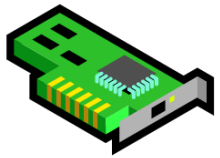
Apache Nifi (Niagara Files)

OPEN SOURCE DATA DISTRIBUTION AND PROCESSING SYSTEM

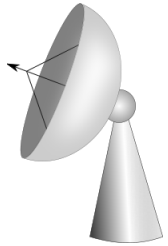
NiFi (Niagara Files) Introduction

- An Open Source Data Distribution
- Framework for managing complex dataflows
- Provides a way to move data from one place/system to another place / system
- Making transformations and routing decisions as necessary in real time streaming
- Scalable directed graphs of data routing, transformation, and system mediation logic
- History
  - 2007 – NiagaraFiles (NiFi) was first incepted at National Security Agency NSA
  - 2014 – Donated to Apache Software Foundation and enters Incubator
  - 2015 – Reaches top level project status

"nye fye" (nī fī) is the preferred pronunciation




DB



Why Apache Nifi Vs Other ETLs

- Powerful
- Scalable
- User intuitive Web based Interface
- Highly Configurable
- Data Provenance
- End to End Data Flow Tracking
- Secure (Provides SSL, SSH, HTTPS, Encrypted Content, Authentication/Authorization)
- Customizable and extensible
- Enables rapid development

What is Dataflow?

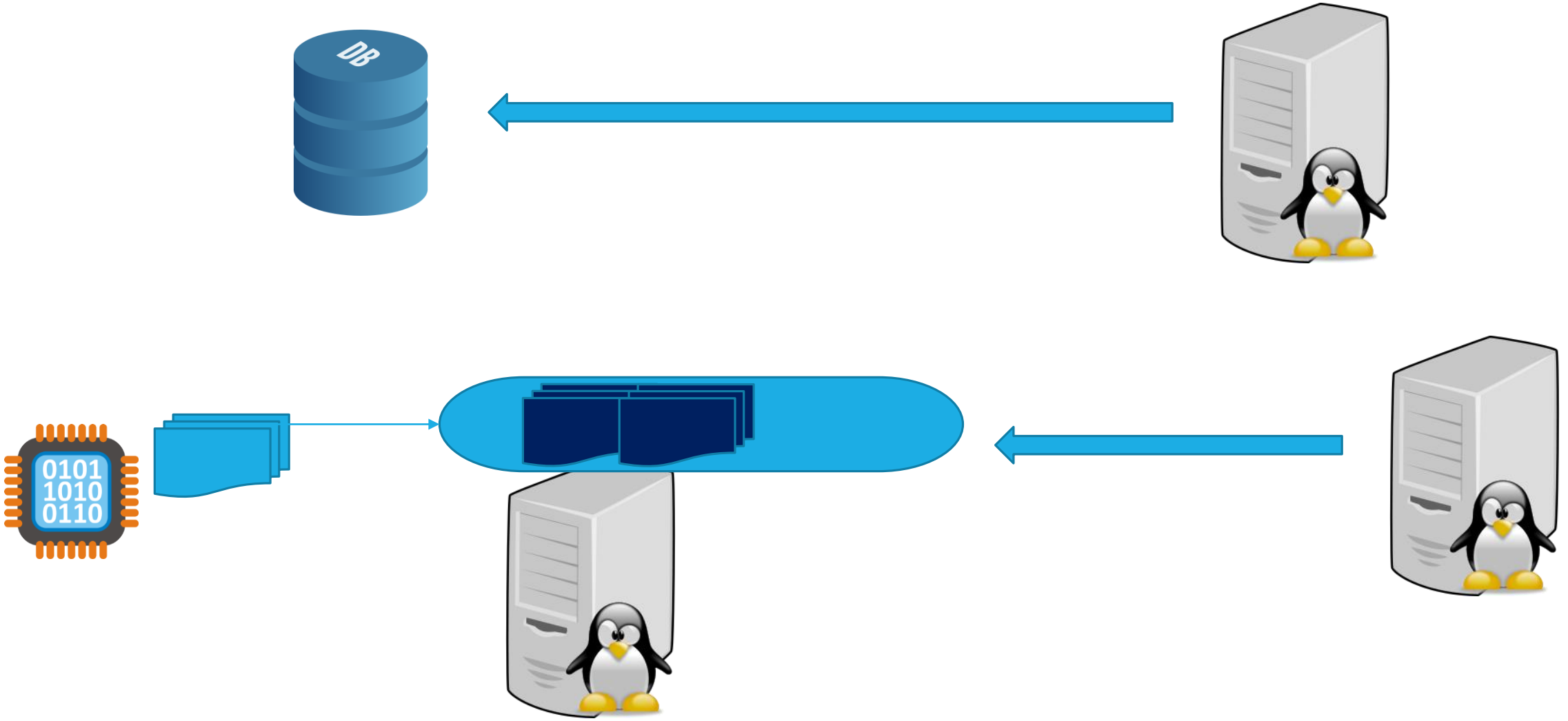
- Moving data between two heterogenous systems.
- Data could be anything like log files, HTTP request, XML, CSV, Audio, Video, Telemetry data
- Data can be moved between anything and everything through internet or intranet

Dataflow and Challenges

- Standards, Formats, Protocol, Variety, Veracity, Partitioned data.
- Ensuring Security, Authentication, Authorization, Network and “Exactly Once” Delivery
- Dynamically changing producer and Consumers architecture and requirements
- Receiving data from unknown source and following unknown standards.

NiFi – Key Features

- Guaranteed Delivery through Data buffering and ack pressure
- Timely and configurable que prioritizing and pressure release
- Data Provenance
- Supports both push and pull models
- Recovering / Recording a rolling log of all fine grained activity
- Visual screens to manage and control
- Templates for rapid development and deployment
- Pluggable and extensible design
- Clustering for highly scaling model


DB
0101
1010
0110

Installation and Starting Nifi

- ✓Supported in Windows and Linux/Mac OS
- ✓Can be installed as standalone or as service [Supported only in Linux and Mac OS]
- ✓Download url -> https://nifi.apache.org/download.html
- ✓In Windows
  - ✓-> Download -> Extract -> execute '\bin\run-nifi.bat'

Installation and Starting Nifi

- ✓ In Linux
  - ✓ -> Download -> Extract ->
    - ✓ bin/nifi.sh run -> To run in foreground
    - ✓ bin/nifi.sh start - > To start in background
    - ✓ bin/nifi.sh status -> To check the status
    - ✓ bin/nifi.sh stop -> To shutdown

Installing as a Service

- bin/nifi.sh install *[with the default name nifi]*
- bin/nifi.sh install dataflow *[with the name dataflow]*
  - sudo service nifi start
  - sudo service nifi stop
  - sudo service nifi status

Terminology and Definitions

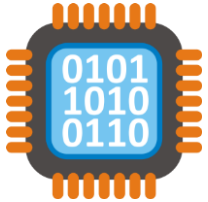
FlowFile – Each object moving through the system. Attributes and associated content

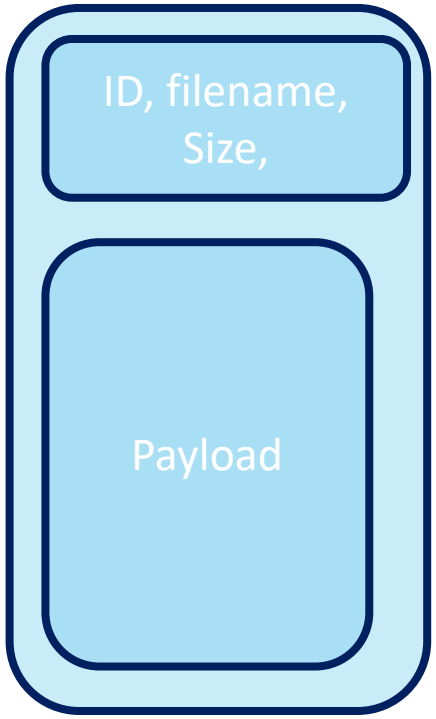
FlowFile Processor – Does the actual work. Does work like routing, transformation, mediation etc. Operates on one or more Flowfile

Connection – Provides actual linkage between processors. Acts as Queues between processor with various rates and priorities. Provides load and pressure configuration

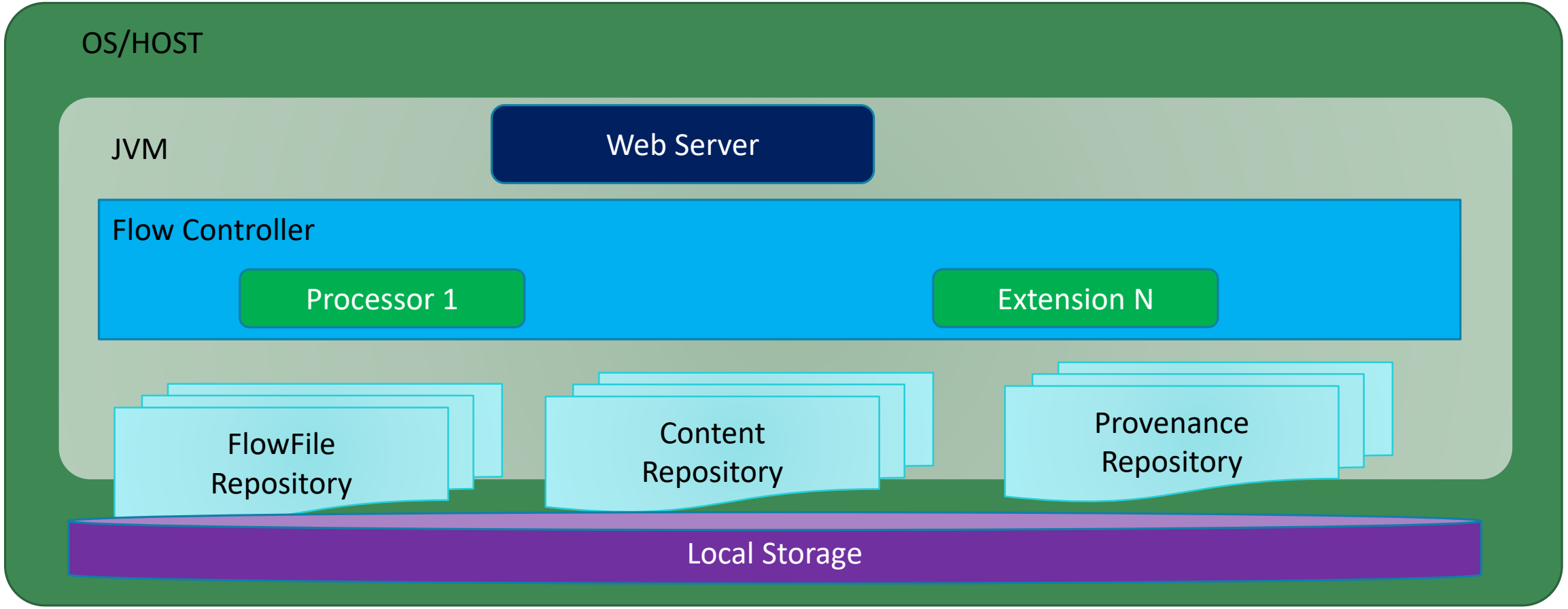
Flow controller – Maintains information on how processors connects and manages the resources. Acts like broker to facilitate exchange and flow of flowfiles.

