vxsandbox Documentation

Release 0.5.0a

Praekelt Foundation

March 04, 2015

Contents

This library provides Vumi application workers that launch sandboxed processes when messages are received.

Sandboxed processes communicate with the controlling worker via simple JSON-formatted commands that are send and received over `stdin` and `stdout`. Errors may be logged over `stderr`.

Sandboxed processes are given access to additional functionality using `resources`. Resources provide additional commands.

In addition to the generic sandbox worker, a custom Javascript sandbox worker is provided that launches sandboxed Javascript code using Node.js.

CHAPTER 1

Contents

1.1 Workers

The following sandbox application workers are available:

1.1.1 Generic Sandbox worker

The generic sandbox worker that launches the executable given in its config with the provided arguments when a message or event is received.

Configuration options

class vxsandbox.worker.**SandboxConfig**(*config_data*, *static=False*)

Configuration options:

Parameters

- **amqp_prefetch_count** (*int*) – The number of messages fetched concurrently from each AMQP queue by each worker instance.
- **transport_name** (*str*) – The name this application instance will use to create its queues.
- **send_to** (*dict*) – 'send_to' configuration dict.
- **sandbox** (*dict*) – Dictionary of resources to provide to the sandbox. Keys are the names of resources (as seen inside the sandbox). Values are dictionaries which must contain a *cls* key that gives the full name of the class that provides the resource. Other keys are additional configuration for that resource.
- **executable** (*str*) – Full path to the executable to run in the sandbox.
- **args** (*list*) – List of arguments to pass to the executable (not including the path of the executable itself).
- **path** (*str*) – Current working directory to run the executable in.
- **env** (*dict*) – Custom environment variables for the sandboxed process.
- **timeout** (*int*) – Length of time the subprocess is given to process a message.
- **recv_limit** (*int*) – Maximum number of bytes that will be read from a sandboxed process' stdout and stderr combined.

- **rlimits** (*dict*) – Dictionary of resource limits to be applied to sandboxed processes. Defaults are fairly restricted. Keys maybe names or values of the RLIMIT constants in Python *resource* module. Values should be appropriate integers.
- **logging_resource** (*str*) – Name of the logging resource to use to report errors detected in sandboxed code (e.g. lines written to stderr, unexpected process termination). Set to null to disable and report these directly using Twisted logging instead.
- **sandbox_id** (*str*) – This is set based on individual messages.

Worker class

class `vxsandbox.worker.`**`Sandbox`**(*options*, *config=None*)

Sandbox application worker.

`CONFIG_CLASS`

alias of `SandboxConfig`

`sandbox_id_for_message`(*msg_or_event*)

Return a sandbox id for a message or event.

This implementation simply returns `msg_or_event['sandbox_id']`. Sub-classes may override this to retrieve a more appropriate id.

`sandbox_protocol_for_message`(*msg_or_event*, *config*)

Return a sandbox protocol for a message or event.

This implementation ignores `msg_or_event` and returns a sandbox protocol based on the given `config`. Sub-classes may override this to retrieve a custom protocol if needed.

1.1.2 Javascript Sandbox

The Javascript sandbox worker runs Javascript code in a Node.js instance. The provided Javascript code is run inside a thin wrapper which handles sending and receiving sandbox commands.

People writing Javascript code for the sandbox are strongly encouraged to use vumi-jssandbox-toolkit which provides a rich set of features for interacting with the sandbox resources.

Configuration options

class `vxsandbox.worker.`**`JsSandboxConfig`**(*config_data*, *static=False*)

JavaScript sandbox configuration.

Configuration options:

Parameters

- **amqp_prefetch_count** (*int*) – The number of messages fetched concurrently from each AMQP queue by each worker instance.
- **transport_name** (*str*) – The name this application instance will use to create its queues.
- **send_to** (*dict*) – 'send_to' configuration dict.
- **sandbox** (*dict*) – Dictionary of resources to provide to the sandbox. Keys are the names of resources (as seen inside the sandbox). Values are dictionaries which must contain a *cls* key that gives the full name of the class that provides the resource. Other keys are additional configuration for that resource.

