Draft

ACS Developers Guide

Agile Core Services (ACS) 3

Component of
Consolidated Afloat Networks and Enterprise Services (CANES)

Document Version 0.03 (13)

GBK-61600-000091-v0.03

21 JUN 2018

Prepared by:
SPAWAR Systems Center Pacific
San Diego, CA 92152-5001

Distribution Statement D: *Distribution authorized to Department of Defense and U.S. DoD contractors only for administrative or operational use (31March2017). Other requests must be referred to the Tactical Networks Program Office (PMW 160), Navy Program Executive Office for C4I, or the SPAWAR Office of Congressional and Public Affairs.*

All portions of this document are unclassified.

Draft

ACS Developers Guide

FOR

AGILE CORE SERVICES (ACS) 3

Document Version 0.03 (13)

GBK-61600-000091-v0.03

21 JUN 2018

APPROVALS:

Raymond Barrera, ACS Lead System Engineer, SSC Pacific

Mark Zabriskie, ACS Project Manager, SSC Pacific

Jason Wong, ACS Lead Engineer, PMW 160

Emily Nguyen, ACS Assistant Program Manager, PMW 160

RECORD OF CHANGES

**A - ADDED M - MODIFIED D – DELETED*

VERSION NUMBER	DATE	NUMBER OF FIGURE, TABLE OR PARAGRAPH	AMD*	TITLE OR BRIEF DESCRIPTION	CHANGE REQUEST NUMBER
0.01	23 Mar 2018		A	Initial Version	
0.02	23 May 2018		A,M	Peer Review fixes	
0.03	21 Jun 2018		A	Add sections: Sparks, IntellEx, ELK, OWF, OpenShift, App Best Practices, and Custom NiFi Processor	ACS3EXT-379

TABLE OF CONTENTS

Section **Page**

1. Introduction **1**
1.1 Intended Use and Audience **1**
1.2 Document Overview **1**
1.3 Reference Documents **3**

2. Architecture **4**
2.1 System Overview **4**
2.1.1 ACS Common (ACS-C) Services 5
2.1.2 ACS Data Analytics (ACS-DA) Services 6
2.2 Major Commercial off-the-shelf and Open Source Software **6**

3. Subsystems **8**
3.1 Ingress and Egress Dataflow Management Framework **8**
3.2 Data Pipeline Framework **9**
3.3 Data Processing and Analytic Framework **11**
3.4 Mediation Framework **12**
3.5 Geospatial Data Access Layer and GIS Services **13**
3.6 Platform as a Service **16**
3.7 Data Representation and Persistence Framework **17**
3.8 Data Visualization Framework **18**
3.9 Marketplace **20**
3.10 Security Management **22**
3.11 ACS System Management Framework **25**

4. Components **27**
4.1 NiFi **27**
4.1.1 Overview 27
4.1.2 Interfaces 27
4.1.2.1 Web GUI 27
4.1.2.2 REST Interface 28
4.2 NiFi ACS Extension **28**
4.3 Kafka **29**
4.4 Hadoop **31**
4.4.1 Interfaces 31
4.4.1.1 Web GUI 31
4.4.1.2 Java API 33
4.5 Storm **33**
4.5.1 Overview 33
4.5.2 Interfaces 33
4.5.2.1 Web GUI 33
4.5.2.2 REST API 34
4.5.2.3 JAVA API 35
4.6 Spark **35**
4.6.1 Overview 35
4.6.2 Interfaces 35
4.6.2.1 Web GUI 35
4.6.2.2 Java API 36
4.7 IntellEx **36**

4.7.1 Platform Capabilities 37
4.7.2 System Overview 38
4.7.2.1 Data Collection and Synchronization 38
4.7.2.2 Knowledge Representation and Persistence 39
4.7.2.3 Interest Models 39
4.7.2.4 Data Exposure 40
4.7.3 Interfaces 41
4.7.3.1 Web GUI 41
4.8 DRIF 41
4.9 OpenShift 43
4.9.1 Interfaces 43
4.9.1.1 Web-Based Console 43
4.9.1.2 Command-Line Tool 44
4.9.1.3 Source-to-Image 44
4.9.1.4 Integrated Logging and Metrics 44
4.10 Accumulo 45
4.10.1 Interfaces 45
4.10.1.1 Web GUI 45
4.10.1.2 Java API 46
4.10.1.3 CLI 47
4.11 Solr 47
4.12 Rya 48
4.13 PostgreSQL 49
4.14 GeoMesa 49
4.14.1 Overview 49
4.14.2 Architecture Overview 50
4.14.3 GeoMesa and GeoServer 50
4.14.4 Integration with GeoMesa 51
4.14.5 GeoMesa Command-Line Tools 52
4.14.6 Geo-Indexing Topology 53
4.15 ArcGIS 53
4.15.1 Overview 53
4.15.2 Interfaces 53
4.15.2.1 Admin Web GUI 53
4.15.2.2 Admin CLI 54
4.15.2.3 Portal 54
4.15.2.4 Web Map Service 55
4.15.2.5 Web Feature Service 55
4.15.2.6 Web Coverage Service 56
4.15.2.7 Java API 56
4.15.2.8 JavaScript API 57
4.15.2.9 Python API 57
4.16 Ozone Widget Framework (OWF) 57
4.16.1 Overview 57
4.16.2 Interfaces 59
4.16.2.1 Web GUI 59
4.16.2.2 REST Interface 62
4.17 OpenAM 62
4.17.1 Overview 62
4.17.2 Interfaces 62
4.17.2.1 Administration GUI 62

4.17.2.2 Administration CLI 63
4.17.2.3 APIs 64
4.17.2.4 Policy Agent Interfaces 64
4.17.2.5 Service Interfaces 64
4.18 National Maritime Intelligence Center (NMIC) Centralized Authorization & Authentication Service (NCAAS) 65
4.18.1 Overview 65
4.18.2 Interfaces 65
4.18.2.1 Administration GUI 65
4.18.2.2 Administration CLI 67
4.18.2.3 Administrative Console Interface 67
4.18.2.4 Policy Agent Interfaces 67
4.18.2.5 Service Interfaces 67
4.19 ELK (Elasticsearch, Logstash, Kibana) 69
4.19.1 Logging Overview 69
4.19.2 ELK Logging 70
4.19.2.1 Elasticsearch 70
4.19.2.2 Logstash 70
4.19.2.3 Kibana 71
4.19.2.4 ELK Data Pipeline 71
4.19.3 Beats 72
4.20 Zookeeper 73

5. Data Stores 75
5.1 Data Elements 75
5.2 Data Models 75
5.2.1 Artifact Models 75
5.2.2 Domain Models 76
5.3 Data Stores 76
5.3.1 Artifact Store 76
5.3.2 Linked Graph Store 77
5.3.3 Semi-Structured Data Store 77
5.3.3.1 Time Based Event Store 78
5.3.3.2 Non-time sharded Document/Object Store 78
5.3.4 Semantic Store 79
5.4 Data Stores and Service APIs 80
5.4.1 Cohesive Data and Knowledge Store Strategy 80
5.4.2 Data Store Listing 80
5.4.3 Data Store Comparison 81
5.4.4 API Types 82

6. Cluster Hosts 84
6.1 Ansible Host Names 84
6.2 Web URLs 84

7. DataFlow 86
7.1 Terminology 86
7.2 NiFi Service 86
7.3 NiFi UI 87
7.4 Creating DataFlow 87
7.4.1 GetHTTP Processor 88

7.4.1.1 Create GetHTTP Processor 88
7.4.1.2 Configuring GetHTTP Processor 89
7.4.1.3 Adding SSL Service 90
7.4.2 DPFInputWrapper Processor 94
7.4.3 PublishKafka Processor 95
7.4.4 Connecting Processors 96
7.4.5 Auto-Terminate Relationship 97
7.5 Testing DataFlow 97
7.5.1 Processor Validation 97
7.5.2 Starting and Stopping Processors 97
7.5.3 Data Provenance 98
7.5.4 Lineage Graph 99
7.6 REST API 100
7.6.1 System Diagnostics 100
7.6.2 Flow Status 102
7.6.3 Querying Processor 102

8. Kafka Messaging 104
8.1 Kafka Service 104
8.2 Kafka Topic 104
8.2.1 Listing Topics 104
8.2.2 Viewing a Topic 105
8.2.3 Creating a Topic 105

9. Storm Topology 106
9.1 Overview 106
9.1.1 Storm Architecture 106
9.1.2 Topologies 107
9.2 Storm UI 109
9.2.1 Storm UI Landing Page 109
9.2.2 Storm UI Topology Home 111

10. Artifact and Domain Model 115
10.1 Artifact Model 115
10.1.1 Artifact Model File Requirements 115
10.1.2 Artifact Model File 115
10.1.3 Artifact Model Lookup File 119
10.1.4 Artifact Model Repository File 119
10.1.5 Artifact Model Type Discovery File 119
10.2 Domain Model 120
10.2.1 Domain Model File Requirements 120
10.2.2 Domain Model File 120
10.2.3 Domain Model Concept Definition file 122
10.2.4 Domain Model Predicates Definition file 122
10.2.5 Domain Model Repository file 123

11. Hadoop / HDFS 124
11.1 Hadoop Services 124
11.2 HDFS Overview 124
11.2.1 NameNode and DataNodes 124
11.2.2 Web Interface 124

11.2.3 Shell Commands ... 124
11.2.4 Hadoop Configuration Settings ... 126

12. Structured Object Datastore ... 131
12.1 Overview ... 131
12.2 Data Structure ... 131
12.2.1 Key Structure ... 131
12.2.1.1 Key URI Scheme ... 131
12.2.2 Property Structure ... 132
12.2.3 Attribute-Based Access Control (Visibility) ... 133
12.2.4 Ancestor Paths ... 133
12.3 Table Structures ... 134
12.3.1 The so_manager Table ... 134
12.3.2 Instance Management Table ... 134
12.3.3 Core Table ... 134
12.3.3.1 Key Encoding ... 135
12.3.3.2 Property Encoding ... 135
12.3.4 Indexing ... 136
12.3.4.1 Kind Index ... 136
12.3.4.2 Ascending Reverse Index ... 137
12.3.4.3 Descending Reverse Index ... 137
12.3.4.4 Composite Index ... 137
12.3.5 Index Definitions ... 137

13. Querying Structured Object with SOQL ... 138
13.1 Overview ... 138
13.2 SOQL Introduction ... 138
13.3 SOQL Section to SQL Section Mapping ... 139
13.4 Constraints ... 140
13.4.1 WHERE clause ... 140
13.4.2 Namespace Constraints (IN clause) ... 140
13.4.3 Key Constraints (WITH clause) ... 140
13.4.3.1 Ancestor Key Constraints (WITH ANCESTOR clause) ... 141
13.5 Query Scope ... 142
13.5.1 ANCESTORS Scope ... 143
13.5.2 PARENTS Scope ... 143
13.5.3 SELF Scope ... 144
13.5.4 CHILDREN Scope ... 144
13.5.5 DESCENDANTS Scope ... 144
13.6 Using SOQL ... 144
13.6.1 Web UI ... 144
13.6.2 Command Line Interface ... 145

14. ArcGIS GeoEvent Server ... 147
14.1 GeoEvent Definitions ... 147
14.2 GeoEvent Tags ... 148
14.3 GeoEvent Inputs ... 149
14.4 GeoEvent Outputs ... 150
14.5 GeoEvent Services ... 154
14.5.1 GeoEvent Filters ... 155
14.5.1.1 Attribute Filters ... 156

14.5.1.2 Spatial Filters 156
14.5.1.3 GeoEvent Property Filters 156
14.5.2 Publishing GeoEvent Service 156
14.5.3 Monitoring GeoEvent Server 156
14.5.3.1 Setting Rate Thresholds 157
14.5.3.2 Max Rate and Time Since Last 157
14.5.3.3 Rate (over the last 5 minutes) 157
14.5.3.4 Average rate (over last 5 minutes) 158
14.5.4 Starting and stopping inputs, outputs, and GeoEvent Services 158
14.6 Portal for ArcGIS 158

15. DG Analytics with Spark 160
15.1 Analytics with Spark 160
15.2 Introduction 160
15.3 Creating a simple Spark job 160
15.4 Creating a Spark job for processing flight data off a Kafka topic 163
15.5 Troubleshooting 166

16. DG IntellEx Alerts 169
16.1 Objective 169
16.2 Preparation 169
16.3 NiFi Setup 169
16.4 IntellEx Setup 170
16.4.1 Workspace 170
16.4.2 Data Source 170
16.4.3 Fact Type 172
16.4.3.1 Aircraft Fact Type 172
16.4.3.2 Flight Fact Type 173
16.4.4 Data Handler 175
16.4.4.1 JSON Parsing Script 175
16.4.4.2 Data Handler 177
16.4.5 Interest Model Rules 180
16.4.5.1 Rule Type 180
16.4.5.2 Rule Package 180
16.4.6 Interest Model Product (Alert) Generation 181
16.4.6.1 Alert Template 181
16.4.6.2 forEach 182
16.4.6.3 exists and notExists 182
16.4.7 Alert Distribution Policy 183
16.4.7.1 Channels 183
16.4.7.2 Decimation 183
16.4.8 Interest Model 184
16.5 Generating Alerts 185

17. ELK Logging 187
17.1 Overview 187
17.2 Configuring Beats 187
17.2.1 Configuring Filebeat 187
17.2.1.1 Optional Configuration 188
17.2.2 Configuring Topbeat 190
17.2.3 Configuring Packetbeat 191
17.3 Kibana 192

17.3.1 User Interface 193
17.3.1.1 Discover Section 193
17.3.1.2 Visualize Section 193
17.3.1.3 Dashboard Section 193
17.3.2 How to View Infrastructure Metrics with Topbeat 193
17.3.3 How to View Network Analytics with Packetbeat 195
17.3.4 How to View Log Data with Filebeat 197

18. Organizing Web Applications with OWF 202
18.1 Overview 202
18.2 OWF In ACS 202
18.3 OWF UI Components 203
18.3.1 OWF Toolbar 203
18.3.2 Dropdown User Menu 204
18.4 OWF App Components 204
18.4.1 App Component Types 204
18.4.1.1 Singleton App Components 204
18.4.1.2 Background App Components 205
18.4.2 App Component Menu 205
18.4.3 Create an App Component 205
18.4.3.1 Link to Another Web App 205
18.5 OWF App 208
18.5.1 My Apps Menu 208
18.5.2 Opening an App 208
18.5.3 Create an App 208

19. OpenShift 212
19.1 Architecture 212
19.2 Managing Users 212
19.2.1 Adding a User 212
19.2.2 Viewing User and Identity Lists 212
19.2.3 Managing User and Group Labels 213
19.2.4 Deleting a User 213
19.3 Creating and Installing Images 214
19.3.1 Installation 214
19.3.2 Creating the project and image rpm 214
19.3.3 Ansible Container basics 215
19.3.3.1 container.yml 215
19.3.3.2 Example deployment: 217
19.4 Basic CLI Operations 217
19.4.1 Test the default service and login 217
19.4.2 Using project "default" 218
19.4.3 List Project 219
19.4.4 Validate user can login 221
19.4.5 Validate docker login 221
19.4.6 Create New Project 221
19.5 Web UI 222
19.6 Hello World! 223

20. App Development Best Practices 225
20.1 Twelve-Factor Apps 225

20.1.1 Codebase ... 225
20.1.2 Dependencies ... 225
20.1.3 Config ... 225
20.1.4 Backing Services ... 226
20.1.5 Build, Release, Run ... 226
20.1.6 Processes ... 226
20.1.7 Port Binding ... 227
20.1.8 Concurrency ... 227
20.1.9 Disposability ... 227
20.1.10 Development/Production Parity ... 227
20.1.11 Logs ... 228
20.1.12 Admin Process ... 228
20.2 Secrets Management ... 228
20.3 Compute and Storage Limitation for Apps ... 229
20.4 Trusted Base images ... 229

21. Performance Tuning ... 230
21.1 Accumulo ... 230
21.1.1 Pre-splitting ... 230
21.1.2 Change table.file.compress.type to Snappy ... 231
21.1.3 Give Accumulo More Memory ... 231
21.1.4 Create Additional Accumulo Tablet Servers and HDFS Datanodes ... 231
21.1.5 Results from Accumulo Tuning ... 232

22. Tools ... 233
22.1 Test Harness ... 233
22.1.1 Purpose ... 233
22.1.2 Ingest TRIG Files ... 233
22.1.3 Producing AIS-OTH Data ... 233
22.1.3.1 Example - 100 messages per second ... 234
22.1.3.2 Example - 1500 messages per second ... 235
22.1.4 Troubleshooting ... 235
22.1.4.1 Data Not Going to Kafka ... 235
22.2 DPF Utility ... 236
22.2.1 Background ... 236
22.2.2 Using the DPF Utility to Register Concepts ... 236
22.2.3 Known Limitations ... 236

23. Troubleshooting ... 237
23.1 Clear Accumulo Tables ... 237
23.1.1 Step-by-Step Guide ... 237
23.2 Clear Kafka ... 237
23.2.1 Step-by-Step Guide ... 237
23.3 Openshift ... 237
23.3.1 acs-extras is needed for the install/Flannel not found ... 237

Appendix A Flight Data Demo ... 239
Introduction ... 239
Procedures ... 239

Appendix B ACS Twitter Demo ... 268

Overview **268**
Import Twitter data into ACS **268**
Configure data workflow **269**
HDFS processor 270
Data Storage Directory in HDFS 271
Verify Twitter Data is Ingested Locally and on HDFS **272**

Appendix C Create a Custom NiFi Processor **274**
Overview **274**
Prerequisites **274**
Json Processor Build **274**
Use Maven Archetype for NiFi 274
Pom.xml 275
Point NiFi to Correct Class 276
Configure Custom Processor Java File 277
Abstract Processor Class 279
Tag and Capability Documentation 279
Documenting Properties 279
Documenting Relationships 280
Property Validation 280
Performing Work When Triggered 280
Configure Test Processor Java File 281
Build NAR File with Maven 283
Deploy NAR File 283
Verify the Custom Processor is Loaded into NiFi 284
Apply Custom Processor to Data Flow 284

Appendix D Acronyms **290**

LIST OF FIGURES

Figure **Page**

Figure 1 ACS Subsystem 4
Figure 2 ACS System Management Framework Components 7
Figure 3 Data Ingest and Processing Framework Block Diagram 9
Figure 4 Data Pipeline Framework Block Diagram 10
Figure 5 Data Processing and Analytics Framework Block Diagram 12
Figure 6 Mediation Framework Block Diagram 12
Figure 7 Geospatial Data Access Layer and GIS Services Block Diagram 14
Figure 8 Platform as a Service Block Diagram 16
Figure 9 Data Representation and Persistence Framework Block Diagram 18
Figure 10 Data Visualization Framework Block Diagram 19
Figure 11 Marketplace Block Diagram 21
Figure 12 Security Management Block Diagram 24
Figure 13 ACS System Management Framework Block Diagram 25
Figure 14 Ingress and Egress DataFlow Management Framework 27
Figure 15 NiFi GUI 28
Figure 16 Kafka Interface 30
Figure 17 Hadoop GUI Datanodes 32
Figure 18 Storm Detailed Statistics 34

Figure 19 Spark Web GUI 36
Figure 20 IntellEx Functional Architecture 38
Figure 21 Generalized Data Ingest Using Workflows 39
Figure 22 Decoupling Analytics from Source 39
Figure 23 Interest Models - Data Flow 40
Figure 24 Alert Generation Process 40
Figure 25 IntellEx ConfigUI 41
Figure 26 Organization of the DRIF Cloud 42
Figure 27 DRIF RDF Transformations 42
Figure 28 OpenShift Web Console 43
Figure 29 Logging and Metrics 45
Figure 30 Accumulo Web GUI 46
Figure 31 Solr Web GUI 48
Figure 32 GeoMesa Architecture 50
Figure 33 GeoServer 51
Figure 34 Possible Ingest Architecture 51
Figure 35 Possible Query Architecture 52
Figure 36 ArcGIS Admin Web GUI 54
Figure 37 Ozone Widget Framework 58
Figure 38 Ozone Widget Framework Component Diagram 59
Figure 39 OWF Web GUI 60
Figure 40 OWF User Menu 60
Figure 41 OWF Administration Tools 61
Figure 42 OWF Apps and Pages 61
Figure 43 OpenAM Administration GUI 63
Figure 44 OpenAM Policy Agent Request/Response 64
Figure 45 Open AM Services 65
Figure 46 NCAAS Administration GUI - Managing Services 66
Figure 47 NCAAS Administration GUI - Editing Policy 66
Figure 48 ELK Data Pipeline 72
Figure 49 Beats 73
Figure 50 ZooKeeper Server of NiFi Cluster 74
Figure 51 Cohesive Data and Knowledge Store Strategy 80
Figure 52 DataFlow NiFi Canvas 87
Figure 53 Data Flow Example 87
Figure 54 Add GetHTTP Processor 89
Figure 55 Configure GetHTTP Processor 90
Figure 56 Create New SSL Service 91
Figure 57 Add Standard SSL Context Service 92
Figure 58 Edit SSL Context Service 93
Figure 59 Enable Controller Service 94
Figure 60 Create DPFInputWrapper Processor 95
Figure 61 Create PublishKafka Processor 96
Figure 62 Auto-Terminate Example DataFlow 97
Figure 63 Start/Stop Example DataFlow 98
Figure 64 Data Provenance 99
Figure 65 Data Lineage 100
Figure 66 Storm Cluster 107
Figure 67 Data Flow through a typical Storm topology 108
Figure 68 Worker Process 109
Figure 69 Cluster Summary 109

Figure 70 Topology Summary 110
Figure 71 Supervisor Summary 110
Figure 72 Storm UI Topology Summary 111
Figure 73 Topology Actions 111
Figure 74 Topology Stats 112
Figure 75 Spouts All Time 112
Figure 76 Bolts All Time 113
Figure 77 Topology Visualization 114
Figure 78 Storm UI Topology Summary 128
Figure 79 Storm Topology Kill Button 128
Figure 80 Storm Kill Topology 128
Figure 81 Data Structure 131
Figure 82 Key Structure 131
Figure 83 Key URI Scheme 132
Figure 84 Property Structure 133
Figure 85 Instance Management Table 134
Figure 86 Core Table 135
Figure 87 Property Encoding 135
Figure 88 Property Component Description 135
Figure 89 Property Type Mapping 136
Figure 90 Kind Index 136
Figure 91 Ascending Reverse Index 137
Figure 92 Descending Reverse Index 137
Figure 93 Composite Index 137
Figure 94 Structured Object Query Language Interface 145
Figure 95 GeoEvent Definitions - demo_flight_geo 147
Figure 96 GeoEvent Definitions 148
Figure 97 GeoEvent Tags 149
Figure 98 GeoEvent Inputs 150
Figure 99 GeoEvent Outputs 151
Figure 100 GeoEvent Outputs - bds-out-demo-flight-geo 151
Figure 101 Data Source - demo_flight_geo 152
Figure 102 GeoEvent Outputs - stream-service-out-demo-flight-geo 153
Figure 103 Stream Service Options 153
Figure 104 Stream Service Options Expanded 154
Figure 105 GeoEvent Services 155
Figure 106 GeoEvent Monitor 157
Figure 107 ArcGIS Portal 158
Figure 108 ArcGIS Portal - demo_flight_geo 159
Figure 109 Spark History UI 166
Figure 110 Spark Master UI 166
Figure 111 IntellEx Alert Data Flow 169
Figure 112 Configure NiFi PostHTTP Processor 170
Figure 113 Configure IntellEx Source 171
Figure 114 Configure Aircraft Fact Type 173
Figure 115 Configure Flight Fact Type 174
Figure 116 Flight Data Workflow 179
Figure 117 Low Altitude Flights Alert 186
Figure 118 Kibana Discover Page 194
Figure 119 Kibana Dashboard 194
Figure 120 Load Saved Dashboard 194

Figure 121 Topbeat Dashboard 195
Figure 122 Topbeat Metrics 195
Figure 123 Kibana Discover Page 196
Figure 124 Kibana Dashboard 196
Figure 125 Load Saved Dashboard 196
Figure 126 Load the Packetbeat dashboard 197
Figure 127 Packetbeat Metrics 197
Figure 128 Filebeat Logs 198
Figure 129 Sample Log Information 199
Figure 130 Expanded Log Entries 199
Figure 131 Log Entries Captured by Filebeat 200
Figure 132 Filtered Audit Log Entries 200
Figure 133 Failed User Login Attempts 201
Figure 134 Add or Remove Fields 201
Figure 135 OWF - Initial Screen 203
Figure 136 OWF - Initial Screen 203
Figure 137 OWF Dropdown User Menu 204
Figure 138 OWF App Components Carousel 205
Figure 139 OWF My Apps 206
Figure 140 OWF App Components Manager 206
Figure 141 OWF App Components Editor 207
Figure 142 OWF App Components Carousel 207
Figure 143 OWF Create New App 209
Figure 144 OWF Create New App 210
Figure 145 OWF Create New App 211
Figure 146 OpenShift Logon 222
Figure 147 OpenShift Project 223
Figure 148 OpenShift Project 224
Figure 149 Accumulo Tablet Server Status 230
Figure 150 Create a New Processor 239
Figure 151 Workspace 240
Figure 152 GetHTTP Processor 240
Figure 153 Configure 241
Figure 154 Set GetHTTP's URL 241
Figure 155 SSL Context Service 242
Figure 156 Create New Service 242
Figure 157 Context Service's Value Field 243
Figure 158 Edit 243
Figure 159 Setup Truststore 244
Figure 160 Enable the SSL Context Service 244
Figure 161 Enable Controller Service 245
Figure 162 New Processor 245
Figure 163 Configure DPFInputWrapper 246
Figure 164 Ingest Data 247
Figure 165 ArcGIS GeoEvent Manager 261
Figure 166 Kafka-Reader.dpf spout 264
Figure 167 GeoEvent services, inputs, and outputs 265
Figure 168 Kafka-Reader.dpf spout, Processing bolt, and GeoSOPersistenceBolt 266
Figure 169 Phoenix, AZ Airport 267
Figure 170 NiFi Flow 268
Figure 171 Import into NiFi 269

Figure 172 Flow Diagram 269
Figure 173 Twitter post instances 270
Figure 174 Configure Processor 271
Figure 175 Verify Twitter data 272
Figure 176 Start NiFi Processors 272

LIST OF TABLES

Table **Page**

Table 1 NiFi ACS Extension 29
Table 2 NCAAS Service Interfaces - REST Services 68
Table 3 NCAAS Service Interfaces - SOAP Services 68
Table 4 Data Store Listing 81
Table 5 Data Store Comparison 81
Table 6 API Types 83
Table 7 ACS URLs 84
Table 8 Component Description 108
Table 9 HDFS Fields of Interest 127
Table 10 SOQL Section Mapping 139
Table 11 SOQL Literals 139
Table 12 Query Scope 142
Table 13 OWF - Tested Browsers 202
Table 14 Results From Accumulo Tuning 232

1. INTRODUCTION

Big data are data sets that are so voluminous and complex that traditional data-processing application software are inadequate to deal with them. ACS was designed to handle big data challenges including capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source. Organizing all of this data in order to enable real-time analysis and coordination among many different entities both local and remote. Information that is shared may include positional coordinates, photos, videos, messages, information updates, etc.

ACS is a sub-system of the Consolidated Afloat Networks and Enterprise Services (CANES) program and is defined by the need to provision loosely-coupled systems and capabilities to the warfighter, which can be reused, discovered, and linked to build mission threads. ACS provides the warfighter with common enterprise solutions to improve information sharing and combat effectiveness. The suite of capabilities facilitates information access and exchange to allow dynamic collaboration and accelerated decision-making amid unprecedented quantities of operational data. ACS 3 provides a set of common software infrastructure services for common core services, data ingest, data representation, data storage, data indexing, data access, and framework to support analytics and applications leading to the deployment of data analytics and applications capable of performing in Denied-Disconnected, Intermittent, Low (D-DIL) bandwidth environments across distributed Naval Forces afloat, and where appropriate, ashore.

