

How To Leave Rackspace

Ed Leafe
Rackspace
GlueCon 2014

Disclaimer:

I Work For Rackspace

(no, I'm not getting fired)

Lock-in

Relationship











Image courtesy of www.learnvest.co



imagecourtesy©www.ebonv.com

So just leave, right?



Emotions

Stuff

(Lots of Stuff)





Data

Average US speed:
20Mb / sec

The Right Internet Package for You

AT&T U-verse High Speed Internet provides the speed, reliability, and connectivity you deserve.

[Expand All >](#)

+ Pro - 3Mbps	\$29⁹⁵ /mo	for 12 mos. 1-yr term req'd	Check availability >
+ Elite - 6Mbps	\$34⁹⁵ /mo	for 12 mos. 1-yr term req'd	Check availability >
+ Max Plus - 18Mbps	\$44⁹⁵ /mo	for 12 mos. 1-yr term req'd	Check availability >
+ Power - 45Mbps	\$64⁹⁵ /mo	for 12 mos. 1-yr term req'd	Check availability >

Average US speed:
150MB / min

Average US speed:
~9GB / hour

Assuming 1TB of data:

It would take
4.75 days
just to download





1 Container:
2,000 Servers

2 Petabytes
(Roughly)

How long to
download that?

9,500 Days

26 Years!

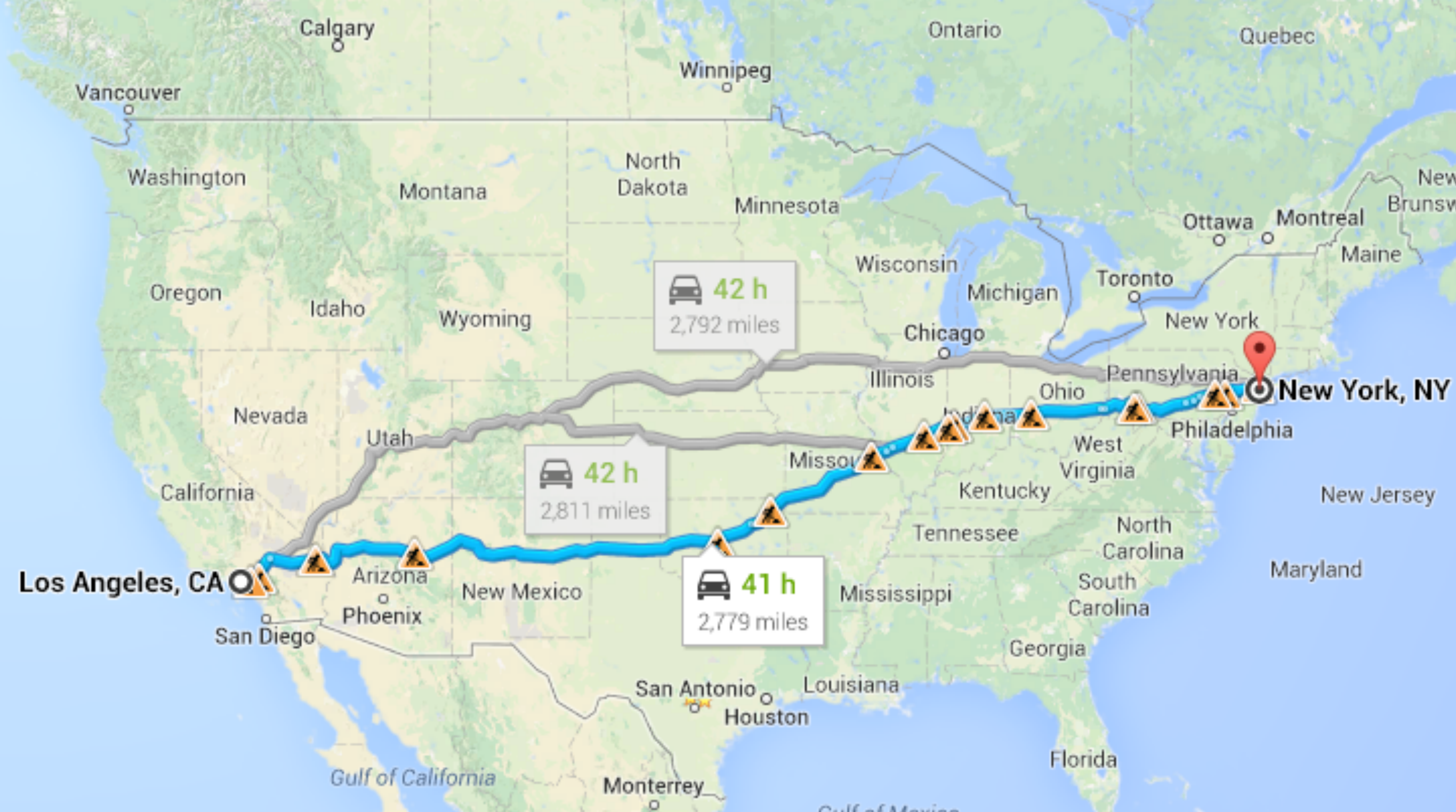
Bandwidth Charges

Rackspace: \$0.12/GB

$$1 \text{ TB} = \$120$$

$$2 \text{ PB} = \$250,000$$

Transfer:
LA to NYC



41 Hours

2800 miles

Truck Averages

6 MPG

\$4 / Gallon

Total:
\$1,900

Summary

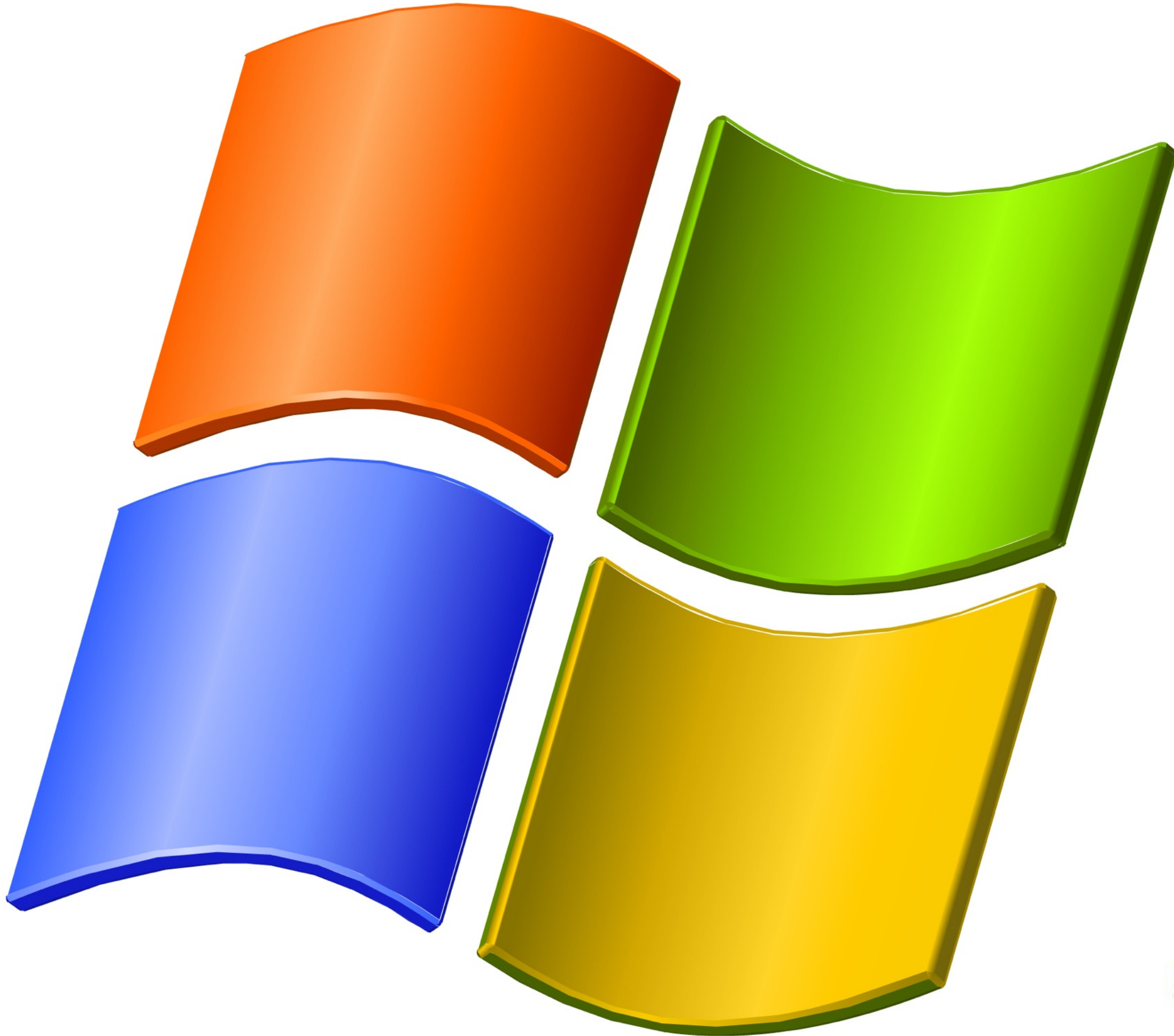
	Download	Truck
Time	26 Years	A week
Money	\$250,000	\$2,000

