Big Data BBQ

Dining at the OpenStack buffet to create cloud applications

Michael McCune
Senior Software Engineer
Red Hat

Nikita Konovalov
Software Engineer
Mirantis

April 28, 2016

Caveats

- This is not a blessing of one technology over another
- Some of the samples and features discussed may involve master branch or experimental source code, care should be taken when replicating it in production environments

Problem

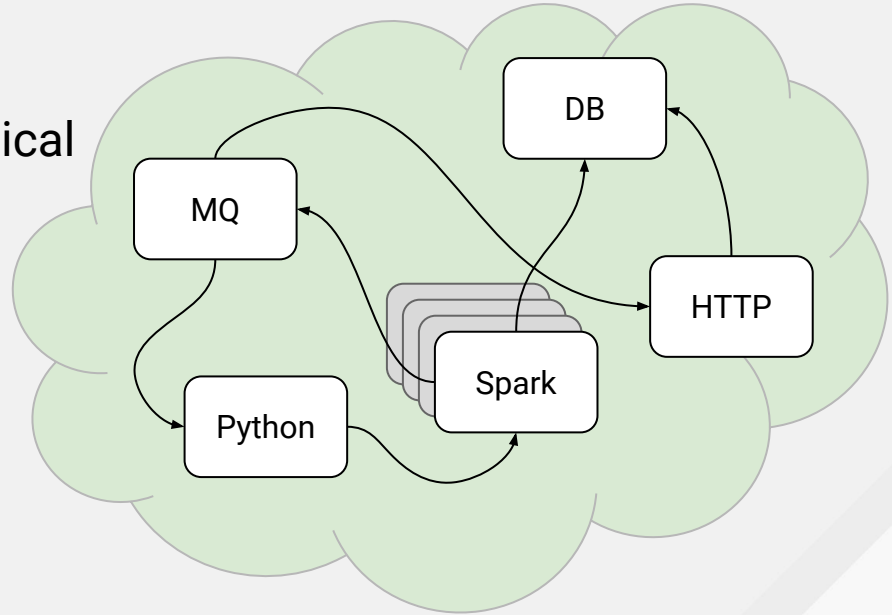
- Want to create big data applications on OpenStack
- Want them to use cloud resources
- Don’t want to bug operations team for continuous deployments

Solution

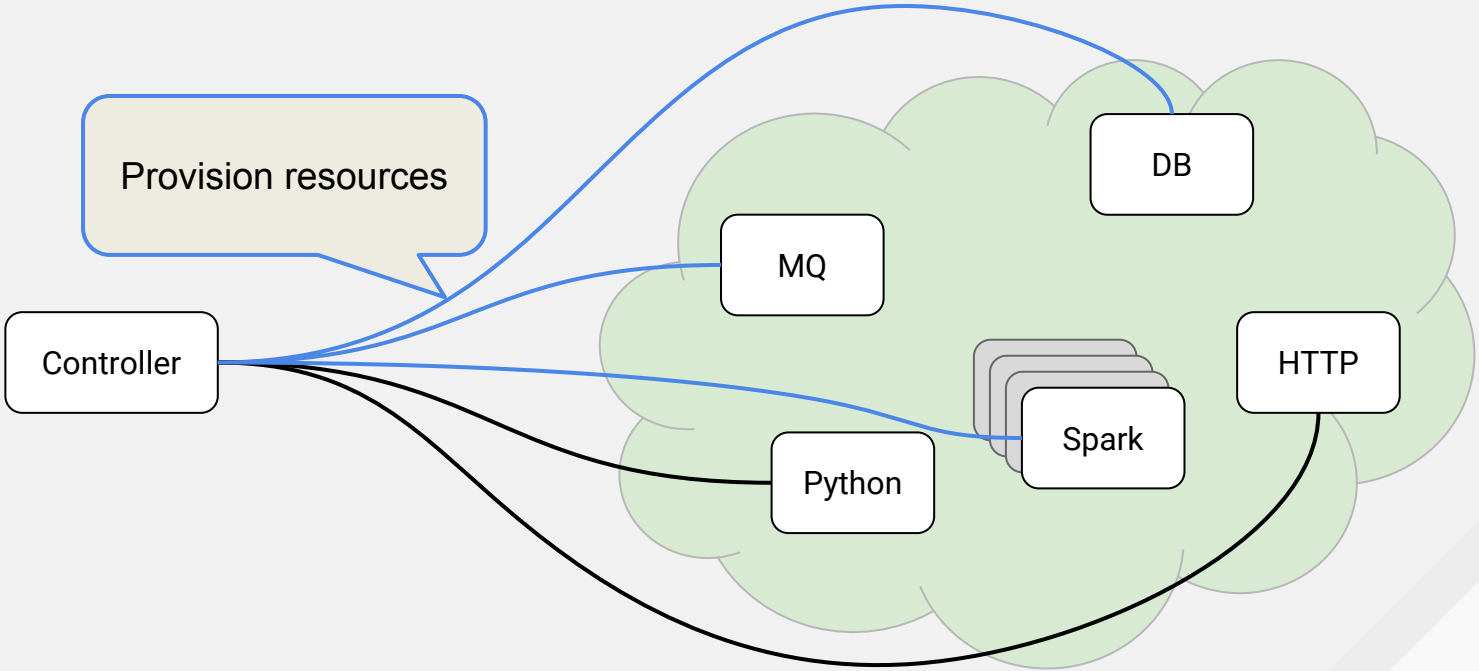
- Build cloud applications
- Create a resilient infrastructure
- Empower developers to run their own upgrades

