Sultan Documentation

Release 0.6.3

Aeroxis, LLC

Sep 12, 2017

Command and Rule Over Your Shell

CHAPTER 1

Installing Sultan

Sultan is simple and lightweight. To install Sultan, simply run the following in your command line:

```
pip install --upgrade sultan
```

CHAPTER 2

Frequently Asked Questions

What is Sultan?

Sultan allows you to interface with command-line utilities from Python without having to write your scripts in Bash.

Why use Sultan?

Leverage the Power of Bash from Python

Bash, while it seems arcaine, actually is quiet powerful! Creating a Tar Archive of a directory requires us to read up on the tar API or check StackOverflow, but if you know how to do it in Bash, you'd be done before you finish reading the StackOverflow post!

This promotes simplicity, and follows the KISS principle (Keep It Simple Stupid!)

Better Syntax:

Bash's syntax for loops, conditionals, and functions work well, but require a lot of nuances that we're just not used to with a modern language like Python.

Sultan allows you to use Python Syntax, and never touch Bash's arcaine syntax ever again!

Project Management:

Sultan was designed because Bash just does not scale well with most projects. As much as we can try, Bash does not have a proper package management system. So, we're left with script files that are sourced, and called, but it gets quiet complicated as soon as you have more than 10 scripts.

Python has a great package and module system that allows us to create complex projects using the Python language, but also leverage a lot of great tools and functionality that we've grown to love and expect from Bash.

This promotes reusability, with the DRY (Don't Repeat Yourself) principle. If you create a great solution with Sultan, publish it on **PyPi** and others will use it.

Unit Testing

Out of the Box, Bash does not come with any Unittesting frameworks, but Python does! You can build unittests and integration tests with Sultan to ensure your code does *EXACTLY* what you want it to do.

Why can't I use *subprocess*?

Python's standard library offers the subprocess library, but it isn't very "Pythonic". The 'subprocess' module has a bunch of methods for writing commands to the shell, but the code is overly verbose, and tough to read.

Any reason to use this over ansible or saltstack?

Sultan is just a simpler interface to command line utilities. It helps bypass the arcane language constructs of Bash (among other things. See **Why use Sultan?** above).

Sultan was created to help with scripts that we create with Bash, that tend to get complex. When these scripts get complex, Bash just gets to be a pain to deal with, since it lacks proper package management, it lacks unit testing, and <insert library that you need for managing complex scripts>. So Sultan allows scripts to be reusable and tested with standard Python.

Ansible and Salt are powerful for provisioning a system. Sultan can't compete in that realm, but it does help with complex scripts. Even if you want Ansible or Salt to perform something on a remote box, like installing a package, it requires some overhead in setting them up. Sultan is simple with no external dependencies, and installs itself with just "pip install sultan".

Sultan simply wraps the subprocess module in Python's standard library, but it also provides a nice to read logging system, and provides you with relevant information when a command fails.

All in all, it can't compete with standard DevOps tools used for provisioning. It does help with not having to use Bash heavily, if you're a Python programmer.

CHAPTER 3

Sultan Examples

This tutorial will go through various examples to help in better understanding how to use Sultan. Each example will build on the lessons learned from the previous examples.

WARNING * WARNING * WARNING

When you're using Sultan, you are running commands directly on your local shell, so please, do not run untested and untrusted code. You are taking the risk if you are running untrusted code.

Sultan runs *POpen* with *shell=True*, and according to Python documentation, this can be a security hazard if combined with untrusted input. More information can be found here:

- Python 2: https://docs.python.org/2/library/subprocess.html#frequently-used-arguments
- Python 3: https://docs.python.org/3/library/subprocess.html#frequently-used-arguments

Example 1: Getting Started

We typically use *yum* or *apt-get* to install a package on our system. This example installs a package on our system using Sultan. Here is how to get started:

```python
from sultan.api import Sultan

s = Sultan()
s.yum("install", "-y", "tree").run()
```

Sultan allows multiple syntaxes depending on what your first command is. Suppose you want to not use separate tokens, and instead you want to use one string, you can write the above example as such:

```python
from sultan.api import Sultan

s = Sultan()
s.yum("install -y tree").run()
```

Suppose your user is not a root-user, and you want to call to sudo to install the *tree* package. You'd do the following:

```python
from sultan.api import Sultan

with Sultan.load(sudo=True) as s:
    s.yum('install -y tree').run()
```

NOTE: For the sake of brevity, this tutorial will now start to assume that *Sultan* has been imported from *sultan.api* and, the variable *s* has been instantiated as an instance of *Sultan* (*s = Sultan()*). This will change in situations where the documentation requires a different usage.

Example 2: Sultan with Context Management

There are times when we want to manage the context of where Sultan executes your code. To aid with this, we use Sultan in Context Management mode.

