



This section will discuss interaction with the OCI command group, new to Singularity 3.1.0. If you are using versions of Singularity before this, you won't be able to use these functions. If you want to learn more about OCI, see opencontainers.org.

- [background](#): some quick background on the OCI runtime spec
- [setup](#): create a bundle folder with a config.json to drive it
- [quick start](#): Quick start to create an OCI image object
- [create](#): create an OCI image using the client

The commands for state, kill, pause, resume, etc. will be discussed in context of the above.

Background

To give you some quick background, the [runtime-spec](#) of the open container initiative (OCI), you'll see that it defines a way to create a container from something called a "bundle"

What is a bundle?



A bundle is just a folder on your computer with the contents of an operating system, libraries, and software, along with a configuration file (config.json). The configuration file conforms to this "runtime specification," and generally describes permissions, binds,

Why do we have this specification?

It might seem silly, but what this runtime specification is a standard set of terms for generating and interacting with containers. What does this mean in practice? We can have tools and infrastructure (for example, Kubernetes, a container orchestration system) that know how to interact with many different kinds of containers. How? Because they implement the OCI specification, and the communication is standardized.

Usage

Install

If you haven't yet installed Singularity Python:

```
$ pip install spython
```

or see [the install docs](#) for different variants of that.

Setup

Now that you are familiar with the idea of a bundle, let's:

- create a folder with an operating system
- add a config.json template there
- create an OCI Image with the Singularity Python client

one loaded with a client.

```
spython shell
```

Once in ipython, let's use Singularity build with sandbox to dump a busybox base operating system into a temporary folder. Singularity Python also will provide you with a dummy (limited) configuration file) to use to test.

```
from spython.utils import get_installdir

# Here is the dummy config.json for you!
config = "%s/oci/config.json" % get_installdir()

# Let's now build a bundle into /tmp/bundle
image = client.build("docker://busybox:1.30.1",
                    image='/tmp/bundle',
                    sandbox=True,
                    sudo=False)
```

If you need to import the client on your own (if you don't use spython shell)

```
from spython.main import Client as client
```

Next, copy the config to your bundle folder.

```
import shutil
shutil.copyfile(config, '%s/config.json' % image)
```

Now that you have your bundle and config.json file, you can create an Oci Image from it.



The easiest way to create a running Oci Image from a bundle is to instantiate an OciImage instance. If you didn't import the client, do this:

```
from spython.main import Client as client
```

Here is the OciImage object to instantiate! Notice that we are providing the full path to the bundle folder, along with an id for our container.

```
$ image = client.oci.OciImage(bundle='/tmp/bundle',
                             container_id='figbars')
[singularity-python-oci:figbars]
```

The variable "image" now holds a complete OciImage class (not attached to the client) and with "mycontainer" set as the container_id. If you are in ipython and press TAB, you will see all the expected functions.

| | | | |
|--------------|--------------------|--------------------|---------------|
| attach() | execute() | mount() | resume() |
| container_id | get_container_id() | OciImage | RobotNamer |
| create() | get_uri() | parse_image_name() | run() |
| delete() | kill() | remove_uri() | run_command() |

It's created off the bat! And further, the container id you defined is stored with the object, so you don't need to ask for it again.

```
image.state()

{'attachSocket': '/var/run/singularity/instances/root/figbars/attach.sock',
 'bundle': '/tmp/bundle',
 'controlSocket': '/var/run/singularity/instances/root/figbars/control.sock',
```

```
ociVersion': '1.0.1-dev',  
  ': 20854,  
  tus': 'created'}
```

Here is the stored container id:

```
image.container_id  
'figbars'
```

And we can see that the default is that sudo is used to create it:

```
image.sudo  
True
```

By default, if you don't provide a bundle directory it won't be created.

```
$ image = client.oci.OciImage(container_id='figbars')
```

This would be a good way to get a handle for an (already created) image, or something you are otherwise not ready to create. You can also explicitly tell the function not to create the image.

```
$ image = client.oci.OciImage(bundle='/tmp/bundle',  
                               container_id='figbars',  
                               create=False)
```

Press TAB after typing “image.” to see the options:

```
in [39]: image.  
  attach()          debug          get_uri()          parse_image_name()  
  bundle            delete()         kill()            quiet  
  container_id      execute()       mount()          remove_uri()      >  
  create()          get_container_id() OciImage         resume()
```

Give everything a test! For example, try executing a command to your image:

```
result = image.execute(command=["ls", "/"])  
print(result)  
bin  
boot  
cdrom  
dev  
etc  
home  
initrd.img  
initrd.img.old  
lib  
lib32  
lib64  
lost+found  
media  
mnt  
opt  
proc  
root  
run  
sbin  
srv  
sys
```

```
uz
vmlinuz.old
```

You can then clean up.

```
image.pause()
# or
image.kill()
image.delete()
```

Create

Let's discuss another option for create - one that also starts from our bundle folder, but instead just uses the client to interact with it (without directly creating an instance). Here is how to create it:

```
metadata = client.oci.create(bundle=image, container_id='robot-man')
```

As of 3.1.0, the command above does require sudo, so it will prompt you for it unless you already have effective user id as 0. The return of the above will be a json object that describes metadata for the container, the same that we saw above when we asked for the `image.state()`.

```
{'attachSocket': '/var/run/singularity/instances/root/robot-man/attach.sock',
 'bundle': '/tmp/bundle',
 'controlSocket': '/var/run/singularity/instances/root/robot-man/control.sock',
 'createdAt': 1551744049459537512,
 'id': 'robot-man',
```

```
'status': 'created'}
```

Notice that the status is “created.” If you wanted to get this result again with the client:

```
$ client.oci.state('robot-man', sudo=True)

{'attachSocket': '/var/run/singularity/instances/root/robot-man/attach.sock',
 'bundle': '/tmp/bundle',
 'controlSocket': '/var/run/singularity/instances/root/robot-man/control.sock',
 'createdAt': 1551744049459537512,
 'id': 'robot-man',
 'ociVersion': '1.0.1-dev',
 'pid': 20602,
 'status': 'created'}
```

important notice how we included `sudo=True` with the above - since we created the Oci Image with `sudo`, it lives in `root`'s space. If we call without `sudo`, or ask for the state of a non existing container, we will get `None` for a result.

```
$ client.oci.state('doesntexist')

# No sudo
$ client.oci.state('mycontainer')
```


Both of the above return no result (`None`).

We haven't gone through all of the examples, but if you would like a specific example please [let us know](#).



Singularity Python is maintained by [Vanessa Sochat](#).

Contribute on [GitHub](#).

The Singularity Hub