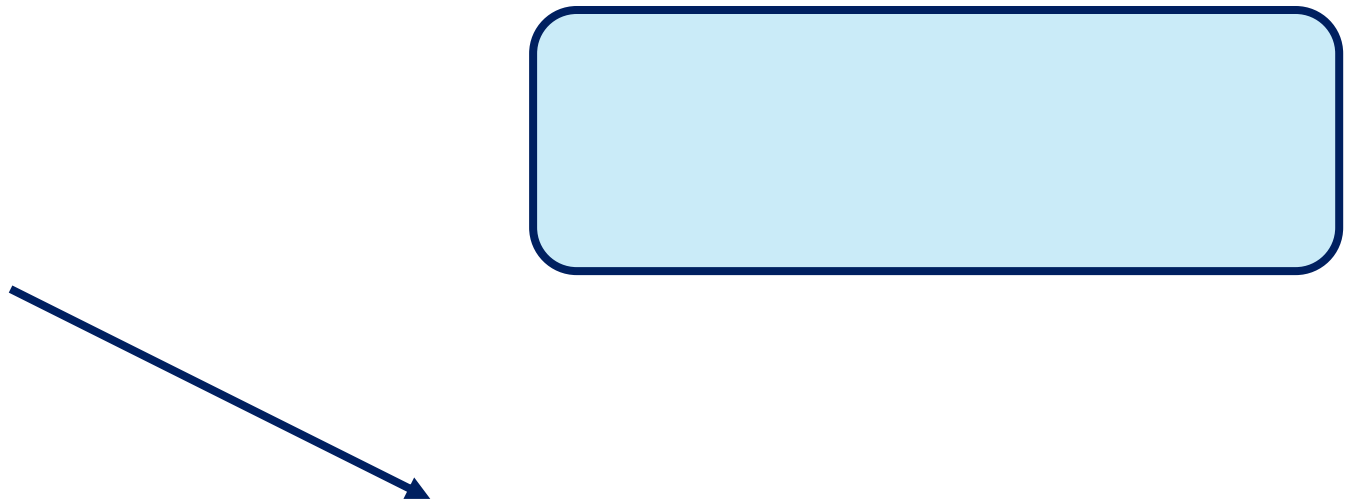
Process group – Group of logically similar processors, connections for easy maintenance and management.


0101
1010
0110

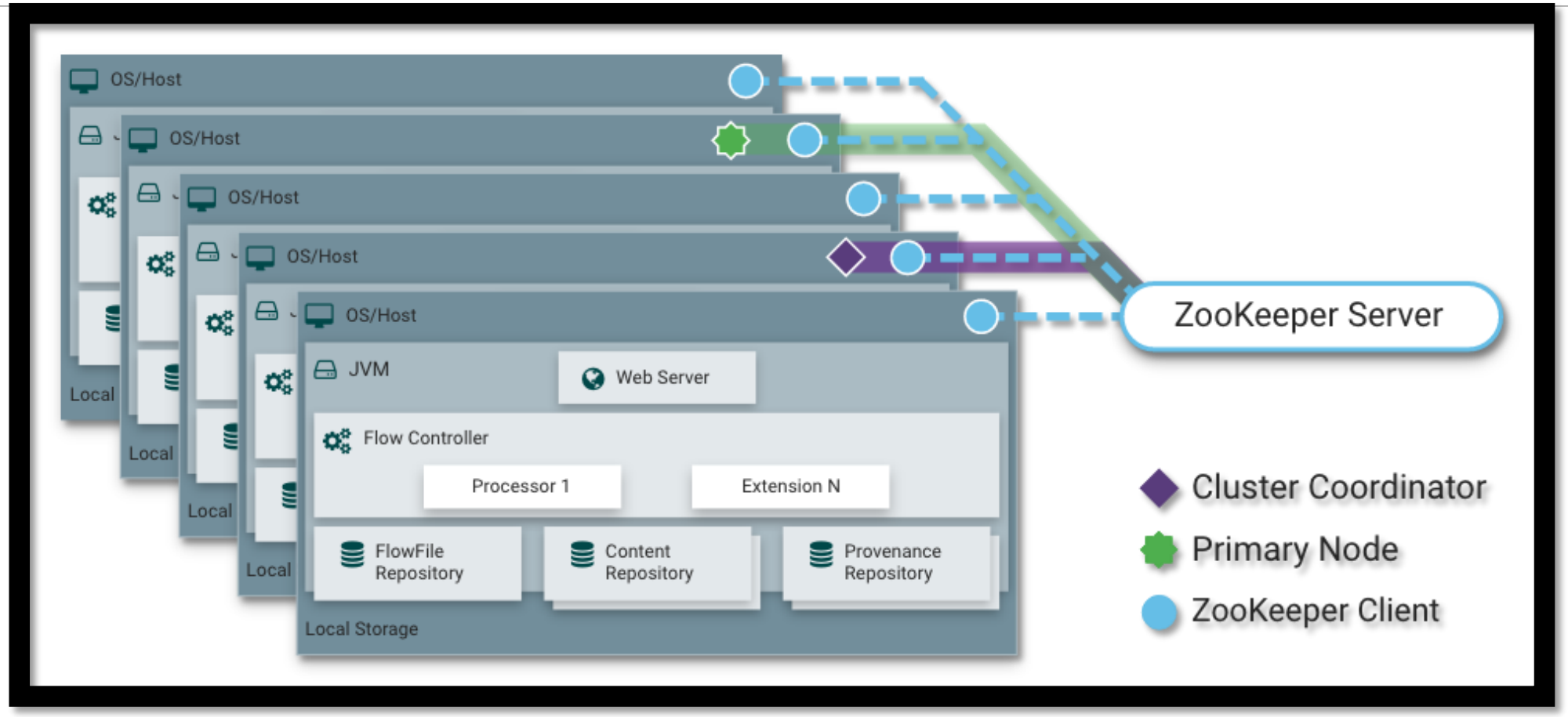

ID, filename,
Size,
Payload

Architecture





Scalable Architecture



UI Overview

- Highly Interactive and built with rich in features.
- Has following 6 major sections.
  - Component Toolbar
  - Global Menu
  - Search
  - Status Bar
  - Navigate Palette
  - Operate Palette

Component Toolbar

Has following icons to work with

- Processors -> Does the actual work against the flowfile
- Input Port -> To get data from a processor with is outside the group
- Output Port -> To transfer data from the processor group to a processor which is outside the group
- Processor Group –> Groups logically together components.
- Remote Process Group –> Groups a logically together components in a remote process group
- Funnel -> To transfer the output of a processor to multiple processors. Basically it works as fan-in and fan-out architecture
- Template -> Add dataflow template. Templates used to create reusable template for similar pattern scenarios.
- Label -> Add text to Nifi canvas about any component


Input Folder
Output Folder

Status Bar

- Active Threads
- Total queued data
- Transmitting Remote Process Groups
- Not Transmitting Remote Process Groups
- Running Components
- Stopped Components
- Invalid Components
- Disabled Components
- Up to date Versioned Process Groups
- Locally modified Versioned Process Groups
- Stale Versioned Process Groups
- Locally modified and Stale Versioned Process Groups
- Sync failure Versioned Process Groups

Processor Categories

Data Ingestion -> ingest data from a file system like HDFS or normal file system or any other streaming data system E.g.: GetFile, GetHTTP, GetFTP, GetKafka, etc.

Routing and Mediation -> Route the data based on the content or attributed to different processors

Database Access -> To do all type of operations against a databases. E.g: ExecuteSQL, PutDatabaseRecord, etc

Attribute Extraction -> To extract, change, update flowfile attributes. E.g: UpdateAttribute, ExtractText, AttributeToJSON, etc

System Interaction -> Run or execute system process, commands in Nifi host operating system. E.g: ExecuteScript, ExecuteGroovyScript, etc.

Processor Categories

Data Transformation -> To alter content of flowfiles. E.g: ReplaceTExt

Sending Data -> To send data or message to remote system. E.g: PutKafka, PutFile, PutFTP, PutEmail, etc.,

HTTP -> Deals with request and response of HTTP and HTTPS. Eg: InvokeHTTP, PostHTTP, etc.,

AWS -> Works with Amazon Web Services. PutSNS, FetchS3Object, etc.,

Connecting Components

Connecting two components and acts like a buffer /queue

Settings

- ✓Flowfile Expiration – Data not processed in timely fashion will be removed from the system. 0 Secs that is the default value, indicates data will never expire.
  - ✓Expiration can be done effectively with prioritizes.

Connecting Components

- ✓Back Pressure – How much data should be allowed to exist in the queue before the component that is the source of the connection is no longer scheduled to run
  - ✓Back pressure object threshold
  - ✓Back pressure data size threshold (B for bytes, KB for kilobytes and MB for megabytes and so on
  - ✓Default back pressure object threshold of 10000 objects and 1GB of back pressure data size.
- ✓Prioritization
  - ✓ **FirstInFirstOutPrioritizer -** The one that reached the connection first will be processed first
  - ✓**NewestFlowFileFirstPrioritizer -** The one that is newest in the dataflow will be processed first.
  - ✓**OldestFlowFileFirstPrioritizer -** The one that is oldest in the dataflow will be processed first. 'default’
  - ✓**PriorityAttributePrioritizer –** Processed based on the value of "priority" attribute

Connection Context Menu

Configure: To change the configuration of the connection.

View status history: Connection's statistical information.

List queue: Lists the queue of FlowFiles that may be waiting to be processed.

Go to source: The view of the canvas will jump to the source of the connection.

Go to destination: Changes the view to the destination component on the canvas .

Bring to front: Brings the connection to the front of the canvas.

Empty queue: To clear the queue of FlowFiles that may be waiting to be processed.

Delete: To delete a connection between two components.

Processor Settings

Penalty - The amount of time that the FlowFile is inaccessible is determined by the Dataflow Manager by setting the "Penalty Duration" setting in the Processor Configuration dialog. Default 30 secs. The Processor will not attempt to process that specific FlowFile again for 30 seconds. In the meantime, it is able to continue to process other FlowFiles.

Yield - It will not be able to perform any useful function for some period of time. The Processor is telling the framework that it should not waste resources triggering this Processor to run, because there's nothing that it can do - it's better to use those resources to allow other Processors to run.

Bulletin Level – Based on the log written by the processor this bulletin is generated as configured by the users. Default is WARN. Only warning and above will be added to bulletin

Processor Scheduling

Timer Driven – Default. Processor will be scheduled to run on a regular interval. Its is decided by run schedule.

*Even Driven – Not supported in all processors. Support may be removed in future.

CRON Driven – Processor will be scheduled to run periodically. Works very similar to any other Cron trigger.

Eg:

0 0 13 * * ? - To schedule the processor to run at 1:00 PM every day.

0 20 14 ? * MON-FRI - To schedule the processor to run at 2:20 PM every Monday - Friday.

0 15 10 ? * 6L 2011-2017 - To schedule the processor to run at 10:15 AM, on the last Friday of every month, between 2011 and 2017.

