CLOUDERA

DataFlow Workshop
Apache NiFi

Contents

WHAT IS DATAFLOW MANAGEMENT?

DATAFLOW MANAGEMENT:
The systematic process by which data is acquired from all producers and delivered to all consumers

DATAFLOW MANAGEMENT CONSIDERATIONS

- Promote Loosely Coupled Systems
  - Types of coupling: Format, Schema, Protocol, Priority, Size, Interest, ….
- Promoted Highly Cohesive Systems
  - Producers should focus on production (not the intricacies of consumption)
  - Consumers should focus on storage or processing (not the details of production)
- Provide Provenance
  - The who/what/when/where/why of data
  - Inter and Intra Process Latency
  - Enable enterprise version control for data

DATAFLOW MANAGEMENT CONSIDERATIONS

- Empower Understanding and Interaction
  - Ability to see the flow, safely and quickly iterate and experiment
  - Breaking production is bad – so too is not being able to evolve fast enough
- Secure
  - Bridge between security domains
  - Data Plane (transport)
  - Control Plane (Command and Control, Monitoring)
- Self Service
  - Centralized teams – hard to scale – slow turnaround times
  - Centralized systems – multi-tenant management works

NIFI OVERVIEW

WHY NIFI?

- Moving data is multifaceted in its challenges and these are present in different contexts at varying scopes
- Provide common tooling and extensions that are commonly needed but be flexible for extension
  - Leverage existing libraries and expansive Java ecosystem for functionality
  - Allow organizations to integrate with their existing infrastructure
- Empower folks managing your infrastructure to make changes and reason about issues that are occurring
  - Data Provenance to show context and data's journey
  - User Interface/Experience a key component

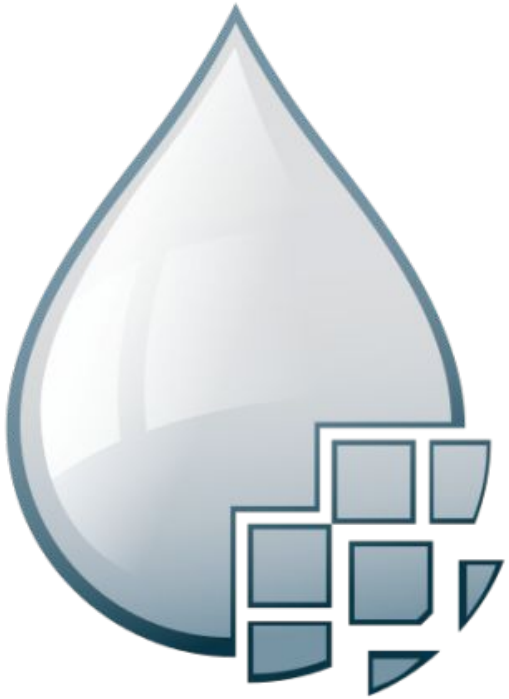
https://dzone.com/articles/apache-nifi-10-cheatsheet

THE NSA YEARS

- Created in 2006
- Improved over eight years
  - Simple initial vision – Visio for real-time dataflow management
- Key Lessons Learned
  - What scale means – down, up, and out
  - The fearsome force known as Compliance Requirements
  - The power of provenance!
  - Operational best-practices and anti-patterns
- NSA donated the codebase to the ASF in late 2014

APACHE NIFI

Key features



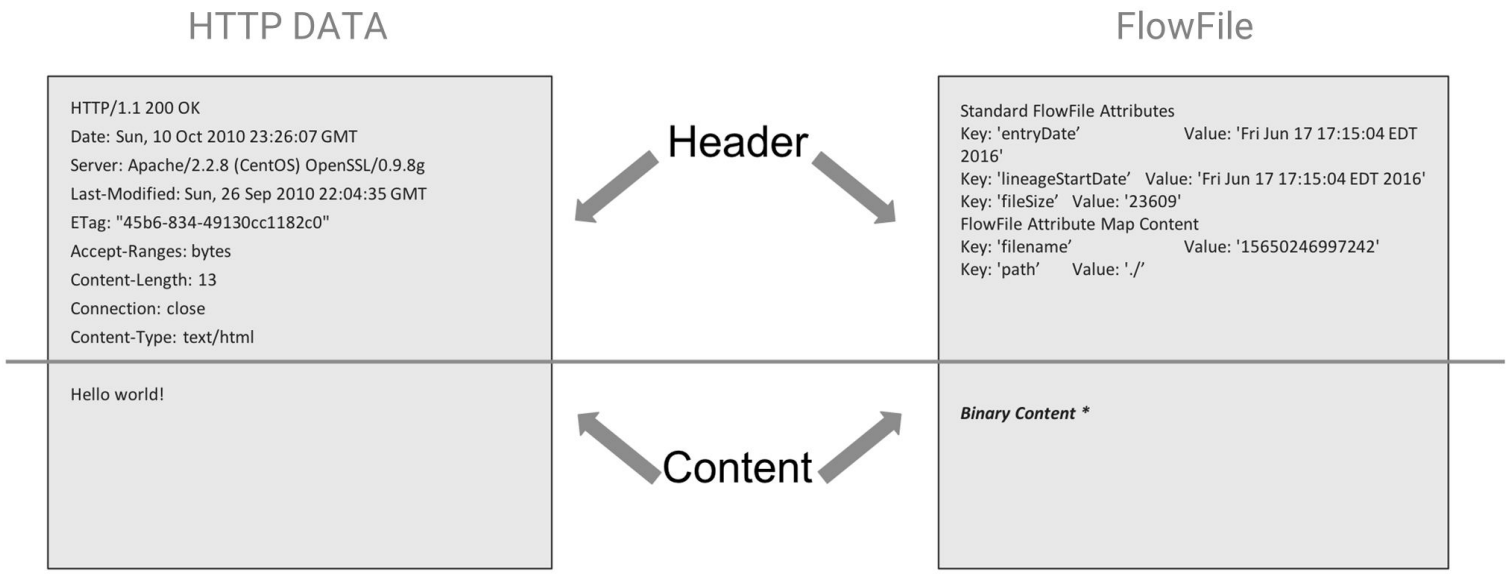
- Guaranteed delivery
- Data buffering
  - Backpressure
  - Pressure release
- Prioritized queuing
- Flow specific QoS
  - Latency vs. throughput
  - Loss tolerance
- Data provenance
- Supports push and pull models

- Recovery/recording a rolling log of fine-grained history
- Visual command and control
- Flow templates
- Pluggable/multi-role security
- Designed for extension
- Clustering

NIFI CORE CONCEPTS

NIFI—TERMINOLOGY

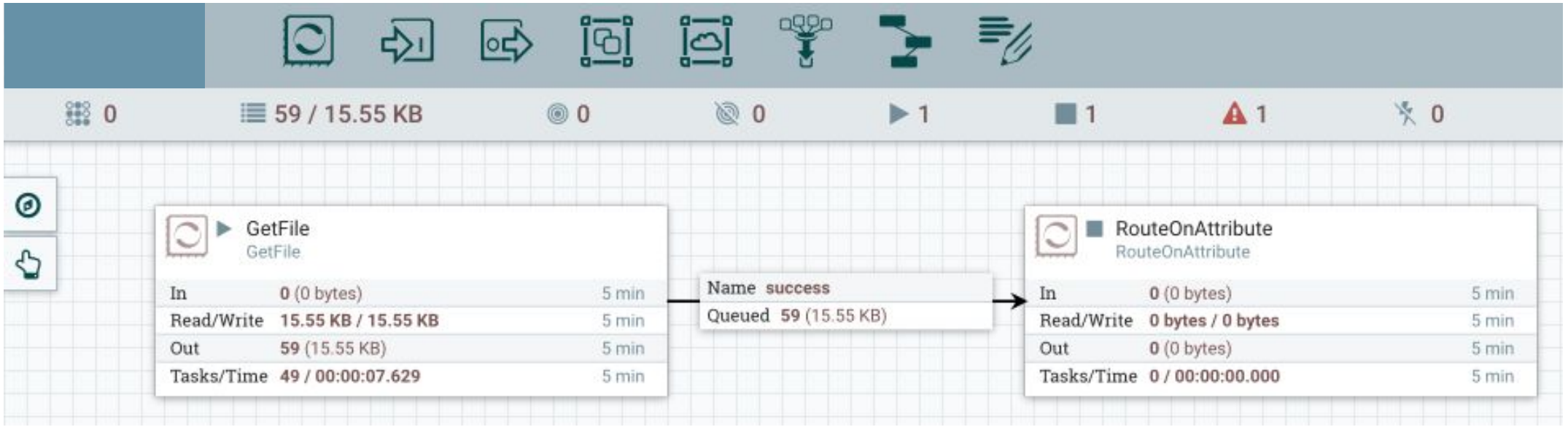
- FlowFile
  - Unit of data moving through the system
  - Content + Attributes (key/value pairs)
- Processor
  - Performs the work, can access FlowFiles
- Connection
  - Links between processors
  - Queues that can be dynamically prioritized
- Process Group
  - Set of processors and their connections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



NIFI IS BASED ON FLOW BASED PROGRAMMING (FBP)

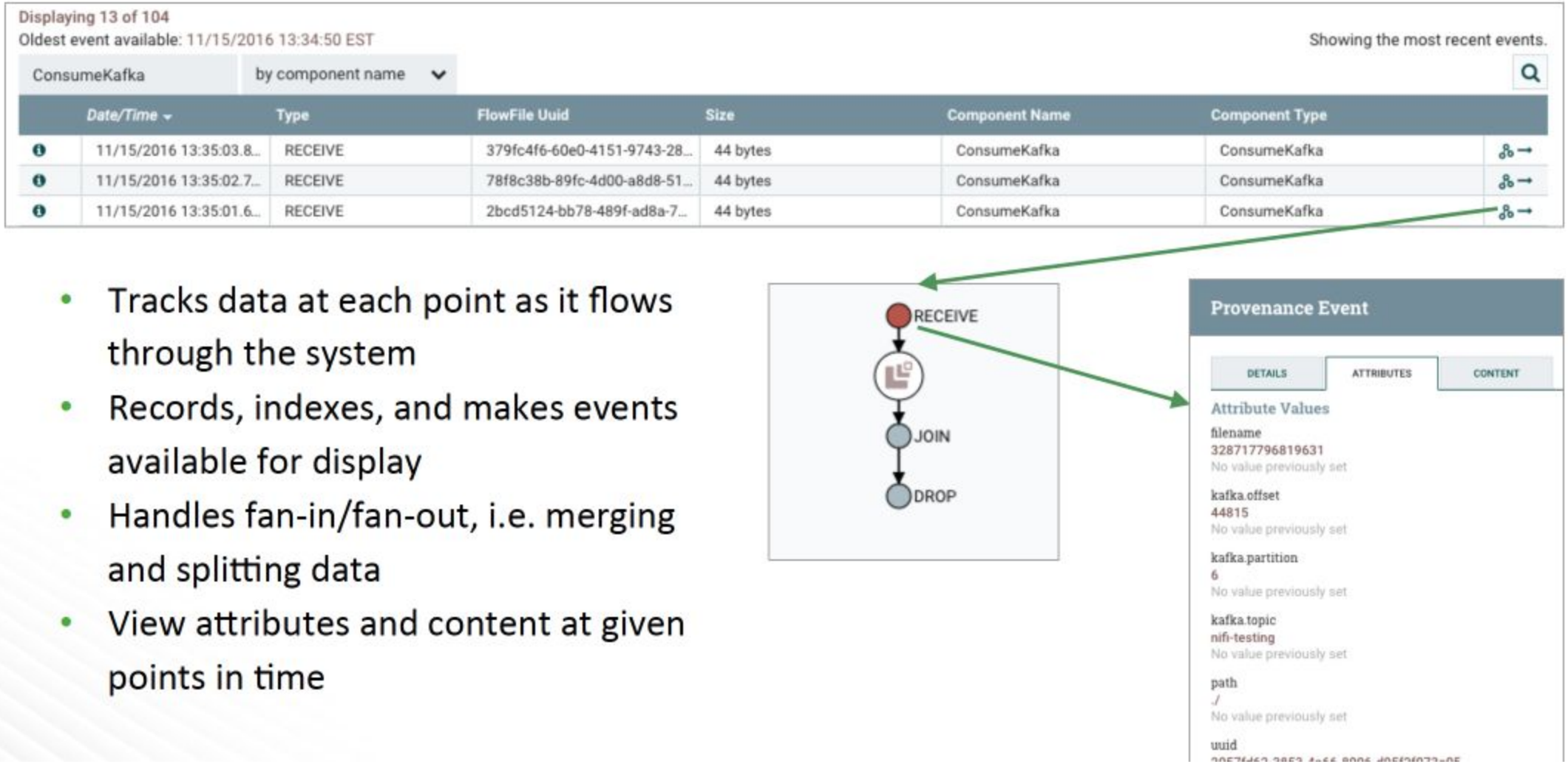
FBP Term	NiFi Term	Description
Information Packet	FlowFile	Each object moving through the system.
Black Box	FlowFile Processor	Performs the work, doing some combination of data routing, transformation, or mediation between systems.
Bounded Buffer	Connection	The linkage between processors, acting as queues and allowing various processes to interact at differing rates.
Scheduler	Flow Controller	Maintains the knowledge of how processes are connected, and manages the threads and allocations thereof which all processes use.
Subnet	Process Group	A set of processes and their connections, which can receive and send data via ports. A process group allows creation of entirely new component simply by composition of its components.

VISUAL COMMAND AND CONTROL



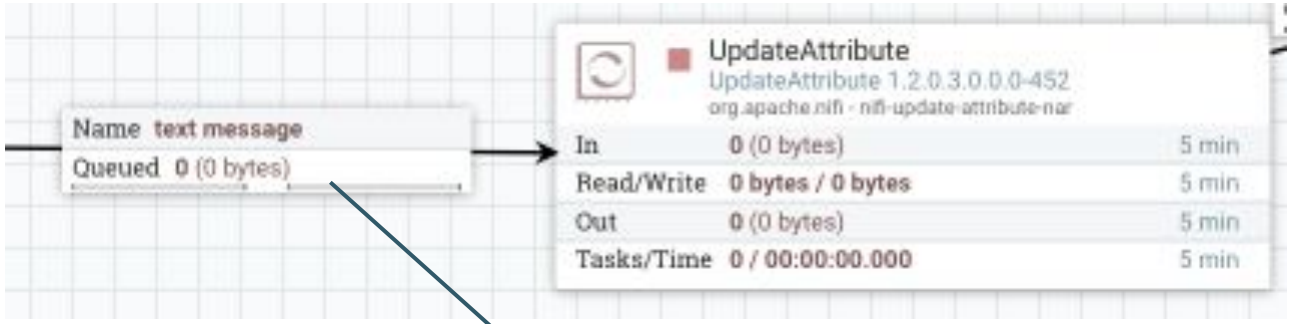
- Drag and drop processors to build a flow
- Start, stop, and configure components in real time
- View errors and corresponding error messages
- View statistics and health of data flow
- Create templates of common processor & connections

PROVENANCE/LINEAGE

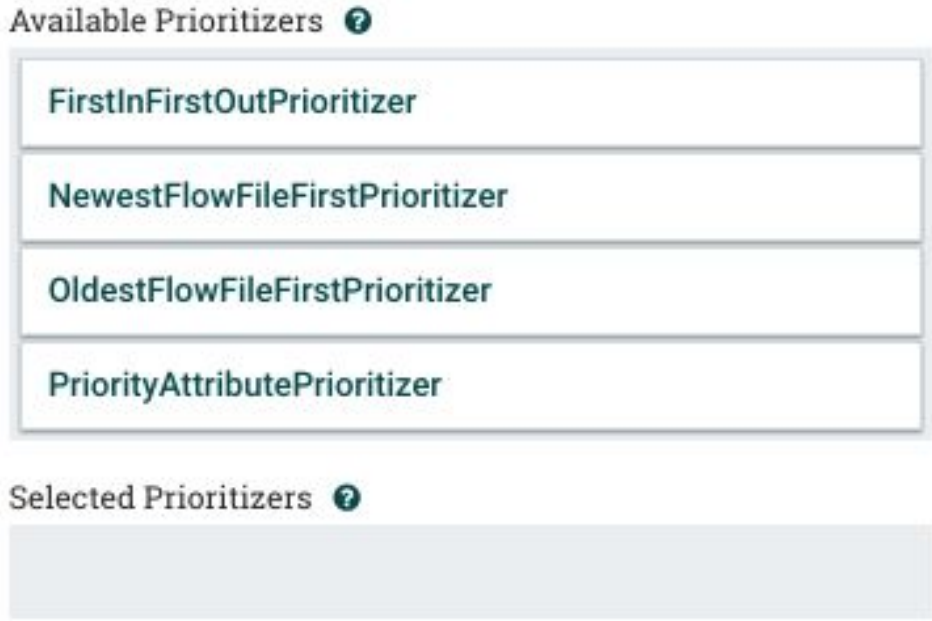


- Tracks data at each point as it flows through the system
- Records, indexes, and makes events available for display
- Handles fan-in/fan-out, i.e. merging and splitting data
- View attributes and content at given points in time

PRIORITIZATION

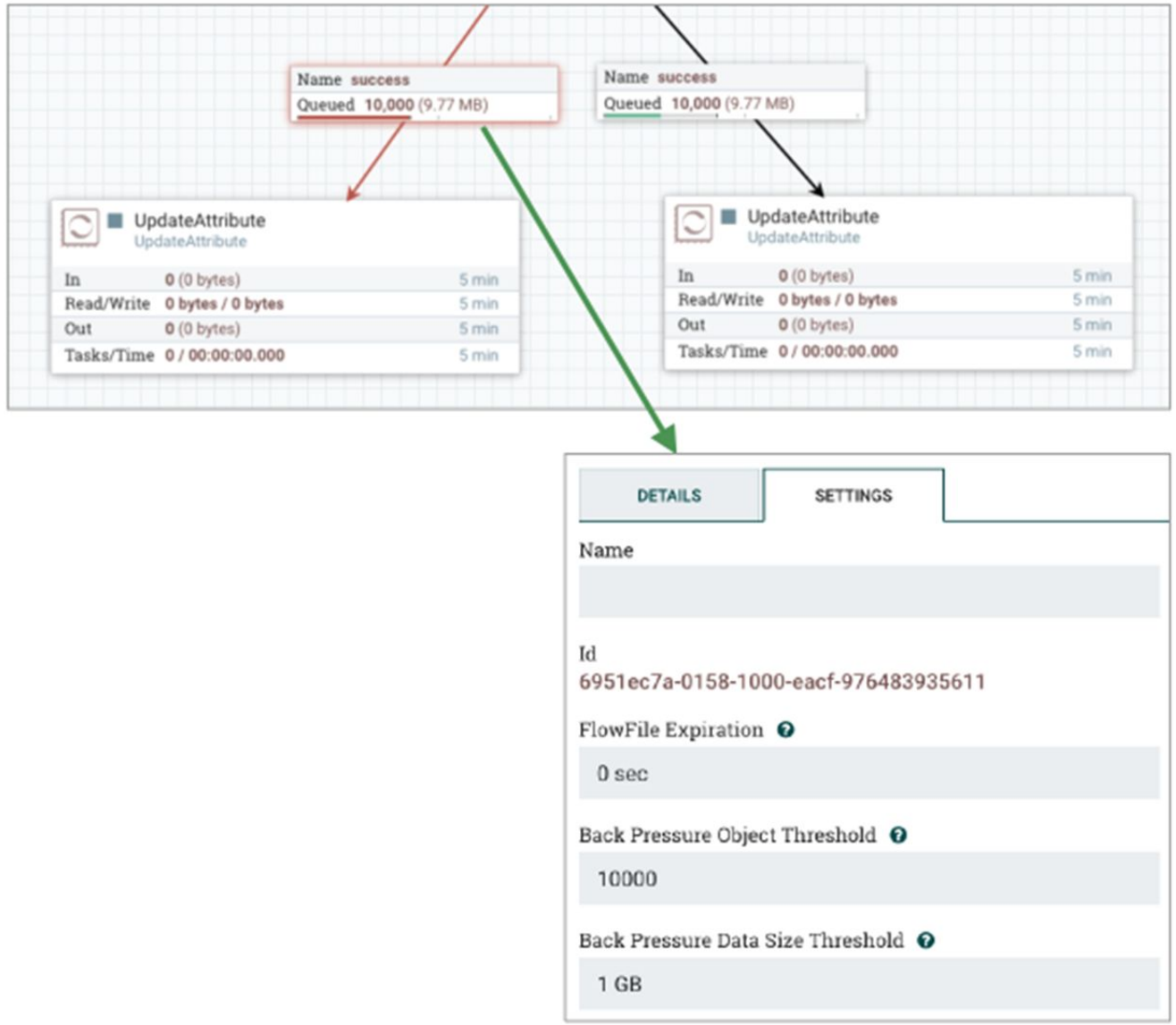


- Configure a prioritizer per connection
- Determine what is important for your data – time based, arrival order, importance of a data set
- Funnel many connections down to a single connection to prioritize across data sets
- Develop your own prioritizer if needed



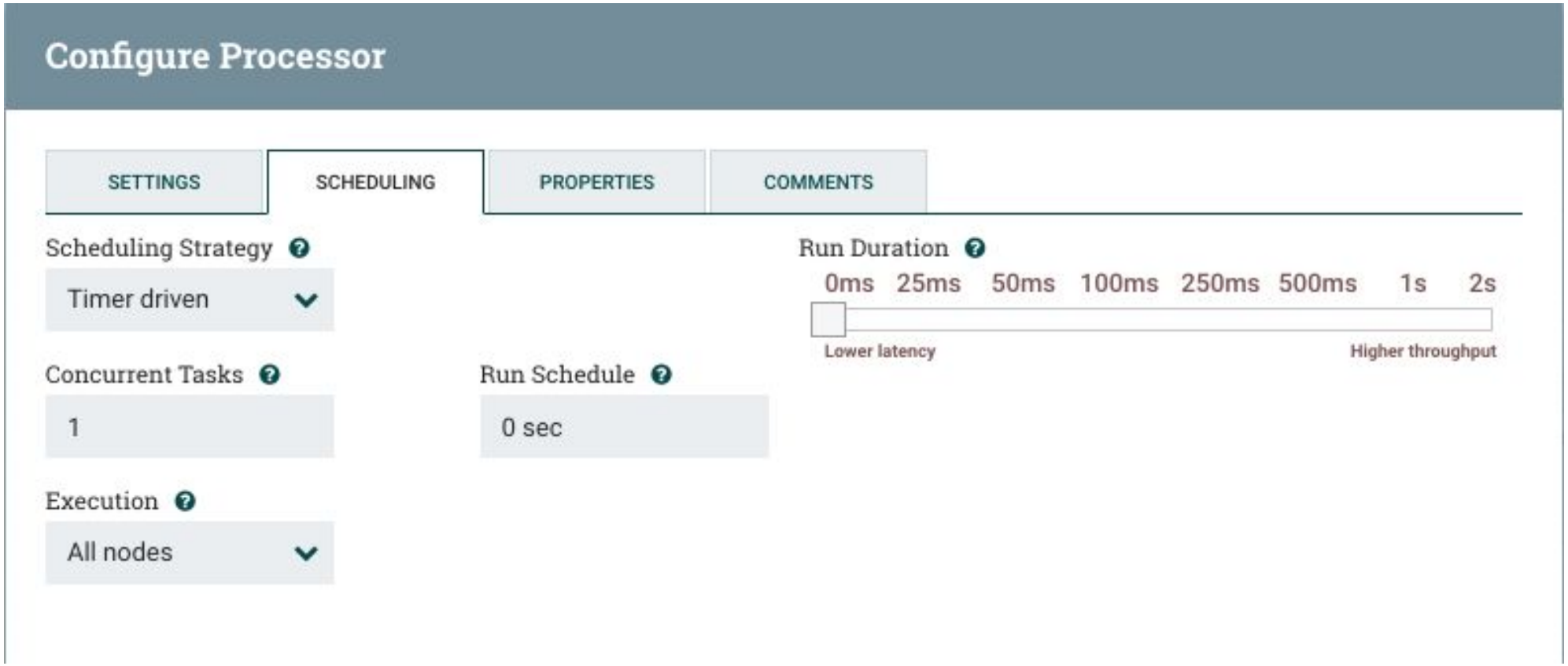
BACK PRESSURE

- Configure back-pressure per connection
- Based on number of FlowFiles or total size of FlowFiles
- Upstream processor no longer scheduled to run until below threshold