Cloud applications

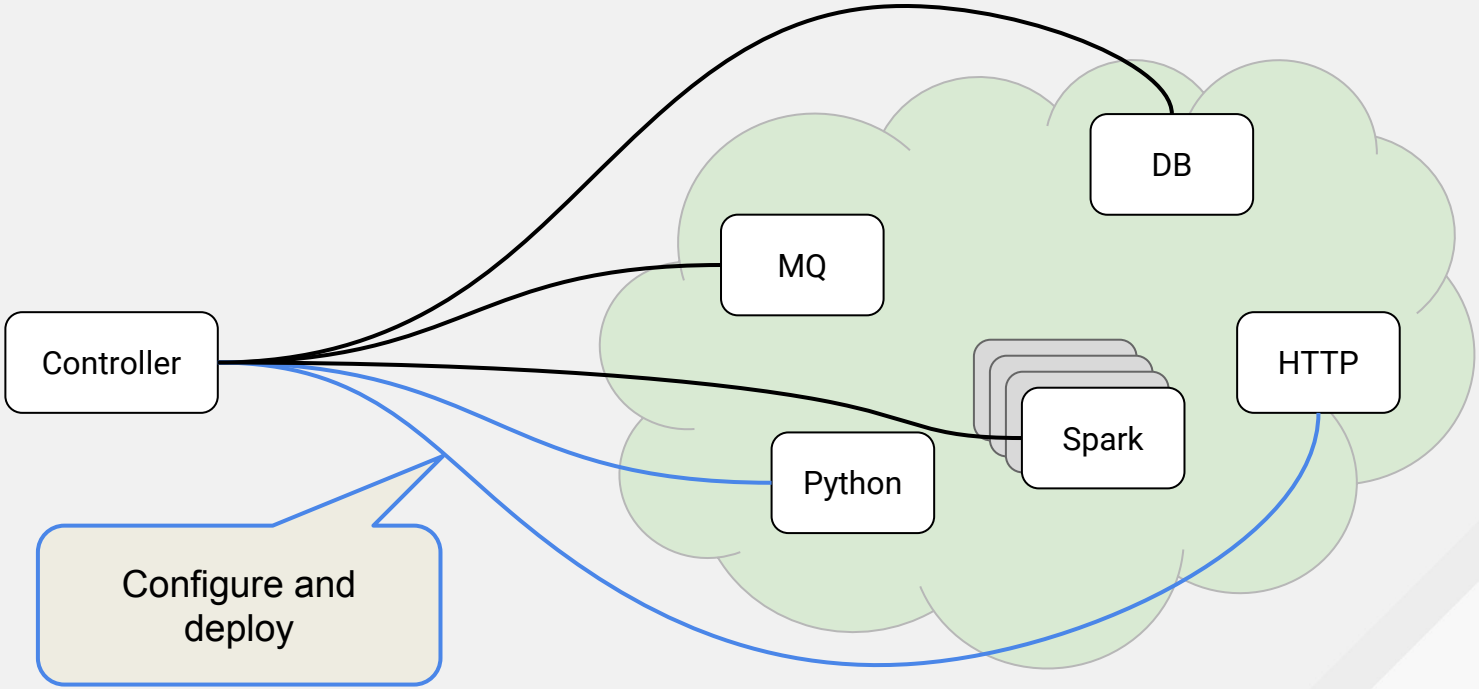
- Collections of applications
- Mixed types; console, server, graphical
- Resource controllers