1.1 INTENDED USE AND AUDIENCE

This Developer's Guide is a programming guide for application developers. It is intended to provide sufficient guidance to allow application developers to:

- Identify resources used in ACS
- Create new data models
- Ingest data into models
- Perform metadata search and content retrieval

This guide assumes that readers possess a working knowledge of:

- Java
- XML
- J2EE Architecture and Technologies
- Web Services Standards, Interfaces, and Technologies

This guide assumes that a working ACS system is ready for use. Refer to Installation and Setup Guide for more information on how to install and set up an ACS system.

1.2 DOCUMENT OVERVIEW

The main sections of this Developer's Guide are organized as follows:

- **Section 1 Introduction** - Provides background and overview of material for this document.

- **Section 2 Architecture** - Provides an overview of ACS3 components architecture.
- **Section 3 Subsystems** - Provides an overview of ACS3 subsystems.
- **Section 4 Components** - Describes ACS components.
- **Section 5 Data Stores** - Provides an overview of data stores, models and how to create one.
- **Section 6 Cluster Hosts** - Provides an overview of the host naming convention.
- **Section 7 DataFlow** - Describes how to use NiFi.
- **Section 8 Kafka Messaging** - Describes what Kafka does.
- **Section 9 Storm Topology** - Provides a view of streaming processing computation framework.
- **Section 10 Artifact and Domain Model** - Describes what Artifact Domain Models are.
- **Section 11 Hadoop / HDFS** - Describes the Hadoop Distributed File System.
- **Section 12 Structured Object Datastore** - Describes the Structured Object Datastore.
- **Section 13 Querying Structured Object with SOQL** - SQL-inspired language for performing queries on a Structured Object datastore.
- **Section 14 ArcGIS GeoEvent Server** - Describes mapping functions in ACS.
- **Section 15 DG Analytics with Spark** - Describes a Spark example.
- **Section 16 DG EntellEx Alerts** - Describes an IntellEx Alerts example.
- **Section 17 ELK Logging -** Describes Beats configuration and Kibana interface.
- **Section 18 Organizing Web Applications with OWF -** Describe OWF application organization.
- **Section 19 OpenShift -** Describes OpenShift common usage.
- **Section 20 App Development Best Practices -** Describes twelve-factor methodology for build application services.
- **Section 21 Performance Tuning** - Provides tuning guidelines to improve data ingest performance.
- **Section 22 Tools** - Provides an overview of tools used in ACS.
- **Section 23 Troubleshooting** - Provides a common list of troubleshooting commands.
- **Appendix A Flight Data Demo** - Provides an overview of the Flight Data Demo Steps.
- **Appendix B ACS Twitter Demo** - Provides an overview of the ACS Twitter Demo.
- **Appendix C Create a Custom NiFi Processor** - Provides an overview of how to create a custom NiFi processor.
- **Appendix D Acronyms** - Provides an acronym list.

1.3 REFERENCE DOCUMENTS

Document	Title
GBK-61600-000078-v0.09	ACS Installation Guide
GBK-61600-000092-v0.01	System Administration Guide
	Software Configuration Item Documents
GBK-61600-00083-v0.10	Software Version Description

2. ARCHITECTURE

2.1 SYSTEM OVERVIEW

ACS 3 services shall be modular to allow installation of the applicable ACS services needed to support ACS consumer applications to carry out their mission requirements on their installed Navy platforms without bringing unused ACS components. Beginning with CANES ACS 3, ACS is divided into two major modular component requirement groupings: ACS Common Services (ACS-C) and ACS Data Analytics Services (ACS-DA). ACS provides a collection of Computer Software Configuration Items (CSCIs) to support data analytic and common service functionality as described in the following figure and the text that follows.

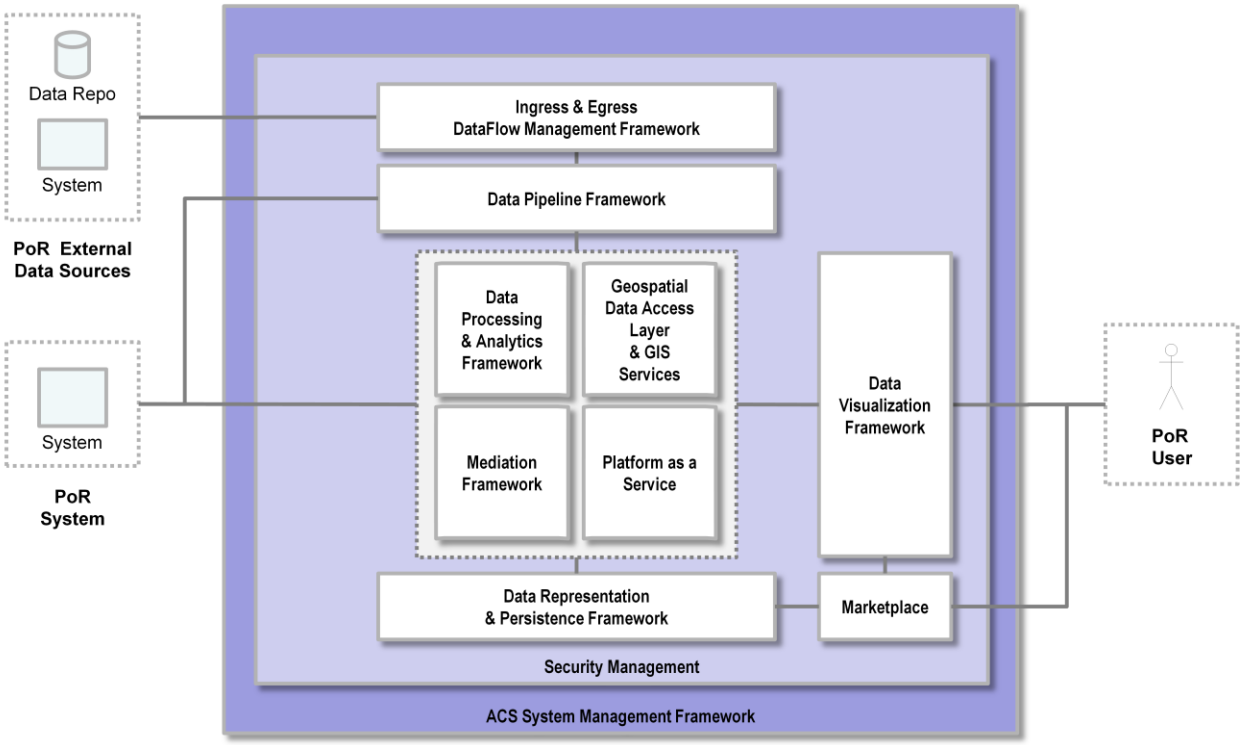


Figure 1 ACS Subsystem

- The Ingress and Egress DataFlow Management Framework CSCI supports automation and management of DataFlows (automated and managed flow of information between systems) into and out of ACS.
- The Data Pipeline Framework CSCI provides a scalable and reliable data pipeline enabling access and transfer of data records between large numbers of data producers and consumers. The data pipeline enables a high throughput of data organized by Topic (a stream of related records).
- The Data Processing and Analytics Framework CSCI enables rapid and efficient Big Data (in volume, velocity, and variety) processing, analytics, and queries across a shared data repository.

- The Geospatial Data Access Layer & GIS Services CSCI provides common geospatial data and base maps as well as the ability to store, process, and analyze geographic data.
- The Mediation Framework CSCI enables an enterprise service bus that supports protocol conversion, data transformation, messaging, and enterprise service orchestration.
- The Platform as a Service CSCI supports hosting and management of server-side applications for use by ACS and ACS customers leveraging virtualization to maximize resource utilization, provide necessary isolation of hosted applications, and to minimize required administration and maintenance.
- The Data Representation and Persistence Framework CSCI provides a means to persist and query large volumes of data in an efficient, reliable and secure means. The framework provides a clustered data store supporting Big Data needs, as well as traditional Relational Database Management Systems (RDBMS) storage.
- The Marketplace CSCI provides user interfaces (UIs) and services for discovery, distribution, installation, and updating of Widgets (dynamic, lightweight, configurable, and composable, web-based user interface components) and Widget supporting services deployed and utilized within ACS.
- The Data Visualization Framework CSCI provides a mechanism to host, organize, access, and share Widgets.
- The Security Management CSCI provides centralized services for use by POR applications to authenticate and authorize users and systems.
- The ACS System Management Framework CSCI provides automated installation, configuration, centralized logging, auditing and monitoring for ACS and ACS-hosted software.

2.1.1 ACS Common (ACS-C) Services

The CANES ACS-C component requirements of ACS 3 carry forward the existing ACS 2.x requirements in the current CANES baseline. CANES ACS-C services requirements include the following:

- Machine-to-machine messaging
- Data mediation
- Discovery
- Hosting environment for containerized applications and container-based application support services (hosting component of Platform as a Service [PaaS])
- Web security (Single Sign-On)
- Application/widget storefront
- Geospatial data access layer

2.1.2 ACS Data Analytics (ACS-DA) Services

ACS-DA shall provide applications with common, shared enabling capabilities to collect, analyze, visualize, and share information. ACS-DA shall provide ACS consumer applications the capability for ingesting and persisting large data sets, accessing that data, building analytics, and visualizing the results by authorized users in secure environments. ACS-DA supports scalable, distributed computing, and meets its consumer application needs in the distributed Naval Forces Afloat working over Denied-Disconnected, Intermittent, Low (D-DIL) environments and where appropriate, ashore. The CANES ACS-DA requirements in ACS 3 include the provision of a common shared Data Analytics Framework with which ACS-DA consumer applications can integrate. The CANES ACS-DA Services Framework consists of the following functional area frameworks:

- Data ingress and egress dataflow management
- Data representation and persistence
- Data processing and analytics
- Data security management
- Data visualization
- System management (monitoring and logging)

2.2 MAJOR COMMERCIAL OFF-THE-SHELF AND OPEN SOURCE SOFTWARE

ACS design considerations favor software in wide use with excellent support. Commercial off-the-shelf (COTS) software is preferred over government off-the-shelf (GOTS) software. This is especially true where the GOTS must be maintained by the ACS program office, PMW 160. However, GOTS software should be reused from existing developmental efforts when possible, when the software meets ACS requirements.

COTS software includes Open Source Software (OSS). DoD regulations treat open source software as COTS, and DoD acquisition regulations encourage preferential acquisition of such commercial items over program-developed items. See http://dodcio.defense.gov/Open-Source-Software-FAQ/.

ACS design choices are influenced by the desire to utilize commercially supported COTS (including OSS) and externally supported GOTS products. Internally supported GOTS software is selected only when necessary.

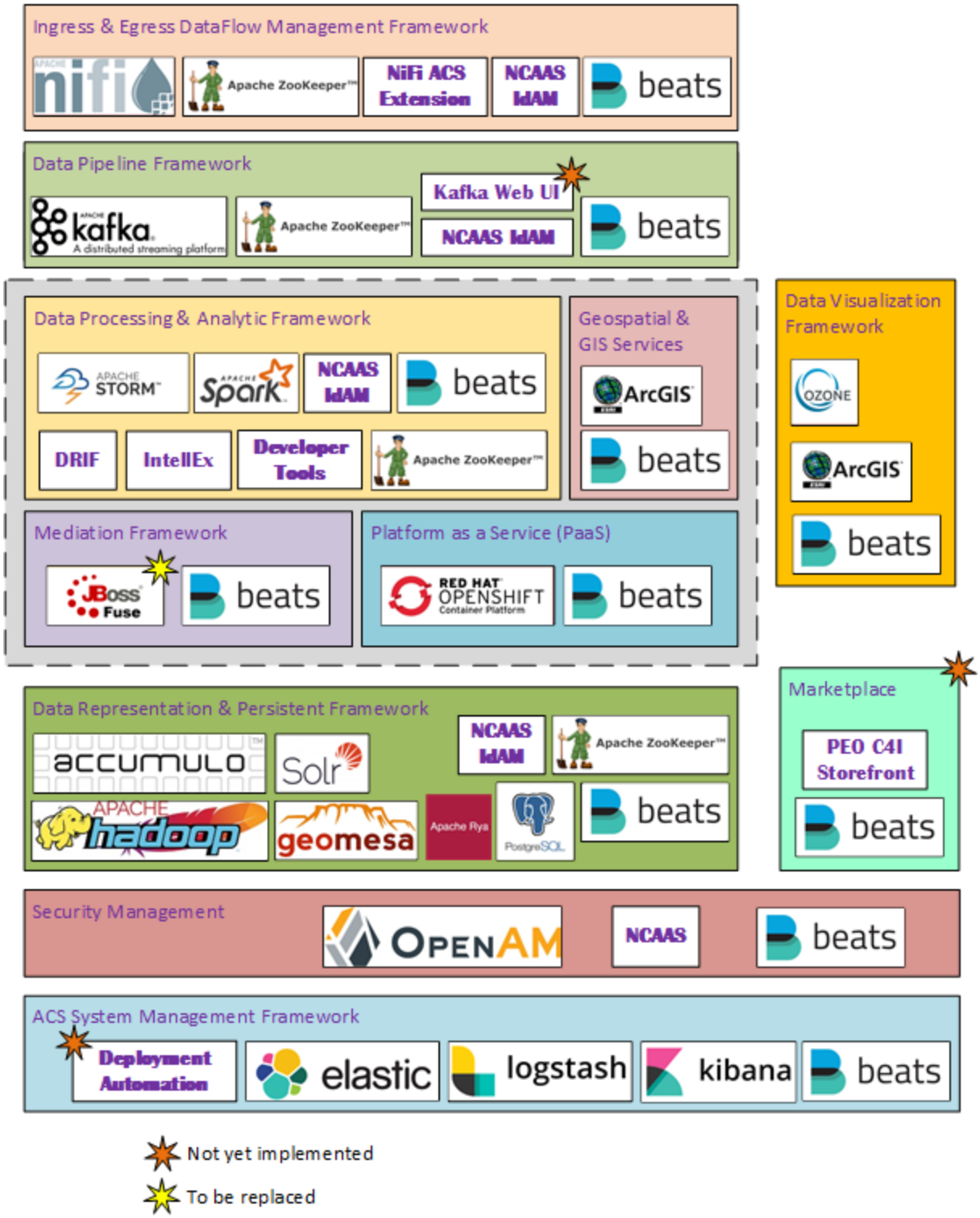


Figure 2 ACS System Management Framework Components

3. SUBSYSTEMS

3.1 INGRESS AND EGRESS DATAFLOW MANAGEMENT FRAMEWORK

The Ingress and Egress DataFlow Management Framework supports the automation and management of DataFlows (defined as the automated and managed flow of information between systems) into ACS from external data sources and systems. External data sources may include, but are not limited to, data contained in a filesystem, database, message bus, or external system. The Ingress and Egress DataFlow Management Framework also supports automation and management of DataFlows out of ACS. Outbound DataFlows may support, but are not limited to, alert delivery, preloading an afloat ACS instance with data from an ashore ACS instance, or transfer of data between two ACS instances.

The Ingress and Egress DataFlow Management Framework provides for assured data delivery, DataFlow optimization (such as data buffering and prioritized queuing), Quality of Service (QoS) management, multi-tenant visual web-based DataFlow configuration, maintenance of DataFlow data provenance, and clustering to support reliability and scalability.

Configuration of DataFlows are executed through a web-based UI supporting the design, control, feedback, and monitoring of DataFlows; Programs of Record (PoR) DataFlow Managers use this UI (after authentication and authorization) to specify configuration of Programs of Record DataFlows.

The Ingress and Egress DataFlow Management Framework supports multiple PoRs with automation and management of DataFlows in a multi-tenant manner enabling multiple groups of users (tenants) to manage DataFlows with varying levels of authorization based on user privileges.

All changes to DataFlow configurations made through the user-interface (by a single or multiple tenants) are saved to a common configuration file. This configuration may be archived and later rolled back as necessary to return all DataFlows to a known configuration.

The Ingress and Egress DataFlow Management Framework supports zero-master clustering for reliability and scalability where a single node is automatically elected by the system as the Cluster Coordinator. Failover is handled automatically through an election of a new Cluster Coordinator should failure occur with a previous Cluster Coordinator. A DataFlow Manager may configure DataFlows by interacting with the UI on any node in the cluster; any changes made are replicated to all cluster nodes.

A high-level Ingress and Egress DataFlow Management Framework architecture block diagram is shown in the following figure.

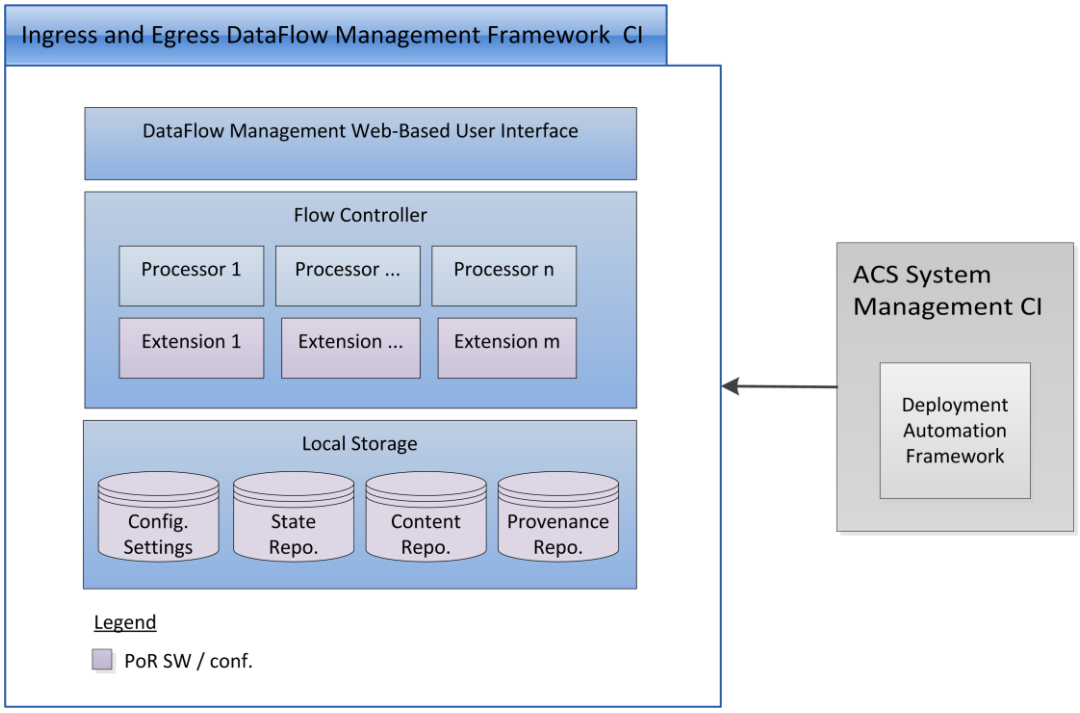


Figure 3 Data Ingest and Processing Framework Block Diagram

This block diagram depicts three major sub-components of the Ingress and Egress DataFlow Management Framework. The first is the DataFlow Management Web-Based User Interface used by DataFlow Manager. The second is the Flow Controller which executes DataFlow processes specified by the DataFlow Manager (the DataFlow Manager can utilize pre-existing processors and PoR-provided extensions). The third sub-component is the Local Storage. Local storage is required for: DataFlow configurations, flow controller processing state information, data content from the DataFlows, and provenance information for the DataFlows. The Deployment and Automation Framework (provided by the ACS System Management Framework) is the mechanism by which the Ingress and Egress DataFlow Management Framework is installed and configured.

3.2 DATA PIPELINE FRAMEWORK

The Data Pipeline Framework provides a scalable and reliable data pipeline capability enabling the access and transfer of data records (high volume and high velocity) between large numbers of data producers and consumers. The data pipeline enables a high throughput of data (records) organized into related streams (topics). The data pipeline has several key capabilities:

- **Scalability:** Topic data may be partitioned (sharded) and distributed to support high throughput of large amounts of data and load balancing.
- **Reliability:** Topic data is persisted to disk and may be further replicated across multiple servers for fault tolerance.
- **Supports Streaming and Batch Processing:** Consumers can read records at their own speed from any data point in a given topic allowing for both stream and batch processing.

A high-level architecture block diagram of the Data Pipeline Framework is shown in the following figure.

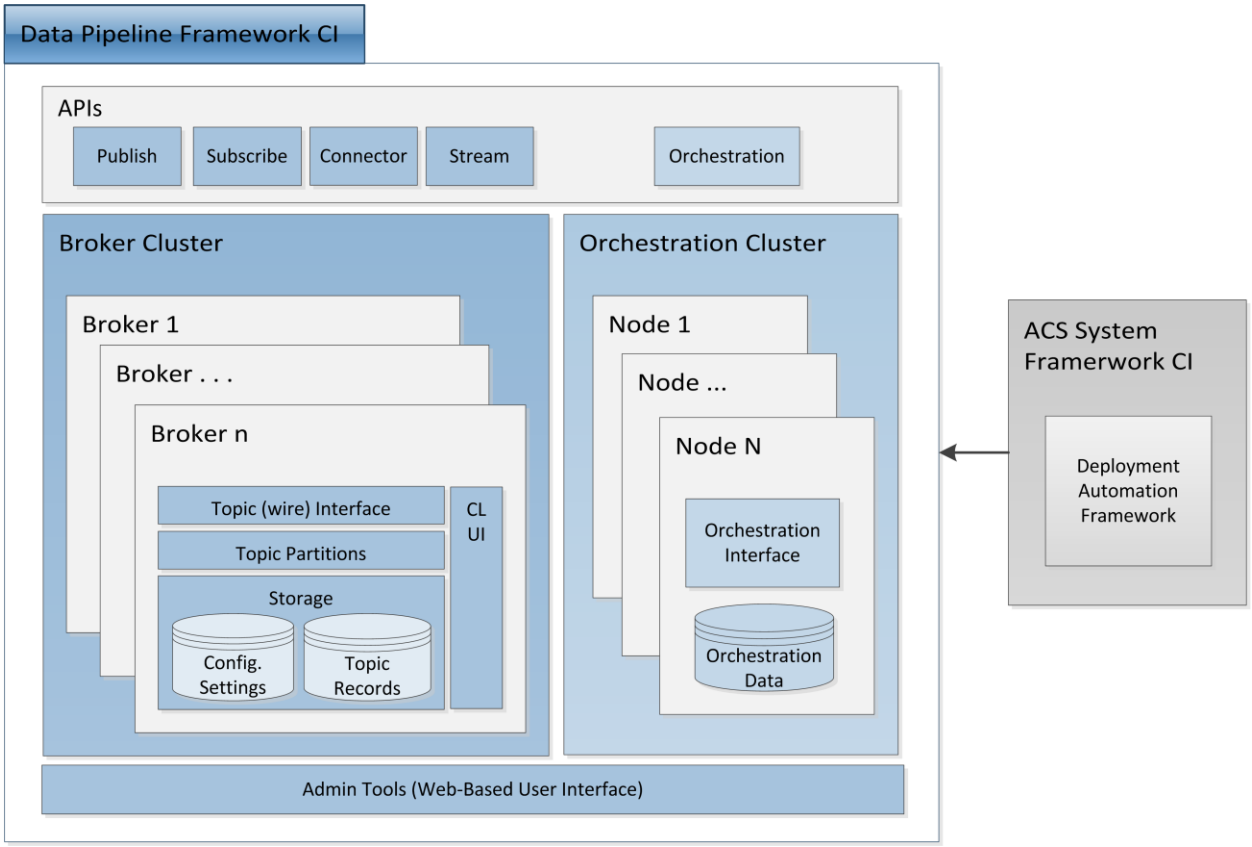


Figure 4 Data Pipeline Framework Block Diagram

The Data Pipeline Framework includes the following components:

- Set of API interfaces
- Broker
- Orchestration clusters (a set of like cooperating servers which enable scaleability and reliability)
- Administrative tools

The Data Pipeline Framework provides two main classes of customer interfaces for producing, consuming, and processing data records. The intended primary customer interface is a set of high-level programming language Application Programming Interfaces (APIs) to: publish, subscribe, and process data records; connect external systems with the data pipeline to transfer data; and manage topics. Alternatively, a low-level Transmission Control Protocol (TCP) wire interface is provided to publish to or consume data from topics when ACS-provided high-level APIs are not appropriate.

The Broker Cluster is the primary set of servers that support publishing and subscription of data records organized by topic. Each broker in the cluster has two main interfaces. The first is the externally accessible wire interface mentioned above; the second is a command line (CL) UI intended only for internal ACS use. Each broker contains an internal repository for configuration

information (e.g. topics, topic partitioning and replication settings). Each broker also contains a data repository containing published records sent to the broker. Records are persisted to disk as soon as possible to support reliability. Record retention is configurable based on repository size and/or record age.

The records for a given topic may be, and usually are, binned into a set of related partitions with single partitions containing a subset of records. Individual records themselves are not split across partitions, in support of high performance to handle a large number of records and/or a high number of record producers and consumers. These partitions may be distributed across a single broker or multiple brokers. Furthermore, partitions may be replicated across multiple brokers to support reliability should a given broker fail. This design enables a high throughput and robust data pipeline serving a large number of producers and consumers.

3.3 DATA PROCESSING AND ANALYTIC FRAMEWORK

The Data Processing and Analytics Framework Configuration Item (CI) enables rapid and efficient big data (in volume, velocity, and variety) processing, analytics, and queries across a shared data repository.

As shown in the following figure, the Data Processing and Analytics Framework supports:

- Streaming Processing Framework for non-interactive manipulation and analysis of data moving at high velocity
- Batch Processing Framework for efficient, non-interactive analysis of big data stored at rest in the Data Representation and Persistence
- Shared Semantics Framework enabling a common vocabulary across heterogeneous data sources
- Data Processing Framework provided Application Programmer Interfaces (APIs) for data ingest, normalization, enrichment, and fusion
- Pre-Defined Query Processor to support interactive queries across heterogeneous data (structured, unstructured and semantic)
- Alert Processor to generate alerts by comparing data against alert criteria
- Developer Tools to support use of the Streaming, Batch, Shared Semantic, and Data Processing Frameworks. Note: The Developer Tools are only available in ACS test environments, they are not fielded.

The Deployment and Automation Framework (provided by the ACS System Management) is the mechanism by which the Data Processing and Analytics Framework is installed and configured.

The Data Processing and Analytics Framework utilizes clustering for the Streaming, Batch, and Data Processing Frameworks as necessary to enable parallel processing for improved performance, robustness, and scalability.

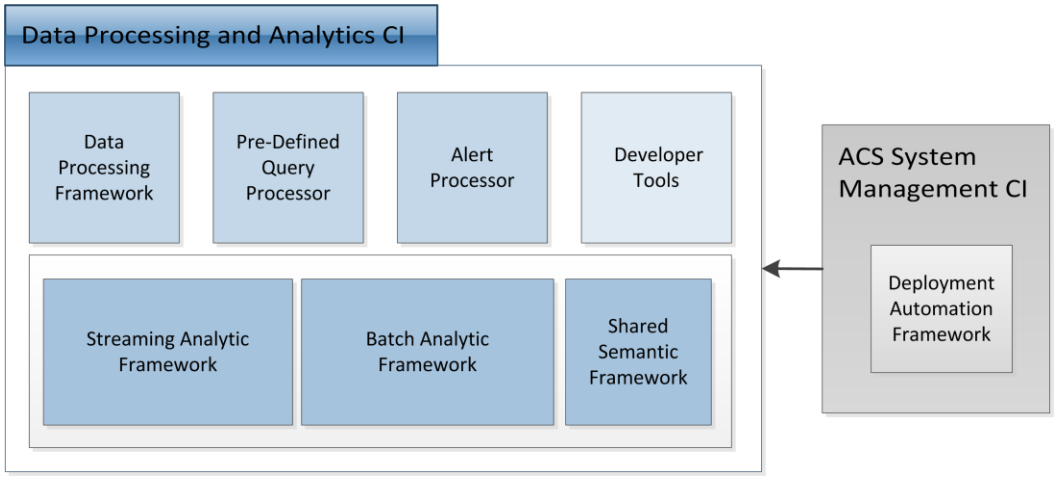


Figure 5 Data Processing and Analytics Framework Block Diagram

3.4 MEDIATION FRAMEWORK

The Mediation Framework Configuration Item (CI) provides middleware platforms for hosting and integrating modular Java software components and sharing data via machine to machine messaging using topics and queues. Together these components support implementation of common enterprise integration patterns, in particular data mediation patterns.

The following figure depicts a block level diagram for the Mediation Framework.

