cloudera

BYOP: Custom Processor Development with Apache NiFi

Andy LoPresto | @yolopey
Cloudera Data in Motion Security
Dataworks Summit Barcelona | March 21, 2019

Agenda

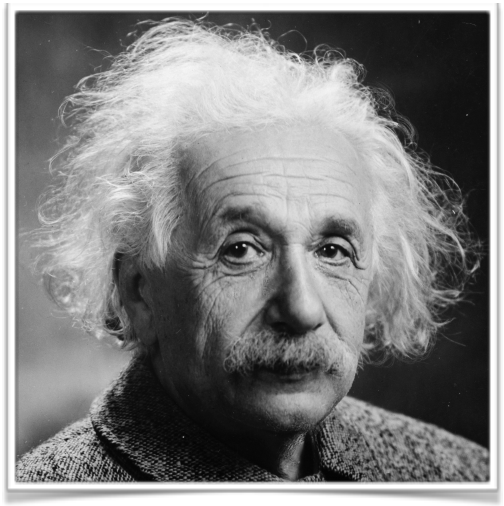
- Apache NiFi out of the box
- Extending NiFi functionality
- Custom processor development
- Testing
- Deploying
- Best practices
- Takeaways
- Additional resources
- Questions

** All slides provided online https://github.com/alopresto/slides*

Gauging Audience Familiarity With NiFi

"What's a NeeFee?"

No experience with dataflow
No experience with NiFi



"I can pick this up pretty quickly"

Some experience with dataflow
Some experience with NiFi

"I refactored the Ambari integration endpoint to allow for mutual authentication TLS during my coffee break"

Forgotten more about NiFi
than most of us will ever know

Apache NiFi

One Minute Intro to the NiFi Ecosystem

NiFi

- Server/DC class
- GUI & REST API
- Interacts with hundreds of services

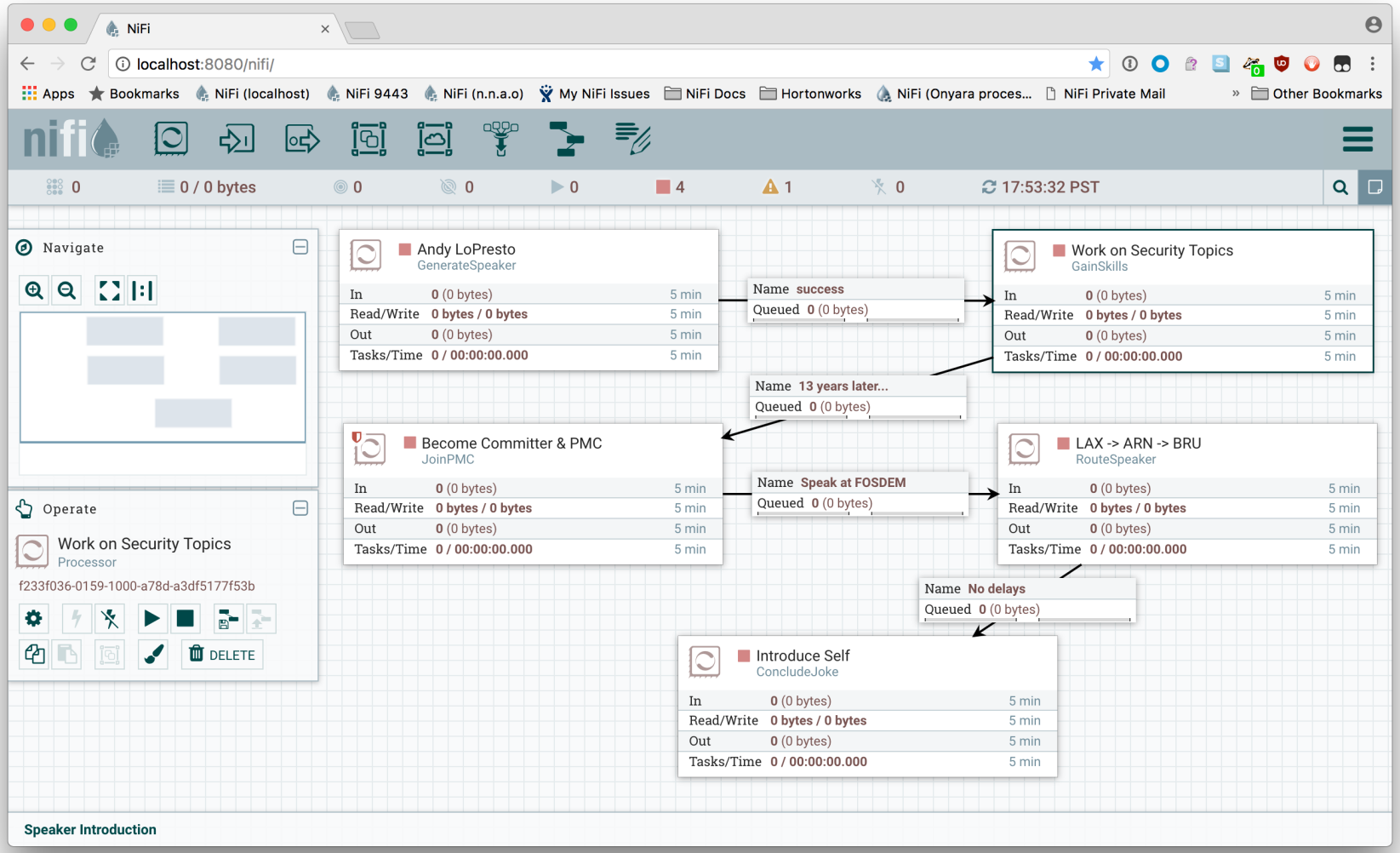
MiNiFi (JVM/C++)

- Agent class
- Runs on minimal hardware
- Interacts with NiFi
- Simple event processing

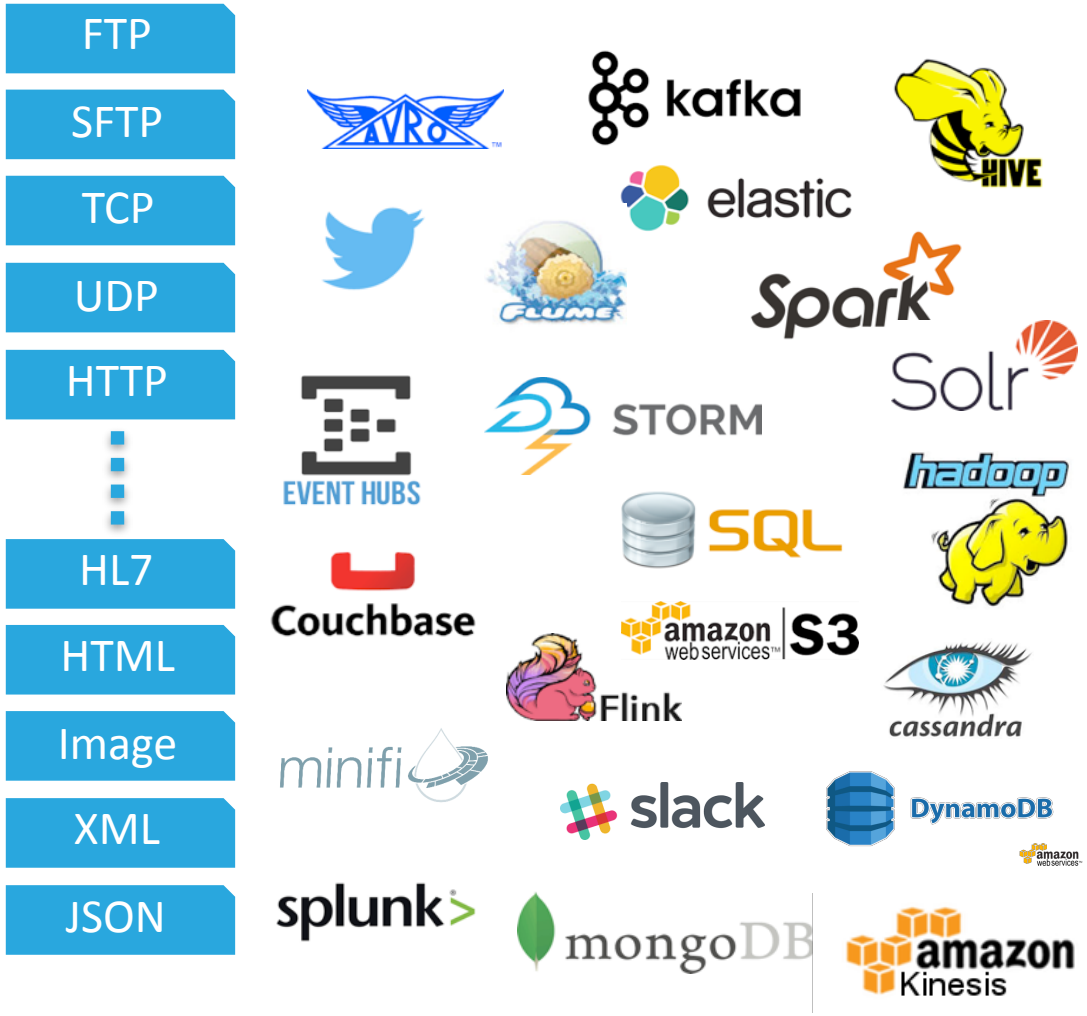
NiFi Registry

- Standalone application
- Handles asset management
- Flow versioning & deployment
- Extension registry

User Interface



Deeper Ecosystem Integration: 286+ Processors, 61 Controller Services



Hash	Encrypt	GeoEnrich
Merge	Tail	Scan
Duplicate	Evaluate	Replace
Extract	Execute	Convert
Split	Fetch	Transform

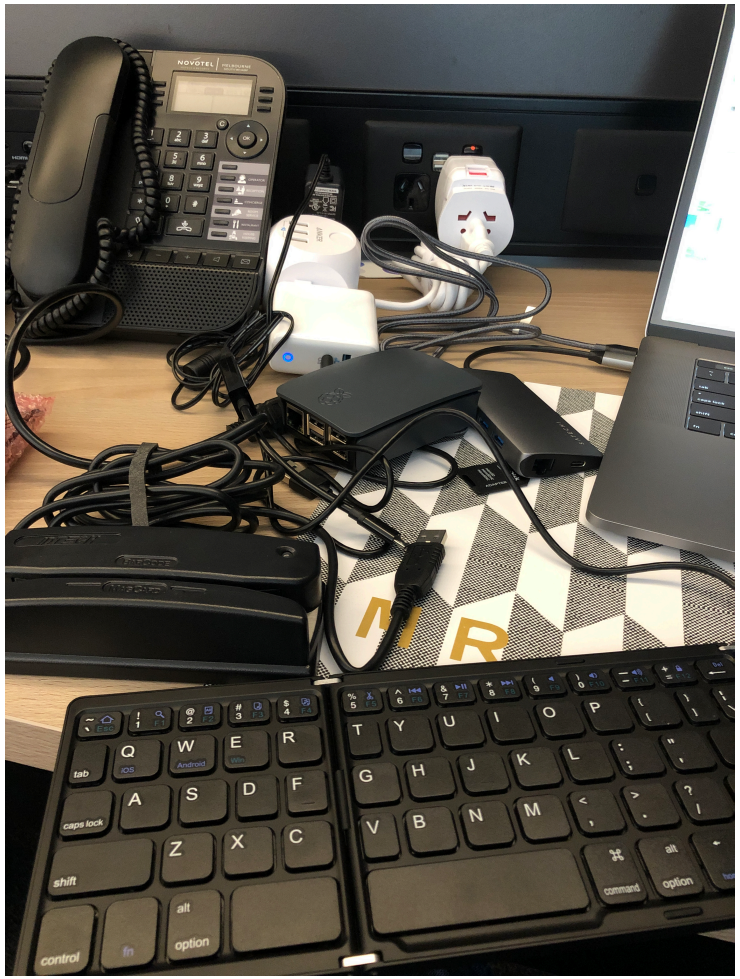
Parse Records	Convert Records
Route Content	Distribute Load
Route Context	Generate Table Fetch
Route Text	Jolt Transform JSON
Control Rate	Prioritize Delivery

All Apache project logos are trademarks of the ASF and the respective projects.

Extending NiFi functionality

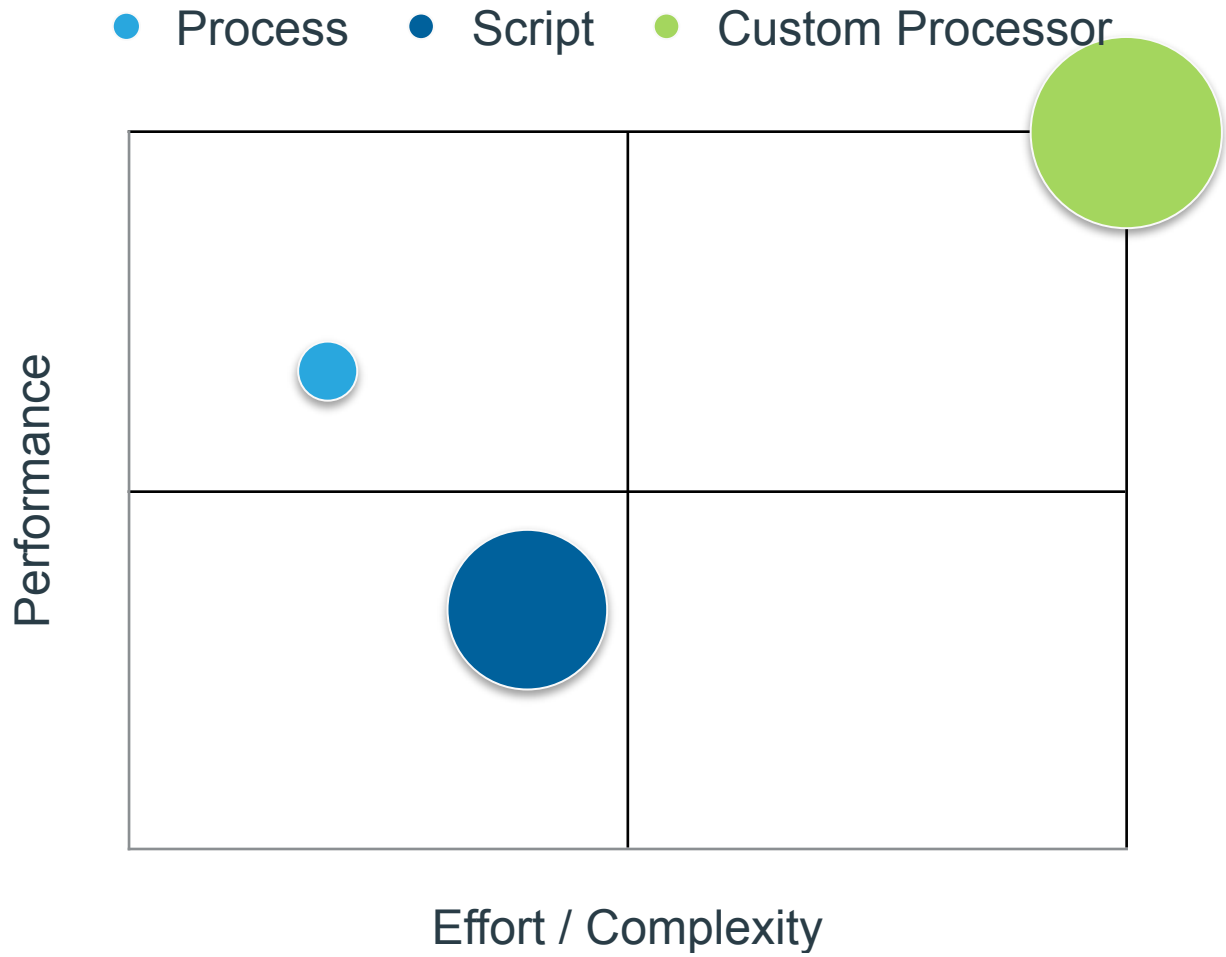
With ~300 processors, what else is left?

