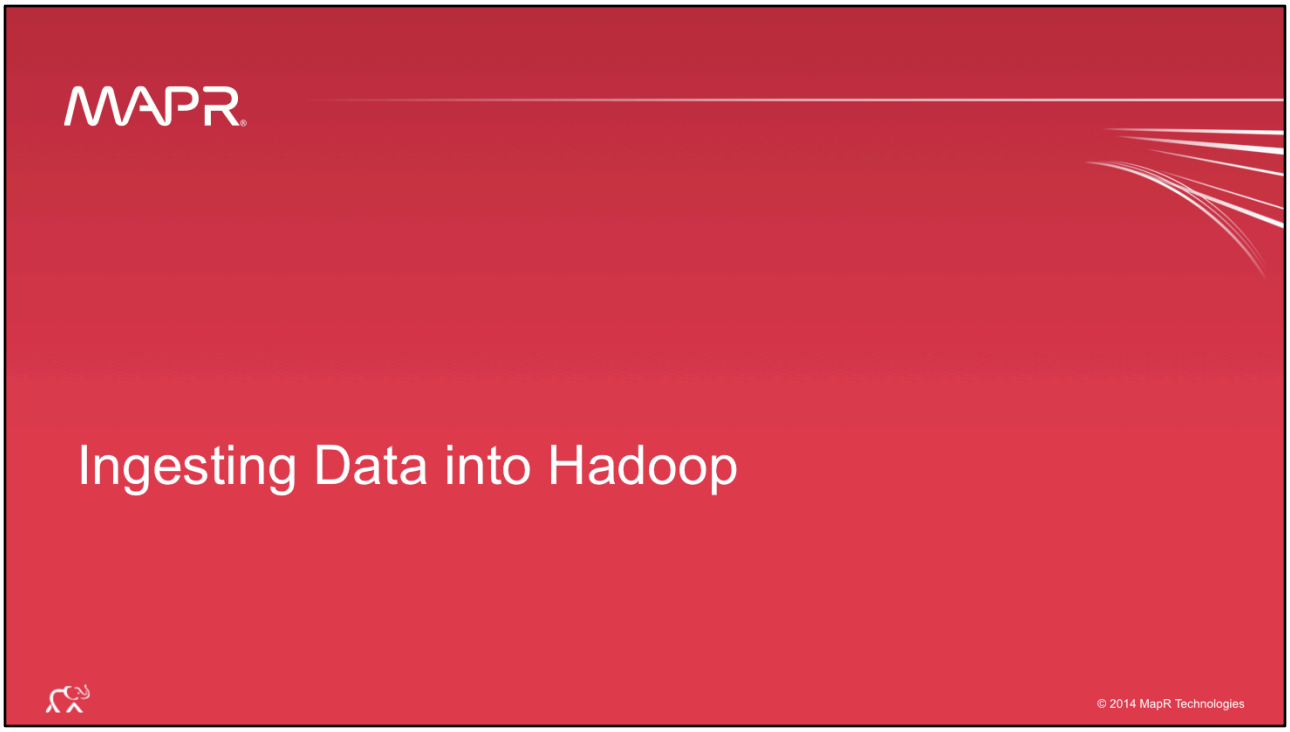


This lesson is a short introduction to ingesting data into Hadoop.

Discuss the Motivation for Apache Flume

Data comes in all shapes and sizes. Furthermore, some data is like a river or stream – it just keeps flowing in. Examples of streaming data are shown in the slide above, including Syslog logs, Web server logs, Facebook data, and Twitter feeds.

You Can Write Scripts to Import the Data

```
#!/usr/bin/python
import twitter
pyPass = 'anAwesomePassword'
api = twitter.Api(consumer_key='gibberish', consumer_secret='moregibberish', access_token_key='evenlongergibberish', access_token_secret='thelastbitofgibberish')
#print api.VerifyCredentials()
#Twitter users with interesting links
feedList = ['binaryghost', 'viticci', 'marksiegal', 'ttscoff', 'gromble', 'drbunsen', 'waltonjones', 'TJLuoma', 'eddie_smith', 'chewingpencils', 'jeffhunsberger', 'nateboateng']
# The adventure begins
for user in feedList:
    # Get all of the user tweets
    statuses = api.GetUserTimeline(user)
    # Loop through all tweets and look for URLs
    for status in statuses:
        # A reasonably generous URL regex pattern to match
        urls = re.findall('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', status.text)
        # Everytime we find a URL, do some magic
        if urls:
            for url in urls:
                # Put together the tweet info
                tweetMsg = status.text
                rssTag = ['rss_'+user, 'rss_tweets']
                sourceURL = 'https://twitter.com/'+user+'/status/'+str(status.id)
                try:
                    # Let's make sure the URL is valid by trying to visit it
                    request = urllib2.Request(url)
                    request.add_header('Accept-encoding', 'gzip')
                    response = urllib2.urlopen(request)
                    data = response
                    maintype = response.headers['Content-Type'].split(';')[0].lower()
                    # We don't want image links
                    if maintype not in ('image/png', 'image/jpeg', 'image/gif'):
                        # Need to handle gzip content if we want to grab the page title. Lots of sites send gzip now.
                        if response.info().get('Content-Encoding') == 'gzip':
                            buf = StringIO(response.read())
                            f = gzip.GzipFile(fileobj=buf)
                            data = f.read()
                        # I guess I just prefer a real link to shitty t.co links
                        fullURL = response.url

                        if fullURL is not None:
                            print fullURL
                            # This could be done with lxml but BeautifulSoup is easy
                            soup = BeautifulSoup(data)
                            # Get the title for the page
                            myTitle = soup.html.head.title
                            pyTitle = myTitle.string
                            print pyTitle
                            # Assemble the bookmark notes. Create Twitter link for RSS viewing
                            bookmarkExtended = '<p>'+user+'</p>\n<p>' + tweetMsg + '</p>\n\n' + '<a href="'+sourceURL+'">Twitter Source</a>'
                            try:
                                p = pinboard.open(pyAccount, pyPass)
                                postResult = p.add(url=fullURL, description=pyTitle, extended=bookmarkExtended, tags= (rssTag))
                            except (RuntimeError, TypeError, NameError):
                                print RuntimeError
                                print NameError
                                print TypeError
                except urllib2.HTTPError, err:
                    # If it's an http error like 404, just skip the link. We don't have time for this junk.
                    continue
```