Scheduling CRON

Field	Valid values
Seconds	0-59
Minutes	0-59
Hours	0-23
Day of Month	1-31
Month	1-12 or JAN-DEC
Day of Week	1-7 or SUN-SAT
Year (optional)	empty, 1970-2099

- * - Indicates that all values are valid for that field.
- ? - Indicates that no specific value is specified. This special character is valid in the Days of Month and Days of Week field.
- L - You can append L to one of the Day of Week values, to specify the last occurrence of this day in the month. For example, 1L indicates the last Sunday of the month.

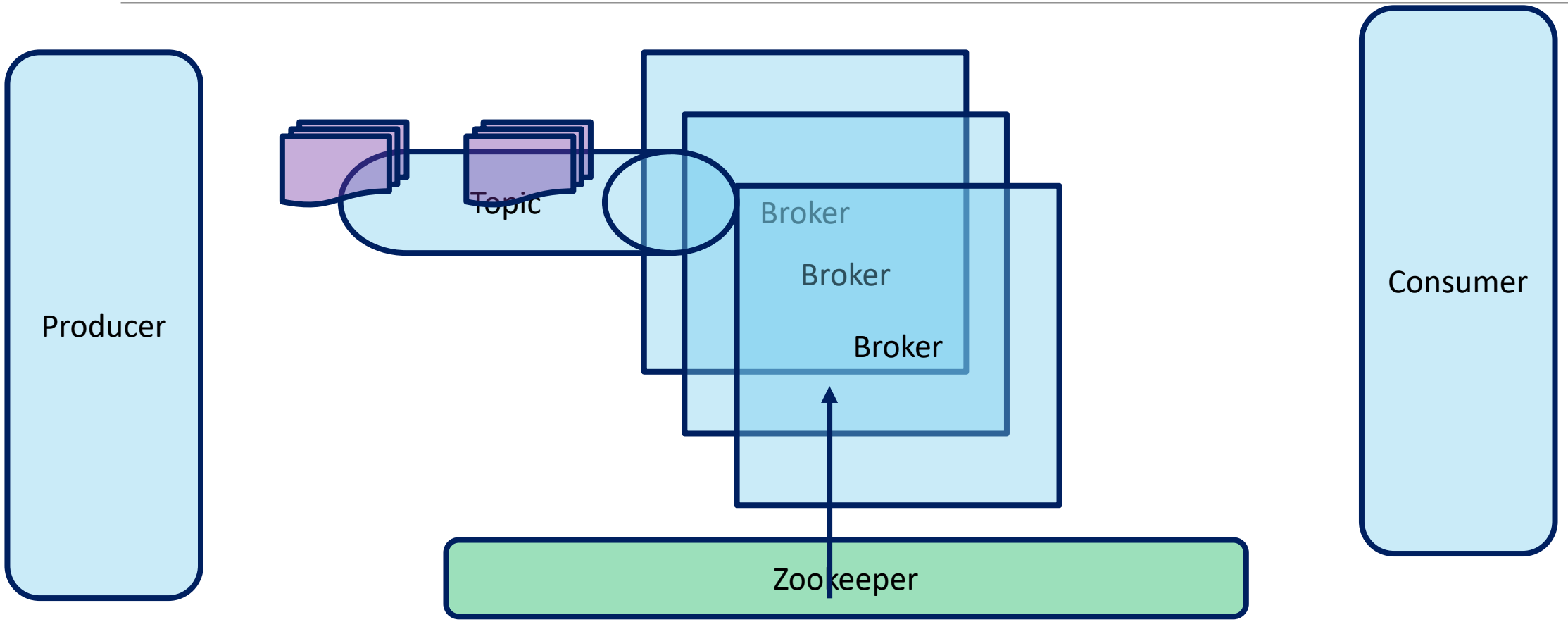
Processor Scheduling

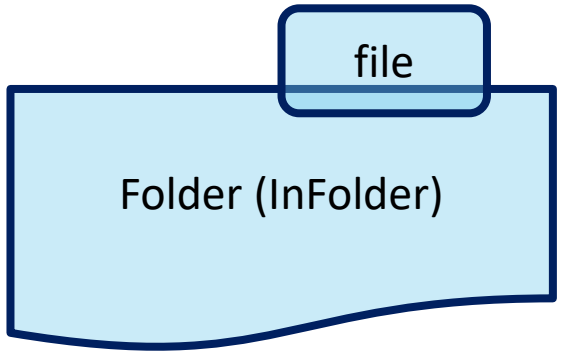
Concurrent Tasks – How many threads the processor will used. Or how many FlowFiles should be processed in parallel. This controls the throughput of the flow at the same time it consumes the resources proportionally.

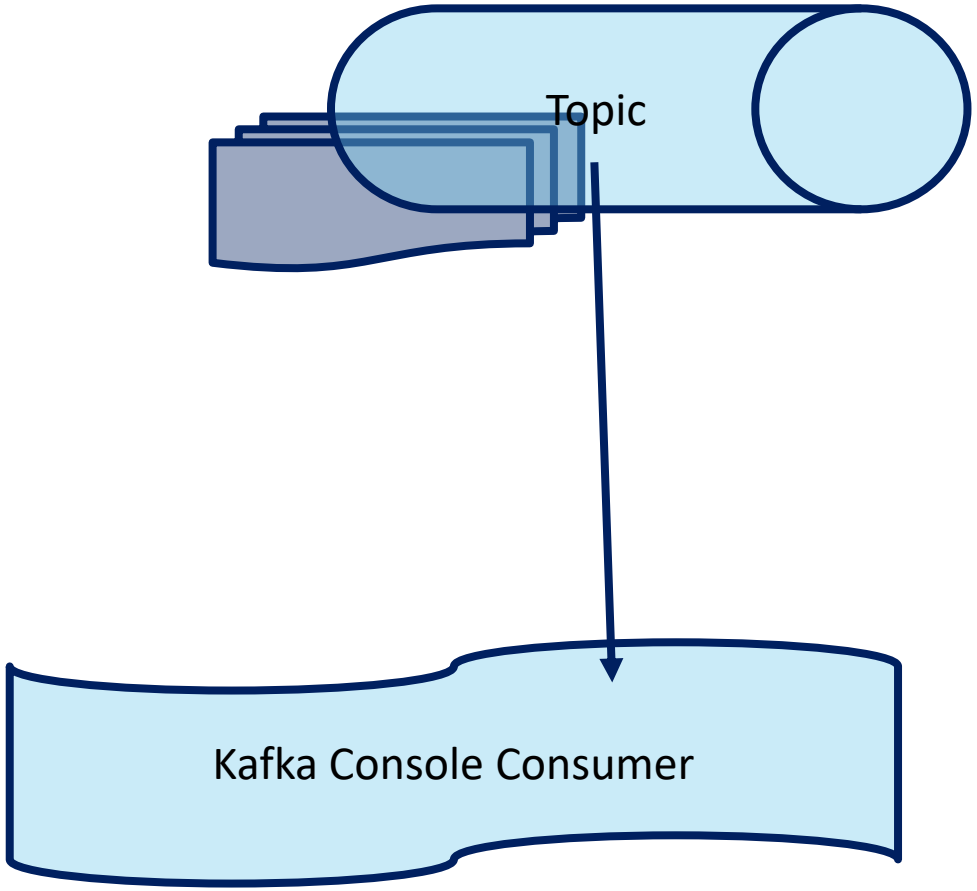
Run Schedule – How frequently the processor should be scheduled to run. Default is 0. Its runs as often as possible.

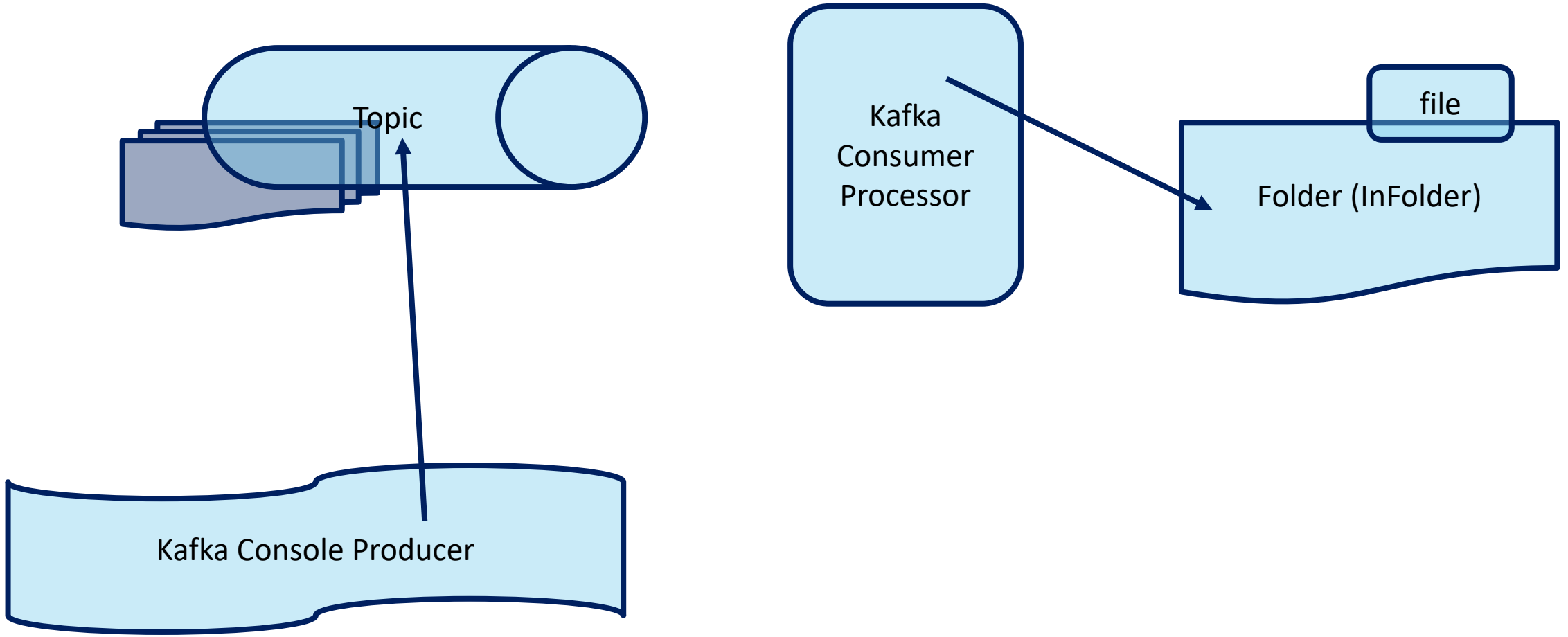
Run Duration – Controls how long the processor should be scheduled to run each time it is triggered. It includes updating the repository as well. This will determine tradeoff between latency and throughput.