- Custom/proprietary protocols
- Read serial interface
  - *USB devices, GPIO on Raspberry Pi*
- Transactional operations
- Experimentation
- Commercial API/SDK
- Legacy code



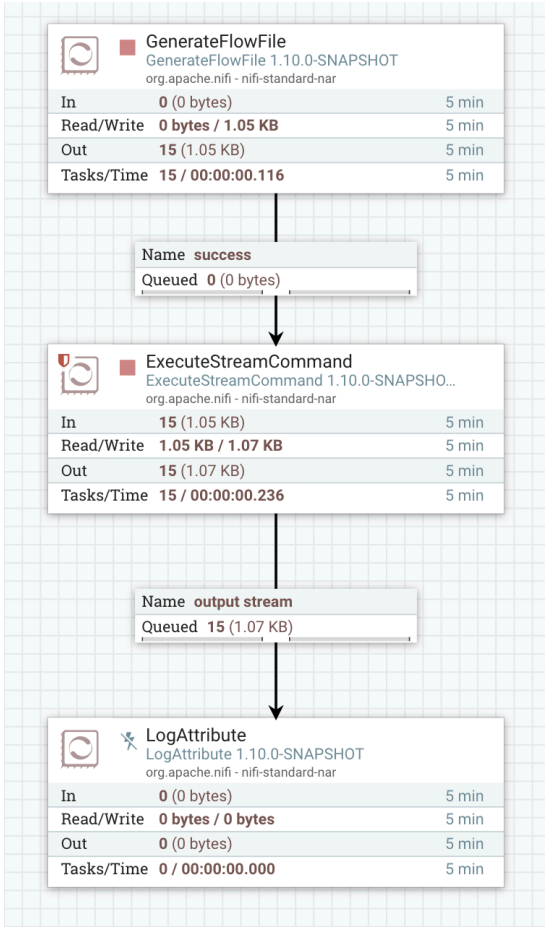
Capabilities of custom development

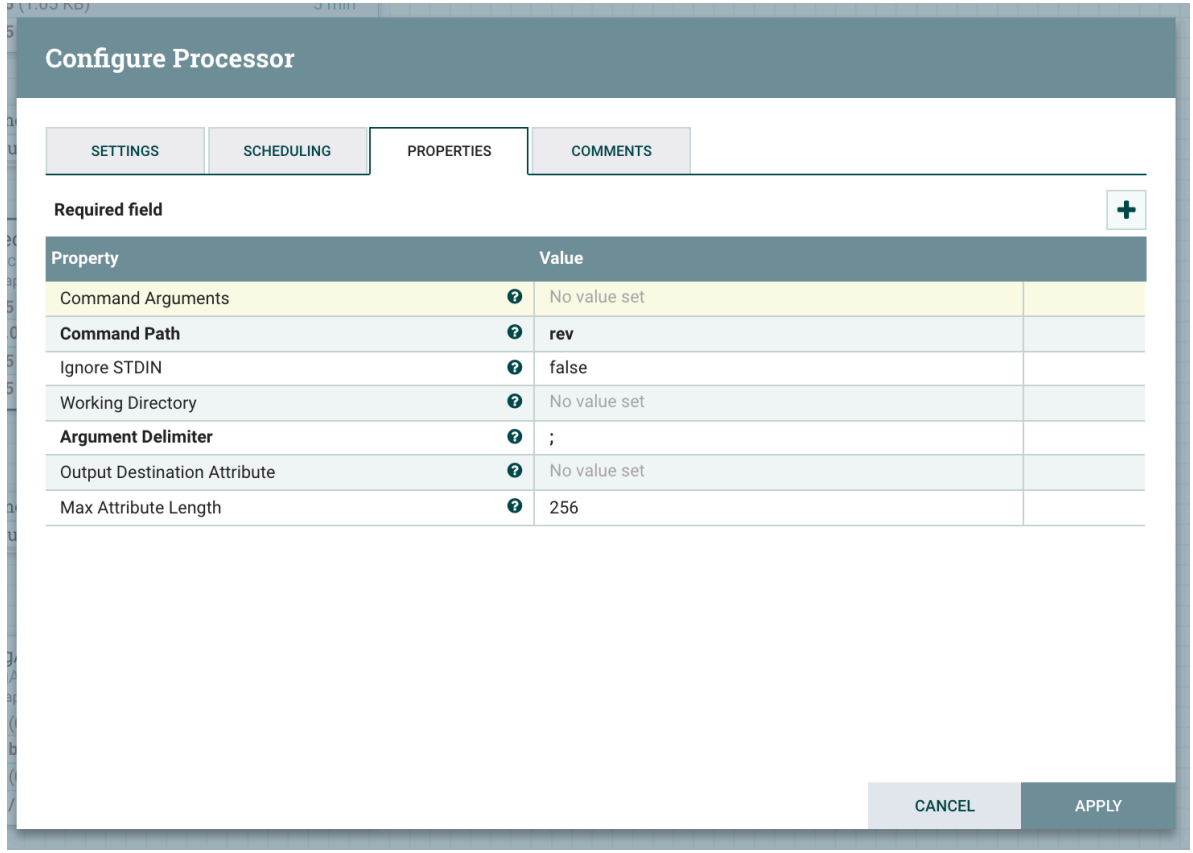
- ExecuteProcess / ExecuteStreamCommand
- ExecuteScript / InvokeScriptedProcessor
  - ExecuteGroovyScript
- Custom Processor



ExecuteStreamCommand

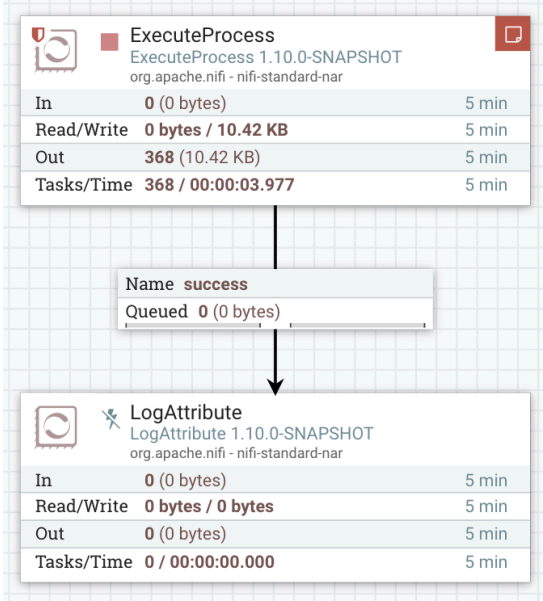
- Pipes flowfile content to STDIN
- Populates outgoing flowfile content from STDOUT
  - Can also direct to named attribute

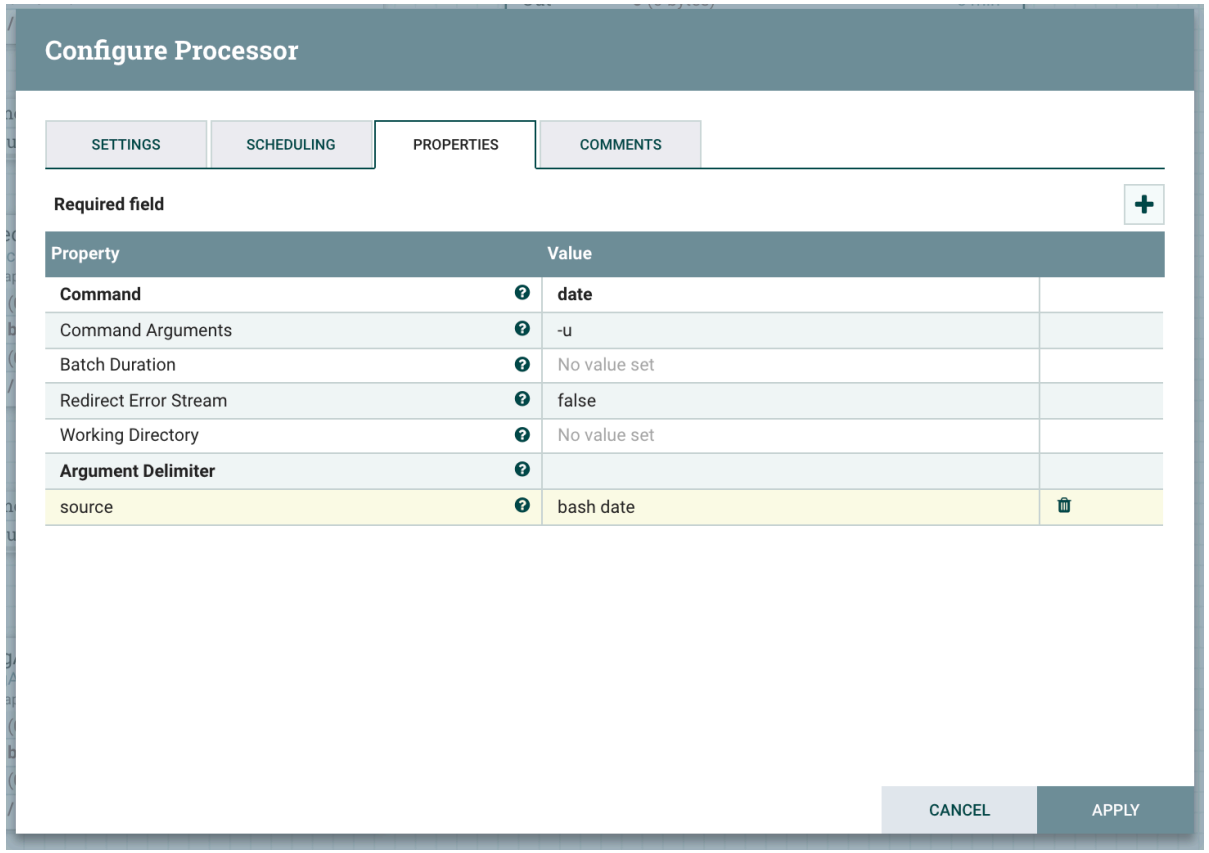




ExecuteProcess

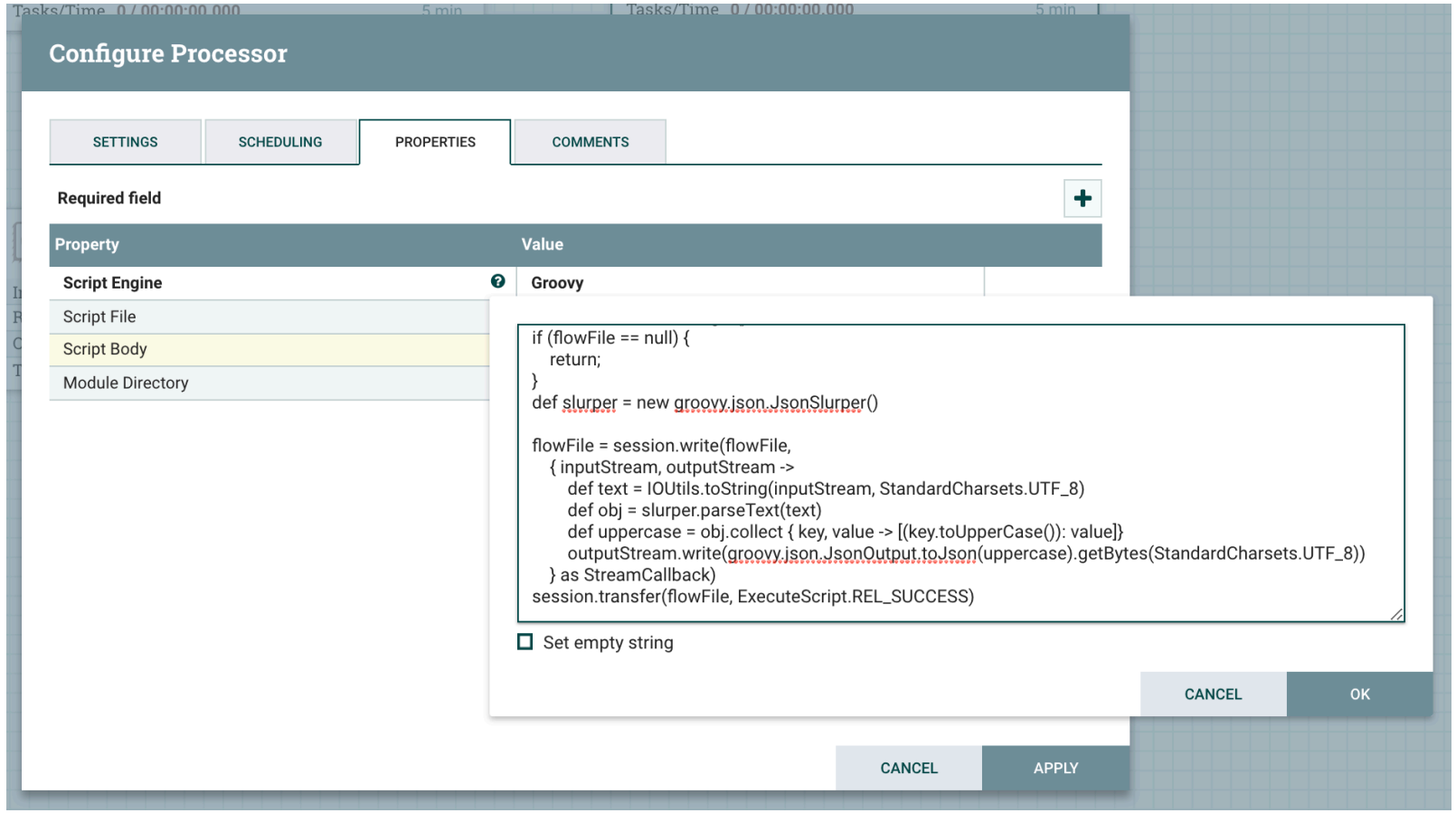
- Executes arbitrary process and captures output
- *No incoming flowfile*