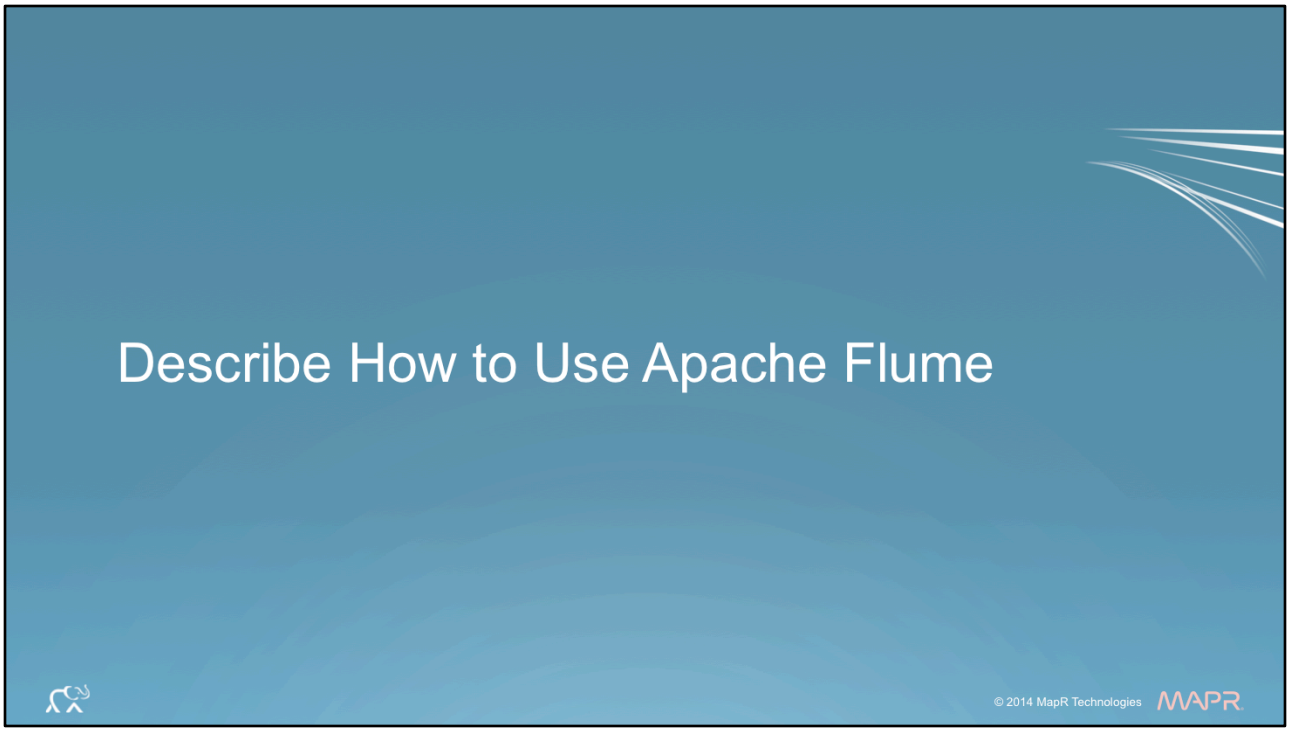
This is just for twitter

In order to process the data, you have to import it into your cluster. The screen shot aboves shows an example of a script that is used to import Twitter data for a single account.

But Using Scripts Has Issues for Streaming Data

- Each data source has a unique API
- How often should you copy streamed data files into the cluster?
- What happens if the script fails?
- Scripts can't be parallelized

There are issues when using scripts for importing streaming data into your cluster. Each data source (Twitter, Web server, …) has it's own unique API or method for importing data, which means you'll end up writing lots of scripts. One question when importing data that is of the streaming type: how often should you copy files into the cluster? Every minute? Every five minutes? Once per day? Also, what happens if your script fails? How can you tell that it failed? Last, scripts run on a single machine are single-threaded which means they cannot be parallelized.



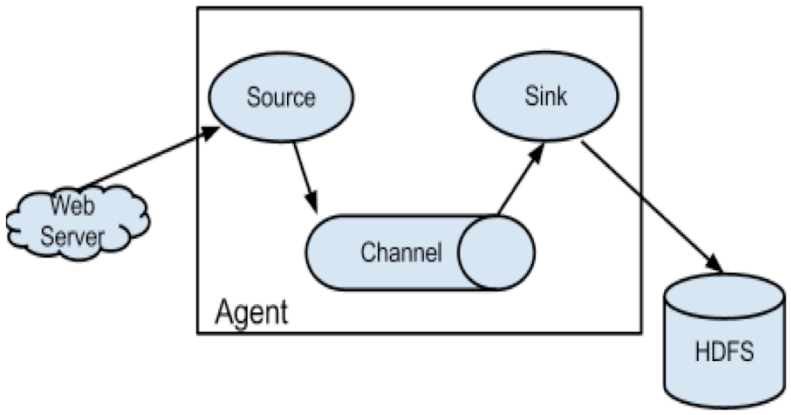
This section provides a brief description on how to use Apache Flume.

What is Flume?

- Reliable, scalable service that collects large amounts of streaming data
- Tunable reliability features
- Agent-based service that includes:
  - Source
  - Sink
  - Channel

Flume is a reliable, scalable service you can use to collect streaming data in your Hadoop cluster. You can configure Flume to identify and react to failures while importing data, which unburdens you from having to implement this failure logic yourself. Flume runs as an agent and includes a source (of streaming data), sink (destination to store streaming data), and a channel (mechanism and logic for getting the data from source to sink).

Flume Architecture



The diagram above depicts the Flume architecture for a Web server's streaming log data. The Web server may be configured to either write to log files which the Flume agent can read, or the Web server can use "piped logging" to write to Flume directly. In either case, the agent will connect this data source to a channel (or set of channels) you configure to:

1. Transform the data
2. Fan a single data source out to multiple sinks
3. Fan multiple data sources in to a single sink

Ultimately, Flume will store the data on HDFS into a configurable directory hierarchy (usually based on source and timestamps).

