

A close-up photograph of a robotic arm, likely from a manufacturing or laboratory setting. The arm is dark-colored and has numerous thin, white cables or wires bundled together and extending from its joints. The background is a solid, bright blue color. The overall image has a slightly grainy, vintage aesthetic.

Writing Execution Modules

For SaltStack with a little bit of Python

Agenda

1. Writing an Salt execution module
2. Scenario #1: Bug in the cron module
3. Scenario #2: Writing a telegram module

Required knowledge and skills

Basic Python knowledge, you should be able to read, write and understand a simple Python script.

Basic Salt knowledge, you should know what a Salt module is.



Modules are easy to write!

A Salt execution module is a **Python** module placed in a directory called `_modules/` at the root of the Salt fileserver, e.g.:

`/srv/salt/_modules/`.

The module's default name is its basename, e.g.: `foo.py` becomes `foo`, and can be overwritten by using a `__virtual__` function.

If the module has errors, the Salt minion will skip the faulty module from being loaded!

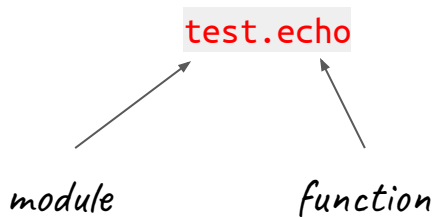


```
# Modules placed in `_modules/' will be synced
# when any of the Salt functions are called:
$ salt '*' state.apply
$ salt '*' saltutil.sync_modules
$ salt '*' saltutil.sync_all
```

Let's verify that!

In the Salt source code all modules are stored in: `salt/modules/*`.

Example: `salt '*' test.echo 'foo'`.



```
# Module `test.echo` in `salt/modules/test.py`
...

def echo(text):
    """
    Return a string - used for testing the connection
    CLI Example:
    .. code-block:: bash
        salt '*' test.echo 'foo bar baz quo qux'
    """
    return text

...
```

Writing the `foo` module

We will write the `foo` module as a simple module example in:

`/srv/salt/_modules/foo.py`.

When the `foo.bar` method is called, it will return the argument just like `test.echo` does.

```
...
Foo execution module
...

def bar(text):
    """
    Return a string - used as an example module

    CLI Example:

    .. code-block:: bash

        salt '*' foo.bar 'Hello SaltStack!'

    """
    return text
```

Execute the `foo` module

The (updated) Salt modules needs to be synced to all the minions using the following command:

```
salt '*' saltutil.sync_modules
```

The `foo` module is now available for execution:

```
salt '*' foo.bar 'Hello SaltStack!'
```



```
# Sync the modules to all the minions
$ salt '*' saltutil.sync_modules

minion1:
  - modules.foo

# Execute the newly created Salt module
$ salt '*' foo.bar 'Hello SaltStack!'

minion1:
  Hello SaltStack!
```

Adding documentation to Salt modules

To add documentation you simply have to add a [Python docstring](#) to the function.

When the `sys.doc` call is executed it will return the docstring to the calling terminal.



```
$ salt '*' sys.doc test.echo  
test.echo:
```

Return a string - used for testing the connection

CLI Example:

```
salt '*' test.echo 'foo bar baz quo qux'
```




Scenario #1

Bug in the **cron** module

Scenario #1: bug in the `cron` module

The identifier is not working in

`salt.states.cron.present` when a special is used.

In order to fix this issue, the

`salt.modules.cron.set_special` will have to allow an identifier to be set/used.

Let's submit an issue on [GitHub](#)!



In this case the IDENTIFIER is used:

```
my-cron:
  cron.present:
    - name: my-cron
    - identifier: IDENTIFIER
    - minute: '00'
    - hour: '*'
```

In this case the IDENTIFIER isn't used:

```
my-cron:
  cron.present:
    - name: my-cron
    - identifier: IDENTIFIER
    - special: '@hourly'
```


Scenario #1: submit issue on GitHub

When submitting an issue your are asked to supply a description of the issue and some debug information.