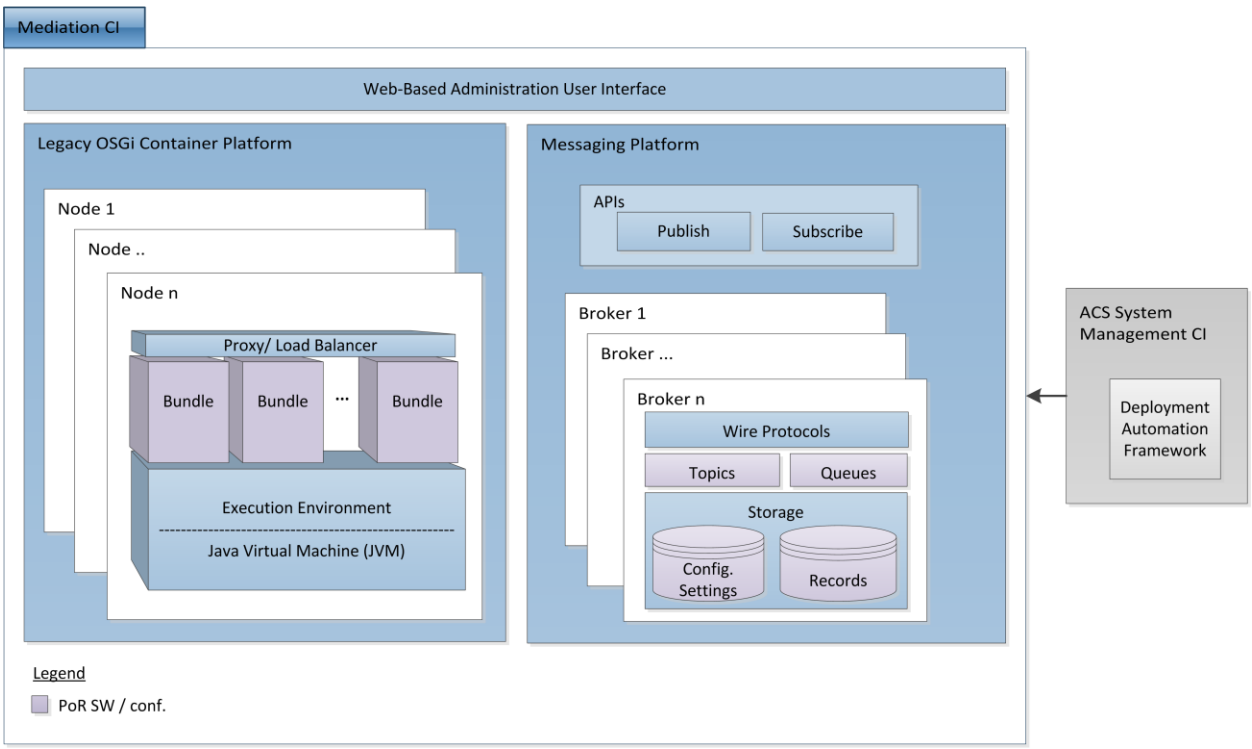


Figure 6 Mediation Framework Block Diagram

As shown in the previous figure, the Mediation Framework provides:

- Web-Based Administration User Interface supporting management of the Legacy OSGi and Messaging Platforms
- Legacy Open Service Gateway initiative (OSGi) Container Platform, clustered to enable high-volume and fault-tolerant performance, with each node consisting of:
  - OSGi container execution environment running on top of a Java Virtual Machine to standardize the hosting, management and operation of Java software components packaged as OSGi Bundles
  - Customer-provided OSGi Bundles. A Bundle is a packaged (specifically JAR files augmented with metadata in each JAR's manifest file) set of Java software components which typically implement and expose services and processes. Each Bundle is an independent unit; dependencies are deployed as separate Bundles. Bundles may execute methods on other deployed Bundles directly (via method invocation) or indirectly with other Bundles via messaging integration patterns. Bundles may be leveraged to transform, route, and mediate data and services.
  - Proxy/Load Balancer serving as an intermediary to distribute requests from clients seeking resources to nodes on the OSGi platform.
- Lightweight Messaging Platform for message publication and subscription. The Messaging Platform supports fault-tolerant, durable message publication (if configured) where messages are persisted to disk upon receipt and before acknowledgement to the publisher. The Message Platform consists of:
  - Multiple Broker Nodes, each of which contains:
    - Topics enabling publication/subscription where each message is sent to all active topic subscribers (one-to-many or many-to-many patterns)
    - Queues where each message is intentionally limited to only one subscriber (one-to-one or many-to-one patterns)
    - Storage consisting of configuration data (e.g. names of topics and queues) and the records (i.e. messages)
    - Wire-level protocol interfaces to support publishing and subscribing agnostic to programming language or application programming interface
  - Application Programmer Interfaces (APIs) consisting of:
    - Publish APIs enabling producers to publish messages to topics or queues
    - Subscribe APIs enabling consumers to receive messages from topics or queues

3.5 GEOSPATIAL DATA ACCESS LAYER AND GIS SERVICES

The Geospatial Data Access Layer & GIS Services provides a full suite of geographic data persistence, analytic, query, and mapping capabilities based around standard Open Geospatial Consortium (OGC) interfaces. These services provide a means to implement current and future extensions to include extensions for: GIS event processing, adapters to data stored in the ACS Data Representation and Persistence Framework and GIS data and service sharing via a Portal

interface. The Geospatial Data Access Layer & Geospatial Information Services leverages databases provided by the Data Representation and Persistence Framework for GIS data storage, modification, and retrieval.

The following figure depicts a block level diagram for the ACS Geospatial Data Access Layer & Geospatial Information Services (GIS) Services.

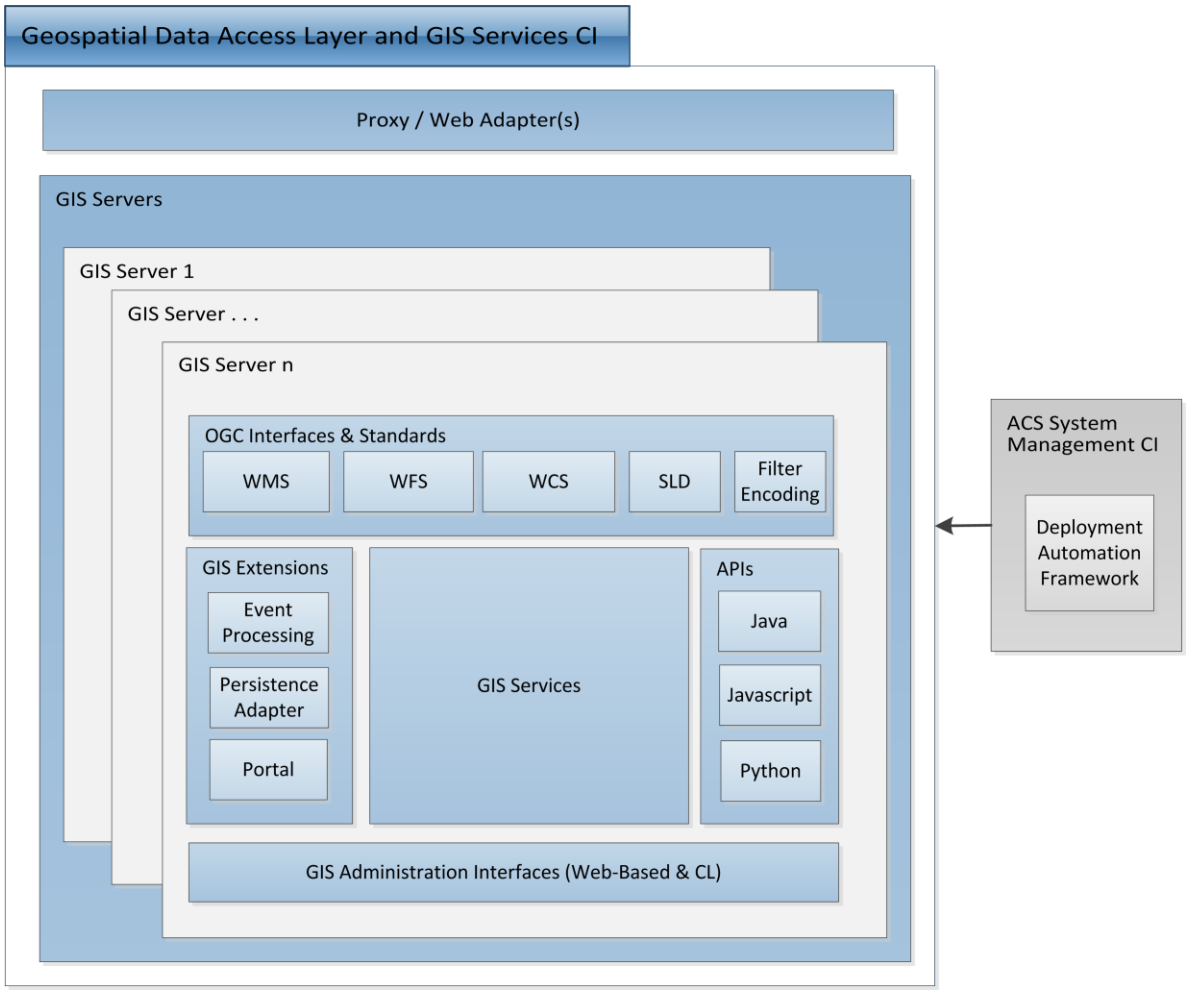


Figure 7 Geospatial Data Access Layer and GIS Services Block Diagram

As shown in the previous figure, the Geospatial Data Access Layer & GIS Services provides for the ability to deploy multiple GIS Servers to supports high-volume (e.g. data, users, and requests), robustness and fault tolerance. Each GIS Server contains the following components:

- Geospatial Information Services (GIS) providing a full suite of geographic capabilities including GIS data persistence, GIS analytics, GIS data retrieval and query, and rendering of GIS data
- OGC Interfaces and supporting standards providing standardized access to GIS Services including:
    - Web Map Service (WMS): interface for requesting custom map images from a GIS Server with a user-specified query

  - Web Feature Service (WFS): interface that provides transactions on and access to geographic point, line, and polygon features including operations for storing, modifying, and querying features
  - Web Coverage Service (WCS): interface to provide access to coverage data in forms that are useful for client-side processing. As with WMS and WFS, WCS allows retrieval of a GIS Server data based on query criteria. Unlike WMS, WCS returns data with its original semantics (instead of pictures) which may be interpreted and extrapolated. Unlike WFS, which returns discrete geospatial features, WCS returns coverages representing space/time-varying phenomena
  - Styled Layer Descriptor (SLD): interface which extends the WMS interface to specify user-defined symbolization and coloring of geographic features and coverage data addressing the need for users and software to be able to control the visual portrayal of geospatial data
  - Filter Encoding: interface for querying GIS features based on user-specified queries which may be based on spatial, temporal, comparison (e.g. =, >, <) , and logical (e.g. intersect, union) operators
- GIS Application Programmer Interfaces (APIs) to provide additional capabilities when an OGC interface is insufficient to meet requirements. Java APIs and Python APIs / scripts support interaction with GIS Servers and GIS data. Javascript APIs and libraries support client rendering and use of GIS data
- GIS Extensions capability to support extending the GIS Server beyond its initial capabilities. Extensions are provided for:
  - Event Processing: supports customer-specified event processing of geospatial data provided by the Data Processing and Analytics Framework
  - Persistence Adapter: to support interaction with the Cell-Level Secure Key-Value (NoSQL) Data Store running on the Hadoop Distributed File System provided by the Data Representation and Persistence Framework
  - Portal: providing a web-based portal capability enabling users to integrate maps, GIS feature data, and analytics with text and visualization controls (e.g. tables, charts, graphs) to generate sharable presentation and briefing products
- GIS Administration Interfaces, both Web-Based and Command Line (CL), are provided to manage and administrate the GIS Server and its GIS data

A Proxy / Web Adapter capability is provided to route requests to GIS Servers. The Proxy / Web Adapter supports:

- Integration of GIS Servers with existing HTTP servers
- Integration with the ACS Security Management
- Load balancing of requests across multiple GIS Server instances
- Filtering of request to GIS Server ports and/or URLs.

3.6 PLATFORM AS A SERVICE

The ACS Platform as a Service supports hosting and management of server-side applications for use by ACS and ACS customers with a primary goal of minimizing the complexity of providing and maintaining infrastructure required by hosted applications. ACS plans to leverage the PaaS internally to provide ACS Common Services as well Data Analytic Services as appropriate.

The PaaS leverages both hardware and software virtualization to maximize resource utilization, provide necessary isolation of hosted applications, and to minimize required administration and maintenance. This PaaS leverages virtual machines to partition and manage hardware resources and leverages software virtualization, in this case Operating System (OS) virtualization, to package and deploy applications efficiently maximizing virtual machine utilization.

A high-level architecture block diagram of the PaaS as shown in the following figure.

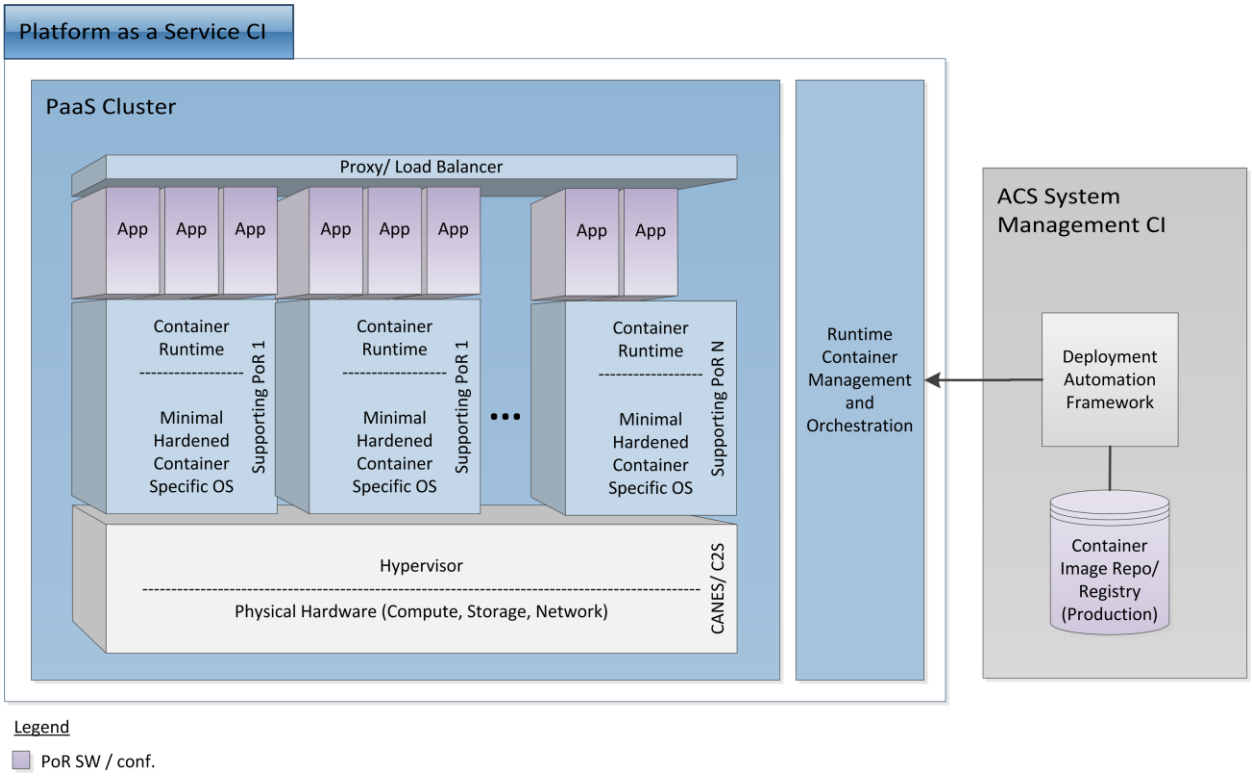


Figure 8 Platform as a Service Block Diagram

As shown in the previous figure, the ACS PaaS leverages hardware virtualization provided by CANES and/or Commercial Cloud Services (C2S) upon which PaaS software is hosted. ACS customer applications are isolated from other ACS customer applications at the virtual machine level. PaaS virtual machines are organized by customer; applications for a given customer are hosted on one or more customer specific virtual machines. Separation of applications by customer prevents one customer from impacting another (e.g. by consuming too many resources) and minimizes potential security impacts should one customer application be compromised.

The PaaS virtual machines run a specific OS tailored to secure application hosting to include:

- Hardening practices are employed. For example: the PaaS OS provides minimal services and functionality required to support application hosting, ports are closed by default (only

required ports are opened), and read-only file systems are used when possible reducing attack vectors.

- The PaaS OS supports OS virtualization enabled by a Container Runtime which provides a virtualized isolated OS for each hosted application. Each hosted application shares a common OS kernel with remaining OS resources virtualized (e.g. virtualized isolated file system, memory, network-interfaces, and inter-process communications) therefore isolating applications from each other at the OS level. In addition, the PaaS OS supports standardized application packaging and configuration. OS virtualization, standardized application packaging, along with management interfaces are provided by the PaaS OS Container Runtime.

Each application is deployed and hosted in the PaaS in an Application Container, or more simply Container. Containers provide a standard, portable, reusable, and automatable way to package, configure, and run applications.

Containers are saved within a file called Container Images. Container Images contain all required files and configurations to run the application packaged in the container. Unlike a virtual machine, container images do not contain an OS as the OS is provided by the PaaS virtual machine.

Connections to a PaaS hosted application from an application consumer are made and routed through a Proxy/ Load balancer which routes requests to the appropriate running application instance. The Proxy/ Load balancer allows for load balancing and horizontal scaling across application container instances.

Management of application containers in the PaaS (to include deployment and undeployment, configuration, starting and stopping, monitoring, etc.) is provided through a Runtime Container Management and Orchestration component which supports: the orchestration of related application containers (and the necessary network requirements to allow related application containers to communicate with each other); provides for the ability to monitor application container resource consumption, execution, and health across all PaaS virtual machines; allows for an application developer to specify (through configuration) how many application containers need to be running for a given container image and what resources, such as memory, processing, and disk need to be allocated to each.

3.7 DATA REPRESENTATION AND PERSISTENCE FRAMEWORK

The Data Representation and Persistence Framework provides a means to persist and query large volumes of data in an efficient, reliable and secure means. The framework provides a clustered data store supporting Big Data needs, as well as traditional Relational Database Management Systems (RDBMS) storage. In addition, indexing capabilities are provided both within and outside of the Big Data and RDBMS data stores to support efficient data query and lookup. A shared Network File System (NFS) is provided for internal ACS use.

The Data Representation and Persistence Framework:

- Supports Structured, Unstructured, Semantic, Tabular, Relational data types
- Provides efficient Big Data storage and retrieval supporting cell-level security and clustering for performance and fault-tolerance in the case of failure of one or more data storage nodes

- Leverages Commercial Off-The-Shelf (COTS) frameworks
- Provides Indexing within and external to databases to support timely queries

The following figure depicts a block level diagram for the Data Representation and Persistence Framework.

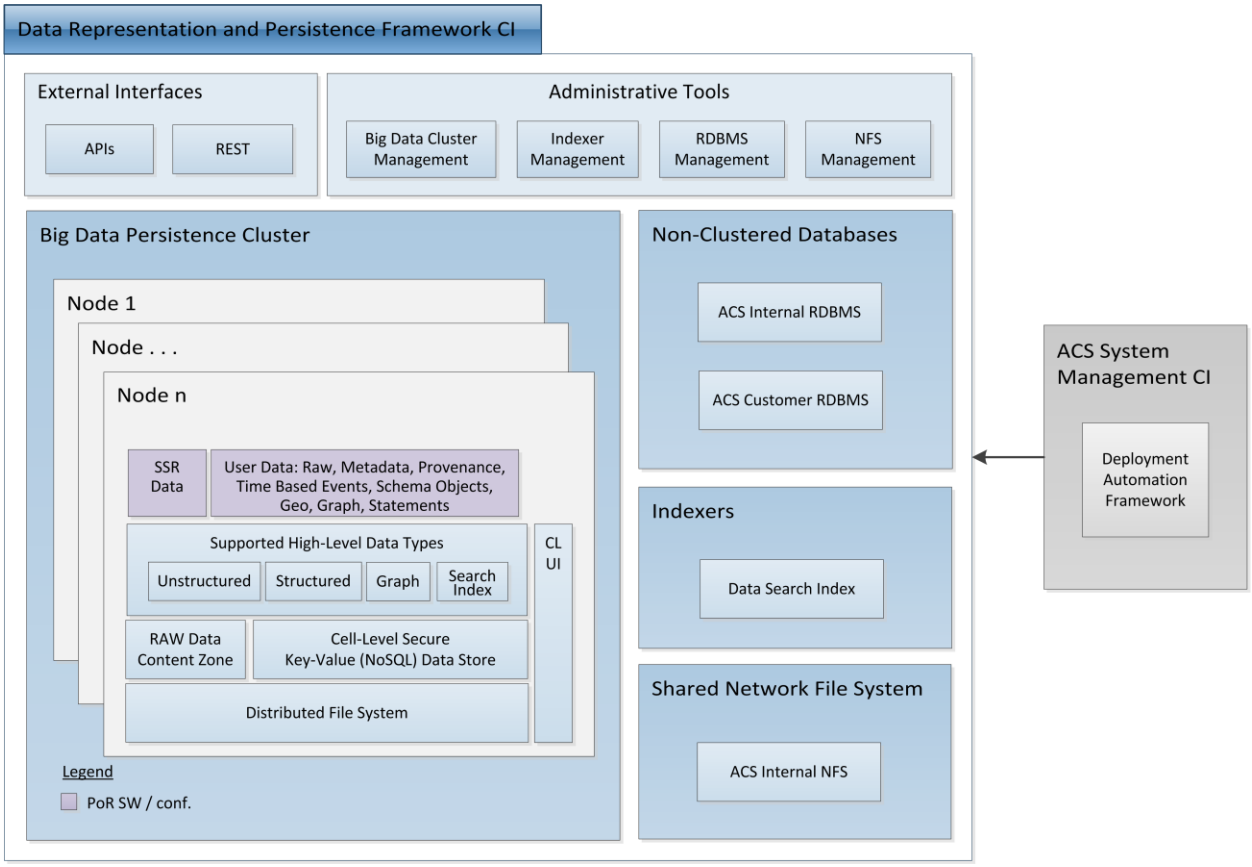


Figure 9 Data Representation and Persistence Framework Block Diagram

3.8 DATA VISUALIZATION FRAMEWORK

The Data Visualization Framework provides a mechanism to host, organize and use lightweight UIs (known as Widgets) running in an iFrame. Widgets are dynamic, lightweight, configurable, and composable web-based UI components based on a Widget Application Programming Interface (API). Widgets are often a summary or limited view into a larger web application. They may operate singularly or as a family of Widgets, usually related by purpose, for execution of specific mission functions. For example, simple single-purpose Widget may be used for chart annotation; while a collection of more complex Widgets may be used to assist mission workflow. Before end user use, Widgets are distributed and installed into the Data Visualization Framework by the ACS Marketplace.

The Data Visualization Framework supports the following goals:

- Speed to Capability: Enables rapid updating and fielding of UIs; system owners can provide incremental capabilities directly to users utilizing ACS for automatic (upon approval) distribution, updating, installation, and configuration

- Modularity: Supports decoupling of UIs from data and business logic
- Discoverability: Widgets (and their back end systems, services, and data) as well as Dashboards (a configured collection of Widgets) are discoverable by users
- Reusability: Common Dashboards and Widgets may be leveraged by different Systems and users to maintain a common user experience promoting improved operator effectiveness and reduction in training requirements

The Data Visualization Framework provides the following capabilities:

- Configurability: Supports the use of Dashboards to organize, layout, and display Widgets as a collection (usually related to mission function or purpose)
- Pluggable Security Framework: Provides an isolation layer (for later re-configurability without ACS customer rework) to the operational security mechanism, provided by the ACS Security Management, for Widget authentication and authorization based on user credentials, role, and/ or attributes
- Inter-Widget Communication: Supports light-weight inter-widget communication (within a single web-browser page) for widgets within a given Dashboard

The following figure depicts a block level diagram for the Data Visualization Framework.

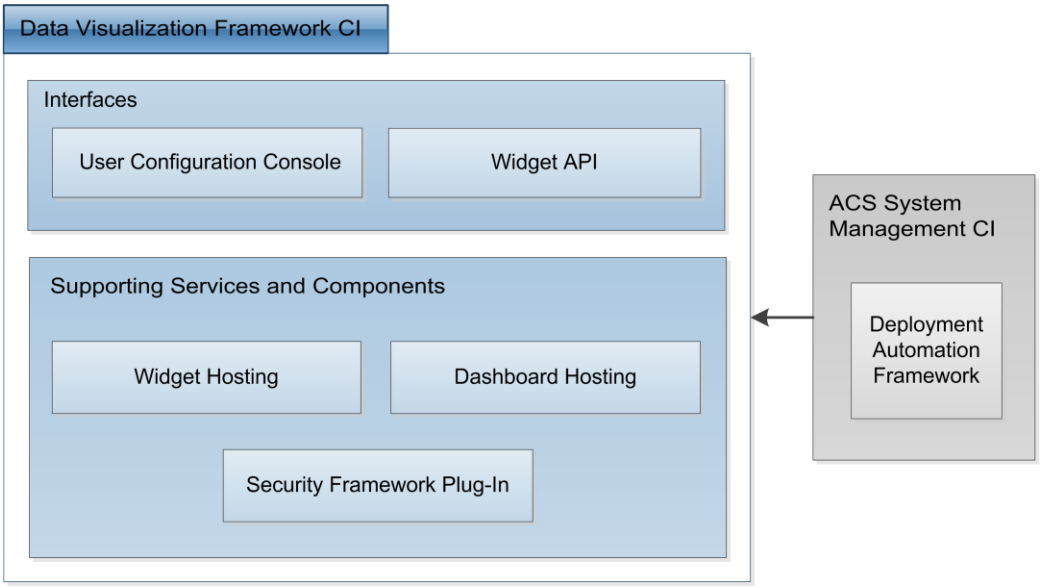


Figure 10 Data Visualization Framework Block Diagram

As shown in the previous figure, the Data Visualization Framework CI consists of Interfaces and Supporting Services and Components. The Data Visualization Framework CI is installed and configured by the Deployment Automation Framework component of the ACS System Management CI. The Data Visualization Framework CI provides three supporting services: Widget Hosting; Dashboard Hosting; and a Security Framework Plug-In. The Data Visualization Framework also utilizes a small Data Visualization Framework Repository (shown in Figure 3), provided by the Data Representation and Persistence CI, for Widget and Dashboard configuration data as well as user preferences.

The Data Visualization Framework provides and utilizes the following services:

- Widget Hosting: An environment to host Widgets within ACS as well as register externally hosted (outside of ACS) Widgets. Widgets, associated with a configurable URL, are served to end users over Hypertext Transfer Protocol Secure (HTTPS). Widgets are distributed, installed, and configured by the Marketplace CI for hosting in the Data Visualization CI.
- Dashboard Hosting: Provides for layout and organization of Widgets within a user's web browser. Dashboards allow for the grouping of Widgets and facilitate development of application workflows and UI integration. Dashboards may be predefined before deployment (e.g. by a PoR developer) or configured in a production environment by a user based on mission need or common widget usage. Dashboards are sharable and discoverable. Each Dashboard may contain one or more panes, each with one or more Widgets.
- Security Framework Plug-In: Provides a swappable interface to the security framework provided by the ACS Security Management CI for Widget authentication and authorization based on user credentials, role, and/ or attributes. Requests to Widgets and Dashboards from a user web browser are intercepted by the Plug-In and routed to the ACS Security Management CI for authentication or authorization upon which a security token is placed in the user's web browser session. Widgets and Dashboards in the session may use this token to obtain user credentials, roles, and attributes and take action (to enable or disable Widget capabilities) based on the token data.

The Data Visualization Framework provides the following external interfaces:

- User Configuration Console: Provides a web based UI which supports Data Visualization Framework end users with creation, updating, and configuration of Dashboards as well as configuration of user preferences
- Widget API: Supports PoR developers with Widget development. The API supports common Widget capabilities such as: local Inter-Widget communication (within the user web browser page session), storage and retrieval of user preferences (such as Dashboard configurations and Widget bookmarks), logging, responding to Widget events (such as drag and drop, user selection, and Widget startup/ shutdown), and Widget chrome (toolbars, menus, etc.). The API also specifies the packaging requirements for Widgets.

