us-west-2'

Idempotent (bool) Will not raise errors if no matching queues are found. (default: False)

Examples

```
{ "actor": "aws.sqs.Delete",
  "desc": "Delete queue async-tasks",
  "options": {
    "name": "async-tasks",
    "region": "us-east-1"
  }
}
```

```
{ "actor": "aws.sqs.Delete",
  "desc": "Delete queues with 1234 in the name",
  "options": {
    "name": "1234",
    "region": "us-east-1"
  }
}
```

Dry Mode

Will find the specified queue, but will have a noop regarding its deletion. Dry mode will fail if no queues are found, and idempotent flag is set to False.

class kingpin.actors.aws.sqs.**WaitUntilEmpty** (*args, **kwargs)

Wait indefinitely until for SQS queues to become empty

This actor will loop infinitely as long as the count of messages in at least one queue is greater than zero. SQS does not guarantee exact count, so this can return a stale value if the number of messages in the queue changes rapidly.

Options

Name (str) The name or regex pattern of the queues to operate on

Region (str) AWS region (or zone) string, like 'us-west-2'

Required (bool) Fail if no matching queues are found. (default: False)

Examples

```
{ "actor": "aws.sqs.WaitUntilEmpty",
  "desc": "Wait until release-0025a* queues are empty",
  "options": {
    "name": "release-0025a",
    "region": "us-east-1",
    "required": true
  }
}
```

Dry Mode

This actor performs the finding of the queue, but will pretend that the count is 0 and return success. Will fail even in dry mode if `required` option is set to True and no queues with the name pattern are found.

6.7 kingpin.actors.base

Base Actor object class

An Actor object is a class that executes a single logical action on a resource as part of your deployment structure. For example, you may have an Actor that launches a server array in RightScale, or you may have one that sends an email.

Each Actor object should do one thing, and one thing only. Its responsible for being able to execute the operation in both ‘dry’ and ‘non-dry’ modes.

The behavior for ‘dry’ mode can contain real API calls, but should not make any live changes. It is up to the developer of the Actor to define what ‘dry’ mode looks like for that particular action.

class kingpin.actors.base.**BaseActor** (*desc, options, dry=False, warn_on_failure=False, condition=True, init_context={}, timeout=None*)

Abstract base class for Actor objects.

option (*name*)

Return the value for a given Actor option.

readfile (*path*)

Return file contents as a string.

Raises: InvalidOptions if file is not found, or readable.

timer (*f*)

Coroutine-compatible function timer.

Records statistics about how long a given function took, and logs them out in debug statements. Used primarily for tracking Actor execute() methods, but can be used elsewhere as well.

Example usage:

```
>>> @gen.coroutine
... @timer()
... def execute(self):
...     raise gen.Return()
```

timeout (**args, **kwargs*)

Wraps a Coroutine method in a timeout.

Used to wrap the self.execute() method in a timeout that will raise an ActorTimedOut exception if an actor takes too long to execute.

Note, Tornado 4+ does not allow you to actually kill a task on the IOLoop. This means that all we are doing here is notifying the caller (through the raised exception) that a problem has happened.

Fairly simple Actors should actually ‘stop executing’ when this exception is raised. Complex actors with very unique behaviors though (like the rightscale.server_array.Execute actor) have the ability to continue to execute in the background until the Kingpin application quits. It is not the job of this method to try to kill these actors, but just to let the user know that a failure has happened.

class kingpin.actors.base.**HTTPBaseActor** (*desc, options, dry=False, warn_on_failure=False, condition=True, init_context={}, timeout=None*)

Abstract base class for an HTTP-client based Actor object.

This class provides common methods for getting access to asynchronous HTTP clients, wrapping the executions in appropriate try/except blocks, timeouts, etc.

If you’re writing an Actor that uses a remote REST API, this is the base class you should subclass from.

6.8 kingpin.actors.exceptions

All common Actor exceptions

exception `kingpin.actors.exceptions.ActorException`

Base Kingpin Actor Exception

exception `kingpin.actors.exceptions.RecoverableActorFailure`

Base exception that allows script executions to continue on failure.

This exception class is used to throw an error when an Actor fails, but it was an expected and/or acceptable failure.

This should be used for exceptions that are somewhat normal ... for example, trying to delete a ServerArray thats already gone.

exception `kingpin.actors.exceptions.UnrecoverableActorFailure`

Base exception for unrecoverable failures.

This exception class should be used for critical failures that should always stop a set of Kingpin actors in-place, regardless of the actors `warn_on_failure` setting.

Examples would be when credentials are incorrect, or an unexpected exception is caught and there is no known recovery point.

exception `kingpin.actors.exceptions.ActorTimedOut`

Raised when an Actor takes too long to execute

exception `kingpin.actors.exceptions.InvalidActor`

Raised when an invalid Actor name was supplied

exception `kingpin.actors.exceptions.InvalidOptions`

Invalid option arguments passed into the Actor object.

This can be used both for the actual options dict passed into the actor, as well as if a the wrong options were used when connecting to a remote API.

exception `kingpin.actors.exceptions.InvalidCredentials`

Invalid or missing credentials required for Actor object.

exception `kingpin.actors.exceptions.UnparseableResponseFromEndpoint`

Invalid response returned from a remote REST endpoint.

exception `kingpin.actors.exceptions.BadRequest`

An action failed due to a HTTP 400 error likely due to bad input.

6.9 kingpin.actors.group

Group a series of other `BaseActor` into either synchronous or asynchronous stages.

class `kingpin.actors.group.BaseGroupActor(*args, **kwargs)`

Group together a series of other `kingpin.actors.base.BaseActor` objects

Acts [<list of `kingpin.actors.base.BaseActor` objects to execute>]

class `kingpin.actors.group.Sync(*args, **kwargs)`

Execute a series of `kingpin.actors.base.BaseActor` synchronously.

Groups together a series of Actors and executes them synchronously in the order that they were defined.

Options

Acts An array of individual Actor definitions.

Contexts A list of dictionaries with *contextual tokens* to pass into the actors at instantiation time. If the list has more than one element, then every actor defined in `acts` will be instantiated once for each item in the `contexts` list.

Timeouts

Timeouts are disabled specifically in this actor. The sub-actors can still raise their own `kingpin.actors.exceptions.ActorTimedOut` exceptions, but since the group actors run an arbitrary number of sub actors, we have chosen to not have this actor specifically raise its own `kingpin.actors.exceptions.ActorTimedOut` exception unless the user sets the `timeout` setting.

