SETTING UP AND MANAGING APACHE HADOOP CLUSTER WITH PUPPET

ALEXEY SVYATKOVSKIY

PRINCETON UNIVERSITY

OUTLINE

- SCOPE OF THIS TALK
- PUPPET ORCHESTRATION TOOL
- CORE COMPONENTS OF HADOOP CLUSTER
- HADOOP PUPPET CLASS HIERARCHY
- HADOOP CLUSTER CONFIGURATION
- SUMMARY

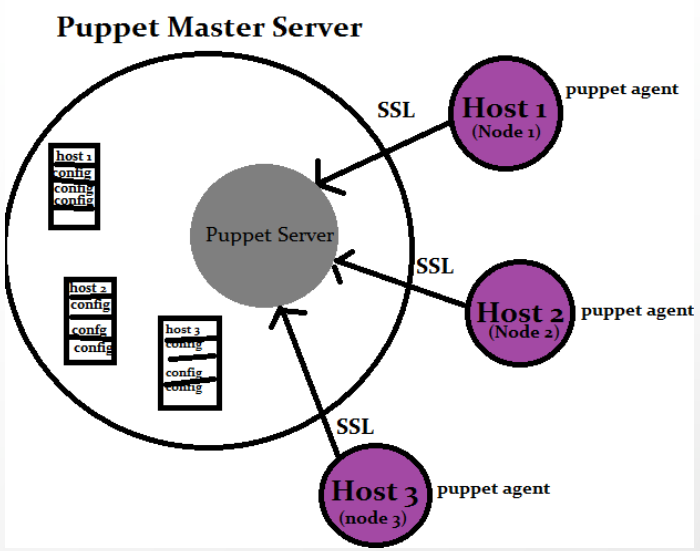
Princeton University data center

SCOPE OF THE TALK

- THE GOAL OF THIS TALK IS TO PROVIDE A BRIEF OVERVIEW OF AN EXISTING PUPPET CONFIGURATION FOR HADOOP CLUSTER
  - RELY ON CLOUDERA DISTRIBUTION OF HADOOP, BUT ALLOW FOR INCLUSION OF 3RD PARTY SOFTWARE
  - MINIMUM KNOWLEDGE OF PUPPET IS ASSUMED
- THE EXAMPLE CONFIGURATION IS FOR A 10 NODE CLUSTER CONTAINING 4 SERVICE NODES AND 6 DATA NODES IS PRESENTED AND DISCUSSED
  - ALL OF THE COMPONENTS OF CORE HADOOP ECOSYSTEM ARE SPLIT AMONG 4 SERVICE NODES
  - FOR A BIGGER CLUSTER, ONE CAN CONSIDER MORE SERVICE NODES
  - EASY TO GROW CLUSTER BY ADDING DATANODES WITH MINIMAL CHANGES IN THE CONFIGURATION

PUPPET ORCHESTRATION TOOL

- PUPPET IS A TOOL THAT ALLOWS TO KEEP UP WITH CONFIGURATION OF YOUR SYSTEM IN A PRODUCTION ENVIRONMENT
  - CONSISTENCY AMONG NODES
  - CONTINUOUS APPLICATION DEPLOYMENT (MULTIPLE TIMES A DAY)
  - AUTOMATION ACROSS NODES ON CLUSTER, LESS TIME ON REPETITIVE TASKS
  - AGENT-MASTER ARCHITECTURE. SET UP A GIT REPOSITORY ON PUPPET MASTER SERVER, COMMIT CONFIGURATION CHANGES TO THAT REPOSITORY. RUN PUPPET-AGENT SERVICES ON NODES, PERIODICALLY PUSHING CHANGES IN THE CONFIGURATION FROM MASTER TO THE AGENTS.