Cloud applications

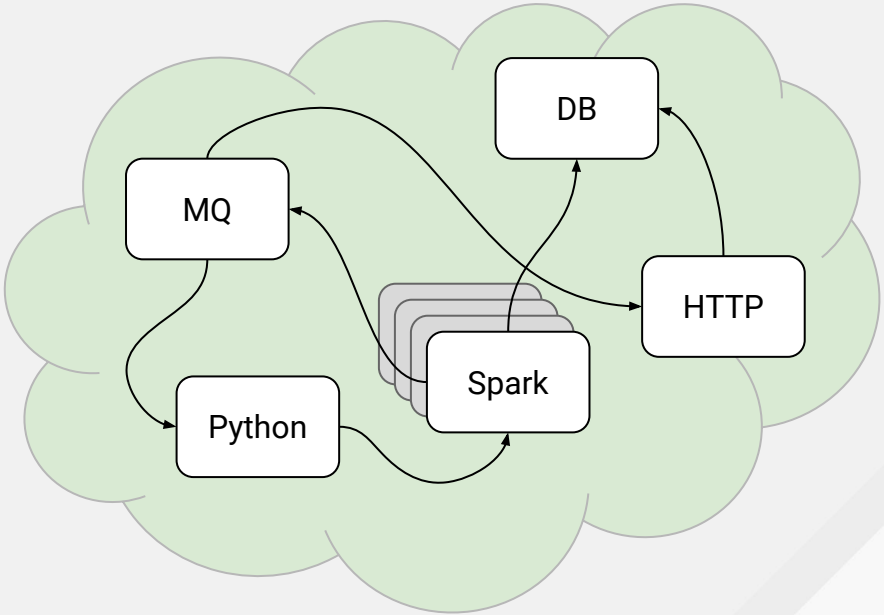


Cloud applications



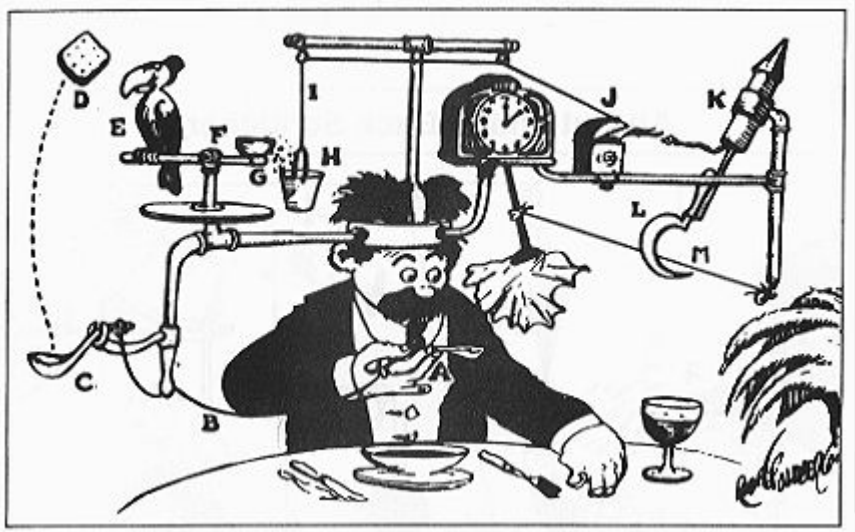
Building a big data application

- What data will you process
- How will you process the data
- Where will you store the results



Planning

- Moving pieces
- The data pipeline
- Storage
- Custom applications



Resiliency

- Build composable pieces
- Be aware of network functionality
- Stateless applications

Martin Kalfatovi, CC BY-NC-SA 2.0

Empowering developers

- Appropriate resource levels
- Access to cloud services
- Production versus Development

Tools of the trade

- Version control
- Configuration management
- Test gating
- Python REPL

Sounds great, now what?

- Foundational elements
- Data processing clusters
- Message queues
- Data stores
- Spark components
- Custom pieces

Alan Chia, CC BY-SA 2.0

OpenStack tooling

- OpenStack Python clients
- Oslo projects
  - oslo.config
  - oslo.log
- Development environments
  - Devstack
  - Kolla
  - OpenStack-Ansible

Deployment

- Create custom images
- Modify running instances
- Re-deploy or update

Deployment

```
if [ "${DIB_DEBUG_TRACE:-0}" -gt 0 ]; then
    set -x
fi
set -eu
set -o pipefail

pushd /opt

echo "Cloning big-data-bbq"
git clone https://github.com/elmiko/big-data-bbq
echo "Installing requirements"
pip install -r big-data-bbq/requirements.txt

popd
```