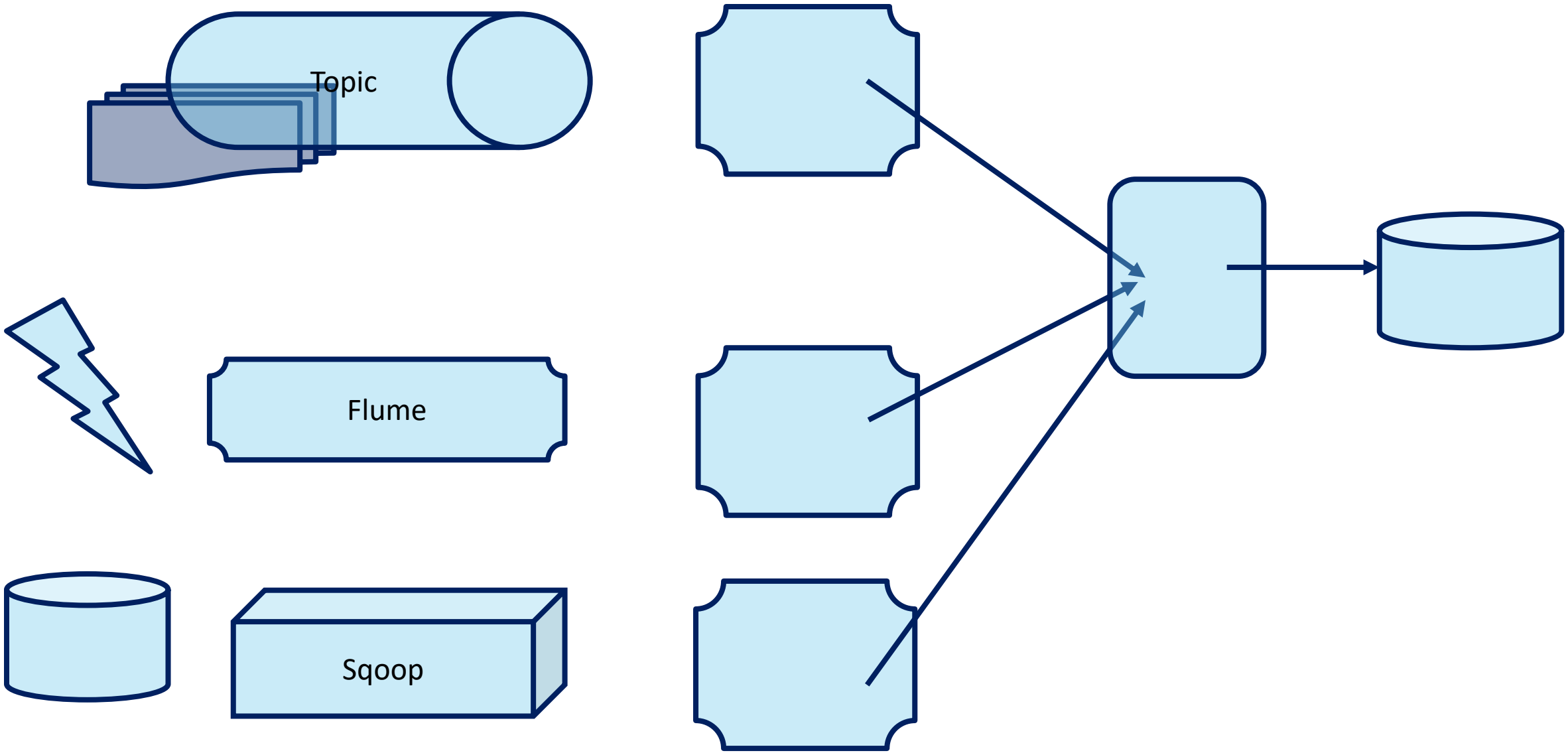
Kafka Introduction

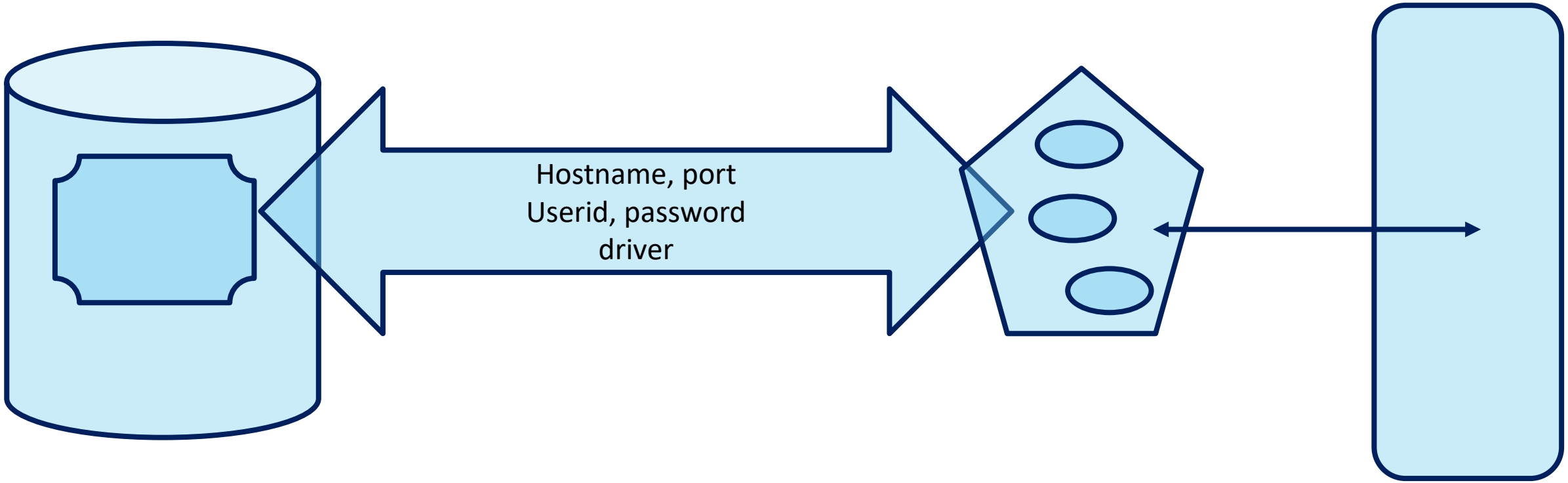


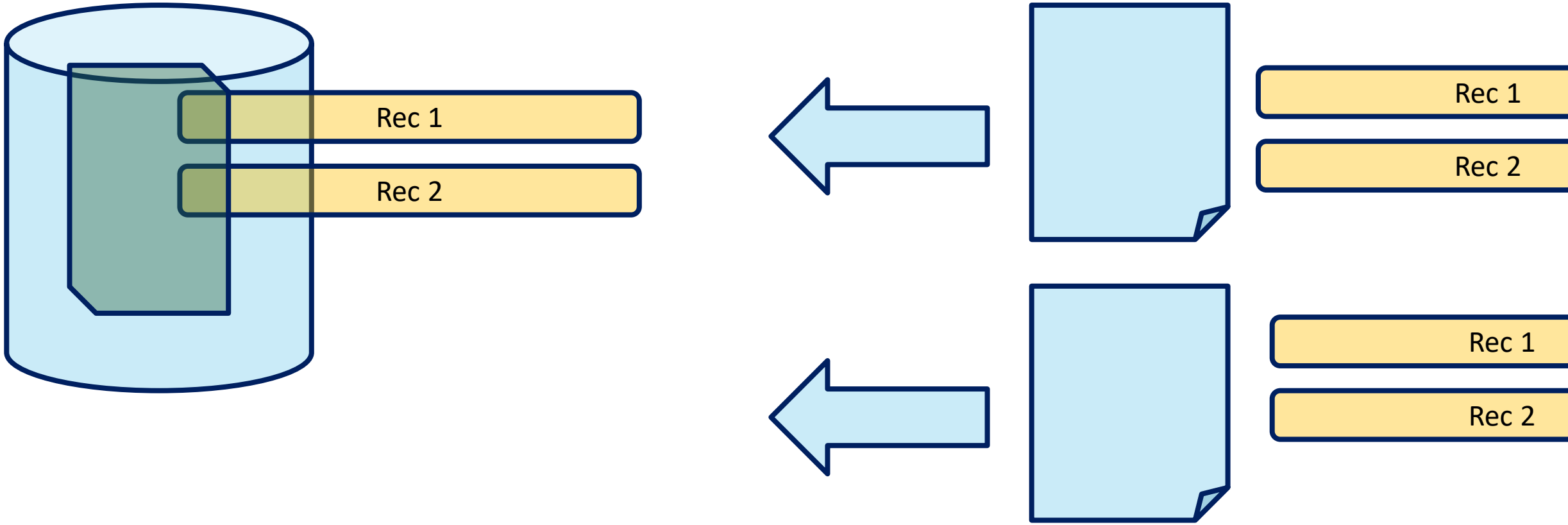

file
Folder (InFolder)

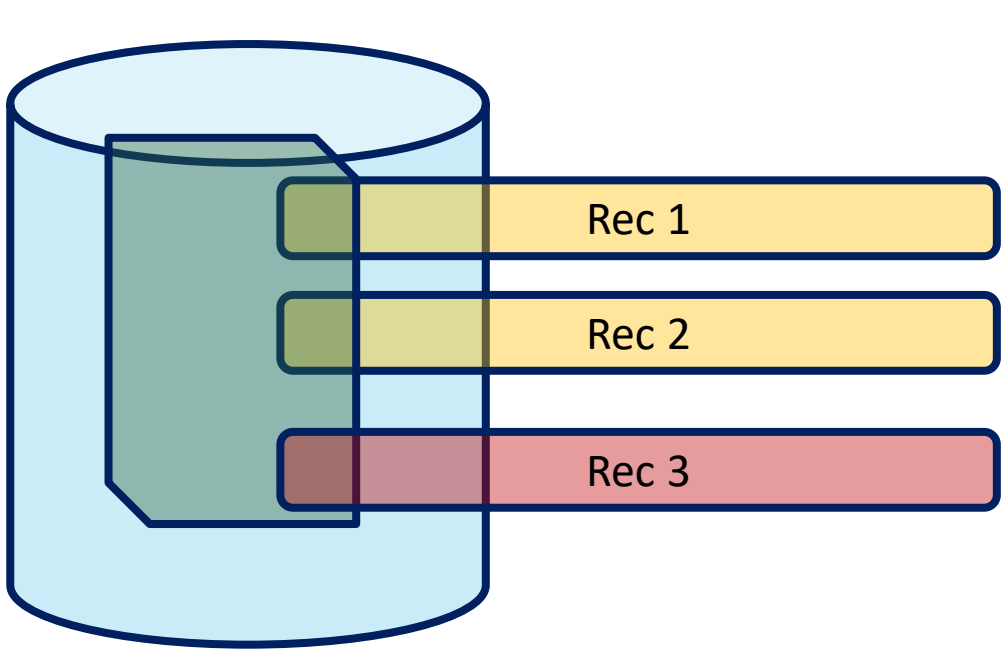

Topic
Kafka Console Consumer

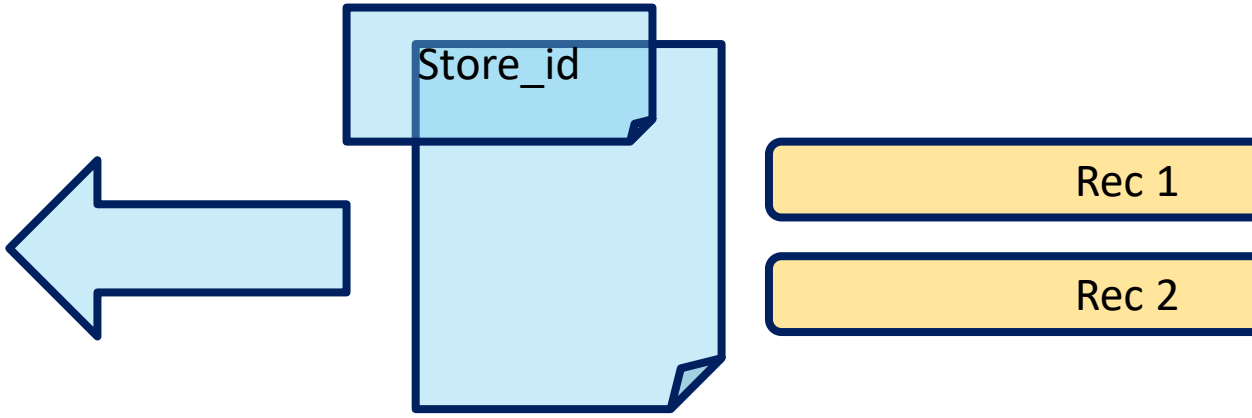

Topic
Kafka Console Producer
Kafka
Consumer
Processor
file
Folder (InFolder)