Examples

Creates two arrays ... but sleeps 60 seconds between the two, then does not sleep at all after the last one:

```
{ "desc": "Clone, then sleep ... then clone, then sleep shorter...",
  "actor": "group.Sync",
  "options": {
    "contexts": [
      { "ARRAY": "First", "SLEEP": "60", },
      { "ARRAY": "Second", "SLEEP": "0", }
    ],
    "acts": [
      { "desc": "do something",
        "actor": "server_array.Clone",
        "options": {
          "source": "template",
          "dest": "{ARRAY}"
        }
      },
      { "desc": "sleep",
        "actor": "misc.Sleep",
        "options": {
          "sleep": "{SLEEP}"
        }
      }
    ]
  }
}
```

Dry Mode

Passes on the Dry mode setting to the acts that are called. Does **not** stop execution when one of the acts fails. Instead Group actor will finish all acts with warnings, and raise an error at the end of execution.

This provides the user with an insight to all the errors that are possible to encounter, rather than abort and quit on the first one.

Failure

In the event that an act fails, this actor will return the failure immediately. Because the acts are executed in-order of definition, the failure will prevent any further acts from executing.

The behavior is different in the dry run (read above.)

class `kingpin.actors.group.Async(*args, **kwargs)`

Execute several `kingpin.actors.base.BaseActor` objects asynchronously.

Groups together a series of Actors and executes them asynchronously - waiting until all of them finish before returning.

Options

Concurrency Max number of concurrent executions. This will fire off N executions in parallel, and continue with the remained as soon as the first execution is done. This is faster than creating N Sync executions.

Acts An array of individual Actor definitions.

Contexts A list of dictionaries with *contextual tokens* to pass into the actors at instantiation time. If the list has more than one element, then every actor defined in `acts` will be instantiated once for each item in the `contexts` list.

Timeouts

Timeouts are disabled specifically in this actor. The sub-actors can still raise their own `kingpin.actors.exceptions.ActorTimedOut` exceptions, but since the group actors run an arbitrary number of sub actors, we have chosen to not have this actor specifically raise its own `kingpin.actors.exceptions.ActorTimedOut` exception unless the user sets the `timeout` setting.

Examples

Clone two arrays quickly.

```
{ "desc": "Clone two arrays",
  "actor": "group.Async",
  "options": {
    "contexts": [
      { "ARRAY": "NewArray1" },
      { "ARRAY": "NewArray2" }
    ],
    "acts": [
      { "desc": "do something",
        "actor": "server_array.Clone",
        "options": {
          "source": "template",
          "dest": "{ARRAY}",
        }
      }
    ]
  }
}
```

Dry Mode

Passes on the Dry mode setting to the sub-actors that are called.

Failure

In the event that one or more `acts` fail in this group, the entire group acts will return a failure to Kingpin. Because multiple actors are executing all at the same time, the all of these actors will be allowed to finish before the failure is returned.

6.10 kingpin.actors.hipchat

The Hipchat Actors allow you to send messages to a HipChat room at stages during your job execution. The actor supports dry mode by validating that the configured API Token has access to execute the methods, without actually sending the messages.

Required Environment Variables

HIPCHAT_TOKEN HipChat API Token

HIPCHAT_NAME HipChat message from name (defaults to Kingpin)

class kingpin.actors.hipchat.**HipchatBase**(*args, **kwargs)
Simple Hipchat Abstract Base Object

class kingpin.actors.hipchat.**Message**(*args, **kwargs)
Sends a message to a room in HipChat.

Options

Room (str) The string-name (or ID) of the room to send a message to

Message (str) Message to send

Examples

```
{ "actor": "hipchat.Message",
  "desc": "Send a message!",
  "options": {
    "room": "Operations",
    "message": "Beginning Deploy: v1.2"
  }
}
```

Dry Mode

Fully supported – does not actually send messages to a room, but validates that the API credentials would have access to send the message using the HipChat `auth_test` optional API argument.

class kingpin.actors.hipchat.**Topic**(*args, **kwargs)
Sets a HipChat room topic.

Options

• **room** - The string-name (or ID) of the room to set the topic of

• **topic** - String of the topic to send

Examples

```
{ "actor": "hipchat.Topic",
  "desc": "set the room topic",
  "options": {
    "room": "Operations",
    "topic": "Latest Deployment: v1.2"
  }
}
```

Dry Mode

Fully supported – does not actually set a room topic, but validates that the API credentials would have access to set the topic of the room requested.

6.11 kingpin.actors.librato

The Librato Actor allows you to post an Annotation to Librato. This is specifically useful for marking when deployments occur on your graphs for cause/effect analysis.

Required Environment Variables

LIBRATO_TOKEN Librato API Token

LIBRATO_EMAIL Librato email account (i.e. username)

class kingpin.actors.librato.**Annotation** (*args, **kwargs)
 Librato Annotation Actor

Posts an Annotation to Librato.

Options

Title The title of the annotation

Description The description of the annotation

Name Name of the metric to annotate

Examples

```
{ "actor": "librato.Annotation",
  "desc": "Mark our deployment",
  "options": {
    "title": "Deploy",
    "description": "Version: 0001a",
    "name": "production_releases"
  }
}
```

Dry Mode

Currently does not actually do anything, just logs dry mode.

6.12 kingpin.actors.misc

These are common utility Actors that don't really need their own dedicated packages. Things like sleep timers, loggers, etc.

Optional Environment Variables

URLLIB_DEBUG Set this variable to enable extreme debug logging of the URLLIB requests made by the RightScale/AWS actors. *Note, this is very insecure as headers/cookies/etc. are exposed*

class kingpin.actors.misc.**Macro** (*args, **kwargs)
 Parses a kingpin JSON file, instantiates and executes it.

Parse JSON

Kingpin JSON has 2 passes at its validity. JSON syntax must be valid, with the exception of a few useful deviations allowed by `demjson` parser. Main one being the permission of inline comments via `/* this */` syntax.

The second pass is validating the Schema. The JSON file will be validated for schema-conformity as one of the first things that happens at load-time when the app starts up. If it fails, you will be notified immediately.

Lastly after JSON is established to be valid, all the tokens are replaced with their specified value. Any key/value pair passed in the `tokens` option will be available inside of the JSON file as `%KEY%` and replaced with the value at this time.

In a situation where nested Macro executions are invoked the tokens *do not* propagate from outer macro into the inner. This allows to reuse token names, but forces the user to specify every token needed. Similarly, if environment variables are used for token replacement in the main file, these tokens are not available in the subsequent macros.

