# Big Data: and the Compute Canada Cloud

*May 2016*

computecanada
regional partner

# Overview

- Slides available at: https://www.ace-net.ca/training/workshops-seminars/

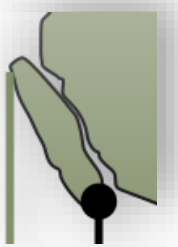
- Part I (30 mins)
  - Introduction to CC – Cloud
  - Create your first VM
  - No experience required

- Part II (90 mins)
  - Advanced cloud usage
  - Automating setup of a Big Data cluster on CC – cloud
  - Useful background knowledge:
    - Command line (bash)
    - Programing (Python)

# Compute Canada Cloud

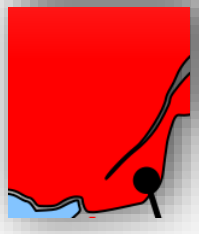
- What is a “Cloud”
  - Hardware emulation in software
- Why use cloud for big data?
  - Big data tools don’t easily fit into classical HPC framework
- Important differences between “Cloud” and classical HPC
  - Classical HPC
    - Platform as a service (PaaS)
      - CC/ACENET manages hardware, OS
    - Software as a service (SaaS)
      - CC/ACENET manages hardware, OS, and software
  - Infastructure as a service (IaaS)
    - CC manages the hardware
    - You manage OS
    - You manage Software
- Want to add more support for cloud
  - Take it from IaaS in the direction of PaaS or SaaS
- Increase in amount of cloud hardware
  - Facilitate more users
  - Makes Paas & SaaS more important

# Current CC-Cloud Resources



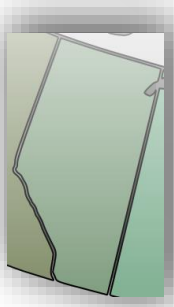
## West Cloud (Victoria)

- 42 nodes
  - 2 x Intel E5-2650v2 (hypervisor)
  - 32 x 256 GB RAM (compute)
  - 8 x 512 GB RAM (compute)
  - 16 cores per node (640 compute cores)
- ~200 TB Usable storage



## East Cloud (Sherbrooke)

- 38 nodes
  - 2 x Intel E5-2650v2 (hypervisor)
  - 36 x 128 GB of RAM (compute)
  - 16 cores per node (576 compute cores)
- 100TB usable storage



## Alberta Prototyping Cloud

- 150 TB of storage

ACENET
accelerate discovery

computecanada
regional partner

# Upcoming CC Resources

- GP1 @ UoV (all OpenStack)
  - Early 2016, 40% expansion in 2017
  - 3,000+ CPUs
- GP2 @ SFU (Mix of OpenStack and batch)
  - Mid 2016, 40% expansion in 2017
  - 18,000+ CPUs
  - 192 GPU Nodes
- GP3 @ UoW (Mix of OpenStack and batch)
  - Late 2016, 40 % expansion in 2017
  - 19,000+ CPUs
  - 64 GPU Nodes
- LP @ UoT (batch)
  - Mid 2017
  - 66,000 CPUs
- Would like the number of cloud nodes on the GP2-3 machines to be flexible based on demand

https://www.computecanada.ca/wp-content/uploads/2015/11/Compute-Canada-Technology-Briefing-2015.pdf

# Cloud Accounts

Info about cloud at:
https://www.computecanada.ca/research-portal/national-services/compute-canada-cloud/


- 2 Virtual Machines
- 2 public Ips (1 @ west cloud)
- 15G of RAM
- 4 VCPUs
- 40G of permanent storage
- 2 Volumes
- 2 Snapshots

Technical Questions/problems:
cloud@computecanada.ca

- RAC
  Resource Allocation Competitions
- RPP
  Research Portals & Platforms
- Call for proposals each fall

  https://www.computecanada.ca/research-portal/accessing-resources/resource-allocation-competitions/

- Current available cloud resources are heavily used
  - Awaiting arrival of GP1
    (~ 3X increase in # cores)

rac@computecanada.ca

# Creating a VM on OpenStack

- Create a VM
  - Add an SSH Public Key
  - Launching a VM
  - Allocate a floating IP
  - Assigning a floating IP to VM
  - Add SSH Security Rule
- Windows:
  - mobaXterm to connect and create key-pair http://mobaxterm.mobatek.net/download.html
- Linux/Mac
  - Use built-in terminal
- Create an ssh key-pair