Deployment

```
env.host_string='10.0.0.3'
env.key_filename='/home/mike/.ssh/cloud_key'
env.user='fedora'

with hide('running', 'stdout', 'stderr'):
   with cd('/opt'):
       run('git clone http://github.com/elmiko/big-data-bbq')
       sudo('pip install -r big-data-bbq/requirements.txt')
```

Configurations

- Command line parameters
- File based options
- Protecting sensitive information

Configurations

```
sahara_cluster_option = [
   cfg.StrOpt('float_pool_network_name', default='public',
              help='The name of the floating pool network to use.'),
   cfg.StrOpt('management_network_name', default='private',
              help='The name of the management network to use.'),
]

conf.register_opts(sahara_cluster_options)
conf.register_opts(password.Password.get_options(), group='keystone')
```

Configurations

```
[DEFAULT]

# The name of the floating pool network to use. (string value)
#float_pool_network_name = public

# The name of the management network to use. (string value)
#management_network_name = private


[keystone]

# Authentication URL (string value)
auth_url = http://mystack.myhost.org:5000/v3

# User ID (string value)
user_id = elmiko
```

Authentication

- Common to all OpenStack components*
- Widely repeated code
- Reusable sessions

Authentication

```
auth = password.Password.load_from_conf_options(conf, group='keystone')
sesh = session.Session(auth=auth)

keystone_client = keystone.Client(session=sesh)

sahara_client = sahara.Client(session=sesh)

trove_client = trove.Client(conf.keystone.username,
                            conf.keystone.password,
                            conf.keystone.project_name,
                            conf.keystone.auth_url)
```

Data processing clusters

- Operation templates
- Provisioning and scaling
- Monitoring cluster progress

Data processing clusters

```
kwargs = {'name': name,
          'plugin_name': 'spark',
          'hadoop_version': '1.6.0',
          'net_id': mgmt_net,
          'cluster_configs': {'general': {'Enable NTP service': False}},
          'node_groups': [
            {'count': 1,
             'name': 'spark160-master',
             'flavor_id': '2',
             'node_processes': ['namenode', 'master'],
             'floating_ip_pool': float_pool
            },
            {'count': 3,
             'name': 'spark160-worker',
```

Data processing clusters

```
template_id = sahara_client.cluster_templates.create(**kwargs).id

image_id = sahara_client.images.find(name='my_spark_image')[0].id

keypair_id = nova_client.keypairs.find(name='cloud_key').id

cluster = sahara_client.clusters.create(name='my spark cluster',
                                        plugin_name='spark',
                                        hadoop_version='1.6.0',
                                        cluster_template_id=template_id,
                                        default_image_id=image_id,
                                        user_keypair_id=keypair_id)
```

Data processing clusters

```
while cluster.status != 'Active':
    time.sleep(5)
    cluster = sahara_client.clusters.get(cluster.id)
    if cluster.status == "Error":
        raise Exception('cluster blowed up!')

master_ip = None
for group in cluster.node_groups:
    if group.get('name', '') == 'spark160-master':
        master_ip = group.get('instances', [{}])[0].get('management_ip')
```

Message queues

- Named queues
- Writing messages, options
- Reading, affecting the queue
- Subscribing, how to receive

Message queues

```
# setup queue
queue = zaqar_client.queue('my-data-channel')

# send message
message = {'data': {'value': 9000, 'type': 'strength'}}
body = json.dumps(message)
payload = {'body': body, 'ttl': 60}
queue.send(payload)
```

Message queues

```
# setup queue
queue = zaqar_client.queue('my-data-channel')

# receive message
while True:
    messages = [m for m in queue.pop(count=10)]
    if len(messages) == 0:
        time.sleep(1)
        continue
    for msg in messages:
        process_message(msg.body)
```

Data stores

- What type of store
- Maintenance and longevity
- Providing access to data

Data stores

```
databases = [{'name': database_name}]
users = [{'name': username,
          'password': password,
          'databases': [{'name': database_name}]}]
nics = [{'net-id': network_id}]

instance = trove_client.instances.create(
   name=name, flavor_id=flavor, datastore=datastore_id,
   datastore_version=datastore_version_id, databases=databases,
   users=users, volume={'size': 10}, nics=nics)
```

Data stores

```
while instance.status != 'ACTIVE':
    time.sleep(5)
    instance = trove_client.instances.find(id=instance.id)
    if instance.status == 'ERROR':
        raise Exception('Oops, database instance create FAIL!')

ips = []
for ip in instance.ip:
    m = re.match('^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$', ip)
    if m is not None:
        ips.append(m.group())

conxstr = 'mongodb://{user}:{passwd}@{ip}:27017/{database}'.format(
    username, password, ips[0], database_name)
```