Flume Use Case Patterns

Use Case	Example
Multi-hop	Sink in one flow becomes source for another flow
Fan-in	Consolidate multiple flows into a single channel
Multiplex	Selectively route an event to one or more channels

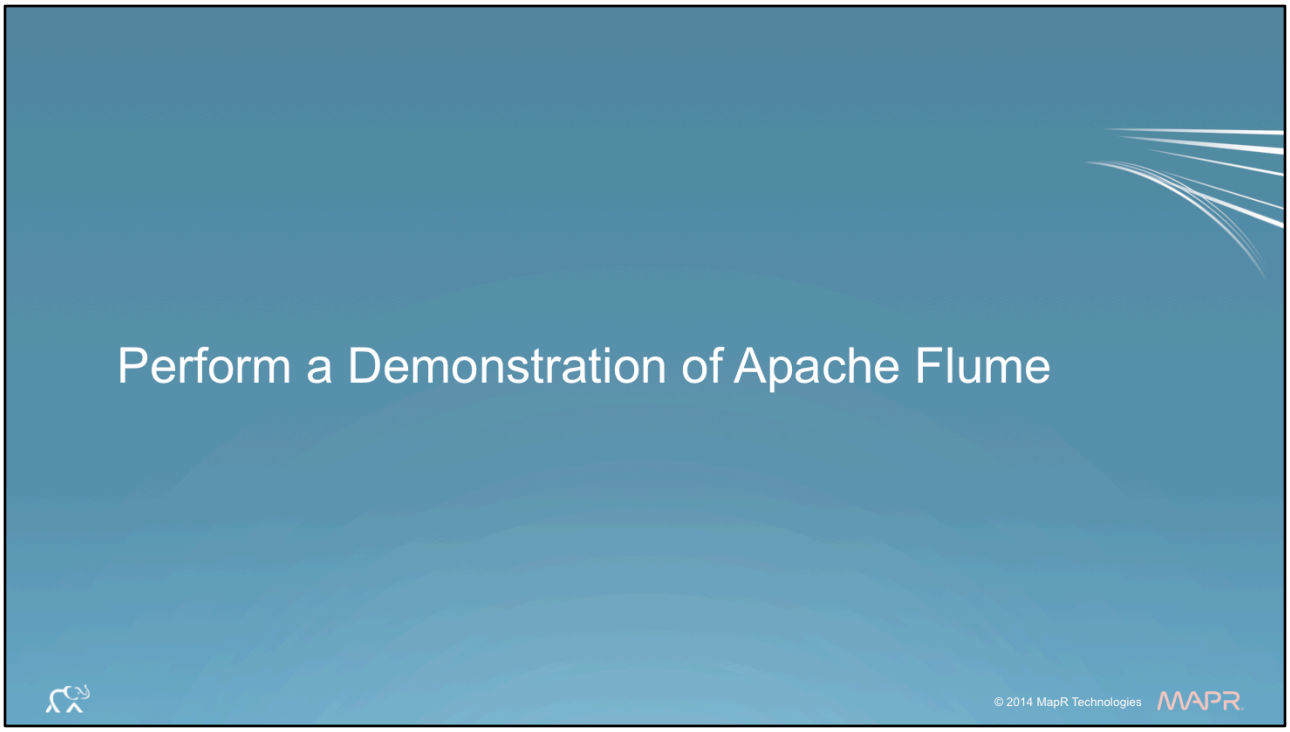
 9

The table above identifies the 3 main uses case “patterns” for Flume. One use case is “multi-hop” where multiple source-channel-sink configurations are plumbed together in a series. A second use case is called “fan-in” where your objective is to take multiple streaming data sources and converge them into a single source. The third use case is called “multiplex” where you wish to select different channels for each event based on user-defined properties (e.g. event type).

References for Flume

- http://doc.mapr.com/display/MapR/Flume
- http://flume.apache.org
- Using Flume (O'Reilly)

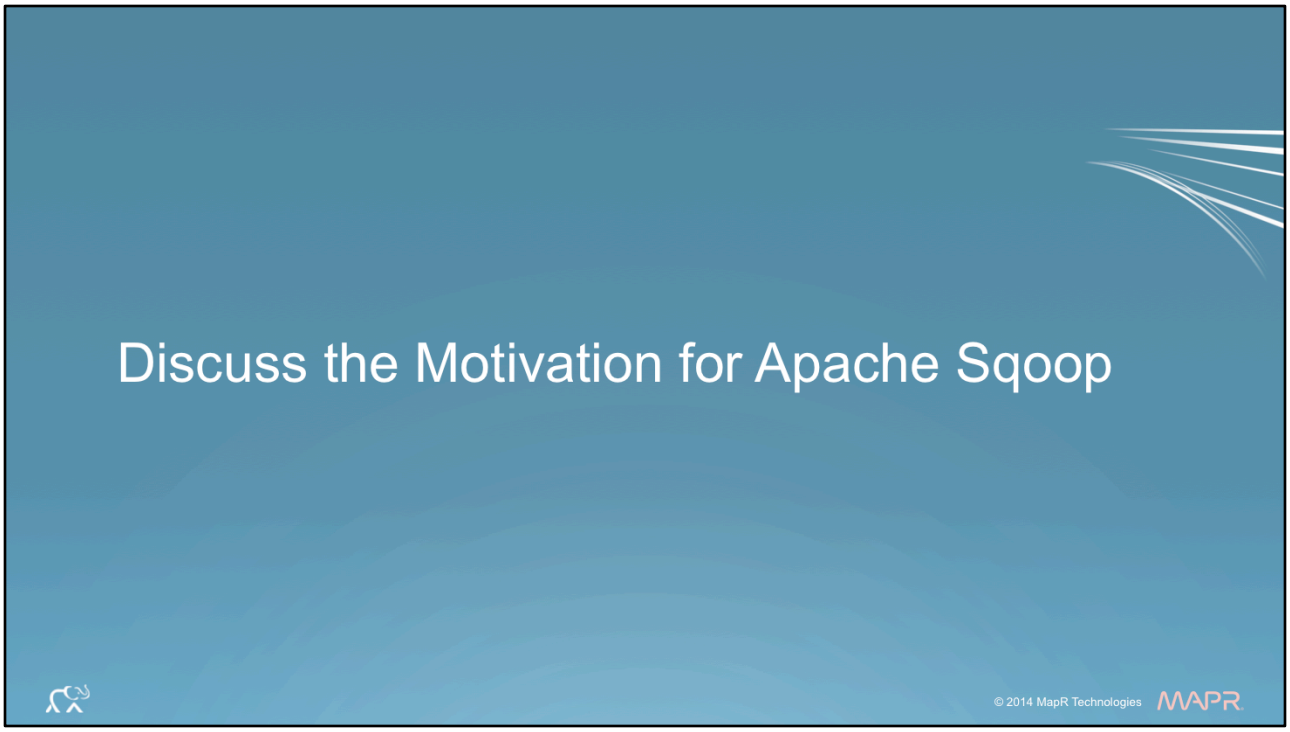
The slide identifies some references for your to start using Apache Flume. There are many more resources available for you, though. Just google it!