Suppose we want to cat out the contents of */etc/hosts*, we'd do the following:

```
with Sultan.load(cwd="/etc") as s:
    s.cat("hosts").run()
```

Example 3: Compounding with And (&&) and Or (||)

There are times when we need multiple commands to run at once. We use the *and_()* command to get through this. Here is an example:

```
# runs: 'cd /tmp && touch foobar.txt'
with Sultan.load() as s:
    s.cd("/tmp").and_().touch("foobar.txt").run()
```

There are also times that we want to run 2 commands, but run the 2nd command even if the first command fails. For this, you will need to use the *or_()* command. Here is an example:

```
# runs: 'mkdir /tmp || mkdir /bar'
with Sultan.load() as s:
    s.mkdir('/tmp').or_().mkdir('/bar').run()
```

Example 4: Redirecting with Pipes (|)

In Bash, we use the pipe | operator to redirect the output of the call to a command to another command. We do this in Sultan with the *pipe* command. Here is an example:

```
# runs: 'ls -l | sed -e "s/[aeio]/u/g"'
with Sultan.load() as s:
    s.ls('-l').pipe().sed('-e', '"s/[aeio]/u/g"').run()
```

Example 5: Redirecting Output to File

In Bash, we often want to redirect the output of a command to file. Whether the output is in *stdout* or *stderr*, we can redirect it to a file with Sultan. Here is an example:

```
# runs: 'cat /etc/hosts > ~/hosts'
with Sultan.load() as s:
    s.cat("/etc/hosts").redirect(
        "~/hosts",
        append=False,
        stdout=True,
        stderr=False).run()
```

In the example above, we redirected the output of */etc/hosts* to *~/hosts*. We only outputted the *stdout*, and didn't append to the file if it existed. Feel free to customize this method as it fits your needs.

Example 6: Read from Standard Input

Python has the *raw_input* built-in to read from standard input. Sultan's API wraps around *raw_input* to ask the user for their input from the command line and returns the value.

Here is the example:

```
name = s.stdin("What is your name?")
print "Hello %s" % name
```

Example 7: Running as Another User

Sultan can run commands as another user. You need to enable *sudo* mode to do this.

Here is an example:

```
# runs: sudo su - hodor -c 'cd /home/hodor && ls -lah .;'
with Sultan.load(sudo=True, user='hodor', cwd='/home/hodor') as s:
    sultan.ls('-lah', '.').run()
```

Example 8: Running as Root

Sultan can run commands as the *root* user. You need to only enable *sudo* mode to do this.

Here is an example:

```
# runs: sudo su - root -c 'ls -lah /root;'
with Sultan.load(sudo=True) as sultan:
    sultan.ls('-lah', '/root').run()
```

Example 9: Disable Logging

If you need to disable logging all together, simply add set 'logging' to False while loading Sultan with Context.

Here is an example:

```
# runs without logging
with Sultan.load(logging=False) as sultan:
    sultan.ls('-lah', '/tmp').run()
```

Example 10: Commands with Hyphones (i.e.: apt-get)

There are commands that are available in the shell that use hyphens which conflict with the function naming conventions of Python, for example **apt-get**. To get around this, use double underscores (__).

Here is an example:

```
with Sultan.load(sudo=True) as sultan:
    sultan.apt__get('install', 'httpd').run()
```

which runs:

```
sudo apt-get install httpd;
```

Example 11: Sourcing a File before Running a Command

This is rare, but there are times that we would like to source a file before running a command. We can manually do this with the Bash And Operator (&&) but Sultan has the ability to do this automatically for you.

Here is an example:

```
with Sultan.load(
        cwd='/home/davydany/projects/sultan',
        src='/home/davydany/.virtualenv/sultan/bin/activate') as s:
    s.pip('install', '-r', 'requirements.txt').run()
```

which runs:

```
source /home/davydany/.virtualenv/sultan/bin/activate && cd /home/davydany/projects/
↪sultan && pip install -r requirements.txt;
```

Example 12: Results from a Command

When you run a command, your shell gives back results in stdout and stderr. Sultan returns a Result object which has **stdout**, **stderr** and **traceback** object.

Here is an example that shows how to get the results of a command:

with Sultan.load() as s:

> result = s.yum('install', '-y', 'postgresql') result.stdout # the stdout result.stderr # the stderr result.traceback # the traceback

stdout and **stderr** returns a list, where each element is a line from **stdout** and **stderr**.

Most times, you don't need to access the results of a command, but there are times that you need to do so. For that, the **Result** object will be how you access it.

CHAPTER 4

Sultan SSH Examples

This tutorial will go through various examples to help in better understanding how to use Sultan over SSH. Each example will build on the lessons learned from the previous examples.

WARNING * WARNING * WARNING

When you're using Sultan, you are running commands directly on your local shell, so please, do not run untested and untrusted code. You are taking the risk if you are running untrusted code.