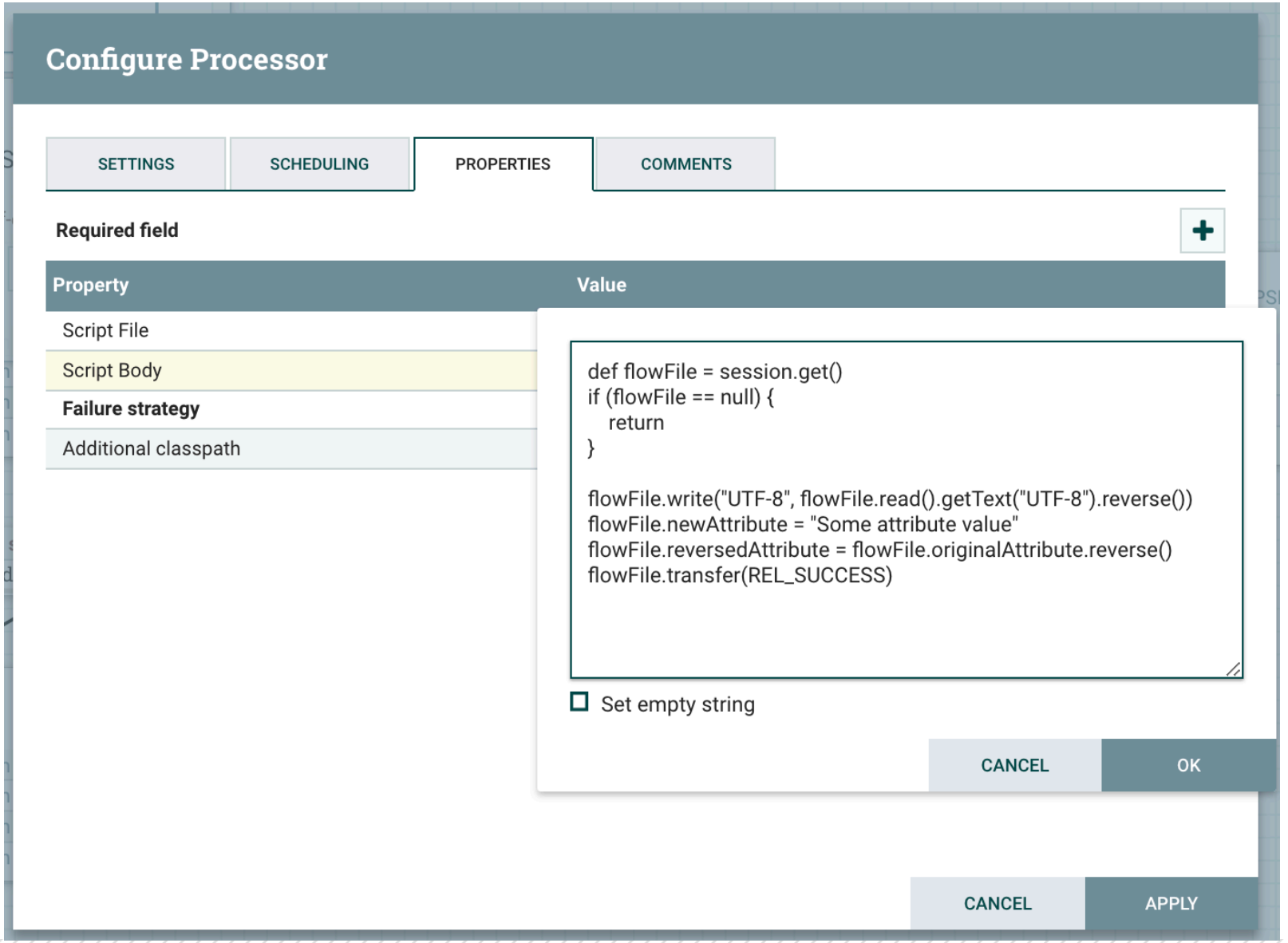
ExecuteScript

- Executes arbitrary code in a JSR-223 compatible language
  - Clojure
  - Groovy
  - Ruby
  - Python (Jython - *no C libs*)
  - Lua
  - Javascript
- Source is inline or file system



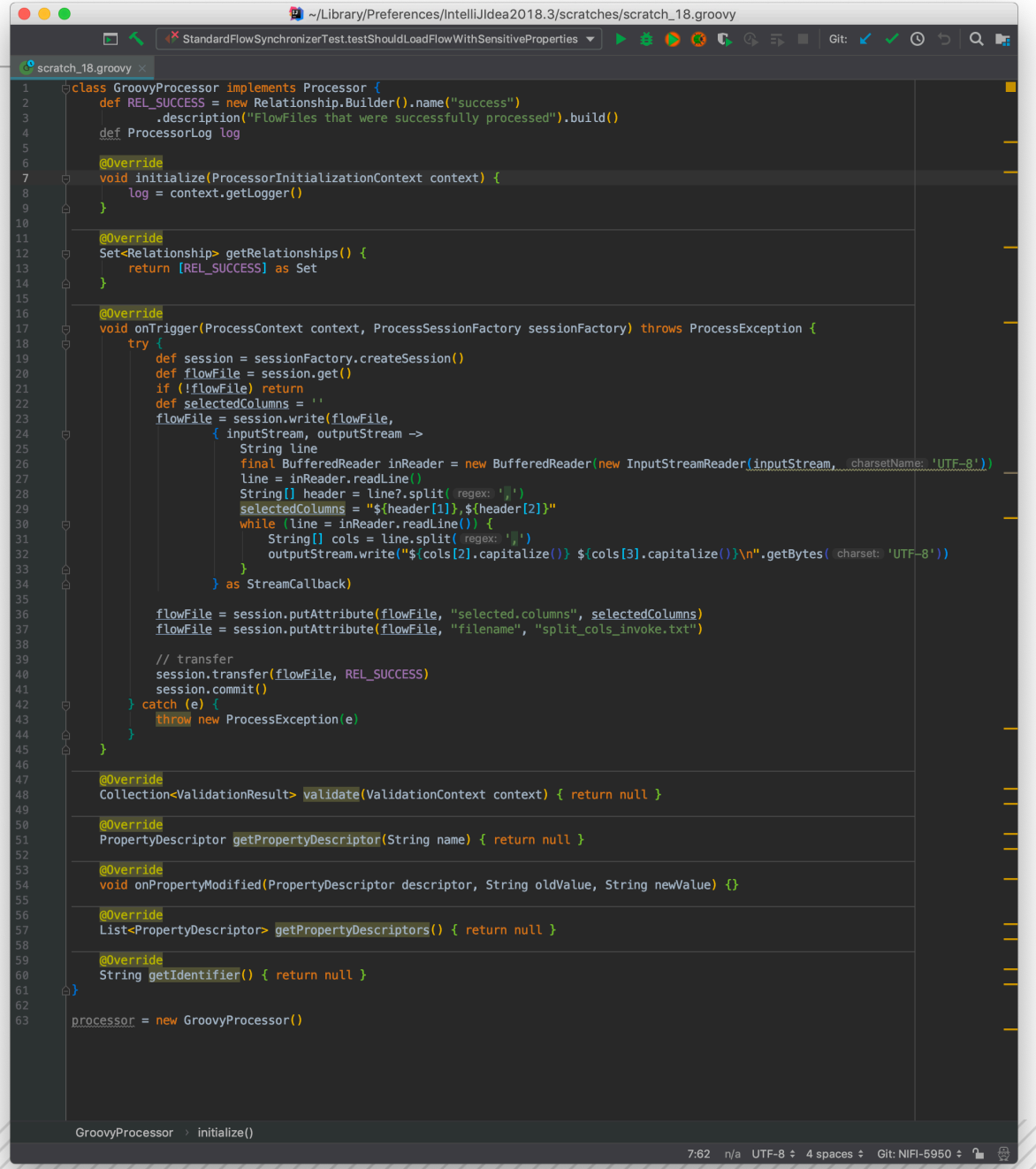
ExecuteGroovyScript

- Community-contributed
- Groovy only
  - Allows for friendlier syntax
  - Allows for Controller Service direct reference



InvokeScriptedProcessor

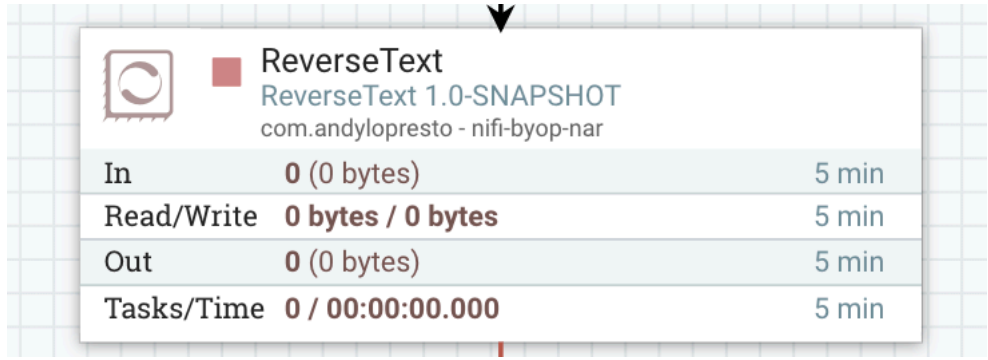
- *"ExecuteScript = onTrigger()"*
- Allows for additional methods
- Not compiled for each flowfile
- Supports custom relationships
- Improves performance



Custom processor development

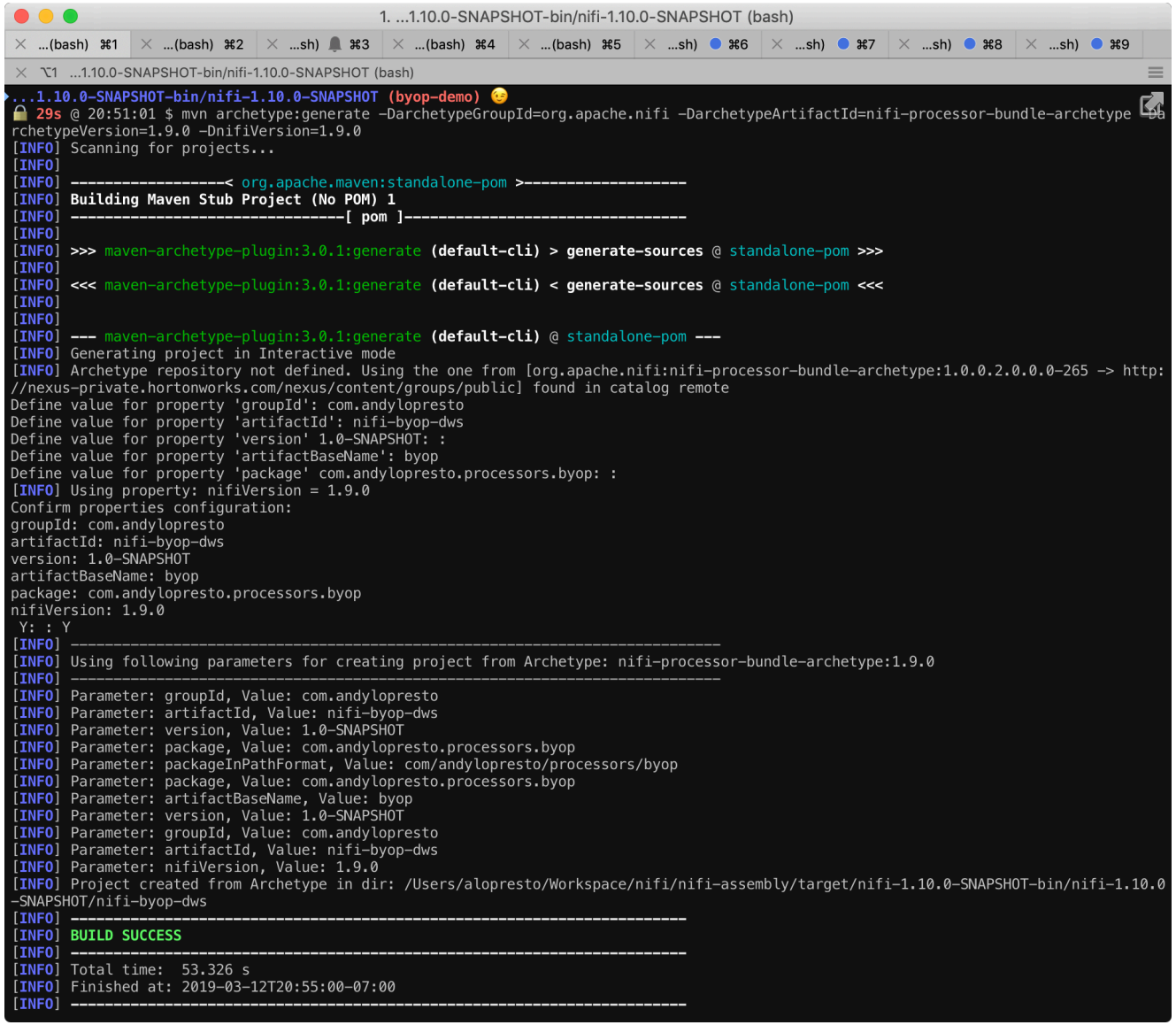
Custom Processor

- High performance
- Reusability & convenience
- Versioning



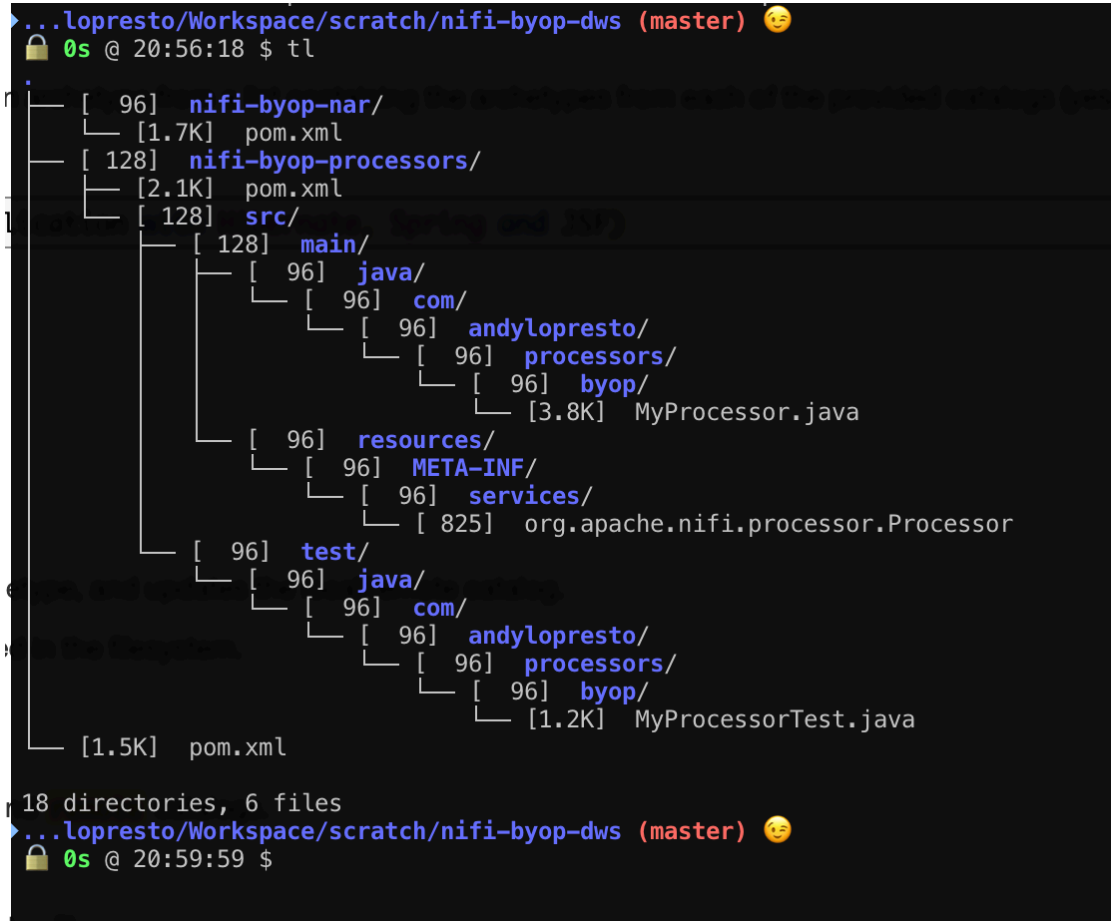
Using the Maven Archetype

- Prompts for metadata input
  - `groupId`
  - `artifactId`
  - `version`
  - `artifactBaseName`
  - `package`
- Generates a Maven project structure