Pre-Instantiation

In an effort to prevent mid-run errors, we pre-instantiate all Actor objects all at once before we ever begin executing code. This ensures that major typos or misconfigurations in the JSON will be caught early on.

Execution

`misc.Macro` actor simply calls the `execute()` method of the most-outer actor; be it a single action, or a group actor.

Options

File String of local path to a JSON file.

Tokens Dictionary to search/replace within the file.

Examples

```
{ "desc": "Stage 1",
  "actor": "misc.Macro",
  "options": {
    "file": "deployment/stage-1.json",
    "tokens": {
      "TIMEOUT": 360,
      "RELEASE": "%RELEASE%"
    }
  }
}
```

Dry Mode

Fully supported – instantiates the actor inside of JSON with `dry=True`. The behavior of the consecutive actor is unique to each; read their description for more information on dry mode.

class `kingpin.actors.misc.Sleep` (*desc*, *options*, *dry=False*, *warn_on_failure=False*, *condition=True*, *init_context={}*, *timeout=None*)

Sleeps for an arbitrary number of seconds.

Options

Sleep Integer of seconds to sleep.

Examples

```
{ "actor": "misc.Sleep",
  "desc": "Sleep for 60 seconds",
  "options": {
    "sleep": 60
  }
}
```

Dry Mode

Fully supported – does not actually sleep, just pretends to.

class `kingpin.actors.misc.GenericHTTP` (*desc*, *options*, *dry=False*, *warn_on_failure=False*, *condition=True*, *init_context={}*, *timeout=None*)

A very simple actor that allows GET/POST methods over HTTP.

Does a GET or a POST to a specified URL.

Options

Url Destination URL

Data Optional POST data as a `dict`

Username Optional for HTTPAuth.

Password Optional for HTTPAuth.

Examples

```
{ "actor": "misc.GenericHTTP",
  "desc": "Make a simple web call",
  "options": {
    "url": "http://example.com/rest/api/v1?id=123&action=doit",
    "username": "secret",
    "password": "%SECRET_PASSWORD%"
  }
}
```

Dry Mode

Will not do anything in dry mode except print a log statement.

6.13 kingpin.actors.packagecloud

The packagecloud actor allows you to perform maintenance operations on repositories hosted by packagecloud.io using their API:

<https://packagecloud.io/docs/api>

Required Environment Variables

PACKAGECLOUD_ACCOUNT packagecloud account name, i.e.
https://packagecloud.io/PACKAGECLOUD_ACCOUNT

PACKAGECLOUD_TOKEN packagecloud API Token

class kingpin.actors.packagecloud.**PackagecloudBase** (*args, **kwargs)
 Simple packagecloud Abstract Base Object

class kingpin.actors.packagecloud.**Delete** (*args, **kwargs)
 Deletes packages from a PackageCloud repo.

Searches for packages that match the `packages_to_delete` regex pattern and deletes them. If `number_to_keep` is set, we always at least this number of versions of the given package intact in the repo. Also if `number_to_keep` is set, the older versions of a package (based on upload time) packages will be deleted first effectively leaving newer packages in the repo.

Options

Number_to_keep Keep at least this number of each package (defaults to 0)

Packages_to_delete Regexp of packages to delete, e.g. `pkg1|pkg2`

Repo Which packagecloud repo to delete from

Examples

```
{ "desc": "packagecloud Delete example",
  "actor": "packagecloud.Delete",
  "options": {
    "number_to_keep": 10,
    "packages_to_delete": "deleteme",
    "repo": "test"
  }
}
```

class kingpin.actors.packagecloud.DeleteByDate(*args, **kwargs)

Deletes packages from a PackageCloud repo older than X.

Adds additional functionality to the `Delete` class with a `older_than` option. Only packages older than that number of seconds will be deleted.

Options

Number_to_keep Keep at least this number of each package (defaults to 0)

Older_than Delete packages created before this number of seconds

Packages_to_delete Regex of packages to delete, e.g. pkg1|pkg2

Repo Which packagecloud repo to delete from

Examples

```
{ "desc": "packagecloud DeleteByDate example",
  "actor": "packagecloud.DeleteByDate",
  "options": {
    "number_to_keep": 10,
    "older_than": 600,
    "packages_to_delete": "deleteme",
    "repo": "test"
  }
}
```

class kingpin.actors.packagecloud.WaitForPackage(*args, **kwargs)

Searches for a package that matches name and version until found or a timeout occurs.

Options

Name Name of the package to search for as a regex

Version Version of the package to search for as a regex

Repo Which packagecloud repo to delete from

Sleep Number of seconds to sleep for between each search

Examples

```
{ "desc": "packagecloud WaitForPackage example",
  "actor": "packagecloud.WaitForPackage",
  "options": {
    "name": "findme",
    "version": "0.1",
    "repo": "test",
    "sleep": 10,
  }
}
```

6.14 kingpin.actors.pingdom

Pingdom actors to pause and unpause checks. These are useful when you are aware of an expected downtime and don't want to be alerted about it. Also known as Maintenance mode.

Required Environment Variables

PINGDOM_TOKEN Pingdom API Token

PINGDOM_USER Pingdom Username (email)

PINGDOM_PASS Pingdom Password

class kingpin.actors.pingdom.**PingdomBase** (*args, **kwargs)
Simple Pingdom Abstract Base Object

class kingpin.actors.pingdom.**Pause** (*args, **kwargs)
Start Pingdom Maintenance.

Pause a particular “check” on Pingdom.

Options

Name (Str) Name of the check

Example

```
{ "actor": "pingdom.Pause",
  "desc": "Run Pause",
  "options": {
    "name": "fill-in"
  }
}
```

Dry run

Will assert that the check name exists, but not take any action on it.

class kingpin.actors.pingdom.**Unpause** (*args, **kwargs)
Stop Pingdom Maintenance.

Unpause a particular “check” on Pingdom.

Options

Name (Str) Name of the check

Example

```
{ "actor": "pingdom.Unpause",
  "desc": "Run unpause",
  "options": {
    "name": "fill-in"
  }
}
```

Dry run

Will assert that the check name exists, but not take any action on it.

6.15 kingpin.actors.rightscale.api

Base RightScale API Access Object.

This package provides access to the RightScale API via Tornado-style `@gen.coroutine` wrapped methods. These methods are, however, just wrappers for threads that are being fired off in the background to make the API calls.

Async vs Threads

In the future, this will get re-factored to use a native Tornado `AsyncHTTPClient` object. The methods themselves will stay the same, but the underlying private methods will change.