- ALTERNATIVES: PUPPET PROGRAMS VS SCRIPTS

Puppet programs	scripts
Standardized programming language	No code standard
Declarative	Procedural
Abstract resources common across OS: services, packages, files/directories and users	N/A
Take into account difference in package managers across OS: Yum – RedHat Apt-get - Ubuntu	N/A

- ALTERNATIVES: DOCKER. BUILD YOUR OWN DOCKER IMAGES FROM DOCKERFILES OR PULL EXISTING FROM DOCKERHUB. CONTAINERS: RUNNING INSTANCES OF DOCKER IMAGES ON A CLUSTER

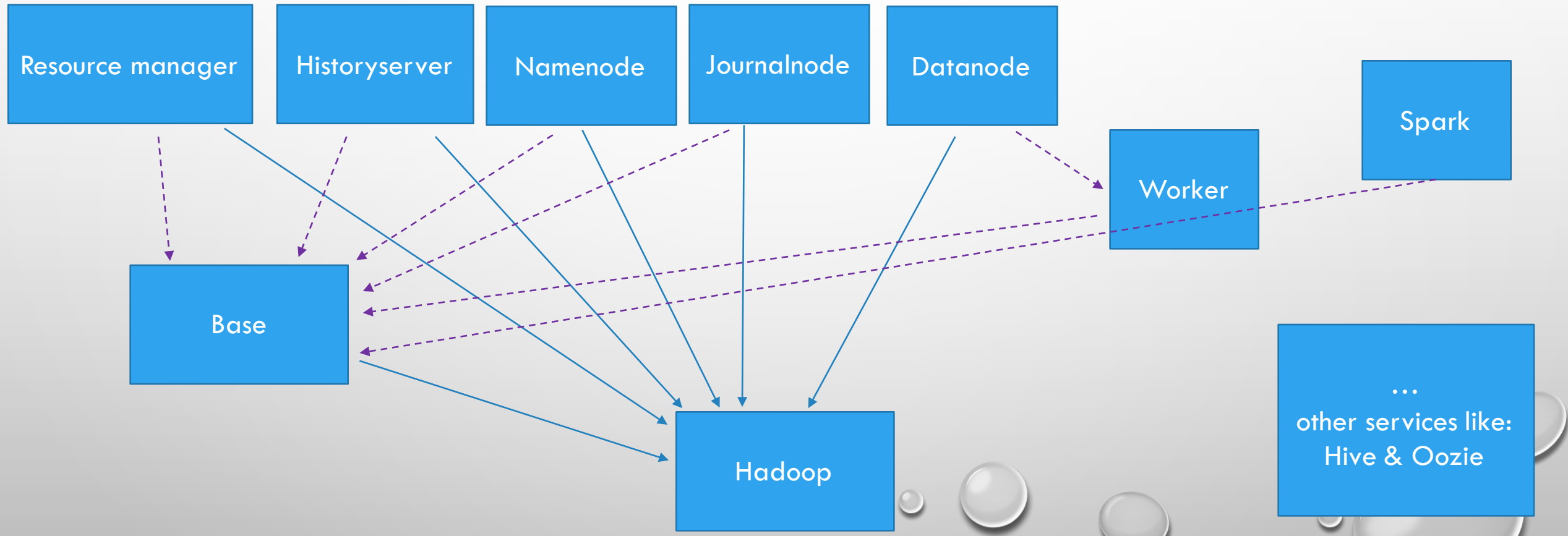
HADOOP CLUSTER COMPONENTS

- RESOURCE MANAGER – THE YARN RESOURCE MANAGER (RM) IS RESPONSIBLE FOR TRACKING THE RESOURCES IN A CLUSTER AND SCHEDULING APPLICATIONS (FOR EXAMPLE, SPARK JOBS)
- (JOB) HISTORY SERVER – YARN REQUIRES A STAGING DIRECTORY FOR TEMPORARY FILES CREATED WHILE RUNNING JOBS. ALSO, DIRECTORY FOR LOG FILES OF FINISHED JOBS
- NAMENODES – HADOOP CLUSTER FOLLOWS A MASTER-WORKER PATTERN, HERE NAMENODE IS THE MASTER TO DATANODES (WORKERS). NAMENODES MANAGE THE FILESYSTEM TREE AND METADATA. IN THE HIGH AVAILABILITY CONFIGURATION, 2 OR MORE NAMENODES IS RAN ON THE SAME CLUSTER, IN AN ACTIVE/PASSIVE CONFIGURATION. THESE ARE REFERRED TO AS THE **ACTIVE** NAMENODE AND THE **STANDBY** NAMENODE(S).
- JOURNAL NODES - IN ORDER FOR A STANDBY NODE TO KEEP ITS STATE SYNCHRONIZED WITH THE ACTIVE NODE IN THIS IMPLEMENTATION, BOTH NODES COMMUNICATE WITH A GROUP OF SEPARATE DAEMONS CALLED **JOURNALNODES**. WHEN ANY FILESYSTEM MODIFICATION IS PERFORMED BY THE ACTIVE NODE, IT LOGS A RECORD OF THE MODIFICATION TO A MAJORITY OF THESE JOURNALNODES. THE STANDBY NODE IS CAPABLE OF READING THE EDITS FROM THE JOURNALNODES, AND IS CONSTANTLY WATCHING THEM FOR CHANGES TO THE EDIT LOG. AS THE STANDBY NODE SEES THE EDITS, IT APPLIES THEM TO ITS OWN NAMESPACE. IN THE EVENT OF A FAILOVER, THE STANDBY WILL ENSURE THAT IT HAS READ ALL OF THE EDITS FROM THE JOURNALNODES BEFORE PROMOTING ITSELF TO THE ACTIVE STATE.
- DATANODE – WORKER NODE. DAEMONS RUNNING ON DATANODES INCLUDE JOB TRACKER AND HDFS
- OTHER (OPTIONAL) CLUSTER COMPONENTS INCLUDE: SPARK, HIVE, OOZIE, HBASE, HUE…

Require declared

Inheritance

HADOOP (PUPPET) CLASS HIERARCHY



PUPPET CLASS EXAMPLES: CLASS HADOOP, HADOOP::BASE

init.pp

```
class hadoop (
  # java defaults
  $java_package,
  $java_home,
  # name of the cluster,
  $cluster_name,
  # -Xmx for NameNode and DataNode.  Default: undef
  $hadoop_heapsize = undef,
  # shall we do high availability for namenodes?
  $namenode_ha = false,