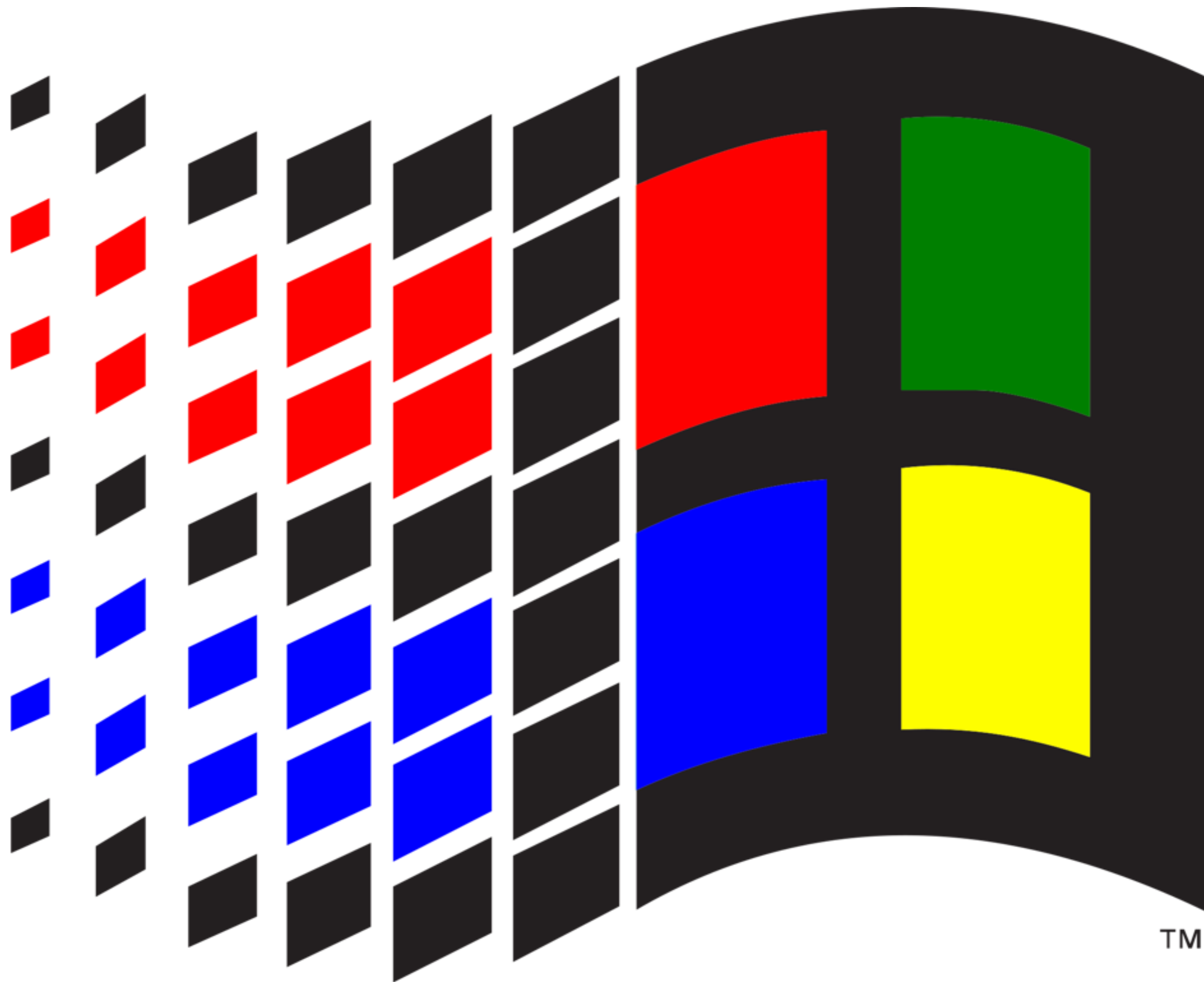
Data Gravity

APIs

Application Interoperability

A story...