Spark streaming

- Cluster connectivity
- Time slices
- Queues or direct communications

Spark streaming

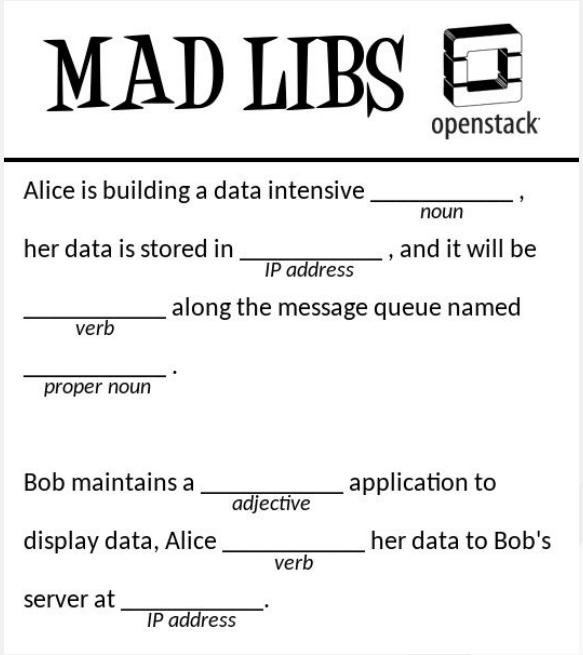
```
args = parser.parse_args()

sconf = SparkConf().setAppName(args.appname)
if args.master:
   sconf.setMaster(args.master)
sc = SparkContext(conf=sconf)
ssc = StreamingContext(sc, 1)

lines = ssc.socketTextStream(args.socket, args.port)
lines.foreachRDD(lambda rdd: process_generic(rdd, args.mongo_url, args.rest_url))

ssc.start()
ssc.awaitTermination()
```

Spark streaming

```
def process_generic(rdd, mongo_url, rest_url):
    count = rdd.count()
    if count is 0:
        return
    count_packet_id = uuid.uuid4().hex
    normalized_rdd = rdd.map(lambda e: repack(e, count_packet_id)).cache()
    store_packets(count_packet_id, count, normalized_rdd, mongo_url)
    signal_rest_server(count_packet_id,
                       count,
                       dict(normalized_rdd.map(
                           lambda e: (e['service'], 1)).reduceByKey(add).collect()),
                       rest_url)
```

Composing applications

- Dynamic resources
- Analytics components
- Custom glue
- Templating



MIRANTIS redhat.

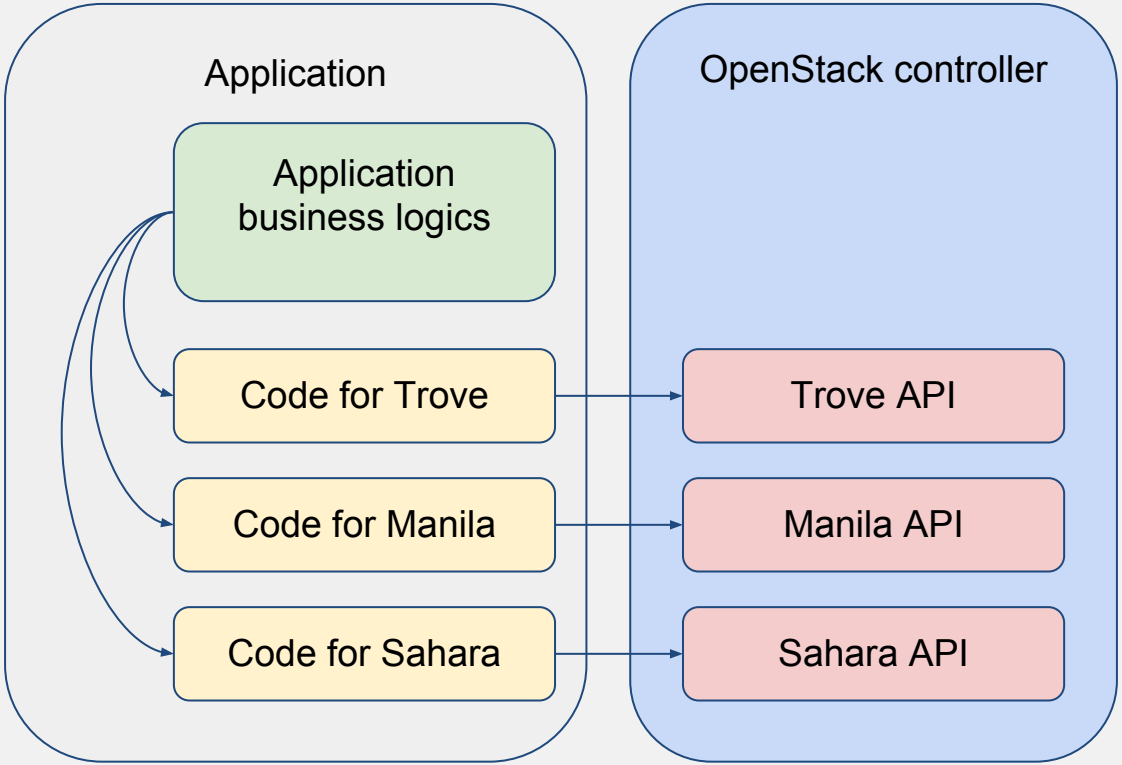
An alternative approach with Heat and Murano