  # array of namenodes
  $namenode_server,
  # Any additional opts to pass to NameNode node on startup
  $hadoop_namenode_opts = undef,
  # array of paths for hadoop namenode directory
  $dfs_namenode_name_dir,
  $namenode_jmxremote_port = 9980,
  # history server
  $history_server,
  # array of journal servers, used in case of HA
  $journalnode = undef,
  # where journal server will store data
  $dfs_journalnode_edits_dir = undef,
  # Path relative to JBOD mount point for HDFS data directories
  $dfs_data_path = 'hdfs/dn',
  # array with data mount directories
  $datanode_mounts,
  # default block size, 64MB is the default
  $dfs_block_size = 67108864,
  # Boolean which enables backend datanode-side support for the experimental
  # DistributedFileSystem#getFileVBlockStorageLocations API..
  # This is required if you want to use Impala.
  $dfs_datanode_hdfs_blocks_metadata_enabled = false,
  $datanode_jmxremote_port = 9981,
  # Data node interface from which to report IP address,
  $datanode_interface = undef,
  # Comma seperated interfaces for client to use or IP/range
  $client_local_interfaces = undef,
  # enable webhdfs?
  $dfs_webhdfs_enabled = false,
  # httpfs, on by default it seems.
  $httpfs_enabled = true,
```

…continued

Base.pp

```
class hadoop::base inherits hadoop {
  File {
    owner => 'root',
    group => 'root',
    mode  => '0644'
  }

  # before we get started make sure java is setup
  package { $::hadoop::java_package:
    ensure => installed
  }->
  package { 'bigtop-utils':
    ensure => installed
  }->
  file_line { 'Java home':
    path  => '/etc/default/bigtop-utils',
    line  => "export JAVA_HOME=${::hadoop::java_home}",
    match => '^(# )?export JAVA_HOME(=.*|$)'
  }->
  # we require these packages, at the minimum
  package { ['hadoop-client','zookeeper']:
    ensure  => installed,
    require => Package[$::hadoop::java_package]
  }->
  file { $::hadoop::config_directory:
    ensure => 'directory',
    mode   => '0755'
  }->
  hadoop::alternative { 'hadoop-conf':
    link => '/etc/hadoop/conf',
    path => $::hadoop::config_directory,
  }->
  hadoop::log4jproperties { "${::hadoop::config_directory}/log4j.properties":
    values  => $::hadoop::log4j_properties,
  }

  file { "${::hadoop::config_directory}/core-site.xml":
    content => template('hadoop/core-site.xml.erb'),
  }

  file { "${::hadoop::config_directory}/hdfs-site.xml":
    content => template('hadoop/hdfs-site.xml.erb'),
  }
```