Results of the Maven Archetype

- `MyProcessor.java`
  - Source code for processor
- `org.apache.nifi.processor.Processor`
  - Manifest file listing implementations
- `MyProcessorTest.java`
  - JUnit test
- `pom.xml`
  - Maven Project Object Model file



Metadata

- `@Tags`
  - Helpful for discovery
- `@CapabilityDescription`
  - Shown in documentation
  - Annotations for behavior
- `@SeeAlso`
  - References similar/complementary processors
- `@Reads/WritesAttributes`
  - Listed in documentation

```
43  @Tags({"example"})
44  @CapabilityDescription("Provide a description")
45  @SeeAlso({})
46  @ReadsAttributes({@ReadsAttribute(attribute="", description="")})
47  @WritesAttributes({@WritesAttribute(attribute="", description="")})
48  public class MyProcessor extends AbstractProcessor {
```

Selected other annotations

- `@EventDriven`
  - Indicates not based on timer
- `@SideEffectFree`
  - Indicates safe to run w/o side effects
- `@SupportsBatching`
  - Indicates multiple flowfiles processed together
- `@InputRequirement`
  - Can't be the first processor in a flow
- `@SystemResourceConsideration`
  - Warns users of implications
- `@Experimental`
  - Self-explanatory
- `@Restricted`
  - Controls access

```
67    @EventDriven
68    @SideEffectFree
69    @SupportsBatching
70    @InputRequirement(Requirement.INPUT_REQUIRED)
71    @Tags({"encryption", "decryption", "password", "JCE", "OpenPGP", "PGP", "GPG"})
72    @CapabilityDescription("Encrypts or Decrypts a FlowFile using either symmetric
        generated salt, or asymmetric encryption using a public and secret key.")
73    @SystemResourceConsideration(resource = SystemResource.CPU)
74    public class EncryptContent extends AbstractProcessor {
```

Implementing the code

- Add property descriptors and relationships
- Implement `onTrigger()` method
  - This is where the logic is done
  - Reads properties & attributes
  - Configures external services
  - Executes data transformation
  - Transfers flowfile
- Add custom `StreamCallback` class
  - Handles “behavior” of reversing String
- “32” lines of code

```java
/.../
package com.andylopresto.processors.byop;

import ...

@Tags({"simple", "text", "demo", "reverse"})
@CapabilityDescription("Simple demo processor which reverses flowfile text content. ")
public class ReverseText extends AbstractProcessor {
    public static final Relationship REL_SUCCESS = new Relationship.Builder()
            .name("REL_SUCCESS")
            .description("Success relationship")
            .build();

    private List<PropertyDescriptor> descriptors;

    private Set<Relationship> relationships;

    @Override
    protected void init(final ProcessorInitializationContext context) {
        this.descriptors = Collections.emptyList();

        final Set<Relationship> relationships = new HashSet<>();
        relationships.add(REL_SUCCESS);
        this.relationships = Collections.unmodifiableSet(relationships);
    }

    @Override
    public Set<Relationship> getRelationships() { return this.relationships; }

    @Override
    public final List<PropertyDescriptor> getSupportedPropertyDescriptors() { return descriptors; }

    @OnScheduled
    public void onScheduled(final ProcessContext context) {}

    @Override
    public void onTrigger(final ProcessContext context, final ProcessSession session) throws ProcessException {
        FlowFile flowFile = session.get();
        if ( flowFile == null ) {
            return;
        }

        // TODO: This should check the mime/type if available, and the content encoding to attempt to detect non-text content

        // TODO: This should be done in a streaming manner

        session.write(flowFile, new ReverseTextCallback());
        session.putAttribute(flowFile, s: "reversed", s1: "true");
        session.transfer(flowFile, REL_SUCCESS);
    }
}

class ReverseTextCallback implements StreamCallback {

    @Override
    public void process(InputStream inputStream, OutputStream outputStream) throws IOException {
        // This example only reads the first line
        String originalContent = new BufferedReader(new InputStreamReader(inputStream)).readLine();
        outputStream.write(new StringBuilder(originalContent).reverse().toString().getBytes(StandardCharsets.UTF_8));
    }
}
```

Testing

Write the tests (should be first)

- Test-driven development (TDD)
- Helpful `TestRunner` class
  - Handles enqueueing arbitrary flowfile content & attributes
  - Allows intelligent assertions

```java
/.../
package com.andylopresto.processors.byop;

import ...


public class ReverseTextTest {
    private static final Logger logger = LoggerFactory.getLogger(ReverseTextTest.class);
    private static final String MESSAGE = "This is a plaintext message. ";
    private static final String REVERSED_MESSAGE = new StringBuilder(MESSAGE).reverse().toString();

    private TestRunner testRunner;

    @Before
    public void init() { testRunner = TestRunners.newTestRunner(ReverseText.class); }

    @Test
    public void testProcessor() {
        // Arrange
        testRunner.enqueue(MESSAGE);

        // Act
        testRunner.run();
        logger.info("Ran once with " + MESSAGE);

        // Assert
        testRunner.assertAllFlowFilesTransferred(ReverseText.REL_SUCCESS, i: 1);
        MockFlowFile flowFile = testRunner.getFlowFilesForRelationship(ReverseText.REL_SUCCESS).get(0);
        flowFile.assertContentEquals(REVERSED_MESSAGE);
        flowFile.assertAttributeEquals( attributeName: "reversed", expectedValue: "true");
    }

}
```

```
Tests passed: 1 of 1 test – 124 ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_192.jdk/Contents/Home/bin/java ...
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/Users/alopresto/.m2/repository/ch/qos/logback/logback-classic/1.2.3/logback-classic-
SLF4J: Found binding in [jar:file:/Users/alopresto/.m2/repository/org/slf4j/slf4j-simple/1.7.25/slf4j-simple-1.7.25.jar
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]
21:46:17.497 [main] INFO com.andylopresto.processors.byop.ReverseTextTest - Ran once with This is a plaintext message.

Process finished with exit code 0
```

We'll come back to this…

Deploying

Build the Maven project

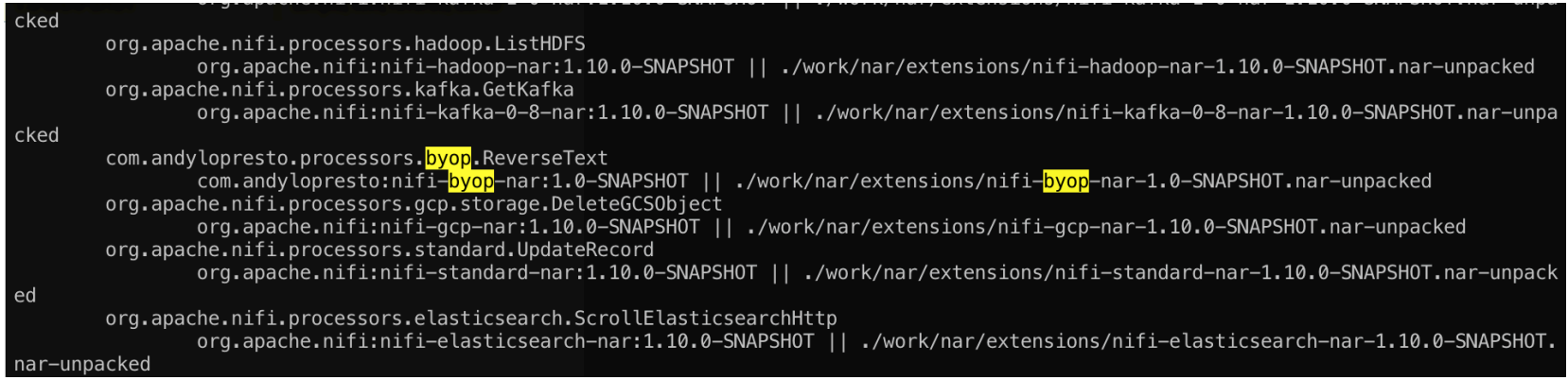
- `mvn clean install`
- Builds the project
  - Compiles source code
  - Compiles tests
  - Runs tests
  - Outputs target file(s)

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for nifi-byop-dws 1.0-SNAPSHOT:
[INFO]
[INFO] nifi-byop-dws ..................................... SUCCESS [  2.263 s]
[INFO] nifi-byop-processors .............................. SUCCESS [  2.957 s]
[INFO] nifi-byop-nar ..................................... SUCCESS [  0.242 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  7.125 s
[INFO] Finished at: 2019-03-12T22:01:00-07:00
[INFO] ------------------------------------------------------------------------
```

