

**BUILD YOUR FIRST OPENSTACK
APPLICATION
WITH OPENSTACK PYTHONSDK**

VICTORIA MARTINEZ DE LA CRUZ

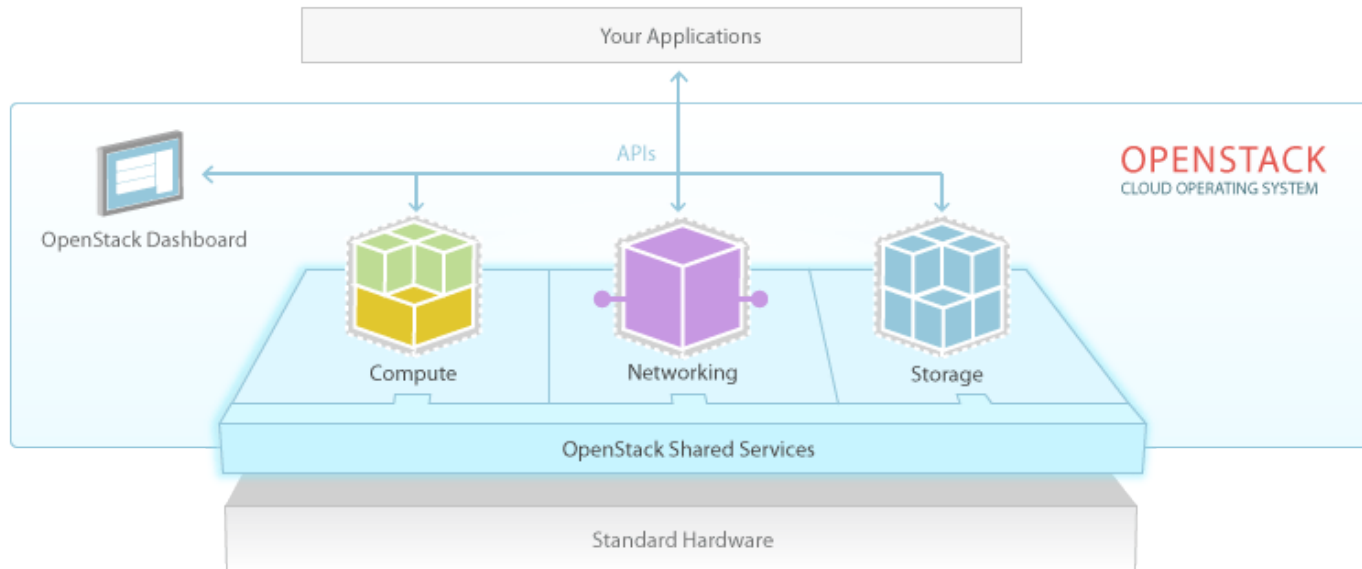
SOFTWARE ENGINEER AT RED HAT

CO-FOUNDER LINUXCHIX ARGENTINA


WHAT IS OPENSTACK?

BRIEF OVERVIEW


OPENSTACK OVERVIEW



IAAS... AND MORE!



SWIFT
Object Storage




KEYSTONE
Identity




NOVA
Compute



NEUTRON
Networking



CINDER
Block Storage



GLANCE
Image Service

+




HORIZON
Dashboard



CEILOMETER
Telemetry



HEAT
Orchestration



TROVE
Database



SAHARA
Elastic Map Reduce



IRONIC
Bare-Metal Provisioning



ZAQAR
Messaging Service



MANILA
Shared Filesystems



DESIGNATE
DNS Service



BARBICAN
Key Management



MAGNUM
Containers



MURANO
Application Catalog



CONGRESS
Governance

RUNNING APPS ON OPENSTACK

HOW IT WAS... A FEW YEARS BACK

OSD GRACE HOPPER CELEBRATION 2014



OSD GRACE HOPPER CELEBRATION 2014

- Leveraging OpenStack scalability and resiliency in times of need and disaster
- Defining a cloud-ready architecture for an standard application
- Deploying the application in no-time by just running an script



IN PREPARATION FOR OSD GHC 2014

Architecture the app to be cloud-ready → 2 webserver + 1 db +
1 load balancer



IN PREPARATION FOR OSD GHC 2014

Talk with an OpenStack cloud using python-*client

OPENSTACK PYTHON NOVA CLIENT

```
# List servers in Nova
from novaclient.v1_1 import client

conn = client.Client(user, password,
                     project, auth_url)

from server in conn.servers.list():
    print(server.name)
```

OPENSTACK PYTHON GLANCE CLIENT

```
# List images in Glance
from glanceclient.v2 import client

conn = client.Client(auth_url, token)

from image in conn.images.list():
    print(image["name"])
```