…continued

PUPPET CLASS HADOOP::NAMENODE

Namenode.pp

```
class hadoop::namenode($primary=true) inherits hadoop {
  Class['hadoop::base'] -> Class['hadoop::namenode']

  if $primary {
    $format_command = '/usr/bin/hdfs namenode -format -nonInteractive'
  } else {
    # bootstrap standby dfs.name.dir from primary active NameNode
    if !$::hadoop::namenode_ha {
      fail('Cannot use Standby NameNode in a non HA setup.')
    }
    $format_command = '/usr/bin/hdfs namenode -bootstrapStandby -nonInteractive'
  }

  # install namenode daemon package
  package { 'hadoop-hdfs-namenode':
    ensure => 'installed',
  }->
  # Ensure that the namenode directory has the correct permissions.
  file { $::hadoop::dfs_namenode_name_dir:
    ensure => 'directory',
    owner  => 'hdfs',
    group  => 'hdfs',
    mode   => '0700',
  }->
  # If $dfs_name_dir/current/VERSION doesn't exist, assume NameNode has not
  # been formated.  Format it before the namenode service is started.
  exec { 'hadoop-namenode-format':
    command => $format_command,
    creates => "${::hadoop::dfs_namenode_name_dir_main}/current/VERSION",
    user    => 'hdfs',
  }->
  service { 'hadoop-hdfs-namenode':
    ensure     => 'running',
    enable     => true,
    hasstatus  => true,
    hasrestart => true,
  }
```

```
if $primary {
  if $::hadoop::namenode_ha {
    # primary NameNode will be transitioned to active once NameNode
    # has been formatted, before common HDFS directories are created.
    exec { 'haaadmin-transitionToActive':
      command     => "/usr/bin/hdfs haadmin -transitionToActive ${::hadoop::primary_namenode_id}",
      unless      => "/usr/bin/hdfs haadmin -getServiceState    ${::hadoop::primary_namenode_id} | /bin/grep -q active",
      user        => 'hdfs',
      # Only run this command if the namenode was just formatted
      # and after the namenode has started up.
      refreshonly => true,
      subscribe   => Exec['hadoop-namenode-format'],
      require     => Service['hadoop-hdfs-namenode'],
    }
    # Make sure NameNode is running and active
    # before we try to create common HDFS directories.
    Hadoop::Directory {
      owner   => 'hdfs',
      group   => 'hdfs',
      mode    => '0755',
      require => Exec['haaadmin-transitionToActive'],
    }
  } else {
    # Make sure NameNode is running
    # before we try to create common HDFS directories.
    Hadoop::Directory {
      owner   => 'hdfs',
      group   => 'hdfs',
      mode    => '0755',
      require =>  Service['hadoop-hdfs-namenode'],
    }
```

...continued

DESIGNING A PUPPET CONFIGURATION FOR HADOOP CLUSTER

- THE GOAL IS TO IDENTIFY ALL OF THE SERVICES AND PACKAGES THAT NEED TO BE INSTALLED ON THE NODES, CREATE NECESSARY FILES AND DIRECTORIES (WITH PROPER PERMISSIONS) AND CREATE USER ACCOUNTS
  - FOR COMPLEX CLUSTERS LIKE HADOOP, SERVICES NEED TO BE STARTED IN A CERTAIN ORDER
- THERE IS A SET OF COMMON SERVICES/PACKAGES THAT WILL GO TO ALL NODES: SSHD, PUPPET-AGENT, YP-CLIENT, LINUX CORE COMPONENTS AS WELL AS HADOOP::BASE COMPONENTS
  - WE PUT IT INTO COMMON.PP MANIFEST
  - OTHER NODE SPECIFIC SERVICES GO INTO DEDICATED MANIFESTS: NAMENODE.PP, DATANODE.PP ETC
  - .ERB FILE TEMPLATES ARE USED FOR CONFIGURATION FILES AND ARE LOADED FROM MANIFESTS
  - HIERA .YAML FILES ARE USED TO EASILY LOAD THE NECESSARY MANIFESTS INTO A DEDICATED CATALOG FOR EACH NODE