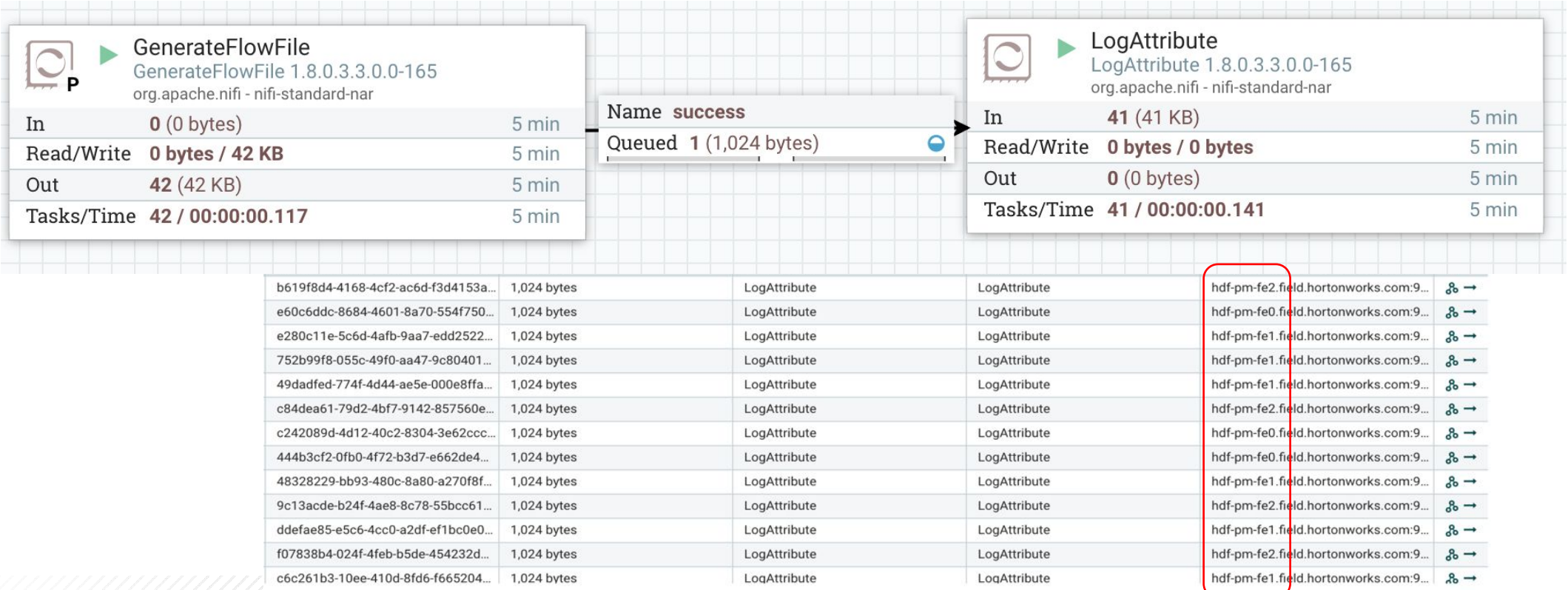
LATENCY VS. THROUGHPUT

Choose between lower latency, or higher throughput on each processor



CONNECTION LOAD BALANCING

- Improve NiFi cluster throughput
- Defined at connection level
- Configurable balancing strategies
- Critical for scale up paradigm in Kubernetes
- Alleviates S2S balancing “hack” customers use



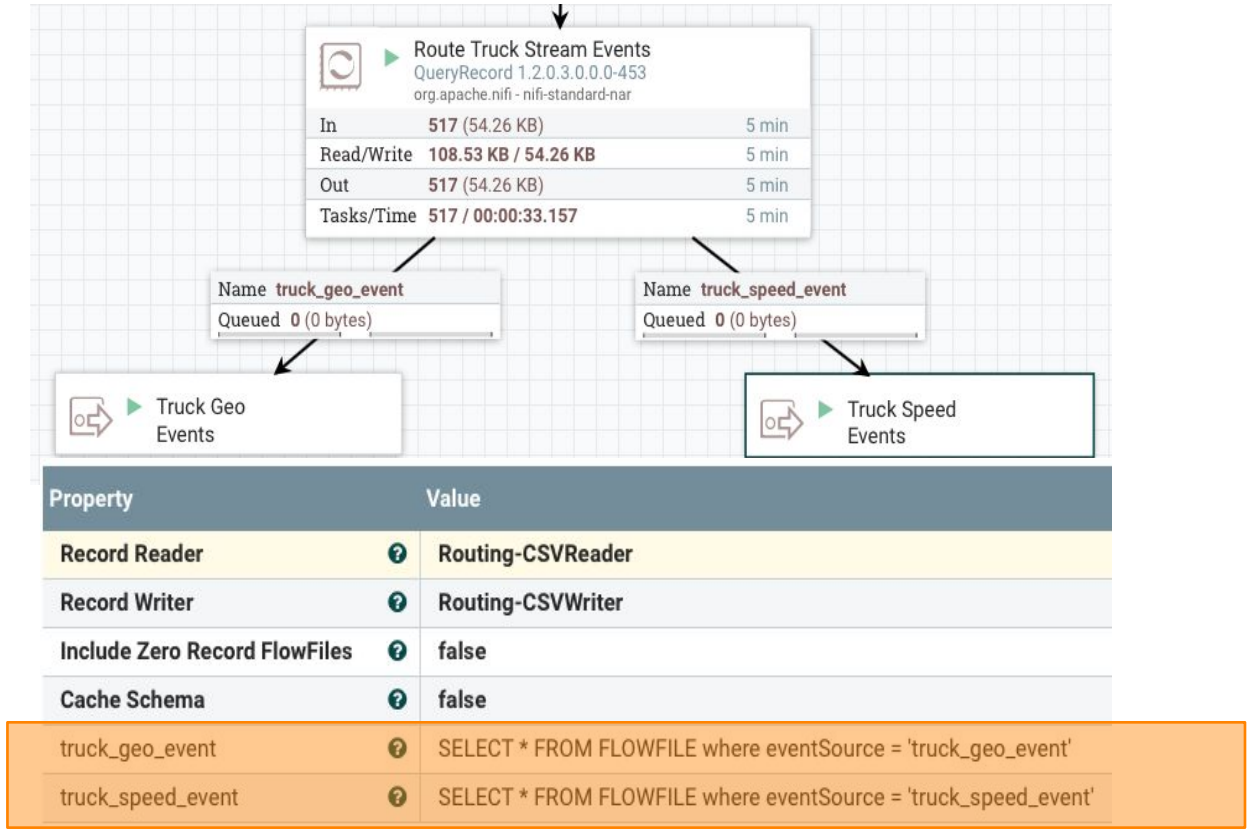
SQL BASED ROUTING WITH NiFi's QueryRecord Processor

QueryRecord Processor- Executes a SQL statement against records and writes the results to the flow file content.

CSVReader: Looking up schema from SR, it will converts CSV Records into ProcessRecords

SQL execution via Apache Calcite: execute configured SQL against the ProcessRecords for routing

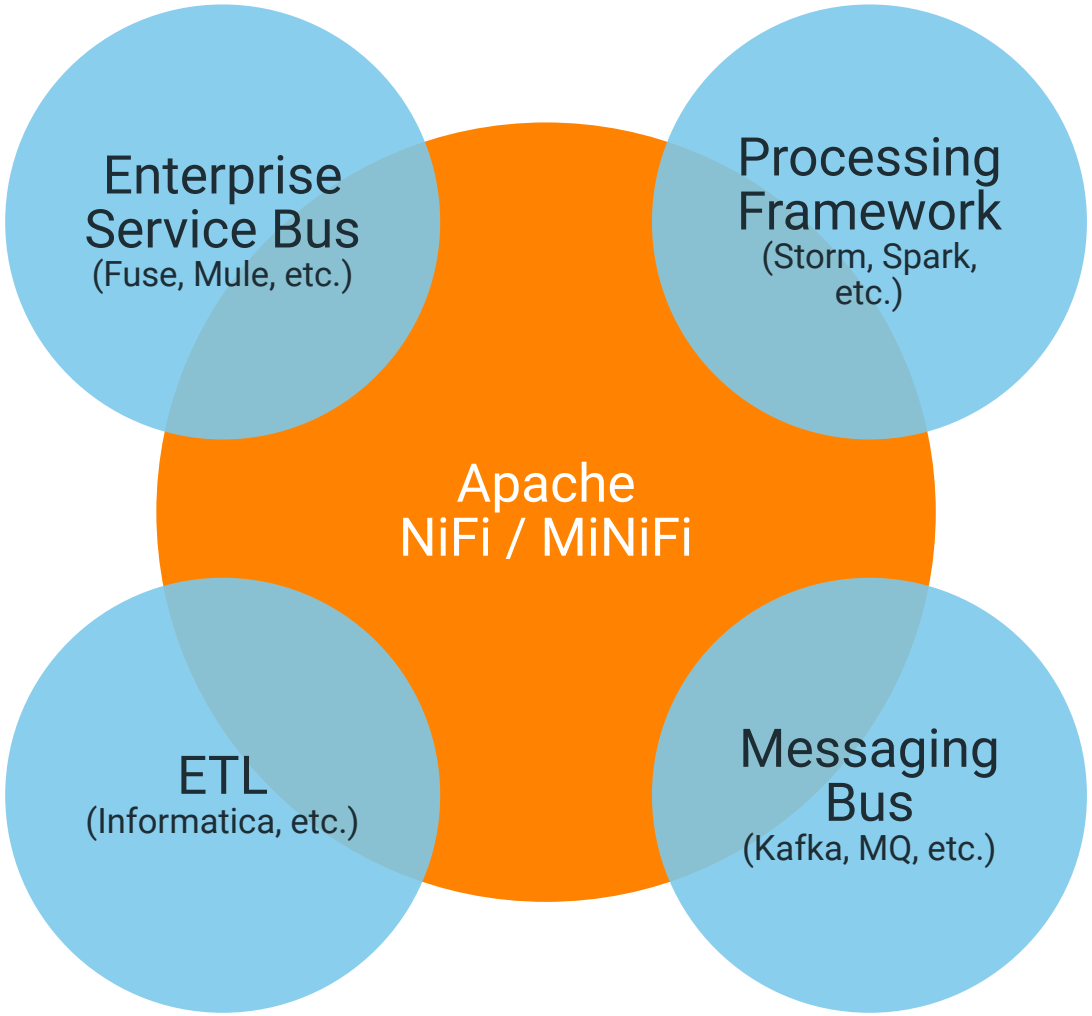
CSVRecordSetWriter: Converts the result of the query from Process records into CSV for the for the flow file content



Do routing(routing geo and speed streams) using standard SQL as opposed to complex regular expressions.

NIFI POSITIONING

NIFI POSITIONING



APACHE NIFI / PROCESSING FRAMEWORKS

NiFi

- Simple event processing
- Primarily feed data into processing frameworks, can process data, with a focus on simple event processing
- Operate on a single piece of data, or in correlation with an enrichment dataset (enrichment, parsing, splitting, and transformations)
- Can scale out, but scale up better to take full advantage of hardware resources, run concurrent processing tasks/threads (processing terabytes of data per day on a single node)

⚠ Not another distributed processing framework, but to feed data into those

Processing Frameworks (Storm, Kafka Streams, Spark, etc.)

- Complex and distributed processing
- Complex processing from multiple streams (JOIN operations)
- Analyzing data across time windows (rolling window aggregation, standard deviation, etc.)
- Scale out to thousands of nodes if needed

⚠ Not designed to collect data or manage data flow

APACHE NIFI / MESSAGING BUS SERVICES

NiFi

- Provide dataflow solution
- Centralized management, from edge to core
- Great traceability, event level data provenance starting when data is born
- Interactive command and control – real time operational visibility
- Dataflow management, including prioritization, back pressure, and edge intelligence
- Visual representation of global dataflow

⚠ Not a messaging bus, flow maintenance needed when you have frequent consumer side updates

Messaging Bus (Kafka, Pulsar, JMS, etc.)

- Provide messaging bus service
- Low latency
- Great data durability
- Decentralized management (producers & consumers)
- Low broker maintenance for dynamic consumer side updates

⚠ Not designed to solve dataflow problems (prioritization, edge intelligence, etc.)

⚠ Traceability limited to in/out of topics, no lineage

⚠ Lack of global view of components/connectivities

APACHE NIFI / INTEGRATION, OR INGESTION, FRAMEWORKS

NiFi

- End user facing dataflow management tool
- Out of the box solution for dataflow management
- Interactive command and control in the core, design and deploy on the edge
- Flexible failure handling at each point of the flow
- Visual representation of global dataflow and connectivities
- Native cross data center communication
- Data provenance for traceability

⚠ Not a library to be embedded in other applications

Integration framework (Spring Integration, Camel, etc), ingestion framework (Flume, etc)

- Developer facing integration tool with a focus on data ingestion
- A set of tools to orchestrate workflow
- A fixed design and deploy pattern
- Leverage messaging bus across disconnected networks

⚠ Developer facing, custom coding needed to optimize

⚠ Pre-built failure handling, lack of flexibility

⚠ No holistic view of global dataflow

⚠ No built-in data traceability

APACHE NIFI / ETL TOOLS

NiFi

- NOT schema dependent
- Dataflow management for both structured and unstructured data, powered by separation of metadata and payload
- Schema is not required, but you can have schema
- Minimum modeling effort, just enough to manage dataflows
- Do the plumbing job, maximize developers' brainpower for creative work

⚠ Not designed to do heavy lifting transformation work for DB tables (JOIN datasets, etc.). You can create custom processors to do that, but long way to go to catch up with existing ETL tools from user experience perspective (GUI for data wrangling, cleansing, etc.)

ETL (Informatica, etc.)

- Schema dependent
- Tailored for Databases/WH
- ETL operations based on schema/data modeling
- Highly efficient, optimized performance

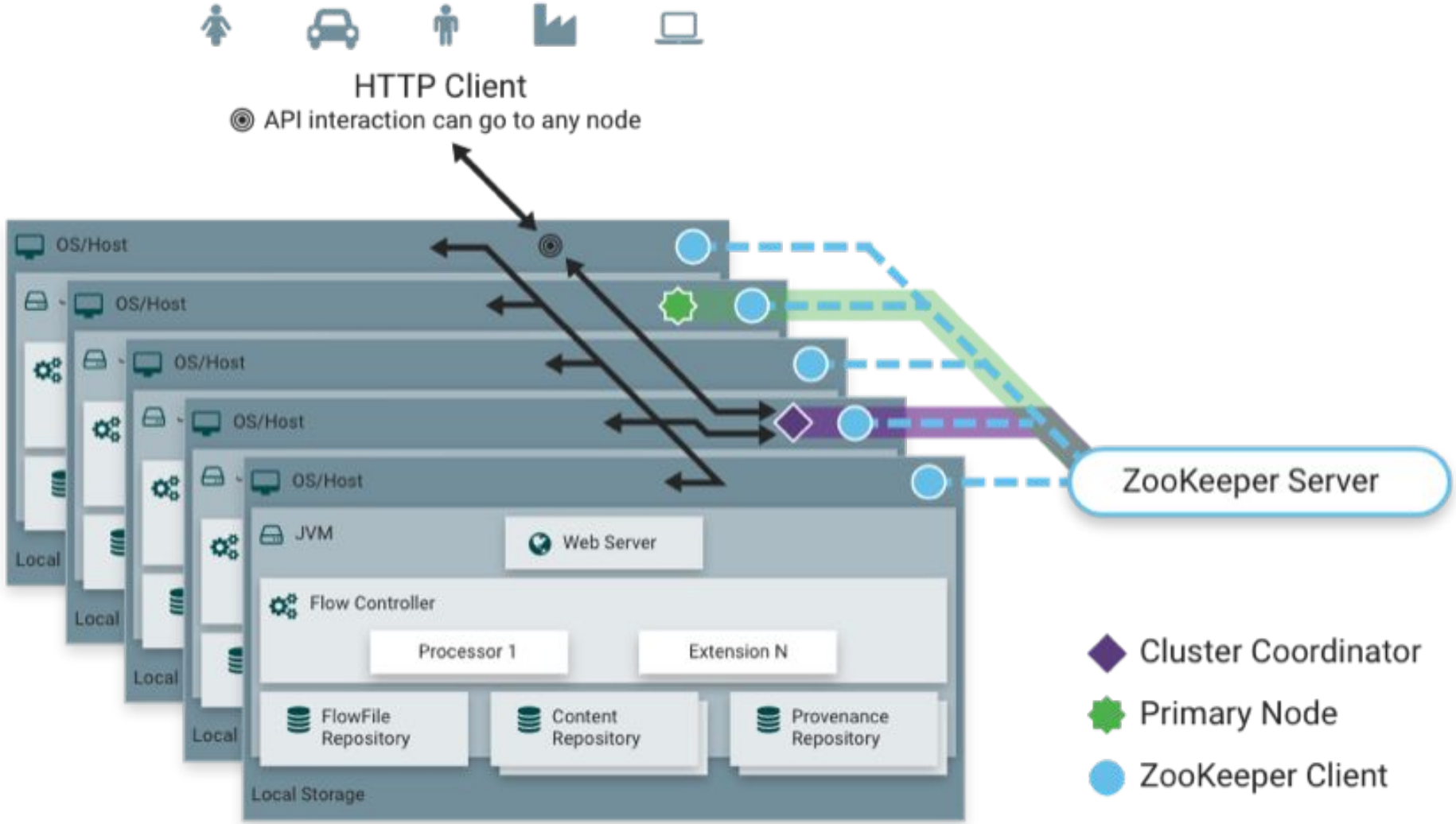
⚠ Must pre-prepare your data, time consuming to build data modeling, and maintain schemas

⚠ Not geared towards handling unstructured data, PDF, Audio, Video, etc.

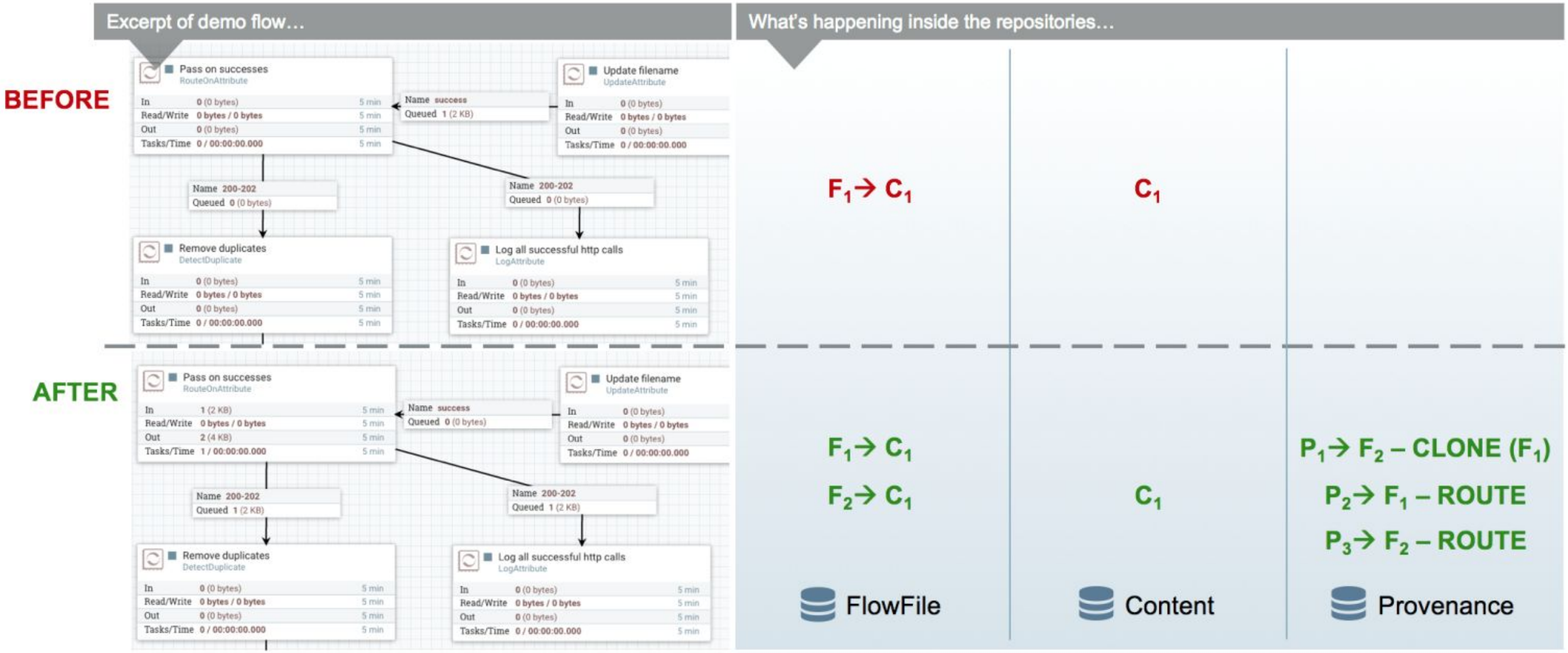
⚠ Not designed to solve dataflow problems