- Provisioning all components through single endpoint
- Using generic syntax
- The syntax is declarative
- Higher level of API

Cloud client call workflow

- Application instantiates and authorizes OpenStack clients
- Application calls clients to reach OpenStack service on controller nodes
- Application handles clients’ responses

Cloud client call workflow



Pros and cons

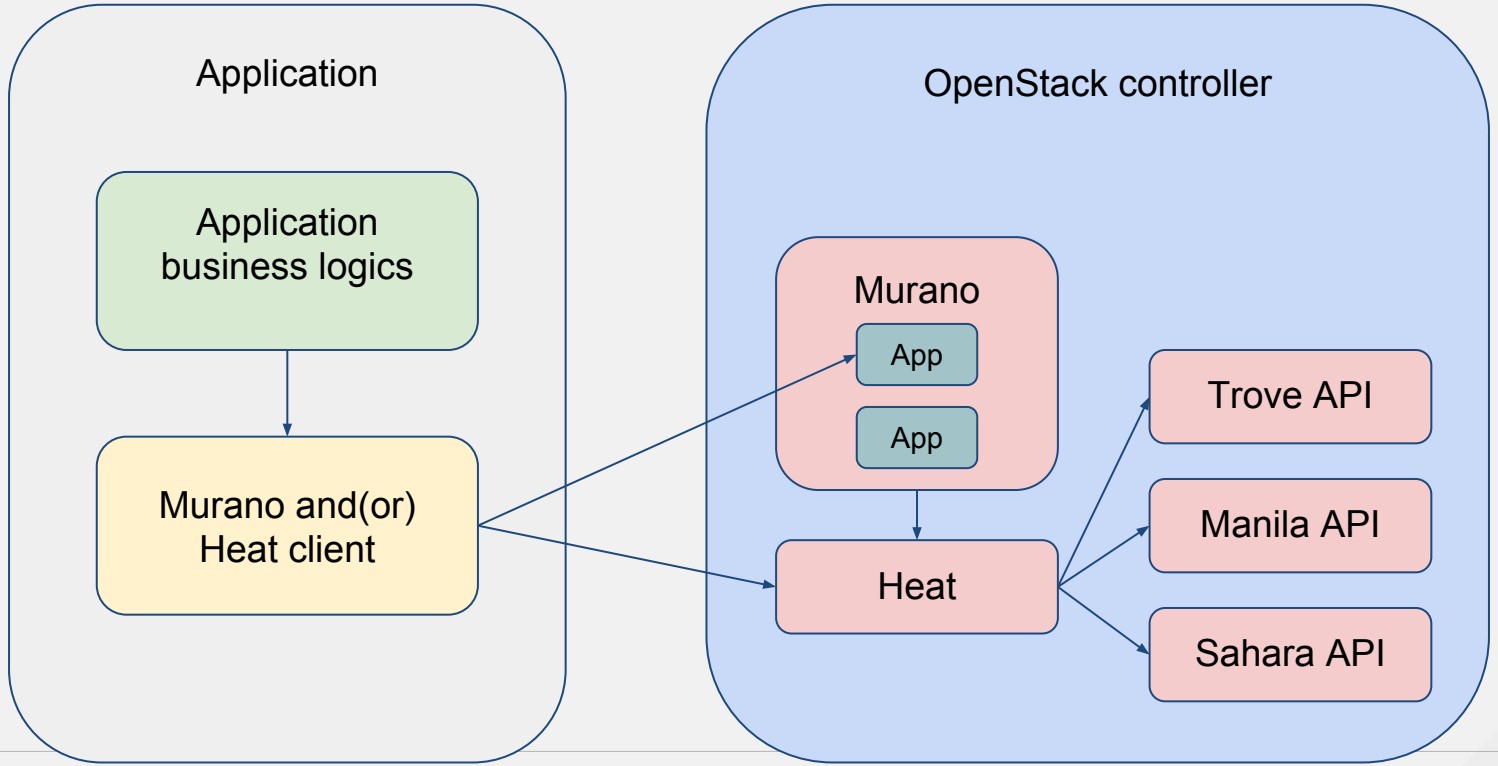
Advantages:

- Full control of the environment
- All operations may run completely independently

Disadvantages:

- Each client handle only a specific service
- You as a developer handle async waits, retries, errors, etc.