3.9 MARKETPLACE

The Marketplace provides UIs and services for discovery, distribution, installation, and updating of Widgets (dynamic, lightweight, configurable, and composable web-based UI components based on a Widget Platform Application Programming Interface) and Widget supporting services (back end supporting services often based on Java; here after called Widget Services) deployed and utilized within ACS.

The Marketplace supports three main goals:

- Speed to Capability: Enables rapid fielding of warfighter capabilities by automating the distribution, updating, and installation of Widgets and Widget Services deployed in ACS. Addresses PEO C4I's Strategic Plan for 2015-2021 (goal two) to improve speed to capability by innovatively applying technology and processes.

- Simplified User Experience: Provides a common distribution, update, and installation service for Widgets and Widget Services deployed in ACS therefore reducing administration burden
- Driving Down Cost: Reduces fielding and deployment costs. All Program of Records (PORs) have costs associated with distributing, updating, and installing components. The ACS Marketplace CI supports reduction in these costs by providing common capabilities for distribution, discovery, and installation of Widgets and Widget Services in ACS.

The following figure depicts a block level diagram for the Marketplace.

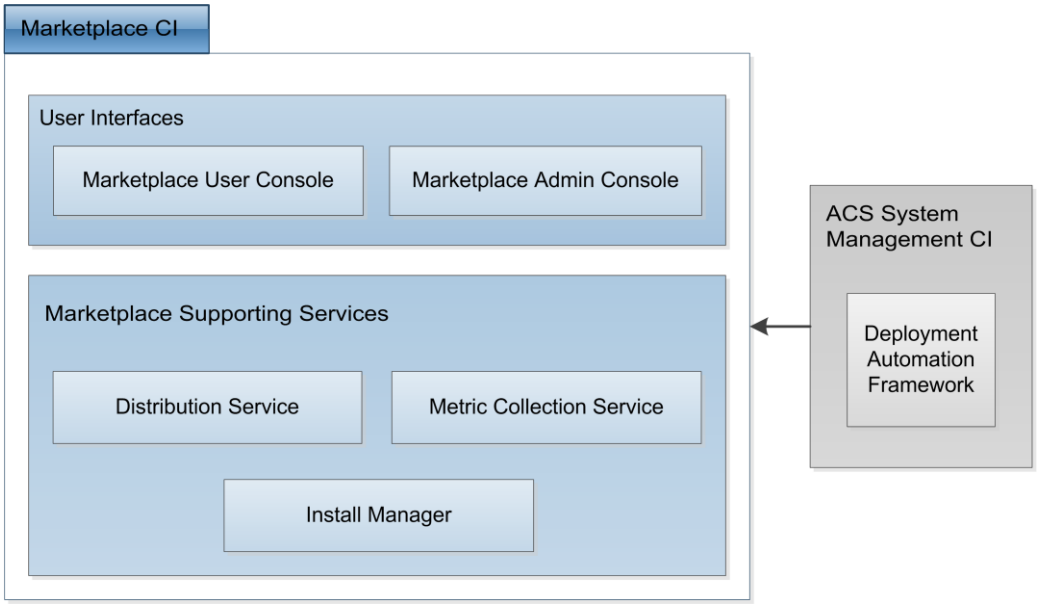


Figure 11 Marketplace Block Diagram

As shown in the previous figure, the Marketplace CI consists of UIs and a set of Marketplace Supporting Services. The Marketplace CI is installed and configured by the Deployment Automation Framework component of the ACS System Management CI. The Marketplace CI provides three supporting services: a Distribution Service; an Install Manager; and a Metric Collection Service. The Marketplace also utilizes a Marketplace Repository, provided by Data Representation and Persistence CI, for Widgets, Widget Service binaries, and related meta-data. Together these components support discovery, distribution and installation of Widgets and Widget Services upon request by administrators.

The Distribution Service is responsible for reliable and secure downloading of approved Widgets and Widget Services from a shore Distribution Center to an operational ACS instance (where the Distribution Service is running). Initially, only metadata for each approved Widget and/ or Widget Service is downloaded; upon administrator approval, remaining files supporting installation and operational use are downloaded. This design enables improved bandwidth allocation, especially in a D-DIL (disconnected, intermittent, and limited) environment.

The Install Manager provides a common automatable means to install, configure, update, and uninstall Widgets and Widget Services. The Install Manager track versions of deployed artifacts and provides this information to the Metric Collection Service.

The Metric Collection Service captures metadata regarding versions of artifacts available in the Marketplace Repository (i.e. downloaded Widgets and Widget Services) as well as which

versions are deployed (i.e. in operational use in the local ACS instance). The Metric Collection Service compares the set of available/deployed artifacts in the production ACS environment against versions in the Shore Distribution Site thereby providing a list of available updates. The Metric Collection Service also provides installation metrics to the Shore Distribution Site for configuration management and metric purposes.

The Marketplace CI provides two browser-based UIs: the Marketplace User Console; and the Marketplace Admin Console with the following capabilities:

- The Marketplace User Console provides users with the POR information describing the Widgets and Supporting Services, such as version number, version date, supporting POR functions, screen shots of the Widget and user feedback (scores and comments). With this metadata, Users can search and make educated decisions before downloading new Widgets or Widget Services.
- The Marketplace Administrator Console provides an administrator-only view to support the management of Widgets and Widget Services within the production ACS instance. From this console, an administrator can approve user download requests, schedule downloads, install and configure Widgets and Widget Services, manage updates, and view installation and usage metrics.

3.10 SECURITY MANAGEMENT

The Security Management CI enables policy-based access to network-accessible applications, data, and services hosted within and/ or external to ACS. A policy defines ***who*** (e.g. a person, system) can execute a specific ***action*** (e.g. read or write) on a specific ***resource*** (e.g. URL-accessible service) under specific ***conditions*** (e.g. role, security clearance, citizenship). The Security Management CI supports both human-to-machine and machine-to-machine interactions.

The Security Management CI supports:

- Authentication to verify identification (e.g. human or system) utilizing an externally provided trusted identity store (e.g. Active Directory)
- Authorization to grant access to resources based on identity, roles, or attributes
- Federation to securely share identity information across heterogeneous systems or domain boundaries
- Single Sign On (SSO) to enable a single authentication for multiple actions on independent resources within a single session
- High availability to: minimize single points of failure (for authentication, authorization, federation, and SSO); implement load balancing to handle high volumes of service requests; enable failover of user sessions to minimize session data loss
- Support for multiple programming interfaces (e.g. Java, C, and REST)

The Security Management CI maximizes the use of standards, in particular:

- X.509 Public Key Certificates (to include soft and Common Access Cards Hardware Tokens)
- Security Assertion Markup Language (SAML) 2.0
- Open Authorization (OAuth)

- Online Certificate Status Protocol (OCSP)

The Security Management CI Attribute Based Access Control (ABAC) and Policy Based Access Control (PBAC). ABAC is an access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions.

- Attributes are characteristics of the subject, object, or environment conditions. Attributes contain information given by a name-value pair.
- A subject is a human user or NPE, such as a device that issues access requests to perform operations on objects. Subjects are assigned one or more attributes.
- An object is a system resource for which access is managed by the ABAC system, such as devices, files, records, tables, processes, programs, networks, or domains containing or receiving information.
- An operation is the execution of a function at the request of a subject upon an object. Operations include read, write, edit, delete, copy, execute, and modify.
- Policy is the representation of rules or relationships that makes it possible to determine if a requested access should be allowed, given the values of the attributes of the subject, object, and possibly environment conditions.
- Environment conditions: operational or situational context in which access requests occur. Environment conditions are detectable environmental characteristics, and are independent of subject or object, and may include the current time, day of the week, location of a user, or the current threat level.

PBAC is a standard that combines both Role Based Access Control (RBAC) and ABAC. PBAC is not based on any one implementation (XACML, for example), but rather is a method to manage access decisions and authorizations regardless of technical implementation. PBAC is capable of scaling and building on existing identification and authentication systems, while providing an access-control solution according to context-based policies.

The Security Management CI depends upon and utilizes a trusted identity store and OCSP responder from the production deployment environment. The Security Management CI relies upon the external trusted identity store and OCSP responder to ensure only Public Key Certificates issued by a DoD Public Key Infrastructure (PKI) or by an approved external PKI are utilized for authentication and authorization decisions.

The following figure depicts a block level diagram for the Security Management CI.

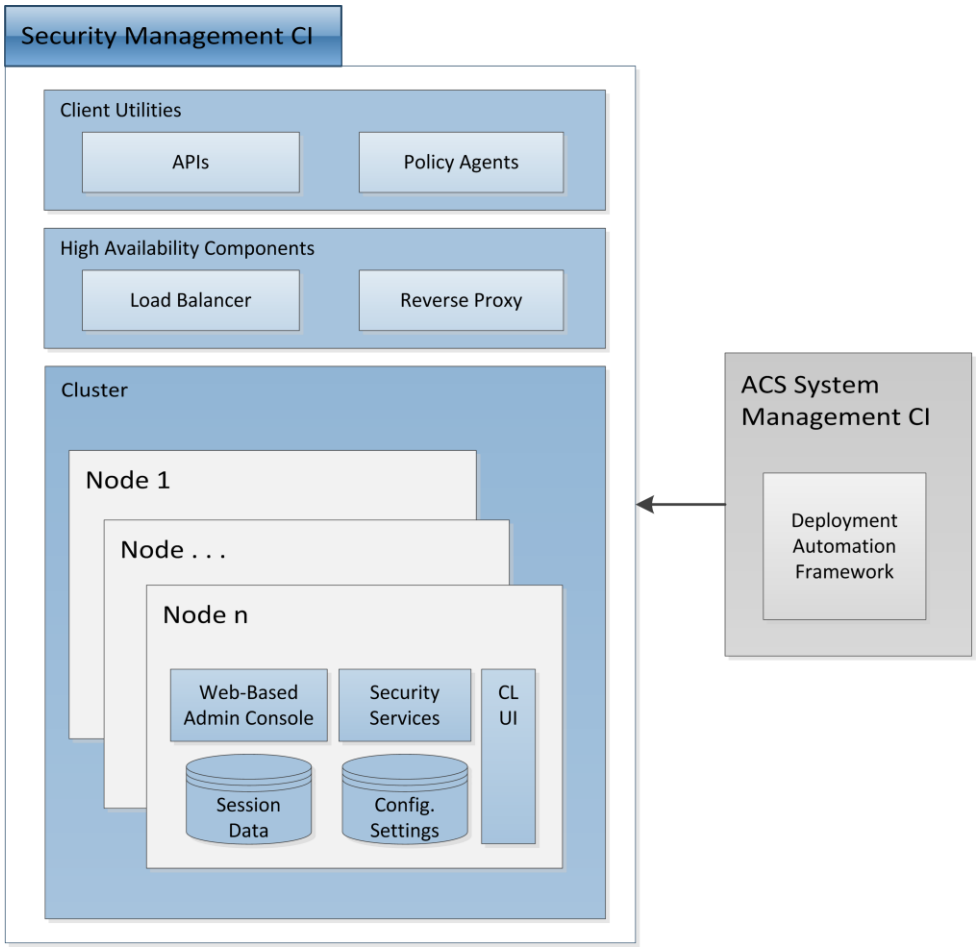


Figure 12 Security Management Block Diagram

As shown in the previous figure, the Security Management CI provides Client Utilities for securing POR endpoints deployed within and external to ACS. In particular, the Security Management CI provides prebuilt, deployable, and configurable Policy Agents which can be utilized for web-based applications/ services. Policy Agents function as gatekeepers to ensure clients are authenticated and authorized before accessing the resource the Policy Agents are protecting. Application Programming Interfaces (API) supporting C, Java, and REST are provided for instances where a pre-built Policy Agent is not available or suitable.

The Security Management CI provides High Availability Components to include Load Balancers and Reverse Proxies to support scalability and reliability. The Load Balancers distribute traffic across multiple Security Management CI nodes (when the Security Management CI is configured with more than one node). The Reverse Proxies route client requests to the requested resource being protected.

The Security Management CI may be configured to run on more than one node for performance or reliability/ availability when a node fails. Each node has a Web-Based Admin Console used for administration and monitoring as well as a Command Line (CL) UI used only by the Deployment Automation Framework for installation and configuration. Each Node executes Security Services which the APIs and Policy Agents communicate with for authentication and authorization requests. Each node stores configuration data as well as session data locally.

3.11 ACS SYSTEM MANAGEMENT FRAMEWORK

ACS offers many management capabilities, some of which address working with data and associated flows, controlling access to the data, and even controlling deployment of applications themselves that work on the data. While some management aspects of these capabilities are inherent within the subcomponents, the ACS System Management Framework Configuration Item (CI) provides two main capabilities. The first is the Deployment Automation Framework used to install and configure ACS CIs. The second is a collection of utilities used for logging, monitoring, and auditing of ACS components and PoR data (to include ingest, processing, access, etc.).

The following figure depicts a block level diagram for the ACS System Management Framework.

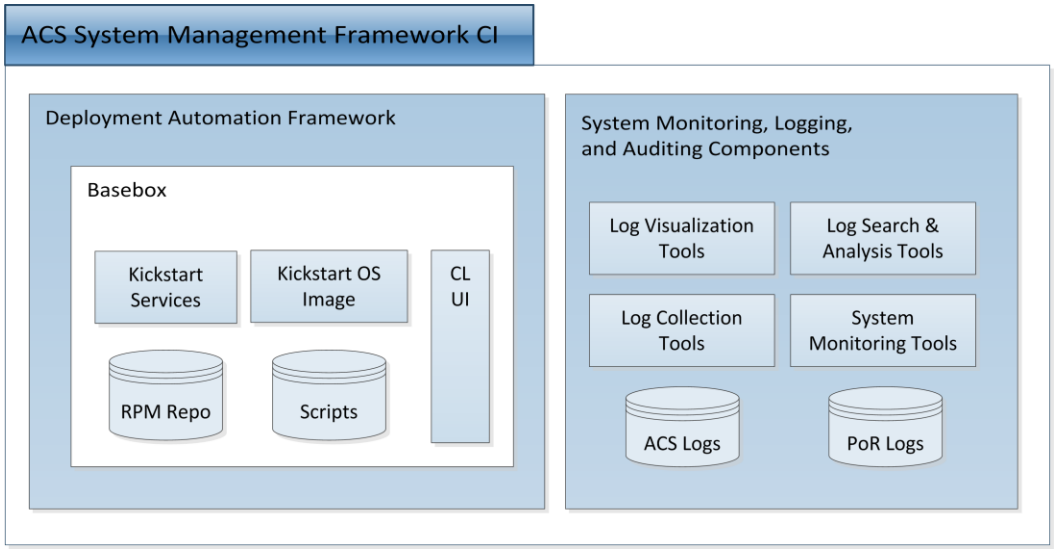


Figure 13 ACS System Management Framework Block Diagram

While individual ACS CIs provide tailored management capabilities specific to each CI, the ACS System Management Framework CI provides overarching installation, configuration, monitoring, logging, and auditing capabilities for all ACS CIs. As shown in the previous figure, the ACS System Management Framework CI provides:

- A Deployment Automation Framework consisting of a single Basebox, typically installed as a Virtual Machine (VM), which has the following types of components:
  - RPM (Red Hat Package Manager) Repository containing RPM files for ACS software as well as PoR software that deploys in ACS (excluding PaaS-deployed software, which has its own container image format)
  - Scripts used to automate the install and configuration of ACS components and PoR software (e.g. data models, data processors, pre-defined alert criteria, widgets), installed within ACS
  - Kickstart Operating System (OS) Image that is a bootable base image upon which all ACS CIs are built.

  - Kickstart Services are ACS-dependent services, which may or may not be available in every environment in which ACS is deployed (especially developmental environments). These services are used during installation until ACS is configured to utilize equivalent services from the installation environment. Kickstart Services include Domain Name Service (DNS), Dynamic Host Configuration Protocol (DHCP), Preboot eXecution Environment (PXE) Server, and Lightweight Directory Access Protocol (LDAP).
  - Command Line (CL) UI accessible via Secure Socket Shell (SSH). The Basebox does not provide a graphical user interface
- System Monitoring, Logging and Auditing Components include:
  - Repository for ACS Logs which contains logging information for each ACS CI
  - Repository for PoR Logs which contains logging information for each PoR software component deployed within ACS (to include PoR data logging, such as data source, data provenance, data enrichment, and data access)
  - System Monitoring Tools to collect and display information on ACS component status
  - Log Collection Tools to collect and route logs to the appropriate ACS / PoR log repositories. Log Collection Tools include Log Collection Agents and Log Collection Services. Log Collection Agents may be utilized within an ACS CI or within PoR ACS-deployed software (such as a service deployed in the PaaS). Log Collection Services collect data logs from Log Collection Agents for persistence.
  - Log Search and Analysis Tools to query and analyze log data (both ACS and PoR logs) based on user identity and authorization
  - Log Visualization Tools to display, sort and filter logs in a manner meaningful for analysis, auditing and monitoring

4. COMPONENTS

4.1 NIFI

4.1.1 Overview

Apache NiFi is the primary product used to satisfy the Ingress and Egress DataFlow Management Framework CI requirements. As shown in the following figure, the Ingress and Egress DataFlow Management Framework CI : DataFlow Management Web-Based User Interface is used to create, configure, manage, and test DataFlows. These DataFlows are executed by the NiFi Flow Controller shown in the figure. A typical use case is to configure a DataFlow to obtain data from an external source(s), perform some (generally minimal) data processing, and push the data onto a topic hosted by the Data Pipeline CI; see this CI's SRS and SDD for more details on common use cases and Ingress and Egress DataFlow Management Framework CI implementation.

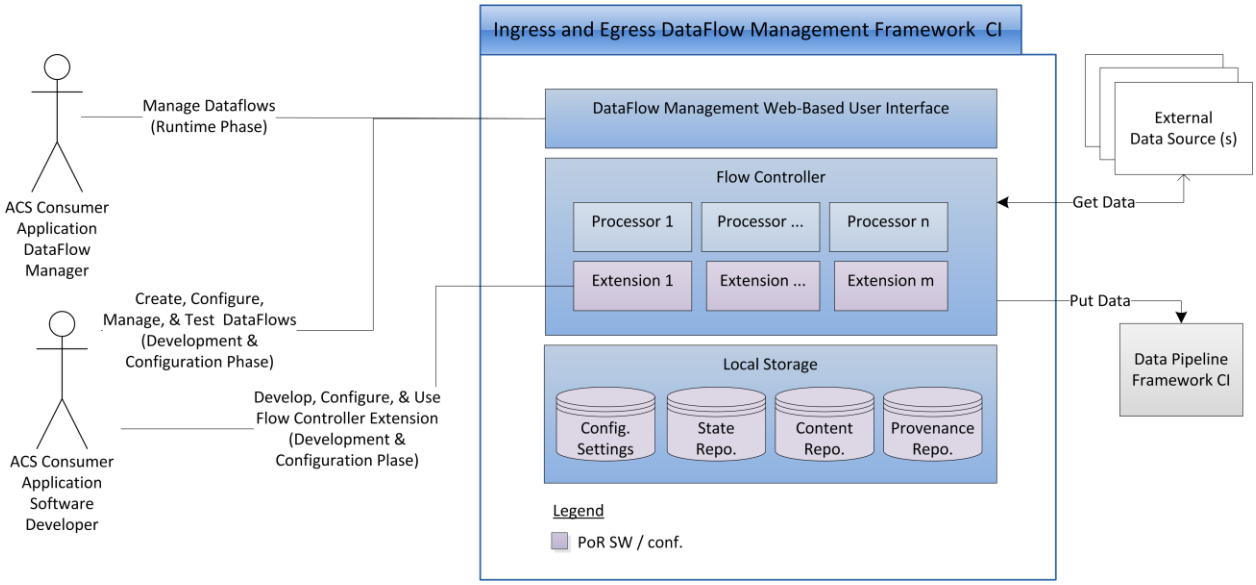


Figure 14 Ingress and Egress DataFlow Management Framework

4.1.2 Interfaces

4.1.2.1 Web GUI

The Ingress and Egress DataFlow Management Framework CI primary interface is a web-based GUI used to visualize, edit, configure, monitor, and manage DataFlows. The GUI is configured to only accept connection via an encrypted channel running on port 443 after authentication and authorization. Authentication and authorization is handled by the IdAM plugin that integrates with the ACS Security Framework CI to support authentication and authorization of ACS administrators and ACS customer Dataflow managers when accessing the NiFi web-based UI.

The GUI provides a visual canvas to reuse, wire together, and configure components (e.g. NiFi-provided and ACS-provided processors) used to process DataFlows. The GUI provides features

to monitor and troubleshoot DataFlows. The GUI provides tools to create templates and process groups for reuse and simplification of the display of DataFlows.

A sample GUI screenshot showing a simplified DataFlow which uses several Processors (both NiFi-provided and ACS-provided) along with the processor interconnections and configurations is provided in the following figure.

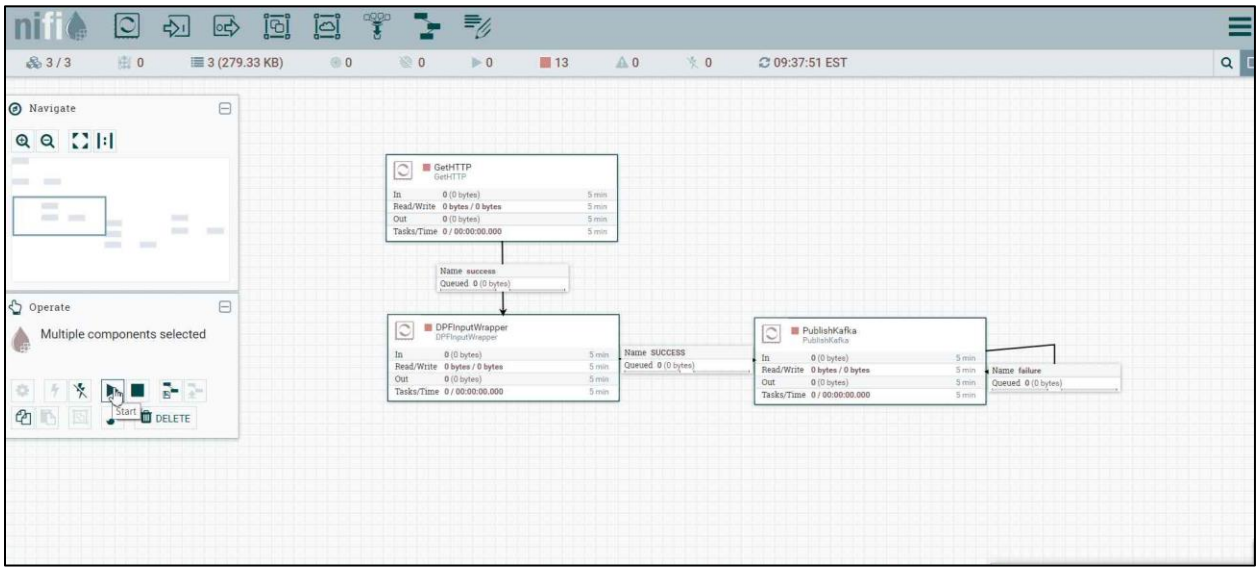


Figure 15 NiFi GUI

Further information regarding the NiFi GUI may be found at https://nifi.apache.org/docs/nifi-docs/html/user-guide.html.

4.1.2.2 REST Interface

The REST API is used internally by the NiFi GUI to provide the ability to command and control different processors or elements of a NiFi instance in real time without having any adverse effects to application activities or downtime. Some examples of using the REST API through the NiFi GUI are to change flows and creating templates, while other processors are still running. It leverages HTTP to use GET, PUT, POST, and DELETE to control actions upon NiFi instances. See the official NiFi documentation for further details at https://nifi.apache.org/developer-guide.html, https://nifi.apache.org/docs/nifi-docs/rest-api/index.html and for documents specific to NiFi version 1.1.2 https://github.com/apache/nifi/tree/rel/nifi-1.1.2/nifi-docs.

4.2 NIFI ACS EXTENSION

ACS consumer applications software developers may use the developer interfaces to extend NiFi capabilities for their specific needs within guidelines provided by extension interfaces and guidance provided in this ICD and the ACS Software Developer Guide. These extensions are then installed for use within the Flow Control and configured for use via the Ingress and Egress DataFlow Management Framework CI : DataFlow Management Web-Based User Interface.

ACS provided NiFi extensions to support ACS specific data markup and storage requirements (e.g. extensions to provide common data transport and persistence envelopes.)

Table 1 NiFi ACS Extension

Processor Name	Processor Function
GeospatialFilter	A simple geospatial filter processor
CmfLocationExtractor	Wraps payload in information needed by the DPF topology to process
RulesProcessor	Runs rules based on data passed as attributes
AddDPFInputProvenanceAttachment	Adds an attachment containing provenance XML to a DpfInput flow file
ConsumeJmsHornetQ	Consumes HornetQ JMS Message
ContentZonePersistence.	Loads artifact in ContentZone and returns a URI.
ConvertKeyValueStringToXml.	
ConvertOthToCsv2	Converts USMTF formatted OTH-G messages into CSV
DPFInputWrapper	Wraps payload in information needed by the DPF topology to process
ExtractImageExifInfo	Extracts EXIF information from Images
GetTCP	Gets bytes from a given TCP port and concatenates the contents together generating flow files
IBSXmlBuilder	Collects IBS data in stream format and separates them into individual messages
ListenSMTP	Starts an SMTP server and processes incoming messages
MessageBuilder	Collects sequential fragmented Flow Files and extracts specific portions of them based on Regular Expressions
TCRIPutSCP	transfers stream using scp to remote system
TDOUnWrapper	Unwraps a Trusted Data Format (TDF) payload
TDOWrapper	Formats stream in Trusted Data Format (TDF)
WrapXmlDocument	Formats stream in Trusted Data Format (TDF)

4.3 KAFKA

Apache Kafka (https://kafka.apache.org/) is a distributed streaming platform.

A streaming platform has three key capabilities:

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system
- Store streams of records in a fault-tolerant durable way
- Process streams of records as they occur

Kafka is generally used for two broad classes of applications:

- Building real-time streaming data pipelines that reliably get data between systems or applications
- Building real-time streaming applications that transform or react to the streams of data

In Kafka the communication between the clients and the servers is done with a simple, high-performance, language agnostic TCP protocol. This protocol is versioned and maintains backwards compatibility with older version.

Apache Kafka is the primary product used to satisfy the Data Pipeline Framework CI requirements. As shown in the following figure, the Data Pipeline Framework CI: Admin Tools GUI Interface is used to manage, monitor, and administer the Kafka clusters. Also shown are typical use cases for the Kafka Producer, Consumer, Streams, and Connector APIs as well as the Kafka Wire Protocol. See this CI's SRS and SDD for more details on common use cases.