NIFI ARCHITECTURE

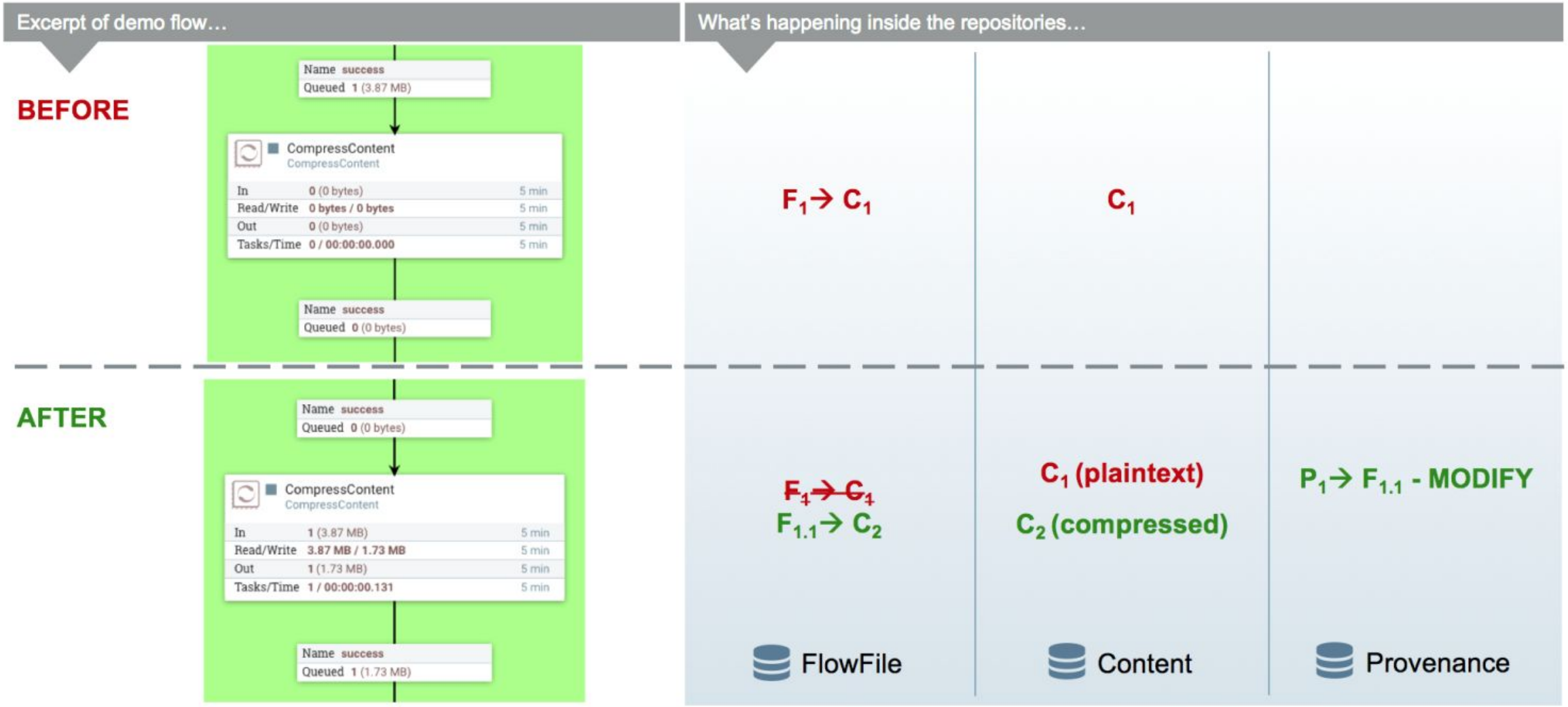
ARCHITECTURE



NIFI ARCHITECTURE – REPOSITORIES - PASS BY REFERENCE



NIFI ARCHITECTURE – REPOSITORIES – COPY ON WRITE



PERFORMANCE & SCALING

- Optimize I/O…
  - Separate partition for each repository
  - Multiple partitions for content repository
  - RAID configurations for redundancy & striping
- Tune JVM Memory, GC, and # of threads
- Scale up with a cluster
  - 100s of thousands of events per second per node

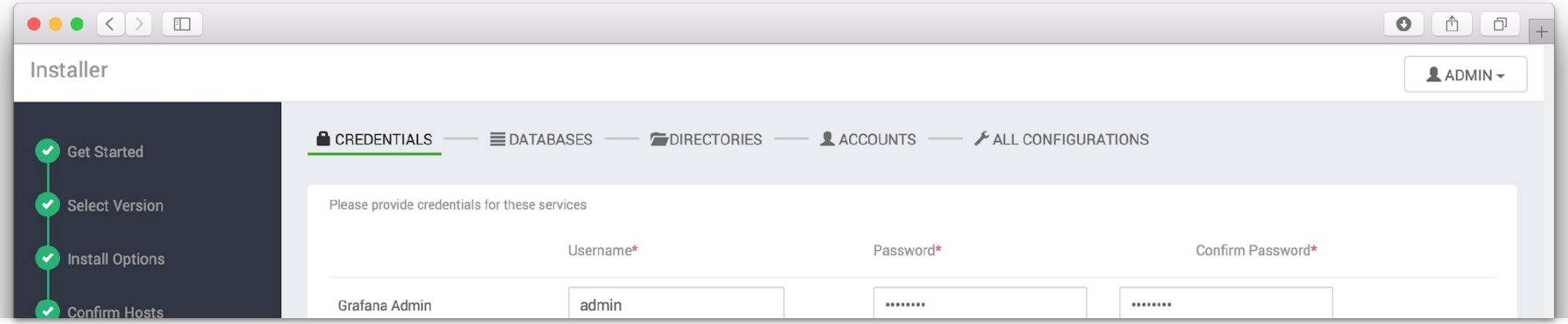
APACHE AMBARI

AMBARI - CENTRALIZED CONFIGURATION MANAGEMENT

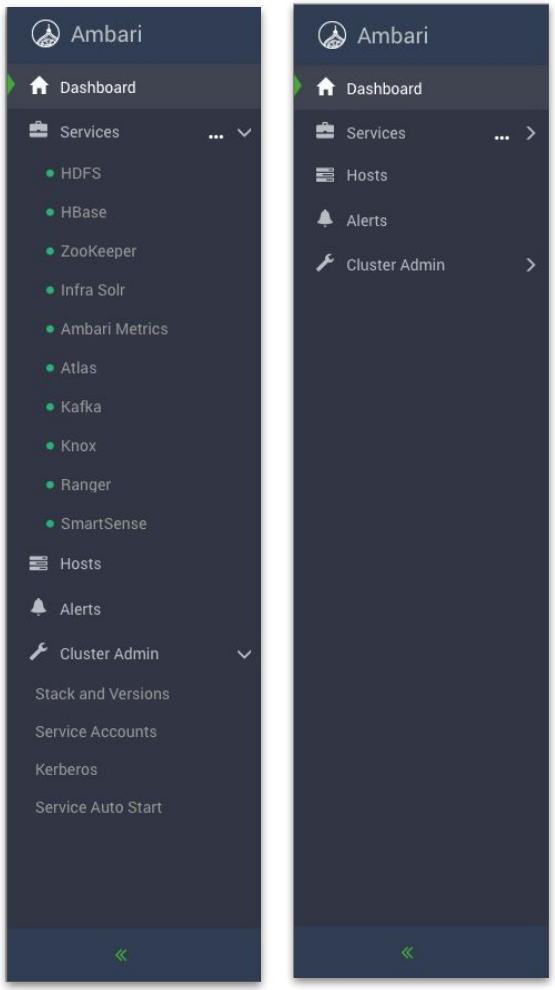
Most Commonly Modified Configurations

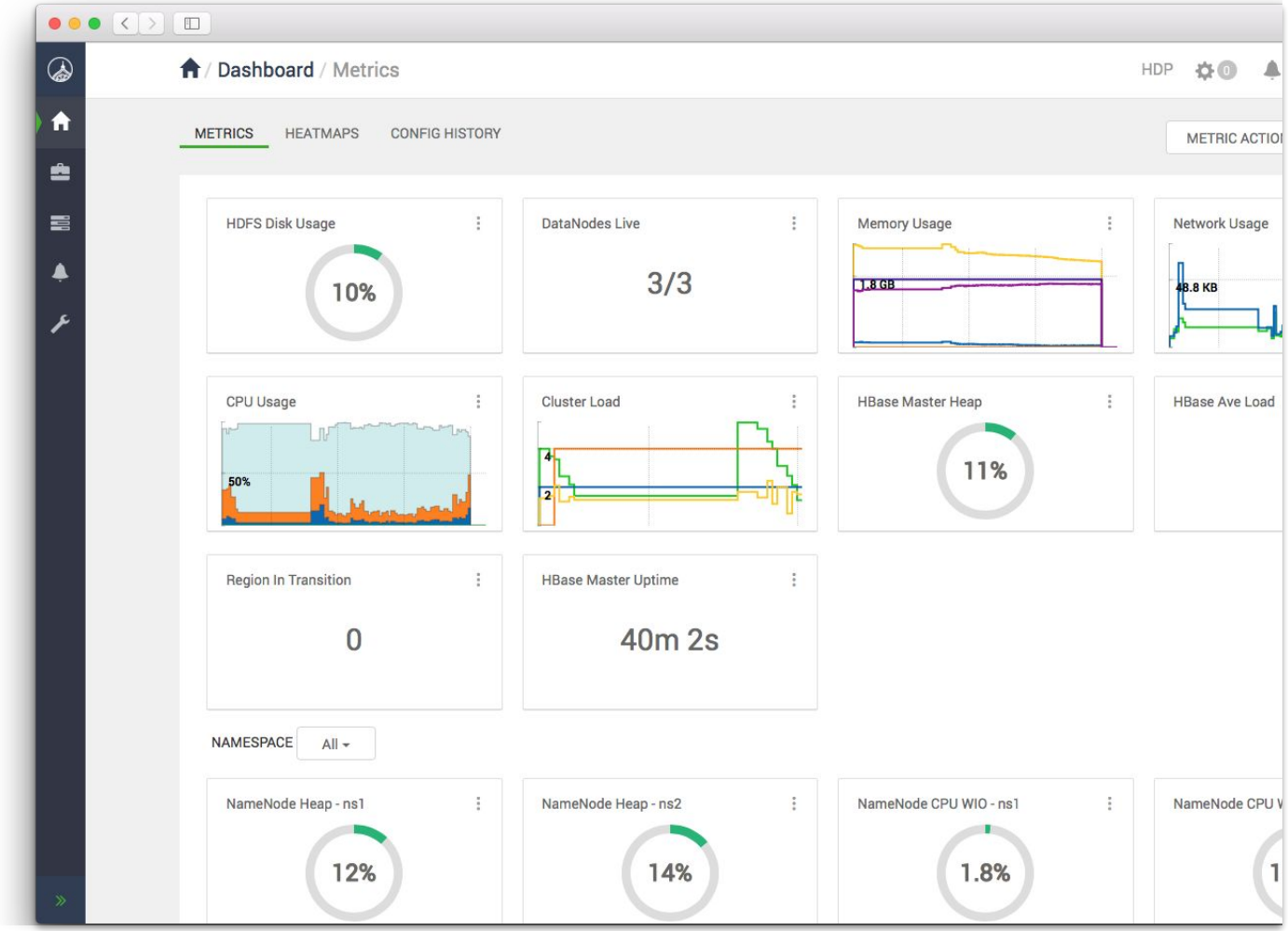
CREDENTIALS | DATABASES | DIRECTORIES | ACCOUNTS

Now all in one place!