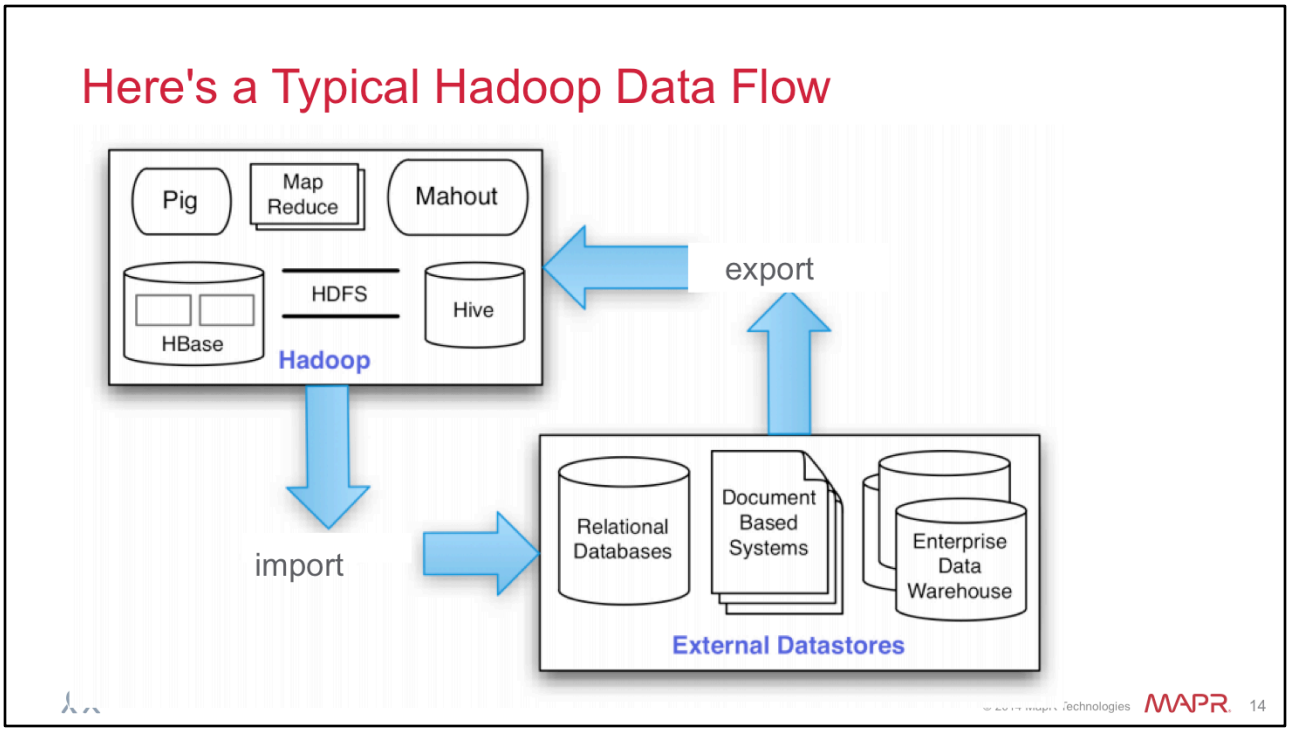
In this section, we perform a very brief demonstration of Apache Flume.

Demo: Using Flume for Syslog

In this demonstration, we will ingest streaming data from a syslog service and write it to our Hadoop file system.

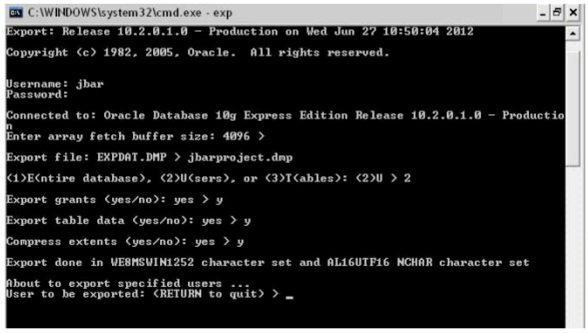


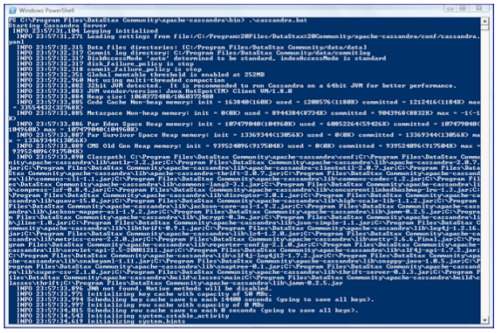
This section provides a brief motivation for Apache Sqoop.



A typical data flow in a Hadoop cluster is shown in the slide above. Data from various sources are ingested from external data sources like RDBMS, document data stores, and EDWs. The Hadoop cluster processes, transforms, and analyzes this data using a variety of tools like Pig and Hive. These results may then, in turn, be imported to external data stores for future reference.

You Can Manually Import and Export Data





How do you get the data in and out of Hadoop and these external data stores? One way is to do it manually. Every data store has some way to import and export data.

But Manual Solutions Have Inherent Issues

- Import/export procedures for each type of data store is unique
- Errors don't fail gracefully
- Manual solutions are difficult to
  - Automate
  - Make robust
  - Make performant

However, you will encounter issues trying to manually import and export data between your Hadoop cluster and external data stores. For one, these procedures are unique to the data store, so every data store requires its own set of procedures. Errors don't fail gracefully when importing and exporting data. Your procedures will need to understand failure detection and failure recovery. In general, manual solutions are difficult to automate, make robust, and make perform well.