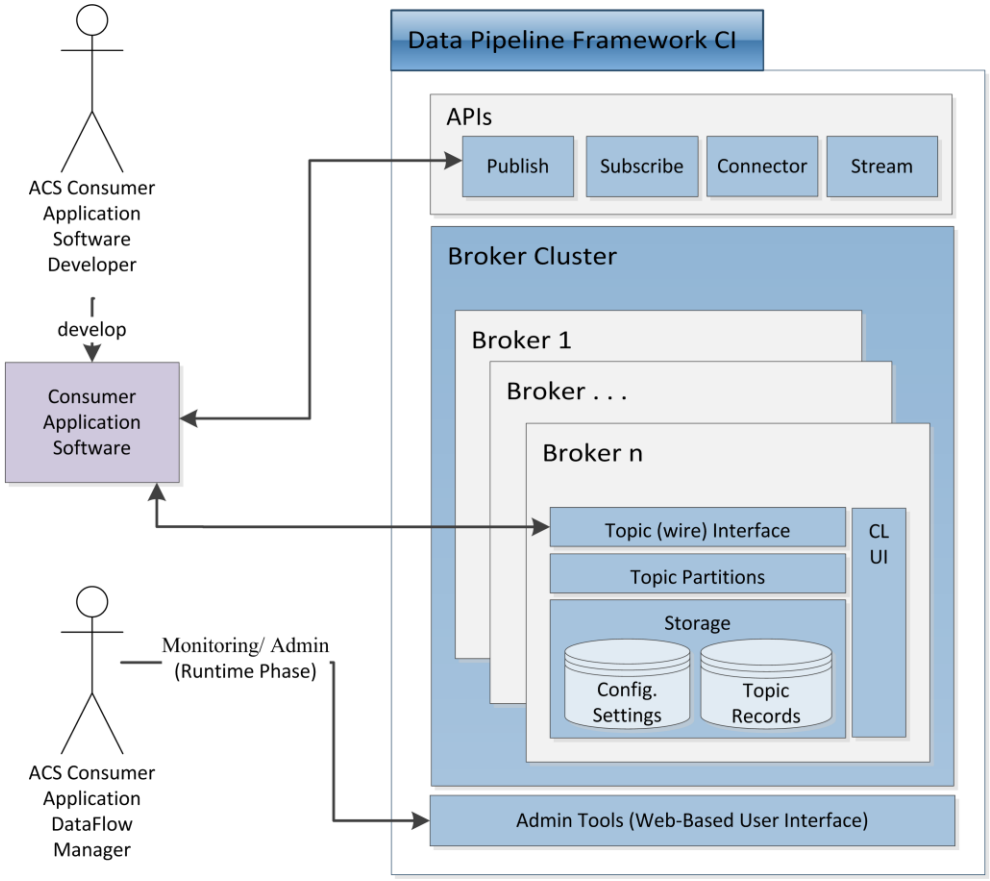


Figure 16 Kafka Interface

4.4 HADOOP

Forms the base of our cluster of machines and services. Here's a great intro and overview. Hadoop handles the distribution of large amounts of data across multiple computers in drives. Hadoop keeps everything organized, makes copies, backups, etc. Luckily we can treat hadoop like a mounted filesystem, and just drop our data in there. Hadoop will handle the rest.

Hadoop is an Apache Software Foundation community driven, open-source project for distributed storage and processing of large sets of data using the MapReduce programming model. The Hadoop Java API allows developers to write applications that interface with Hadoop in order to take advantage of its large dataset processing capabilities.

The core of Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional architecture.

4.4.1 Interfaces

4.4.1.1 Web GUI

The Hadoop Web GUI provides an overview of the Hadoop cluster, including the status of nodes, the disk storage used, the live nodes, capacity and other statistics useful for assessing the state of the system. Additional detail is provided in other tabs. For example, in the picture below, one can see the Datanode information showing each node's last contact, capacity, blocks and version.

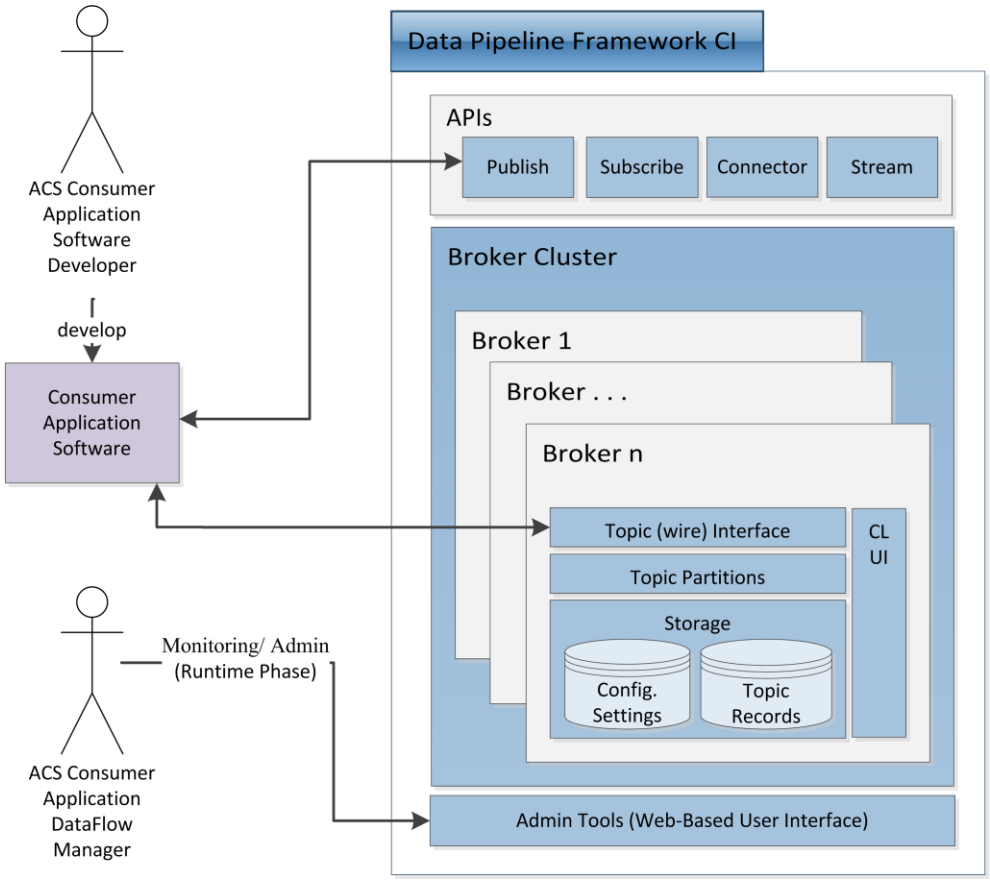


Figure 17 Hadoop GUI Datanodes

HDFS is the primary distributed storage used by Hadoop applications. A Hadoop Distributed File System (HDFS) cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data. As noted in the Hadoop HDFS User Guide, Hadoop is organized around the following concepts:

- Hadoop, including HDFS, is well suited for distributed storage and distributed processing using commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. MapReduce, well known for its simplicity and applicability for large set of distributed applications, is an integral part of Hadoop.
- HDFS is highly configurable with a default configuration well suited for many installations. Most of the time, configuration needs to be tuned only for very large clusters.
- Hadoop is written in Java and is supported on all major platforms
- Hadoop supports shell-like commands to interact with HDFS directly
- The NameNode and Datanodes have built in web servers that makes it easy to check current status of the cluster

4.4.1.2 Java API

The Hadoop Java API provides packages and classes defining a comprehensive interface for Java developers to access the Hadoop/Hadoop Distributed Filesystem (HDFS) capability, which supports the ACS Data Representation and Persistence Framework component. The API provides numerous classes in more than 100 packages. The Hadoop API consist of three different types of classes that include Common, MapReduce, and YARN. The complete detail defining the interface is provided in the Hadoop API javadoc. The complete set of documentation can be found at: https://hadoop.apache.org/docs/r2.6.0/api/index.html?overview-summary.html

Hadoop provides a package with multiple classes that supports the storage and processing through HDFS. The basic structure involves clusters consisting of nodes that can be managed by users or administrators. Further details may be found at: https://hadoop.apache.org/docs/r2.6.0/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html.

The Hadoop Java API allows software developers to manage Hadoop/HDFS storage and processing. Hadoop/HDFS allows for high-throughput access to application data and supports distributed processing of high-volume, high-velocity, high-variability data across clusters of computers for high performance, high availability. For more information, the complete Hadoop Java API documentation is available at: https://hadoop.apache.org/docs/r2.6.0/api/index.html?overview-summary.html

4.5 STORM

4.5.1 Overview

Apache Storm (https://storm.apache.org/) is a distributed real-time computation system for processing large volumes of high-velocity streaming data in real time. Storm makes it easy to reliably process unbounded streams of data, doing for realtime processing what Hadoop did for batch processing. Storm is simple and can be used with any programming language.

Storm has many use cases: realtime analytics, online machine learning, continuous computation, distributed RPC, ETL, and more. Storm is fast: a benchmark clocked it at over a million tuples processed per second per node. It is scalable, fault-tolerant, guarantees your data will be processed, and is easy to set up and operate.

Storm integrates with the queueing and database technologies you already use. A Storm topology consumes streams of data and processes those streams in arbitrarily complex ways, repartitioning the streams between each stage of the computation however needed.

4.5.2 Interfaces

4.5.2.1 Web GUI

The Storm Web GUI provides an overview of all Storm jobs running, their status, the data flow, and other statistics useful for assessing the state of the system. The Storm Web GUI also provides the ability to start/stop and control basic aspects of the Storm jobs.

The Storm Web GUI shows the inputs, outputs and processing loads, average times and percentage statistics. In the visualization, spout components are represented as blue, while bolts are colored between green and red depending on their associated capacity metric. The width of

the lines between the components represent the flow of tuples relative to the other visible streams. Detailed statistics are also available in a tabular view shown below. The main GUI is separated into three main pages: the first containing overall cluster information, the second containing detailed information on the topology, the third containing detailed information on components.

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	2	2	152.000	1	
3h 0m 0s	6	6	108.750	4	
1d 0h 0m 0s	28	28	148.958	24	
All time	28	28	148.958	24	

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
00.Filesystem-Path-Selector.hdfs	1	1	0	0	0.000	0	0				
00.Kafka-Reader.attachment	1	1	0	0	0.000	0	0				
00.Kafka-Reader.di	1	1	0	0	0.000	0	0				
00.Kafka-Reader.dpf	1	1	24	24	148.958	24	0				

Showing 1 to 4 of 4 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
00.Filesystem-File-Loader	1	1	0	0	0.000	0.000	0	0.000	0	0				
02.Processing	1	1	4	4	0.000	132.500	24	132.208	24	0	acs-sde-8cluster-dev1-node3.sd.spawar.navy.mil	6700	mil.army.inscom.dpf.mapresolve.resolver.ModelResolverException: Failed to resolve mapping at mil.army.inscom.dpf.mapresolve.MappingResolver.resolve(MappingResolver.java:156) at mil.army.inscom.dpf.u	Thur, 6 Jul 2017 13:48:35
GeoSOPersistenceBolt	1	1	0	0	0.000	67.750	4	67.500	4	0				
XX.Quarantine	1	1	0	0	0.000	0.000	0	0.000	0	0				

Figure 18 Storm Detailed Statistics

The Storm Web GUI provides monitoring and basic administrative information about Apache Storm jobs, including spouts, bolts, and topologies. Interaction is handled via mouse and keyboard to visualize status.

- Topologies - The logic for a realtime application is packaged into a Storm topology. A Storm topology is analogous to a MapReduce job. One key difference is that a MapReduce job eventually finishes, whereas a topology runs forever (or until you kill it, of course). A topology is a graph of spouts and bolts that are connected with stream groupings. These concepts are described below.
- Streams - The stream is the core abstraction in Storm. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion. Streams are defined with a schema that names the fields in the stream's tuples. By default, tuples can contain integers, longs, shorts, bytes, strings, doubles, floats, booleans, and byte arrays. You can also define your own serializers so that custom types can be used natively within tuples.
- Spouts - A spout is a source of streams in a topology. Generally spouts will read tuples from an external source and emit them into the topology
- Bolts - All processing in topologies is done in bolts. Bolts can do anything from filtering, functions, aggregations, joins, talking to databases, and more.

4.5.2.2 REST API

The ACS Storm REST API provides an interface, in the Representational State Transfer (REST) style, for accessing Storm cluster information, including status and configuration information.

The interface also allow a manager to start/stop topologies. This API is predominantly in support of monitoring and administration functions, and supports the Storm Web GUI. The API uses the http protocol and returns results in the JSON format. By using a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running.

The complete API is available at http://storm.apache.org/releases/1.0.4/STORM-UI-REST-API.html.

4.5.2.3 JAVA API

The Storm Java API provides packages and classes defining a comprehensive interface for Java developers to access the Storm capability, which supports the ACS Data Processing and Analytics Framework CI's Stream Processing Framework component. The API provides more than 2000 classes in more than 150 packages. The complete detail defining the interface is provided in the Storm API javadoc. In the figure below, the first page of the Storm API javadoc is shown. The complete set of documentation can be found at:
http://storm.apache.org/releases/1.0.4/javadocs/index.html

Further details may be found at: http://storm.apache.org/releases/1.0.4/javadocs/index.html

4.6 SPARK

4.6.1 Overview

Apache Spark (https://spark.apache.org/) supports distributed real-time batch processing. Apache Spark is a fast, in-memory data processing engine with elegant and expressive development APIs to allow data workers to efficiently execute streaming, machine learning or SQL workloads that require fast iterative access to datasets

4.6.2 Interfaces

4.6.2.1 Web GUI

The ACS Spark Web GUI will provide statistics similar to those provided by SparkContext to assist administrators in developing analytics related to the operation of their pre-deployed capabilities. Every SparkContext launches a web UI, by default on port 4040, that displays useful information about the application. This includes:

- List of scheduler stages and tasks
- Summary of RDD sizes and memory usage
- Environmental information.
- Information about the running executors

The Spark Web GUI is designed for ACS system administrators to monitor the status of batch jobs through the visualization of different statistics. From this GUI, then analytics can be derived through the connections with the other interfaces such as the Spark Java API and the Spark REST interface.

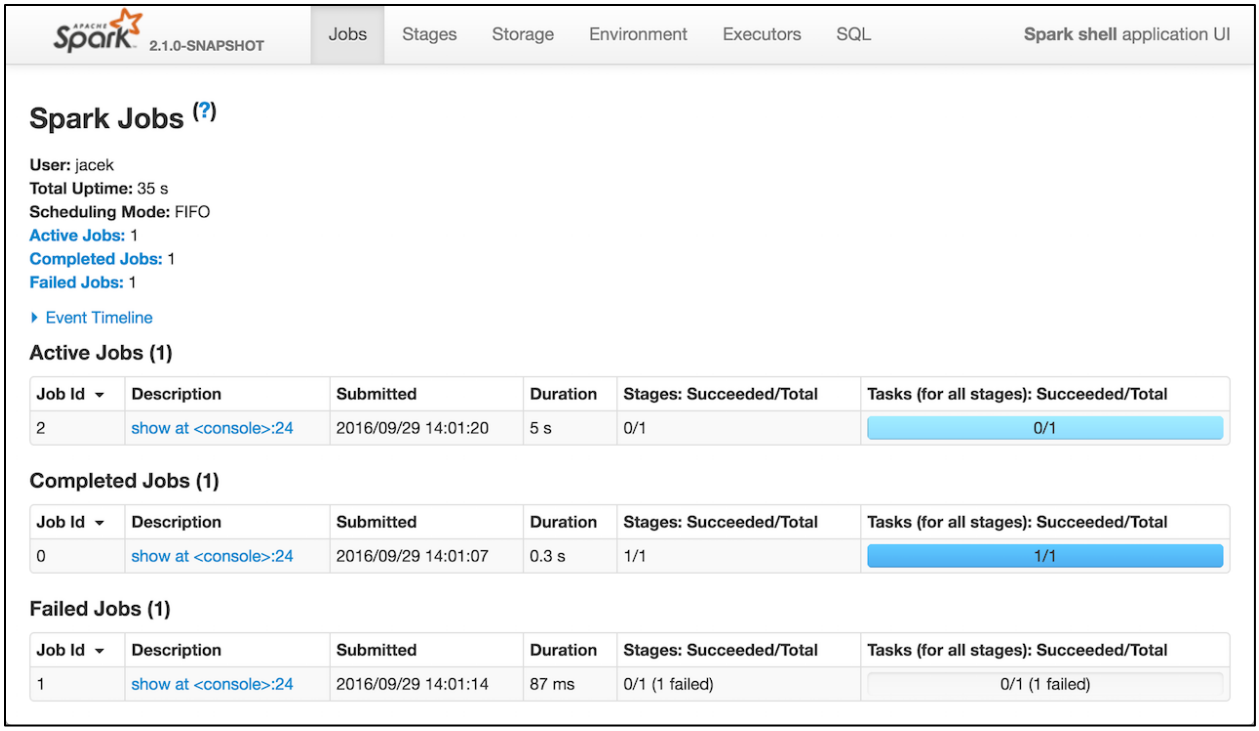


Figure 19 Spark Web GUI

4.6.2.2 Java API

The Spark Java API provides a comprehensive set of classes and methods enabling Java developers to develop applications performing sophisticated data analytics for batch and micro-batch processing. The Spark Java API package, org.apache.spark.api.java, is shown in the javadoc intro page below, providing a high-level overview of the classes included in this package. For example, the JavaDoubleRDD class provides over 100 methods, with another 75 inherited methods, all to manage the Resilient Distributed Datasets that are at the heart of running big data analytics on Spark.

This Spark Java API provides an interface for Java developers to write Java applications using ACS-provided Spark batch data analytics. The Spark Java API is the primary developer interface to the Batch Analytic Framework component of the ACS Data Processing and Analytics CI.

The Spark Java API provides access to all of the essential Spark capabilities. The full interface documentation is available at: https://spark.apache.org/docs/2.0.0-preview/programming-guide.html

4.7 INTELLEX

Fundamentally, IntellEx is an operator-on-the-loop data integration and smart alerting information system. It falls into the class of complex event processing (CEP) tools. IntellEx is an open, flexible platform with a variety of standards-based interfaces making it a good teammate within larger service oriented architectures, or as a standalone capability. It provides for multi-resolution configuration making it tunable from minimally trained end users, to moderately trained knowledge managers, and ultimately to well-trained software developers.

4.7.1 Platform Capabilities

- **Data Integration (DI)** – The term "data integration" is used to reference IntellEx's capability to provide for Generalized Data Ingest, Conditioning, Indexing, and Warehousing via configurable workflows. Data ingested by IntellEx comes from various sources available locally, or in the cloud.
- **"Smart Push"** Condition of Interest Alerting – Context-based alerting of user, role, mission, or machine relevant information. Interest Models define consumer-centric views of information relevancy to include what type of data or events the consumer is interested in, what content the alerts should contain, and how often they should be alerted to updates or changes to information matching their Interest Model.
- **Mission Integrated Analytics** – Automated Fleet Warfighting Readiness Assessment is an example use of IntellEx platform automation and mission-relevant Interest Models to produce estimates of how ready Naval platforms and units are to execute specific mission types, against specific adversaries, in specific environmental conditions. Sensor data fusion and monitoring is an example of using IntellEx to identify conditions or situations of interest at the tactical edge of an Army Company, resulting in the issuance of notifications ("tips & cues") to soldiers.
- **Composite Information Exposure (CIE)** – Standards-based interfaces provide both synchronous and asynchronous access to integrated datasets within the IntellEx Knowledge Base. Integrated datasets are composite information products of possibly many distributed and heterogeneous sources. DI and CIE have been used to centralize data cleansing, greatly facilitate downstream exploitation of diverse datasets, and reduce redundant synchronization points within Service Oriented Architectures.
- **Operator-On-The-Loop** automation – Robust web-based user interfaces provide the ability for novice and expert users alike to build, test, and maintain IntellEx from source-to-product. The user may create, subscribe, and view alert feeds, and define and view real-time dashboards of related information.

4.7.2 System Overview

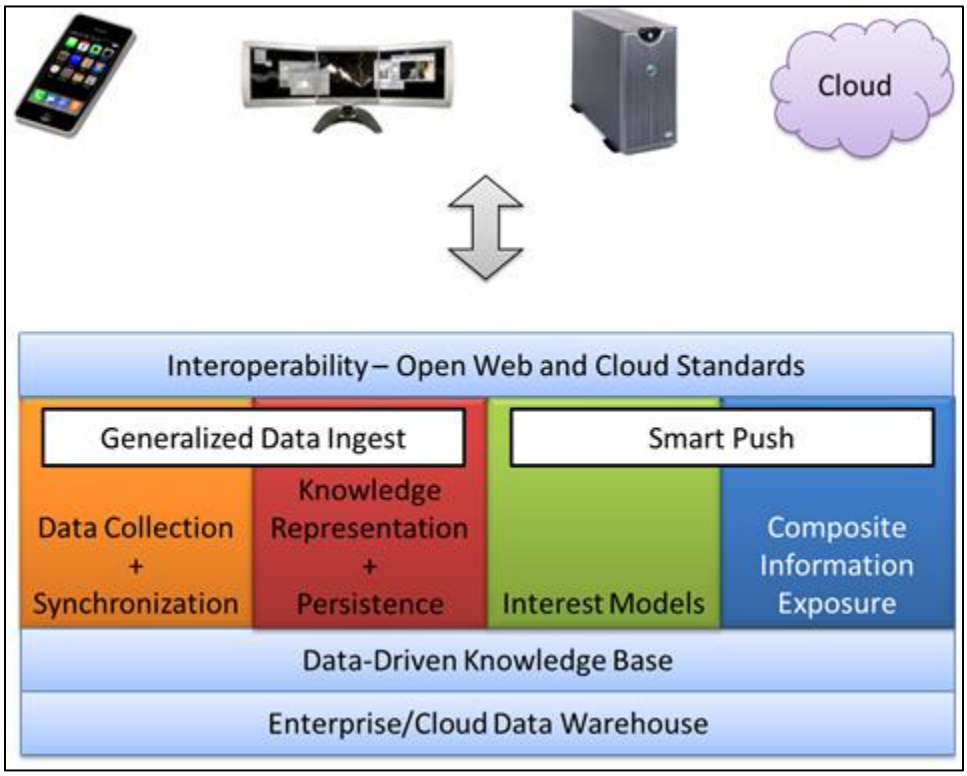


Figure 20 IntellEx Functional Architecture

The figure above shows a generalized view of the IntellEx architecture. Within a standardized framework IntellEx provides 4 primary functions: Data Collection and Synchronization, Knowledge Representation and Persistence, Interest Models, and Composite Information Exposure. The role of each of these is described below:

4.7.2.1 Data Collection and Synchronization

Distributed information processing, such as that done within Service Oriented Architectures and across the cloud, requires enterprise wide Patterns and Practices (P&Ps) to provide a seemingly integrated aggregate capability. One such P&P is for Data Initialization and Synchronization. This P&P defines how distributed services share data across a Local or Wide Area Network. While decomposition of capability into modular independent services can maximize reuse, it also maximizes a dependency on synchronization. An approach to synchronization must be taken on a case-by-case basis in light of heterogeneous requirements and constraints between business applications, network environments, and infrastructure. IntellEx's Data Collection and Synchronization module provides configurable workflows such that Generalized Data Ingest is robust to most environments out-of-the-box or with minimal new development needed. This guide describes how to create configurations necessary to meet the domain-specific needs of data ingestion for IntellEx. The figure below illustrates the architecture of this module.

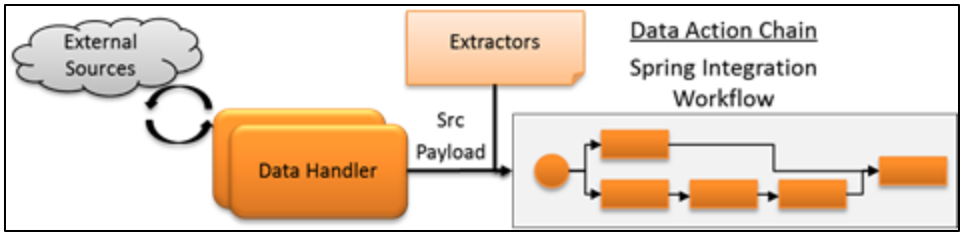


Figure 21 Generalized Data Ingest Using Workflows

4.7.2.2 Knowledge Representation and Persistence

The data ingested by the Data Collection and Synchronization module of IntellEx is stored in the internal knowledge base and persisted to an external database. Models (called Fact Types) define the properties of data objects that have been extracted from raw data. The instances of the data objects are called Facts. The granularity of fact types is domain and use case specific. As a general rule, incoming data should be extracted into "atomic" facts that are the smallest unit of data that will be processed as a whole. So, for example, if multiple address books are ingested from a data source, two fact types should be used: (1) to represent the address book, and (2) to represent each address.

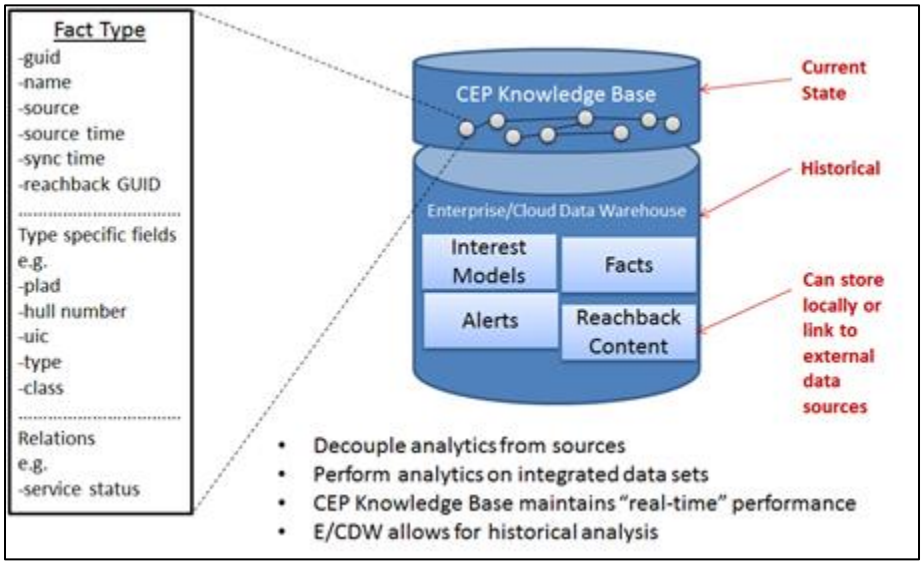


Figure 22 Decoupling Analytics from Source

4.7.2.3 Interest Models

Once data has been loaded into the IntellEx knowledge base, business rules will be applied to monitor for conditions of interest, and data outputs will be created (alerts and notifications) when those conditions are realized. The rules and related configurations are referred to as Interest Models. These models comprise the heart of the system and its CEP processing.

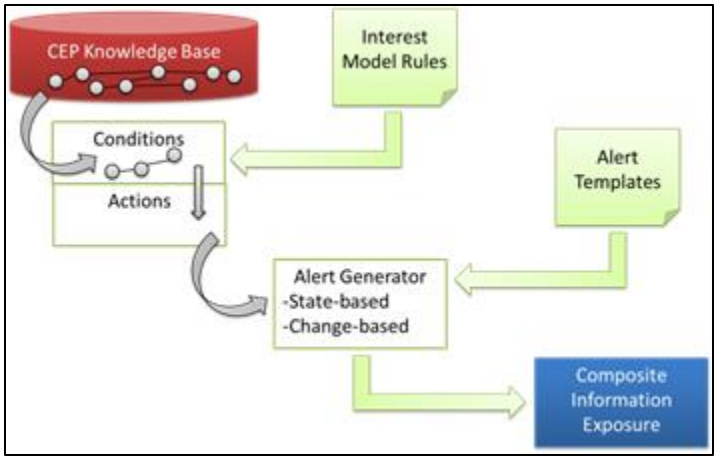


Figure 23 Interest Models - Data Flow

4.7.2.4 Data Exposure

After creation of alerts and notifications, they need to be disseminated to interested parties, and made available for retrieval by downstream processes. The Data Exposure capabilities of IntellEx provide numerous ways to send alerts to various endpoints (such as email), or "slice" into the alert archive to retrieve data of interest.