TM



What You Can do
to Avoid Lock-in

Deployment Automation



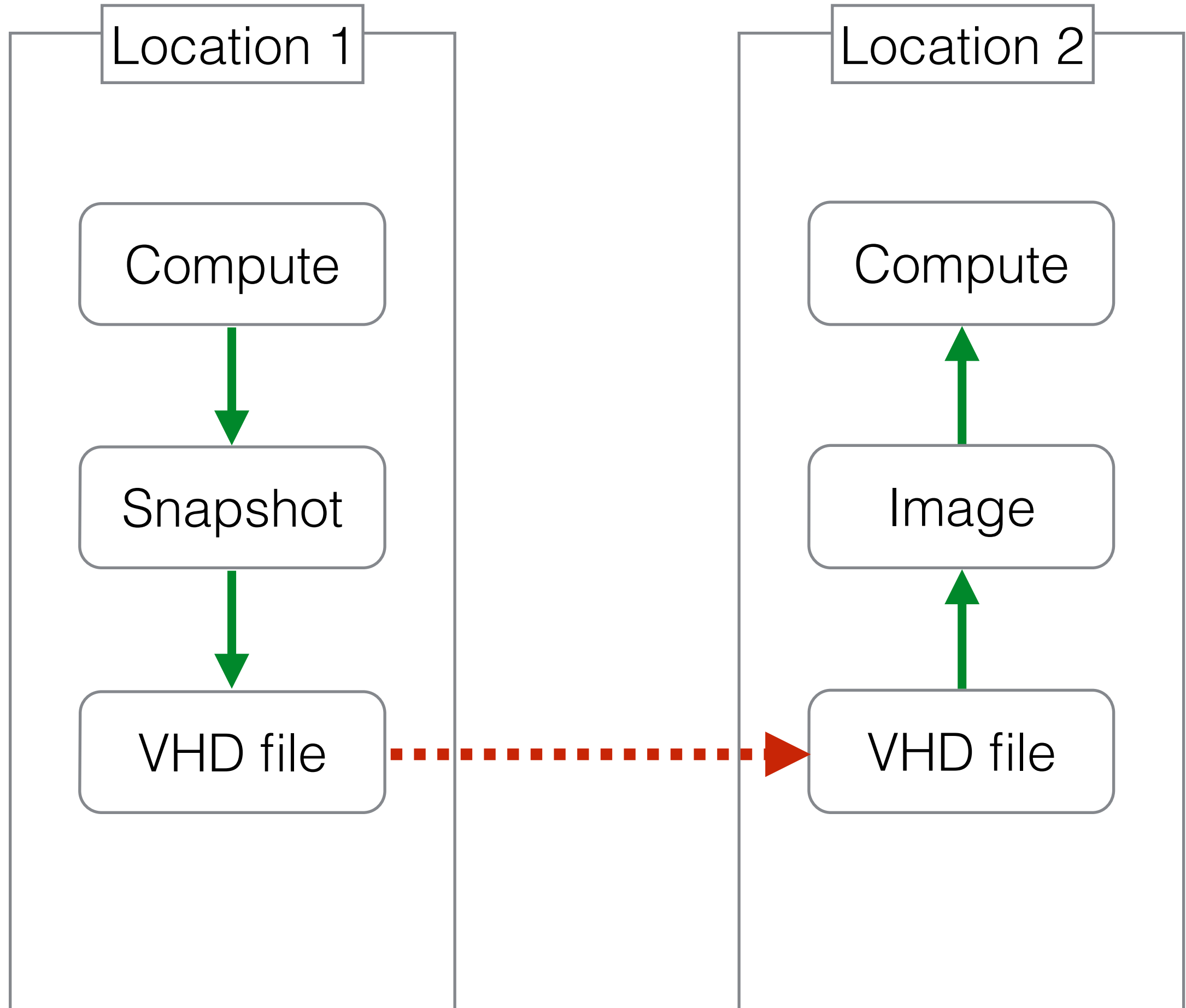
SALTSTACK

ANSIBLE



Great When You're
Getting Started

Moving Compute Resources



Code

Pyrax

Python SDK for OpenStack Clouds

Setting Up

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import print_function

import datetime
import Queue
import threading
import time

import pyrax
import pyrax.exceptions as exc
import pyrax.utils as utils
```


Create Contexts

```
rs = pyrax.create_context(env="rackspace")  
rs.keyring_auth()
```

```
hp = pyrax.create_context(env="hp")  
hp.keyring_auth()
```

Create Clients

```
rs_region = "ORD"  
hp_region = "region-a.geo-1"
```

```
rs_compute = rs.get_client("compute", rs_region)  
rs_obj = rs.get_client("object_store", rs_region,  
                        public=False)  
rs_image = rs.get_client("image", rs_region)  
  
hp_compute = hp.get_client("compute", hp_region)  
hp_obj = hp.get_client("object_store", hp_region)  
hp_image = hp.get_client("image", hp_region)
```

Create Clients

```
rs_region = "ORD"  
hp_region = "region-a.geo-1"
```

```
rs_compute = rs.get_client("compute", rs_region)  
rs_obj = rs.get_client("object_store", rs_region,  
                        public=False)  
rs_image = rs.get_client("image", rs_region)
```

```
hp_compute = hp.get_client("compute", hp_region)  
hp_obj = hp.get_client("object_store", hp_region)  
hp_image = hp.get_client("image", hp_region)
```

Create Clients

```
rs_region = "ORD"  
hp_region = "region-a.geo-1"
```