The methods in this object are specifically designed to support common operations that the RightScale Actor objects need to do. Operations like ‘find server array’, ‘launch server array’, etc. This is not meant as a pure one-to-one mapping of the RightScale API, but rather a mapping of conceptual operations that the Actors need.

Method Design Note

RightScale mixes and matches their API calls... some of them you pass in a major method and then supply a resource ID to act on. Others you pass in the resource_id and get back a list of methods that you can execute.

For consistency in our programming model, this class relies on you passing in rightscale.Resource objects everywhere, and it does the resource->ID translation.

exception kingpin.actors.rightscale.api.**ServerArrayException**

Raised when an operation on or looking for a ServerArray fails

6.16 kingpin.actors.rightscale.base

The RightScale Actors allow you to interact with resources inside your Rightscale account. These actors all support dry runs properly, but each actor has its own caveats with dry=True. Please read the instructions below for using each actor.

Required Environment Variables

RIGHTSCALE_TOKEN RightScale API Refresh Token (from the *Account Settings/API Credentials* page)

RIGHTSCALE_ENDPOINT Your account-specific API Endpoint (defaults to <https://my.rightscale.com>)

exception kingpin.actors.rightscale.base.**ArrayNotFound**

Raised when a ServerArray could not be found.

exception kingpin.actors.rightscale.base.**ArrayAlreadyExists**

Raised when a ServerArray already exists by a given name.

class kingpin.actors.rightscale.base.**RightScaleBaseActor** (*args, **kwargs)

Abstract class for creating RightScale cloud actors.

6.17 kingpin.actors.rightscale.server_array

exception kingpin.actors.rightscale.server_array.**InvalidInputs**

Raised when supplied inputs are invalid for a ServerArray.

exception kingpin.actors.rightscale.server_array.**TaskExecutionFailed**

Raised when one or more RightScale Task executions fail.

class kingpin.actors.rightscale.server_array.**ServerArrayBaseActor** (*args, **kwargs)

Abstract ServerArray Actor that provides some utility methods.

class kingpin.actors.rightscale.server_array.**Clone** (*args, **kwargs)

Clones a RightScale Server Array.

Clones a ServerArray in RightScale and renames it to the newly supplied name. By default, this actor is extremely strict about validating that the source array already exists, and that the dest array does not yet exist. This behavior can be overridden though if your Kingpin script creates the source, or destroys an existing dest ServerArray sometime before this actor executes.

Options

Source The name of the ServerArray to clone

Strict_source Whether or not to fail if the source ServerArray does not exist. (default: True)

Dest The new name for your cloned ServerArray

Strict_dest Whether or not to fail if the destination ServerArray already exists. (default: True)

Examples

Clone my-template-array to my-new-array:

```
{ "desc": "Clone my array",
  "actor": "rightscale.server_array.Clone",
  "options": {
    "source": "my-template-array",
    "dest": "my-new-array"
  }
}
```

Clone an array that was created sometime earlier in the Kingpin JSON, and thus does not exist yet during the dry run:

```
{ "desc": "Clone that array we created earlier",
  "actor": "rightscale.server_array.Clone",
  "options": {
    "source": "my-template-array",
    "strict_source": false,
    "dest": "my-new-array"
  }
}
```

Clone an array into a destination name that was destroyed sometime earlier in the Kingpin JSON:

```
{ "desc": "Clone that array we created earlier",
  "actor": "rightscale.server_array.Clone",
  "options": {
    "source": "my-template-array",
    "dest": "my-new-array",
    "strict_dest": false,
  }
}
```

Dry Mode

In Dry mode this actor *does* validate that the source array exists. If it does not, a `kingpin.actors.rightscale.api.ServerArrayException` is thrown. Once that has been validated, the dry mode execution pretends to copy the array by creating a mocked cloned array resource. This mocked resource is then operated on during the rest of the execution of the actor, guaranteeing that no live resources are modified.

Example *dry* output:

```
[Copy Test (DRY Mode)] Verifying that array "temp" exists
[Copy Test (DRY Mode)] Verifying that array "new" does not exist
[Copy Test (DRY Mode)] Cloning array "temp"
[Copy Test (DRY Mode)] Renaming array "<mocked clone of temp>" to "new"
```

class kingpin.actors.rightscale.server_array.Update(*args, **kwargs)

Update ServerArray Settings

Updates an existing ServerArray in RightScale with the supplied parameters. Can update any parameter that is described in the RightScale API docs [here](#):

Parameters are passed into the actor in the form of a dictionary, and are then converted into the RightScale format. See below for examples.

Options

Array (str) The name of the ServerArray to update

Exact (bool) whether or not to search for the exact array name. (default: `true`)

Params (dict) Dictionary of parameters to update

Inputs (dict) Dictionary of next-instance server array inputs to update

Examples

```
{ "desc": "Update my array",
  "actor": "rightscale.server_array.Update",
  "options": {
    "array": "my-new-array",
    "params": {
      "elasticity_params": {
        "bounds": {
          "min_count": 4
        },
        "schedule": [
          { "day": "Sunday", "max_count": 2,
            "min_count": 1, "time": "07:00" },
          { "day": "Sunday", "max_count": 2,
            "min_count": 2, "time": "09:00" }
        ]
      },
      "name": "my-really-new-name"
    }
  }
}
```

```
{ "desc": "Update my array inputs",
  "actor": "rightscale.server_array.Update",
  "options": {
    "array": "my-new-array",
    "inputs": {
      "ELB_NAME": "text:foobar"
    }
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the array, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

This means that the dry mode cannot validate that the supplied inputs will work.

Example *dry* output:

```
[Update Test (DRY Mode)] Verifying that array "new" exists
[Update Test (DRY Mode)] Array "new" not found -- creating a mock.
```

```
[Update Test (DRY Mode)] Would have updated "<mocked array new>" with
params: {'server_array[name]': 'my-really-new-name',
        'server_array[elasticity_params][bounds][min_count]': '4'}
```

class kingpin.actors.rightscale.server_array.**UpdateNextInstance** (*args, **kwargs)
Update the Next Instance parameters for a Server Array

Updates an existing ServerArray in RightScale with the supplied parameters. Can update any parameter that is described in the RightScale [ResourceInstances](#) docs.

Note about the image_href parameter

If you pass in the string default to the image_href key in your params dictionary, we will search and find the default image that your ServerArray's Multi Cloud Image refers to. This helper is useful if you update your ServerArrays to use custom AMIs, and then occasionally want to go back to using a stock AMI. For example, if you boot up your instances occasionally off a stock AMI, customize the host, and then bake that host into a custom AMI.

Parameters are passed into the actor in the form of a dictionary, and are then converted into the RightScale format. See below for examples.

Options

Array (str) The name of the ServerArray to update

Exact (bool) whether or not to search for the exact array name. (default: true)

Params (dict) Dictionary of parameters to update

Examples