This section provides a brief description on how to use Sqoop.

What is Sqoop?

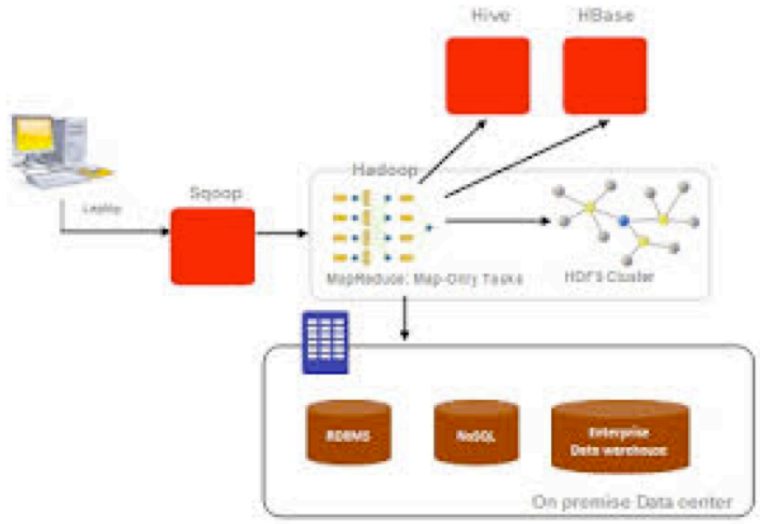
- Command-line tool for transferring data to/from:
  - structured data stores
  - HDFS
- Uses MapReduce for parallelization and fault tolerance
- Transfers may be bulk or incremental
- Integrates into Apache Oozie workflows

Sqoop is a command-line utility that you can use to transfer data between external data stores and your Hadoop cluster. The external data stores include both RDBMS and NoSQL types. Within the Hadoop cluster, you can store data on HDFS, in Hive tables, or in Hbase tables.

Sqoop uses MapReduce for parallelization and fault tolerance. Your transfers may be in bulk or incremental. Sqoop integrates in Apache Oozie workflows.

The only requirement for using an external data store with Sqoop is a JDBC connector.

Sqoop Architecture



The diagram above describes the architecture of Apache Sqoop. End-users can invoke sqoop from any client (including their laptop). Sqoop then calls map-only MapReduce jobs to import and/or export data between the Hadoop cluster and external data stores.

Use Cases for Sqoop

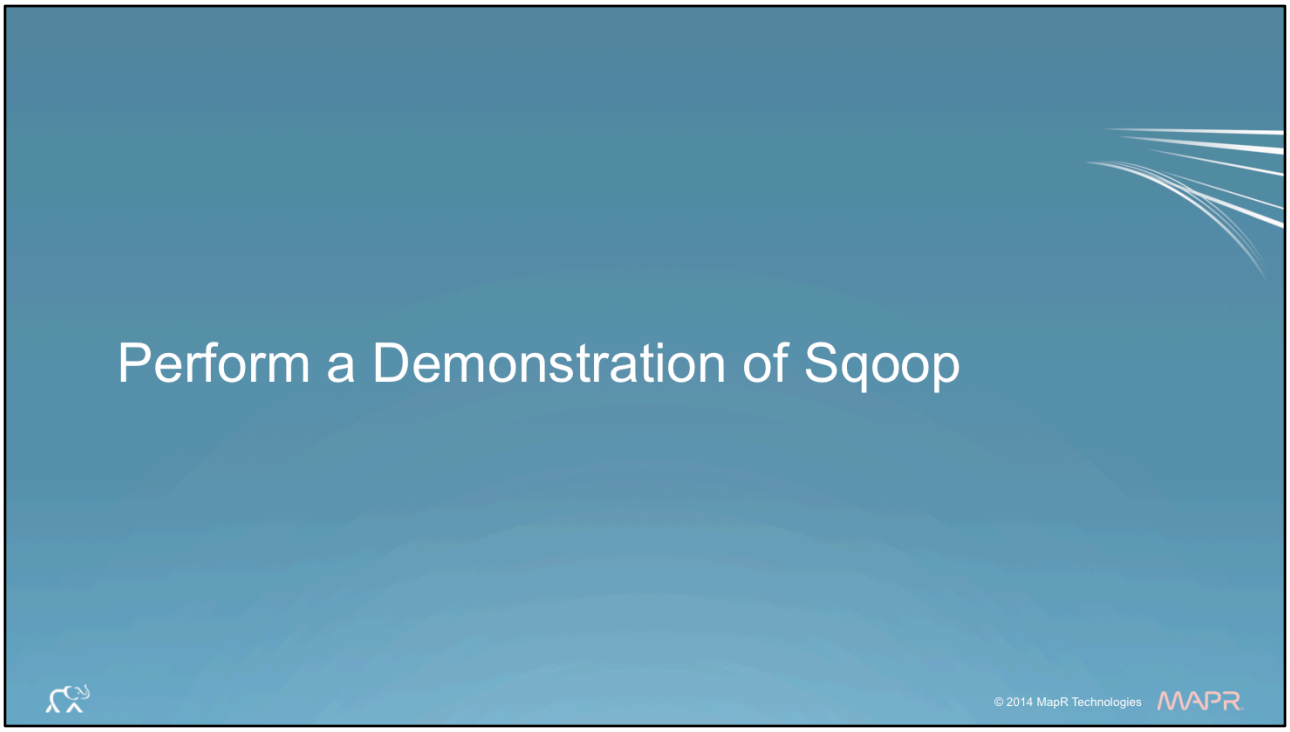
Use Case	Example
Bulk import/ export	Import/export a single table or all tables between a data store and HDFS
Incremental import/export	Import only new rows from a table since last import between HDFS and data store

Sqoop can be used in a few unique ways, as described above. One way to use Sqoop is to perform a bulk import/export of a table or set of tables between Hadoop and an external data store. The other way to use Sqoop is to perform incremental imports/exports such that only the changed data is transferred. Usually, users will perform a bulk import/export operation to start with, and then use incremental import/export on an on-going basis after that.