```
rs_compute = rs.get_client("compute", rs_region)  
rs_obj = rs.get_client("object_store", rs_region,  
                        public=False)  
rs_image = rs.get_client("image", rs_region)
```

```
hp_compute = hp.get_client("compute", hp_region)  
hp_obj = hp.get_client("object_store", hp_region)  
hp_image = hp.get_client("image", hp_region)
```

Take a Snapshot

```
rs_cont_name = "export_images"  
hp_cont_name = "upload_images"
```

```
instance = rs_compute.servers.find(name="glue_server")  
snap = instance.create_image(instance, "glue_snap")  
utils.wait_for_build(snap, verbose=True)
```

Take a Snapshot

```
rs_cont_name = "export_images"  
hp_cont_name = "upload_images"
```

```
instance = rs_compute.servers.find(name="glue_server")  
snap = instance.create_image(instance, "glue_snap")  
utils.wait_for_build(snap, verbose=True)
```


Take a Snapshot

```
rs_cont_name = "export_images"  
hp_cont_name = "upload_images"
```

```
instance = rs_compute.servers.find(name="glue_server")  
snap = instance.create_image(instance, "glue_snap")  
utils.wait_for_build(snap, verbose=True)
```

Take a Snapshot

```
rs_cont_name = "export_images"  
hp_cont_name = "upload_images"
```

```
instance = rs_compute.servers.find(name="glue_server")  
snap = instance.create_image(instance, "glue_snap")  
utils.wait_for_build(snap, verbose=True)
```

Export the Image

```
task = rs_image.export_task(snap, rs_cont_name)
utils.wait_for_build(task, verbose=True,
                     desired=["success", "failure"])
```

Transfer to HP

