UNIVERSITY of WASHINGTON

Welcome to BIGDATA 210: Introduction to Data Engineering

Week 7

Week 7

What we will do?

- Questions from last week
- Status of Projects
- Assignment Review
- Concepts of Apache NiFi
- Hands-On

Basics of Connecting Systems

When we look at basics of connecting systems these must agree:

- Protocol
- Format
- Schema
- Priority
- Size of event
- Frequency of event
- Authorization access
- Relevance

IoT Edge Requirements

- Small Footprints
  - operate with very little power
- Limited Bandwidth
  - can create high latency
- Data Availability
  - exceeds transmission bandwidth
  - recoverability
- Data Must Be Secured
  - throughout its journey
  - both the data plane and control plane

Who all Need Data Provenance and why?

- For Operators
  - Traceability, lineage, Recovery and replay
- For Compliance
  - Audit trail, Remediation
- For Business
  - Value sources, Value IT investment

Fine-grained Security and Compliance

It's not enough to say you have encrypted communications

- Enterprise authorization services – entitlements change often
- People and systems with different roles require difference access levels
- Tagged/classified data

Dataflow Challenges

- Systems fail
  - Networks fail, disks fail, software crashes, people make mistakes.
- Data access exceeds capacity to consume
  - Sometimes a given data source can outpace some part of the processing or delivery chain - it only takes one weak-link to have an issue.
- Boundary conditions are mere suggestions
  - You will invariably get data that is too big, too small, too fast, too slow, corrupt, wrong, or in the wrong format.
- What is noise one day becomes signal the next
  - Priorities of an organization change - rapidly. Enabling new flows and changing existing ones must be fast.

Dataflow Challenges

- Systems evolve at different rates
  -The protocols and formats used by a given system can change anytime and often irrespective of the systems around them. Dataflow exists to connect what is essentially a massively distributed system of components that are loosely or not-at-all designed to work together.
- Compliance and security
  - Laws, regulations, and policies change. Business to business agreements change. System to system and system to user interactions must be secure, trusted, accountable.
- Continuous improvement occurs in production
  - It is often not possible to come even close to replicating production environments in the lab.

Apache NiFi Addresses Modern Data Flow Challenges

- HDF provides 3 key capabilities
  – the ability to collect data from different types of data sources via a highly secure lightweigt agent,
  - the ability to mediate the data flow to/from the data source and the "collector", and
  - the ability to trace, parse, transform data in motion to enable analytics and derive insights within an operationally relevant time window.

Apache NiFi Use Cases

- optimize Splunk investment
  - by pre-filtering data before sending to Splunk for storage
- ingest logs
  - for cyber security and threat detection
- feed data to streaming analytics engines
  - like Apache Spark or Apache Storm
- move their own data internally
  - between data centers on premises or to the cloud

Apache NiFi Use Cases

- capture data from the Internet of Things for
  - Predictive Analytics - Ensure the highest value data is captured and available for analysis
  - Fraud Detection - Move sales transaction data in real time to analyze on demand
  - Accelerated Data Collection - An integrated, data collection platform with full transparency into provenance and flow of data
  - IoT Optimization - Secure, Prioritize, Enrich and Trace data at the edge
  - Big Data Ingest - Easily and efficiently ingest data into Hadoop

Apache NiFi - Key Features

- Guaranteed Delivery
  - achieved through effective use of a purpose-built persistent write-ahead log and content repository
- Data Buffering w/ Back Pressure and Pressure Release
  - as those queues reach specified limits or to age off data as it reaches a specified age
- Prioritized Queuing
  - default is oldest first, but there are times when data should be pulled newest first, largest first, or some other custom scheme
- Flow Specific QoS (latency v throughput, loss tolerance, etc.)
- Data Provenance
  - automatically records, indexes, and makes available provenance data as objects flow through the system even across fan-in, fan-out, transformations, and more
- Recovery / Recording a rolling buffer of fine-grained history
  - designed to act as a rolling buffer of history
  - removed only as it ages off the content repository or as space is needed

Apache NiFi - Key Features

- Visual Command and Control
- Flow Templates
  - allow domain experts to build and publish their flow designs and for others to benefit and collaborate on them
- Security
  - use of protocols with encryption such as 2-way SSL
  - encrypt and decrypt content and use shared-keys
  - pluggable authorization so that it can properly control a user's access
- Designed for Extension
  - Processors, Controller Services, Reporting Tasks, Prioritizers, Customer User Interfaces
- Classloader Isolation
  - ensuring that each extension bundle is exposed to a very limited set of dependencies
- Clustering (scale-out)