- **executable** (*str*) – Full path to the executable to run in the sandbox.
- **args** (*list*) – List of arguments to pass to the executable (not including the path of the executable itself).
- **path** (*str*) – Current working directory to run the executable in.
- **env** (*dict*) – Custom environment variables for the sandboxed process.
- **timeout** (*int*) – Length of time the subprocess is given to process a message.
- **recv_limit** (*int*) – Maximum number of bytes that will be read from a sandboxed process' stdout and stderr combined.
- **rlimits** (*dict*) – Dictionary of resource limits to be applied to sandboxed processes. Defaults are fairly restricted. Keys maybe names or values of the RLIMIT constants in Python *resource* module. Values should be appropriate integers.
- **sandbox_id** (*str*) – This is set based on individual messages.
- **javascript** (*str*) – JavaScript code to run.
- **app_context** (*str*) – Custom context to execute JS with.
- **logging_resource** (*str*) – Name of the logging resource to use to report errors detected in sandboxed code (e.g. lines written to stderr, unexpected process termination). Set to null to disable and report these directly using Twisted logging instead.

Worker class

class `vxsandbox.worker.`**`JsSandbox`**(*options*, *config=None*)

Configuration options:

As for `Sandbox` except:

- *executable* defaults to searching for a *node.js* binary.
- *args* defaults to the JS sandbox script in the *vumi.application* module.
- An instance of `JsSandboxResource` is added to the sandbox resources under the name *js* if no *js* resource exists.
- An instance of `LoggingResource` is added to the sandbox resources under the name *log* if no *log* resource exists.
- *logging_resource* is set to *log* if it is not set.
- An extra 'javascript' parameter specifies the javascript to execute.
- An extra optional 'app_context' parameter specifying a custom context for the 'javascript' application to execute with.

Example 'javascript' that logs information via the sandbox API (provided as 'this' to 'on_inbound_message') and checks that logging was successful:

```
api.on_inbound_message = function(command) {
    this.log_info("From command: inbound-message", function (reply) {
        this.log_info("Log successful: " + reply.success);
        this.done();
    });
}
```

Example 'app_context' that makes the Node.js 'path' module available under the name 'path' in the context that the sandboxed javascript executes in:

```
{path: require('path')}
```

CONFIG_CLASS
alias of JsSandboxConfig

app_context_for_api(*api*)
Called by JsSandboxResource

Returns String containing Javascript expression that returns addition context for the namespace the app is being run in. This Javascript is expected to be trusted code.

javascript_for_api(*api*)
Called by JsSandboxResource.

Returns String containing Javascript for the app to run.

Javascript sandbox resources

This special resource provides a means of sending the Javascript code to be executed into the Node.js process. It is automatically included by the Javascript sandbox worker.

class vxsandbox.worker.**JsSandboxResource**(*name*, *app_worker*, *config*)
Resource that initializes a Javascript sandbox.

Typically used alongside vumi/applicaiton/sandboxer.js which is a simple node.js based Javascript sandbox.

Requires the worker to have a *javascript_for_api* method.

1.1.3 Javascript File Sandbox

A sub-class of JsSandbox that reads the Javascript to execute from a static file.

class vxsandbox.worker.**JsFileSandbox**(*options*, *config=None*)

class CONFIG_CLASS(*config_data*, *static=False*)
Configuration options:

Parameters

- **amqp_prefetch_count** (*int*) – The number of messages fetched concurrently from each AMQP queue by each worker instance.
- **transport_name** (*str*) – The name this application instance will use to create its queues.
- **send_to** (*dict*) – 'send_to' configuration dict.
- **sandbox** (*dict*) – Dictionary of resources to provide to the sandbox. Keys are the names of resources (as seen inside the sandbox). Values are dictionaries which must contain a *cls* key that gives the full name of the class that provides the resource. Other keys are additional configuration for that resource.
- **executable** (*str*) – Full path to the executable to run in the sandbox.
- **args** (*list*) – List of arguments to pass to the executable (not including the path of the executable itself).
- **path** (*str*) – Current working directory to run the executable in.