```
object_name = "%s.vhd" % snap.id
start = time.time()
dlo_parts = rs_obj.dlo_fetch(rs_cont_name, obj_name)
dlo_upload(hp, hp_region, hp_cont_name, dlo_parts,
            obj_name)
end = time.time()
elapsed = end - start
logit("It took %8.2f seconds to transfer "
      "the image." % elapsed)
```

Transfer to HP

```
object_name = "%s.vhd" % snap.id
start = time.time()
dlo_parts = rs_obj.dlo_fetch(rs_cont_name, obj_name)
dlo_upload(hp, hp_region, hp_cont_name, dlo_parts,
            obj_name)
end = time.time()
elapsed = end - start
logit("It took %8.2f seconds to transfer "
      "the image." % elapsed)
```

Transfer to HP

```
object_name = "%s.vhd" % snap.id
start = time.time()
dlo_parts = rs_obj.dlo_fetch(rs_cont_name, obj_name)
dlo_upload(hp, hp_region, hp_cont_name, dlo_parts,
            obj_name)
end = time.time()
elapsed = end - start
logit("It took %8.2f seconds to transfer "
      "the image." % elapsed)
```



```

def dlo_upload(ctx, region, cont, parts, base_name,
               chunk_size=None, max_threads=None):
    """
    Uploads a DLO (Dynamic Large Object) that has been fetched
    into a list of generator objects (typically by dlo_fetch)
    to the specified container.

    It uses a default chunk_size of 64K; you may pass in a
    different value if it improves performance.

    By default this creates a thread for each 'part'. If
    you wish to reduce that for any reason, pass a limiting
    value to 'max_threads'.
    """
    DEFAULT_CHUNKSIZE = 65536
    if chunk_size is None:
        chunk_size = DEFAULT_CHUNKSIZE

    class Uploader(threading.Thread):
        def __init__(self, client, cont, queue, num):
            super(Uploader, self).__init__()
            self.client = client
            logit("Uploader", num, "created.")
            self.cont = cont
            self.queue = queue
            self.num = num

        def run(self):
            while True:
                try:
                    job = self.queue.get(False)
                    nm, gen = job
                    logit("Uploader # %s storing '%s'." % (self.num, nm))
                    self.client.store_object(self.cont, nm, gen,
                                             chunk_size=chunk_size, return_none=True)
                    logit("Uploader # %s finished storing." % self.num)
                except Queue.Empty:
                    break
            logit("**DONE** Uploader # %s terminating." % self.num)

    queue = Queue.Queue()
    for part in parts:
        queue.put(part)
    num_threads = len(parts)
    if max_threads is not None:
        num_threads = min(num_threads, max_threads)

    workers = []
    for num in range(num_threads):
        clt = ctx.get_client("object_store", region, cached=False)
        worker = Uploader(clt, cont, queue, num)
        workers.append(worker)
    for worker in workers:
        worker.start()
    for worker in workers:
        worker.join()

    # Upload the manifest
    headers = {"X-Object-Manifest": "%s/%s" % (cont, base_name)}
    clt = ctx.get_client("object_store", region, cached=False)
    clt.store_object(cont, base_name, "", headers=headers, return_none=True)

```

Uploader Thread

```
class Uploader(threading.Thread):
    def __init__(self, client, cont, queue, num):
        super(Uploader, self).__init__()
        self.client = client
        logit("Uploader", num, "created.")
        self.cont = cont
        self.queue = queue
        self.num = num

    def run(self):
        while True:
            try:
                job = self.queue.get(False)
                nm, gen = job
                logit("Uploader # %s storing '%s'." % (self.num, nm))
                self.client.store_object(self.cont, nm, gen,
                                         chunk_size=chunk_size, return_none=True)
                logit("Uploader # %s finished storing." % self.num)
            except Queue.Empty:
                break
        logit("**DONE** Uploader # %s terminating." % self.num)
```

Uploader Thread