```
{ "desc": "Update my array",
  "actor": "rightscale.server_array.UpdateNextInstance",
  "options": {
    "array": "my-new-array",
    "params": {
      "associate_public_ip_address": true,
      "image_href": "/image/href/123",
    }
  }
}
```

```
{ "desc": "Reset the AMI image to the MCI default",
  "actor": "rightscale.server_array.UpdateNextInstance",
  "options": {
    "array": "my-new-array",
    "params": {
      "image_href": "default",
    }
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the array, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

This means that the dry mode cannot validate that the supplied params will work.

Example dry output:

```
[Update my array (DRY Mode)] Verifying that array "new" exists
[Update my array (DRY Mode)] Array "new" not found -- creating a mock.
[Update my array (DRY Mode)] Would have updated "<mocked array new>"
with params: {'server_array[associate_public_ip_address]': true,
              'server_array[image_href]': '/image/href/'}
```

class kingpin.actors.rightscale.server_array.**Terminate**(*args, **kwargs)

Terminate all instances in a ServerArray

Terminates all instances for a ServerArray in RightScale marking the array disabled.

Options

Array (str) The name of the ServerArray to destroy

Exact (bool) Whether or not to search for the exact array name. (default: true)

Strict (bool) Whether or not to fail if the ServerArray does not exist. (default: true)

Examples

```
{ "desc": "Terminate my array",
  "actor": "rightscale.server_array.Terminate",
  "options": {
    "array": "my-array"
  }
}
```

```
{ "desc": "Terminate many arrays",
  "actor": "rightscale.server_array.Terminate",
  "options": {
    "array": "array-prefix",
    "exact": false,
  }
}
```

Dry Mode

Dry mode still validates that the server array you want to terminate is actually gone. If you want to bypass this check, then set the `warn_on_failure` flag for the actor.

class kingpin.actors.rightscale.server_array.**Destroy**(*args, **kwargs)

Destroy a ServerArray in RightScale

Destroys a ServerArray in RightScale by first invoking the Terminate actor, and then deleting the array as soon as all of the running instances have been terminated.

Options

Array (str) The name of the ServerArray to destroy

Exact (bool) Whether or not to search for the exact array name. (default: true)

Strict (bool) Whether or not to fail if the ServerArray does not exist. (default: true)

Examples

```
{ "desc": "Destroy my array",
  "actor": "rightscale.server_array.Destroy",
  "options": {
    "array": "my-array"
  }
}
```

```
{ "desc": "Destroy many arrays",
  "actor": "rightscale.server_array.Destroy",
  "options": {
    "array": "array-prefix",
    "exact": false,
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the `array`, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

Example *dry* output:

```
[Destroy Test (DRY Mode)] Beginning
[Destroy Test (DRY Mode)] Terminating array before destroying it.
[Destroy Test (terminate) (DRY Mode)] Array "my-array" not found --
creating a mock.
[Destroy Test (terminate) (DRY Mode)] Disabling Array "my-array"
[Destroy Test (terminate) (DRY Mode)] Would have terminated all array
"<mocked array my-array>" instances.
[Destroy Test (terminate) (DRY Mode)] Pretending that array <mocked
array my-array> instances are terminated.
[Destroy Test (DRY Mode)] Pretending to destroy array "<mocked array
my-array>"
[Destroy Test (DRY Mode)] Finished successfully. Result: True
```

class kingpin.actors.rightscale.server_array.**Launch** (*args, **kwargs)

Launch instances in a ServerArray

Launches instances in an existing ServerArray and waits until that array has become healthy before returning. *Healthy* means that the array has at least the user-specified `count` or `min_count` number of instances running as defined by the array definition in RightScale.

Options

Array (str) The name of the ServerArray to launch

Count (str, int) Optional number of instance to launch. Defaults to `min_count` of the array.

Enable (bool) Should the autoscaling of the array be enabled? Settings this to `false`, or omitting the parameter will not disable an enabled array.

Exact (bool) Whether or not to search for the exact array name. (default: `true`)

Examples

```
{ "desc": "Enable array and launch it",
  "actor": "rightscale.server_array.Launch",
  "options": {
    "array": "my-array",
    "enable": true
  }
}
```

```
{ "desc": "Enable arrays starting with my-array and launch them",
  "actor": "rightscale.server_array.Launch",
  "options": {
```

```
"array": "my-array",
"enable": true,
"exact": false
}
```

```
{ "desc": "Enable array and launch 1 instance",
  "actor": "rightscale.server_array.Launch",
  "options": {
    "array": "my-array",
    "count": 1
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the array, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

Example *dry* output:

```
[Launch Array Test #0 (DRY Mode)] Verifying that array "my-array" exists
[Launch Array Test #0 (DRY Mode)] Array "my-array" not found -- creating
a mock.
[Launch Array Test #0 (DRY Mode)] Enabling Array "my-array"
[Launch Array Test #0 (DRY Mode)] Launching Array "my-array" instances
[Launch Array Test #0 (DRY Mode)] Would have launched instances of array
<MagicMock name='my-array.self.show().soul.__getitem__()'
id='4420453200'>
[Launch Array Test #0 (DRY Mode)] Pretending that array <MagicMock
name='my-array.self.show().soul.__getitem__()' id='4420453200'>
instances are launched.
```

class kingpin.actors.rightscale.server_array.**Execute** (*args, **kwargs)

Executes a RightScale script/recipe on a ServerArray

Executes a RightScript or Recipe on a set of hosts in a ServerArray in RightScale using individual calls to the live running instances. These can be found in your RightScale account under *Design -> RightScript* or *Design -> Cookbooks*

The RightScale API offers a *multi_run_executable* method that can be used to run a single script on all servers in an array – but unfortunately this API method provides no way to monitor the progress of the individual jobs on the hosts. Furthermore, the method often executes on recently terminated or terminating hosts, which throws false-negative error results.

Our actor explicitly retrieves a list of the *operational* hosts in an array and kicks off individual execution tasks for every host. It then tracks the execution of those tasks from start to finish and returns the results.