Identifier not working in salt.states.cron when special is used #44530

Edit

New issue

 Closed roaldnfs opened this issue on Nov 14, 2017 · 8 comments



roaldnfs commented on Nov 14, 2017

Contributor



Description of Issue

The `identifier` is not working in `salt.states.cron.present` when a `special` is used.

The `salt.states.cron.present` allows an `identifier` to be set. The underlying code checks whether or not a `special` is used. In the case a special is used, the `salt.module.cron.set_special` is called instead of `salt.modules.cron.set_job` function. The `salt.module.cron.set_special` does nothing with the `identifier`.

Part of the code in `salt.states.cron.present` responsible for the issue:

```
def present(name,
```

Assignees

No one assigned

Labels

Bug

High Severity

P4

State Module

Projects

None yet

Scenario #1: contributing to SaltStack

Within a few hours I've received the following response from [@Ch3LL](#)...

There is a great need for contributions to Salt and patches are welcome!

The [documentation](#) is very extensive in this area.



Ch3LL commented on Nov 14, 2017

Contributor



@roaldnefs mind taking a stab at a PR for this? Seems you have a pretty good grasp on the fix.



Ch3LL added **Bug** **High Severity** **P4** **State Module** labels on Nov 14, 2017



Ch3LL added this to the **Approved** milestone on Nov 14, 2017



roaldnefs commented on Nov 14, 2017

Author

Contributor



@Ch3LL I'll give it a try later this week!



1

Scenario #1: submit the Pull Request

You can 'overwrite' any module by placing a module with the same name in `_modules/` at the root of the Salt fileserver.

Instead I've used the [documentation](#) to setup a local development environment.

Fix bug in cron module and state - Fixes #44530 #44579 Edit

Merged

garthgreenaway merged 4 commits into saltstack:2016.11 from unknown repository on Dec 7, 2017

Conversation 0

Commits 4

Checks 0

Files changed 2

+118 -20



roaldnefs commented on Nov 16, 2017

Contributor

+ 😊 ...

What does this PR do?

Allows the `identifier`, `comment` and `commented` arguments in `salt.states.cron` and `salt.states.cron` to be used.

What issues does this PR fix or reference?

Fixes #44530

Reviewers

garthgreenaway ✓

Assignees

No one assigned

Labels

None yet

Scenario #1: use the bugfix before it's accepted

You can 'overwrite' any module/state by placing a module or state with the same name in `_modules/` and `_states/` at the root of the Salt fileserver.



chrispetsos commented on Dec 14, 2017

+ 🗨️ ...

Any suggestions on how could we bring this fix in newer versions of SaltStack also?



rallytime commented on Dec 14, 2017

Contributor

+ 🗨️ ...

Hi @chrispetsos - we use a merge-forward strategy for the salt repo. The fix added in #44579 will be merged forward to 2017.7, oxygen, and develop to be included in upcoming releases, respectively.



roaldnefs commented on Dec 14, 2017

Author

Contributor

+ 🗨️ ...

@chrispetsos in the meantime, you could include the cron module and state in your `_modules` and `_states` folder. See [Writing Execution Modules](#) for more information.



1



1



chrispetsos commented on Dec 14, 2017

+ 🗨️ ...

So, @roaldnefs you mean I could just copy the respective module and state file from the 2016.11 version into my 2017 master to the folders you mention right? I could just be risking those files having changed between those releases, thus having any undesirable side-effects... Perhaps I'll give it a try...



roaldnefs commented on Dec 14, 2017

Author

Contributor

+ 🗨️ ...

@chrispetsos yes, it is not the most ideal solution. You will have to manually remove the state and module after a new release, but it does ensure that you can use the fix right now.