```
class Uploader(threading.Thread):
    def __init__(self, client, cont, queue, num):
        super(Uploader, self).__init__()
        self.client = client
        logit("Uploader", num, "created.")
        self.cont = cont
        self.queue = queue
        self.num = num

    def run(self):
        while True:
            try:
                job = self.queue.get(False)
                nm, gen = job
                logit("Uploader # %s storing '%s'." % (self.num, nm))
                self.client.store_object(self.cont, nm, gen,
                                         chunk_size=chunk_size, return_none=True)
                logit("Uploader # %s finished storing." % self.num)
            except Queue.Empty:
                break
        logit("**DONE** Uploader # %s terminating." % self.num)
```

Uploader Thread

```
class Uploader(threading.Thread):
    def __init__(self, client, cont, queue, num):
        super(Uploader, self).__init__()
        self.client = client
        logit("Uploader", num, "created.")
        self.cont = cont
        self.queue = queue
        self.num = num

    def run(self):
        while True:
            try:
                job = self.queue.get(False)
                nm, gen = job
                logit("Uploader # %s storing '%s'." % (self.num, nm))
                self.client.store_object(self.cont, nm, gen,
                                         chunk_size=chunk_size, return_none=True)
                logit("Uploader # %s finished storing." % self.num)
            except Queue.Empty:
                break
        logit("**DONE** Uploader # %s terminating." % self.num)
```

Upload a DLO

```
def dlo_upload(ctx, region, cont, parts, base_name,
               chunk_size=None, max_threads=None):
    DEFAULT_CHUNKSIZE = 65536
    if chunk_size is None:
        chunk_size = DEFAULT_CHUNKSIZE
    queue = Queue.Queue()
    for part in parts:
        queue.put(part)
    num_threads = len(parts)
    if max_threads is not None:
        num_threads = min(num_threads, max_threads)
    workers = []
    for num in range(num_threads):
        clt = ctx.get_client("object_store", region, cached=False)
        worker = Uploader(clt, cont, queue, num)
        workers.append(worker)
    for worker in workers:
        worker.start()
    for worker in workers:
        worker.join()
    # Upload the manifest
    headers = {"X-Object-Manifest": "%s/%s" % (cont, base_name)}
    clt = ctx.get_client("object_store", region, cached=False)
    clt.store_object(cont, base_name, "", headers=headers,
                    return_none=True)
```

Upload a DLO

```
def dlo_upload(ctx, region, cont, parts, base_name,
               chunk_size=None, max_threads=None):
    DEFAULT_CHUNKSIZE = 65536
    if chunk_size is None:
        chunk_size = DEFAULT_CHUNKSIZE
    queue = Queue.Queue()
    for part in parts:
        queue.put(part)
    num_threads = len(parts)
    if max_threads is not None:
        num_threads = min(num_threads, max_threads)
    workers = []
    for num in range(num_threads):
        clt = ctx.get_client("object_store", region, cached=False)
        worker = Uploader(clt, cont, queue, num)
        workers.append(worker)
    for worker in workers:
        worker.start()
    for worker in workers:
        worker.join()
    # Upload the manifest
    headers = {"X-Object-Manifest": "%s/%s" % (cont, base_name)}
    clt = ctx.get_client("object_store", region, cached=False)
    clt.store_object(cont, base_name, "", headers=headers,
                    return_none=True)
```

Upload a DLO

```
def dlo_upload(ctx, region, cont, parts, base_name,
               chunk_size=None, max_threads=None):
    DEFAULT_CHUNKSIZE = 65536
    if chunk_size is None:
        chunk_size = DEFAULT_CHUNKSIZE
    queue = Queue.Queue()
    for part in parts:
        queue.put(part)
    num_threads = len(parts)
    if max_threads is not None:
        num_threads = min(num_threads, max_threads)
    workers = []
    for num in range(num_threads):
        clt = ctx.get_client("object_store", region, cached=False)
        worker = Uploader(clt, cont, queue, num)
        workers.append(worker)
    for worker in workers:
        worker.start()
    for worker in workers:
        worker.join()
    # Upload the manifest
    headers = {"X-Object-Manifest": "%s/%s" % (cont, base_name)}
    clt = ctx.get_client("object_store", region, cached=False)
    clt.store_object(cont, base_name, "", headers=headers,
                    return_none=True)
```


Upload a DLO