Options

Array (str) The name of the ServerArray to operate on

Script (str) The name of the RightScript or Recipe to execute

Expected_runtime (str, int) Expected number of seconds to execute. (default: 5)

Concurrency Max number of concurrent executions. This will fire off N executions in parallel, and continue with the remained as soon as the first execution is done. This is faster than creating N

Sync executions. **Note:** When applied to multiple (M) arrays cumulative concurrency accross all arrays will remain at N. It will not be M x N.

Inputs (dict) Dictionary of Key/Value pairs to use as inputs for the script

Exact (str) Boolean whether or not to search for the exact array name. (default: `true`)

Examples

```
{ "desc": "Execute script on my-array",
  "actor": "rightscale.server_array.Execute",
  "options": {
    "array": "my-array",
    "script": "connect to elb",
    "expected_runtime": 3,
    "inputs": {
      "ELB_NAME": "text:my-elb"
    }
  }
}
```

Dry Mode

In Dry mode this actor *does* search for the `array`, but allows it to be missing because its highly likely that the array does not exist yet. If the array does not exist, a mocked array object is created for the rest of the execution.

During the rest of the execution, the code bypasses making any real changes and just tells you what changes it would have made.

Example *dry* output:

```
[Destroy Test (DRY Mode)] Verifying that array "my-array" exists
[Execute Test (DRY Mode)]
    kingpin.actors.rightscale.server_array.Execute Initialized
[Execute Test (DRY Mode)] Beginning execution
[Execute Test (DRY Mode)] Verifying that array "my-array" exists
[Execute Test (DRY Mode)] Would have executed "Connect instance to ELB"
    with inputs '{"inputs[ELB_NAME]": "text:my-elb"}' on "my-array".
[Execute Test (DRY Mode)] Returning result: True
```

6.18 kingpin.actors.rollbar

The Rollbar Actor allows you to post Deploy messages to Rollbar when you execute a code deployment.

Required Environment Variables

ROLLBAR_TOKEN Rollbar API Token

class kingpin.actors.rollbar.**RollbarBase**(*args, **kwargs)
Simple Rollbar Base Abstract Actor

class kingpin.actors.rollbar.**Deploy**(*args, **kwargs)
Posts a Deploy message to Rollbar.

<https://rollbar.com/docs/deployments/other/>

API Token

You must use an API token created in your *Project Access Tokens* account settings section. This token should have *post_server_item* permissions for the actual deploy, and *read* permissions for the Dry run.

Options

Environment The environment to deploy to

Revision The deployment revision

Local_username The user who initiated the deploy

Rollbar_username (*Optional*) The Rollbar Username to assign the deploy to

Comment (*Optional*) Comment describing the deploy

Examples

```
{ "actor": "rollbar.Deploy",
  "desc": "update rollbar deploy",
  "options": {
    "environment": "Prod",
    "revision": "%DEPLOY%",
    "local_username": "Kingpin",
    "rollbar_username": "Kingpin",
    "comment": "some comment %DEPLOY%"
  }
}
```

Dry Mode

Accesses the Rollbar API and validates that the token can access your project.

6.19 kingpin.actors.slack

The Slack Actors allow you to send messages to a Slack channel at stages during your job execution. The actor supports dry mode by validating that the configured API Token has access to execute the methods, without actually sending the messages.

Required Environment Variables

SLACK_TOKEN Slack API Token

SLACK_NAME Slack *message from* name (defaults to *Kingpin*)

class kingpin.actors.slack.**SlackBase** (*args, **kwargs)
Simple Slack Abstract Base Object

class kingpin.actors.slack.**Message** (*args, **kwargs)
Sends a message to a channel in Slack.

Options

Channel The string-name of the channel to send a message to

Message String of the message to send

Examples

```
{ "desc": "Let the Engineers know things are happening",
  "actor": "slack.Message",
  "options": {
    "channel": "#operations",
    "message": "Beginning Deploy: %VER%"
  }
}
```

Dry Mode

Fully supported – does not actually send messages to a room, but validates that the API credentials would have access to send the message using the Slack `auth.test` API method.

This package provides a quick way of creating custom API clients for JSON-based REST APIs. The majority of the work is in the creation of a `_CONFIG` dictionary for the class. This dictionary dynamically configures the object at instantiation time with the appropriate `@gen.coroutine` wrapped HTTP fetch methods.

See the documentation in `docs/DEVELOPMENT.md` for more details on how to use this package to create your own API client.

```
kingpin.actors.support.api.create_http_method(name, http_method)
```

Creates the `get/put/delete/post` coroutined-method for a resource.

This method is called during the `__init__` of a `RestConsumer` object. The method creates a custom method that handles a GET, PUT, POST or DELETE through the Tornado `HTTPClient` class.

Args: `http_method`: Name of the method (`get`, `put`, `post`, `delete`)

Returns: A method appropriately configured and named.

```
kingpin.actors.support.api.create_method(name, config)
```

Creates a `RestConsumer` object.

Configures a fresh `RestConsumer` object with the supplied configuration bits. The configuration includes information about the name of the method being consumed and the configuration of that method (which HTTP methods it supports, etc).

The final created method accepts any args (`*args`, `**kwargs`) and passes them on to the `RestConsumer` object being created. This allows for passing in unique resource identifiers (ie, the `'%res%'` in `'/v2/rooms/%res%/history'`).

Args: `name`: The name passed into the `RestConsumer` object `config`: The config passed into the `RestConsumer` object

Returns: A method that returns a fresh `RestConsumer` object

```
class kingpin.actors.support.api.RestConsumer(name=None, config=None, client=None,
                                              *args, **kwargs)
```

An abstract object that self-defines its own API access methods.

At init time, this object reads its `_CONFIG` and pre-defines all of the API access methods that have been described. It does not handle actual HTTP calls directly, but is passed in a `client` object (anything that subclasses the `RestClient` class) and leverages that for the actual web calls.

```
class kingpin.actors.support.api.RestClient(client=None, headers=None)
```

Very simple REST client for the `RestConsumer`. Implements a `AsyncHTTPClient()`, some convenience methods for URL escaping, and a single `fetch()` method that can handle GET/POST/PUT/DELETES.

This code is nearly identical to the `kingpin.actors.base.BaseHTTPActor` class, but is not actor-specific.

Args: `headers`: Headers to pass in on every HTTP request

```
class kingpin.actors.support.api.SimpleTokenRestClient(tokens, *args, **kwargs)
```

Simple `RestClient` that appends a `'token'` to every web request for authentication. Used in most simple APIs where a token is provided to the end user.