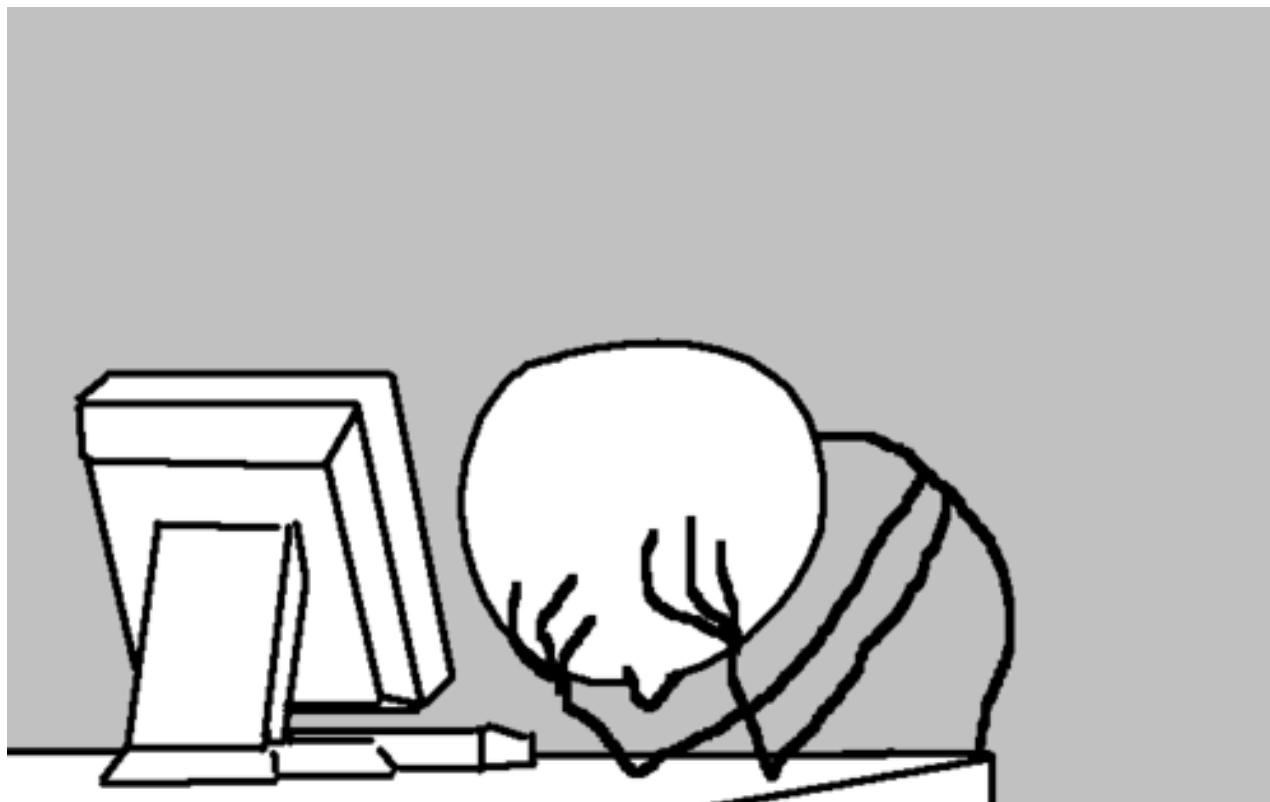
OPENSTACK PYTHON SWIFT CLIENT

```
# List containers in Swift
from swiftclient import client

conn = client.Client(auth_url, user, key,
                     tenant_name,
                     auth_version)

header, containers = conn.getaccount()

from container in containers:
    print(container["name"])
```



OPENSTACK LIBRARIES & SDKS

COMMUNITY TOOLS FOR CLOUD DEVS

WHY?

There is no way to just talk to an OpenStack cloud

- Lots of services. One lib per service, one ux per lib
 - Lots of libs * lots of ux == sad cloud dev

APACHE LIBCLOUD

- Unified API
- Talking to different clouds (lots of plugins!)
- Third party

APACHE LIBCLOUD

```
from libcloud.compute.types import Provider
from libcloud.compute.providers import get_driver

import libcloud.security

libcloud.security.VERIFY_SSL_CERT = False

OpenStack = get_driver(Provider.OPENSTACK)

driver = OpenStack('your username', 'your password',
                   ex_force_auth_url='https://nova-api.trystack.org:5443',
                   ex_force_auth_version='2.0_password')

nodes = driver.list_nodes()

images = driver.list_images()
sizes = driver.list_sizes()
size = [s for s in sizes if s.ram == 512][0]
image = [i for i in images if i.name == 'natty-server-cloudimg-amd64'][0]

node = driver.create_node(name='test node', image=image, size=size)
```

OPENSTACK SHADE

- *Simplicity*
- OpenStack Infra subproject
- Under development, it is expected to change

OPENSTACK SHADE

```
import shade

# Initialize and turn on debug logging
shade.simple_logging(debug=True)

# Initialize cloud
# Cloud configs are read with os-client-config
cloud = shade.openstack_cloud(cloud='epcloud')

# Upload an image to the cloud
image = cloud.create_image(
    'fedora24', filename='fedora24.qcow2', wait=True)

# Find a flavor with at least 512M of RAM
flavor = cloud.get_flavor_by_ram(512)

# Boot a server, wait for it to boot, and then do whatever is needed
# to get a public ip for it.
cloud.create_server(
    'my-server', image=image, flavor=flavor, wait=True, auto_ip=True)
```