1

Scenario #1: the bugfix is accepted



garethgreenaway approved these changes on Dec 7, 2017

[View changes](#)



garethgreenaway merged commit **bb1f8dc** into `saltstack:2016.11`

[View details](#)

[Revert](#)

on Dec 7, 2017

2 of 6 checks passed



rallytime referenced this pull request on Dec 14, 2017

Identifier not working in salt.states.cron when special is used #44530

 **Closed**



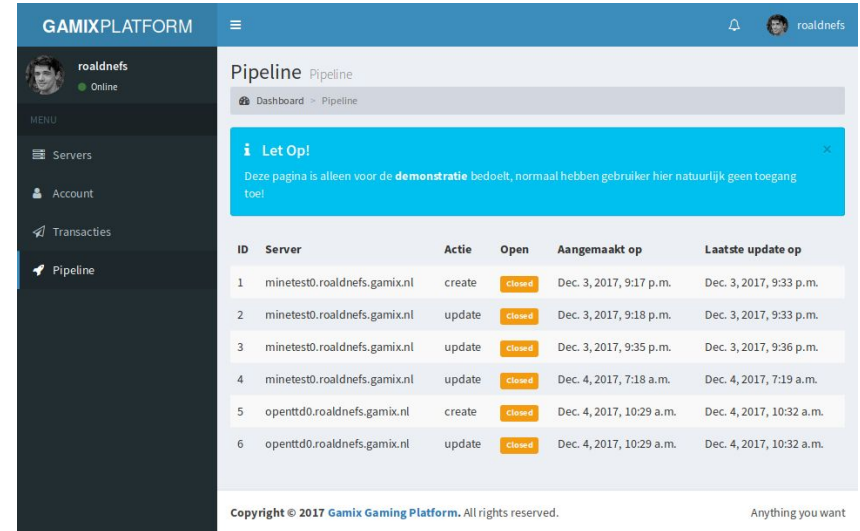
Scenario #2

Writing a `telegram` module

Scenario #2: the school assignment

Assignment: develop a web application where clients can order game servers that can automatically be provisioned.

Solution: write a web application using the Django framework and use salt-cloud and the salt-api to create and delete servers at DigitalOcean. Keep the 'maintainers' updated via **Telegram** (ChatOps).



The screenshot displays the 'GAMIXPLATFORM' interface. On the left is a dark sidebar with a user profile for 'roaldnefs' (Online) and a menu containing 'Servers', 'Account', 'Transacties', and 'Pipeline' (which is highlighted). The main content area is titled 'Pipeline Pipeline' and includes a breadcrumb 'Dashboard > Pipeline'. A blue informational banner reads: 'Let Op! Deze pagina is alleen voor de demonstratie bedoeld, normaal hebben gebruikers hier natuurlijk geen toegang tot!'. Below this is a table with the following data:

ID	Server	Actie	Open	Aangemaakt op	Laatste update op
1	minetest0.roaldnefs.gamix.nl	create	Closed	Dec. 3, 2017, 9:17 p.m.	Dec. 3, 2017, 9:33 p.m.
2	minetest0.roaldnefs.gamix.nl	update	Closed	Dec. 3, 2017, 9:18 p.m.	Dec. 3, 2017, 9:33 p.m.
3	minetest0.roaldnefs.gamix.nl	update	Closed	Dec. 3, 2017, 9:35 p.m.	Dec. 3, 2017, 9:36 p.m.
4	minetest0.roaldnefs.gamix.nl	update	Closed	Dec. 4, 2017, 7:18 a.m.	Dec. 4, 2017, 7:19 a.m.
5	openttd0.roaldnefs.gamix.nl	create	Closed	Dec. 4, 2017, 10:29 a.m.	Dec. 4, 2017, 10:32 a.m.
6	openttd0.roaldnefs.gamix.nl	update	Closed	Dec. 4, 2017, 10:29 a.m.	Dec. 4, 2017, 10:32 a.m.