Heat and Murano approach



Pros and cons

Advantages:

- Access only one API endpoint
- Murano and heat catalogs of Apps
- UI integration into Horizon
- Handling of async ops, errors and retries is done by Heat and Murano

Disadvantages:

- Calls are limited to Murano apps and Heat templates

Building a Heat stack

- Declare parameters
- Declare resources. (using parameters)

Building a Heat template. Parameters.

Parameters are the customization point.

Input of basic types is allowed.

Example:

```
parameters:
  image_id:
    type: string
    description: Image used for servers
  instance_type:
    type: string
    default: m1.small
  num_instances:
    type: number
    description: Number of instances to create
    default: 1
```

Building a Heat template. Resources.

Resources section lists all VMs, Volumes, Shares, DBs, etc.

Parameters may be used here

Example:

```
resources:
  server_group:
    type: OS::Heat::InstanceGroup
    properties:
      LaunchConfigurationName: { get_resource: server_config }
      AvailabilityZones: []
      Size: { get_param: num_instances }
  server_config:
    type: AWS::AutoScaling::LaunchConfiguration
    properties:
      ImageId: { get_param: image_id }
      InstanceType: { get_param: instance_type}
      KeyName: { get_param: key_name }
```

Building a Murano App

- Create an App manifest
- Build an execution plan
- Describe deployment process
- Declare UI components
- Package everything

Building a Murano App. Manifest.

```
Format: 1.0
  Type: Application
  FullName: io.murano.apps.apache.ApacheHttpServer
  Name: Apache HTTP Server
  Description: |
   The Apache HTTP Server Project is an effort to develop and maintain an
   open-source HTTP server for modern operating systems including UNIX and
   Windows NT.
   ...
  Author: My Company, Inc
  Tags: [HTTP, Server, WebServer, HTML, Apache]
  Classes:
   io.murano.apps.apache.ApacheHttpServer: ApacheHttpServer.yaml
```

Building a Murano App. Execution plan.

```
Name: Deploy Apache
 Parameters:
   enablePHP: $enablePHP
 Body: |
   return apacheDeploy('{0}'.format(args.enablePHP)).stdout
 Scripts:
   apacheDeploy:
     Type: Application
     EntryPoint: runApacheDeploy.sh
     Files: []
     Options:
       captureStdout: true
       captureStderr: true
```

Building a Murano App. Deployment process.

```
Properties:
  name:
    Contract: $.string().notNull()
  enablePHP:
    Contract: $.bool()
  instance:
    Contract: $.class(res:Instance).notNull()
Methods:
  initialize:
    Body:
      <App initialization calls>
  deploy:
    Body:
      <App deployment calls>
```