References for Sqoop

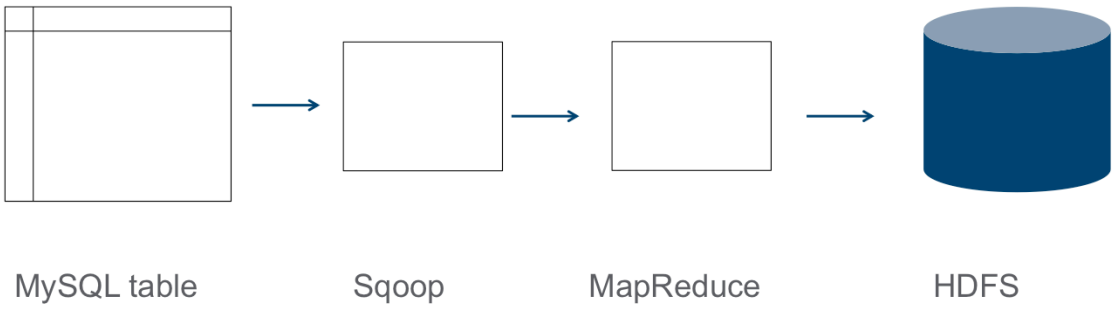
- http://doc.mapr.com/display/MapR/Sqoop
- http://sqoop.apache.org
- Apache Sqoop Cookbook (O'Reilly)

The slide above identifies some references for you to start using Sqoop. There are lots more resources available to you for Sqoop, though. Just google it!

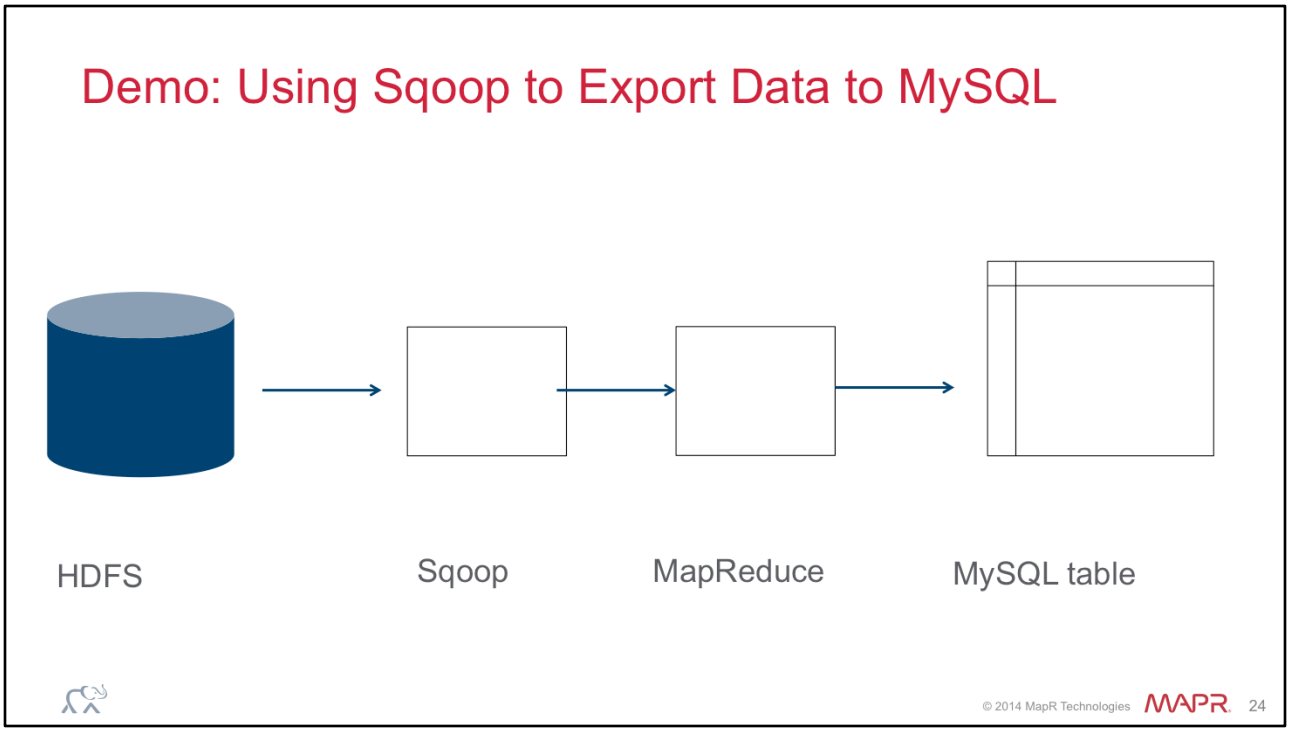


This section performs a brief demonstration of Sqoop.

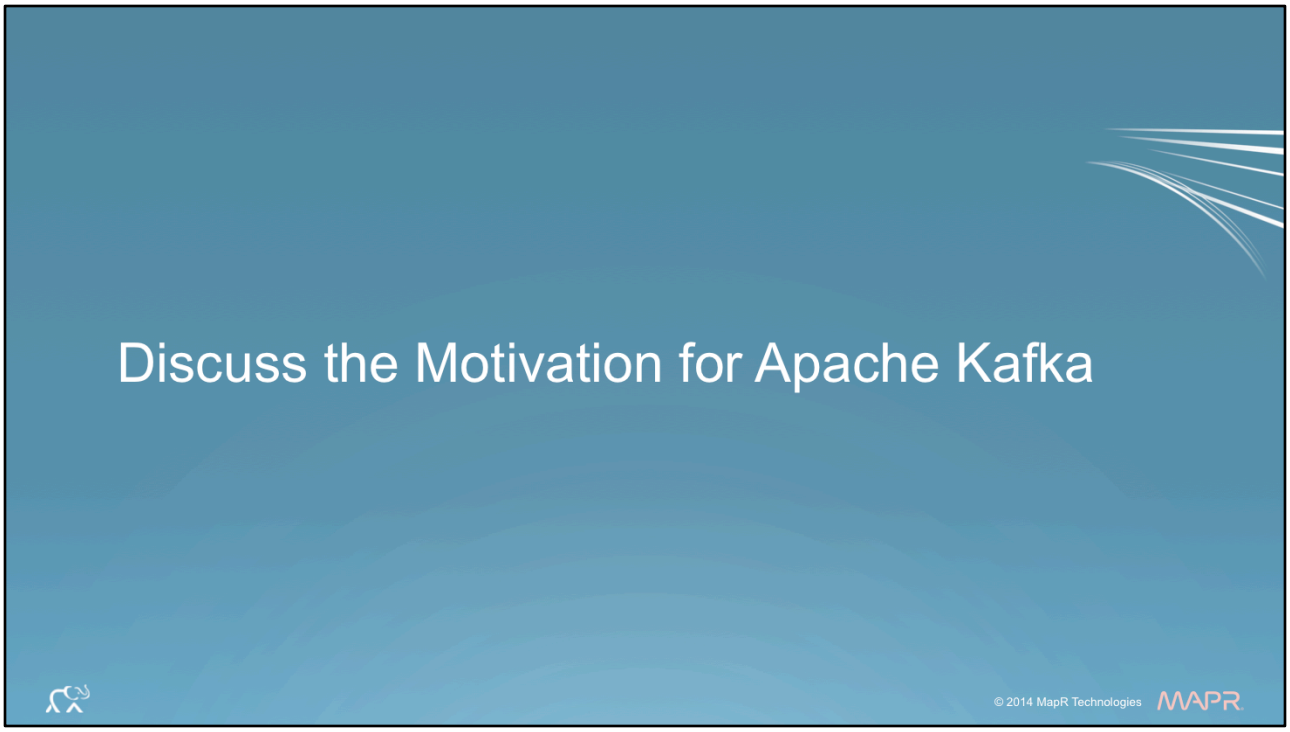
Demo: Using Sqoop to Import Data From MySQL