- **env** (*dict*) – Custom environment variables for the sandboxed process.
- **timeout** (*int*) – Length of time the subprocess is given to process a message.
- **recv_limit** (*int*) – Maximum number of bytes that will be read from a sandboxed process' stdout and stderr combined.
- **rlimits** (*dict*) – Dictionary of resource limits to be applied to sandboxed processes. Defaults are fairly restricted. Keys maybe names or values of the RLIMIT constants in Python *resource* module. Values should be appropriate integers.
- **logging_resource** (*str*) – Name of the logging resource to use to report errors detected in sandboxed code (e.g. lines written to stderr, unexpected process termination). Set to null to disable and report these directly using Twisted logging instead.
- **sandbox_id** (*str*) – This is set based on individual messages.
- **javascript_file** (*str*) – The file containting the Javascript to run
- **app_context** (*str*) – Custom context to execute JS with.

1.2 Resources

The following sandbox resource types are available:

1.2.1 Redis key-value store

A resource that provides access to a Redis-based key-value store.

class `vxsandbox.resources.kv.`**`RedisResource`**(*name*, *app_worker*, *config*)

Resource that provides access to a simple key-value store.

Configuration options:

Parameters

- **redis_manager** (*dict*) – Redis manager configuration options.
- **keys_per_user_soft** (*int*) – Maximum number of keys each user may make use of in redis before usage warnings are logged. (default: 80% of hard limit).
- **keys_per_user_hard** (*int*) – Maximum number of keys each user may make use of in redis (default: 100). Falls back to keys_per_user.
- **keys_per_user** (*int*) – Synonym for *keys_per_user_hard*. Deprecated.

`handle_delete`(**args*, ***kwargs*)

Delete a key.

Command fields:

- `key`: The key to delete.

Reply fields:

- `success`: `true` if the operation was successful, otherwise `false`.

Example:

```
api.request(
    'kv.delete',
    {key: 'foo'},
    function(reply) {
        api.log_info('Value deleted: ' +
                     reply.success);
    }
);
```

handle_get(**args*, ***kwargs*)

Retrieve the value of a key.

Command fields:

- `key`: The key whose value should be retrieved.

Reply fields:

- `success`: `true` if the operation was successful, otherwise `false`.
- `value`: The value retrieved.

Example:

```
api.request(
    'kv.get',
    {key: 'foo'},
    function(reply) {
        api.log_info(
            'Value retrieved: ' +
            JSON.stringify(reply.value));
    }
);
```

handle_incr(**args*, ***kwargs*)

Atomically increment the value of an integer key.

The current value of the key must be an integer. If the key does not exist, it is set to zero.

Command fields:

- `key`: The key to delete.
- `amount`: The integer amount to increment the key by. Defaults to 1.

Reply fields:

- `success`: `true` if the operation was successful, otherwise `false`.
- `value`: The new value of the key.

Example:

```
api.request(
    'kv.incr',
    {key: 'foo',
     amount: 3},
    function(reply) {
        api.log_info('New value: ' +
                     reply.value);
    }
);
```

handle_set(**args*, ***kwargs*)
Set the value of a key.

Command fields:

- `key`: The key whose value should be set.
- `value`: The value to store. May be any JSON serializable object.
- `seconds`: Lifetime of the key in seconds. The default `null` indicates that the key should not expire.

Reply fields:

- `success`: `true` if the operation was successful, otherwise `false`.

Example:

```
api.request(
    'kv.set',
    {key: 'foo',
     value: {x: '42'}},
    function(reply) { api.log_info('Value store: ' +
                                   reply.success); });
```

1.2.2 HTTP client

A resource that provides support for making HTTP requests.

class vxsandbox.resources.http.**HttpClientResource**(*name*, *app_worker*, *config*)
Resource that allows making HTTP calls to outside services.