```
...lopresto/Workspace/scratch/nifi-byop-dws (master) 😉
🔒 9s @ 22:01:00 $ ll nifi-byop-nar/target/
total 376
drwxr-xr-x  12 alopresto  staff   384B Mar 12 22:01 ./
drwxr-xr-x   4 alopresto  staff   128B Mar 12 22:01 ../
-rw-r--r--   1 alopresto  staff    30B Mar 12 22:01 .plxarc
-rw-r--r--   1 alopresto  staff   2.8K Mar 12 22:01 checkstyle-checker.xml
-rw-r--r--   1 alopresto  staff    81B Mar 12 22:01 checkstyle-result.xml
-rw-r--r--   1 alopresto  staff   2.8K Mar 12 22:01 checkstyle-rules.xml
drwxr-xr-x   3 alopresto  staff    96B Mar 12 22:01 classes/
drwxr-xr-x   3 alopresto  staff    96B Mar 12 22:01 maven-archiver/
drwxr-xr-x   3 alopresto  staff    96B Mar 12 22:01 maven-shared-archive-resources/
-rw-r--r--   1 alopresto  staff   164K Mar 12 22:01 nifi-byop-nar-1.0-SNAPSHOT.nar
-rw-r--r--   1 alopresto  staff   729B Mar 12 22:01 rat.txt
drwxr-xr-x   3 alopresto  staff    96B Mar 12 22:01 test-classes/
```

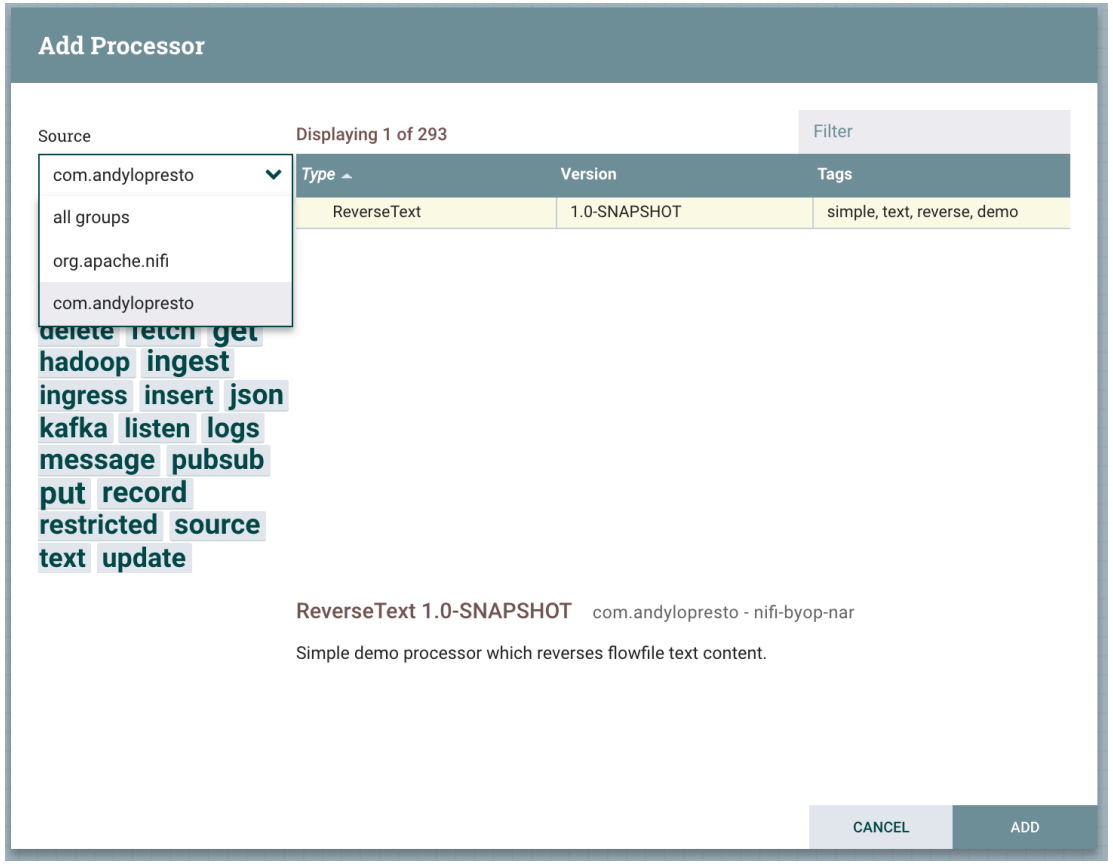
Copy the NAR

- Copy into NiFi's library
  - `$NIFI_HOME/lib`
  - NiFi will load extension automatically during startup



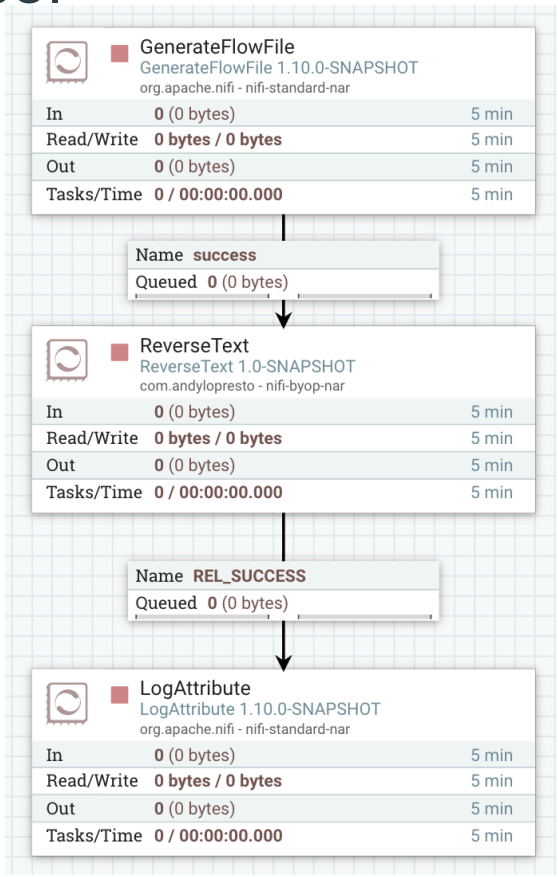
Add the processor

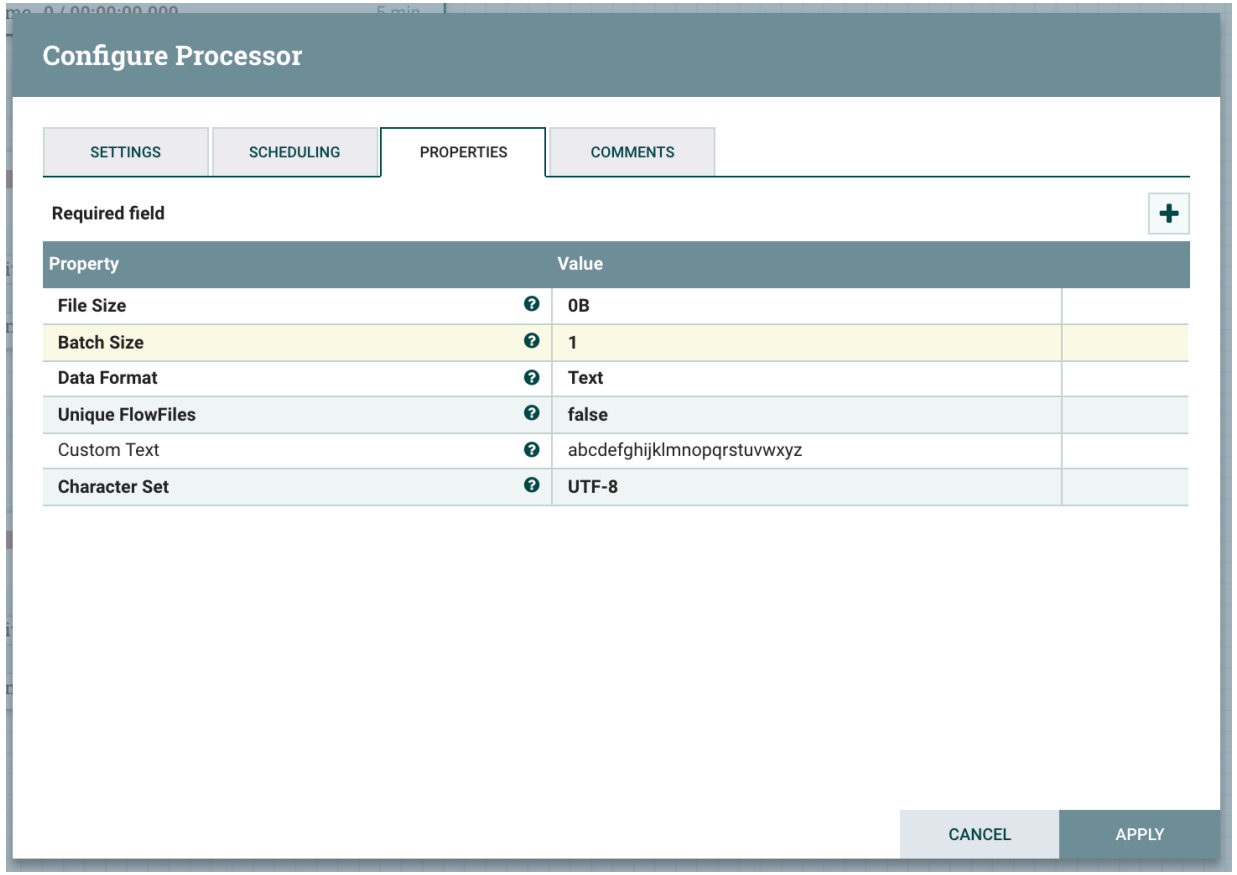
- Can filter by source group
- Can search by tags or processor name



Configure the processor

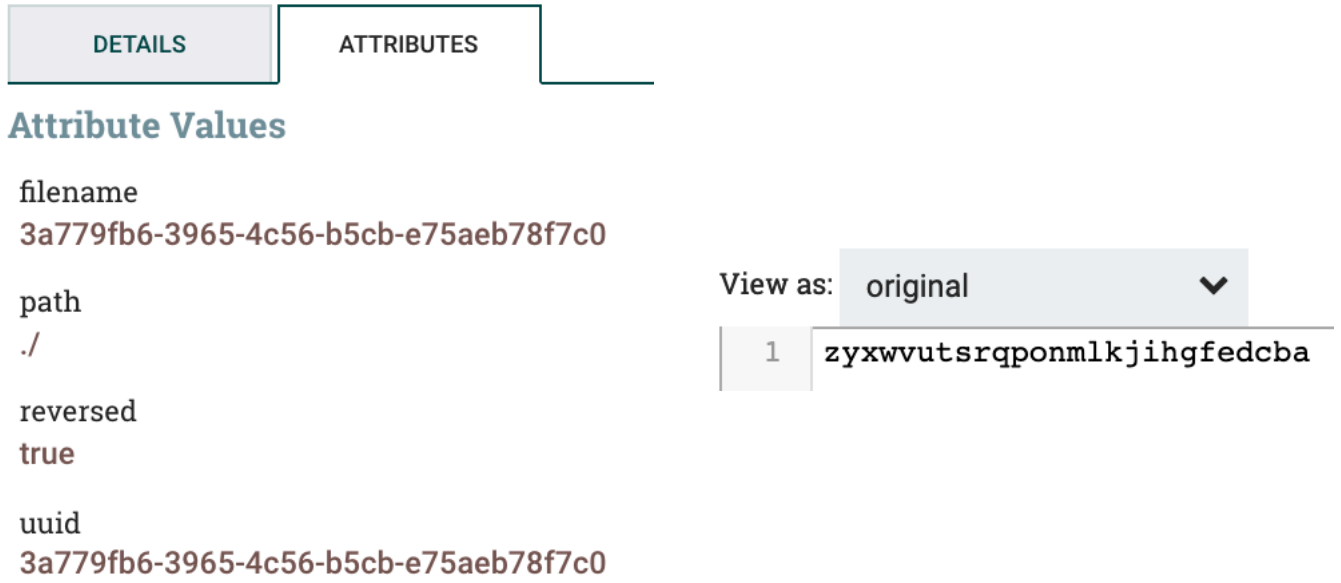
- Feed data with GenerateFlowFile





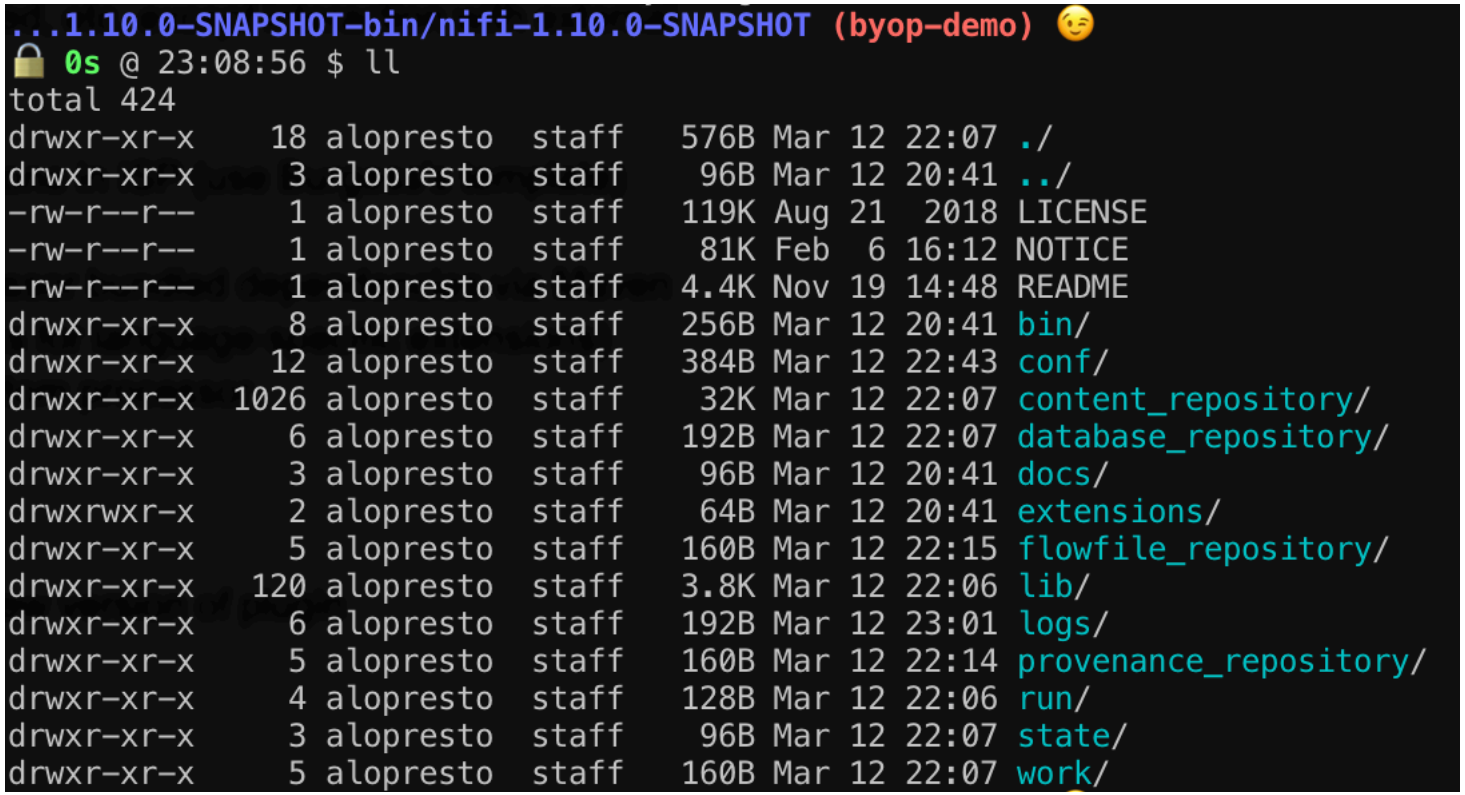
Smoke test

- List queue on the success relationship
- View attribute
  - “reversed” - true
- View content



NAR loading

- NiFi now hot-loads custom NARs in `$NIFI_HOME/extensions`
- Not good for SDLC
  - Doesn't reload matching NAR
  - *Could use build script to append timestamp to filename and version*



Testing (continued)

Testing approaches

- Unit testing
- Integration testing
- In-NiFi testing

Unit testing

- Regression testing
  - Supports refactoring
  - Supports new feature development
- Extract behavior to service class
  - Tests for logic and for “processor” behavior
  - Combinations of properties & attributes/content
- Use the mock classes
  - `MockFlowFile` - allows for internal assertions
  - `MockProcessSession` - allows for flowfile operation (create, transfer, clone, etc.)
  - `MockProcessContext` - allows for property and controller service interaction

Groovy unit testing

- Easy mocking
  - Map coercion
  - Metaclass overrides
- Sparser syntax; less boilerplate
- Spock for BDD

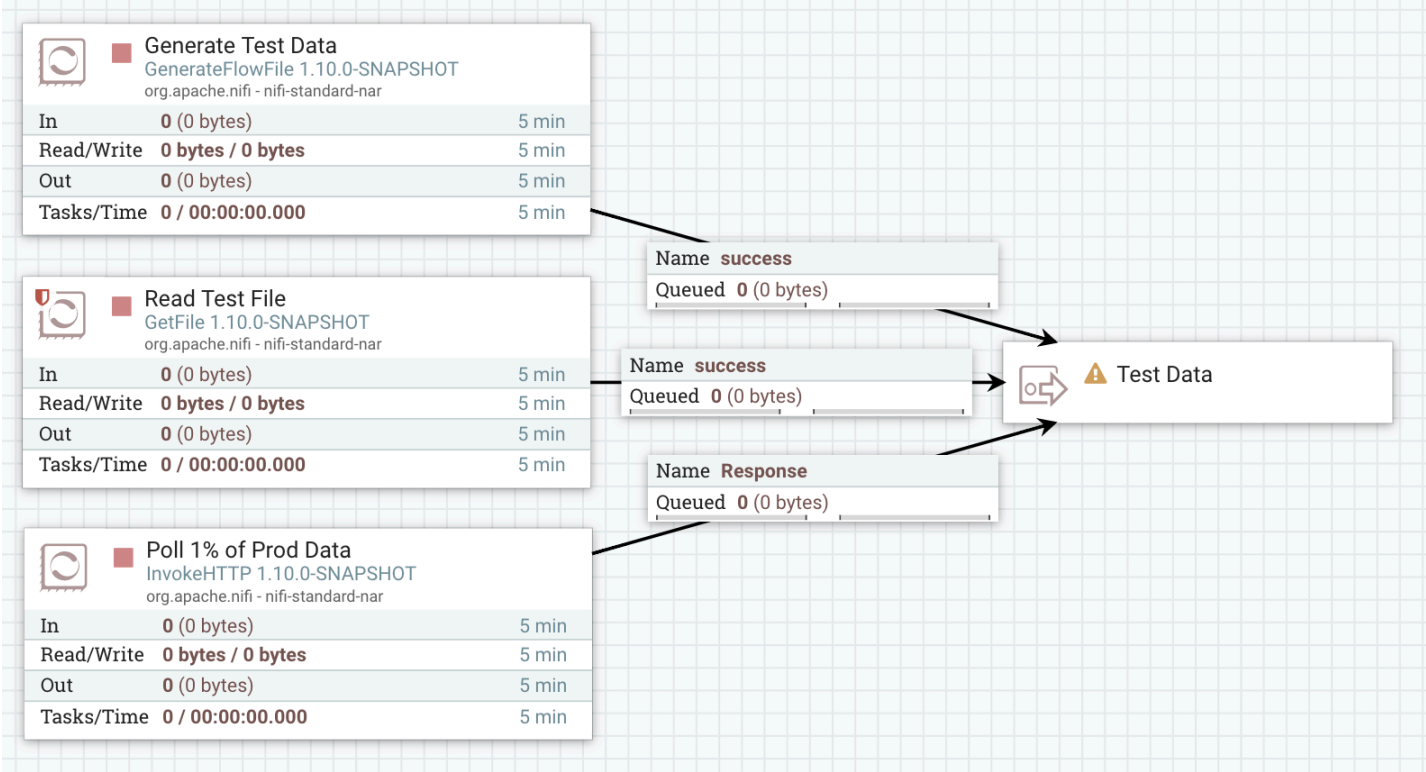
```
Security.addProvider(new BouncyCastleProvider())

logger.metaClass.methodMissing = { String name, args ->
    logger.info("[${name?.toUpperCase()}] ${(args as List).join(" ")}")
}