Topic
Flume
Sqoop


Hostname, port
Userid, password
driver


Rec 1
Rec 2
Rec 1
Rec 2
Rec 1
Rec 2


Rec 1
Rec 2
Rec 3


Store_id
Rec 1
Rec 2

Variable Precedence

- Flow file attributes
- Process Group variables
- Variables defined in the properties file
- System environment variables
- JVM system property

Clustering Nifi

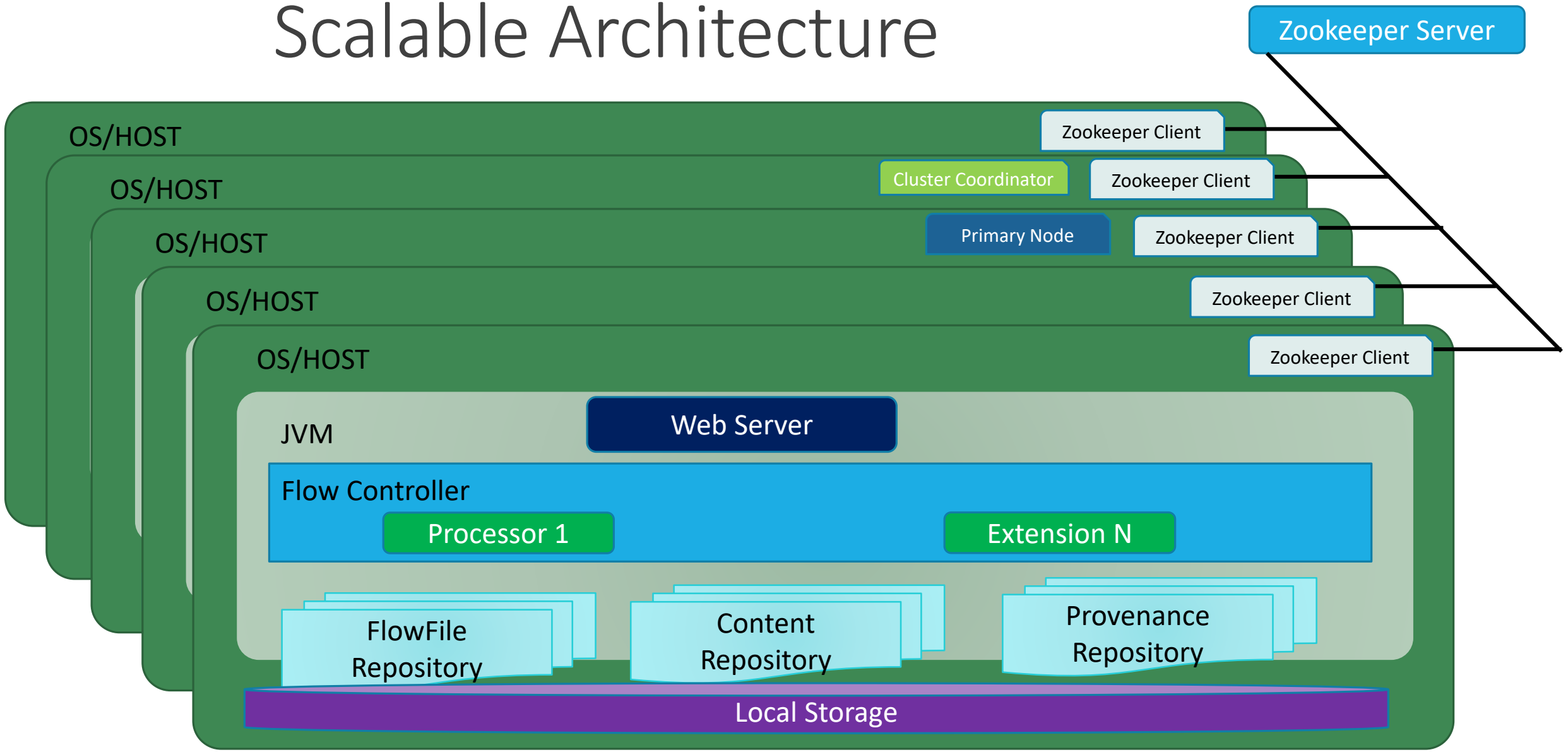
Horizontally scale NiFi to process larger volume of data

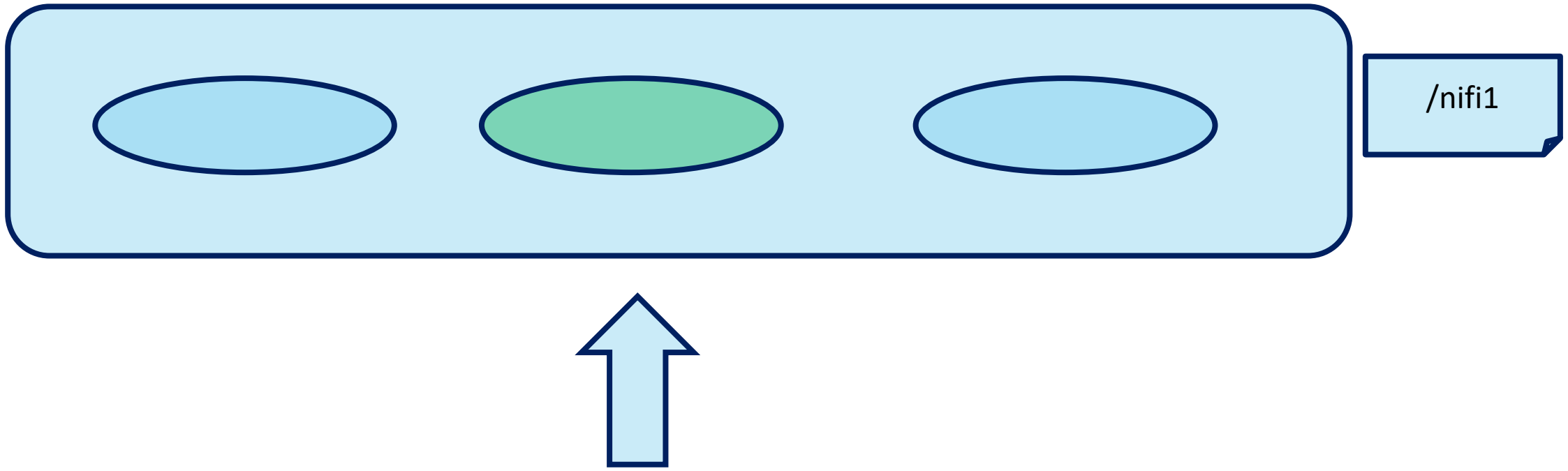
Works as master less architecture

Coordinated by zookeeper

Has primary node concept for the processor which should not run in distributed way