```
def dlo_upload(ctx, region, cont, parts, base_name,
               chunk_size=None, max_threads=None):
    DEFAULT_CHUNKSIZE = 65536
    if chunk_size is None:
        chunk_size = DEFAULT_CHUNKSIZE
    queue = Queue.Queue()
    for part in parts:
        queue.put(part)
    num_threads = len(parts)
    if max_threads is not None:
        num_threads = min(num_threads, max_threads)
    workers = []
    for num in range(num_threads):
        clt = ctx.get_client("object_store", region, cached=False)
        worker = Uploader(clt, cont, queue, num)
        workers.append(worker)
    for worker in workers:
        worker.start()
    for worker in workers:
        worker.join()
    # Upload the manifest
    headers = {"X-Object-Manifest": "%s/%s" % (cont, base_name)}
    clt = ctx.get_client("object_store", region, cached=False)
    clt.store_object(cont, base_name, "", headers=headers,
                    return_none=True)
```

Upload a DLO

```
def dlo_upload(ctx, region, cont, parts, base_name,
               chunk_size=None, max_threads=None):
    DEFAULT_CHUNKSIZE = 65536
    if chunk_size is None:
        chunk_size = DEFAULT_CHUNKSIZE
    queue = Queue.Queue()
    for part in parts:
        queue.put(part)
    num_threads = len(parts)
    if max_threads is not None:
        num_threads = min(num_threads, max_threads)
    workers = []
    for num in range(num_threads):
        clt = ctx.get_client("object_store", region, cached=False)
        worker = Uploader(clt, cont, queue, num)
        workers.append(worker)
    for worker in workers:
        worker.start()
    for worker in workers:
        worker.join()
    # Upload the manifest
    headers = {"X-Object-Manifest": "%s/%s" % (cont, base_name)}
    clt = ctx.get_client("object_store", region, cached=False)
    clt.store_object(cont, base_name, "", headers=headers,
                    return_none=True)
```

Upload a DLO

```
def dlo_upload(ctx, region, cont, parts, base_name,
               chunk_size=None, max_threads=None):
    DEFAULT_CHUNKSIZE = 65536
    if chunk_size is None:
        chunk_size = DEFAULT_CHUNKSIZE
    queue = Queue.Queue()
    for part in parts:
        queue.put(part)
    num_threads = len(parts)
    if max_threads is not None:
        num_threads = min(num_threads, max_threads)
    workers = []
    for num in range(num_threads):
        clt = ctx.get_client("object_store", region, cached=False)
        worker = Uploader(clt, cont, queue, num)
        workers.append(worker)
    for worker in workers:
        worker.start()
    for worker in workers:
        worker.join()
    # Upload the manifest
    headers = {"X-Object-Manifest": "%s/%s" % (cont, base_name)}
    clt = ctx.get_client("object_store", region, cached=False)
    clt.store_object(cont, base_name, "", headers=headers,
                    return_none=True)
```

Import the Image

```
data = hp_obj.fetch(hp_cont_name, obj_name)
```

```
new_image = hp_image.create("glue_image", data=data)
```

Import the Image

```
data = hp_obj.fetch(hp_cont_name, obj_name)
```

```
new_image = hp_image.create("glue_image", data=data)
```

Re-create the Instance

```
hp_instance = hp_compute.servers.create("hp_glue_server",  
                                         image=new_image, flavor=some_flavor)  
utils.wait_for_build(hp_instance, verbose=True)
```

```
logit("New Instance Networks:", hp_instance.networks)
```

Re-create the Instance

```
hp_instance = hp_compute.servers.create("hp_glue_server",  
                                         image=new_image, flavor=some_flavor)  
utils.wait_for_build(hp_instance, verbose=True)  
  
logit("New Instance Networks:", hp_instance.networks)
```


That's It!

Thank You!

Ed Leafe

ed@leafefe.com

ed@openstack.org

ed.leafefe@rackspace.com

Twitter: EdLeafe

G+: EdLeafe

Script and Slides: <https://github.com/EdLeafe/gluecon2014>