- Log in to CC-cloud dashboard **http://east.cloud.computecanada.ca** **http://west.cloud.computecanada.ca**
- Docs: http://docs.computecanada.ca/wiki/Cloud_Quick_Start
- Jon Simpson's webinar on creating a VM http://computecanada.ca/research-portal/national-services/compute-canada-cloud/

```
$ ssh-keygen
$ cat ~/.ssh/id_rsa.pub
```

ACENET accelerate discovery

computecanada regional partner

# Part II Setup

1. Get OpenStack RC file

   *Compute -> Access & Security -> API Access -> Download OpenStack RC File*

2. Put RC file and private key on VM

```
$ sftp -i ~/.ssh/id_rsa ubuntu@<VM public ip>
sftp> put <openstack-rc-file>
sftp> put <ACENET_WORKSHOP_KEY>
sftp> exit
```

# Part II Setup

3. Install openstack client + quick test

```
$ ssh -i .ssh/id_rsa ubuntu@<your public ip>
$ sudo apt-get update
$ sudo apt-get install python-dev python-pip
$ sudo pip install python-openstackclient
$ source <openstack-rc-file>
$ openstack server list
```

# Part II: Advanced Cloud

# OpenStack - CLI

- Access more functionality than available in Dashboard
  - e.g. Download a VM image
- Automation through scripts
  (though python API likely a better choice)
  - Good cloud etiquette to shutdown VMs when not in use
- Used from any machine
  - Internet
  - CLI installed
- To connect CLI to your OpenStack account
  - Source RC settings file on OpenStack dashboard
- CC wiki docs for OpenStack CLI:
  https://docs.computecanada.ca/wiki/OpenStack_Command_Line_Clients

# CLI Example: Create a VM

- Collect components to use for VM
  - Image
  - Flavor
  - Keypair

```
$ openstack image list
$ openstack keypair list
$ openstack flavor list
```

# CLI Example: Create a VM

- Create VM
- Assign Floating IP

```
$ openstack server create
--image Ubuntu_14.04_Trusty-amd64-20150708
--key-name ACENET_WORKSHOP_KEY
--flavor p1-0.75gb
test2

$ openstack ip floating list
$ openstack ip floating add <ip-address>
<server>
$ ssh -i <private-key> ubuntu@<ip-address>
```

# CLI Example: Download VM

- Download a VM from cloud
  - Create image from VM
  - List VM images to get ID/name
  - Then Download
- Launch in Virtual Box on your Desktop
  - Convert qcow2 format to vmdk
    - Use qemu-img tool (qemu-utils package)

```
$ openstack sever image create test2
$ openstack image list
$ openstack image save test2 --file test.qcow2
```

# OpenStack – Python API

- OpenStack CLI uses the Python API under the hood
  - Installing CLI provides Python API
- Using the Python API directly
  - Avoids calling CLI commands via system calls
  - Access to built OpenStack python objects
    - E.g. Server, ip, etc.
- Official docs: docs.openstack.org/user-guide/sdk.html
- Python introspection is also very useful

```
dir()

<item>.__doc__
```

# OS Python API: example

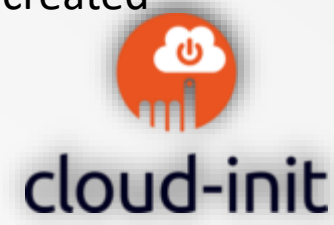
```
$ git clone https://github.com/cgeroux/big-data-workshop.git

$ cd big-data-workshop/openstack-python-api
$ ./openstack-python-api.py
test
test2
```

- Goal: setup Hadoop and Spark in a cluster of VMs in the cloud
  - manage_cluster.py https://github.com/cgeroux/cloud-init.git

# Provisioning: CloudInit & SaltStack

- Next configure and setup a VM after it is created
- CloudInit runs post VM creation
  - Documentation: cloudinit.readthedocs.org
  - One time execution
  - Install/Configure software
  - Edits/creates files
  - Run commands
  - Used in manage_cluster.py to install saltstack
- SaltStack used to manage VMs during & after initial setup
  - Documentation: docs.saltstack.com
  - More functionality than CloudInit
  - Used on many different platforms (not just cloud)
  - Install/Configure software, manage files
  - Many “formulas” available e.g. Hadoop
  - Used in manage_cluster.py to install and configure java, hadoop, and spark
    - https://github.com/cgeroux/hadoop-formula.git
- Now that software is setup on VMs how do we use it?