Flow Based Programming (FBP)

- FlowFile
  - Unit of data moving through the system
  - Content + Attributes (key/value pairs)
- Processor
  - Performs the work, can access FlowFiles
- Connection
  - Links between processors
  - Queues that can be dynamically prioritized
- Process Group
  - Set of processors and their connections
  - Receive data via input ports, send data via output ports

Primary Components

NiFi executes within a JVM living within a host operating system. The primary components of NiFi then living within the JVM are as follows:

- Web Server
  - hosts NiFi's HTTP-based command and control API
- Flow Controller
  - brains of the operation
  - provides threads for extensions to run on and manages their schedule of when they'll receive resources to execute
- Extensions
  - various types of extensions for NiFi which will be described in other documents
  - extensions operate/execute within the JVM

Primary Components

- FlowFile Repository
  - stores state of what it knows about a given FlowFile that is presently active in the flow
  - The default approach is a persistent Write-Ahead Log that lives on a specified disk partition
- Content Repository
  - actual content bytes of a given FlowFile live here
  - The default approach stores blocks of data in the file system.
  - More than one file system storage location can be specified so as to get different physical partitions engaged to reduce contention on any single volume
- Provenance Repository
  - all provenance event data is stored.
  - The repository construct is pluggable with the default implementation being to use one or more physical disk volumes
  - Within each location event data is indexed and searchable

Example Use Case

- IoT platform makes extensive use of HDP already; they basically host the platform for customers like "Special Forces"
- They're looking at NiFi to replace the Ingestors and Translators portion of their architecture
- NiFi would then flow the data into Kafka for downstream data delivery to real-time and historical analytic applications
- NiFi gives them the ability to add new data feeds (with corresponding NiFi processors) in a matter of hours (rather than days/weeks); they use a JSON spec file that contains the info needed to plumb in the new NiFi processor

Example Use Case

- NiFi data provenance capabilities are a big value (knowing where data come from and tracking where/how it flows is a key operational capability)
- NiFi's logging and tracing capabilities make it easy to debug dataflows, and NiFi's ability to replay flows is invaluable as well (ex. they were able to replay a weeks worth of inbound data in an hour)
- They like the ability to fork a flow to plug in a new processor (agility is a key attribute)

Hands-On

Building a basic flow

Starting with a blank NiFi canvas.

1. Drag the processor icon onto the canvas.
2. Explore the available processors
3. Select the GetFile Processor and add it
4. Configure the GetFile Processor
    1. Right-click on the new processor
    2. Select "Configure"
    3. The tabbed dialog should show settings, give the processor a name
    4. Select the properties tab
    5. Input "/tmp/input" (before doing this use "sudo su nifi" to become the nifi user, and make sure you create this directory on your NiFi box, and that it is owned by the nifi user - chown -R nifi:nifi /tmp/input)

UpdateAttribute processor

1. Now add an UpdateAttribute processor
    1. Configure it with a new dynamic property
    2. In the properties tab, press the plus sign in the top right of the dialog
    3. Call the property "filename" and set the value to something meaningful
    4. Add a property called "mime.type" and set this to "application/gzip"

Connecting processors

1. Connect the two processors
    1. Hover over the GetFile processor, until the begin connection icon appears
    2. Drag this onto the UpdateAttribute processor
    3. This brings up the connection configuration dialog
    4. For now just leave this on defaults.

CompressContent processor

1. Add a CompressContent processor and look at its properties, the defaults should work fine here.
2. On settings, make sure you set "Auto-terminate relationships" on for failure
   configure it
3. Now connect up the output of our UpdateAttributes processor to the new CompressContent

PutFile processor

1. Add a PutFile processor
2. Configure the destination location in the PutFile processor properties. Note the conflict resolution strategy, and set this to replace. (nifi will create this directory for you)
3. Set both success and failure relationships to auto-terminate in the settings tab
4. Setup the connection between the CompressContent processor and the PutFile processor (only for the success relation!)

Running the Dataflow

- Press play at the top of the screen (make sure you don't have a processor selected)
- Copy a file (for example /var/log/ambari-agent/ambari-agent.log) to the input folder you chose
- NiFi will pick it up a compress it into the output directory you chose

Adding GetHTTP processor

1. Drag the GetHTTP processor icon onto the Canvas
2. Configure the new processors:
    1. Select scheduling
    2. Set the run schedule to "1 min" (we can also use other times and units of time)
    3. In properties, pick the url of your favourite website
    4. Set a filename, this is what the filename attribute will be set to.
3. Connect this into the UpdateAttribute processor we used before.
4. Now right-click your new processor and "Start" it.