All command on this resource share a common set of command and response fields:

Command fields:

- `url`: The URL to request
- **`verify_options`: A list of options to verify when doing** an HTTPS request. Possible string values are `VERIFY_NONE`, `VERIFY_PEER`, `VERIFY_CLIENT_ONCE` and `VERIFY_FAIL_IF_NO_PEER_CERT`. Specifying multiple values results in passing along a reduced `OR` value (e.g. VERIFY_PEER | VERIFY_FAIL_IF_NO_PEER_CERT)
- **`headers`: A dictionary of keys for the header name and a list** of values to provide as header values.
- `data`: The payload to submit as part of the request.
- **`files`: A dictionary, submitted as multipart/form-data** in the request:

```
[{
    "field name": {
        "file_name": "the file name",
        "content_type": "content-type",
        "data": "data to submit, encoded as base64",
    }
}, ...]
```

The `data` field in the dictionary will be base64 decoded before the HTTP request is made.

Success reply fields:

- `success`: Set to `true`

- `body`: The response body
- `code`: The HTTP response code

Failure reply fields:

- `success`: set to `false`
- `reason`: Reason for the failure

Example:

```
api.request(
    'http.get',
    {url: 'http://foo/'},
    function(reply) { api.log_info(reply.body); });
```

agent_class
alias of `Agent`

handle_delete(*api*, *command*)
Make an HTTP DELETE request.

See `HttpResource` for details.

handle_get(*api*, *command*)
Make an HTTP GET request.

See `HttpResource` for details.

handle_head(*api*, *command*)
Make an HTTP HEAD request.

See `HttpResource` for details.

handle_patch(*api*, *command*)
Make an HTTP PATCH request.

See `HttpResource` for details.

handle_post(*api*, *command*)
Make an HTTP POST request.

See `HttpResource` for details.

handle_put(*api*, *command*)
Make an HTTP PUT request.

See `HttpResource` for details.

1.2.3 Logging

A resource that provides support for logging messages via Twisted's logging infrastructure.

class `vxsandbox.resources.logging.`**LoggingResource**(*name*, *app_worker*, *config*)
Resource that allows a sandbox to log messages via Twisted's logging framework.

handle_critical(*api*, *command*)
Logs a message at the `CRITICAL` log level.

See `handle_log()` for details.

handle_debug(*api*, *command*)
Logs a message at the DEBUG log level.

See handle_log() for details.

handle_error(*api*, *command*)
Logs a message at the ERROR log level.

See handle_log() for details.

handle_info(*api*, *command*)
Logs a message at the INFO log level.

See handle_log() for details.

handle_log(**args*, ***kwargs*)
Log a message at the specified severity level.

The other log commands are identical except that level need not be specified. Using the log-level specific commands is preferred.

Command fields:

- level: The severity level to log at. Must be an integer log level. Default severity is the INFO log level.
- msg: The message to log.

Reply fields:

- success: true if the operation was successful, otherwise false.

Example:

```
api.request(
    'log.log',
    {level: 20,
     msg: 'Abandon ship!'},
    function(reply) {
        api.log_info('New value: ' +
                     reply.value);
    }
);
```

handle_warning(*api*, *command*)
Logs a message at the WARNING log level.

See handle_log() for details.

log(*api*, *msg*, *level*)
Logs a message via vumi.log (i.e. Twisted logging).

Sub-class should override this if they wish to log messages elsewhere. The *api* parameter is provided for use by such sub-classes.

The *log* method should always return a deferred.

1.2.4 Outbound messaging

A resource that provides support for sending replies and other outbound messages.

class vxsandbox.resources.outbound.**OutboundResource**(*name*, *app_worker*, *config*)
Resource that provides the ability to send outbound messages.

Includes support for replying to the sender of the current message, replying to the group the current message was from and sending messages that aren't replies.

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

V

A

C

H

J

L

O

R

S

V