# Big Data Tools

- HDFS: Hadoop distributed file system
  - Foundation for many big data tools
  - Parallel file IO & hardware failure protection

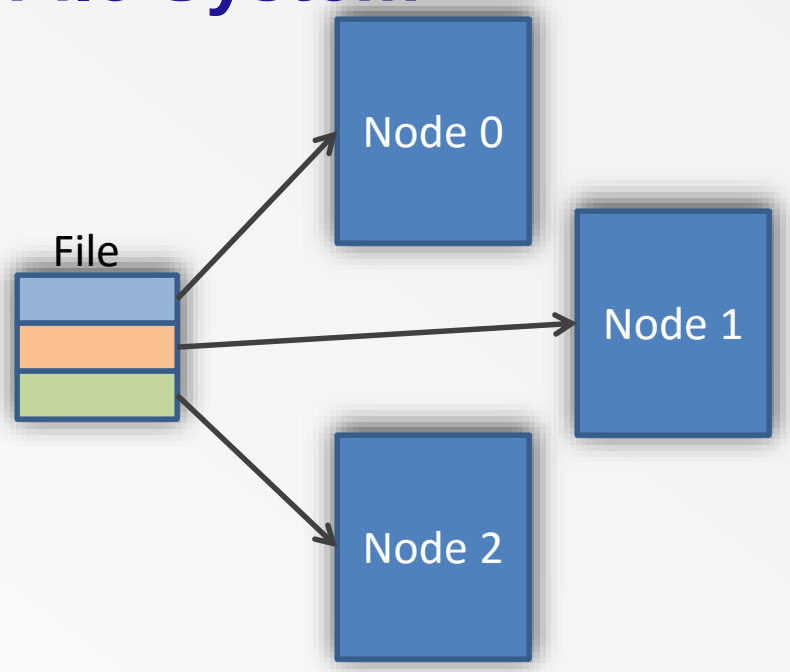
- To work with data in HDFS:
  - MapReduce: used to move compute to data
    - initial release of Hadoop MR in 2011

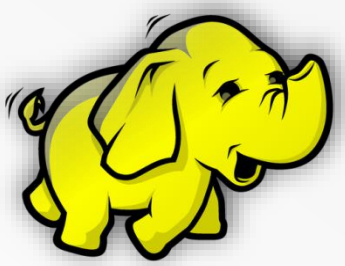
  - Spark: provides in-memory caching of data
    - Improves on MR
    - Initial release of Spark in May 30, 2014

# HDFS

## Hadoop Distributed File System

- Replication for speed and safety
  - Files are broken into chunks
  - Chunks are replicated on different nodes
  - File can be read/written at greater the speed
  - Compute distributed to data (MR)
- To logon to workshop Hadoop/Spark cluster
  - Download temporary private key (guest-key) from given ip to desktop to use with provided user name



# HDFS

- Connect to Hadoop/Spark cluster with

```
$ ssh -i ~/Desktop/guest-key guest#@<ip-of-master>
```

  - `hdfs dfs -<cmd>`
  - `ls,cp,cat,mv,mkdir,rm,rmdir,chmod,du,df`
- Some extra ones:
  - `put,get,appendToFile,help`
- Full list of commands at: http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html

# Working with HDFS

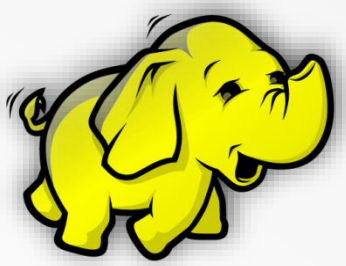
```
$ echo "hello world from inside HDFS" >
test.txt
$ cat test.txt
hello world from inside HDFS
$ hdfs dfs -put test.txt
$ hdfs dfs -ls
```

Assume relative to HDFS “home directory” `/user/guest#`

```
$ hdfs dfs -ls /
$ hdfs dfs -ls /user
$ hdfs dfs -cat test.txt
```

# Hadoop - MapReduce

- Runs compute process in a distributed way on local data
  - Avoid disk IO limit (reading from multiple disks)
  - Reduce network latency (limit amount of data sent)
- A programable data processing pipeline
  - Input divided into “splits” (splitting)
  - “splits” mapped onto key/value pairs (mapping)
  - The pairs are then sorted or shuffled based on their key (shuffling)
  - Then reduced and written to disk (reducing)
- Each stage can be programmed
  - Shuffling (experimental)

# Spark

- A fast general-purpose cluster computing system
  - Designed to use HDFS
- Uses RDD (Resilient Distributed Dataset)
- APIs:
  - Java, Python, R, Scala
- Supports high level tools:
  - Spark SQL : databases
  - Mllib: machine learning
  - GraphX: graph processing
  - Spark Streaming: processing of live data streams