At the bottom of the interface, the footer text reads: 'Copyright © 2017 Gamix Gaming Platform. All rights reserved.' and 'Anything you want'.

Scenario #2: send a message using Python

Telegram allows any user to create a new bot by talking to the @BotFather, when requesting a bot you will receive a new token.

Each Telegram chat has a unique chat ID.

Using the Telegram Bot API you can send a message to an existing chat using a simple HTTP POST request.



```
import requests

# The Telegram chat ID, token and message we would like to send
message = 'Hello Telegram!'
chat_id = '<REDACTED>'
token = '<REDACTED>'

# Telegram API url
url = 'https://api.telegram.org/bot{0}/sendMessage'.format(token)

# Make the HTTP POST request to the Telegram API
response = requests.post(url, data={'chat_id': chat_id, 'text': message})

# Check if the Telegram API returned successfully
result = response.json().get('ok', False)
```

Scenario #2: writing the `telegram` module

Import the required library `requests`, for making the HTTP requests.

The module will be called `telegram`, because the `__virtualname__` is set to `telegram`.

Only import the `telegram` module if the required libraries are installed using the `__virtual__()` function.

```
...

try:
    import requests
    HAS_REQUESTS = True
except ImportError:
    HAS_REQUESTS = False

__virtualname__ = 'telegram'

def __virtual__():
    """
    Return virtual name of the module.

    :return: The virtual name of the module.
    """
    if not HAS_REQUESTS:
        return False
    return __virtualname__

...
```

Scenario #2: telegram.post_message function

Within the Telegram module we will write the `post_message()` function.

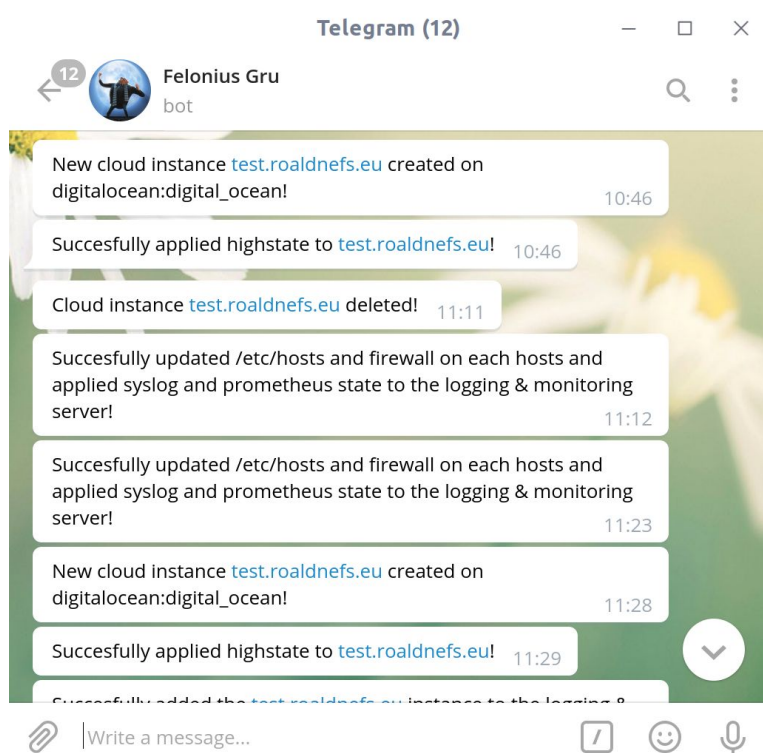
This function will be called when the user executes:

```
salt '*' telegram.post_message \  
    message='Helo Telegram' \  
chat_id='<REDACTED>' token='<REDACTED>'
```

```
def post_message(message, chat_id, token):  
    url = 'https://api.telegram.org/bot{0}/sendMessage'.format(token)  
  
    parameters = dict()  
    if chat_id:  
        parameters['chat_id'] = chat_id  
    if message:  
        parameters['text'] = message  
  
    try:  
        response = requests.post(url, data=parameters)  
        result = response.json()  
        log.debug('Raw response of the telegram request is {0}'.format(response))  
  
    except Exception:  
        log.exception('Sending telegram api request failed')  
        return False  
  
    # Check if the Telegram Bot API returned successfully.  
    if not result.get('ok', False):  
        log.debug(  
            'Sending telegram api request failed due to error {0} ({1})'.format(  
                result.get('error_code'), result.get('description')  
            )  
        )  
        return False  
    return True
```


Scenario #2: example of the **telegram** module

Example of the Telegram module and returner.