mockKeyProvider = [
        getKey   : { String keyId ->
            logger.mock("Requesting key ID: ${keyId}")
            new SecretKeySpec(Hex.decode(KEY_HEX), s: "AES")
        },
        keyExists: { String keyId ->
            logger.mock("Checking existence of ${keyId}")
            true
        }] as KeyProvider
```

Integration testing

- Provide external services for testing
  - Docker containers are popular
  - Easy to spin up & delete
- Example
  - `MongoDBLookupServiceIT`

Testing inside NiFi

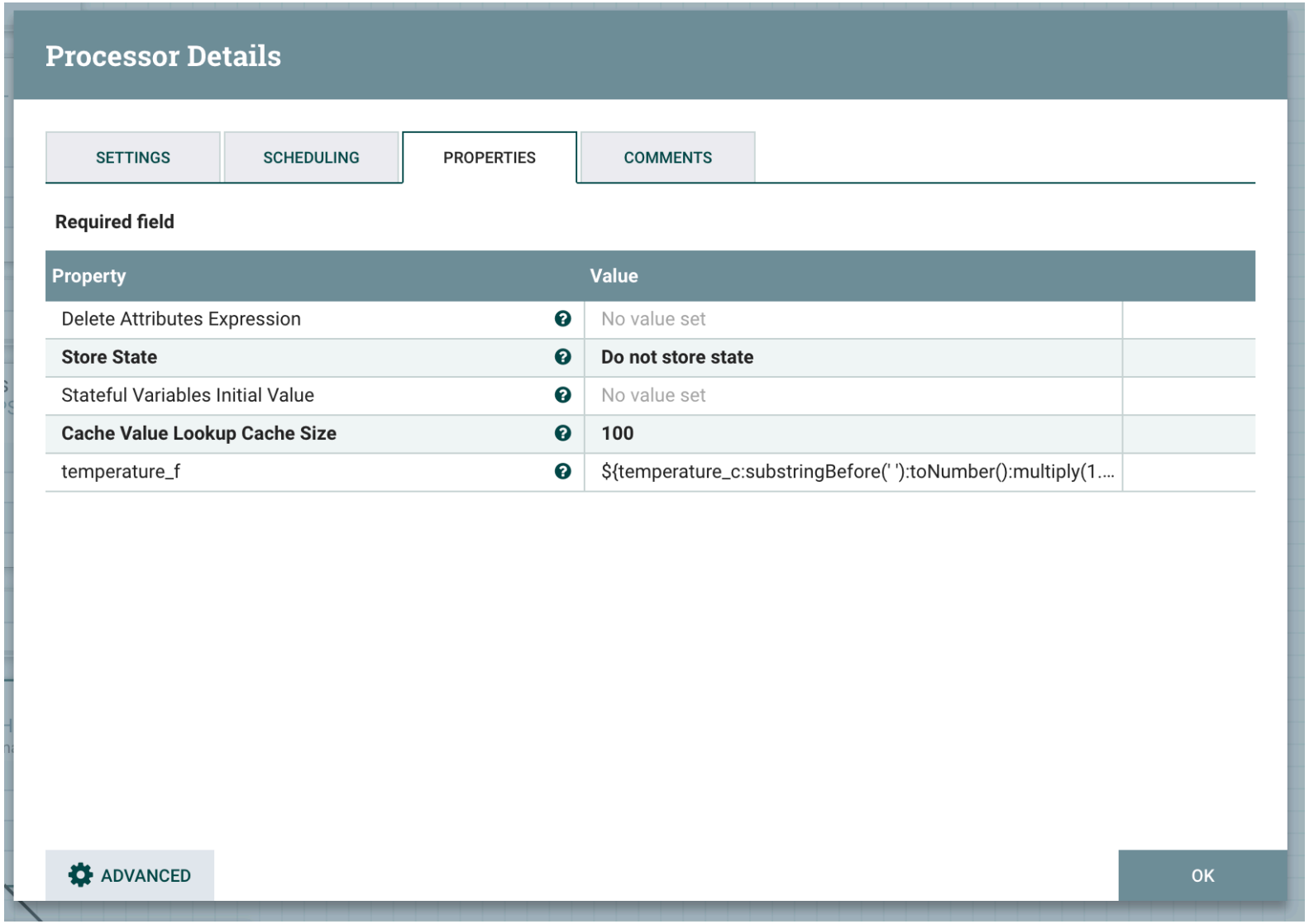
- Create reusable process group which handles test fixtures
  - Use `GenerateFlowFile`
  - Call external APIs
  - Read from test data on file system
  - Read from test database
  - Connect output port to “flow under test”



Other capabilities

Custom User Interface

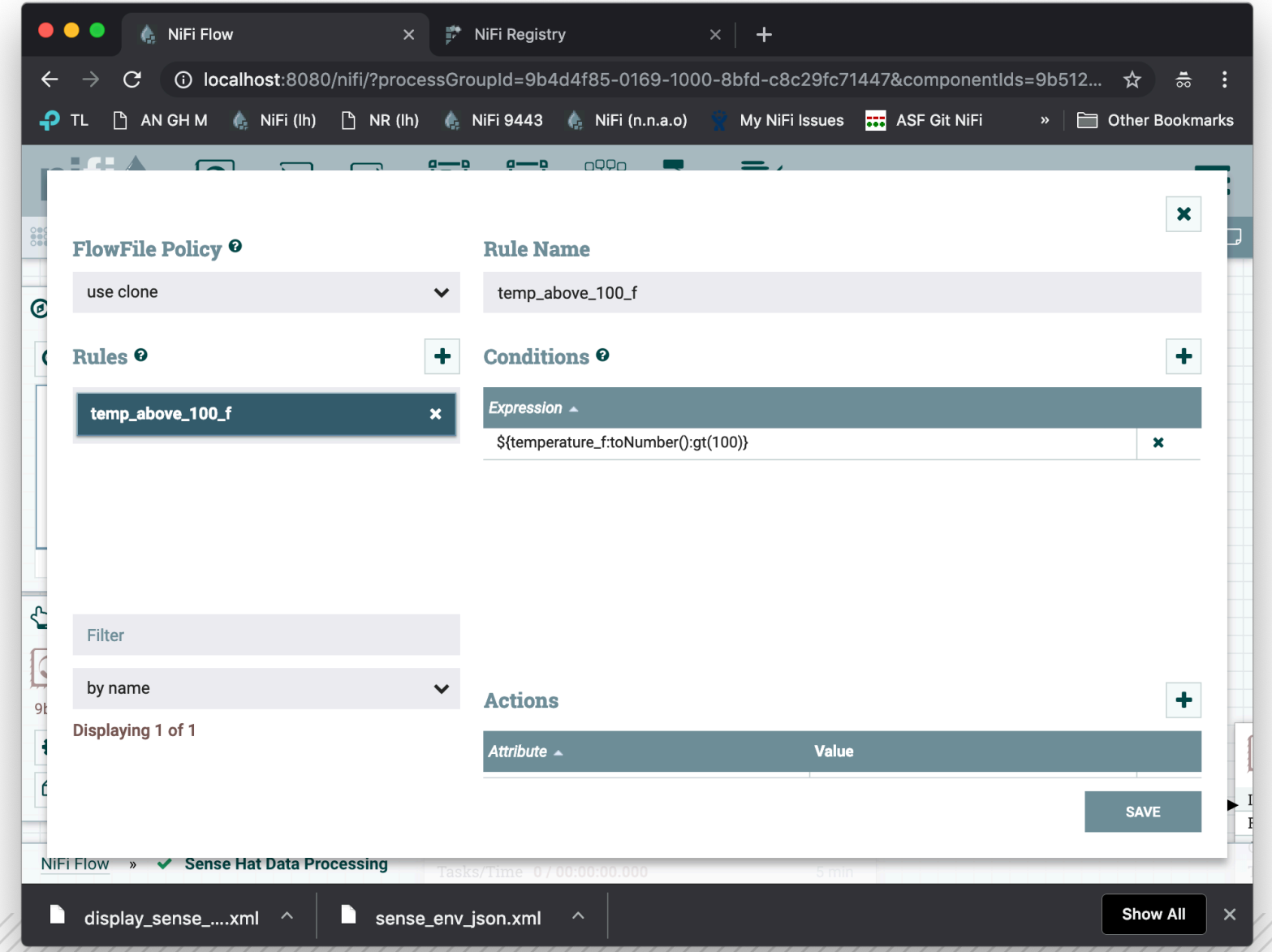
- Sometimes generic property descriptors don't make sense
  - Niels Basjes' `logparser` project with 100+ boolean flags that are auto-detected



https://github.com/nielsbasjes/logparser

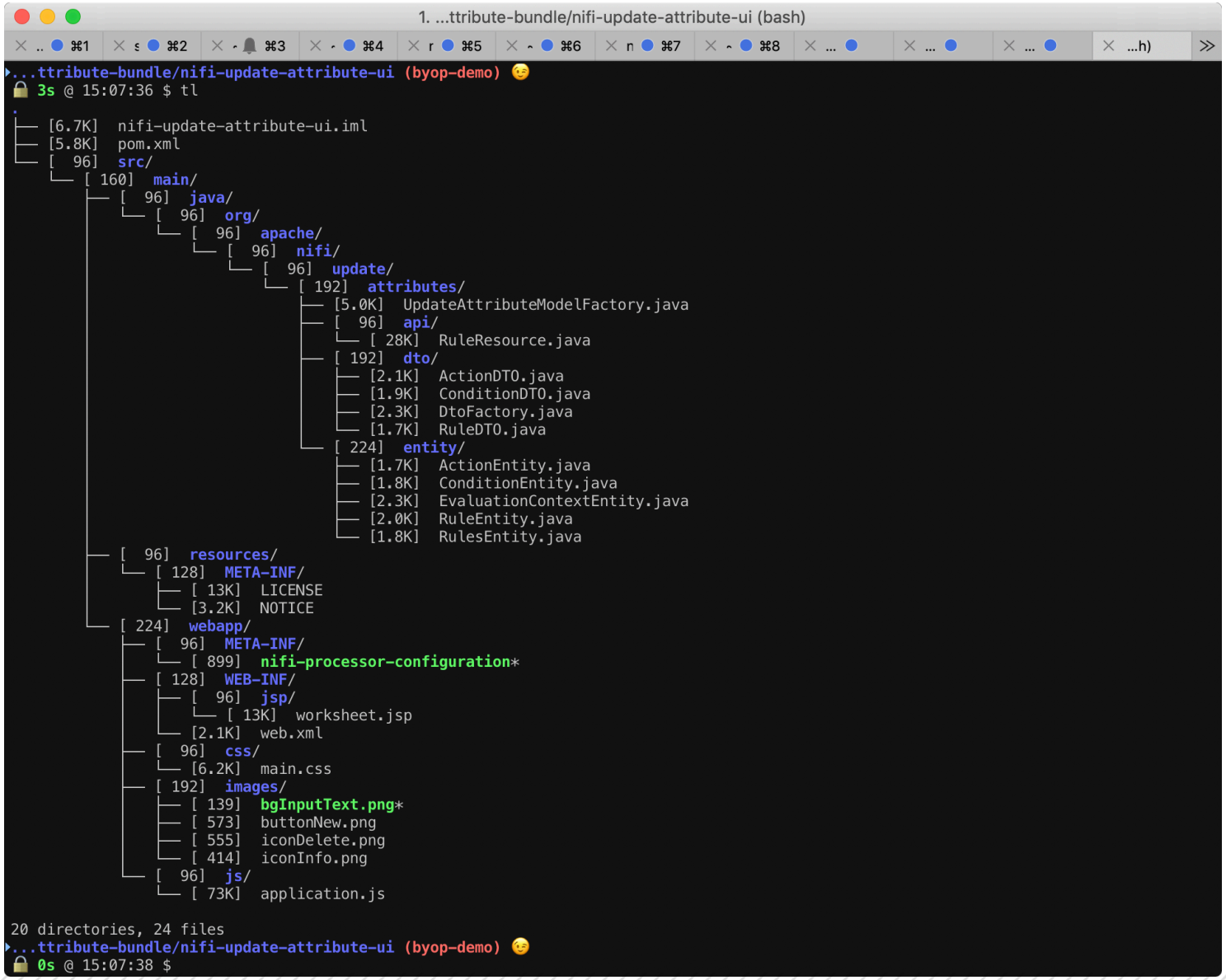
Custom User Interface

- Allows for custom UX
- Validation, feedback, wizards, etc.



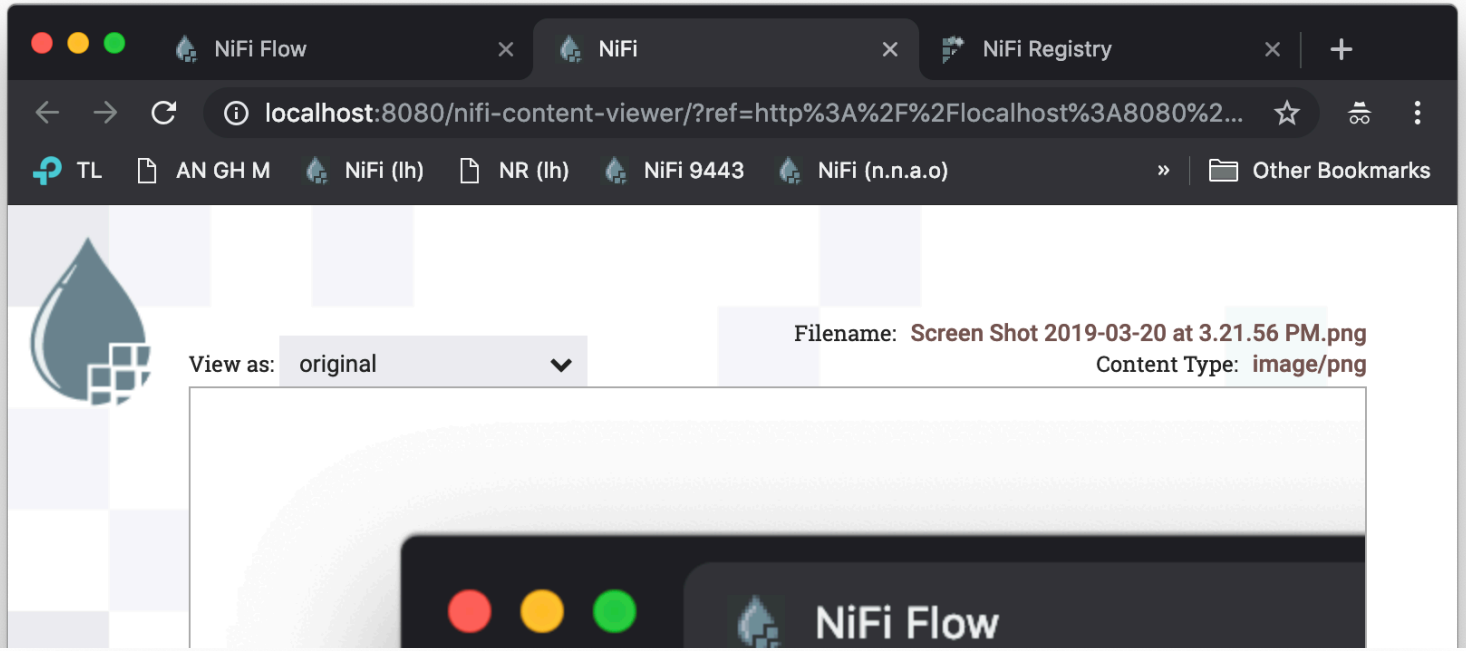
Custom User Interface

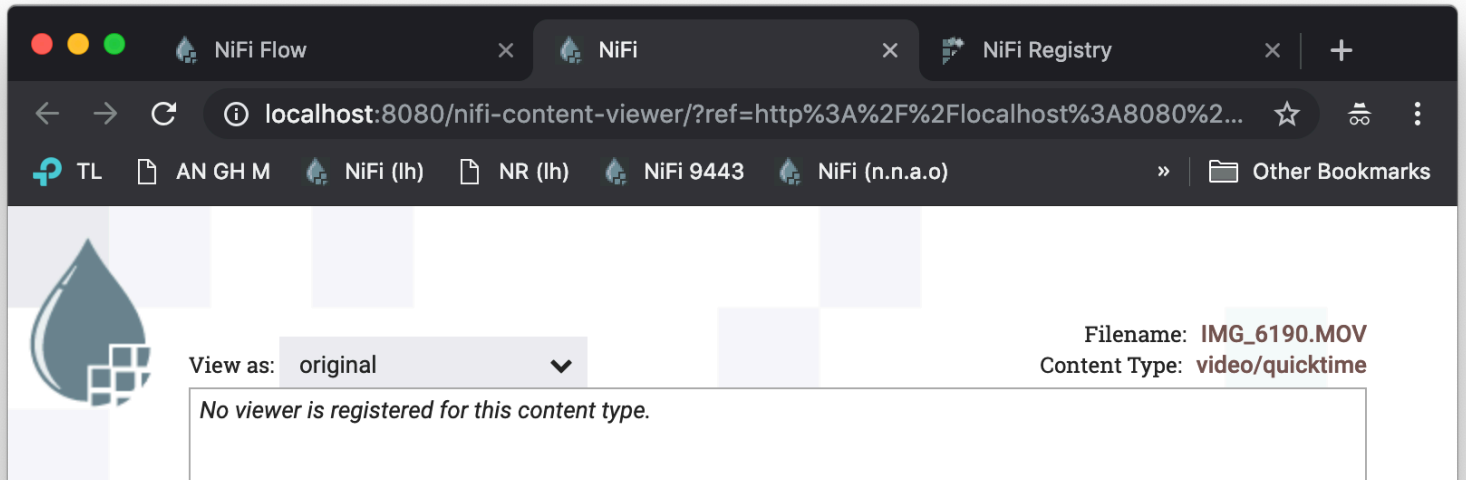
- Code in `nifi-processor-custom-ui`
  - Java classes for objects & logic
  - CSS/JS for presentation



Custom Content Viewer

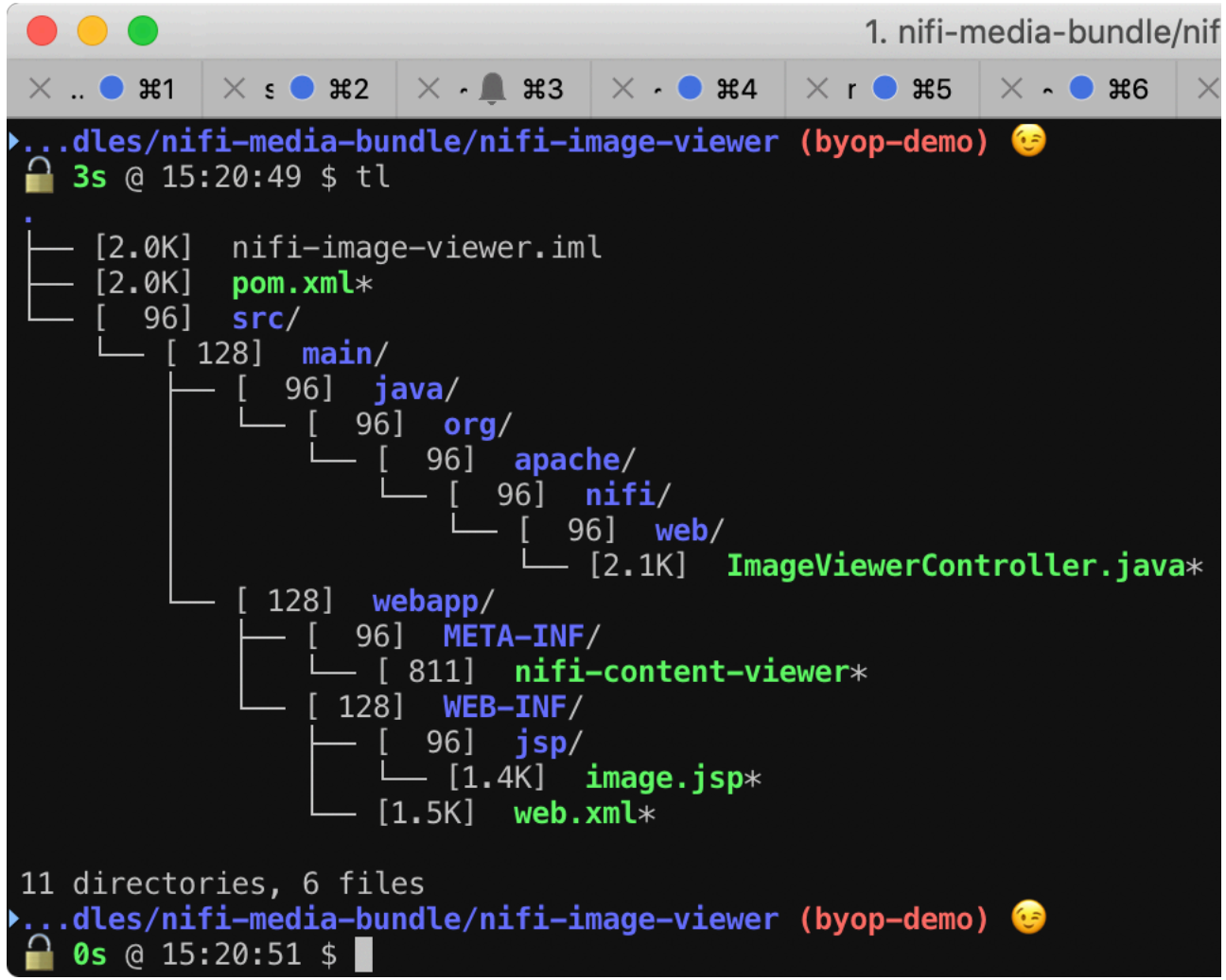
- Special content viewers can be registered
  - Image (default)