HEADNODE

Headnode

classes:
- hadoop::hue
- hadoop::mount
- hadoop::spark
- cses::yp
- cses::ldap::auth
- cses::users
- cses::nfs::server
- cses::cluster::buildacct

Hadoop services:

- HUE – set of web applications that enable to interact with a cluster: submit MapReduce or Spark jobs, define Oozie workflows
- mount – filesystem userspace (FUSE) interface into HDFS. FUSE allows to use traditional linux POSIX compliant commands in applications.
- Spark

Example: class mount

```
class hadoop::mount(
  $mount_point = '/hdfs',
  $read_only   = false,
  $user_dir    = true
) inherits hadoop {
  Class['hadoop::base'] -> Class['hadoop::mount']

  package { 'hadoop-hdfs-fuse':
    ensure => 'installed',
  }

  $device = $::hadoop::namenode_ha ? {
    true    => "hadoop-fuse-dfs#dfs://${::hadoop::nameservice_id}",
    default => "hadoop-fuse-dfs#dfs://${::hadoop::primary_namenode_server}:8020",
  }

  $options = $read_only ? {
    true    => 'ro,allow_other,usetrash,big_writes',
    default => 'rw,allow_other,usetrash,big_writes',
  }
```

Non-hadoop services

- YP and LDAP to manage access to the cluster
- NFS server: NFS partition mounted across all nodes, like a /scratch partition for people using small files, and home directories
- Build account: functions to build user accounts and create home directories

SERVICE NODE 1

Service node 1

classes:
- zookeeper
- hadoop::journalnode
- hadoop::namenode
- hadoop::httpfs

- Zookeeper is a high-performance coordination service for distributed applications. It exposes common services such as naming, configuration management, synchronization, and group services in a simple interface.
- Journal nodes
- Namenode – active namenode
- HTTPFS server provides a REST HTTP API supporting all HDFS File System operations.

SERVICE NODES 2-3

Service node 2

classes:
- zookeeper
- hadoop::hive::master
- hadoop::journalnode
- hadoop::resourcemanager

Besides Zookeeper, and other core components of Hadoop the Hive master is running on the service node 2

Service node 3

classes:
- zookeeper
- hadoop::journalnode
- hadoop::namenode
- hadoop::historyserver
- hadoop::oozie::base
- hadoop::oozie::server

hadoop::namenode::primary : false

In addition to Zookeeper and history server, the standby namenode for failover is running on the service node 3, as well as the Oozie server

HADOOP/SPARK CLUSTER CONFIGURATION

Datanodes

classes:
- hadoop::spark
- hadoop::worker
- cses::nfs::client

There are five Spark packages:
spark-core: delivers core functionality of spark

- spark-worker: init scripts for spark-worker
- spark-master: init scripts for spark-master
- spark-python: Python client for Spark spark-history-server

The **spark-core, spark-worker** and **spark-python** go on all the cluster datanodes and the headnode
We do not need spark-master or spark-history-server when running Spark with YARN

SUMMARY

- PUPPET CONFIGURATION FOR THE HADOOP CLUSTER BASED ON THE CLOUDERA DISTRIBUTION OF HADOOP (CDH 5) FOR A CLUSTER HAVING 4 SERVICE NODES IS PRESENTED AND DISCUSSED
  - EXISTING PUBLICALLY AVAILABLE ALTERNATIVE IS WIKIMEDIA'S PUPPET-CDH: HTTPS://GITHUB.COM/WIKIMEDIA/PUPPET-CDH
  - APPROPRIATE ONLY FOR VERY SMALL HADOOP CLUSTERS, AND HAS NUMBER OF SHORTCOMINGS WHEN USED WITH LARGER CLUSTERS
- PRINCETON PUPPET HADOOP CLUSTER CONFIGURATION CAN BE SHARED WITH OTHER COLLABORATORS AT CMS/DIANA-HEP IF NEEDED