Process Groups and Organization

So far we've been putting all the flows into the root canvas. This is fine for simple flows, but sometimes you need a bit of organization.

First Let's create a simple flow.

1. Create a GetFile processor
2. Create a SplitText processor
    1. configure it with a line count of 1 in the properties

Processor group

Now we want to write this to HDFS. That is often repeated and takes a few steps, lets group them.

1. Drag a processor group from the toolbar.
2. Give it a name
3. Double click into it, you will get a new canvas.

MergeContent processor

Now in the new blank canvas:

1. Drag an input port onto the canvas, and call it "Lines"
2. Add a MergeContent processor and configure it with BinPacking
    1. Set the min size to a few Lines
    2. Set the max age to a few minutes
    3. Set max size to 128MB
3. Add an UpdateAttribute to change the filename

CompressContent and PutHdfs

1. Add CompressContent, use bzip2
2. Add PutHdfs
    1. Configure the properties to point to the hadoop config files (/etc/hadoop/conf/core-site.xml,/etc/hadoop/conf/hdfs-site.xml)
    2. Set a directory (and make sure it exists of HDFS and the nifi user has permissions, /tmp is a good place for now)
3. Add and output port
4. Now connect up all these pieces with the relevant relations.
5. Set the MergeContent schedule to run every 5 minutes.

Connecting

You can now escape up to the root canvas with the bread crumb trail at the top of the screen.

- Now hook the output of the flow on the root canvas to the input port in the ProcessGroup

External Sources

Let's looks at some external data sources

1. Create a ListenHTTP processor
    1. Set the listening port to 9091
    2. Set the base path to "hello"

This is a web listener

Sending Data

1. Feed this into a PutFile processor.

Use curl to send data to this:
curl -d"data" -X POST http://hostname:9091/hello

You will see this in your flow.

Syslog listener

Let's also look at the syslog listener

1. Create a ListenSyslog processor.
    1. Note the batching options here.
    2. NiFi is running as a non-privileged user, so you will not be able to use the default port (514) try 1514 instead, and point syslog to this instead.
2. Push this to a PutFile.

List, Fetch pattern

A common pattern is the List, Fetch pattern.

1. Create a ListSFTP processor
2. Create a FetchSFTP processor
3. Connect them up

Investigate the many ingest options!

Connecting to Kafka

Go back to your HDP cluster, with the PutHDFS piece.
Let's push the individual lines into Kafka

1. Add a PutKafka processor.
2. Configure the broker list property to point to ip_address:6667, and the topic 'test'
3. ssh into the hdp cluster and create the kafka topic (/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --topic test --partitions 1 --replication-factor 1 --zookeeper localhost:2181)

Egress to Kafka

→ Run the PutKafka processor

Test this by pushing files into your flow input folder. Lines appear one by one in Kafka, and as a block in HDFS.

To see the files in Kafka,

```
run /usr/hdp/current/kafka-broker/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test
```

on your kafka box.

Routing on Attributes and Priorities

Prioritisation is a key element of NiFi. Let's set up some queue priorities.

Starting with our SplitText based example on "local" NiFi cluster, let's extract some content, and route things differently.

We are only interested in seeing security events in Kafka, we also want them processed faster

Extracting and labeling security events

1. After SplitLines, add a new processor group
2. Use ExtractText with a new dynamic property called `security`, set the match pattern to `(.*auth.*)`
3. Insert an UpdateAttribute processor, this will get a custom attribute calls `priority` and set it to `10`
4. Add a dynamic property called `failed`, and set this to `${security:contains("failed")}`
5. Add a RouteOnAttribute processor, and pick out two routes with dynamic properties, for example:
    1. `certificate issue,${security:contains('certificate')}`
    2. `login failed,${security:contains('login')}`

Creating funnel and route

1. Route each of these to a different UpdateAttribute processor, and use them to set the priority attribute to a 5 and 1 respectively.
2. Create funnel and route both these processors into the funnel.
3. Now create a route to your remote cluster from the funnel, and set the prioritisers to PriorityAttributePrioritiser in connection settings

Let's also look at better names for our HDFS files, so in our HDFS group, use the UpdateAttribute to set a filename. Change the filename property to `${now():format("yyyyMMddhhmm")}`