# Spark

- Spark solves some of the problems of MR
  - Need for fast response (real-time)
  - Iterative methods (one step depends on the last)
  - Not all algorithms fit nicely into a MapReduce framework
- By:
  - Faster
    - In some cases suitable for real-time
  - Cache data in memory for reuse
  - RDD allow more programing flexibility

# Spark: word count example

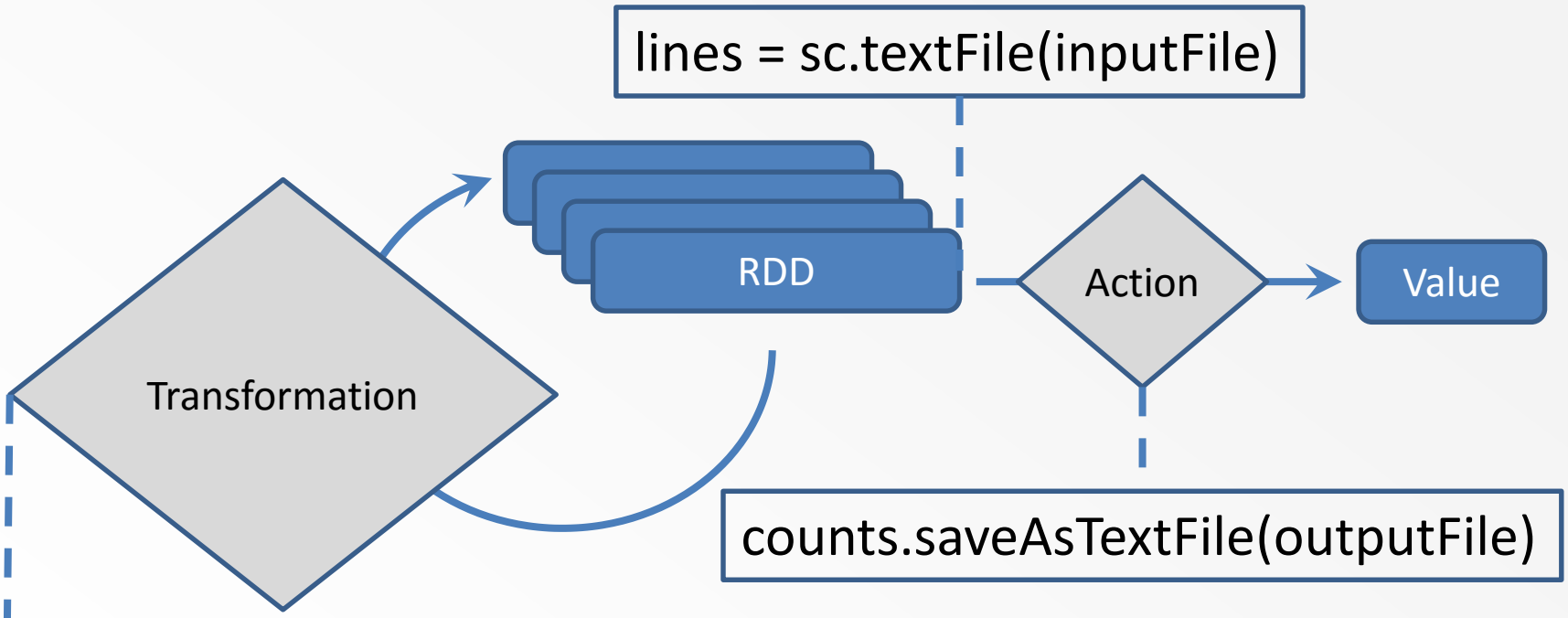
```
$ git clone https://github.com/cgeroux/big-data-workshop.git

$ cd big-data-workshop/spark/wordcount
$ make run
```

View running jobs at: http://206.167.181.71:8088/

```
$ hdfs dfs -cat <input file>
$ make showoutput
```

# Spark RDDs




```
words= lines.flatMap(lambda x: x.split(‘ ‘) )
mappedWords=words.map(lambda x: (x,1))
counts=mappedWords.reduceByKey(add)
```