Sultan runs *POpen* with *shell=True*, and according to Python documentation, this can be a security hazard if combined with untrusted input. More information can be found here:

- Python 2: https://docs.python.org/2/library/subprocess.html#frequently-used-arguments
- Python 3: https://docs.python.org/3/library/subprocess.html#frequently-used-arguments

Example 1: SSH to Remote Host as the Current User

By default, you can simply specify the host to sultan, and calling the commands like you normally do. This uses the username of the user who is executing the script, and connects you to the remote host.

```
from sultan.api import Sultan

with Sultan.load(hostname='aeroxis.com') as sultan:
    s.yum('install', '-y', 'tree').run()
```

Sultan will connect to the remote host, and run *yum install -y tree*. This is what is passed to your shell to execute the command (assuming your username is *davydany*):

```
ssh davydany@aeroxis.com 'yum install -y tree;'
```

Example 2: SSH to Remote Host as a Different User

You can specify a different user to execute the remote commands by using the *user* parameter, like this:

```
with Sultan.load(user='elon.musk', hostname='aeroxis.com') as s:
    s.yum('install', '-y', 'tree').run()
```

And this will execute:

```
ssh elon.musk@aeroxis.com 'yum install -y tree;'
```

Example 3: Passing Additional Options (Port)

Added in v0.6

If you need to pass additional options for the port, use the *SSHConfig* class to configure the SSH Connection.:

```
from sultan.api import Sultan, SSHConfig

port = 2222
config = SSHConfig(port=port)
with Sultan.load(user='elon.musk',
                 hostname='aeroxis.com',
                 ssh_config=config) as s:
    s.yum('install', '-y', 'tree').run()
```

which will yield:

```
ssh -p 2222 elon.musk@aeroxis.com 'yum install -y tree;'
```

Example 4: Passing Additional Options (Identity File)

Added in v0.6

If you need to pass additional options for the port, use the *SSHConfig* class to configure the SSH Connection.:

```
from sultan.api import Sultan, SSHConfig

path_to_identity_file = '/home/elon.musk/keys/elon.musk.identity'
config = SSHConfig(identity_file=path_to_identity_file)
with Sultan.load(user='elon.musk',
                    hostname='aeroxis.com',
                    ssh_config=config) as s:
    s.yum('install', '-y', 'tree').run()
```

which will yield:

```
ssh -i /home/elon.musk/keys/elon.musk.identity elon.musk@aeroxis.com 'yum install -y␣
↪tree;'
```

CHAPTER 5

What is Sultan?

Sultan is a Python package for interfacing with command-line utilities, like *yum*, *apt-get*, or *ls*, in a Pythonic manner. It lets you run command-line utilities using simple function calls.

The simplest way to use Sultan is to just call it:

```
from sultan.api import Sultan
s = Sultan()
s.sudo("yum install -y tree").run()
```

Runs:

```
sudo yum install -y tree;
```

The recommended way of using Sultan is to use it in Context Management mode. Here is how to use Sultan with Context Management:

```
from sultan.api import Sultan

with Sultan.load(sudo=True) as s:
  s.yum("install -y tree").run()
```

Runs:

```
sudo su - root -c 'yum install -y tree;'
```

What if we want to install this command on a remote machine? You can easily achieve this using context management:

```
from sultan.api import Sultan

with Sultan.load(sudo=True, hostname="myserver.com") as sultan:
  sultan.yum("install -y tree").run()
```

Runs:

```
ssh root@myserver.com 'sudo su - root -c 'yum install -y tree;''
```

If you enter a wrong command, Sultan will print out details you need to debug and find the problem quickly.

Here, the same command was run on a Mac:

```
from sultan.api import Sultan

```

```
with Sultan.load(sudo=True, hostname="myserver.com") as sultan:
  sultan.yum("install -y tree").run()
```

Yields:

```
[sultan]: sudo su - root -c 'yum install -y tree;'
Password:
[sultan]: --{ STDERR }-----------------------------------------------------------
↪---------------------------------------
[sultan]: | -sh: yum: command not found
[sultan]: ----------------------------------------------------------------------------
↪---------------------------------------
```

Want to get started? Simply install Sultan, and start writing your clean code:

```
pip install --upgrade sultan
```

If you have more questions, check the rest of the docs, or reach out at Github: https://github.com/aeroxis/sultan

WARNING * WARNING * WARNING

When you're using Sultan, you are running commands directly on your local shell, so please, do not run untested and untrusted code. You are taking the risk if you are running untrusted code.

Sultan runs *POpen* with *shell=True*, and according to Python documentation, this can be a security hazard if combined with untrusted input. More information can be found here:

- Python 2: https://docs.python.org/2/library/subprocess.html#frequently-used-arguments
- Python 3: https://docs.python.org/3/library/subprocess.html#frequently-used-arguments