Sultan Documentation

Release 0.3.1

Aeroxis, LLC

May 09, 2017

Table of Contents

Command and Rule over your Shell 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.

Here is how you'd use Sultan:

```
from sultan.api import Sultan

# simple way
s = Sultan()
s.sudo("yum install -y tree").run()

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

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

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

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:

```
In [1]: with Sultan.load(sudo=True) as s:
  ...:     s.yum("install -y tree").run()
  ...:
[sultan]: sudo su - root -c 'yum install -y tree;'
Password:
[sultan]: Unable to run 'sudo su - root -c 'yum install -y tree;''
[sultan]: --{ TRACEBACK }----------------------------------------------------------------
↪---------------------------------------
[sultan]: | Traceback (most recent call last):
[sultan]: |   File "/Users/davydany/projects/aeroxis/sultan/src/sultan/api.py", line␣
↪159, in run
[sultan]: |     stdout = subprocess.check_output(commands, shell=True, stderr=stderr)
[sultan]: |   File "/System/Library/Frameworks/Python.framework/Versions/2.7/lib/
↪python2.7/subprocess.py", line 573, in check_output
```

```
[sultan]: |     raise CalledProcessError(retcode, cmd, output=output)
[sultan]: | CalledProcessError: Command 'sudo su - root -c 'yum install -y tree;''␣
↪returned non-zero exit status 127
[sultan]: ------------------------------------------------------------------------------
↪--------------------------------------
[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 docs! http://sultan.readthedocs.io/en/latest/

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 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.

Sultan is made 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 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

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.

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:

```
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:

```
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:

```
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 (&&)

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'
s.cd("/tmp").and_().touch("foobar.txt").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"'
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'
s.cat("/etc/hosts").redirect(
    "~/hosts",
    append=False,
    stdout=True,
    stderr=False)
```

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', '.')
```

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')
```

- genindex
- modindex
- search

S

S