Scalable Architecture




/nifi1

Properties to be Configured

nifi.properties

nifi.web.http.host

nifi.web.http.port

nifi.cluster.is.node

nifi.cluster.node.address

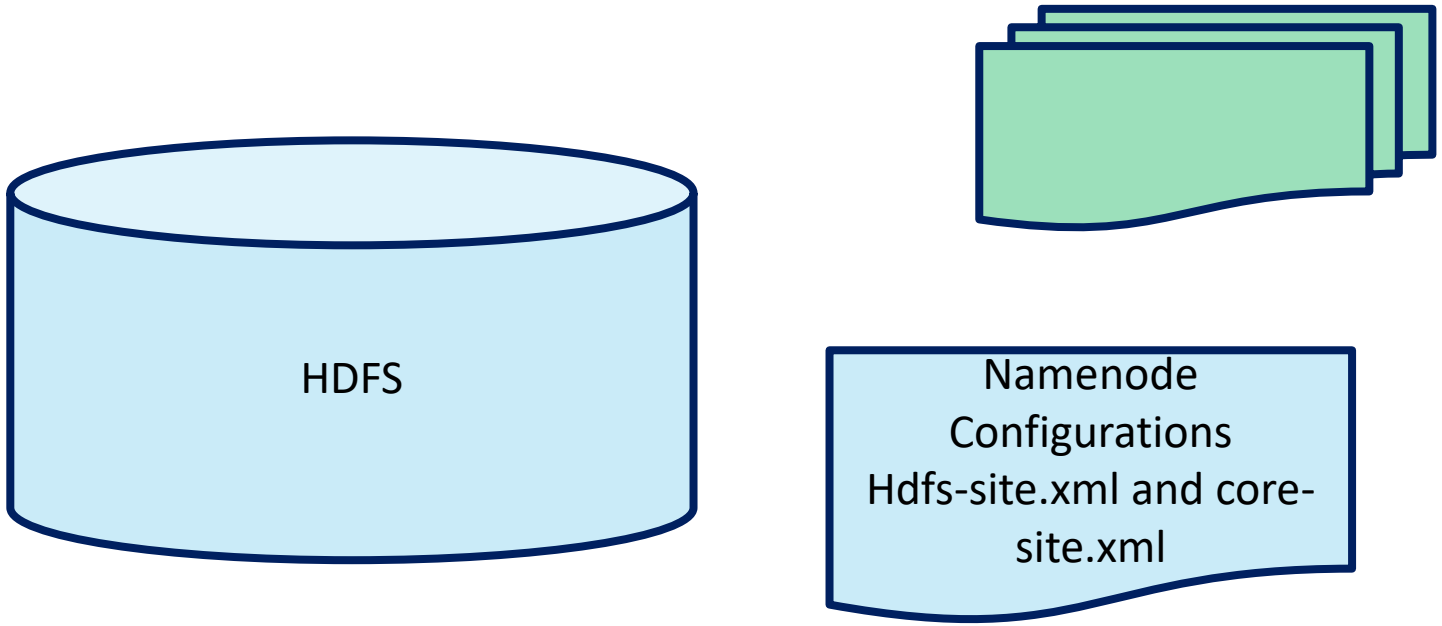
nifi.cluster.node.protocol.port

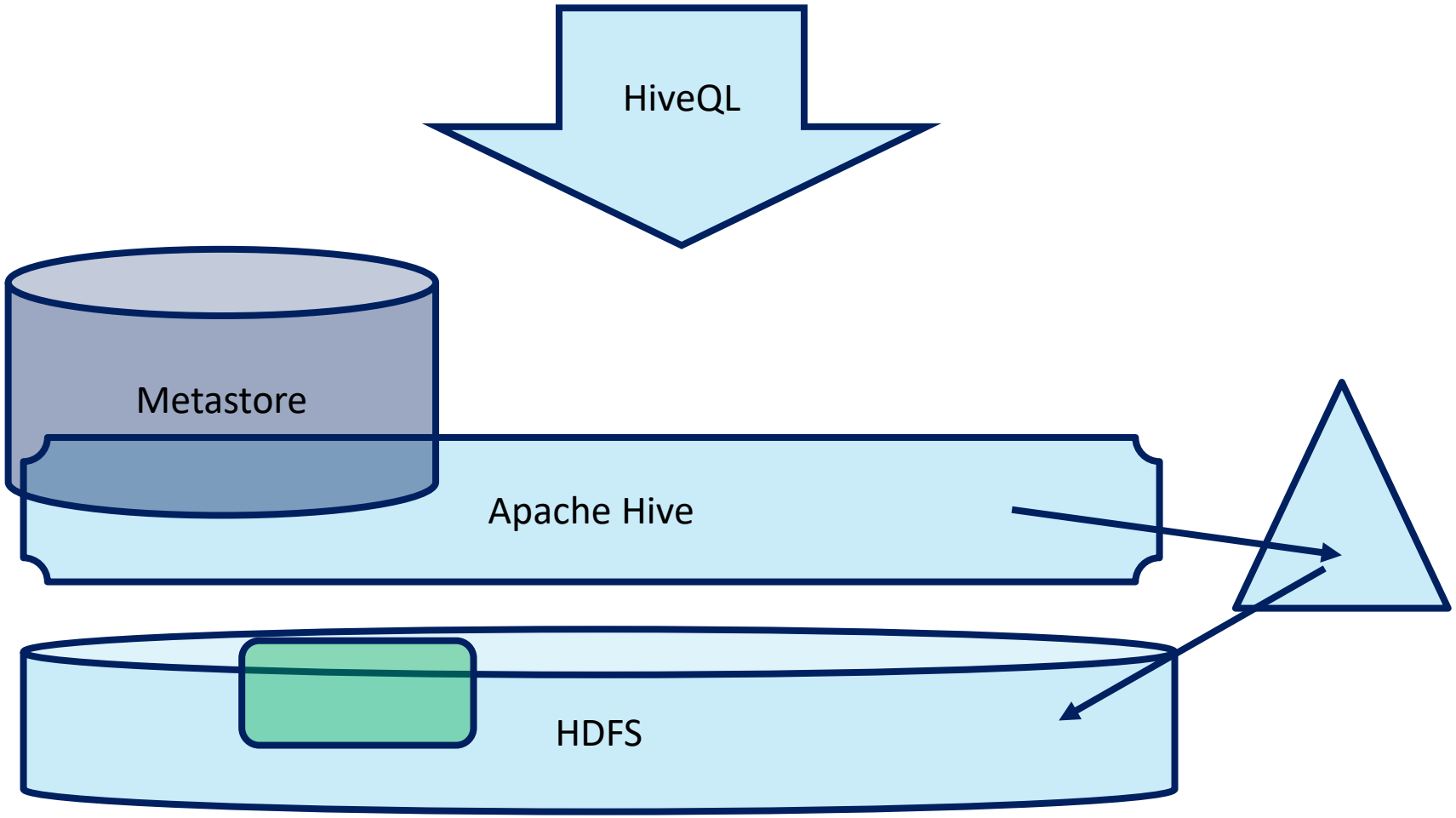
nifi.zookeeper.connect.string

state-management.xml

zk-provider - <property name="Connect String">

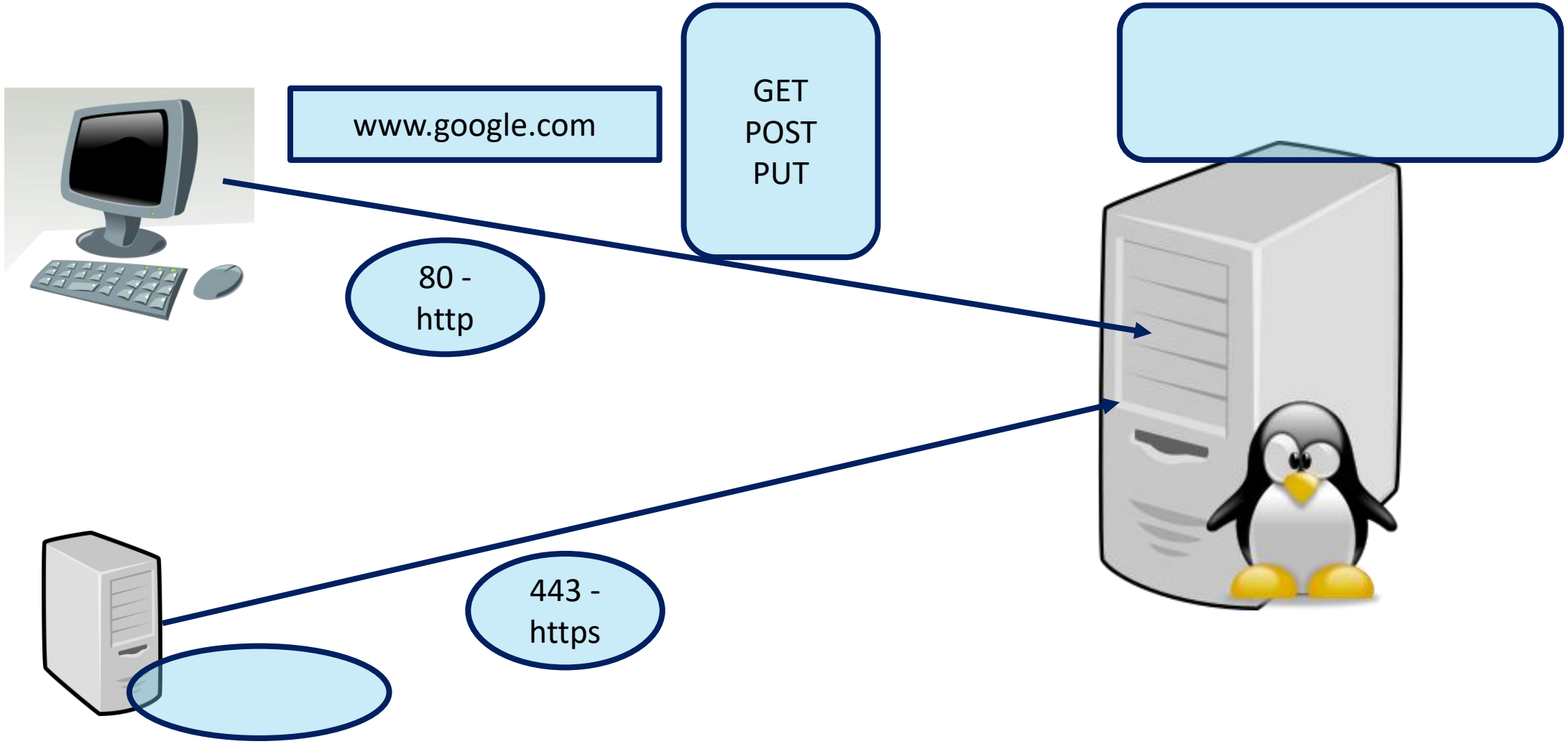
node1:2181,node2:2181,node3:2181


HDFS
Namenode
Configurations
Hdfs-site.xml and core-site.xml


HiveQL
Metastore
Apache Hive
HDFS

HTTP Processors

- GetHTTP (Client)
- PostHTTP (Client)
- ListenHTTP (Server)
- InvokeHTTP (Client)
- HandlerHttpRequest / HandlerHttpResponse


www.google.com
GET
POST
PUT
80 -
http
443 -
https

GetHTTP Processors

GetHTTP – Acts as true client. Connect to an HTTP endpoint and process or converts the FlowFile and it get pushed as flowfile to the upstream

- ✓ Very simple processor with only one outgoing relationship (Success)
- ✓ Limited to GET request only (Cannot process others like POST)
- ✓ Handling errors are difficult
- ✓ Don’t have incoming connection. Should be scheduled
- ✓ Good for simple pooling scenarios to fetch data using HTTP protocol from remote system on regular interval

PostHTTP Processors

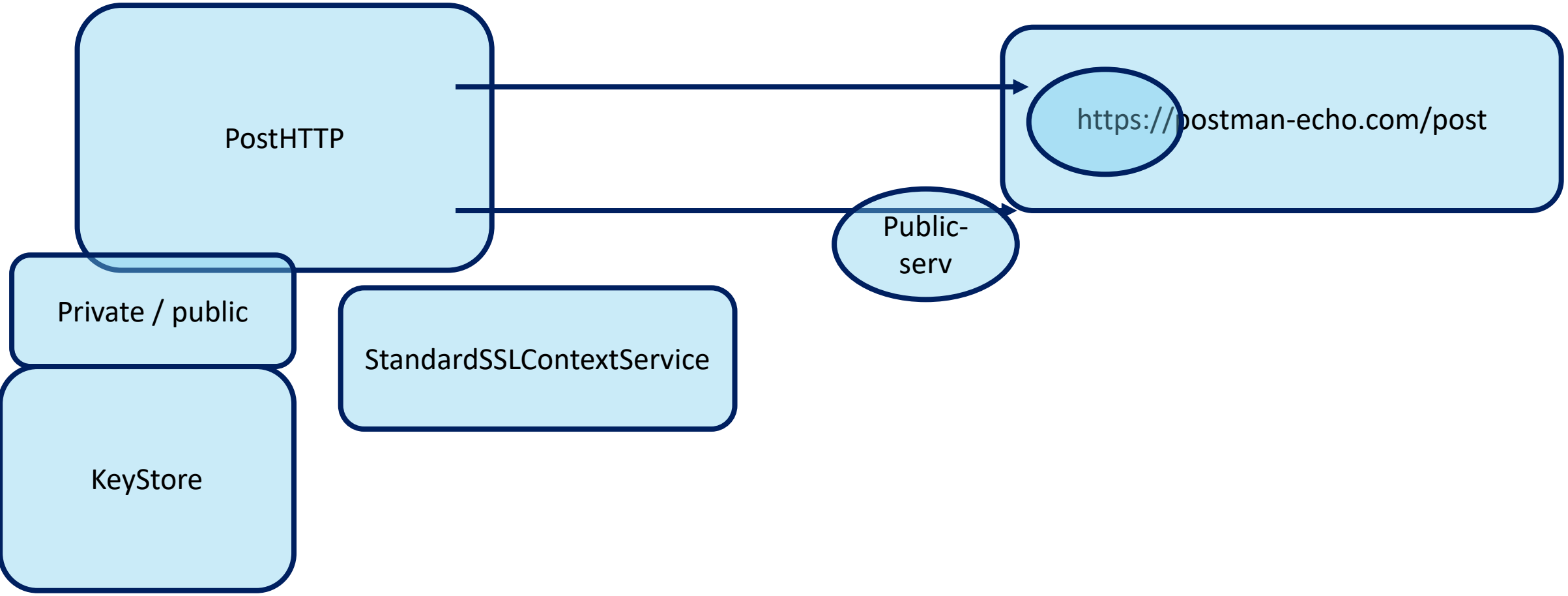
Similar to GetHTTP but sends the POST request.