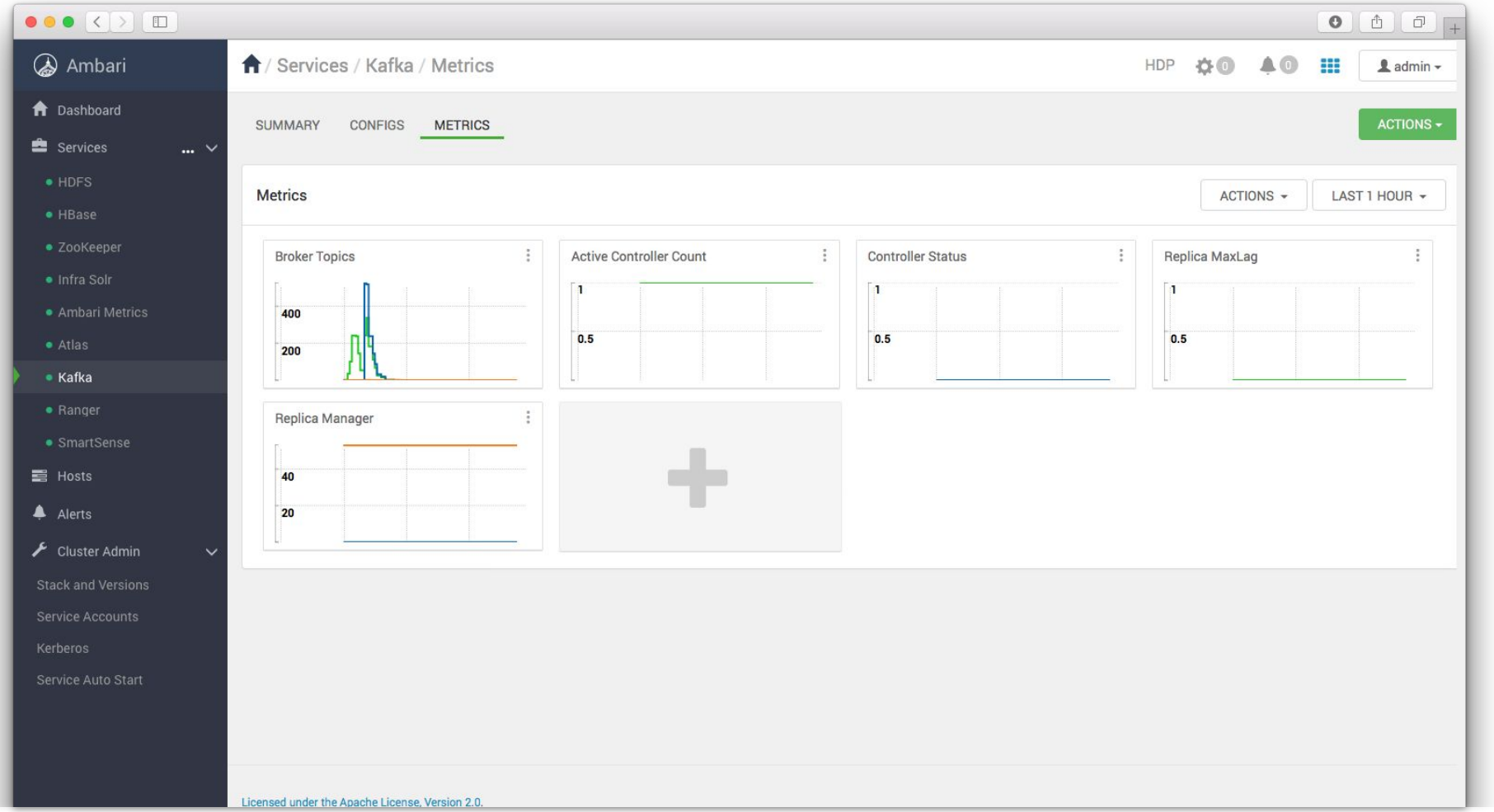


AMBARI - NAVIGATION

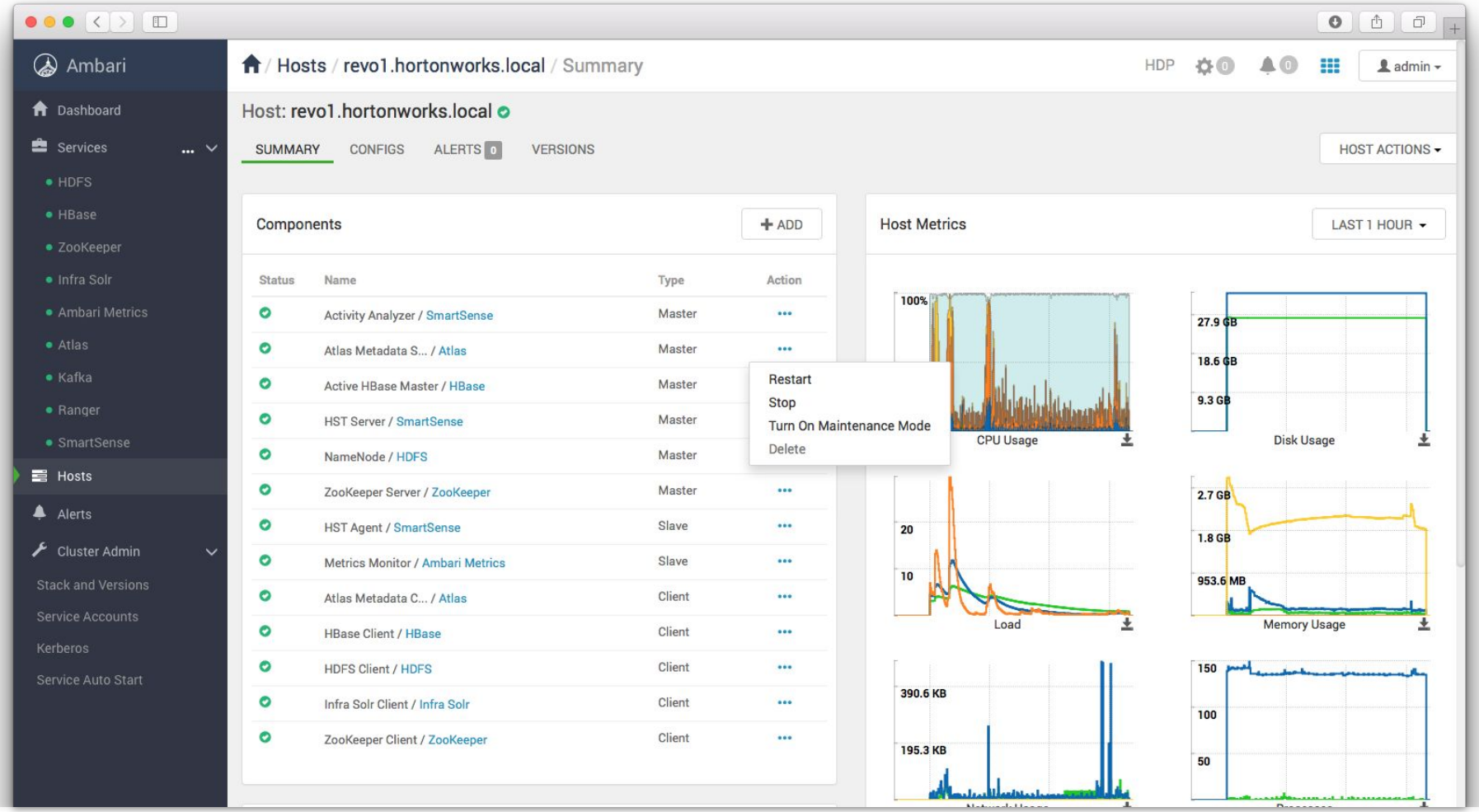




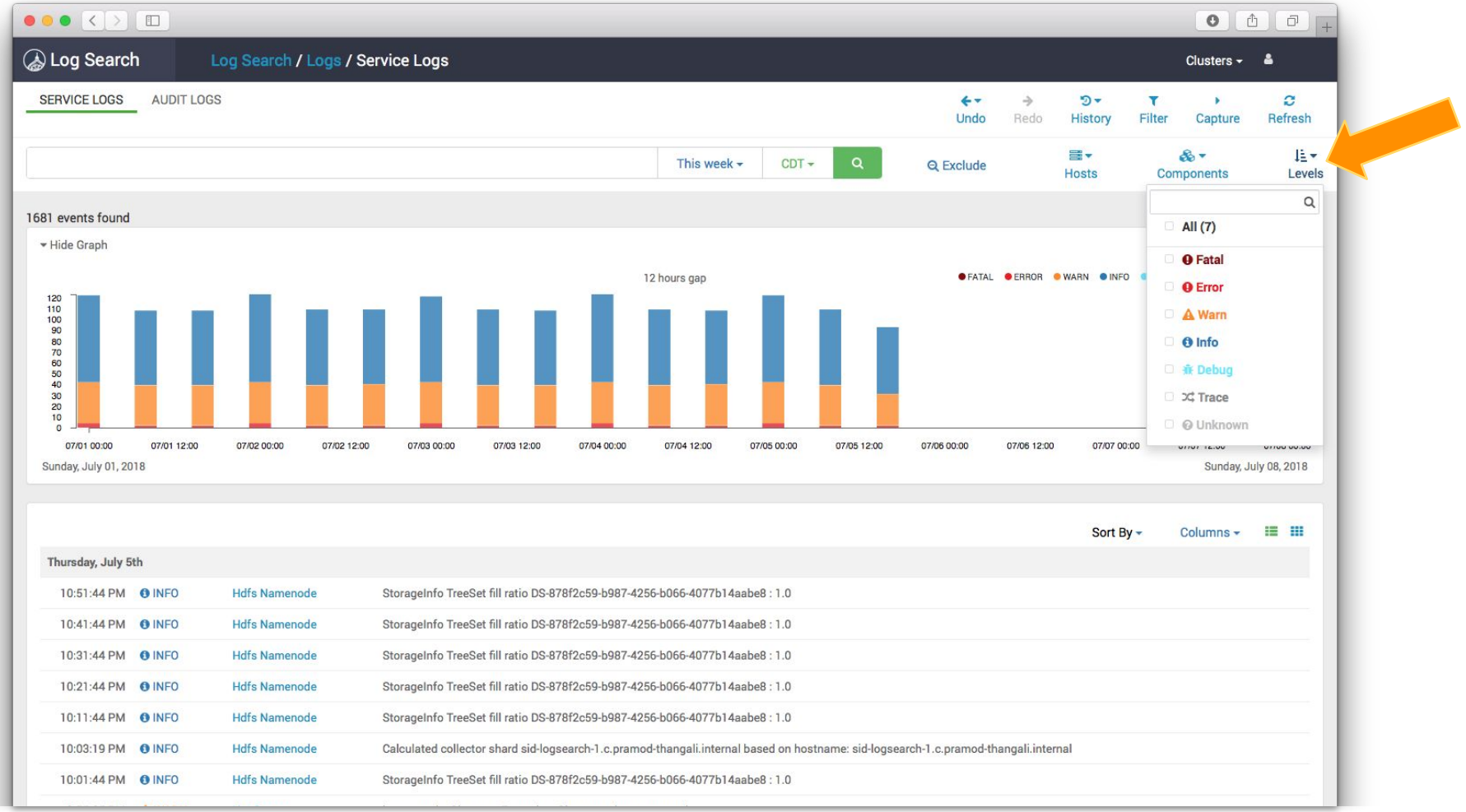
AMBARI - METRICS



AMBARI - HOST LEVEL DETAILS

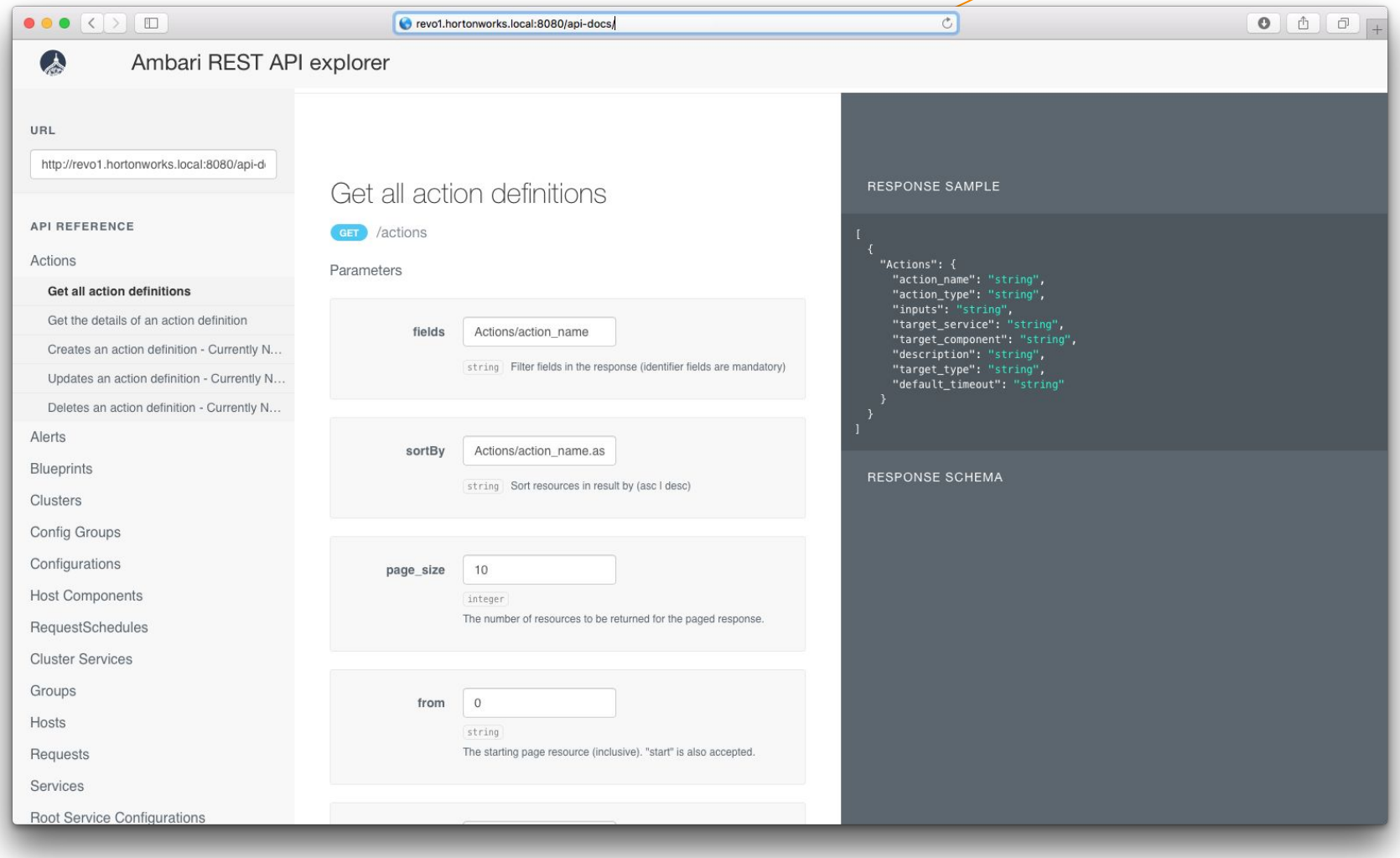


AMBARI - LogSearch UI



AMBARI - REST API

http://…/**api-docs**



EXTENDING NIFI

EXTENSION / INTEGRATION POINTS

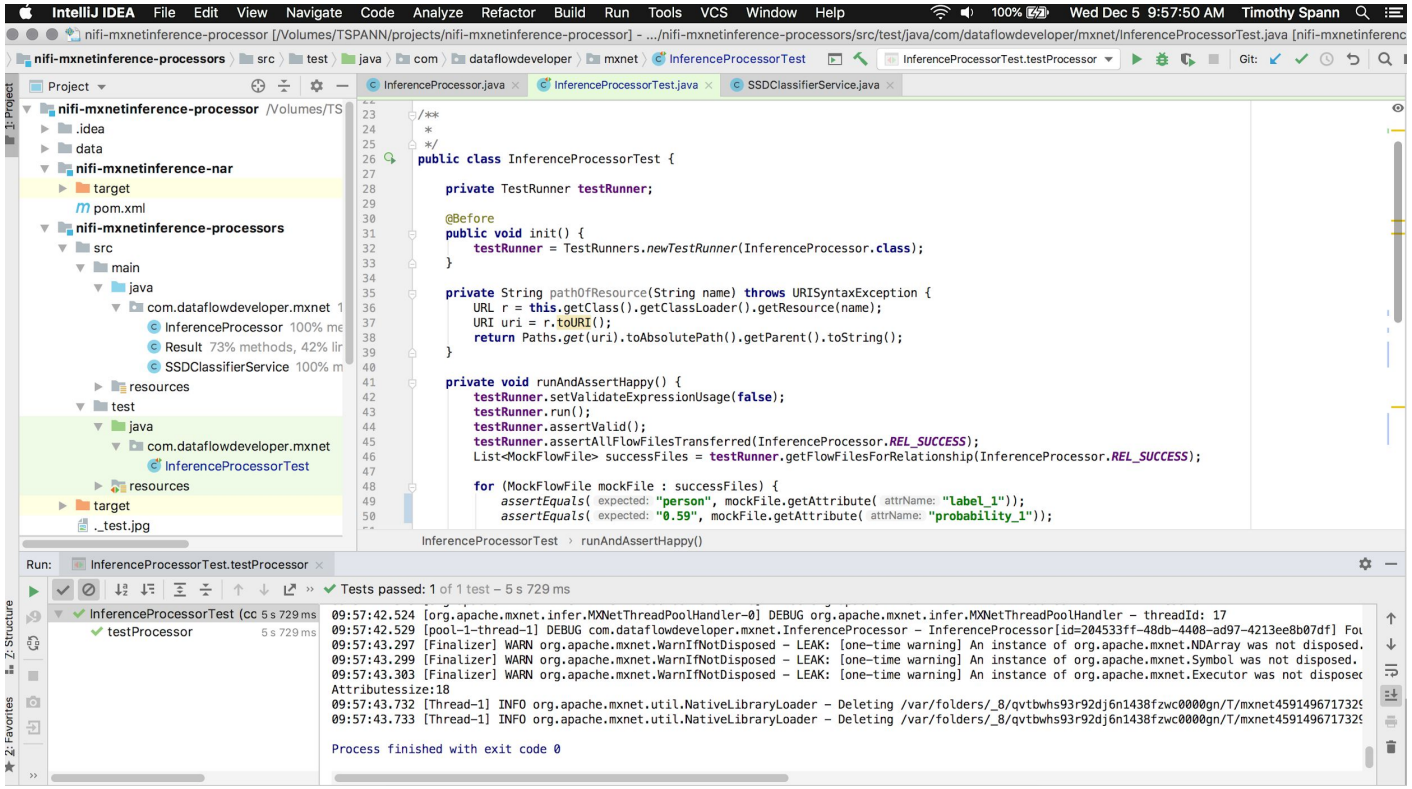
NiFi Term	Description
Flow File Processor	Push/Pull behavior. Custom UI
Reporting Task	Used to push data from NiFi to some external service (metrics, provenance, etc..)
Controller Service	Used to enable reusable components / shared services throughout the flow
REST API	Allows clients to connect to pull information, change behavior, etc..

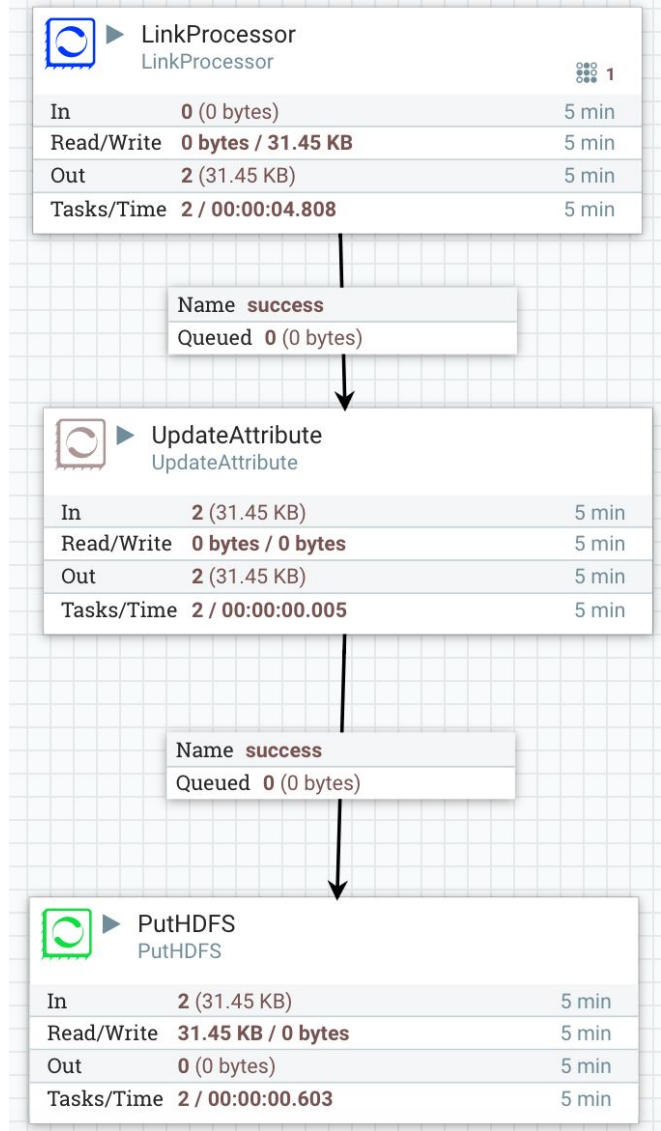
CREATING A CUSTOM NIFI PROCESSOR

What we'll cover:

- How to use Java/Maven to generate our processor framework
- Using Maven to build and compile a custom processor
- How to update (and where to find) the:
  - Relationship paths
  - Processor logic
  - Properties
  - Scheduling Tasks

CUSTOM NIFI PROCESSOR





https://community.hortonworks.com/articles/73355/adding-a-custom-processor-to-nifi-linkprocessor.html
https://www.datainmotion.dev/2019/03/getting-started-with-custom-processor.html
https://www.datainmotion.dev/2019/03/apache-nifi-processor-for-apache-mxnet.html

STEPS FOR CREATING A CUSTOM NIFI PROCESSOR

1. Use Maven to generate the archetype, run: mvn archetype:generate -Dfilter=org.apache:nifi

Select #1: **remote -> org.apache.nifi:nifi-processor-bundle-archetype (-)**
Select version #33**: 1.9.2**
Define value for property 'groupId': com.github.nifi_yourname
Define value for property 'artifactId': yourname_nifi_processor
Define value for property 'version': 1.0-SNAPSHOT: 1.0
Define value for property 'artifactBaseName': demo
Define value for property 'package': com.github.nifi_yourname.processors.demo: :
com.github.nifi_yourname.processors.demo

STEPS FOR CREATING A CUSTOM NIFI PROCESSOR (CONTINUED…)

2. Customize the sample “MyProcessor” project:
   - Located at: ./nifi-demo-processors/src/main/java/com/github/nifi_yourname/processors/demo/MyProcessor.java

3. Add a line at the end of “OnTrigger” to transfer the flowfile into MY_RELATIONSHIP:
   - session.transfer(flowFile,MY_RELATIONSHIP);

4. Create the dependency tree based on the project configuration pom.xml and download/compile all of the required components under the user folders.
   - mvn install

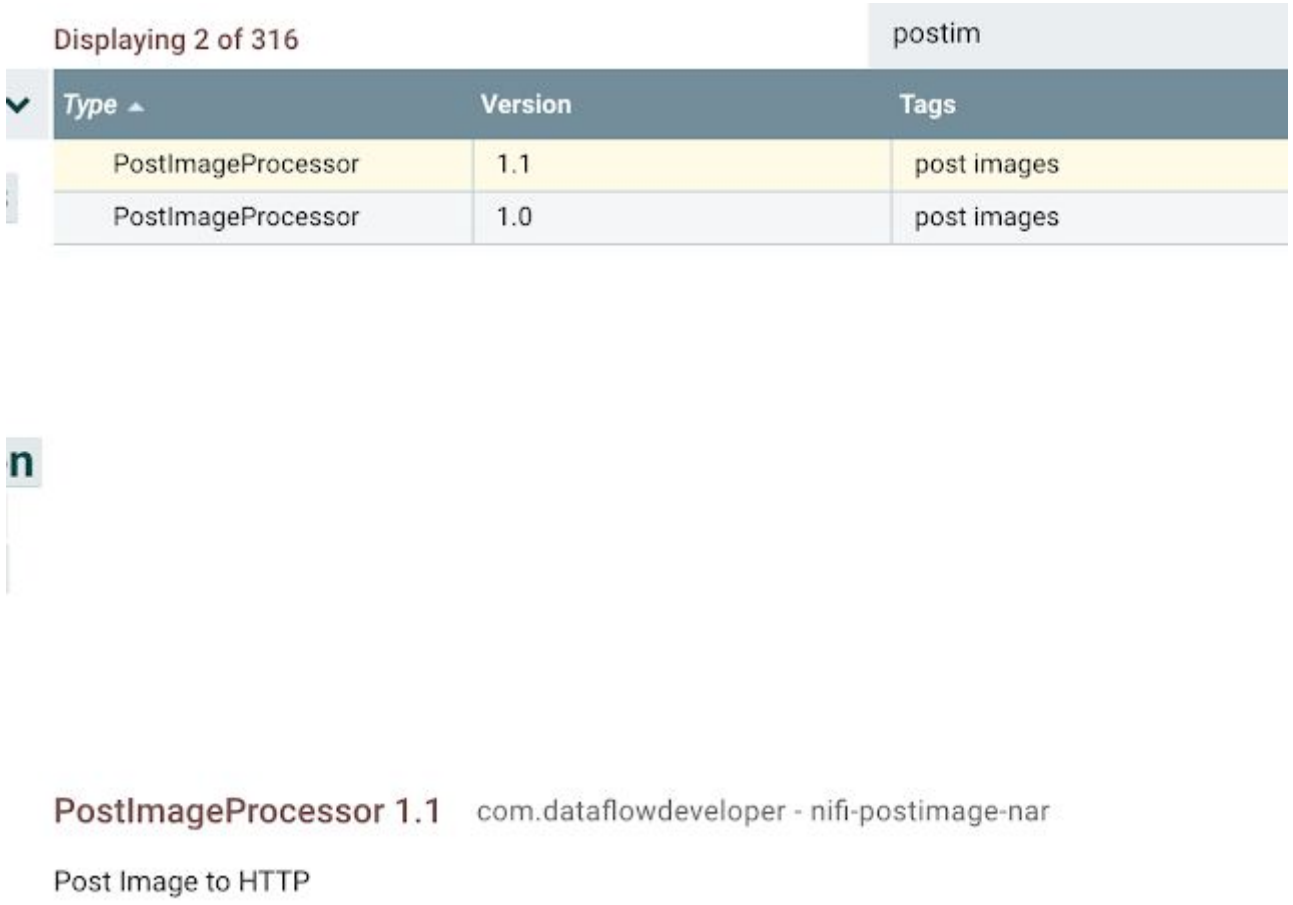
STEPS FOR CREATING A CUSTOM NIFI PROCESSOR (CONTINUED…)

5. Copy the .nar file to your $NIFI_HOME/lib location:
   - Nar File located at: ./nifi-demo-nar/target/nifi-demo-nar-1.0.nar

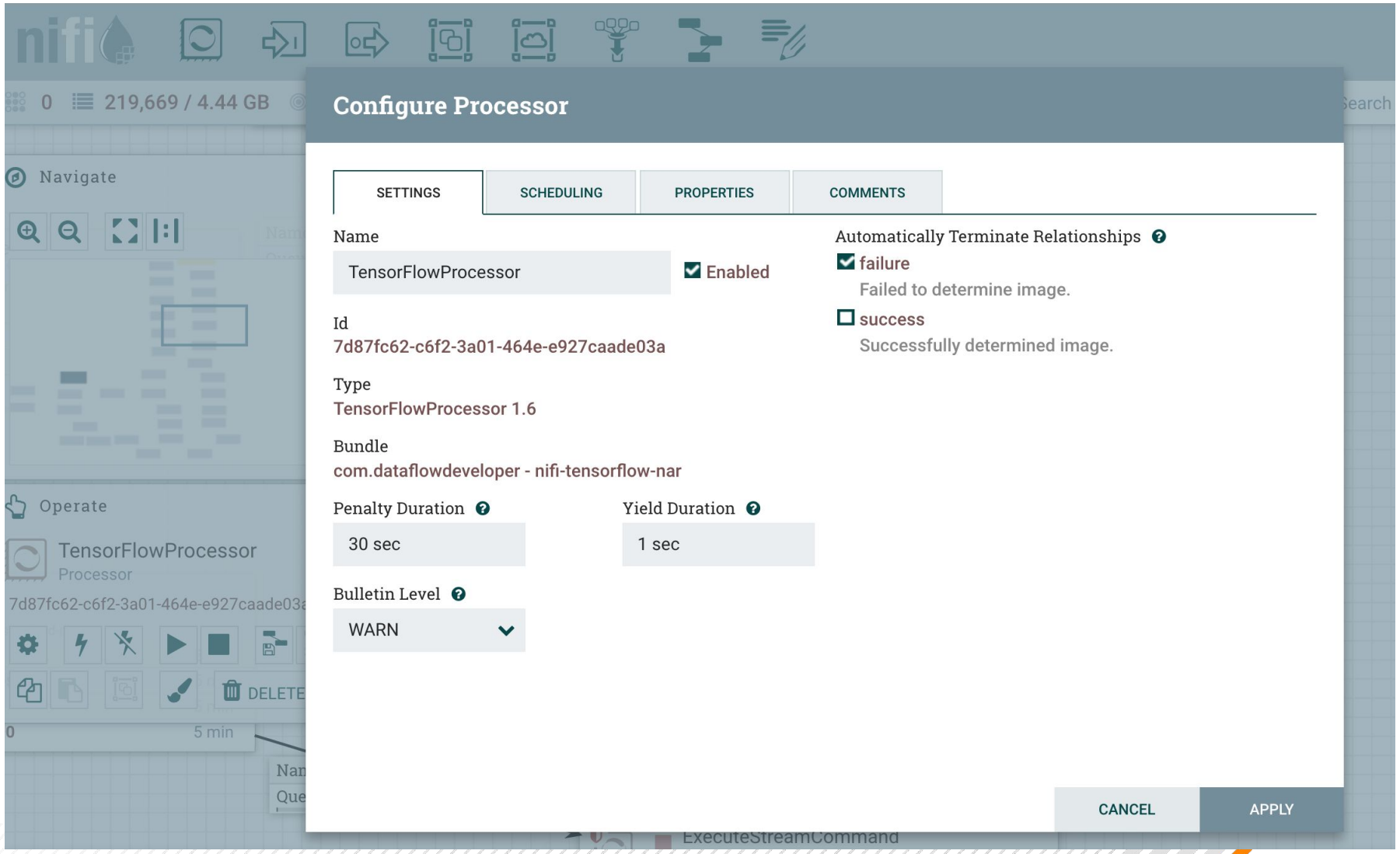
6. (Re)Start NiFi:
   - Mac/Linux: ./bin/nifi.sh run
   - Windows: \bin\run-nifi.bat

7. Open your browser and go navigate to: http://localhost:8080/nifi

ADD CUSTOM PROCESSOR TO THE CANVAS



CUSTOM NIFI PROCESSOR (CONFIGURE PROCESSOR SETTINGS)



CUSTOM NIFI PROCESSOR (CONFIGURE PROCESSOR PROPERTIES)

Property	Value
url	https://slack.com/api/files.upload?token=${slacktoken}&channels=images&filename=${filename}&filetype...
fieldname	file
imagename	${filename}
imagetype	image/jpeg

Configure Processor

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

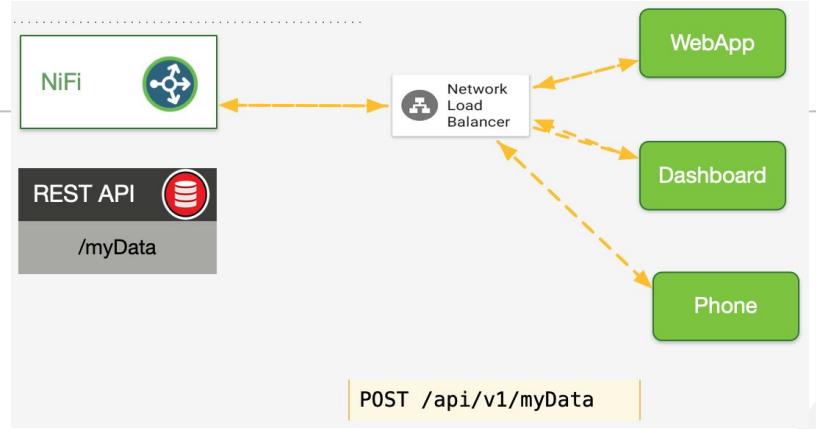
Required field

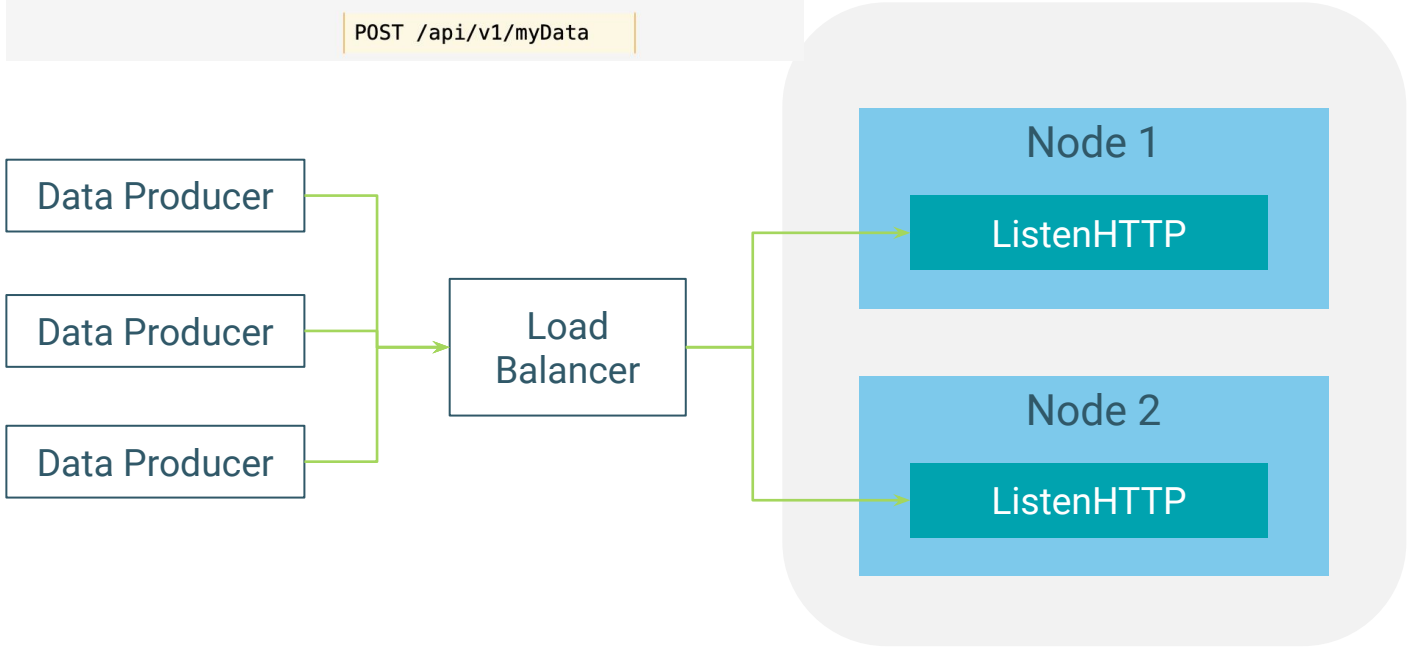
Property	Value
mode...	/Volumes/TSPANN/projects/nifi-mxnetinference-processor/data/models/resnet50_ssd/resnet50_ssd_model