The first demonstration uses Sqoop to import data from a MySQL database into HDFS.



The second demonstration uses Sqoop to export data to MySQL from HDFS.

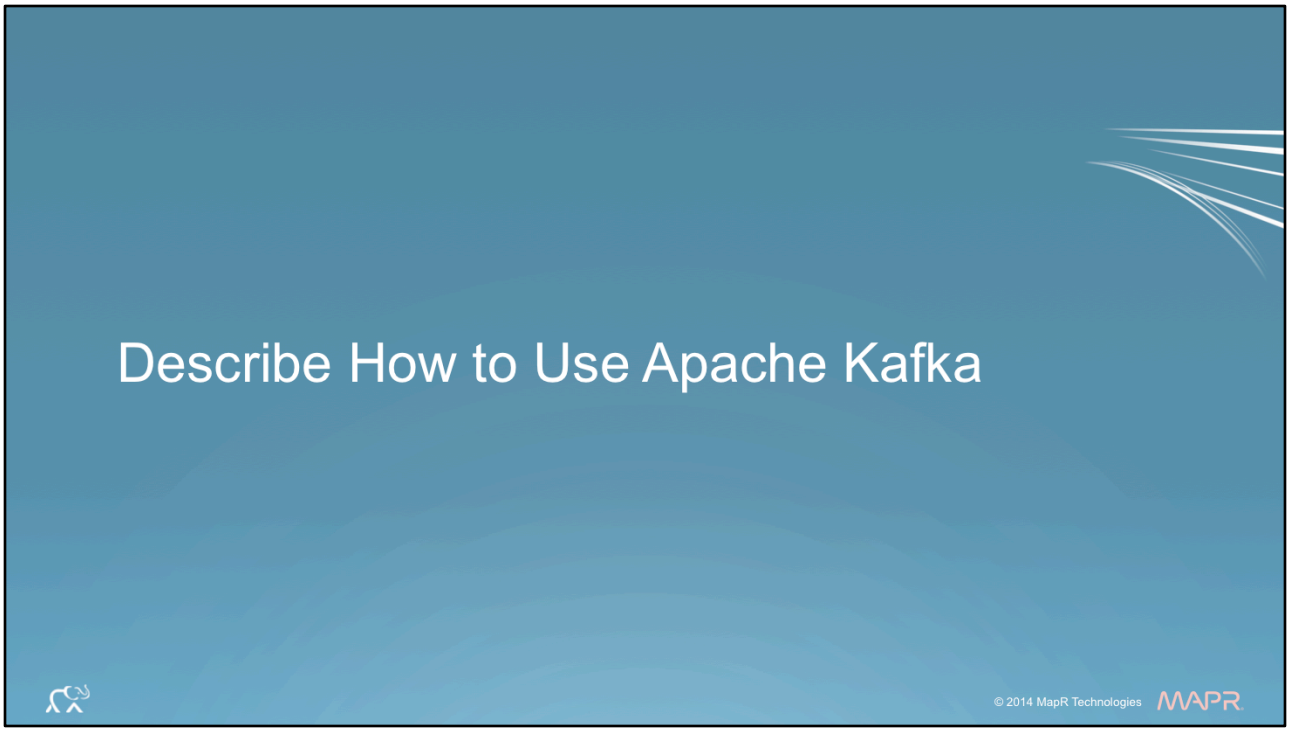

Discuss the Motivation for Apache Kafka
© 2014 MapR Technologies

Here's a Bunch of Streaming Data You Want to Use

- Messaging (decouple message delivery from processing)
- Web site activity tracking (monitor Web site activity in real time)
- Operational metrics (monitor systems real-time)
- Log aggregation (replace existing log aggregation tools)
- Stream processing (transform raw streaming data into cooked data)
- Event sourcing (time series analysis)
- Commit log (distributed system state and configuration)

But each use case requires its own tool

- Messaging (rabiitmq)
- Web site activity tracking (google analytics)
- Operational metrics (ganglia)
- Log aggregation (apache flume)
- Stream processing (apache storm)
- Event sourcing (async hbase)
- Commit log (apache bookeeper)

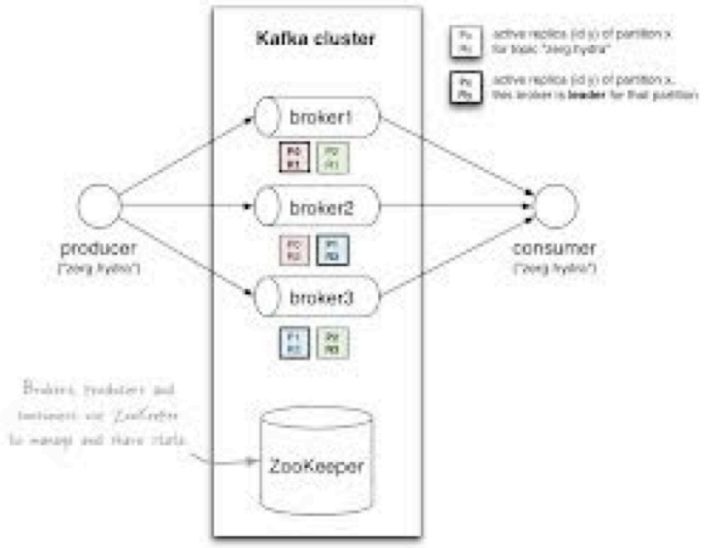


This section provides a brief description on how to use Apache Kafka.