It can process the response as flow file and push it upstream

Processor needs to be configured with Target Host

Supports both Success and Error Relations

Can accept incoming flowfiles


PostHTTP
https://postman-echo.com/post
Public-serv
Private / public
StandardSSLContextService
KeyStore

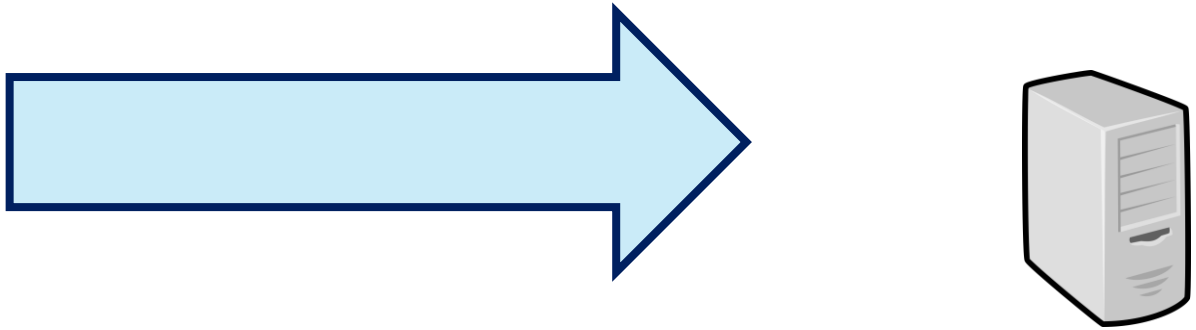
ListenHTTP

Very simple processor with only 1 relationship (Success)

Error handling not possible

Don’t have incoming connections

Very simple and good to have a simple HTTP endpoint for the external client to call



InvokeHTTP

Similar to GetHTTP and PostHTTP with lots of configurable options

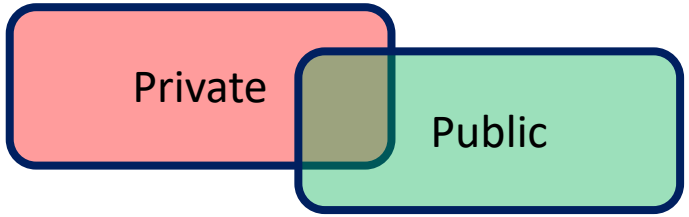
Has options to configure

Has lot more routing options like Original, Response, Failure, Retry, No Retry

url and method (POST, GET, PUT) can be dynamically configured

Allows incoming

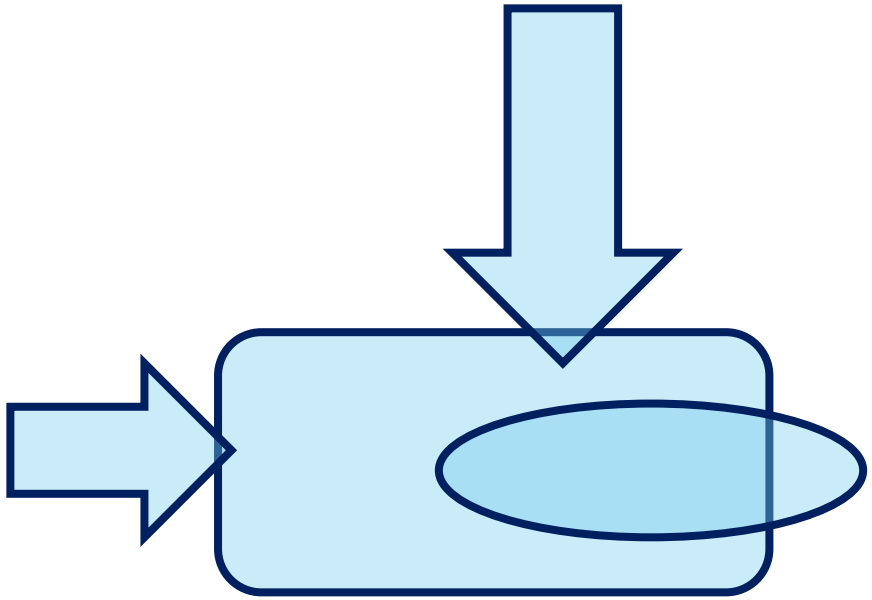
Can act as endpoint and convert the request to flowfile and push it upstream

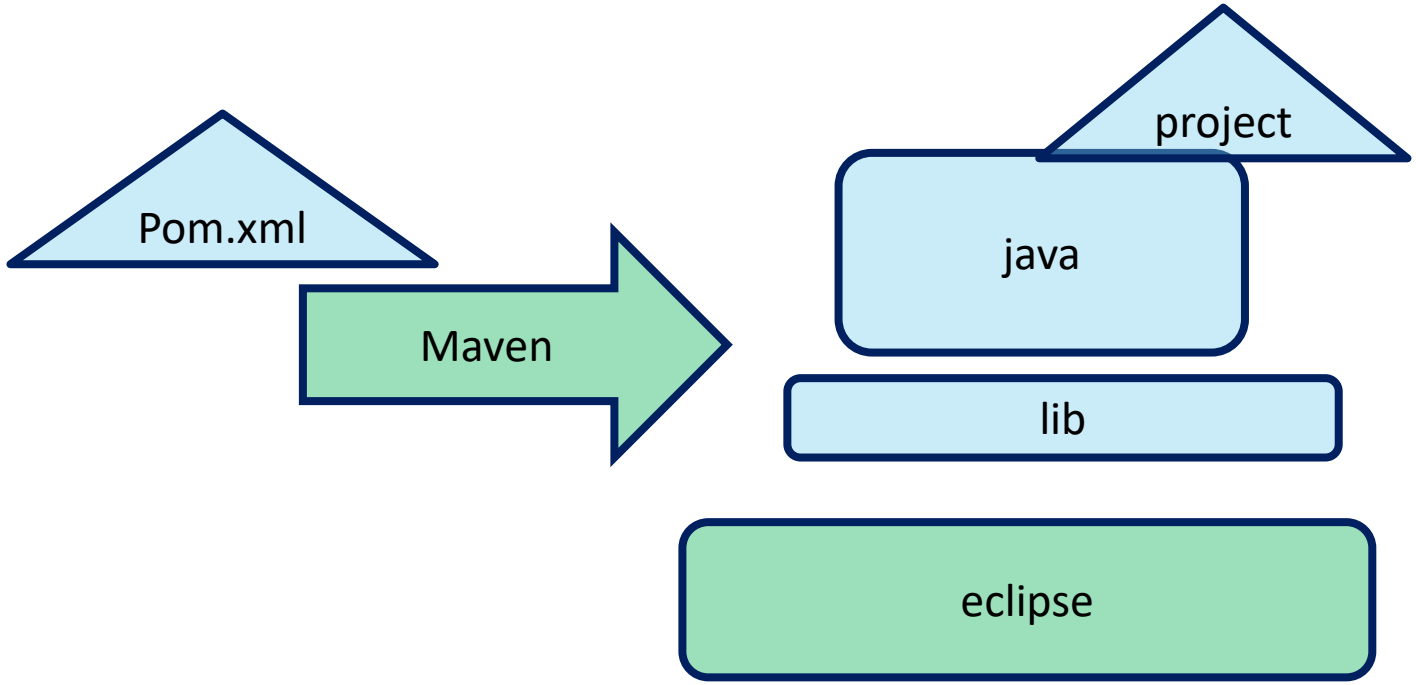

Private
Public


Private
Public
browser
Public
https://postman-echo.com

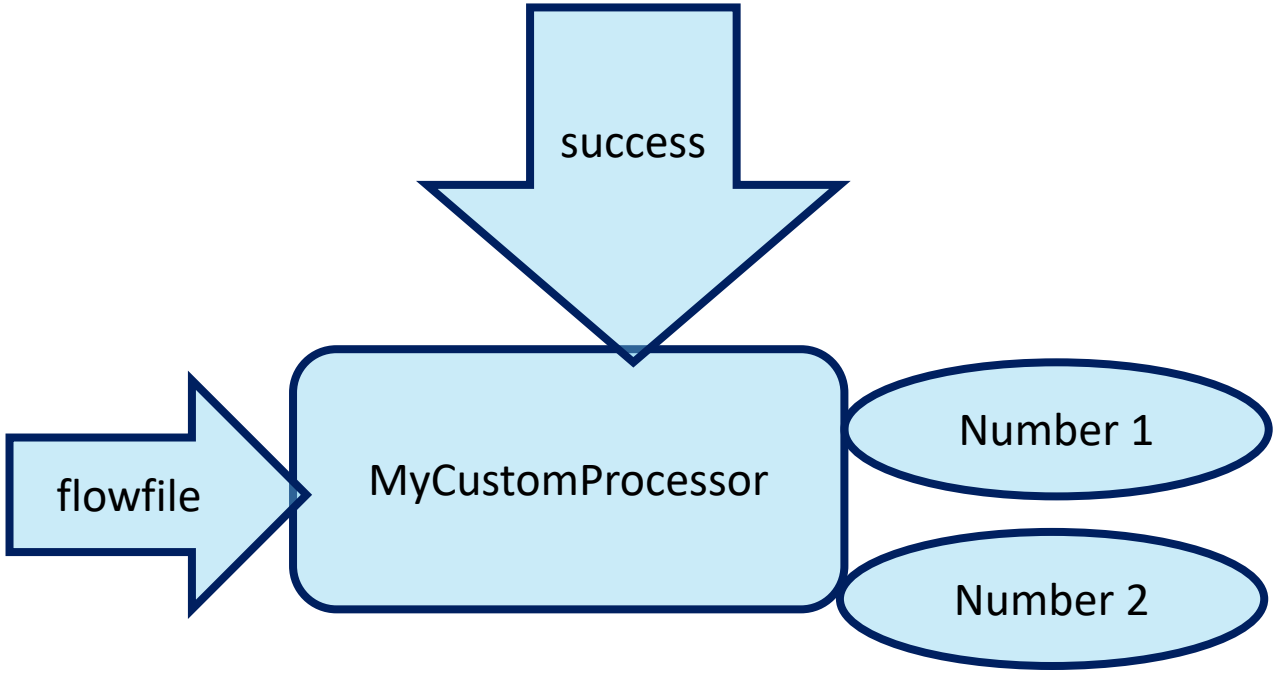
Use case 1 : Extract data from fordgobike