DATA DISTRIBUTION PATTERNS WITH APACHE NIFI

PUSHING TO LISTENERS

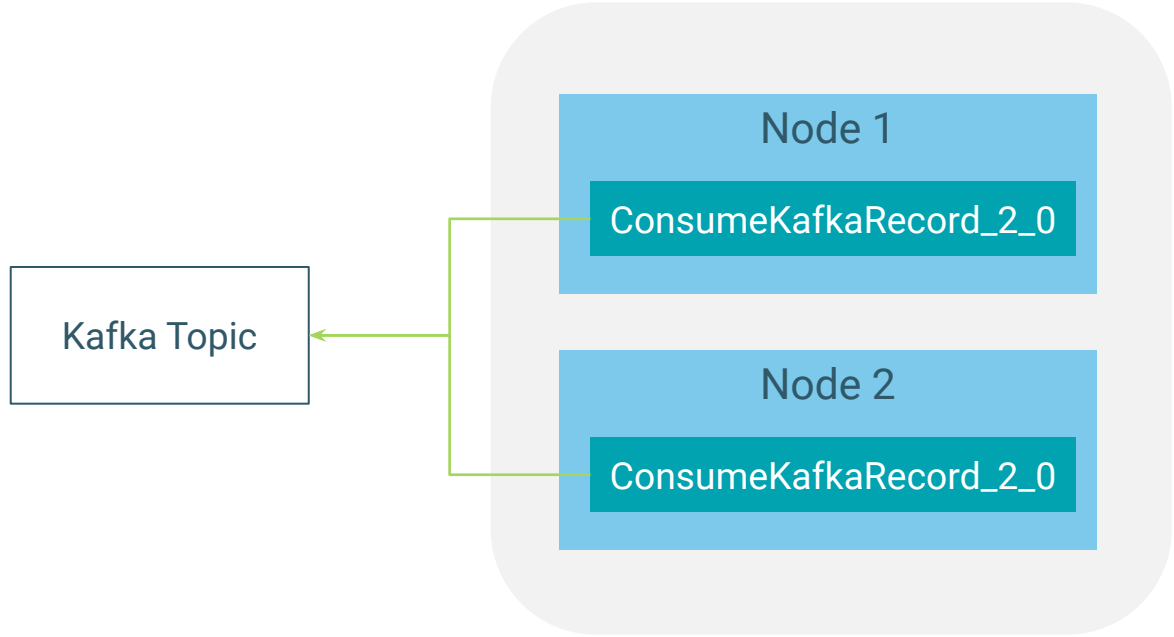
- Processors listening for incoming data on each node in the cluster
- Load balancer sitting in front of the cluster pointing at listeners
- Data producers push data to load balancer which distributes data across the cluster
- Same approach works for ListenSyslog, ListenUDP, and HandleHttpRequest





PULLING ON ALL NODES

- Works when data source ensures each request gets a unique piece of data
- Kafka sees each ConsumeKafkaRecord_ processor as the same client
- Each ConsumeKafkaRecord_ processor gets different data

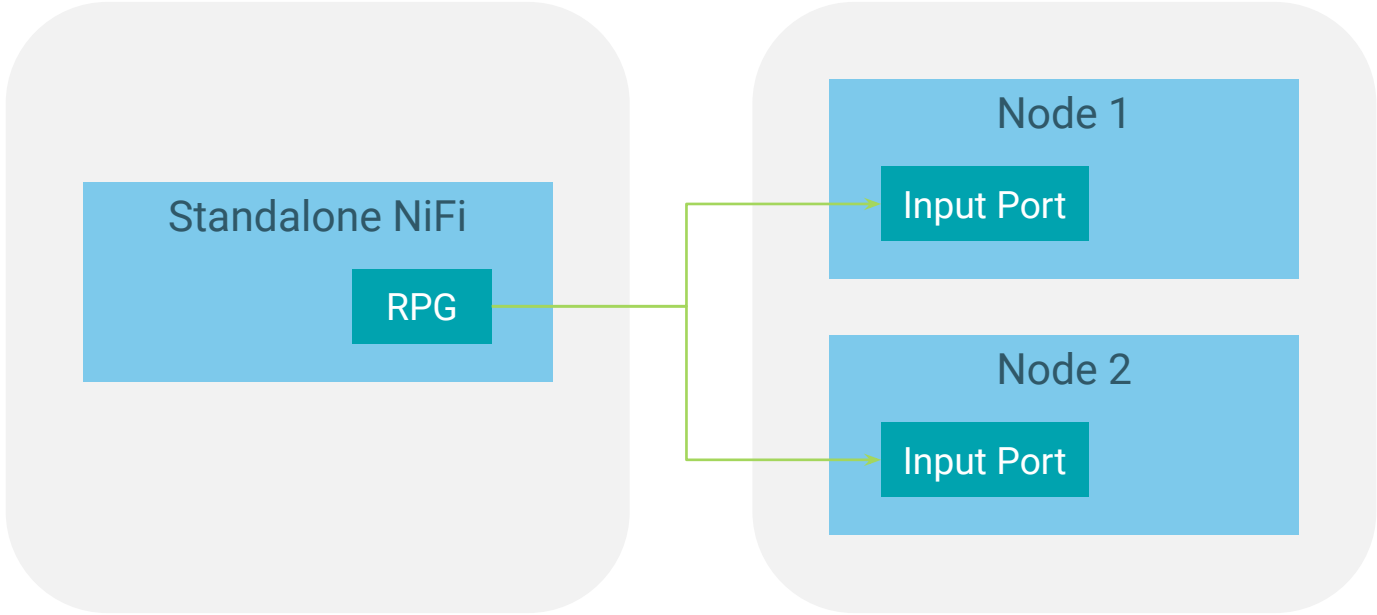


NIFI SITE-TO-SITE

- Direct communication between two NiFi instances
- Push to Input Port on receiver, or Pull from Output Port on source
- Communicate between clusters, standalone instances, or both
- Handles load balancing and reliable delivery
- Secure connections using certificates (optional)

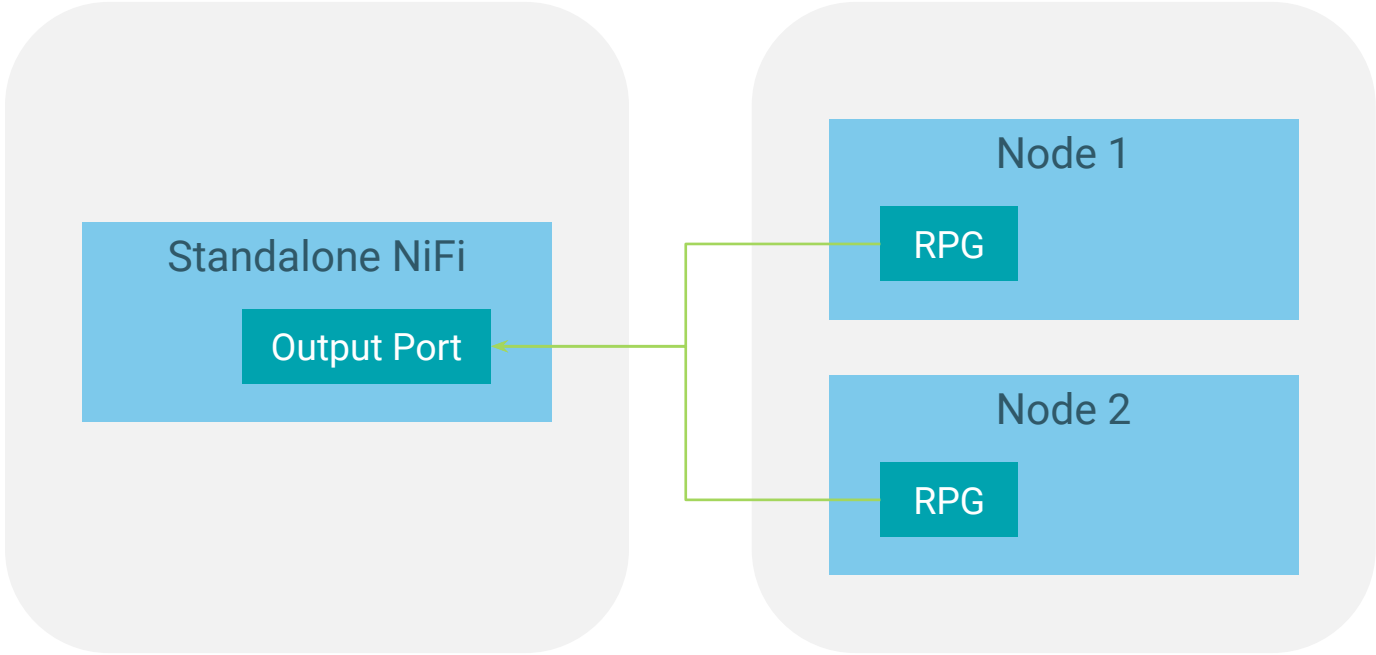
SITE-TO-SITE PUSH

- Site-To-Site makes a direct connection between NiFi instances
- Either side can be a cluster or standalone instance
- In a push scenario, the source connects a Remote Process Group to an Input Port on the destination
- If pushing to a cluster, Site-To-Site takes care of load balancing across the nodes in the cluster



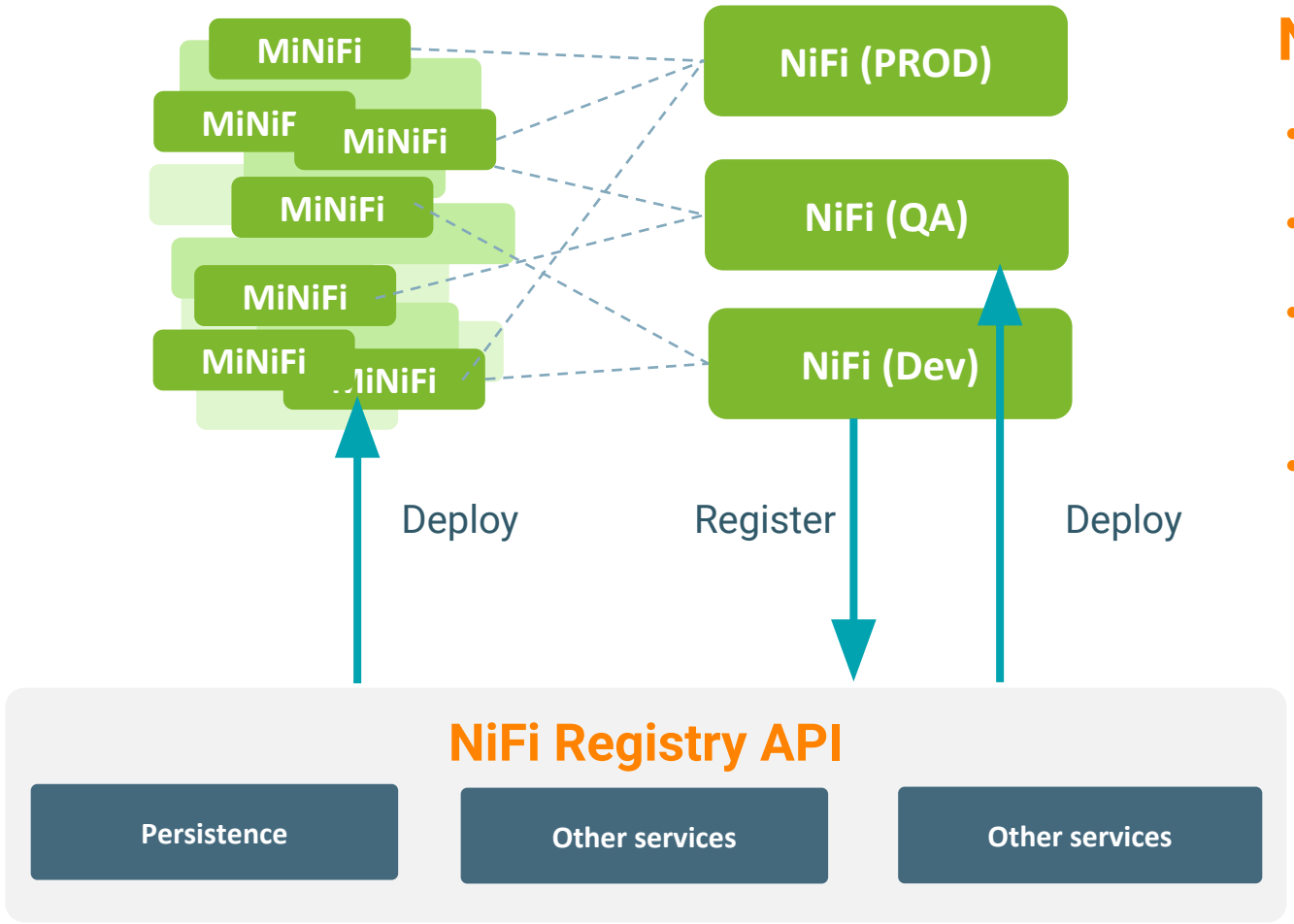
SITE-TO-SITE PULL

- In a pull scenario, the destination connects a Remote Process Group to an Output Port on the source
- Each node will pull different data from the source
- If the source was a cluster, each node would pull from each node in the source cluster



SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)

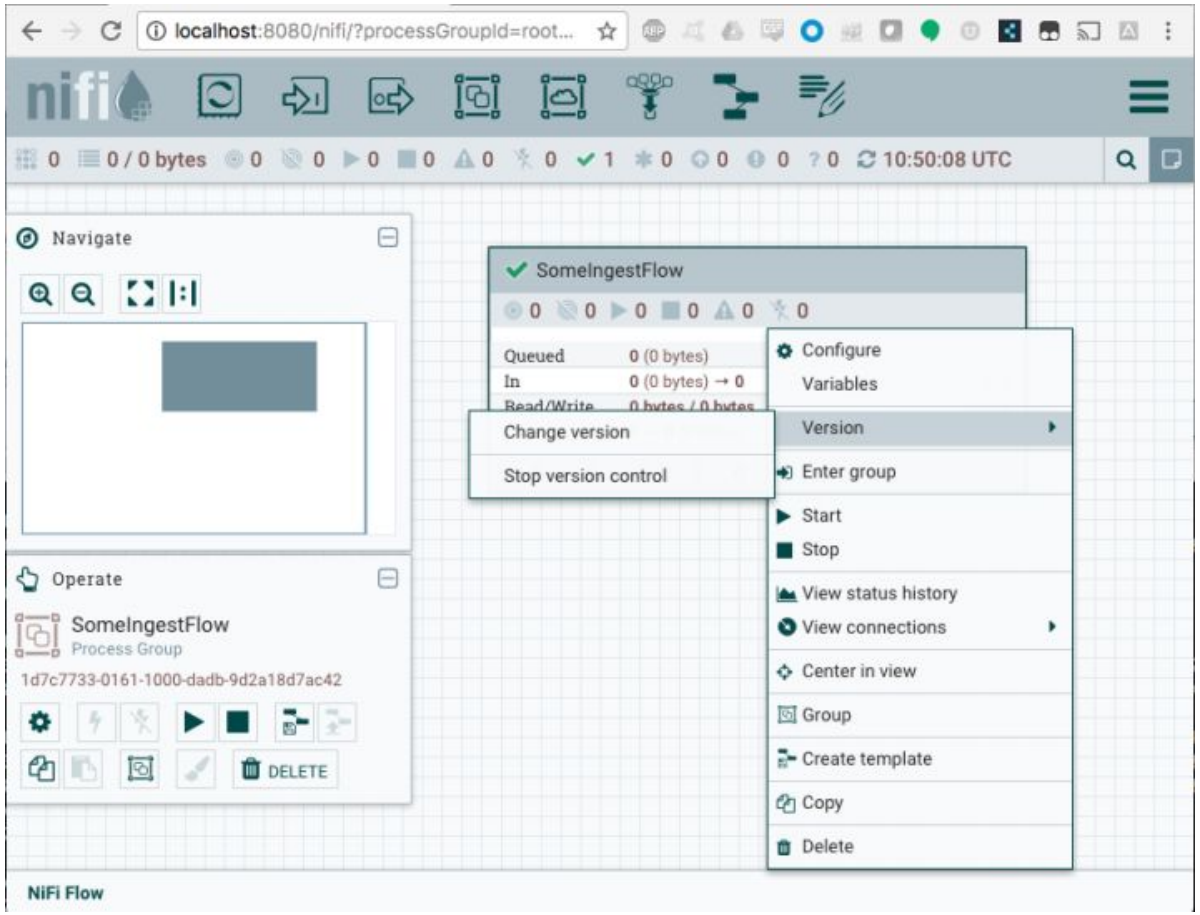
INCREASED DEVELOPER PRODUCTIVITY: APACHE NIFI REGISTRY



NiFi Registry

- Repository of versioned flows
- Portability
- Support multiple registries and interactions between them
- Design and deploy mechanism

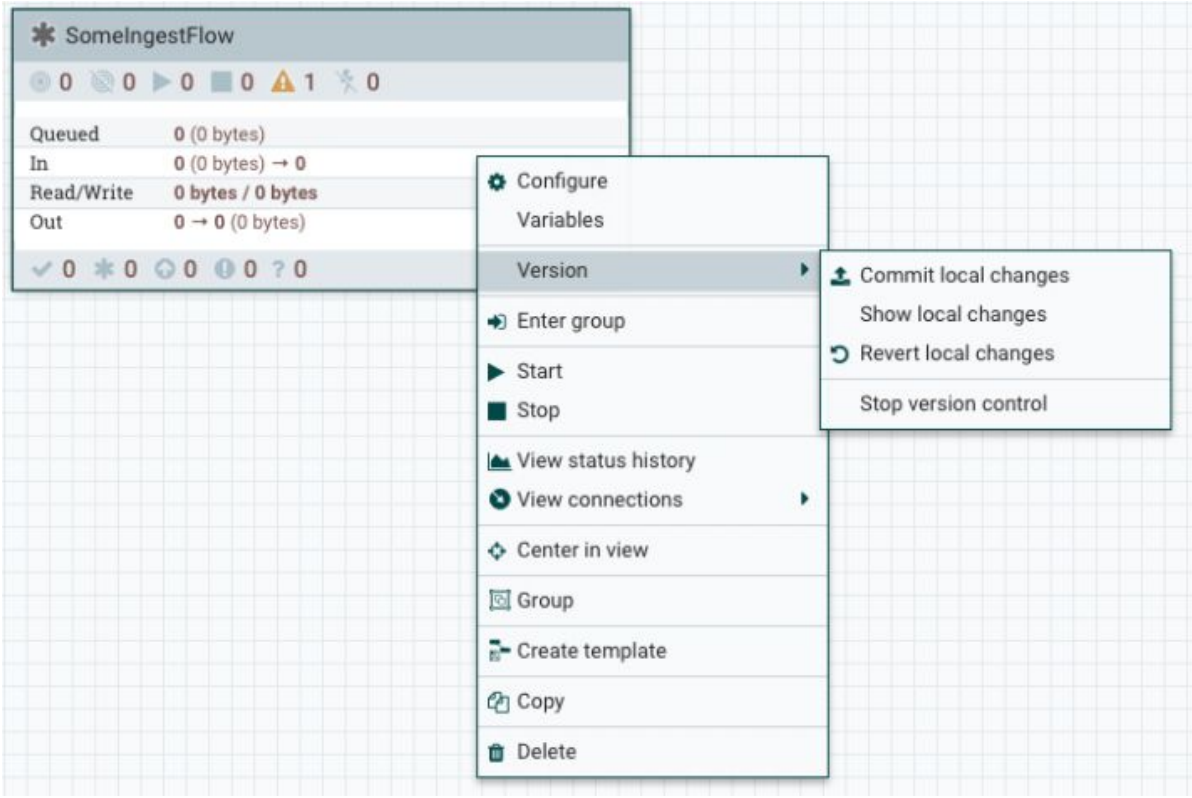
DESIGN & DEPLOY COMPLEMENTING COMMAND & CONTROL



- SDLC Dev: Place Process Groups under Version Control
- Make changes and commit to new version
- Roll Versions back or forward

https://community.hortonworks.com/articles/224554/building-a-custom-apache-nifi-operations-dashboard.html

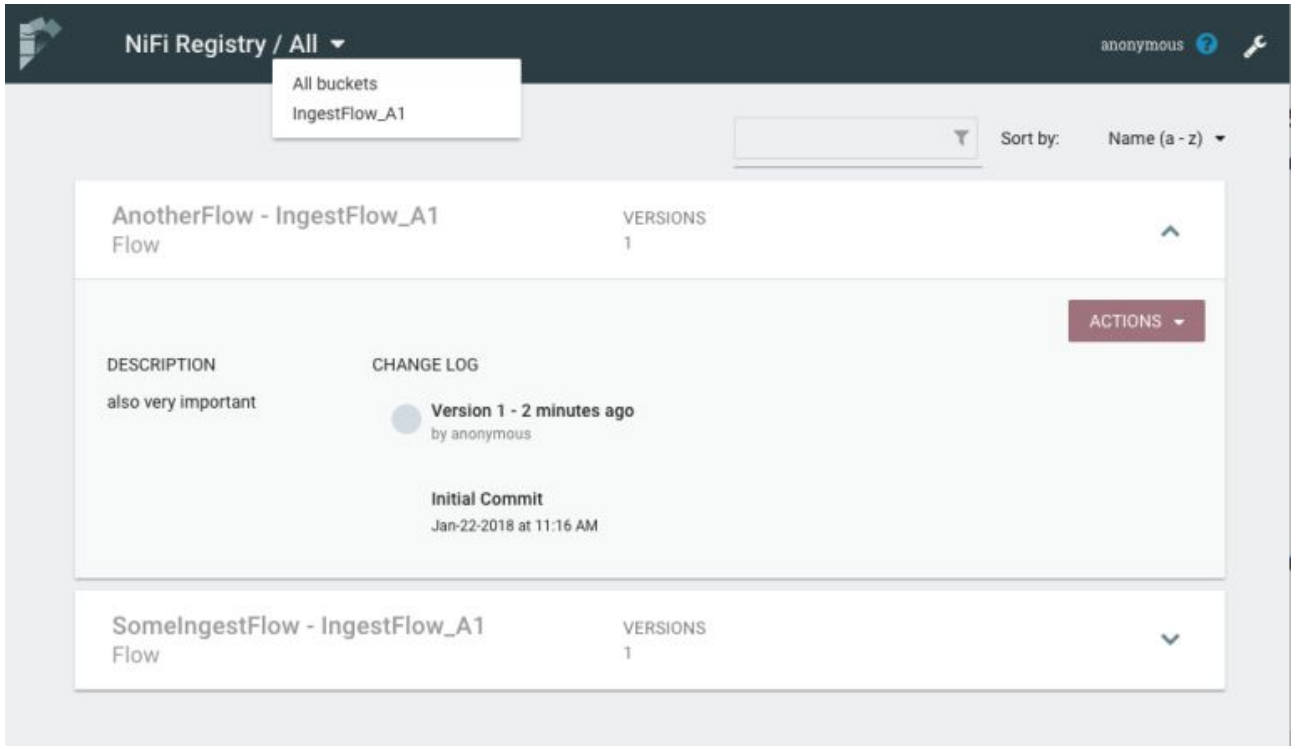
DESIGN & DEPLOY COMPLEMENTING COMMAND & CONTROL



- Get Notifications of local changes or new versions available in Repository
- Revert or Commit local changes via the GUI or Rest-API
- Use Rest-API to integrate with Jenkins, etc.

https://medium.com/@abdelkrim.hadjidj/fdlc-towards-flow-development-life-cycle-with-nifi-registries-82e1ee866fab

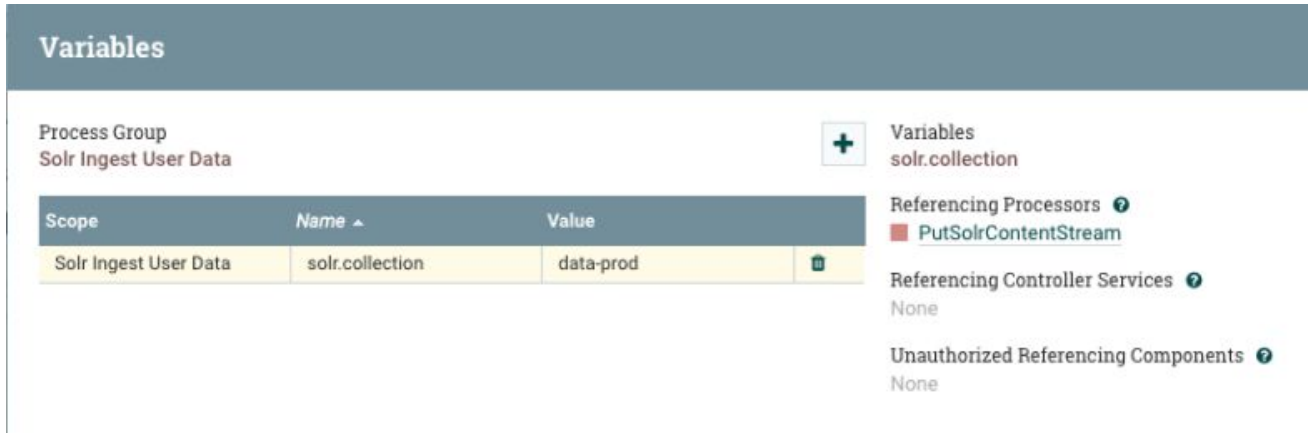
INTEGRATED FLOW REGISTRY SERVICE



- Integrated Flow Registry Service
- Sharable between NiFi environments for Dev/UAT/Prod promotion
- API or GUI driven
- Can be integrated with Enterprise Version Control e.g. GitLab
- 'Buckets' of Flows for security and access control

https://community.hortonworks.com/articles/207858/more-devops-for-hdf-apache-nifi-and-friends.html

INTEGRATED VARIABLE REGISTRY SERVICE



- Integrated Variable Registry
- Sets of key:value pairs available on every Process Group
- Referenced with NiFi Expression Language
- Dynamically changeable at runtime
- Use within Versioned Flows to set Environment Variables
- GUI or API driven

EXTENSION REGISTRY - FUTURE

- Every component (processor, PG, CS, etc.) of NIFI, the specific version of it, and the code behind it, can be tagged to templates
- When you instantiate a template, it'll be able to bring in the specific components that have been thoroughly tested, potentially in a different environment, when the template was published.
- The intention is, given a template, you'll be able to deploy exactly that template to another environment, and that environment will sourced exactly those extensions.