Scenario #2: telegram in SaltStack!

The `telegram` module and returner are now part of SaltStack, so you can use it to!

The documentation is automatically generated from the Python docstrings.

SALTSTACK

OverviewTutorialsDocumentation

[Edit on GitHub](#)[Table of Contents](#)[Glossary](#)[previous](#)[next](#)[all salt modules](#)[Index](#)

SALT.MODULES.TELEGRAM

Module for sending messages via Telegram.

configuration

In order to send a message via the Telegram, certain configuration is required in `/etc/salt/minion` on the relevant minions or in the pillar. Some sample configs might look like:

```
telegram.chat_id: '123456789'
telegram.token: '00000000:xxxxxxxxxxxxxxxxxxxxxxxxxx'
```

salt.modules.telegram.post_message (*message, chat_id=None, token=None*)

Send a message to a Telegram chat.

Parameters

- `message` -- The message to send to the Telegram chat.
- `chat_id` -- (optional) The Telegram chat id.
- `token` -- (optional) The Telegram API token.

Returns

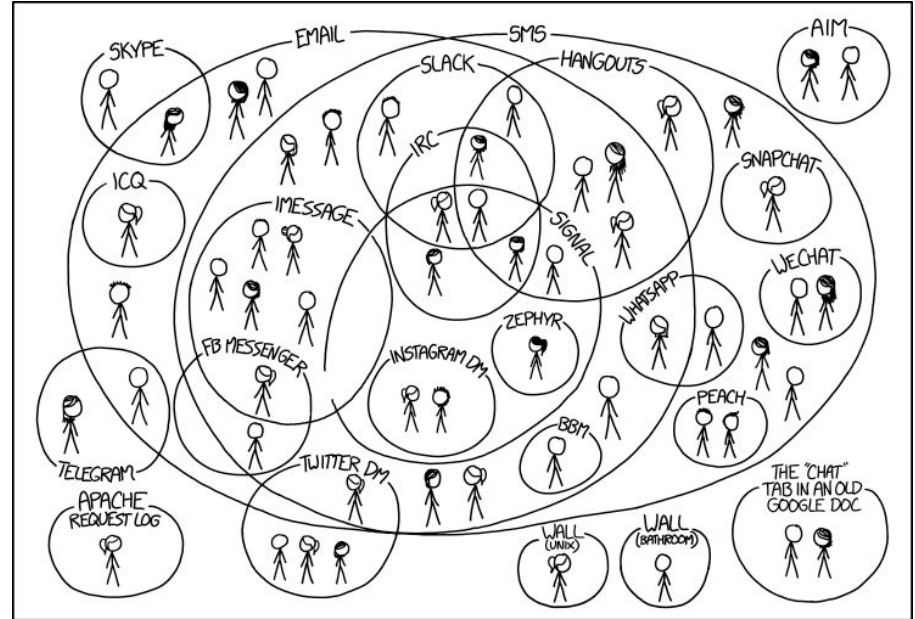
Boolean if message was sent successfully.

CLI Example:

```
salt '*' telegram.post_message message="Hello Telegram!"
```

Contact

Roald Nefs: info@roaldnefs.com



I HAVE A HARD TIME KEEPING TRACK OF WHICH CONTACTS USE WHICH CHAT SYSTEMS.

Source: <https://xkcd.com/1810/>