Custom Processor





Custom Processor



HandlerHttpRequest / HandlerHttpResponse

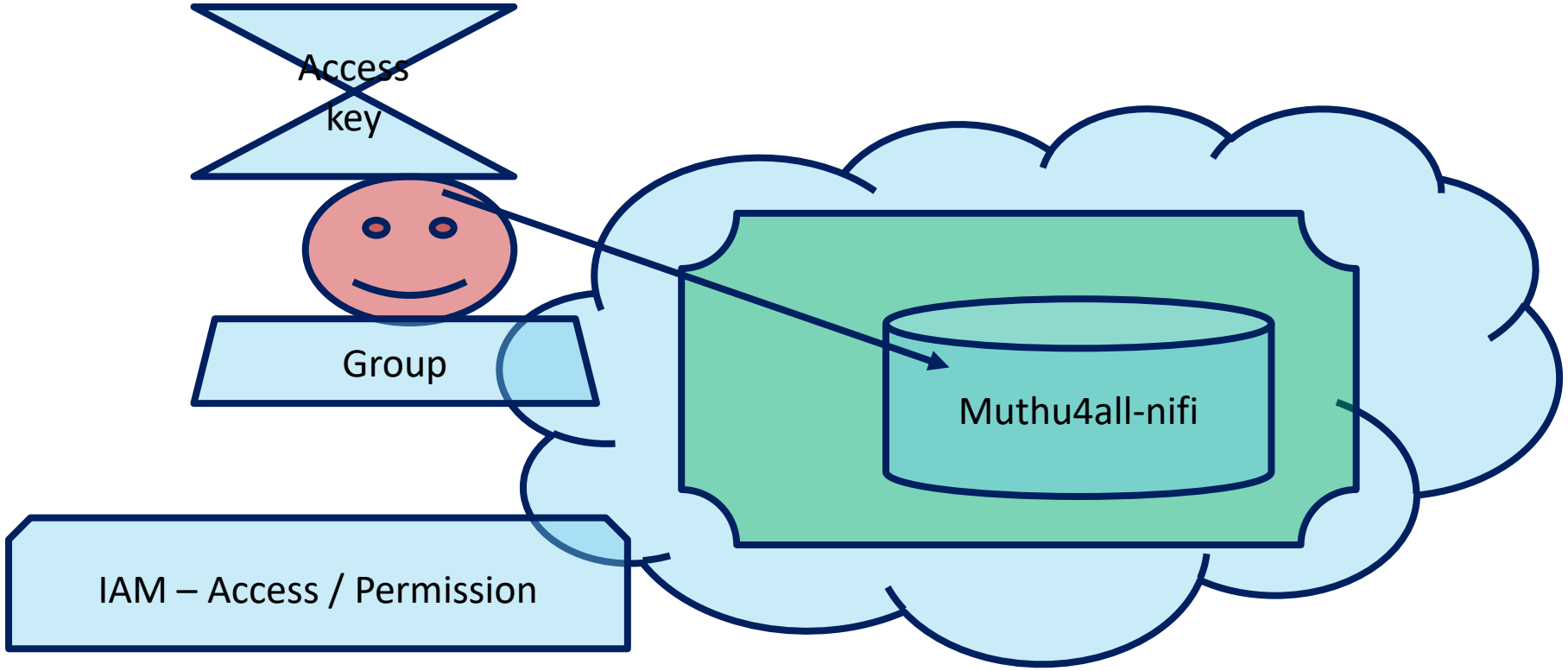
Both together can serve the request like webservice

Connection between request and response is to corelated with StandardHttpContextMap controller service

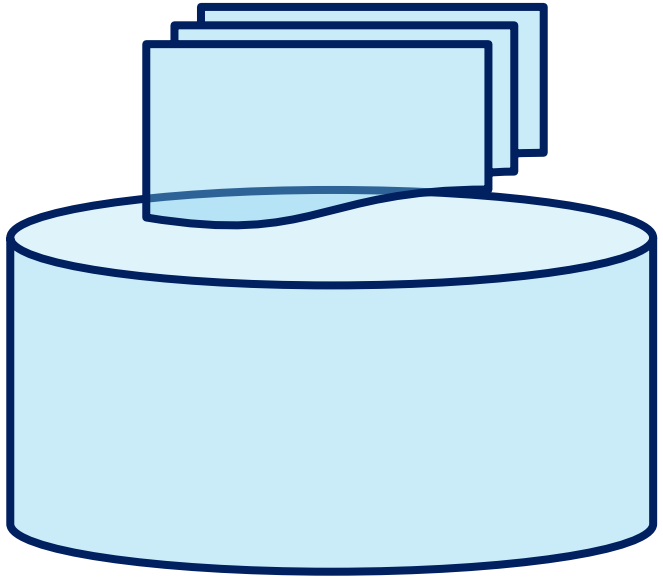
HandlerHttpReqest acts like server or ListenHTTP but it can take incoming flowfiles, and supports HTTP controller service like (StandardHttpContextMap)

Processor	Description	Pro	Con	Use-cases
ListenHTTP	Exposes an HTTP endpoint	Simple to setup	No incoming flowfiles	Provide a simple HTTP webhook into your flow
GetHTTP	Consume an HTTP (GET) endpoint	Simple to setup	No incoming flowfiles	Provide a simple HTTP client to poll an external http based resource
PostHTTP	Consume an HTTP (POST) endpoint	Simple to setup	No incoming flowfiles	Provide a simple HTTP client to poll an external http based resource
InvokeHTTP	Consume an HTTP endpoint	Supports EL / Incoming flowfiles		A much more dynamic GetHTTP flow
HandlerHttpRequest HandlerHttpResponse	Expose an HTTP endpoint and send a response	Allows you to implement a real webservice	2 Seperate processors to setup + controller service	Implement a webservice in your flow

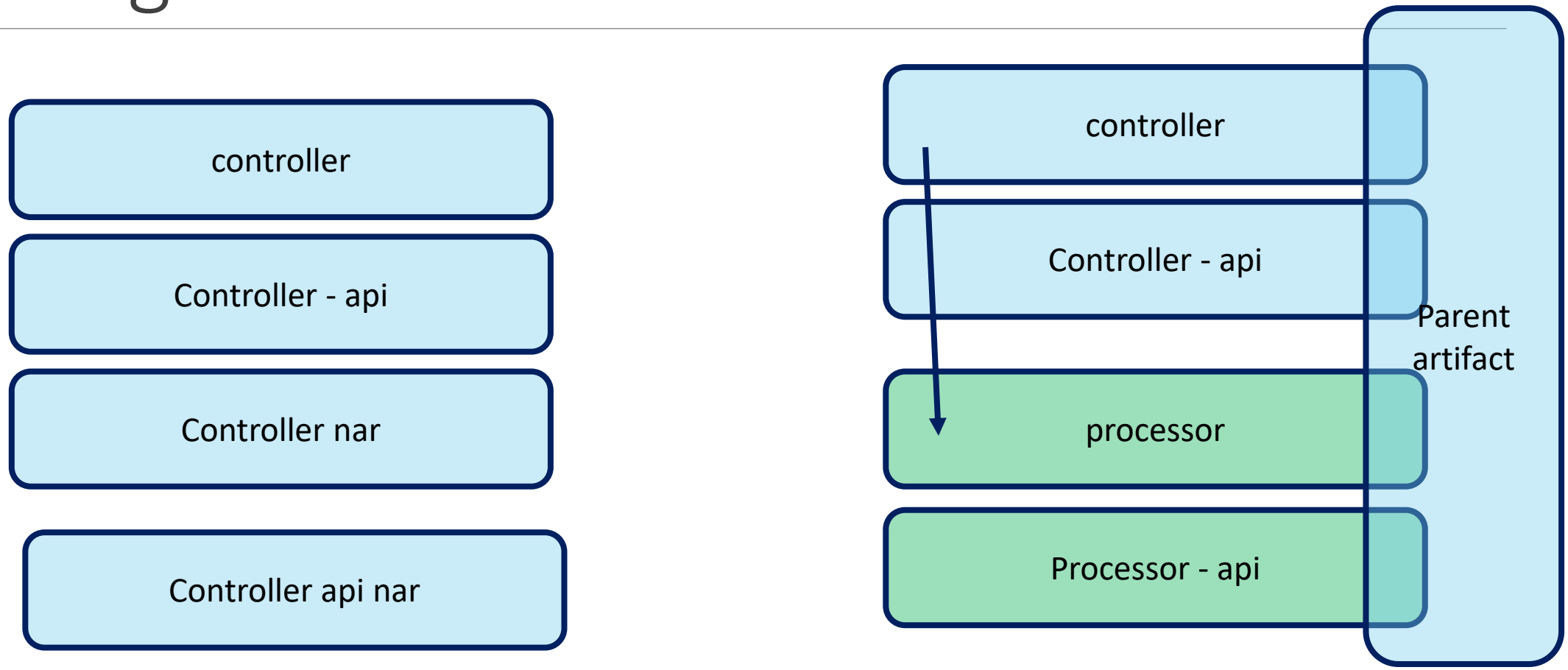
Nifi with AWS S3



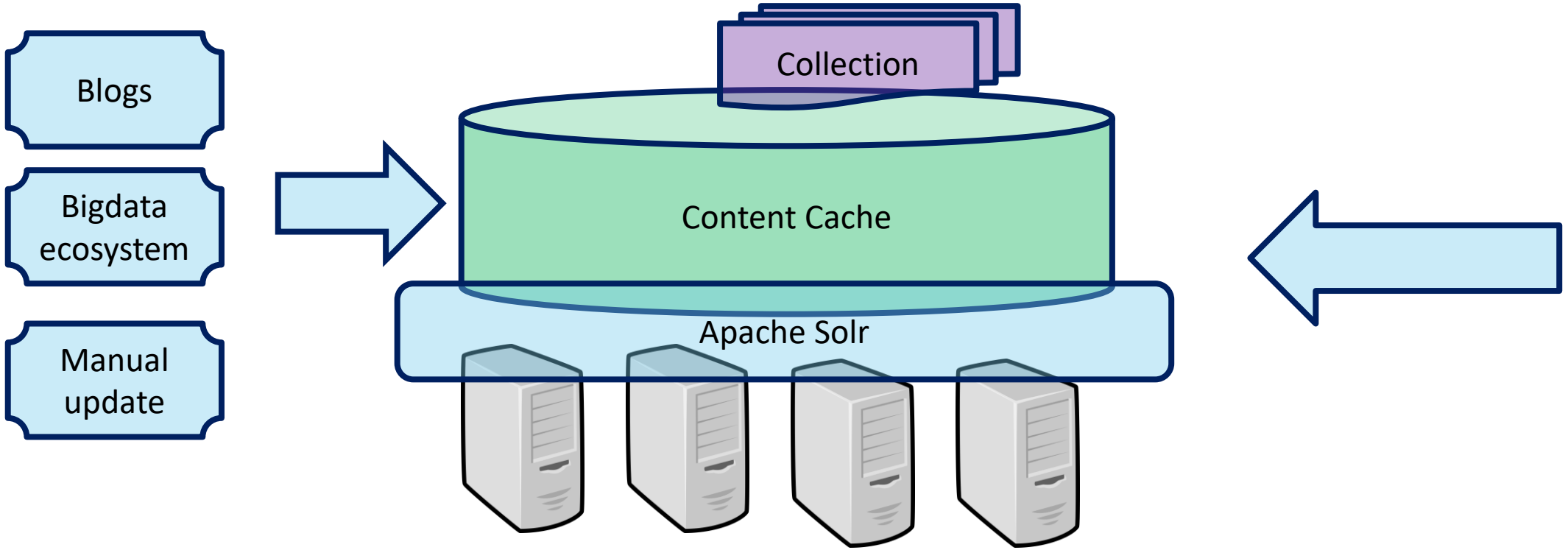
Nifi with MongoDB



Using Custom Controller



Nifi with Apache Solr



Twitter data Visualization