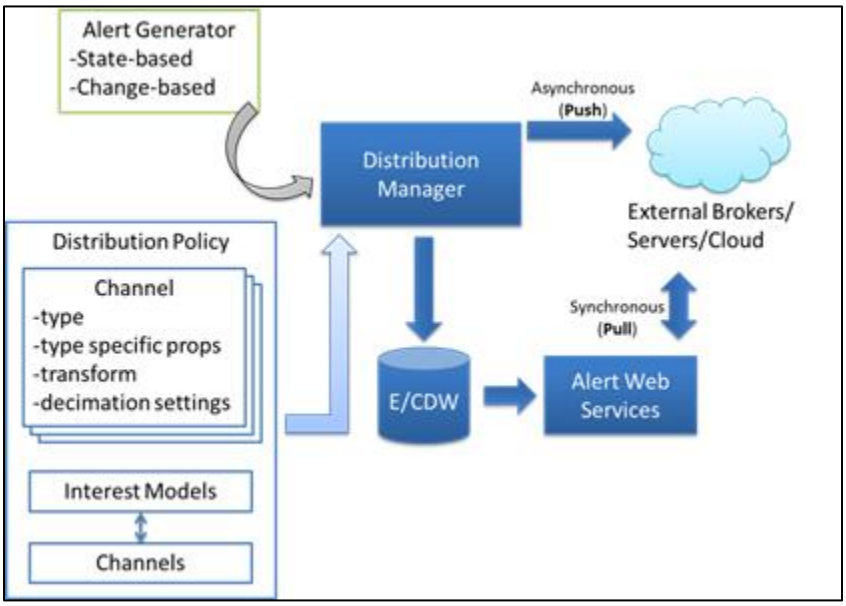


Figure 24 Alert Generation Process

4.7.3 Interfaces

4.7.3.1 Web GUI

In order to create a configuration, one of the user interfaces (ConfigUI or IXDesigner) can be used. A sample GUI screenshot is shown in the following figure

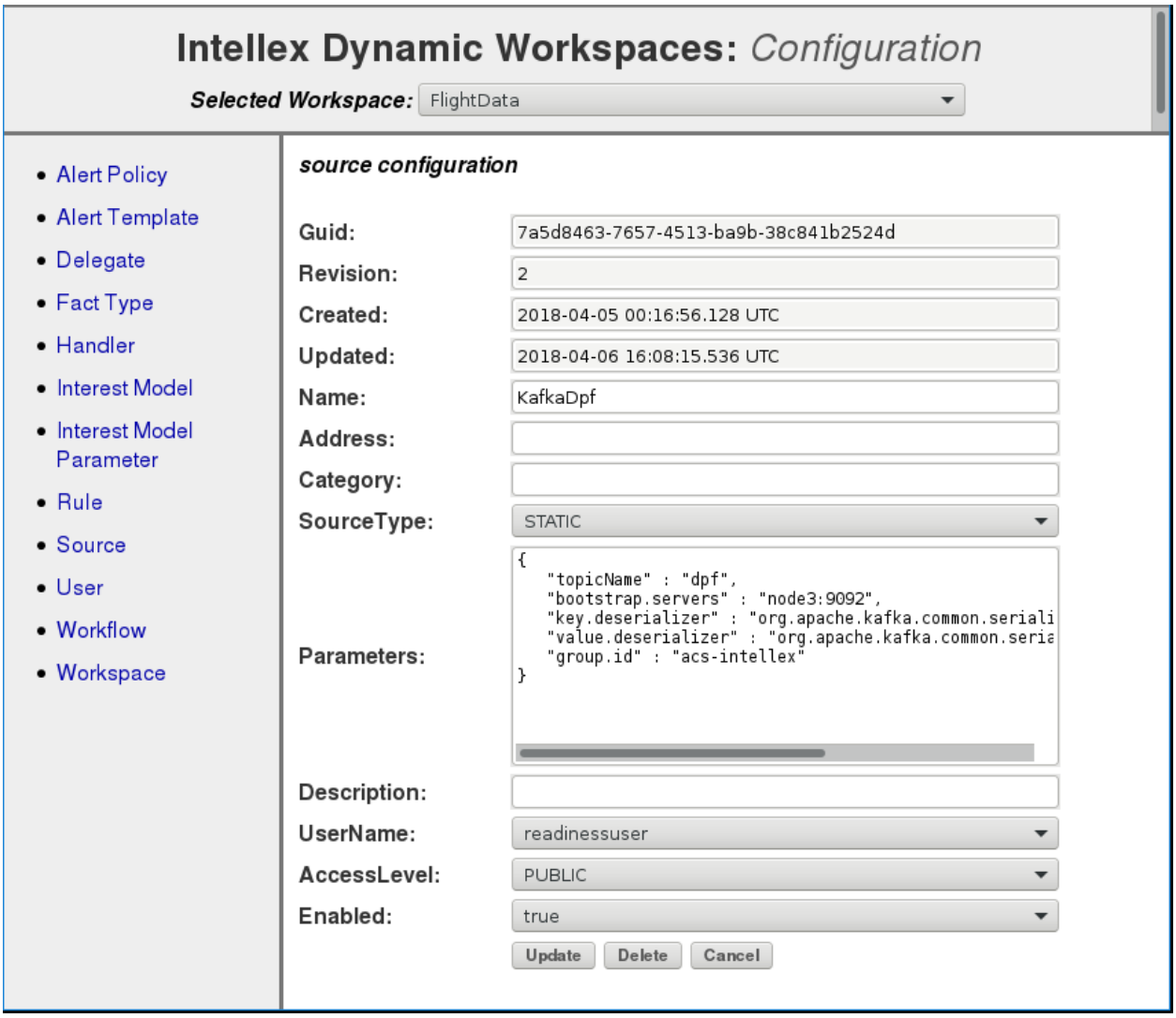


Figure 25 IntellEx ConfigUI

4.8 DRIF

Data Representation Integration Framework (DRIF) stores data in 4 logical segments as noted in the figure below. Segment 0 is implemented across: Accumulo tables (small Artifacts are stored in line with their descriptions), HDFS (large Artifacts are stored on the HDFS), and elsewhere. Segment 1 is implemented in Accumulo tables, and segments 2 and 3 are implemented in Rya.

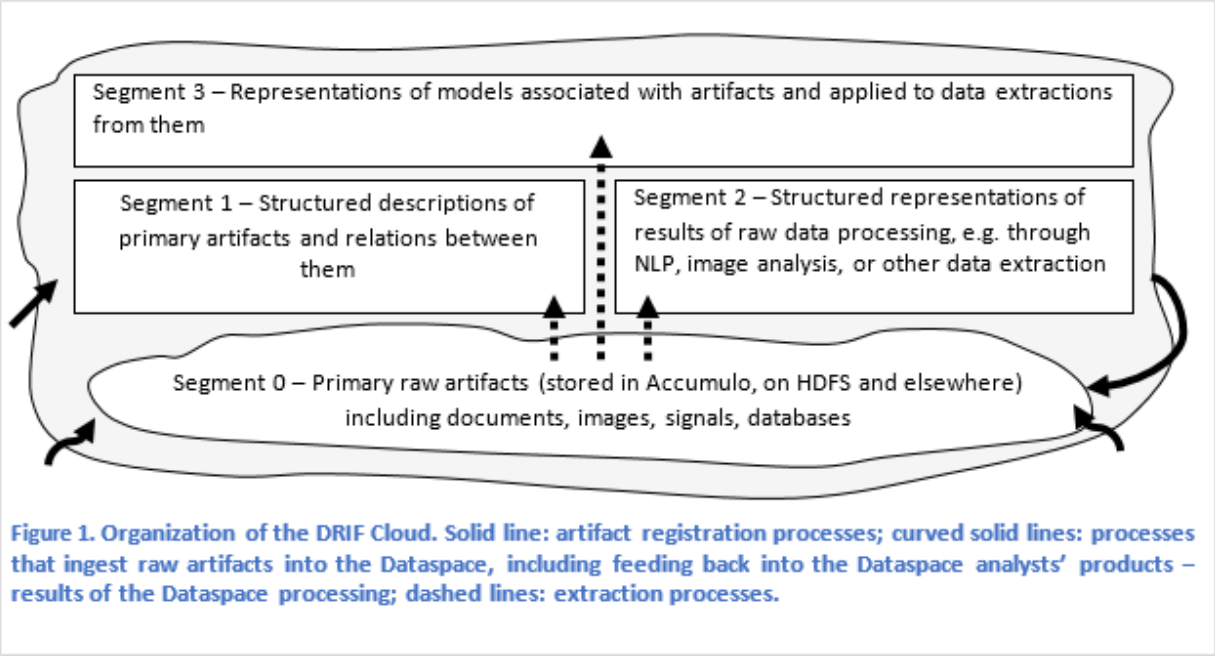


Figure 26 Organization of the DRIF Cloud

Resource Description Framework (RDF) Ingest into DRIF has been integrated into Data Processing Framework (DPF). Given a trig or ttl file, a set of transforms are applied to the content to extract instance and model information from the contents. The information is then transformed into DRIF Terms, Statements, Concepts, Concept Associations, and Predicates.

When RDF data is ingested into DRIF, the native RDF graph is transformed and normalized native RDF Uniform Resource Identifier (URI)s are converted to DRIF URIs. Additionally, literals are converted into nodes with an associated sign equal to the value of the original literal. These transformations are desirable in that they promote data integration, literals can now be directly compared and associated since they have been essentially normalized. A view of this transformation (the original and resulting graph structures) can be seen below.

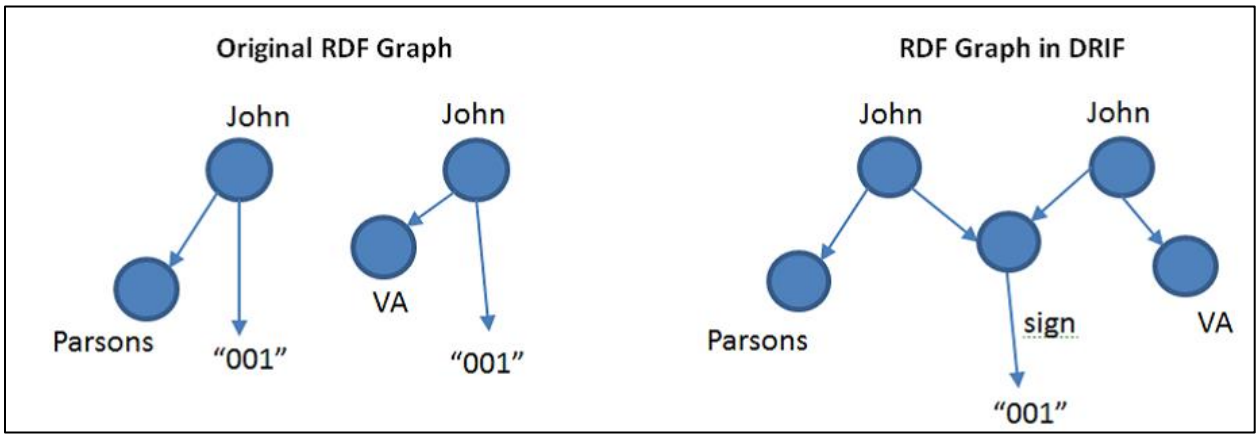


Figure 27 DRIF RDF Transformations

The DRIF Java API can be utilized through a native Java Client that works with DRIF indices on Accumulo, through a REST service, or through an RPC client with a Thrift RPC service. The DRIF API provides different services for retrieving data which include DRIF data search service, entity search service, graph service, model service, search service, and artifact description service. Other provided services include creating DRIF services through the Context API. Through these services users are able to interact with queries, different data artifacts, and related entities.

4.9 OPENSHIFT

OpenShift is a computer software product from Red Hat for container-based software deployment and management. It is a supported distribution of Kubernetes using Docker containers and DevOps tools for accelerated application development.

4.9.1 Interfaces

4.9.1.1 Web-Based Console

The OpenShift platform ships with a feature-rich web console that allows developers to perform the actions needed to deploy and run existing source code projects. For example, one of the features is a graphical representation of an application that consists of multiple containers all load balanced, while also using a database as a backend storage engine.

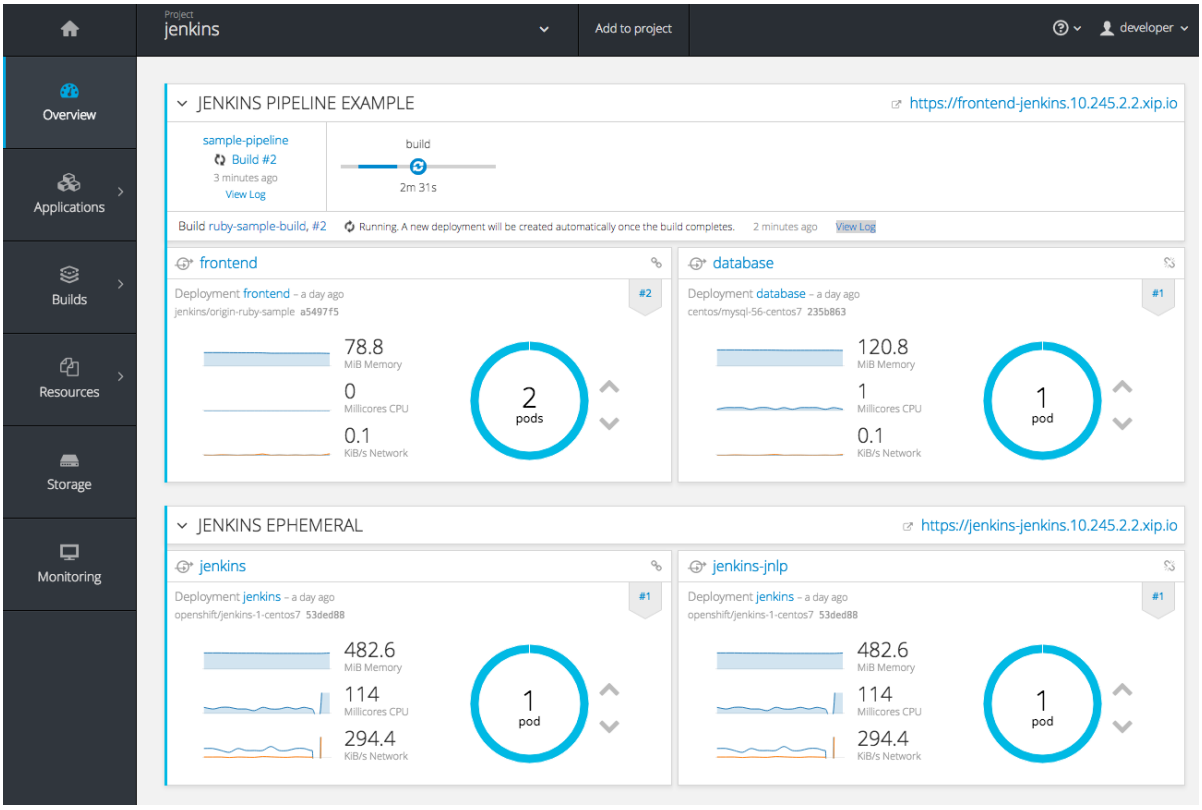


Figure 28 OpenShift Web Console

Other tasks a developer would normally perform via the web console is scaling the application containers, creating projects, viewing log files, viewing the memory and CPU utilization of a container, and other common functions. The web console is a great tool as it provides a single interface for all aspects of the development project.

4.9.1.2 Command-Line Tool

OpenShift also provides a Command Line Interface (CLI) that exposes commands for managing applications, as well as lower level tools to interact with each components of your system. This tool, called oc, is a single binary executable provided for all major operating systems. The tool can be used to perform any operations that can be accomplished via the web console. The greatest benefit of using the command-line tool is that you have a single executable to perform all operations instead of having to interface directly with multiple tools such as the ones provided with Docker and Kubernetes.

4.9.1.3 Source-to-Image

OpenShift comes with the S2I (Source-to-Image) open source project that the OpenShift Team created as part of the OpenShift platform. S2I is a toolkit and workflow for building reproducible Docker images from source code. S2I-based Docker images allow developers to interact with the platform from a pure source code perspective. OpenShift only needs to know the URL of your git repository and then the platform takes care of the rest. S2I products ready-to-run images by injecting source code into a Docker container and letting the container prepare that source code for execution. When a new project and container is created, the platform matches a base image of the desired runtime up with your source code, performs a build, and then creates a new Docker image on the fly.

4.9.1.4 Integrated Logging and Metrics

An important aspect of any software development project is the ability to troubleshoot issues as they arise. One of the first places a developer looks at when beginning the debug process is the application logs for the project. OpenShift provides a comprehensive view of the application logs including runtime logs, build logs, and deployment logs. Having these through the web console and command line tool greatly simplifies the troubleshooting process when using containers.

The OpenShift platform also includes support to log metrics about an application's utilization of system resources. Details such as memory and CPU utilization for a container can be logged.

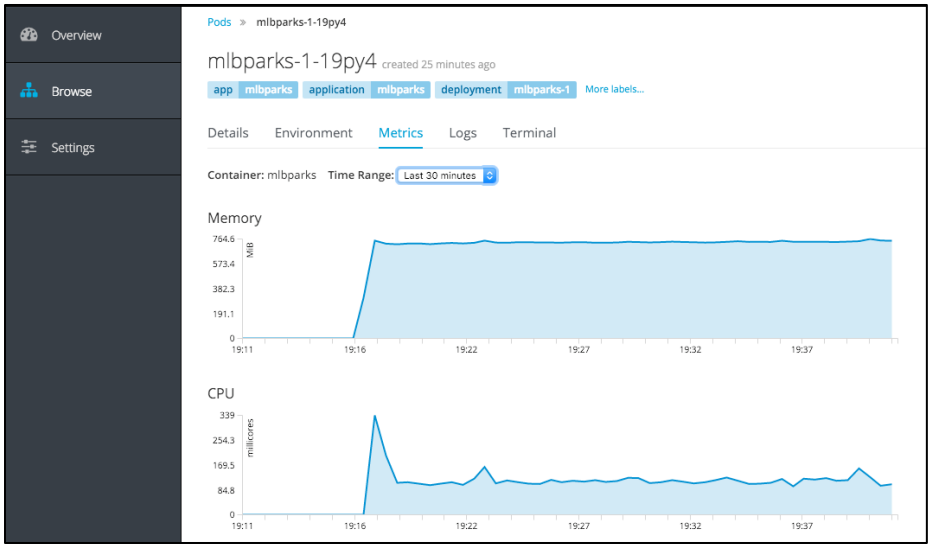


Figure 29 Logging and Metrics

4.10 ACCUMULO

Apache Accumulo is a highly scalable structured store, written in Java, operates over HDFS, and developed originally at NSA. Accumulo supports efficient storage and retrieval of structured data, including queries for ranges, and provides support for using tables as input and output for MapReduce jobs. Accumulo offers features important for intelligence and DoD applications, such as find-grained, cell-level security.

4.10.1 Interfaces

4.10.1.1 Web GUI

The Accumulo Web GUI addresses primarily the monitoring function of ACS Data Representation and Persistence Framework CI. The Accumulo Web GUI provides monitoring and basic administrative information about Hadoop, including disk storage used, capacity and the live nodes. Interaction is handled via mouse and keyboard to visualize status.

The Accumulo Web GUI is designed for ACS Data Storage Managers to monitor the status of the storage system. Developers will rely primarily on other interfaces for the use and management of datastores in their applications. The ACS Accumulo Web GUI provides statistics for the overall Accumulo system of clustered nodes, including the disk storage used, the number of tablets, entries, lookups and uptime. See the following figure.

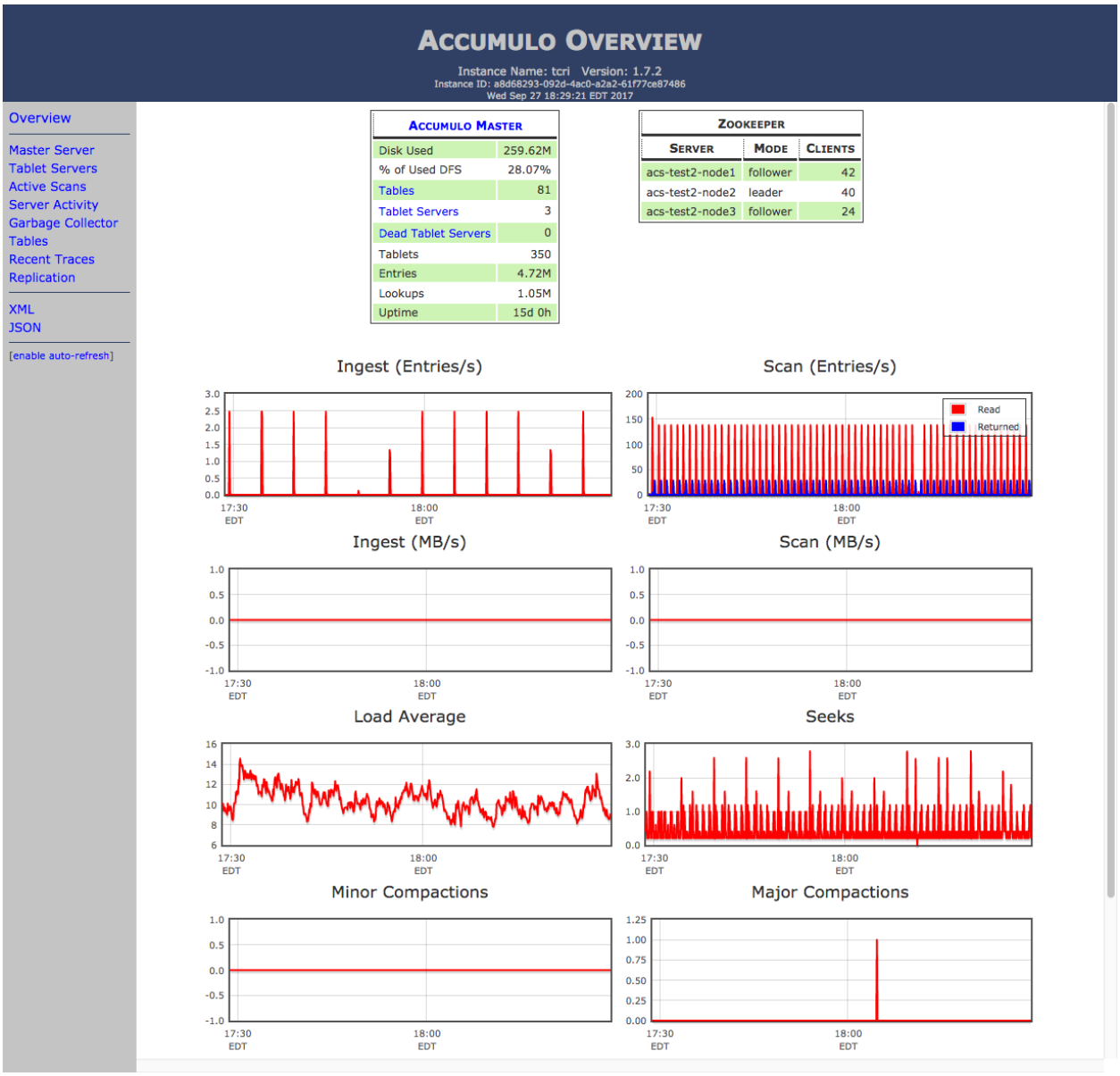


Figure 30 Accumulo Web GUI

4.10.1.2 Java API

The Accumulo Java API provides packages and classes defining a comprehensive interface for Java developers to access Accumulo storage capability, which supports the ACS Data Representation and Persistence Framework component. The API provides numerous classes in 14 different packages. Accumulo operates over HDFS, which is part of Hadoop, and utilizes MapReduce packages as well. The complete detail defining the interface is provided in the Accumulo API javadoc. The complete set of documentation can be found at: https://accumulo.apache.org/1.7/apidocs/

Accumulo is designed as a distributed data storage and retrieval system that supports the storage capability for ACS. For storage a data model is used that consists of key-value pairs, but is not a fully relational database. The basic structure involves TabletServers, a Garbage Collector

process, a server, and clients that can be managed by users or administrators. Further details may be found at: https://accumulo.apache.org/1.7/accumulo_user_manual.html

4.10.1.3 CLI

The Accumulo CLI provides a simple shell that can be used to examine the contents and configuration settings of tables, insert/update/delete values, and change configuration settings. The shell will prompt for the corresponding password to the username specified and then display a prompt. From the prompt, one can type 'help' for a list of the available commands. For information on the use of the interface, see: https://accumulo.apache.org/1.7/accumulo_user_manual.

The Accumulo CLI addresses primarily the configuration and test function of ACS Data Representation and Persistence Framework CI. The Accumulo CLI provides the following basic administration functions:

1. Table Maintenance:
    a. Create tables
    b. Delete tables
    c. Update tables
2. User administrations:
    a. Add users
    b. Remove users
    c. Grant privileges to users
3. Update Configuration Settings of
    a. Table configuration
    b. Instance options

Interaction is done via an interactive shell or at command line that can be batched up in shell scripts.

4.11 SOLR

Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, and centralized configuration. Solr powers the search and navigation features of many of the world's largest internet sites.

The Solr Web GUI provides monitoring and basic administrative information about Solr, including instance version, status, jvm processors and arguments, system and jvm memory status. Interaction is handled via mouse and keyboard to visualize status as shown in the following figure.

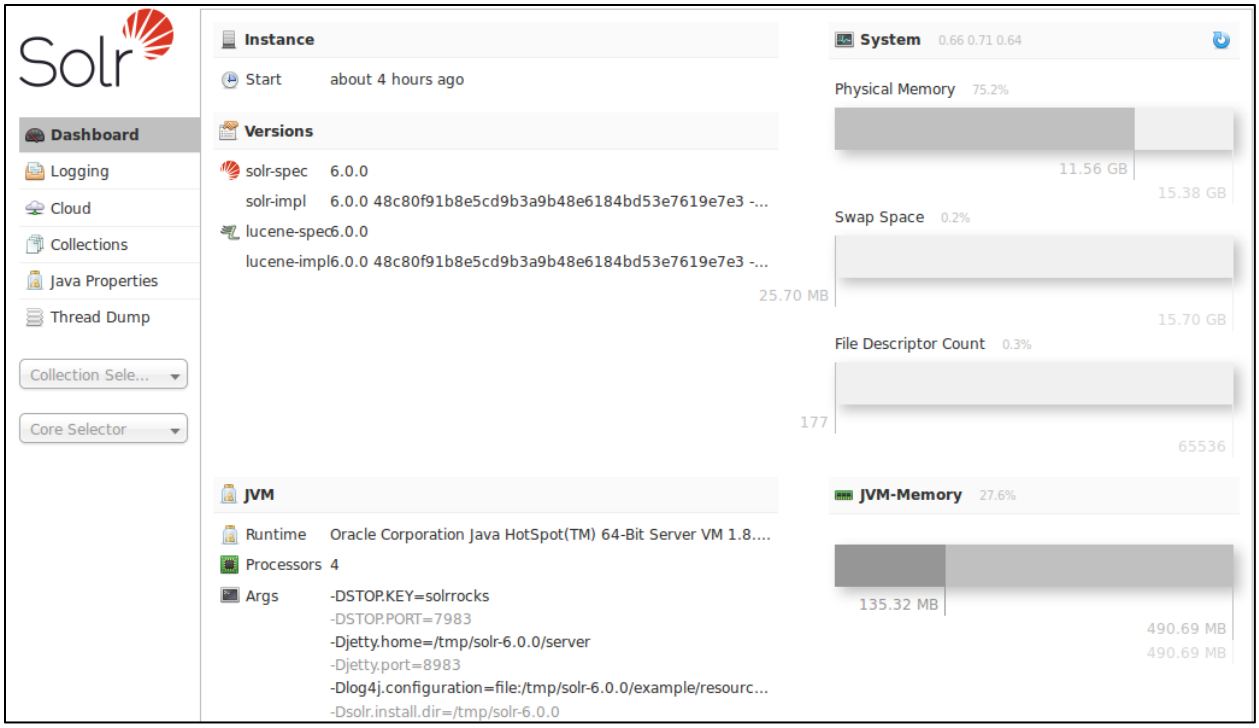


Figure 31 Solr Web GUI

A left-side of the screen is a menu under the Solr logo that provides the navigation through the screens of the UI. The first set of links are for system-level information and configuration and provide access to Logging, Collection/Core Administration, and Java Properties, among other things. At the end of this information is at least one pulldown listing Solr cores configured for this instance.

The center of the screen shows the detail of the option selected. This may include a sub-navigation for the option or text or graphical representation of the requested data. See the sections in the Solr Admin UI guide for each screen for more details. Under the covers, the Solr Admin UI re-uses the same HTTP APIs available to all clients to access Solr-related data to drive an external interface.

For more information on the Solr Web GUI, see the ACS Developer's Guide and the Solr Admin UI documentation.

4.12 RYA

Rya is a scalable RDF data management system built on top of Accumulo. Rya uses novel storage methods, indexing schemes, and query processing techniques that scale to billions of triples across multiple nodes. Rya provides fast and easy access to the data through SPARQL, a conventional query mechanism for RDF data. The Rya Java API allows developers to use this functionality to process large sets of data stored in this type of framework.