Args:

tokens: (dict) A dict with the token name/value(s) to append to every we request.

6.20 kingpin.actors.utils

Misc methods for dealing with Actors.

`kingpin.actors.utils.get_actor` (*config*, *dry*)
Returns an initialized Actor object.

Args:

config: A dictionary of configuration data that conforms to our **v1** schema in `kingpin.schema`. Looks like this:

```
{ 'desc': <string description of actor>, 'actor': <string name of actor> 'options': <dict of options to
pass to actor> 'warn_on_failure': <bool> 'condition': <string or bool> }
```

dry: Boolean whether or not in Dry mode **warn_on_failure:** Boolean

Returns: <actor object>

`kingpin.actors.utils.get_actor_class` (*actor*)
Returns a Class Reference to an Actor by string name.

Args: *actor*: String name of the actor to find.

Returns: <Class Ref to Actor>

class `kingpin.constants.REQUIRED`
Meta class to identify required arguments for actors.

exception `kingpin.exceptions.KingpinException`
Base Exception

exception `kingpin.exceptions.InvalidJSON`
Raised when an invalid JSON schema was detected

`kingpin.schema.validate` (*config*)
Validates the JSON against our schemas.

TODO: Support multiple schema versions

Args: *config*: Dictionary of parsed JSON

Returns: None: if all is well

Raises: Exception if something went wrong.

6.21 kingpin.utils

Common package for utility functions.

`kingpin.utils.str_to_class` (*string*)
Method that converts a string name into a usable Class name

This is used to take the 'actor' value from the JSON object and convert it into a valid object reference.

Args:

cls: String name of the wanted class and package. eg: `kingpin.actor.foo.bar`

Returns: A reference to the actual Class to be instantiated

`kingpin.utils.setup_root_logger` (*level='warn'*, *syslog=None*, *color=False*)
Configures the root logger.

Args: level: Logging level string ('warn' is default) syslog: String representing syslog facility to output to. If empty, logs are written to console. color: Colorize the log output

Returns: A root Logger object

`kingpin.utils.super_httpplib_debug_logging()`

Enables DEBUG logging deep in HTTPLIB.

HTTPLib by default doesn't log out things like the raw HTTP headers, cookies, response body, etc – even when your main logger is in DEBUG mode. This is because its a security risk, as well as just highly verbose.

For the purposes of debugging though, this can be useful. This method enables deep debug logging of the HTTPLib web requests. This is highly insecure, but very useful when troubleshooting failures with remote API endpoints.

Returns: Requests 'logger' object (mainly for unit testing)

`kingpin.utils.exception_logger(func)`

Explicitly log Exceptions then Raise them.

Logging Exceptions and Tracebacks while inside of a thread is broken in the Tornado futures package for Python 2.7. It swallows most of the traceback and only gives you the raw exception object. This little helper method allows us to throw a log entry with the full traceback before raising the exception.

`kingpin.utils.retry(excs, retries=3, delay=0.25)`

Coroutine-compatible Retry Decorator.

This decorator provides a simple retry mechanism that looks for a particular set of exceptions and retries async tasks in the event that those exceptions were caught.

Example usage:

```
>>> @gen.coroutine
... @retry(excs=(Exception), retries=3)
... def login(self):
...     raise gen.Return()
```

Args: excs: A single (or tuple) exception type to catch. retries: The number of times to try the operation in total. delay: Time (in seconds) to wait between retries

`kingpin.utils.tornado_sleep(*args, **kwargs)`

Async method equivalent to sleeping.

Args: seconds: Float seconds. Default 1.0

`kingpin.utils.populate_with_tokens(string, tokens, left_wrapper='%', right_wrapper='%', strict=True)`

Insert token variables into the string.

Will match any token wrapped in '%'s and replace it with the value of that token.

Args: string: string to modify. tokens: dictionary of key:value pairs to inject into the string. left_wrapper: the character to use as the START of a token right_wrapper: the character to use as the END of a token strict: (bool) whether or not to make sure all tokens were replaced

Example: export ME=biz

```
string='foo %ME% %bar%' populate_with_tokens(string, os.environ) # 'foo biz %bar%'
```

`kingpin.utils.convert_json_to_dict(json_file, tokens)`

Converts a JSON file to a config dict.

Reads in a JSON file, swaps out any environment variables that have been used inside the JSON, and then returns a dictionary.

Args: `json_file`: Path to the JSON file to import, or file instance. `tokens`: dictionary to pass to `populate_with_tokens`.

Returns: <Dictionary of Config Data>

`kingpin.utils.create_repeating_log(logger, message, handle=None, **kwargs)`

Create a repeating log message.

This function sets up tornado to repeatedly log a message in a way that does not need to be `yield`-ed.

Example:

```
>>> yield do_tornado_stuff(1)
>>> log_handle = create_repeating_log('Computing...')
>>> yield do_slow_computation_with_insufficient_logging()
>>> clear_repeating_log(log_handle)
```

This is similar to javascript's `setInterval()` and `clearInterval()`.

Args: `message`: String to pass to `log.info()` `kwargs`: values accepted by `datetime.timedelta` namely seconds, and milliseconds.

Must be cleared via `clear_repeating_log()` Only handles one interval per actor.

`kingpin.utils.clear_repeating_log(handle)`

Stops the timeout function from being called.

- `genindex`
- `modindex`
- `search`

a

kingpin.actors.aws.base, [51](#)
kingpin.actors.aws.cloudformation, [51](#)
kingpin.actors.aws.elb, [52](#)
kingpin.actors.aws.iam, [55](#)
kingpin.actors.aws.settings, [56](#)
kingpin.actors.aws.sqs, [56](#)

b

kingpin.actors.base, [57](#)

c

kingpin.constants, [78](#)

e

kingpin.actors.exceptions, [58](#)
kingpin.exceptions, [78](#)

g

kingpin.actors.group, [59](#)

h

kingpin.actors.hipchat, [61](#)

l

kingpin.actors.librato, [62](#)

m

kingpin.actors.misc, [63](#)

p

kingpin.actors.packagecloud, [65](#)
kingpin.actors.pingdom, [66](#)

r

kingpin.actors.rightscale.api, [67](#)
kingpin.actors.rightscale.base, [68](#)
kingpin.actors.rightscale.server_array,
[68](#)
kingpin.actors.rollbar, [75](#)

s

kingpin.actors.slack, [76](#)
kingpin.actors.support.api, [77](#)
kingpin.schema, [78](#)

u

kingpin.actors.utils, [77](#)
kingpin.utils, [78](#)

v

kingpin.version, [80](#)