What is Kafka?

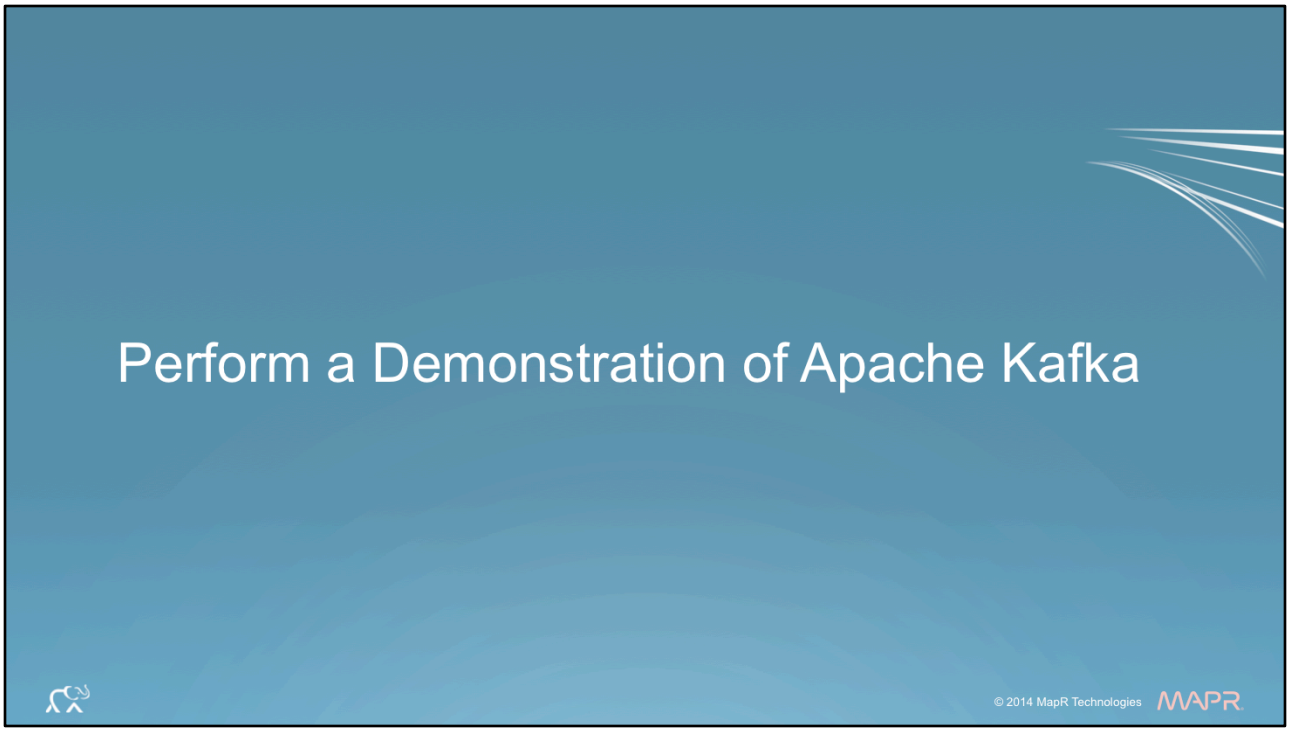
- Distributed, partitioned, replicated commit log service
- Tunable reliability features
- Publish-subscribe service:
  - producers
  - consumers
  - topics

Kafka Architecture



References for Kafka

- http://kafka.apache.org
- http://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
- http://en.wikipedia.org/wiki/Apache_Kafka



In this section, we perform a very brief demonstration of Apache Flume.

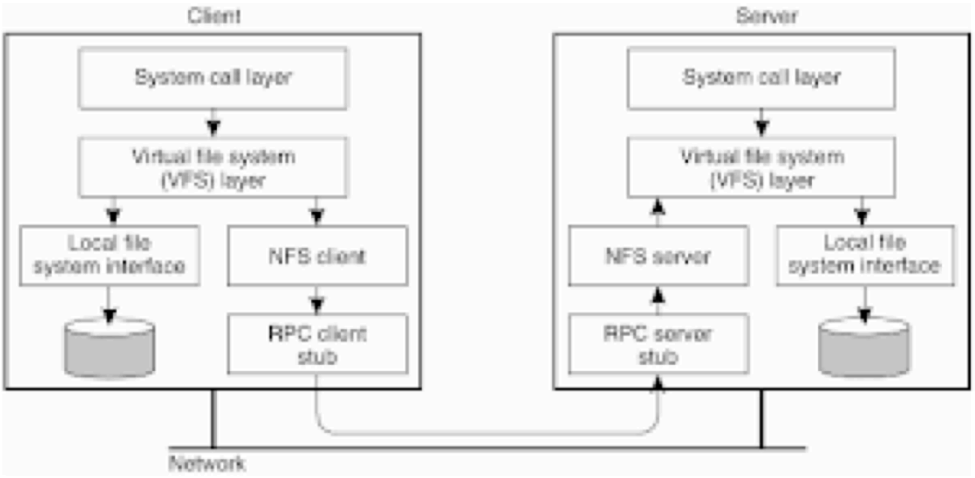
Demo: Using Kafka with Syslog and Spark Streaming


NFS
© 2014 MapR Technologies

What is NFS?

- NFS = network file system
- POSIX compliant
- Supported across all primary operating systems including Windows
- Tried and true, proven architecture

NFS Architecture



How to Configure NFS

- NFS server:
  /etc/sharetab
  /sharedir (nosuid, root_squash)

  exportfs, share, showmount -e

- NFS client:
  /etc/fstab
  nfs-server:/sharedir /mountdir nfs hard,intr,nolock 0 0

  mount, df