RDF is a World Wide Web Consortium (W3C) standard used in describing resources on the Web. The smallest data unit is a triple consisting of subject, predicate, and object. Using this framework, it is very easy to describe any resource, not just Web related. For example, a record for a 747 model aircraft with flight number A123, owned by United Airlines, arriving in San

Diego could be represented as (A123, rdf:model, 747), (A123, rdf:airline, United Airlines), (A123, rdf:arriving, SAN). In general, RDF is an open world framework that allows anyone to make any statement about any resource, which makes it a popular choice for expressing a large variety of data.

For a comprehensive overview of Rya, see the documentation at: https://github.com/apache/incubator-rya/blob/master/extras/rya.manual/src/site/markdown/index.md.

4.13 POSTGRESQL

PostgresSQL is an object-relational database management system (ORDBMS), supporting complex queries, foreign keys, triggers, updatable views, and transactional integrity. The Java Database Connectivity (JDBC) is an application programming interface for the Java programming language, which defines how a client may access a database. The PostgresSQL Java JDBC API enables clients to connect to PostgresSQL using Java. The API is written in Java and communicates in the PostgreSQL native network protocol and is available at: https://jdbc.postgresql.org/documentation/head/index.html.

4.14 GEOMESA

4.14.1 Overview

GeoMesa is an Apache licensed open source suite of tools that enables large-scale geospatial analytics on cloud and distributed computing systems, letting you manage and analyze huge spatio-temporal datasets.

GeoMesa does this by providing spatio-temporal data persistence on top of the Accumulo distributed column-oriented databases for massive storage of point, line, and polygon data. It allows rapid access to this data via queries that take full advantage of geographical properties to specify distance and area. GeoMesa also provides support for near real time stream processing of spatio-temporal data by layering spatial semantics on top of the Apache Kafka messaging system.

Through a geographical information server such as GeoServer, GeoMesa facilitates integration with a wide range of existing mapping clients by enabling access to its databases and streaming capabilities over standard Open Geospatial Consortium (OGC) APIs and protocols such as Web Feature Service (WFS) and Web Map Service (WMS). These interfaces also let GeoMesa drive map user interfaces and serve up data for analytics such as queries, histograms, heat maps, and time series analyses.

GeoMesa features include the ability to:

- Store gigabytes to petabytes of spatial data (tens of billions of points or more)
- Serve up tens of millions of points in seconds
- Ingest data faster than 10,000 records per second per node
- Scale horizontally easily (add more servers to add more capacity)
- Support Spark analytics

- Drive a map through GeoServer or other OGC Clients

4.14.2 Architecture Overview

GeoMesa supports scalable data storage technologies like Apache Accumulo, as well as the Apache Kafka message broker for streaming data. Apache Storm lets you define information sources and manipulations to allow batch distributed processing of streaming data with GeoMesa, and a GeoMesa environment can also take advantage of Apache Spark to do large-scale analytics of stored and streaming data.

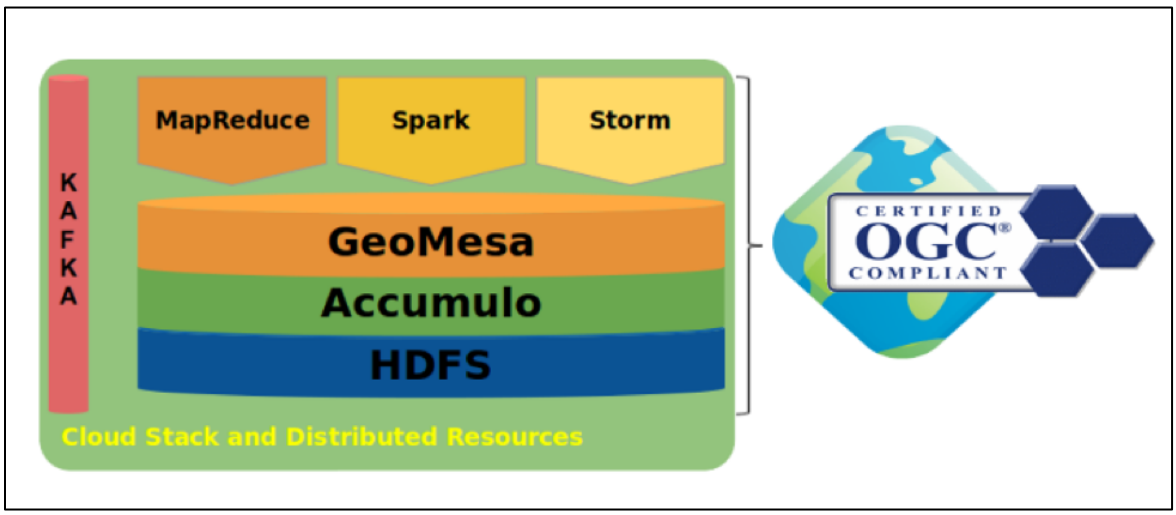


Figure 32 GeoMesa Architecture

4.14.3 GeoMesa and GeoServer

GeoMesa's GeoServer plugins let GeoServer work with data stored using GeoMesa tables. The plugin maps the GeoServer capabilities associated with data sources to the GeoMesa formatted tables and provides a number of optimizations to support specific use cases.

If an application already uses GeoServer, integration with GeoMesa is simply a matter of adding a new datastore to GeoServer and updating the application's configuration.

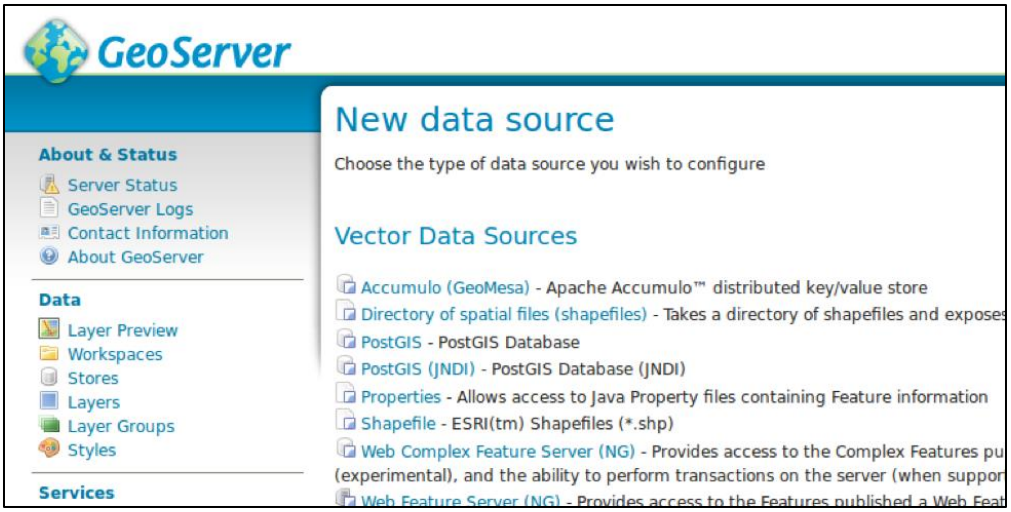


Figure 33 GeoServer

4.14.4 Integration with GeoMesa

GeoMesa implements GeoTools interfaces to provide programmatic access, and HTTP access to the following Open Geospatial Consortium standards:

- Web Feature Service (WFS)
- Web Mapping Service (WMS)
- Web Processing Service (WPS)
- Web Coverage Service (WCS)

Multiple frameworks may be used for streaming and batch ingestion of data. These include the GeoMesa command line tools, map-reduce jobs with Apache Hadoop, and real-time topologies running on Apache Storm. The following diagram shows one possible ingest architecture:

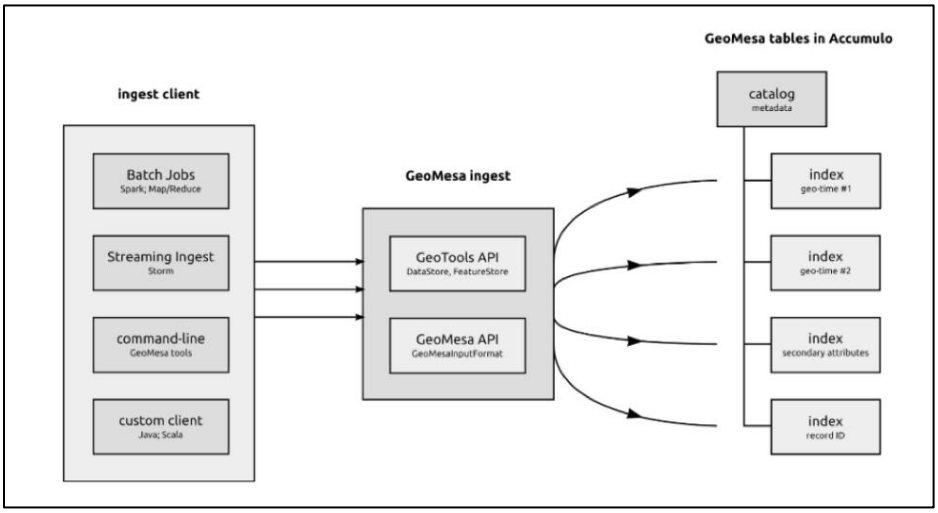


Figure 34 Possible Ingest Architecture

The following shows one possible query architecture, in which the GeoTools and GeoMesa APIs mediate the use of Accumulo iterators for external query clients:

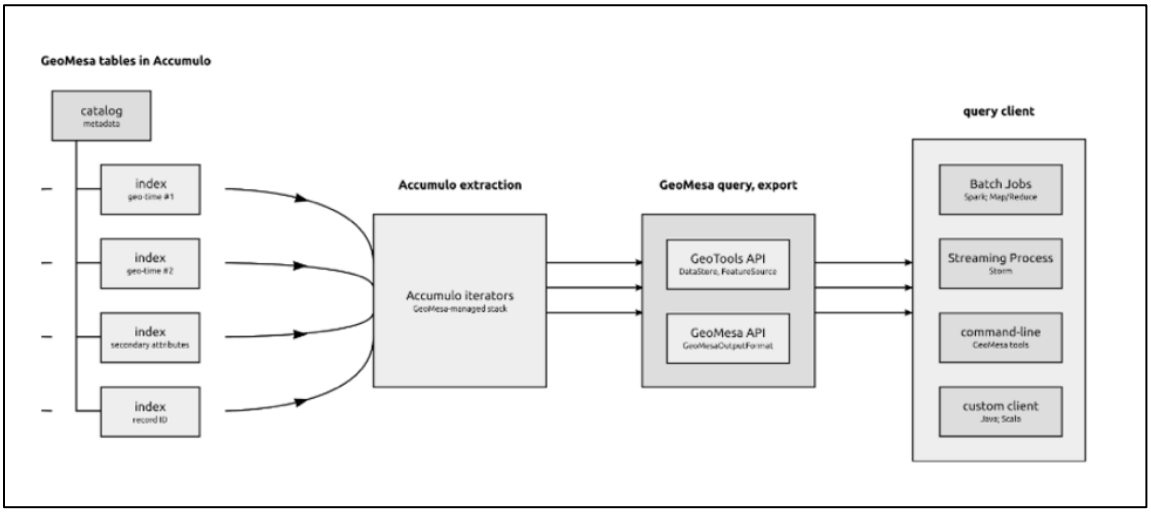


Figure 35 Possible Query Architecture

4.14.5 GeoMesa Command-Line Tools

- GeoMesa provides a set of command-line utilities that make it easy to ingest and query data without writing any code. Commands available for most back-ends:
- Environment and Status Commands
- Schema Commands
- Ingest Commands
- Query and Export Commands
- Analytic Commands
- Defining Simple Feature Types
- Defining Simple Feature Converters
- Logging Configuration
- Remote File System Support
- GeoMesa Scala Console

Specific back-end implementation commands:

- Accumulo Command-Line Tools
- HBase Command-Line Tools
- Bigtable Command-Line Tools
- Cassandra Command-Line Tools
- Kafka Command-Line Tools
- Lambda Command-Line Tools

- FileSystem Command-Line Tools

For more in on GeoMesa see http://www.geomesa.org/documentation/index.html
Command Line Tools are documented here:
http://www.geomesa.org/documentation/user/cli/index.html

4.14.6 Geo-Indexing Topology

The geo-indexing topology works alongside DPF, and can index data read from DRIF, Structured Objects, and Depot. Its two main responsibilities are:

1. when the topology is started, try to identify all geographic feature types, and register them all with GeoMesa data stores
2. as data flow through DPF, identify those entities which are geographic, convert them to GeoTools' SimpleFeature equivalents, and route them to one or both of the following destinations:
    a. Accumulo for long-term persistence
    b. the GeoMesa near-real-time cache in Storm (a Kafka-backed GeoMesa data store)

4.15 ARCGIS

4.15.1 Overview

The primary products chosen to support the Geospatial Data Access Layer & GIS Services CI are:

- ArcGIS Server: Provides mapping and geospatial capabilities via standard OGC standard web services
- ArcGIS Portal: Provides capability to create, manage, and share geospatial data and other contextual information across applications
- GeoEvent Processor and GeoExtensions: Provides real-time event processing of geospatial data

These products together support storing, processing, editing, utilizing, and sharing geospatial data. These products were designed to be scalable, reliable and fault tolerant to support management of big geographic datastores. ArcGIS is a leading commercial mapping solution which underlies the National Geospatial-Intelligence Agency (NGA) Commercial Joint Mapping Toolkit (CJMTK). CJMTK enables NGA to manage, analyze, visualize, and disseminate geospatial intelligence (GEOINT) for the command, control, and intelligence (C2I) community of the US Department of Defense. The ACS program is leveraging the ArcGIS and CJMTK for the design benefits of reuse, total cost of ownership, COTS, proven performance, and support for Open Geospatial Consortium (OGC) web map standards.

4.15.2 Interfaces

4.15.2.1 Admin Web GUI

ACS provides a web user interface for the management of ArcGIS Server. The interface supports administration to include starting and stopping of services, adding and removing machines from

an ArcGIS Server cluster, backup and restarting ArcGIS Server configuration, defining and building map caches, etc.

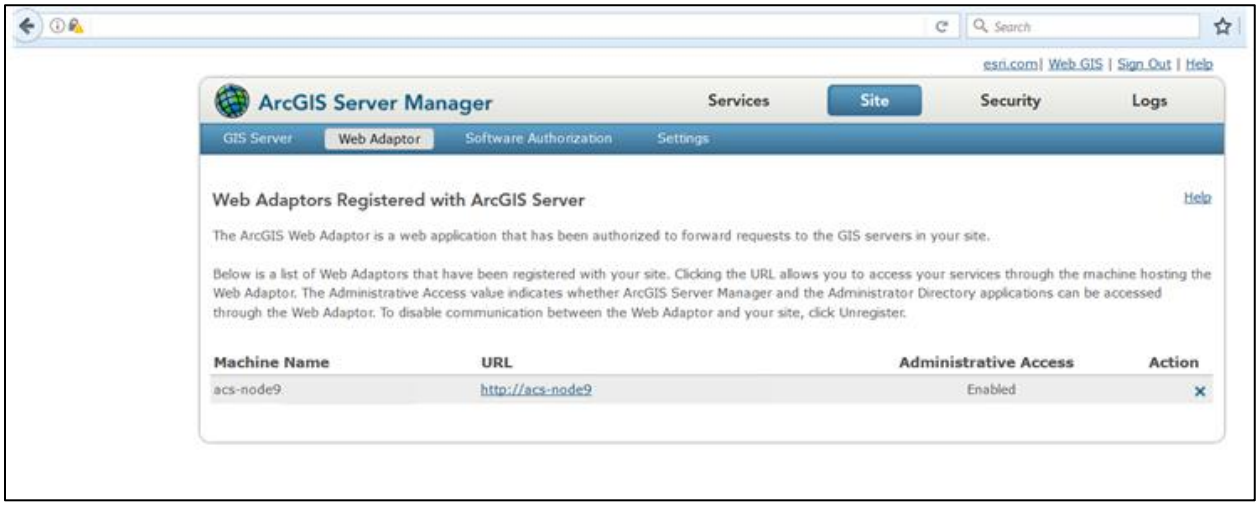


Figure 36 ArcGIS Admin Web GUI

4.15.2.2 Admin CLI

ACS provides a command line interface for ACS internal use only to support administration of ArcGIS Server. The commands available in the CLI are in part provided by a set of Python scripts.

Some of the CLI capabilities are:

- Start and stop services
- Add and remove machines from a cluster
- Publish services from a service definition (.sd) file
- Back up or restore a site configuration
- Define new map caches
- Build and delete map caches
- Convert map caches between exploded and compact formats
- Scan ArcGIS Server for security best practices

For further details see http://server.arcgis.com/en/server/latest/administer/windows/arcgis-server-command-line-utilities.htm.

4.15.2.3 Portal

ArcGIS Portal is a web based graphical user interface that supports ACS consumer application users with sharing of geospatial and related contextual data.

The Portal Interface brings together all the geographic information in the ArcGIS platform and shares it. The functionalities include:

- Create, save and share web maps and scenes
- Create and host web mapping apps

- Search for GIS content
- Create groups for sharing GIS information

The Portal Interface includes the following:

- Portal Website: user can sign in to the portal website to search, use, and share content
- Map Viewer: provide a canvas in which GIS information and services can be combined to make a map
- Web AppBuilder: an embedded interactive app for designing and building web applications from web maps. Applications are HTML-based.
- Scene Viewer: an application for viewing 3D geospatial content. The scene viewer works with browser that support WebGL for rendering 3D graphics.
- ArcGIS apps: user-friendly applications that interact with web maps and scenes from the Portal Interface

Further details may be found at http://server.arcgis.com/en/portal/.

4.15.2.4 Web Map Service

Web Map Service (WMS) produces maps of spatially referenced data dynamically from geographic information. This International Standard defines a “map” to be a portrayal of geographic information as a digital image file suitable for display on a computer screen. A map is not the data itself. WMS-produced maps are generally rendered in a pictorial format such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats.

4.15.2.5 Web Feature Service

This interface allows for the use of the Web Feature Service (WFS) that is available through ArcGIS. It allows applications to access geographic features from a map or enterprise geodatabase and returns actual features with geometry and attributes that clients can use in any type of geospatial analysis. In addition, filters are supported for spatial and attribute queries on the data. More information about functional allocation is available on the ArcGIS documentation page at: http://server.arcgis.com/en/server/latest/publish-services/linux/wfs-services.htm.

Data transactions for WFS consist of five different types of operations:

- Discovery operations allow the service to be interrogated to determine its capabilities and to retrieve the application schema that defines the feature types that the service offers
- Query operations allow features or values of feature properties to be retrieved from the underlying data store based upon constraints, defined by the client, on feature properties
- Locking operations allow exclusive access to features for the purpose of modifying or deleting features
- Transaction operations allow features to be created, changed, replaced and deleted from the underlying data store
- Stored query operations allow clients to create, drop, list and described parameterized query expressions that are stored by the server and can be repeatedly invoked using different parameter values

These types are then broken down into eleven separate operations:

- GetCapabilities (discovery operation)
- DescribeFeatureType (discovery operation)
- GetPropertyValue (query operation)
- GetFeature (query operation)
- GetFeatureWithLock (query & locking operation)
- LockFeature (locking operation)
- Transaction (transaction operation)
- CreateStoredQuery (stored query operation)
- DropStoredQuery (stored query operation)
- ListStoredQueries (stored query operation)
- DescribeStoredQueries (stored query operation)

More information about transactions and the specifications are available on the ArcGIS documentation page at: http://docs.opengeospatial.org/is/09-025r2/09-025r2.html#1

4.15.2.6 Web Coverage Service

Web Coverage Service (WCS) provides a simple HTTP interface for requesting geographic data represented as a binary coverage file. A "coverage" is digital geospatial information representing space/time-varying phenomena.

The Open Geospatial Consortium, Inc. (OGC) Web Coverage Service (WCS) provides an open specification for sharing raster datasets on the web. ArcGIS for Server is able to publish WCS services from imagery collections, maps, or geodatabases that contain rasters.

Client applications work with a WCS service by appending parameters to the service's URL. WCS services published to ArcGIS Server support the following operations:

- Request service-level metadata and a brief description of the data (GetCapabilities)
- Request a full description of one or more coverages (DescribeCoverage)
- Request a coverage in a well-known coverage format (GetCoverage)

WCS 2.0.1 services also support the following OGC extensions:

- Service extensions: Scaling, Interpolation, Range Subsetting, and CRS
- Protocol extensions: KVP/Get and XML/Post
- Format encoding extensions: GeoTIFF

The WCS specification may be found at https://portal.opengeospatial.org/files/09-110r4.

4.15.2.7 Java API

The Geospatial Data Access Layer & GIS Services CI: Java API provides a Java-specific interface that integrates a wide range of mapping and geospatial capabilities, including editing, geocoding, routing, and management of web maps. With the Java API, developers can build maps with the latest ArcGIS Server map and feature services, perform fast searches for location

and routes, and build applications to edit features. This API also supports the Geospatial Data Access Layer & GIS Services CI: GeoEvent Processor. The Java API can be found at https://developers.arcgis.com/java/latest/api-reference/reference/packages.html.

4.15.2.8 JavaScript API

The Geospatial Data Access Layer & GIS Services CI: JavaScript API provides a JavaScript-specific interface that allows developers to manage geospatial maps and features, and integrates with ArcGIS server to help ACS customers build map clients. The JavaScript API can be found at https://developers.arcgis.com/javascript/latest/api-reference/index.html.

4.15.2.9 Python API

The Geospatial Data Access Layer & GIS Services CI: Python API is used for the Geospatial Data Access Layer & GIS Services CI: GIS Admin CLI. The API supports vector and raster analysis, geocoding, map making, routing and directions, as well as organizing and managing items in a GIS with users and groups. The Python API can be found at http://esri.github.io/arcgis-python-api/apidoc/html/.

4.16 OZONE WIDGET FRAMEWORK (OWF)

4.16.1 Overview

The OZONE Widget Framework (OWF) allows data from different servers to communicate inside a browser window without sending information back to the respective servers. This unique capability allows the OWF web portal to offer decentralized data manipulation. It includes a secure, in-browser, pub-sub eventing system which allows widgets from different domains to share information. The combination of decentralized content and in-browser messaging makes OWF particularly suited for large distributed enterprises with legacy stovepipes that need to combine capability. Use it to quickly link applications and make composite tools. Here is OWF at a glance:

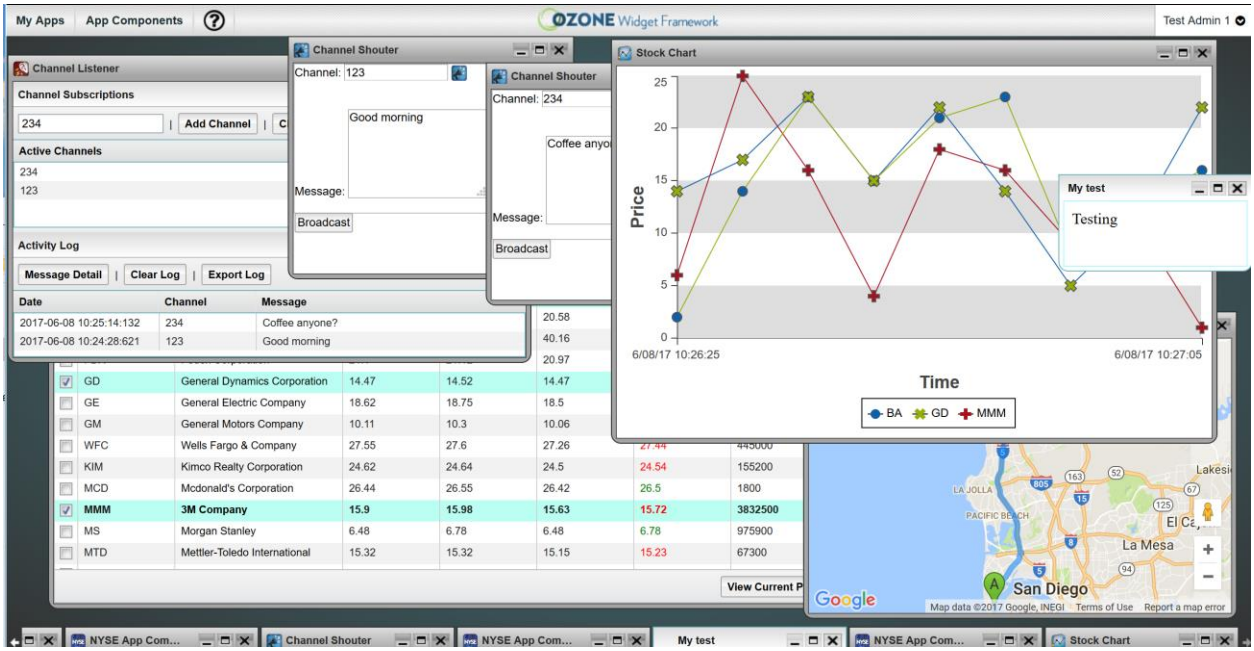


Figure 37 Ozone Widget Framework

OWF consists of a number of components that were designed to be independently deployed or located on the same server. The simplest deployment scenario places them all on the same physical machine. These components are shown in detail as follows:

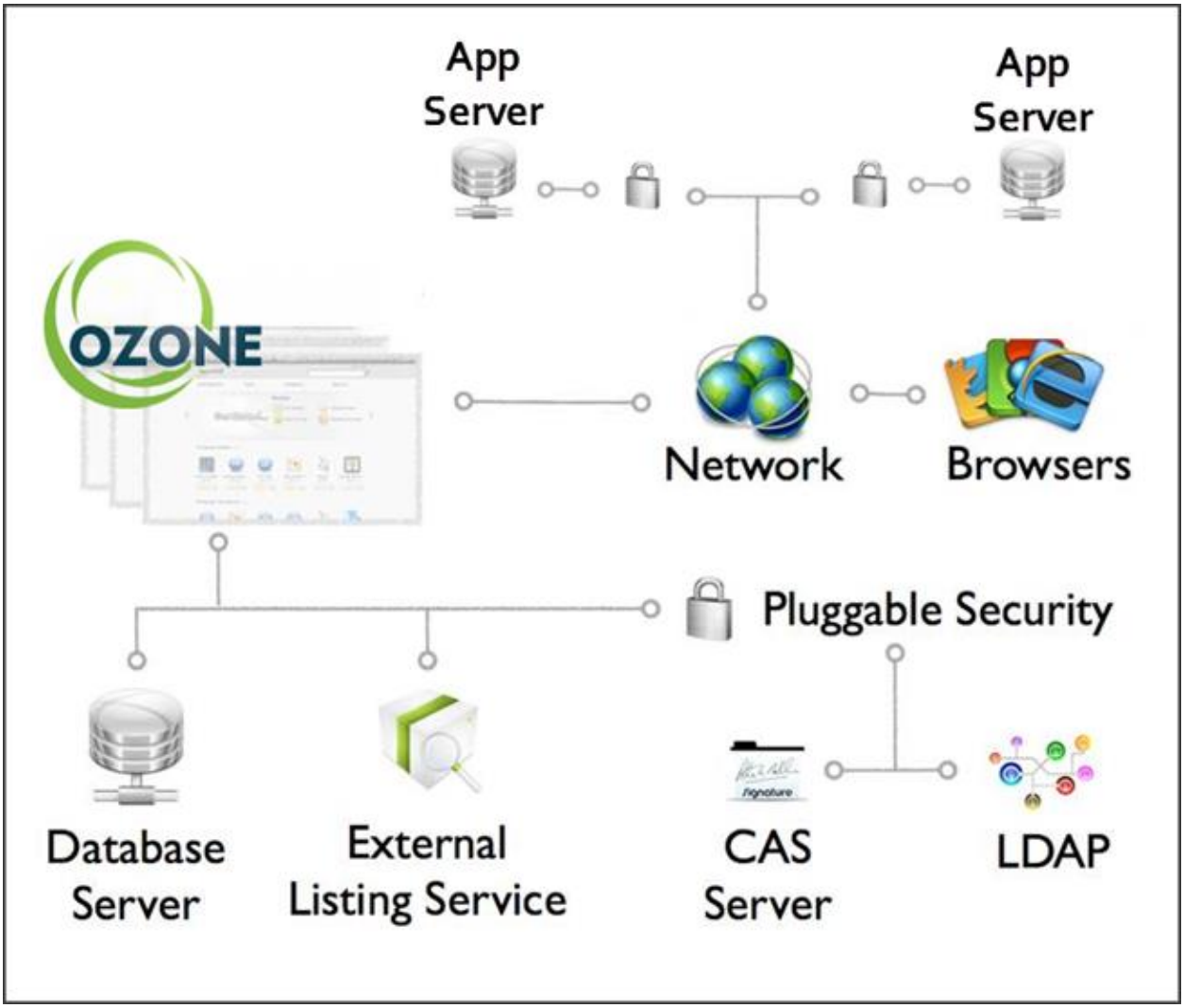


Figure 38 Ozone Widget Framework Component Diagram

4.16.2 Interfaces

4.16.2.1 Web GUI

OWF was designed to be a simple and robust end-user framework. Unlike most widget frameworks today that require writing extensive code, Ozone abstracts out unnecessary technical details from the end users and provides a simple web front-end where end users can compose widgets and restrict communication amongst widgets.