A

ActorException, 58
 ActorTimedOut, 59
 Annotation (class in kingpin.actors.librato), 63
 ArrayAlreadyExists, 68
 ArrayNotFound, 68
 Async (class in kingpin.actors.group), 60

B

BadRequest, 59
 BaseActor (class in kingpin.actors.base), 58
 BaseGroupActor (class in kingpin.actors.group), 59

C

CertNotFound, 53
 clear_repeating_log() (in module kingpin.utils), 80
 Clone (class in kingpin.actors.rightscale.server_array), 68
 CloudFormationBaseActor (class in kingpin.actors.aws.cloudformation), 51
 CloudFormationError, 51
 convert_json_to_dict() (in module kingpin.utils), 79
 Create (class in kingpin.actors.aws.cloudformation), 51
 Create (class in kingpin.actors.aws.sqs), 56
 create_http_method() (in module kingpin.actors.support.api), 77
 create_method() (in module kingpin.actors.support.api), 77
 create_repeating_log() (in module kingpin.utils), 80

D

Delete (class in kingpin.actors.aws.cloudformation), 52
 Delete (class in kingpin.actors.aws.sqs), 56
 Delete (class in kingpin.actors.packagecloud), 65
 DeleteByDate (class in kingpin.actors.packagecloud), 65
 DeleteCert (class in kingpin.actors.aws.iam), 55
 Deploy (class in kingpin.actors.rollbar), 75
 DeregisterInstance (class in kingpin.actors.aws.elb), 54
 Destroy (class in kingpin.actors.rightscale.server_array), 72

E

ELBBaseActor (class in kingpin.actors.aws.elb), 53
 ELBNotFound, 51
 exception_logger() (in module kingpin.utils), 79
 Execute (class in kingpin.actors.rightscale.server_array), 74

G

GenericHTTP (class in kingpin.actors.misc), 64
 get_actor() (in module kingpin.actors.utils), 78
 get_actor_class() (in module kingpin.actors.utils), 78

H

HipchatBase (class in kingpin.actors.hipchat), 62
 HTTPBaseActor (class in kingpin.actors.base), 58

I

IAMBaseActor (class in kingpin.actors.aws.iam), 55
 InvalidActor, 59
 InvalidCredentials, 59
 InvalidInputs, 68
 InvalidJSON, 78
 InvalidMetaData, 51
 InvalidOptions, 59
 InvalidTemplate, 51
 is_retriable_exception() (in module kingpin.actors.aws.settings), 56

K

kingpin.actors.aws.base (module), 51
 kingpin.actors.aws.cloudformation (module), 51
 kingpin.actors.aws.elb (module), 52
 kingpin.actors.aws.iam (module), 55
 kingpin.actors.aws.settings (module), 56
 kingpin.actors.aws.sqs (module), 56
 kingpin.actors.base (module), 57
 kingpin.actors.exceptions (module), 58
 kingpin.actors.group (module), 59
 kingpin.actors.hipchat (module), 61
 kingpin.actors.librato (module), 62

kingpin.actors.misc (module), 63
kingpin.actors.packagecloud (module), 65
kingpin.actors.pingdom (module), 66
kingpin.actors.rightscale.api (module), 67
kingpin.actors.rightscale.base (module), 68
kingpin.actors.rightscale.server_array (module), 68
kingpin.actors.rollbar (module), 75
kingpin.actors.slack (module), 76
kingpin.actors.support.api (module), 77
kingpin.actors.utils (module), 77
kingpin.constants (module), 78
kingpin.exceptions (module), 78
kingpin.schema (module), 78
kingpin.utils (module), 78
kingpin.version (module), 80
KingpinException, 78

L

Launch (class in kingpin.actors.rightscale.server_array), 73

M

Macro (class in kingpin.actors.misc), 63
Message (class in kingpin.actors.hipchat), 62
Message (class in kingpin.actors.slack), 76

O

option() (kingpin.actors.base.BaseActor method), 58

P

p2f() (in module kingpin.actors.aws.elb), 53
PackagecloudBase (class in kingpin.actors.packagecloud), 65
Pause (class in kingpin.actors.pingdom), 67
PingdomBase (class in kingpin.actors.pingdom), 67
populate_with_tokens() (in module kingpin.utils), 79

Q

QueueDeletionFailed, 56
QueueNotFound, 56

R

readfile() (kingpin.actors.base.BaseActor method), 58
RecoverableActorFailure, 59
RegisterInstance (class in kingpin.actors.aws.elb), 54
REQUIRED (class in kingpin.constants), 78
RestClient (class in kingpin.actors.support.api), 77
RestConsumer (class in kingpin.actors.support.api), 77
retry() (in module kingpin.utils), 79
RightScaleBaseActor (class in kingpin.actors.rightscale.base), 68
RollbarBase (class in kingpin.actors.rollbar), 75

S

ServerArrayBaseActor (class in kingpin.actors.rightscale.server_array), 68
ServerArrayException, 68
SetCert (class in kingpin.actors.aws.elb), 53
setup_root_logger() (in module kingpin.utils), 78
SimpleTokenRestClient (class in kingpin.actors.support.api), 77
SlackBase (class in kingpin.actors.slack), 76
Sleep (class in kingpin.actors.misc), 64
StackAlreadyExists, 51
StackNotFound, 51
str_to_class() (in module kingpin.utils), 78
super_httplib_debug_logging() (in module kingpin.utils), 79
Sync (class in kingpin.actors.group), 59

T

TaskExecutionFailed, 68
Terminate (class in kingpin.actors.rightscale.server_array), 72
timeout() (kingpin.actors.base.BaseActor method), 58
timer() (kingpin.actors.base.BaseActor method), 58
Topic (class in kingpin.actors.hipchat), 62
tornado_sleep() (in module kingpin.utils), 79

U

UnparseableResponseFromEndpoint, 59
Unpause (class in kingpin.actors.pingdom), 67
UnrecoverableActorFailure, 59
Update (class in kingpin.actors.rightscale.server_array), 69
UpdateNextInstance (class in kingpin.actors.rightscale.server_array), 71
UploadCert (class in kingpin.actors.aws.iam), 55

V

validate() (in module kingpin.schema), 78

W

WaitForPackage (class in kingpin.actors.packagecloud), 66
WaitUntilEmpty (class in kingpin.actors.aws.sqs), 57
WaitUntilHealthy (class in kingpin.actors.aws.elb), 53