  - Video
  - Visualization





Custom Content Viewer

- Code in `nifi-format-viewer`
  - Java classes for objects & logic
  - CSS/JSP for presentation



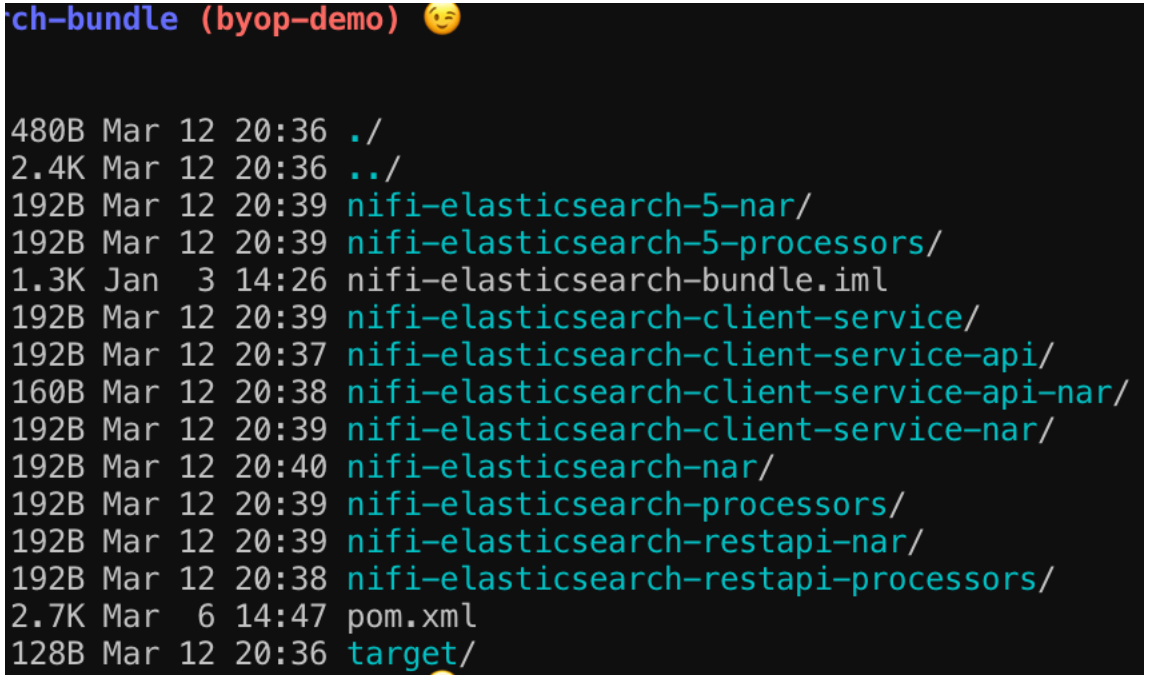
Best practices

ExecuteScript vs. InvokeScriptedProcessor

- ES easier for rapid development
- ISP more performant (should always be used in “production” flows)
  - Allows custom relationships

Module organization

- API module
  - If using controller services; separate the API and implementation into separate modules to allow use & extension
- Can be co-located with processors for simple CS implementations



Module configuration

- Check archetype version
- Check `nifi-api` version
- Inherits from `nifi-nar-bundle` by default
  - If building outside of NiFi, can be removed

Processor structure (con't)

- Initialize relationships and property descriptors in `init()`
  - Large allowable values collections can use builder patterns or `Enum`
- Don't connect to external services in `onScheduled()`
  - Use `onTrigger()`
- Handle dynamic properties
- Custom validation (see `StandardValidators`)

Other processor concerns

- Ensure good documentation
- Handle flowfile content in streaming manner
- Generate proper provenance events for interaction with external systems
  - Send, fetch, etc.
- Define appropriate relationships

Remember to list processors in manifest

- Most common issue when not seeing processor in NiFi
  - `src/main/resources/META-INF/services/`
  - `org.apache.nifi.processor.Processor`