CREATE A NEW, LIVE DATAFLOW IN 7 MINUTES

OUR FIRST 2 LABS FOR TODAY

Lab 1

Consuming the Meetup RSVP data

Lab 2

Accessing cluster

MINIFI

APACHE MINIFI

“Let me get the key parts of NiFi close to where data begins and provide bidirectional data transfer"

- NiFi lives in the data center. Give it an enterprise server or a cluster of them.
- MiNiFi lives as close to where data is born and is a guest on that device or system

APACHE ~~NIFI~~ MINIFI

Key features

- Guaranteed delivery
- Data buffering
  - Backpressure
  - Pressure release
- Prioritized queuing
- Flow specific QoS
  - Latency vs. throughput
  - Loss tolerance
- Data provenance
- Recovery/recording a rolling log of fine-grained history
- Designed for extension
- Design and Deploy
- Warm re-deploys

OUTSIDE THE COMFORTS OF THE DATA CENTER

Realities of computing

- Limited computing capability
- Limited power/network
- Restricted software library/platform availability
- No UI
- Physically inaccessible
- Not frequently updated
- Competing standards/protocols
- Scalability
- Privacy & Security

MINIFI: PRECEDENT FROM NIFI

A quick look at NiFi site to site

- Provides the semantics between two NiFi components across network boundaries
  - A custom protocol for inter-NiFi communication
  - Secure, Extensible, Load Balanced & Scalable Delivery to Cluster

- Extracted out to a client library which powers integration into popular frameworks like Apache Spark, Apache Storm, Apache Flink, and Apache Apex

- Attributes and the FlowFile format maintained

https://nifi.apache.org/docs/nifi-docs/html/user-guide.html#site-to-site

MINIFI: PRECEDENT FROM NIFI

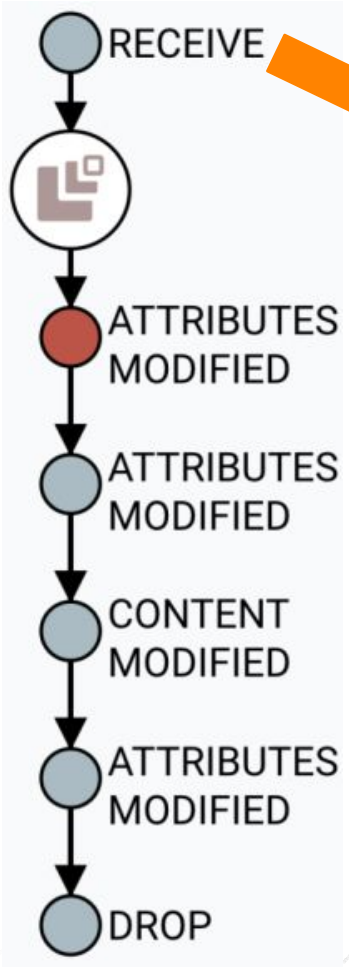
A deeper dive into provenance

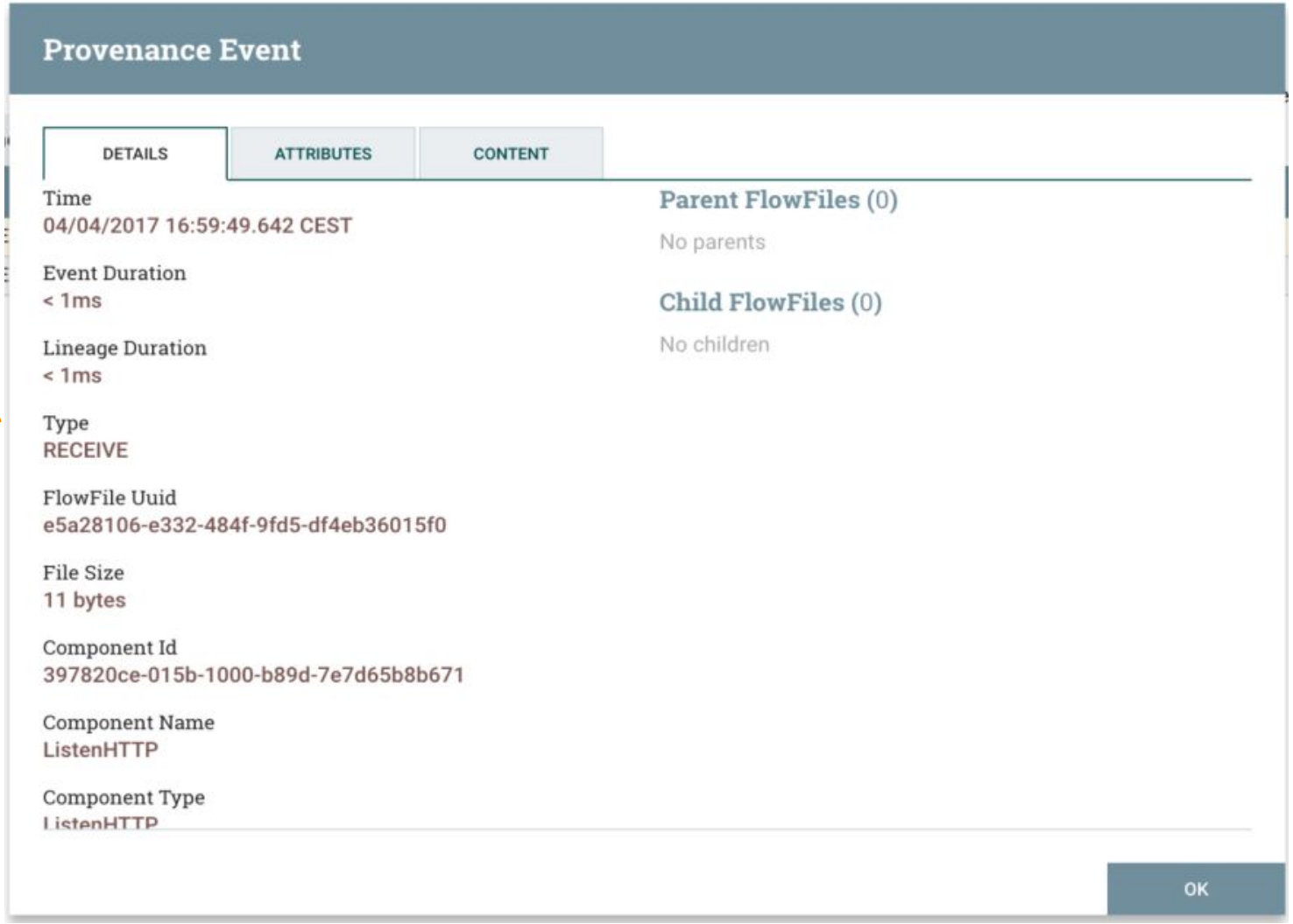
- Fine-grained, event level access of interactions with FlowFiles
  - CREATE, RECEIVE, FETCH, SEND, DOWNLOAD, DROP, EXPIRE, FORK, JOIN …
- Captures the associated attributes/metadata at the time of the event
- A map of a FlowFile's journey and how they relate to other FlowFiles in a system
  - MiNiFi enables us to get more and further illuminate the map of data processing

http://nifi.apache.org/docs/nifi-docs/html/user-guide.html#data-provenance

MINIFI: PRECEDENT FROM NIFI

RECEIVE event





APACHE NIFI - MINIFI

Departures from NiFi in getting the right fit

- The feedback loop is longer and not guaranteed
  - Removal of Web Server and UI

- Declarative configuration
  - Lends itself well to CM processes
  - Extensible interface to support varying formats
    - Currently provided in YAML

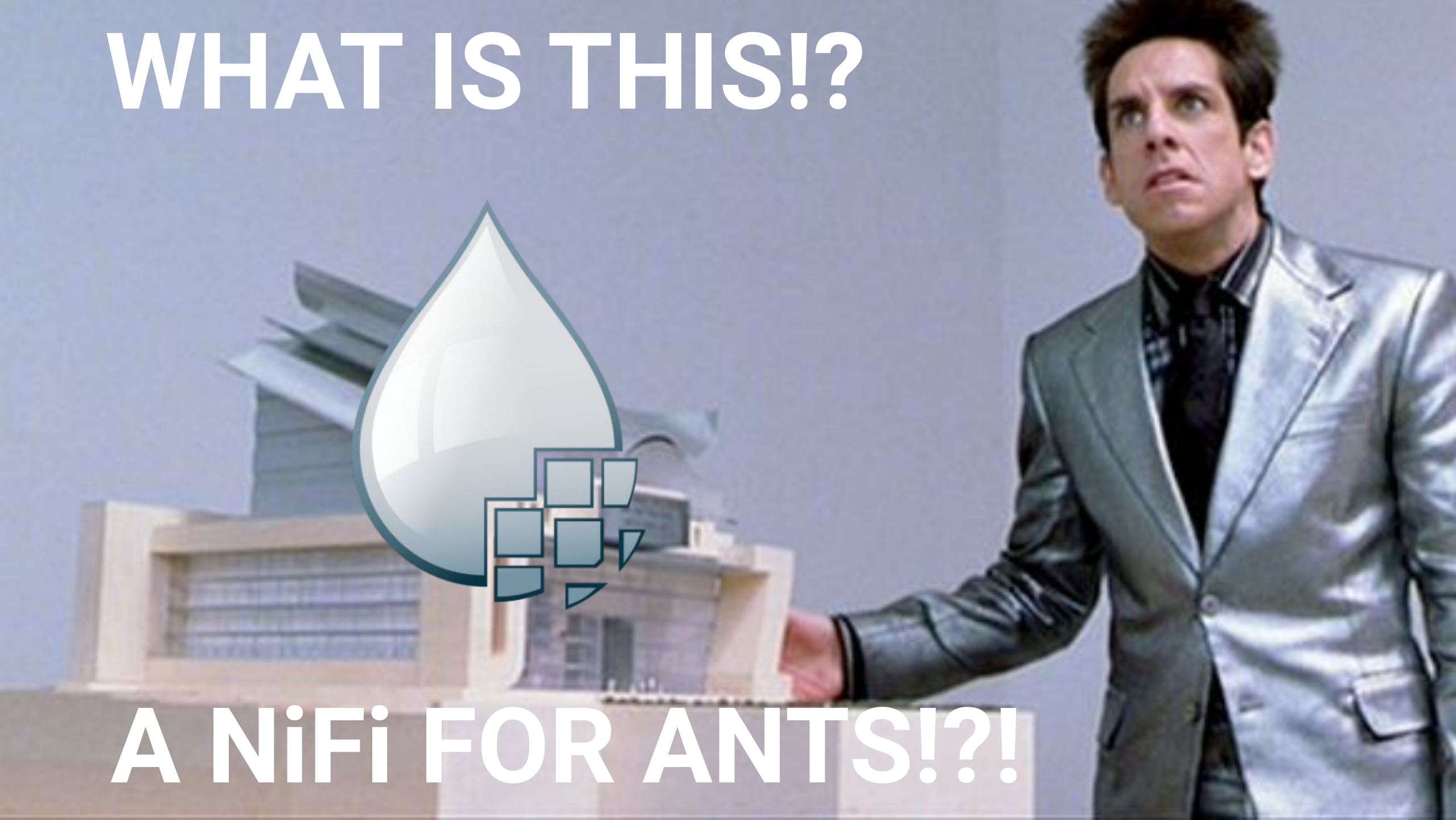
- Reduced set of bundled components

APACHE MINIFI: SCOPING

Provide all the key principles of NiFi in varying, smaller footprints

- ***Go small***: Java – *Write once, run anywhere**
  - Feature parity and reuse of core NiFi libraries
- ***Go smaller***: C++ – *Write once**, run anywhere*
- ***Go smallest***: *Write n-many times, run anywhere*
  Language libraries to support tagging, FlowFile format, Site to Site protocol, and provenance generation without a processing framework
  - Mobile: Android & iOS
  - Language SDKs

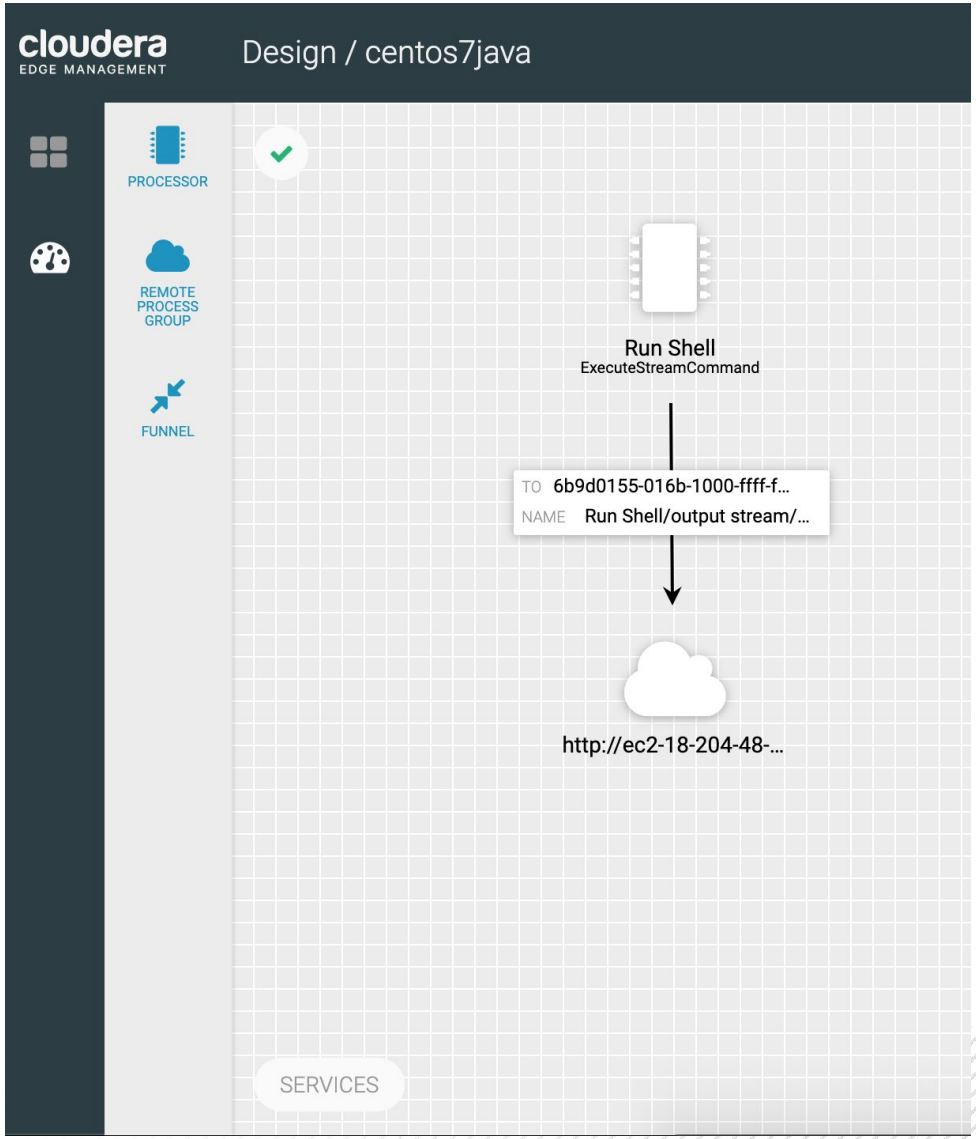
WHAT IS THIS!?

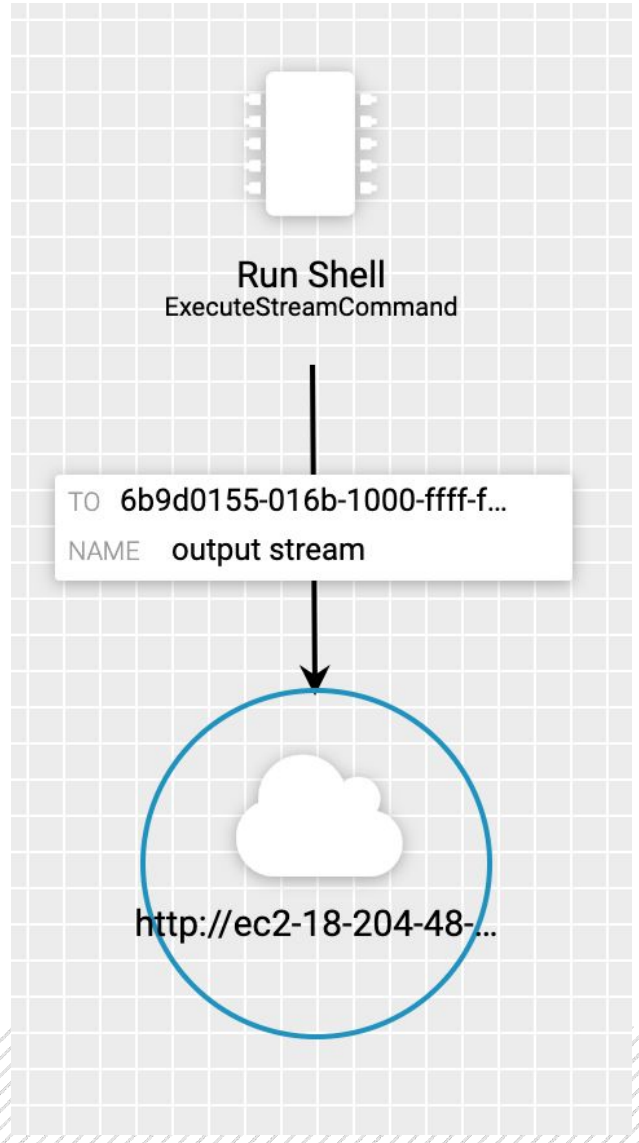


A NiFi FOR ANTS!?!

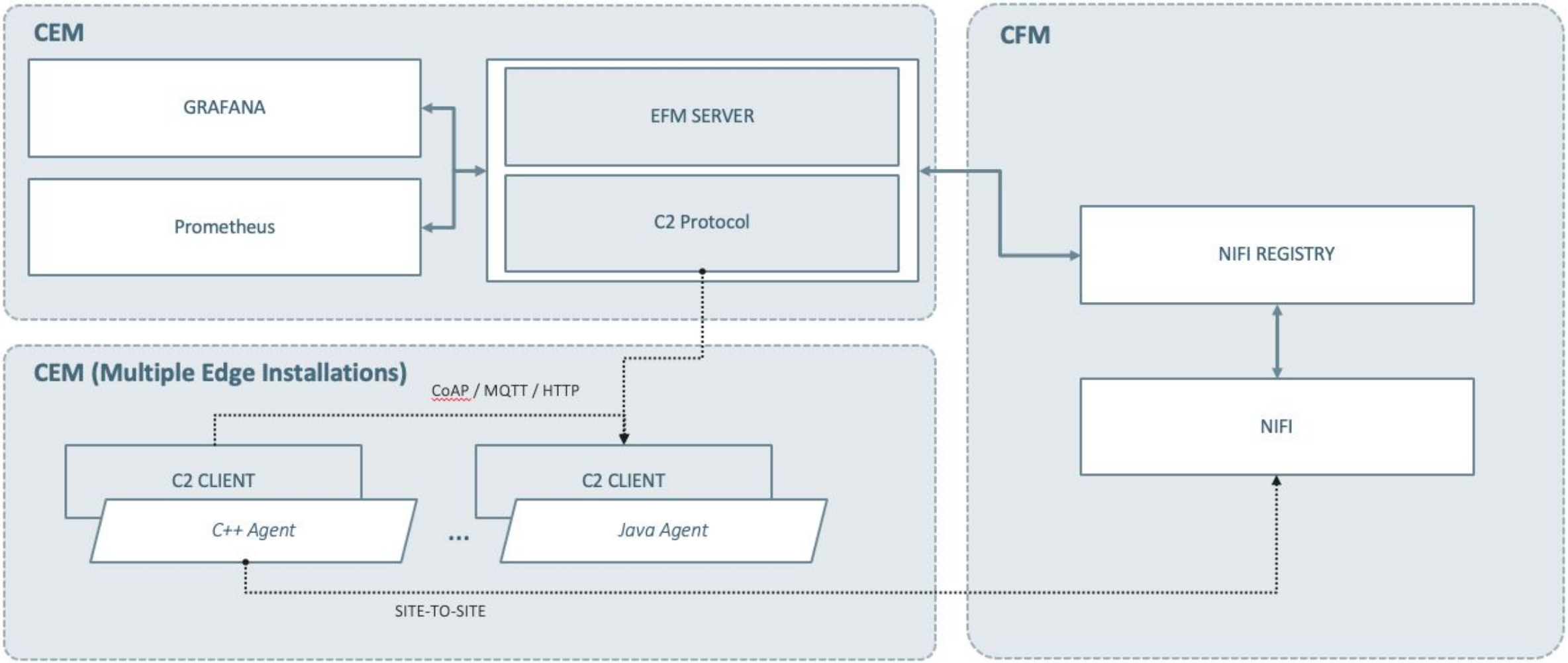
EDGE FLOW MANAGEMENT

EFM Demo



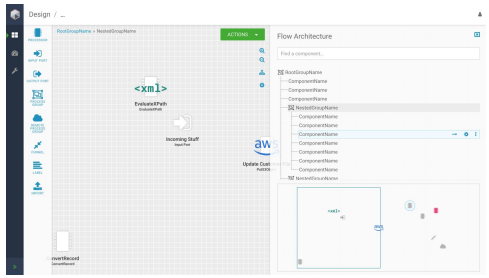


EFM

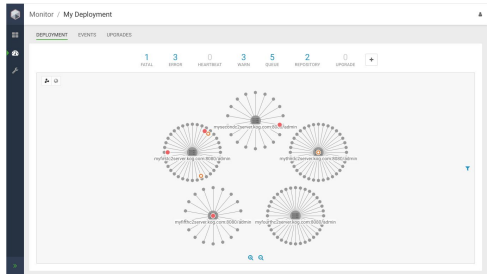


EFM ADDRESSES

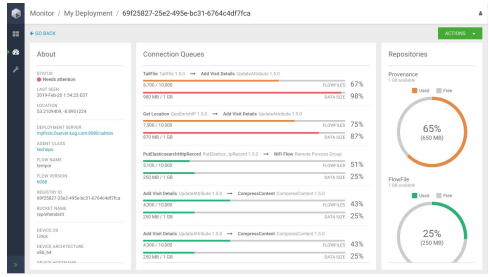
Flow Authorship



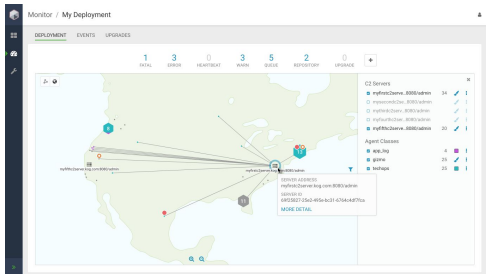
Flow Deployment



Flow Monitoring



The Disconnected Edge



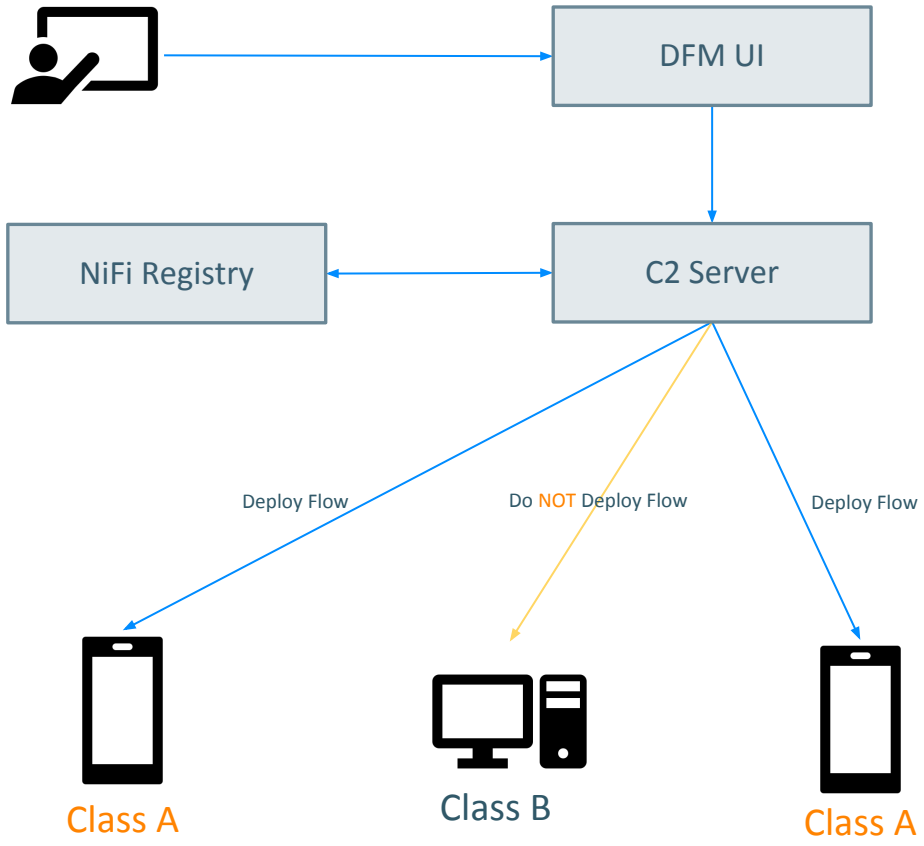
https://github.com/tspannhw/CDF-Workshop/blob/master/efm.md