OPENSTACK PYTHONSDK

- Complete set of libraries, tools, documentation and examples
- Aimed at all types of users
 - Users of OpenStack clouds (probably YOU!)
 - Operators of OpenStack clouds
 - Developers of OpenStack projects
- Install once, run anywhere

OPENSTACK PYTHONSDK

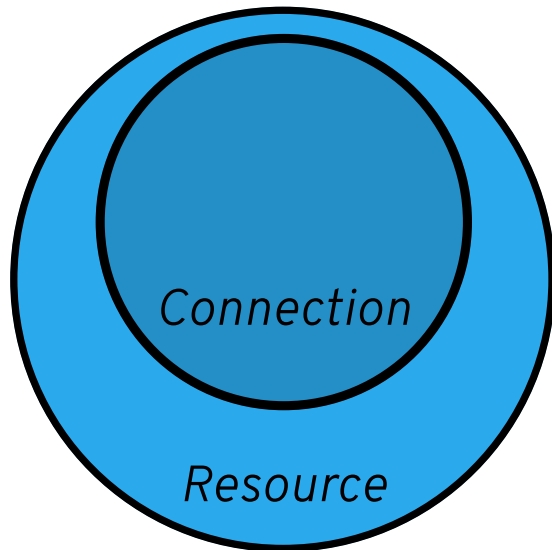
**THE COMMUNITY PYTHONSDK FOR CLOUD
DEVS**

PYTHON OPENSTACKSDK

Write Python automation scripts that create and manage resources in your OpenStack cloud

```
$ pip install openstacksdk
```

PYTHON OPENSTACKSDK



Connection

- Application developer consuming an OpenStack cloud
- Maintains your session, authentication, transport, and profile

Resource

- OpenStack developer requiring finer-grained control

SOME SNIPPETS

Establishing a connection with the cloud

[illegible]

SOME SNIPPETS

Creating a server

```
from openstack import connection

def create_server(conn):
    print("Create Server:")
    image = conn.compute.find_image(IMAGE_NAME)
    flavor = conn.compute.find_flavor(FLAVOR_NAME)
    network = conn.network.find_network(NETWORK_NAME)
    keypair = create_keypair(conn)

    server = conn.compute.create_server(
        name=SERVER_NAME, image_id=image.id, flavor_id=flavor.id,
        key_name=keypair.name, user_data=CLOUD_INIT)

    server = conn.compute.wait_for_server(server)
```

SOME SNIPPETS

Creating a keypair

```
import os

from openstack import connection

def create_keypair(conn):
    keypair = conn.compute.find_keypair(KEYPAIR_NAME)

    if not keypair:
        print("Create Key Pair:")

        keypair = conn.compute.create_keypair(name=KEYPAIR_NAME)

        try:
            os.mkdir(SSH_DIR)
        except OSError as e:
            if e.errno != errno.EEXIST:
                raise e

        with open(PRIVATE_KEYPAIR_FILE, 'w') as f:
            f.write("%s" % keypair.private_key)

        os.chmod(PRIVATE_KEYPAIR_FILE, 0o400)

    return keypair
```

MAKING YOUR APP CLOUD- READY

SOME GROUND RULES

CLOUD READY & CLOUD CENTRIC

Common classification

- Cloud ready: Effectively deployed into either a public or private cloud
- Cloud centric: Built using different tools and runtimes than traditional applications.

DYNAMIC APPLICATION TOPOLOGY

If the topology can change, it will change

- Deploy your application to be as generic and stateless as possible. This will allow to:
 - Selectively scale individual components
 - Simplify maintenance and reuse
 - Fault tolerance
- E.g. Don't hardcore information about networking, delegate it to the networking service

EPHEMERAL STORAGE

Don't assume the local file system is permanent

- Use a remote storage for non-static data
 - Cache
 - Logs
- E.g. You can use the block storage service volumes to store data

STATELESS

Statefulness of any sort limits the scalability of an application

- Remove or, if needed, store the session state in a HA store external to your app server (cache or database)
- E.g. You can use the databases service to spin up a DB instance

STANDARDS

Use standards-based services and APIs for portability to cloud environments

- Avoid using obscure protocols
- Don't rely on OS-specific features

AUTOMATION

Cloud apps need to be installed frequently and on-demand

- Automate configuration setup
- Minimize the dependencies required by the application installation

THX!

Q&A

`vkmc@redhat.com`

`vkmc` at Twitter

`vkmc` at `irc.freenode.org`

OpenStack cloud native deployment for
application developers

D. Flanders

Thursday, 2pm at Room E

OpenStack Open Space

Wednesday, TBD

USEFUL RESOURCES

- <http://developer.openstack.org/firstapp-libcloud/>
- <http://developer.openstack.org/sdks/python/>
- http://docs.openstack.org/user-guide/sdk_overview.html