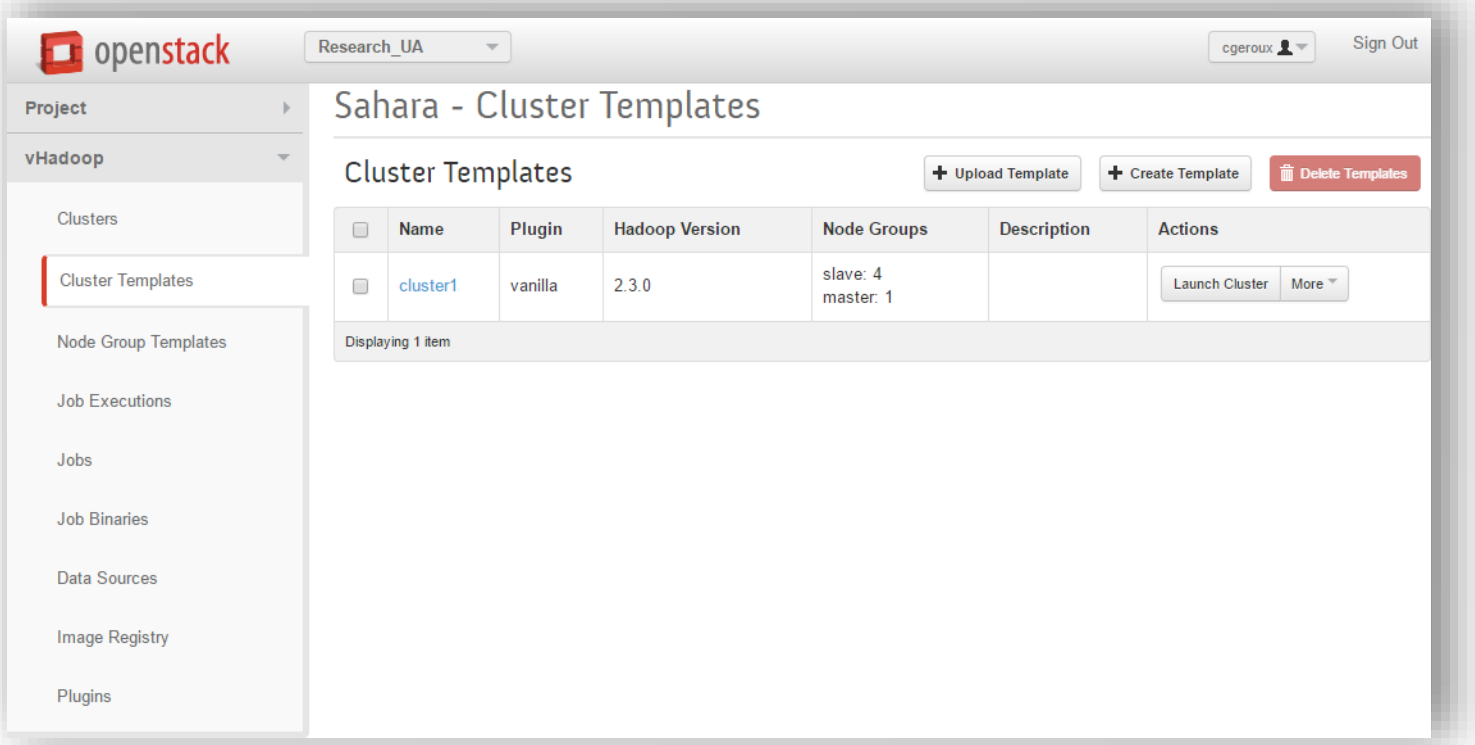
# Other Big Data Tools

## not an exhaustive list

- Hbase
  - Stores data as key/value pairs in tables within HDFS
  - Allows random read/write
  - Supposedly real time
  - Can be used with MR
- Hive
  - Works on top of hadoop
  - Provides SQL-like queries
    - At command line (MR under the hood)
    - Or from within MR
  - Can work with Hbase tables
- Pig
  - PigInterpreter and PigLatin (laungage)
  - Queries converted to MR jobs
  - Works with HDFS and Hbase
- Sqoop
  - Data transfer between relational databases and Hadoop
    - Into Hive or Hbase
  - Also back to relational databases
- Oozie
  - Automates schedualing of jobs
    - Sqoop imports
    - Pig jobs
    - Etc.
- Flume/Chukwa
  - Data aggregation tools
  - Distributed
  - Parallel

# Sahara - OpenStack

- Cluster management and usage via OpenStack GUI
  - Create a hadoop cluster from a cluster template
  - Run MR, Pig, and Hive jobs
  - Select data sources



# Where to go from here

- Automating cluster setup:
  - https://github.com/cgeroux/cloud-init.git
  - http://docs.openstack.org/user-guide/
  - https://docs.saltstack.com/en/latest/
  - https://cloudinit.readthedocs.io/en/latest/

- Spark API
  - http://spark.apache.org/docs/latest/programming-guide.html