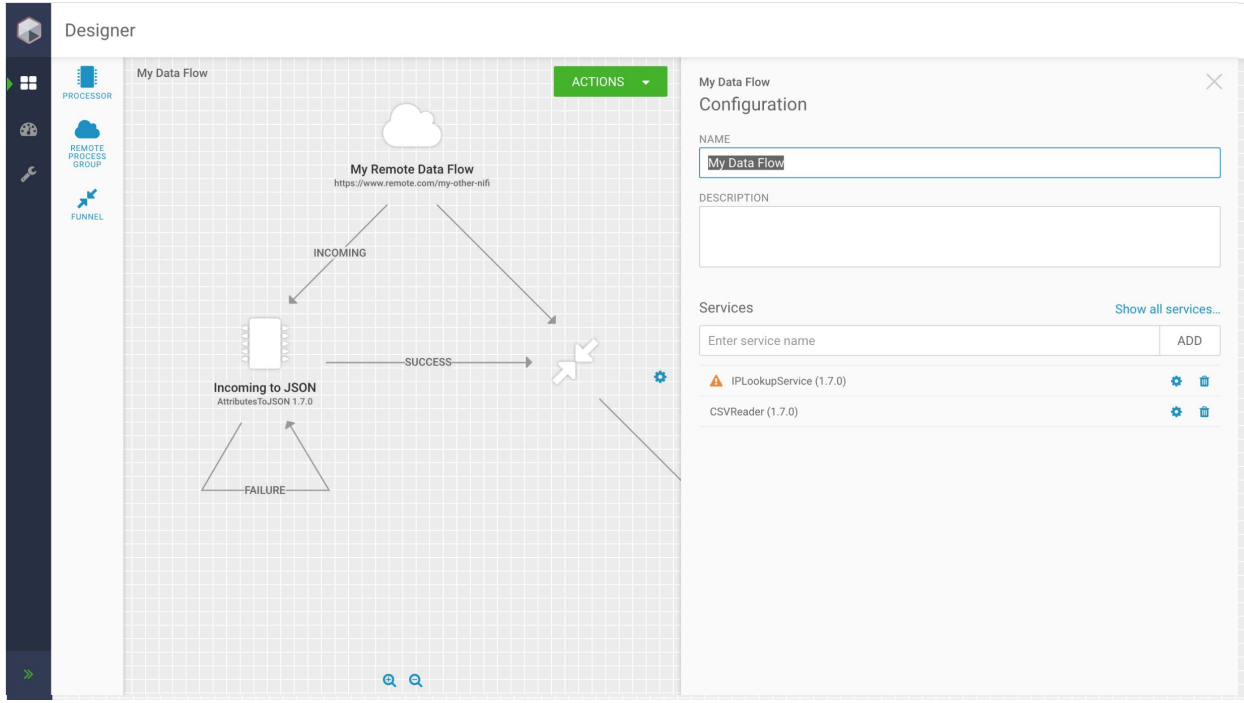
TERMINOLOGY

- Flow Authorship – Process of a flow operator designing/editing a flow in the DFM UI and then deploying those flows to targeted edge devices
- Flow Monitoring – Operational insights into flows deployed on Edge Devices.
- Device Manifest – Payload included in the edge device's heartbeat to their correlating C2 server that fully describes the processors supported by the edge device. This in turn drives the processors that are available to the flow author in the DFM UI
- Edge Device – Device running either MiNiFi Java or MiNiFi C++
- DFM – Dataflow Manager, DPS app that provides the actual UI components to users
- Class – A taxonomical grouping of Edge Devices

FLOW AUTHORSHIP

FLOW AUTHORSHIP





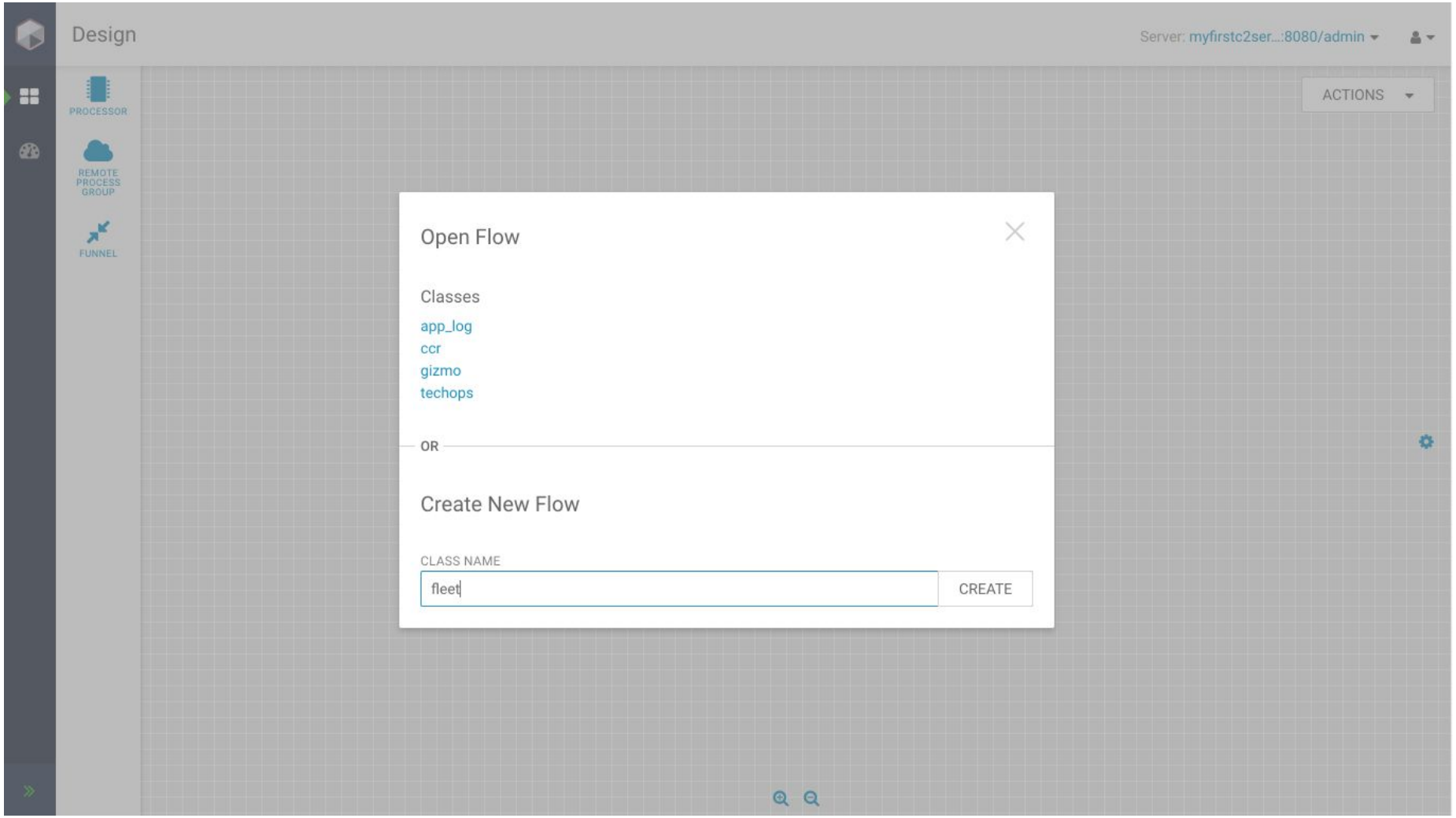
- Flow is sent to C2 server to be delivered to edge
- Flow is saved to NiFi Registry as versioned flow
- Class A flow is pushed to only class A devices

FLOW AUTHORSHIP – NEW DATA FLOW REQUIREMENTS

- A Data Flow developer has been tasked with modifying all gizmo Data Flows because all upstream environments now require JSON formatted data
- Developer must be able to load any existing flows into the DFM Designer
- Developer needs the ability to view all possible processors/services available to them for the specific class of devices
  - This is a very powerful and unique feature we offer
- Once Development is complete the developer needs the capability to commit their changes to the NiFi Registry and Git where it is staged for deployment

FLOW AUTHORSHIP – EFM

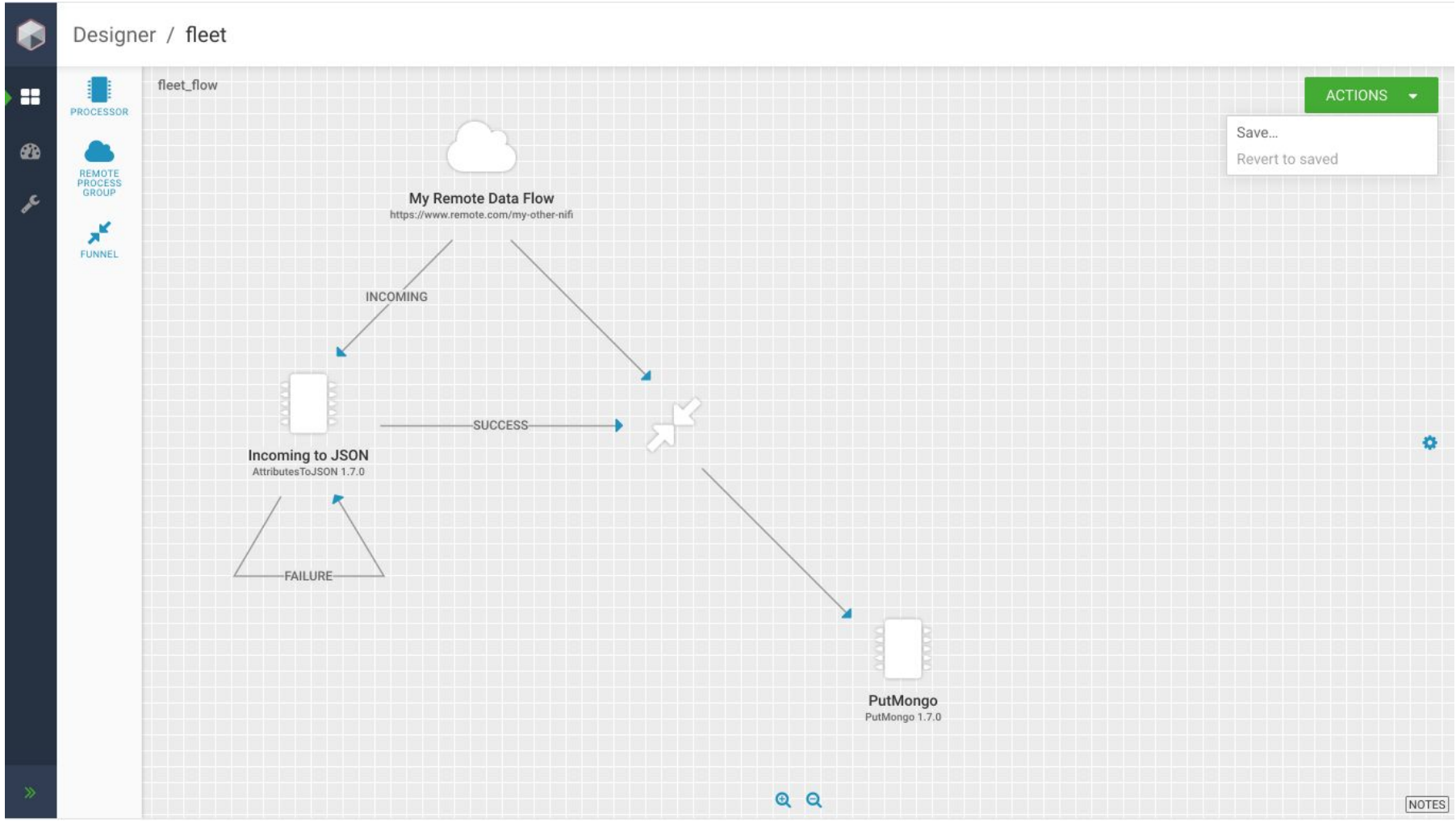
1) Developer enters EFM, navigates to “Designer” window and selects the existing gizmo Data Flow



FLOW AUTHORSHIP – EFM

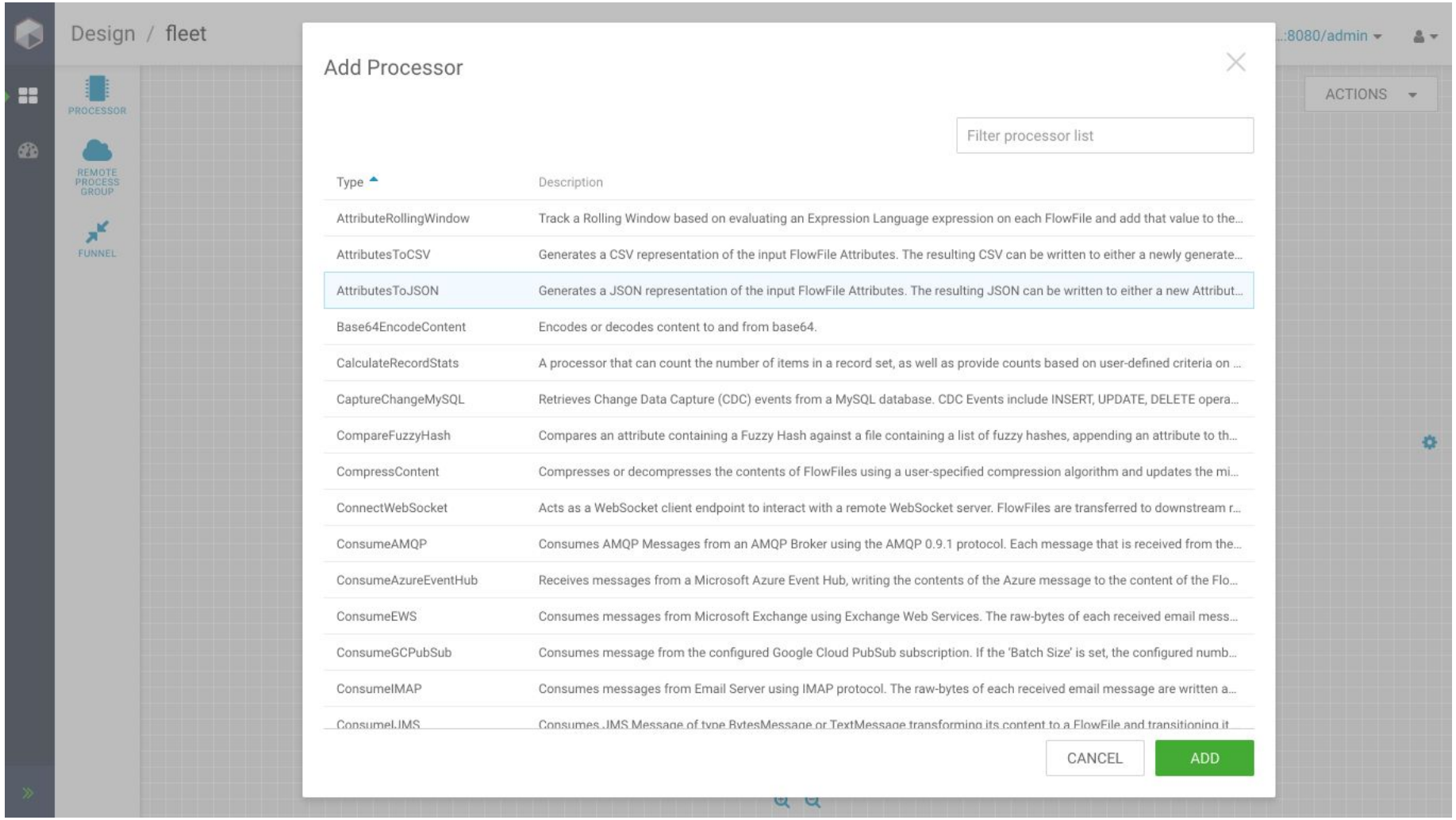
2) Existing gizmo flow loads on canvas.

Developer determines another JSON processor needs to be added



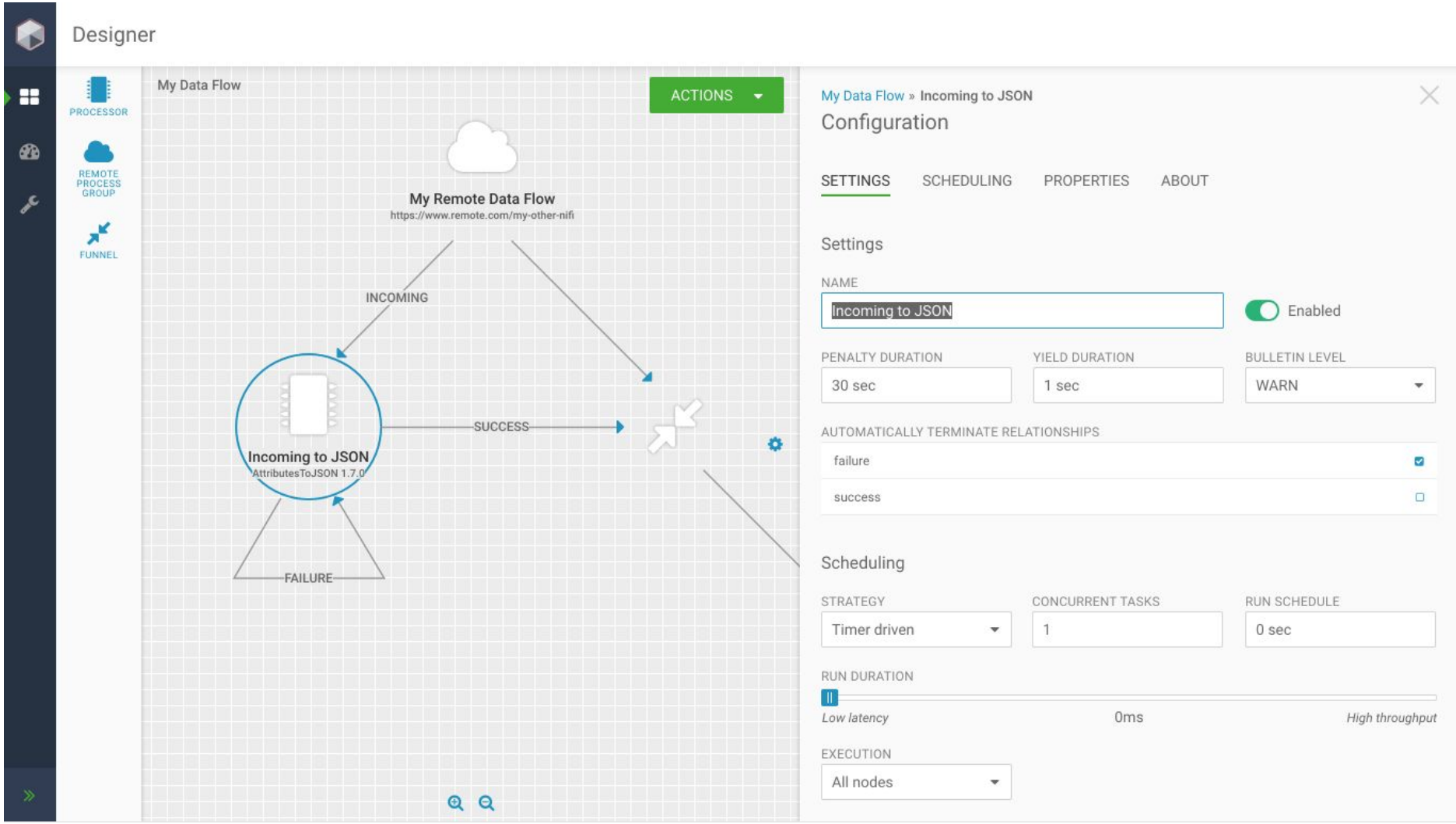
FLOW AUTHORSHIP – EFM

3) Dev clicks ”Processors” icon in designer, all available processors specifically for gizmo agents are listed, dev locates desired processor and clicks ”ADD”