```
org.apache.nifi.processors.standard.ConvertRecord
org.apache.nifi.processors.standard.CountText
org.apache.nifi.processors.standard.CryptographicHashAttribute
org.apache.nifi.processors.standard.CryptographicHashContent
org.apache.nifi.processors.standard.DebugFlow
org.apache.nifi.processors.standard.DetectDuplicate
org.apache.nifi.processors.standard.DistributeLoad
org.apache.nifi.processors.standard.DuplicateFlowFile
org.apache.nifi.processors.standard.EncryptContent
org.apache.nifi.processors.standard.EnforceOrder
org.apache.nifi.processors.standard.EvaluateJsonPath
org.apache.nifi.processors.standard.EvaluateXPath
org.apache.nifi.processors.standard.EvaluateXQuery
org.apache.nifi.processors.standard.ExecuteProcess
org.apache.nifi.processors.standard.ExecuteSQL
org.apache.nifi.processors.standard.ExecuteSQLRecord
org.apache.nifi.processors.standard.ExecuteStreamCommand
org.apache.nifi.processors.standard.ExtractGrok
org.apache.nifi.processors.standard.ExtractText
org.apache.nifi.processors.standard.FetchDistributedMapCache
org.apache.nifi.processors.standard.FetchFile
org.apache.nifi.processors.standard.FetchFTP
org.apache.nifi.processors.standard.FetchSFTP
org.apache.nifi.processors.standard.FlattenJson
org.apache.nifi.processors.standard.ForkRecord
org.apache.nifi.processors.standard.GenerateFlowFile
org.apache.nifi.processors.standard.GenerateTableFetch
org.apache.nifi.processors.standard.GetFile
org.apache.nifi.processors.standard.GetFTP
org.apache.nifi.processors.standard.GetHTTP
org.apache.nifi.processors.standard.GetJMSQueue
org.apache.nifi.processors.standard.GetJMSTopic
org.apache.nifi.processors.standard.GetSFTP
org.apache.nifi.processors.standard.HandleHttpRequest
org.apache.nifi.processors.standard.HandleHttpResponse
org.apache.nifi.processors.standard.HashAttribute
org.apache.nifi.processors.standard.HashContent
org.apache.nifi.processors.standard.IdentifyMimeType
org.apache.nifi.processors.standard.InvokeHTTP
org.apache.nifi.processors.standard.JoltTransformJSON
org.apache.nifi.processors.standard.ListDatabaseTables
org.apache.nifi.processors.standard.ListenHTTP
```

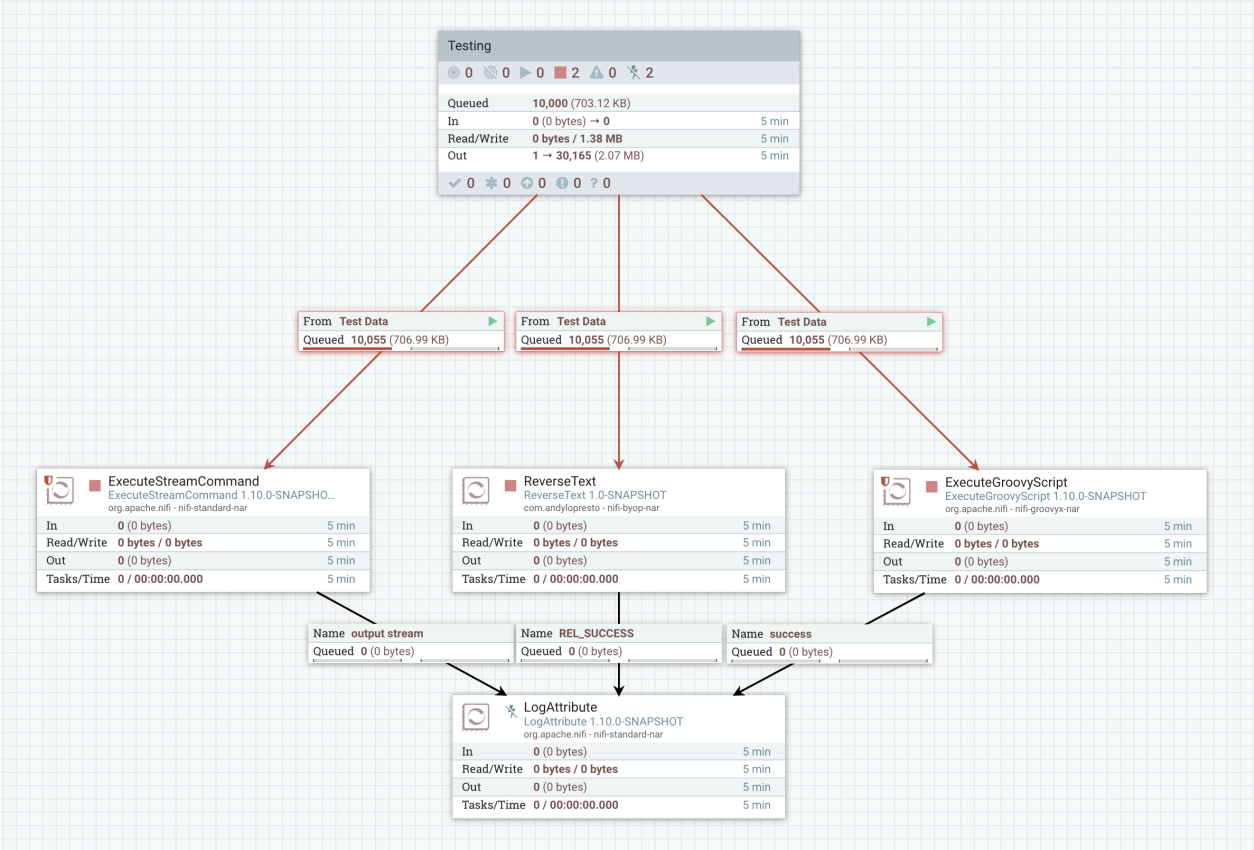
Takeaways

Dynamic vs. compiled processors

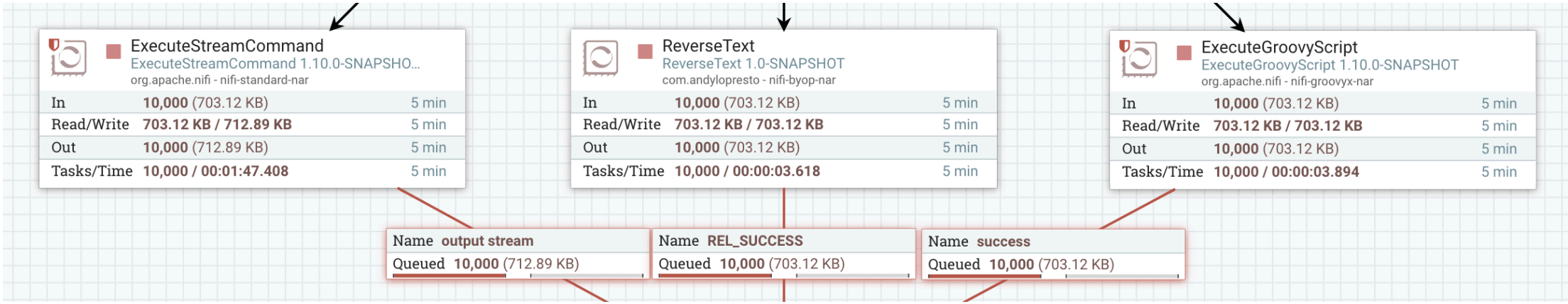
Processor type	Pros	Cons
ExecuteScript/ExecuteGroovyScript	• Very fast development cycle • Easy to use • Source code maintained in version control if stored internally	• Poor performance • Source code not maintained if located on file system • Limited capabilities
InvokeScriptedProcessor	• Fast development cycle • Complex scripting capabilities • Better performance than E(G)S	• Performance still far below compiled processor • Same limitations on code control and interaction with internal framework
Custom Processor	• Full first-class interaction with framework API • Robust testing mechanisms • High performance • Reusability • Versioning • Ease-of-use (drag & drop)	• Slow development cycle • Requires boilerplate code

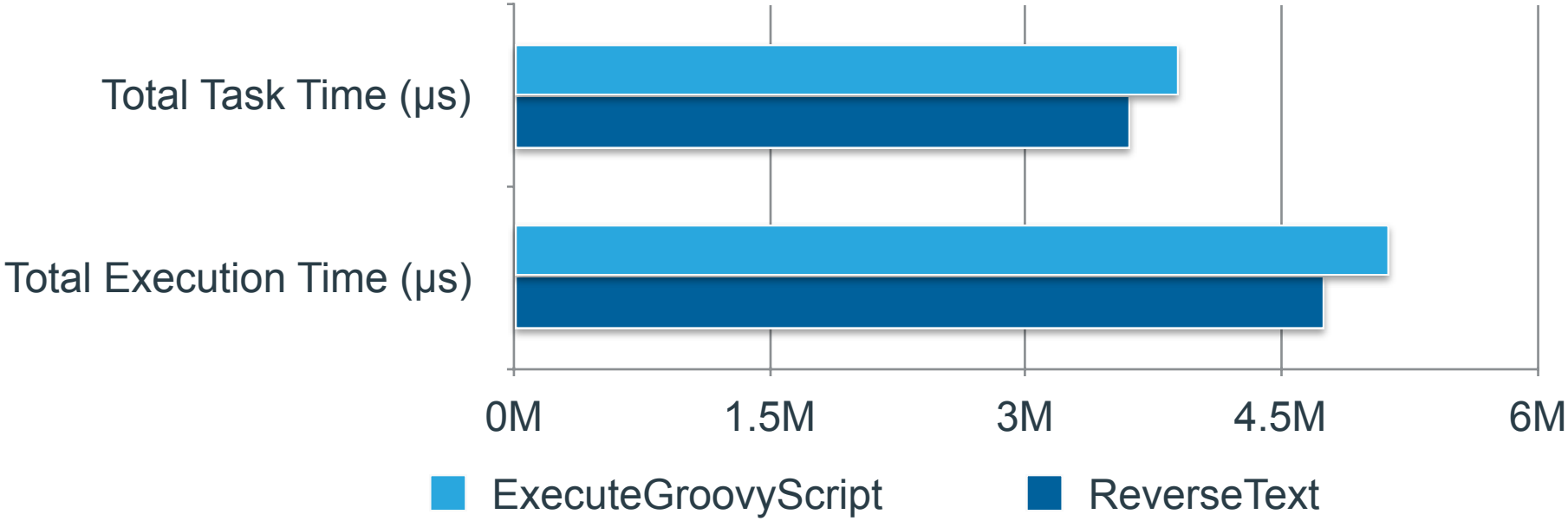
Performance testing

- GenerateFlowFile
- Filled back pressure of queues
- Comparing
  - STDIN -> rev -> STDOUT
  - Custom processor
  - Groovy script

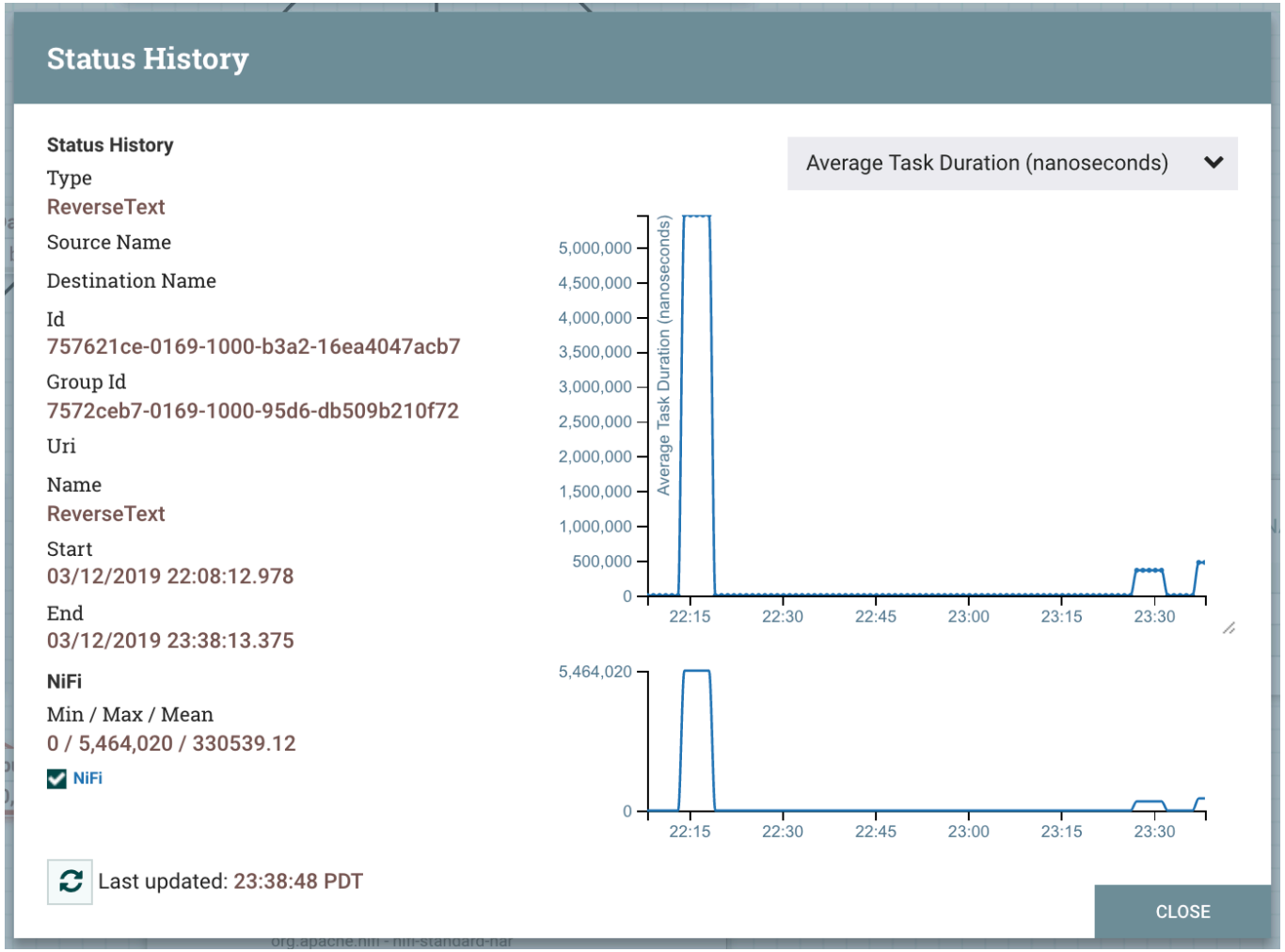


Performance results





Performance results



Notes on performance

- Very simple operation (JVM optimizing)
  - External service calls, loops, etc. will have bigger impact
- Multi-threading / time-series dependent data can influence
- Streaming from file system will impact

Additional resources

Other ideas

- Build custom processor with any JVM-compatible language
  - Community accepts contributions in Java only
- Host custom processor bundles on individual repos
- Extension registry coming soon
  - PRs accepted recently

Community resources

- Apache NiFi Contributor Guide
  - Instructions on cloning repo; building; contributing code; checkstyle guide
- Apache NiFi Developer Guide
  - Processor API; lifecycle; documenting; common patterns; errors; testing
- Walkthroughs
  - Matt Burgess - funnifi.blogspot.com
  - Bryan Bende - bryanbende.com

New Announcements

- NiFi 1.9.1 — 16 Mar 2019 (167+ Jiras)
  - New processors (Kudu, Slack)
  - Hot-loading NARs
  - Security HTTP headers
  - Record processors automatically infer schema
- **MiNiFi C++ 0.6.0 — Voting now…**
- MiNiFi Java 0.5.0 — 7 July 2018
- NiFi Registry 0.3.0 — 25 Sept 2018

Learn more and join us

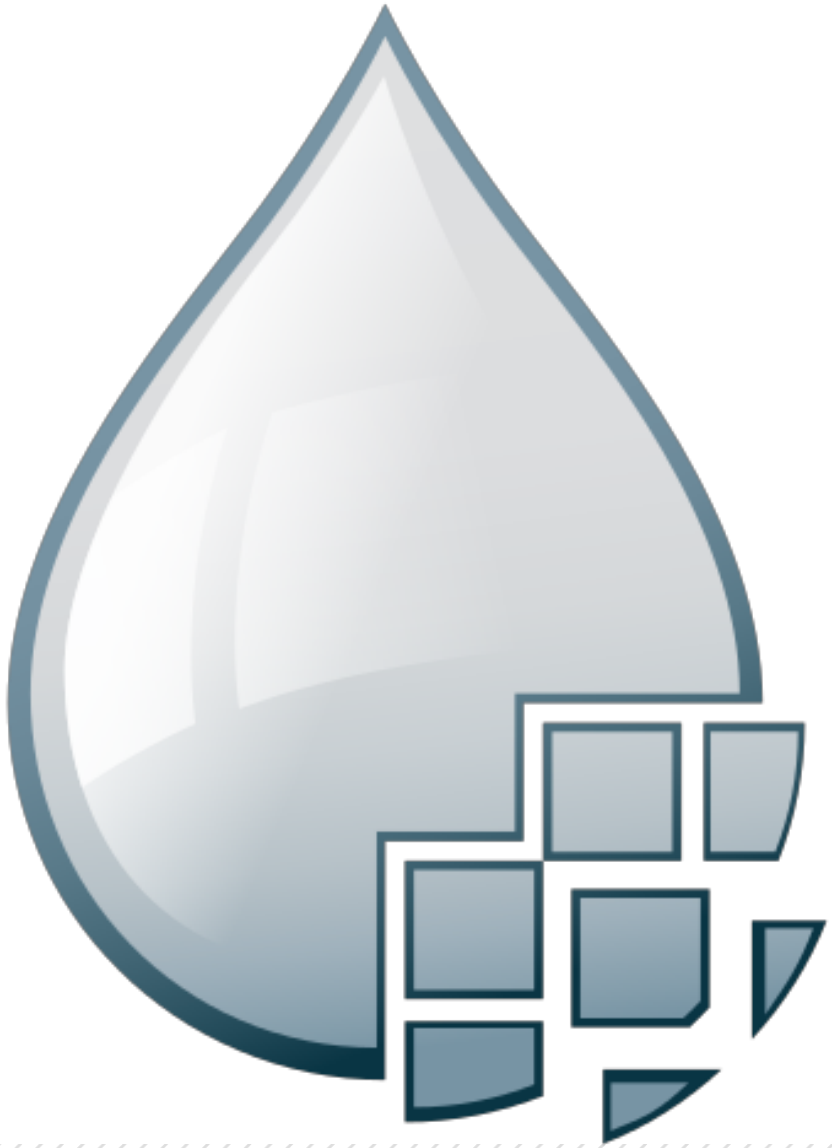
Apache NiFi site
https://nifi.apache.org

Subproject MiNiFi site
https://nifi.apache.org/minifi/

Subscribe to and collaborate at
dev@nifi.apache.org
users@nifi.apache.org

Submit Ideas or Issues
https://issues.apache.org/jira/browse/NIFI

Follow us on Twitter
@apachenifi



NiFi at Dataworks Summit Barcelona

Title	Date		Speakers	Room
Cloudera Data in Motion Meetup - Future of Data Barcelona (free)	Tuesday 3/19	1800 - 2000	Tim Spann, Dan Chaffelson, Andy LoPresto	127-128
Apache NiFi Crash Course	Wednesday 3/20	1400 - 1600	Nathan Gough, Andy LoPresto	111
Data Acquisition Automation for NiFi in a Hybrid Cloud environment – the Path towards DataOps		1450 - 1530	Arda Basar, Ivan Georgiev	129-130
IoT, Streaming, and Dataflow Birds of a Feather		1600 - 1730	Tim Spann, Dan Chaffelson, Purnima Reddy Kuchikulla, Andy LoPresto	129-130
Addressing Challenges with IoT Edge Management	Thursday 3/21	1100 - 1140	Dinesh Chandrasekhar	127-128
Intelligently Collecting Data at the Edge – Intro to Apache MiNiFi		1150 - 1230	Andy LoPresto	127-128
Edge to AI: Analytics from Edge to Cloud with Efficient Movement of Machine Data		1400 - 1440	Tim Spann	129-130
BYOP: Custom Processor Development with Apache NiFi		1600 - 1640	Andy LoPresto	131-132
Apache Deep Learning 201		1650 - 1730	Tim Spann	122-123
Platform for the Research and Analysis of Cybernetic Threats		1650 - 1730	Monica Franceschini	124-125

Questions

THANK YOU

alopresto@cloudera.com | alopresto@apache.org | @yolopey
github.com/alopresto/slides