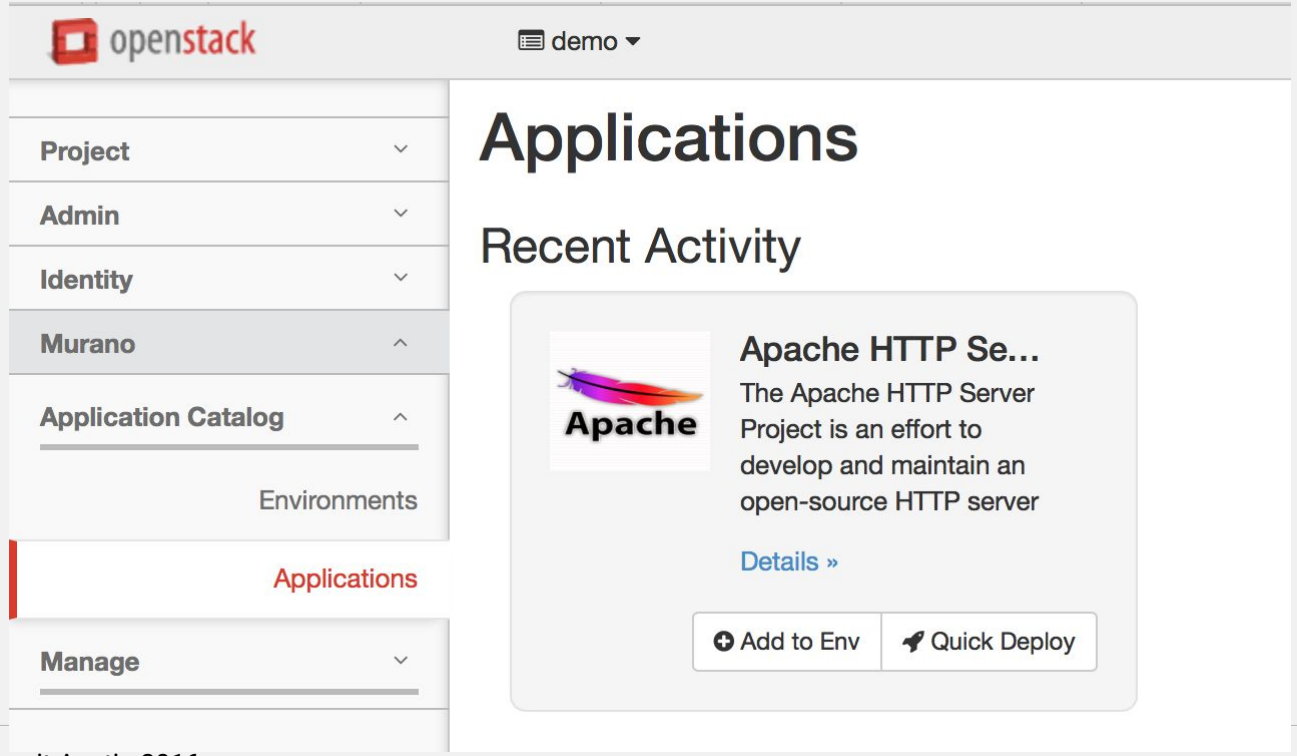
Building a Murano App. UI components

```
Forms:
  - appConfiguration:
      fields:
        - name: name
          type: string
          label: Application Name
          initial: 'ApacheHttpServer'
          description: >-
            Enter a desired name for the application. Just A-Z, a-z, 0-9
        - name: enablePHP
          label: Enable PHP
          type: boolean
          required: false
```

Packaging it all together

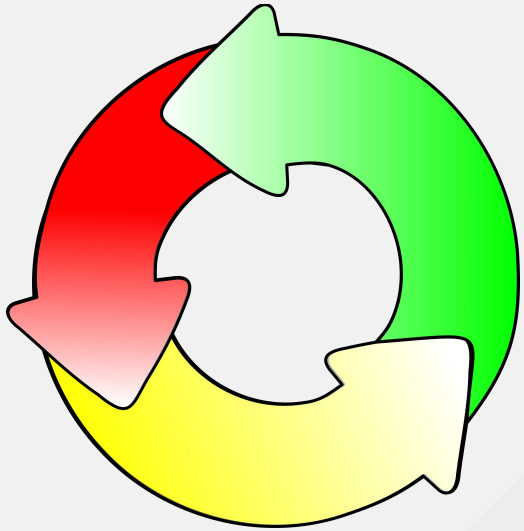
```
|_  Classes
|   |_  ApacheHttpServer.yaml
|
|_  Resources
|   |_  scripts
|       |_ runApacheDeploy.sh
|   |_  DeployApache.template
|
|_  UI
|   |_  ui.yaml
|
|_  logo.png
|
|_  manifest.yaml
```

Ready to go


openstack
demo
Project
Admin
Identity
Murano
Application Catalog
Environments
Applications
Manage
Applications
Recent Activity
Apache HTTP Se...
The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server
Details »
Add to Env
Quick Deploy

The application life-cycle

- Iterating on projects
- Team-based building
- Development vs. production



Projects to watch

- OpenStack SDK
  - http://developer.openstack.org/sdks/python/openstacksdk/
- Murano
  - https://murano.readthedocs.org/
- Spark
  - https://spark.apache.org
- Big Data BBQ presentation examples
  - https://github.com/elmiko/big-data-bbq

Further reading

- OpenStack APIs and SDKs
  - http://developer.openstack.org
- Heat template Examples
  - https://github.com/openstack/heat-templates
- Application Catalog for Heat and Murano
  - https://apps.openstack.org/
- Guide for Murano Examples
  - https://murano.readthedocs.org/en/stable-liberty/draft/appdev-guide/step_by_step.html

Questions?

redhat

MIRANTIS

Get out there and build!