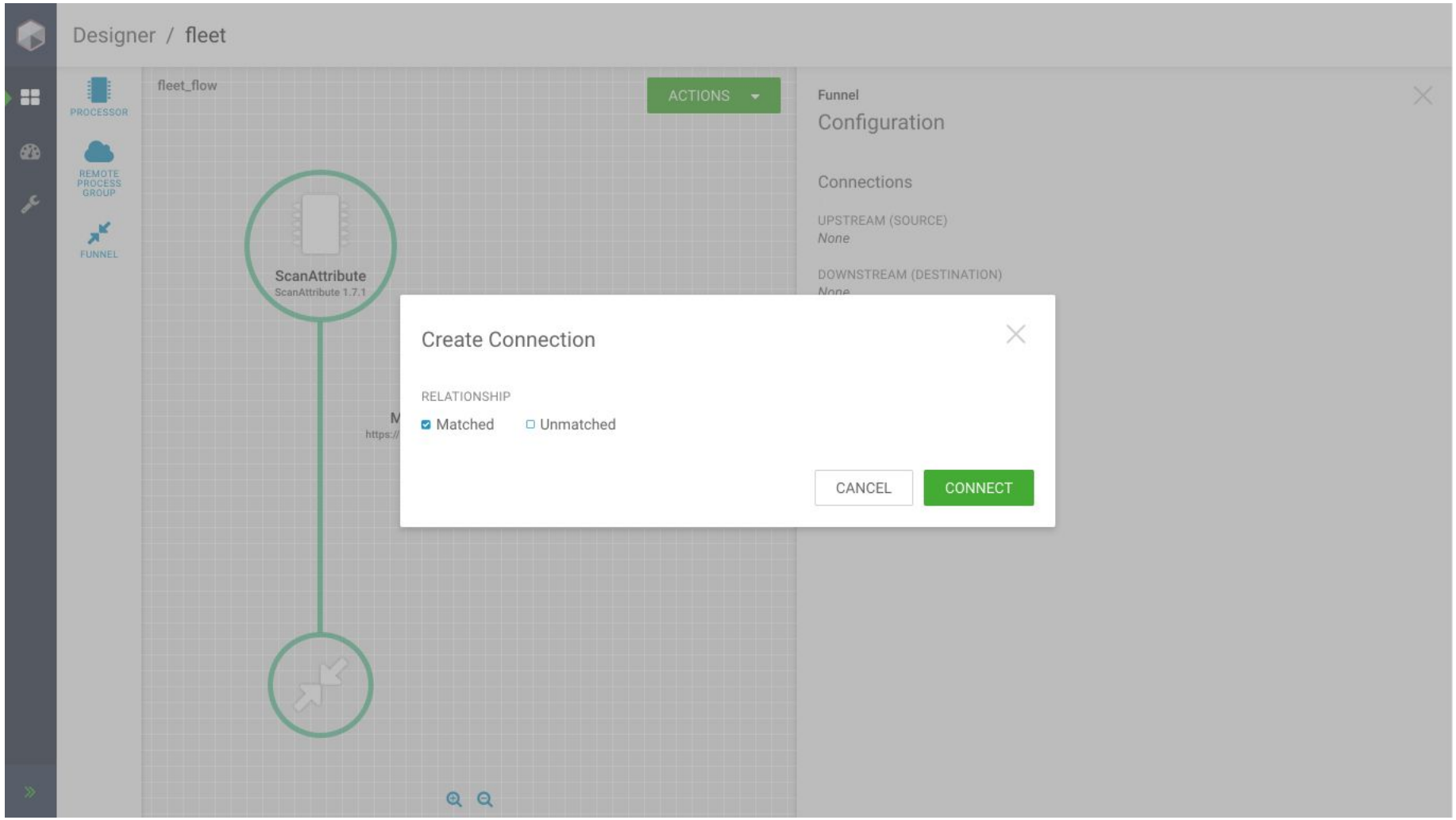
FLOW AUTHORSHIP – EFM

4) Dev then selects JSON processor on canvas and configures it based on the new gizmo requirements



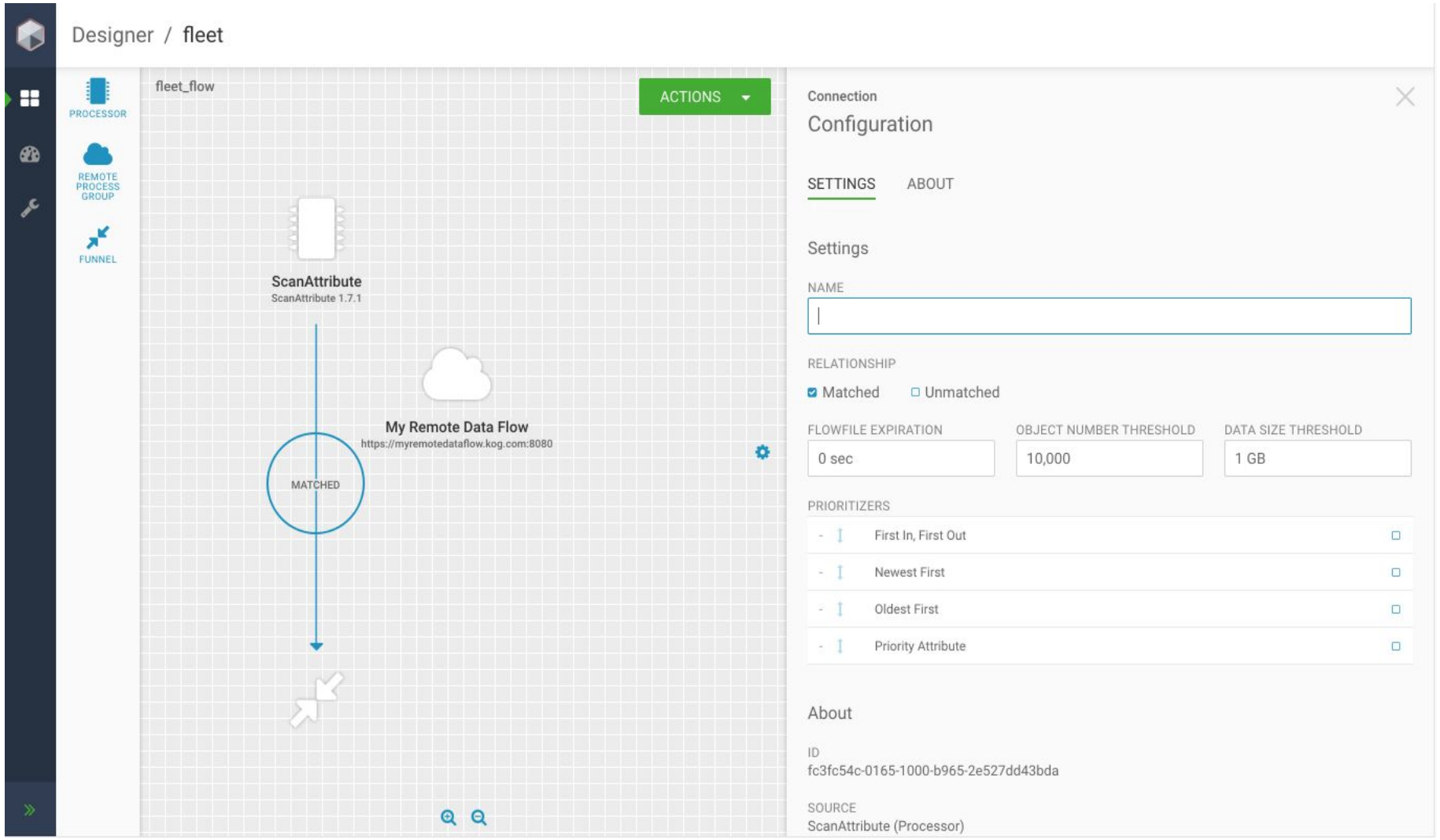
FLOW AUTHORSHIP – EFM

5) Developer creates new connection from JSON processor to upstream Remote Processing Group



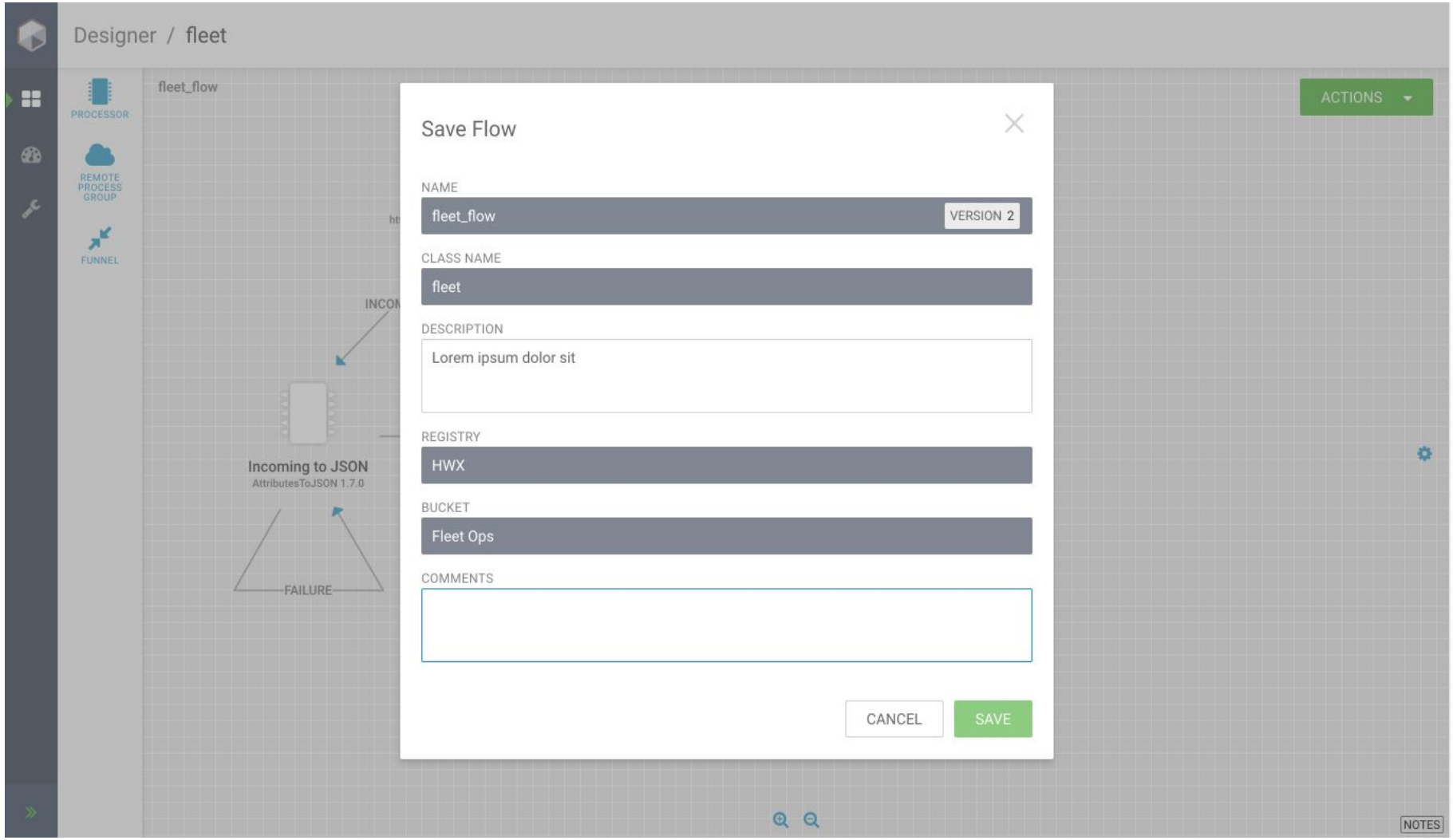
FLOW AUTHORSHIP – EFM

6) Connection properties are configured in the same manner as a processors properties



FLOW AUTHORSHIP – EFM

7) Once dev is complete. Developer enters description about changes in preparation for publishing to NiFi Registry. Note: Saves happen **constantly**



FLOW AUTHORSHIP – NEW DATA FLOW REQUIREMENTS

8) After the ”Publish” button is clicked behind the scenes the C2 Server sends the JSON flow definition to the NiFi Registry

9) The NiFi Registry receives the JSON flow definition and commits that flow to Git

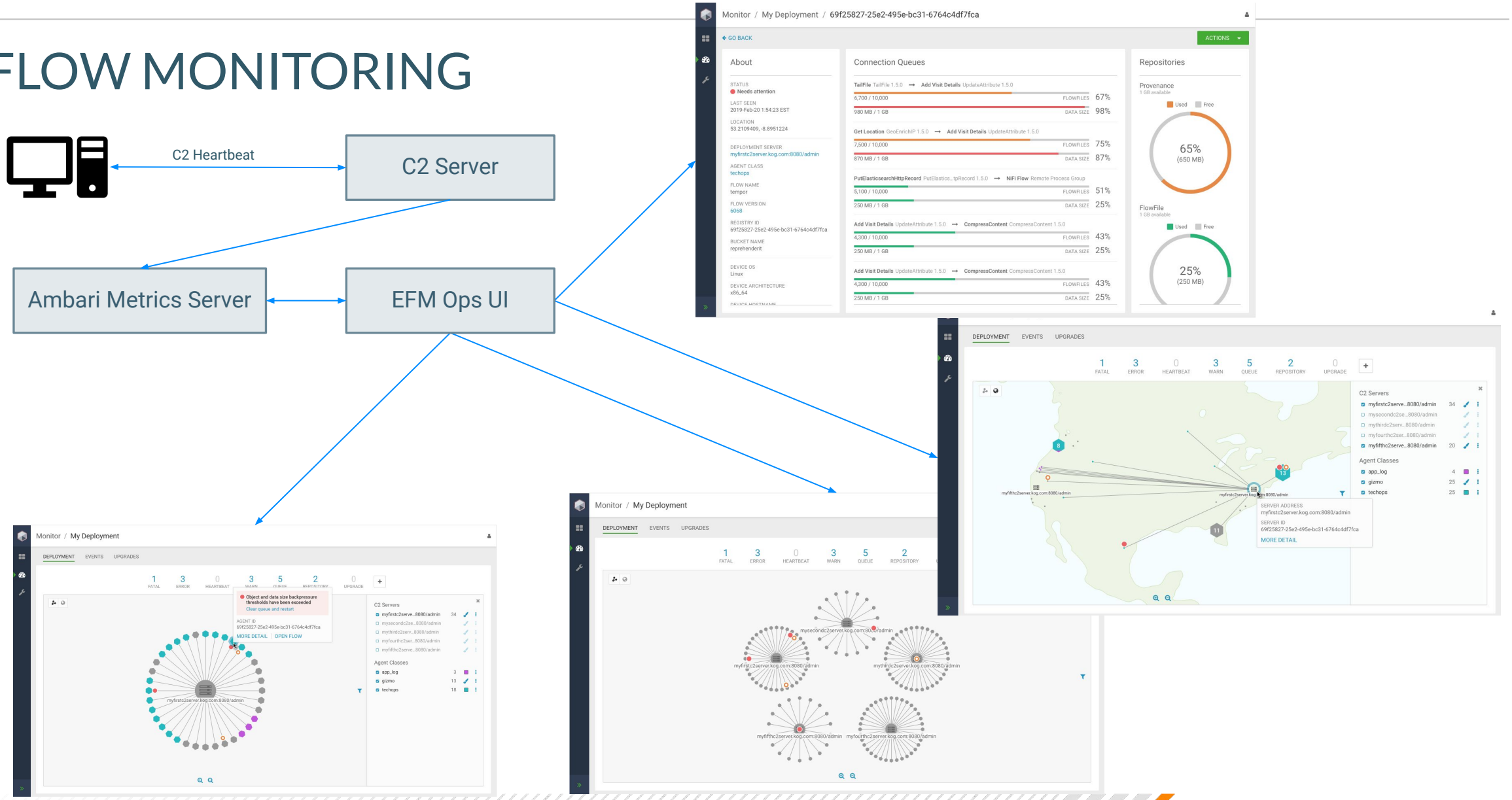
10) The C2 Server now locates all agents that are part of the gizmo class that the flow was developed for

11) Those agents are marked as “updates available” in the C2 Server memory structure

12) As agents send their heartbeats to the C2 Server IF they are part of this gizmo class of agents the C2 Server will send the

FLOW MONITORING (ROADMAP)

FLOW MONITORING

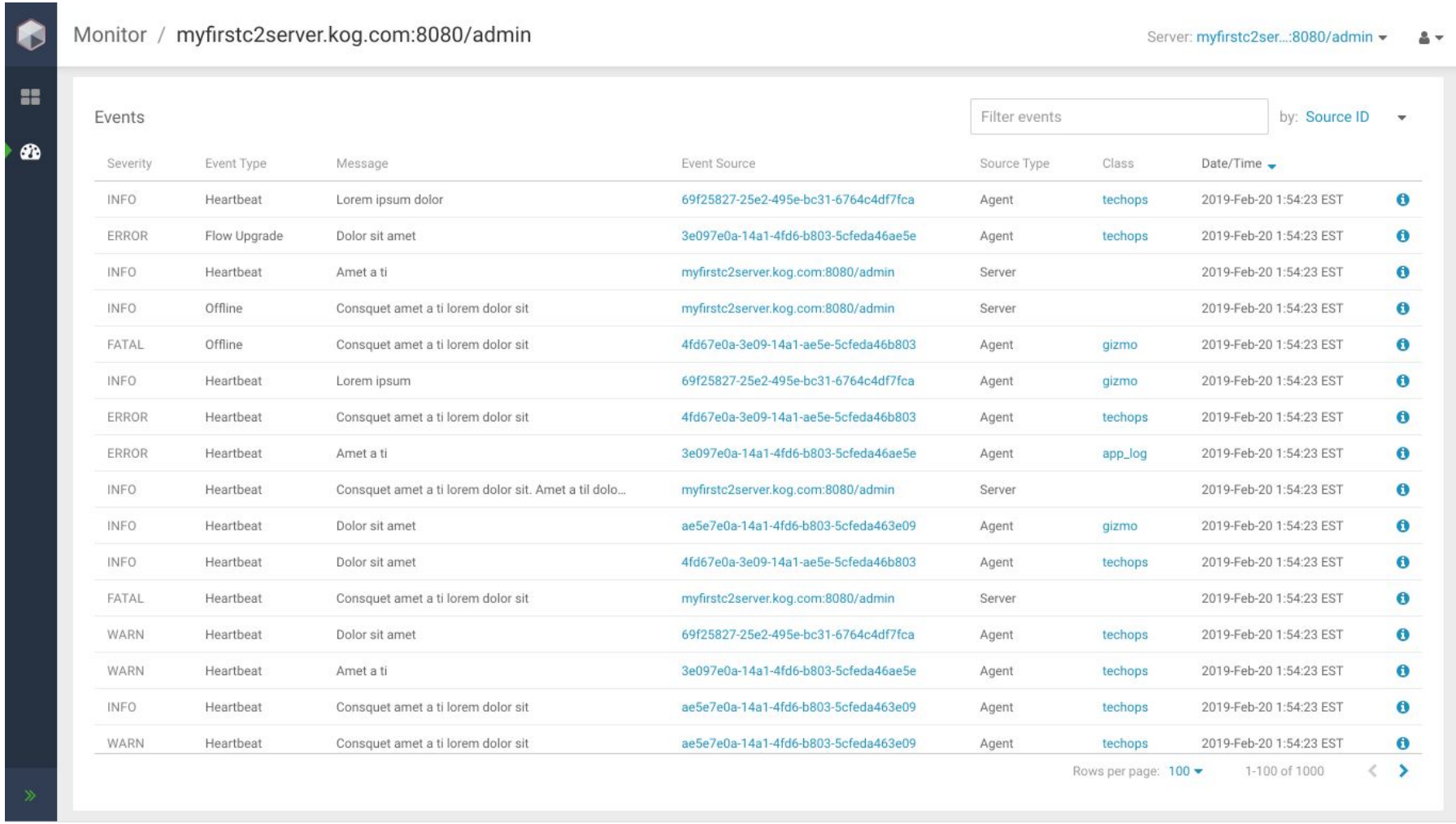


1) Admin visits Grafana dashboard to understand overall environment health

2) Notices an agent with lower than expected heartbeats

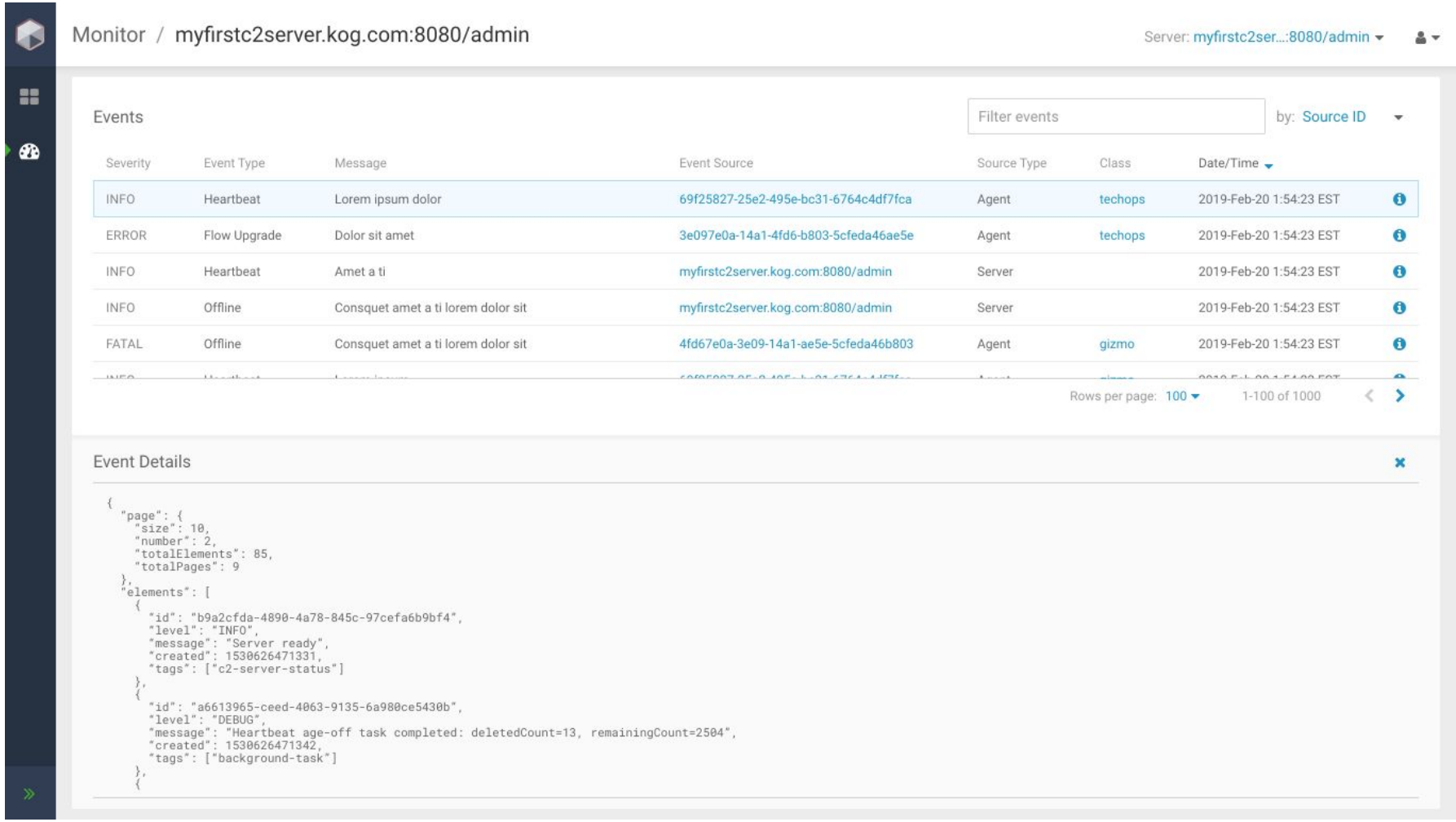
FLOW MONITORING

3) Navigates to EFM Events page to get more detailed information



FLOW MONITORING

4) Filters for Events based on agent id. Views heartbeat details to diagnose problem



LAB 3

Lab 3

Working with MiNiFi