OWF allows its users to place widgets on a web dashboard. Users of Ozone accesses this dashboard through a web browser. Widgets are opened on this dashboard. Users selects a widget type to deploy (for example, a graph plotting widget), and after that a corresponding widget instance is created on the dashboard. Thus, one can create many instances of the same widget and work with them separately.

Using the App Builder, users can design dashboards that incorporate multiple layouts. Users can also incorporate accordion, portal and fit-pane layouts into each dashboard as seen below.

Figure 39 OWF Web GUI

The User Menu is a drop-down feature located on the right-side of the OWF toolbar. All users have access to certain OWF Options described in the OWF User's Guide (User Profile, Themes, About and Sign Out). OWF administrators have access to the Administration OWF Option.

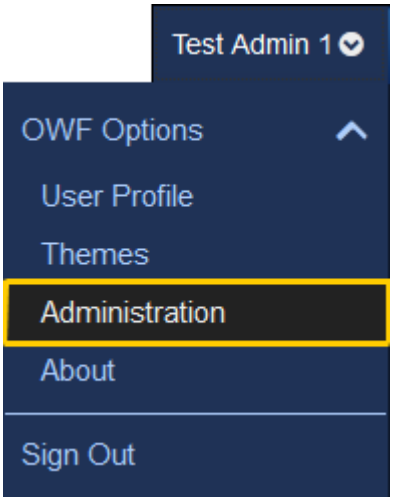


Figure 40 OWF User Menu

Administration Tools allow an administrator to manage applications, application component, users, groups and system configurations using the Apps, App Components, Users, Groups and Configuration managers respectively. The Configuration manager allows administrators to change select OWF configuration settings from within the user interface.

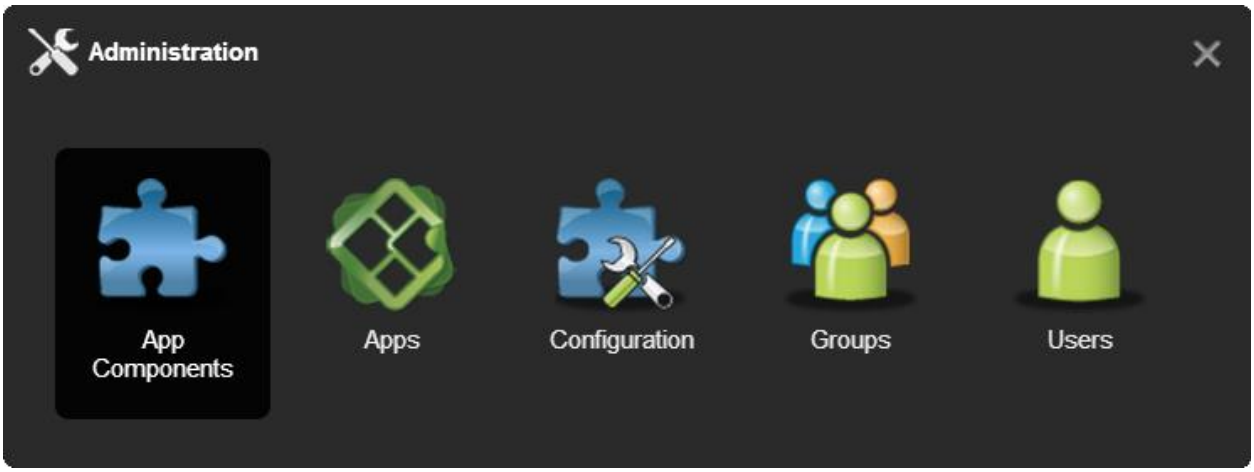


Figure 41 OWF Administration Tools

An application (app) component is a lightweight, single-purpose application that offers a summary or limited view of a larger application. In OWF, an app component is a global description for a piece of Web content that can be configured by the user and displayed within an application.

Apps can have multiple pages. Use the feature to compile resources and make similar things easy to access. Apps and their associated pages appear in the My Apps menu.

App pages are individual layouts associated with an app. App pages can be created, added, restored and deleted by the user. Users can customize app pages to include user preferred layouts and app components.

Figure 42 OWF Apps and Pages

More details are available at https://github.com/ozoneplatform/owf-framework/wiki.

4.16.2.2 REST Interface

The REST API can be used to configure the OWF widgets. More API details are available at https://github.com/ozoneplatform/owf-framework/blob/master/docs-external/rest/OWF-REST.md.

4.17 OPENAM

4.17.1 Overview

OpenAM is an open-source access management, entitlements and federation server platform. OpenAM uses open standards, is commercially supported, provides multi-node deployment strategies for scalability and is widely used by industry and many ACS customers. OpenAM is an all-in-one open source, centralized access management solution, securing protected resources across the network and providing authentication, authorization, web security, and federation services in a single, integrated solution. OpenAM is deployed as a simple .war file and provides production-proven platform independence, flexible and extensible components, as well as a high availability and a highly scalable infrastructure.

4.17.2 Interfaces

4.17.2.1 Administration GUI

OpenAM Administration GUI supports OpenAM configuration and administration. As shown in the figure below, the OpenAM Administration GUI is comprised of a number of tabs which allows administrators to monitor and configure common tasks, access controls, federations, web services, general configuration and sessions.

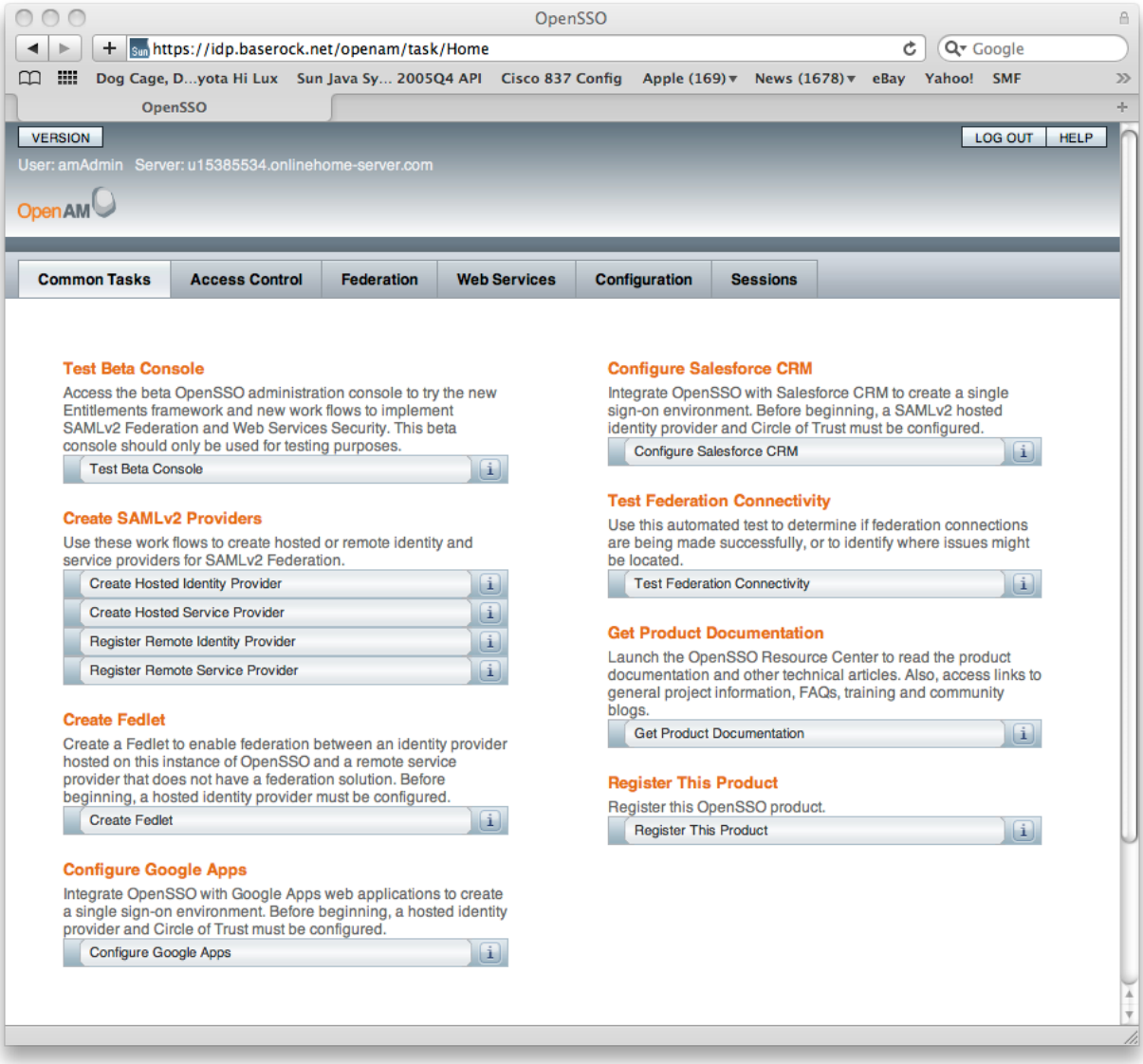


Figure 43 OpenAM Administration GUI

Further details may be found at https://backstage.forgerock.com/docs/openam/13/admin-guide/index.html and https://backstage.forgerock.com/docs/openam/13/admin-guide/index.html#openam-console-overview.

4.17.2.2 Administration CLI

The Security Management CI: OpenAM Administration CLI provides a command line interface enabling administrators to manage OpenAM policy agent installations, perform a silent installation of an OpenAM server with a configuration file, and configure OpenAM core services. Further details may be found in the ICD and at https://backstage.forgerock.com/docs/openam/13/admin-guide/index.htmland https://backstage.forgerock.com/docs/openam/13/admin-guide/index.html#openam-cli-overview.

4.17.2.3 APIs

Developers can access OpenAM services from client applications in Java, C or via REST utilizing the Security Management CI: OpenAM APIs:

- Java APIs support Java application authentication, and authorization in both OpenAM and federated environments. Detailed reference information is provided in the OpenAM Java SDK API Specification
- C APIs support Operating System native applications such as new web server policy agents. C OpenAM APIs are available for Linux and Windows platforms.
- RESTful APIs return JSON or XML over HTTP, allowing authentication, authorization, and identity services from web applications using REST clients written in multiple languages.

Further details may be found at https://backstage.forgerock.com/docs/openam/13/dev-guide/.

4.17.2.4 Policy Agent Interfaces

The Security Management CI: OpenAM Policy Agent Interfaces are a collection of agents to support policy decisions for standard types of ACS consumer applications. OpenAM provides several types of policy agents, including Web, J2EE, SOAP, OAuth, and OpenIG. Further details are provided at https://backstage.forgerock.com/docs/openam/13/admin-guide/index.html#chap-agents

In the figure below, a transaction from a Web Policy Agent to OpenAM is depicted. The Policy Agent passes a GET Request for a resource at a URL with a given User name and time. The OpenAM policy allows access to the given resource based on URL patterns, actions, subjects and conditions such as time periods. In this case, the response sent back by OpenAM is to authorize access to the requested resource for the given actions since the time and user are within the policy bounds.

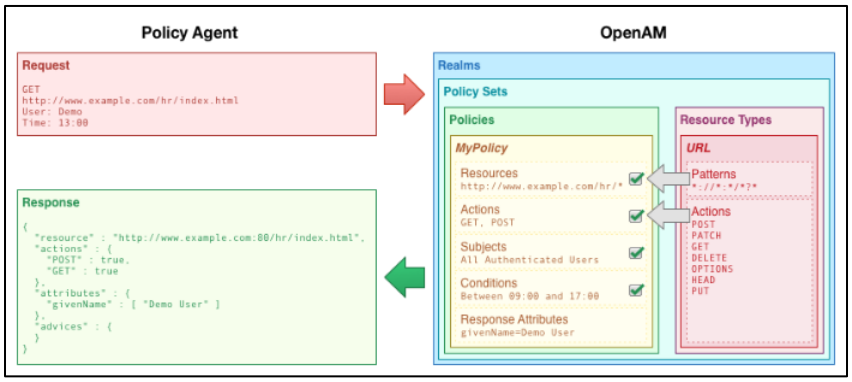


Figure 44 OpenAM Policy Agent Request/Response

4.17.2.5 Service Interfaces

The Security Management CI: OpenAM Service Interfaces are all provided through one HTTPS port. As shown in the figure below, OpenAM Services include federation, policy, identity, and OAuth. All the policy agents, APIs and client interfaces interact with these backend services to enable the authentication and authorization capabilities of the ACS Security Management CI.

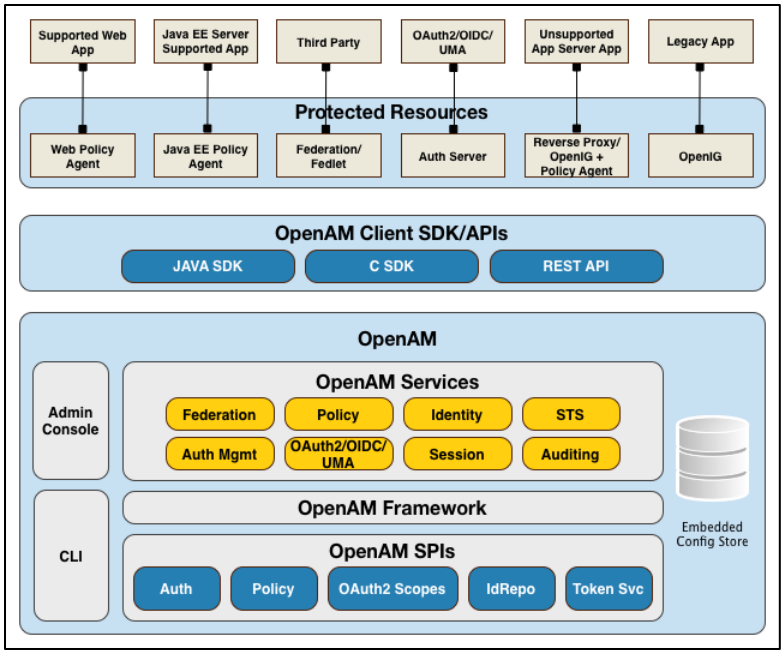


Figure 45 Open AM Services

For further details see https://backstage.forgerock.com/docs/openam/13/admin-guide/index.html.https://backstage.forgerock.com/docs/openam/13/admin-guide/index.html

4.18 NATIONAL MARITIME INTELLIGENCE CENTER (NMIC) CENTRALIZED AUTHORIZATION & AUTHENTICATION SERVICE (NCAAS)

4.18.1 Overview

NCAAS provides authentication and authorization with role-based and attribute-based access control. NCAAS offers some unique features such as a Policy Enforcement Point (PEP) and a Policy Decision Point (PDP), which provides a centralized security architectural solution, as well as aggregation of attributes from multiple DoD endpoints including DoDISS Identity and Authorization Services (DIAS) and Enterprise Identity Attribute Service (EIAS) which are particularly applicable for high-side applications.

4.18.2 Interfaces

4.18.2.1 Administration GUI

The Security Management CI: NCAAS Administration GUI supports NCAAS configuration and monitoring. As shown in the figure below, the Administrator can see all the services, and their current state, as well as administer and manage the services.

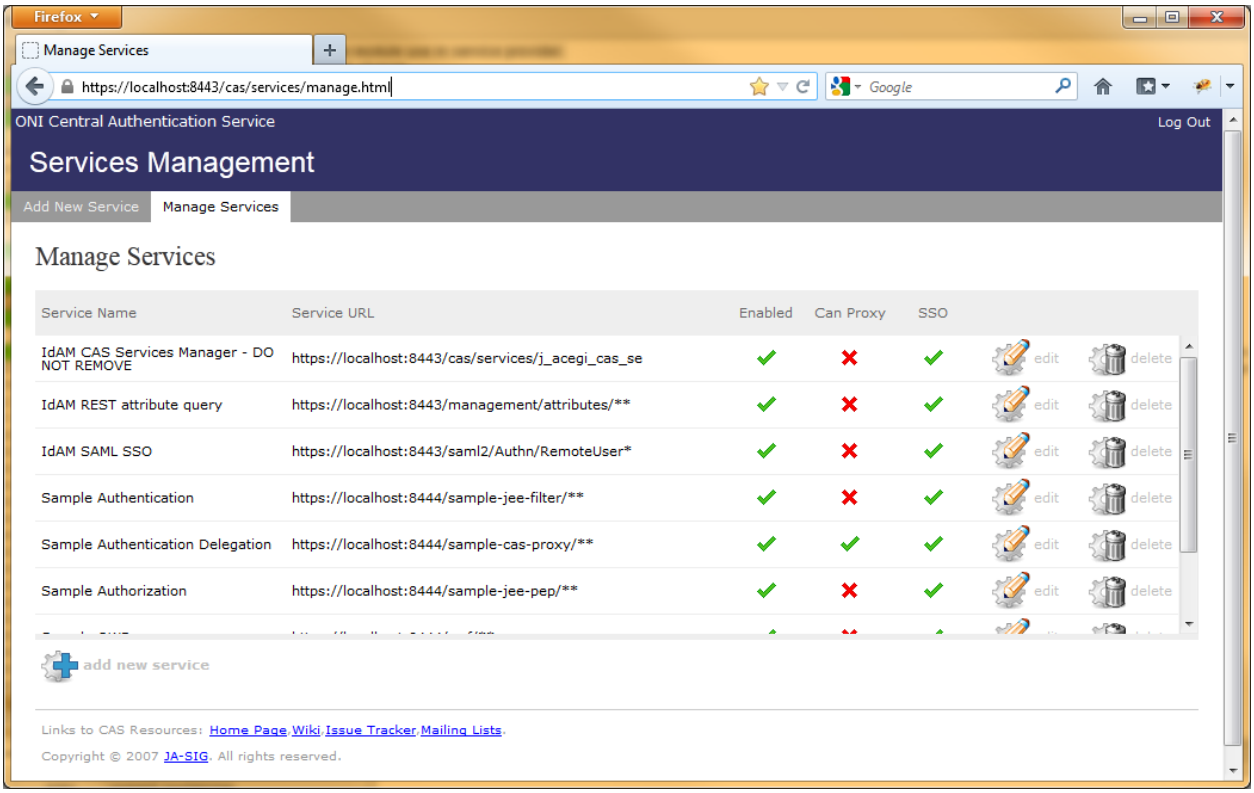


Figure 46 NCAAS Administration GUI - Managing Services

In the figure below, the Administrator can use the Policy Editor to specify the policy as a collection of rules and to specify the rules, what they apply to and their values.

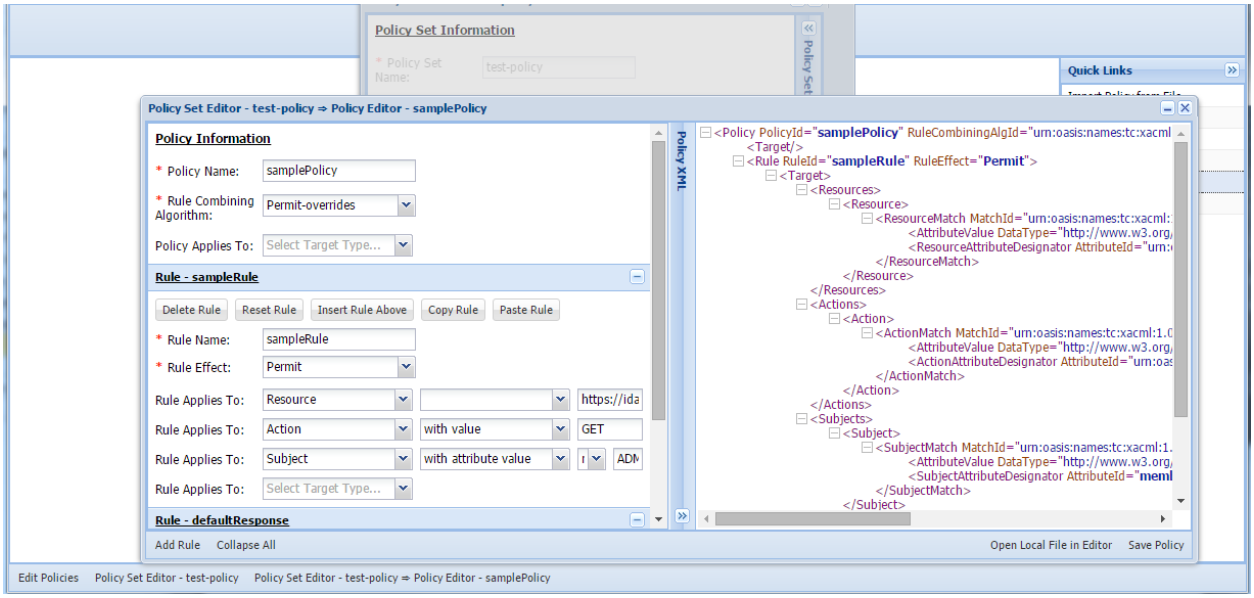


Figure 47 NCAAS Administration GUI - Editing Policy

4.18.2.2 Administration CLI

NCAAS Administration CLI provides a command line interface enabling administrators to manage NCAAS.

The management shell is implemented as an interactive groovy shell; it interprets the commands the user types on the interactive console against the preloaded set of groovy commands and object. If the command entered is valid, the shell then executes the command by communicating with the JMX MBean server running inside the NCAAS server.

4.18.2.3 Administrative Console Interface

The Administration Console Interface is a web-based GUI which allows administrators to discover, approve, download, and install widgets and widget-supporting services as well as updates to the same.

Developers can access Administrative Console Interface from client applications in a variety of programming language using REST and SOAP. The Administrative Console Interface is comprised of REST and SOAP interfaces from three components:

- Jasig for SSO supporting using a username/ password or certificate and for user attribute aggregation from multiple sources
  See https://apereo.github.io/cas/5.1.x/index.html for further details
- Shibboleth Identity Provider for SAML SSO support
  See http://shibboleth.net for further details.
- Picketlink for XACML query support
  See http://www.jboss.org/picketlink for further details.

4.18.2.4 Policy Agent Interfaces

OpenAM Policy Agent Interfaces are a collection of agents to support policy decisions for standard types of ACS consumer applications. OpenAM provides several types of policy agents, including support for Java Web applications and any client that can utilize REST or SOAP.

4.18.2.5 Service Interfaces

NCAAS Service Interfaces are comprised of REST and SOAP interfaces from three components:

- Jasig for SSO support using a username/ password or certificate Jasig also aggregates attributes for a user from multiple sources See https://apereo.github.io/cas/5.1.x/index.html for further details
- Shibboleth Identity Provider for SAML SSO support See http://http://shibboleth.net for further details.
- Picketlink for XACML query support See http://www.jboss.org/picketlink for further details.

Table 2 NCAAS Service Interfaces - REST Services

Service	Relative URL	Description
Jasig CAS RESTful API	/cas/v1/tickets	For obtaining Jasig CAS tickets. Further details available at: https://wiki.jasig.org/display/CASUM/RESTful+API
REST attribute query	/management/attributes/query	Retrieves subject attributes given Distinguished Name.
SAML Identity Providers Metadata	/saml2/profile/Metadata/SAML	A SAML Metadata XML document for NCAAS

Table 3 NCAAS Service Interfaces - SOAP Services

Service	Relative URL	Description
SAML HTTP POST Binding	/saml2/profile/SAML2/POST/SSO	Provided by Shibboleth. http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf
SAML HTTP Redirect Binding	/saml2/profile/SAML2/Redirect/SSO	Provided by Shibboleth. http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf
SAML Enhanced Client Or Proxy (ECP) Profile	/saml2/profile/SAML2/SOAP/ECP	Provided by Shibboleth. https://www.oasis-open.org/committees/download.php/39817/draft-sstc-saml-ecp-2.0-01.pdf
SAML Artifact Resolution	/saml2/profile/SAML2/SOAP/ArtifactResolution	Provided by Shibboleth. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
SAML Attribute Query	/saml2/profile/SAML2/SOAP/AttributeQuery	Provided by Shibboleth. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
Policy Decision Point (PDP)	/pdp/SOAPSAMLXACMLPDP	Responds to XACML queries. Described in the Developer's Guide.

4.19 ELK (ELASTICSEARCH, LOGSTASH, KIBANA)

4.19.1 Logging Overview

Logs provide us with necessary information on how systems are behaving. However, the content and format of the logs varies among different services or say, among different components of the same system. For example, a scanner may log error message related to communication with other devices; on the other hand, a web server logs information on all incoming requests, output responses, time taken for a response, and so on. Similarly, application logs for an e-commerce website will log business-specific logs.

As the logs vary by their content, so will their uses. For example, the logs from a scanner may be used for troubleshooting or for simple status check or reporting while the web server log is used to analyze traffic patterns across multiple products. Analysis of logs from an e-commerce site can help figure out whether packages from a specific location are returned repeatedly and the probable reasons for the same.

The following are some common uses cases where log analysis is helpful:

- Issue debugging
- Performance analysis
- Security analysis
- Predictive analysis

Issue debugging

Debugging is one of the most common reasons to enable logging with your application. The simplest and most frequent use for a debug log is to grep for a specific error message or event occurrence. If a system administrator believes that a program crashed because of a network failure, then the system administrator will try to find a connection dropped message or a similar message in the server logs to analyze what caused the issue. Once the bug or the issue is identified, log analysis solutions help capture application information and snapshots of that particular time can be easily passed across development teams to analyze it further.

Performance analysis

Log analysis helps optimize or debug system performance and give essential inputs and bottlenecks in the system. Understanding a system's performance is often about understanding resource usage in the system. Logs can help analyze individual resource usage in the system, behavior of multiple thread in the application, potential deadlock conditions, and so on. Logs also carry with them timestamp information, which is essential to analyze how the system is behaving over time. For instance, a web server log can help know how individual services are performing based on response times, HTTP response codes, and so on.

Security analysis

Logs play a vital role in managing the application security for any organization. They are particularly helpful to detect security breaches, application misuse, malicious attacks, and so on. When users interact with the system it generates log events, which can help track user behavior, identify suspicious activities, and raise alarms or security incidents for breaches.

The intrusion detection process involves session reconstruction from the logs itself. For example, ssh login events in the system can be used to identify any breaches on the machines.

Predictive analysis

Predictive analysis is becoming popular trend. Logs and events data can be used for very accurate predictive analysis. Predictive analysis model help in identifying potential customers, resource planning, inventory management and optimization, workload efficiency, and efficient resource scheduling.

4.19.2 ELK Logging

The ELK platform is a complete log analytics solution, built on a combination of three open source tools - Elasticsearch, Logstash, and Kibana. ELK utilizes the open source stack of Elasticsearch for deep search and data analytics; Logstash for centralized logging management, which includes shipping and forwarding the logs from multiple servers, log enrichment, and parsing; and finally, Kibana for powerful and beautiful data visualizations.

A brief overview of the Elasticsearch, Logstash, and Kibana systems are given below:

4.19.2.1 Elasticsearch

Elasticsearch is a distributed open source search engine based on Apache Lucene, and release under an Apache 2.0 license which means it can be downloaded, use, and modified free of charge. It provides horizontal scalability, reliability, and multitenant capability for real-time search. Elasticsearch features are available through JSON over a REST API. The searching capabilities allows it to dynamically index data without knowing the structure beforehand. Elasticsearch is able to achieve fast search responses because it uses indexing to search over the texts.

Elasticsearch is used by many big companies, such as GitHub, SoundCloud, Wikipedia, and many others. Some of the use cases are as follows:

- Wikipedia: This uses Elasticsearch to provide a full text search, and provide functionalities, such as search-as-you-type, and did-you-mean suggestions.
- SoundCloud: This uses Elasticsearch to provide real-time search capabilities for millions of users across geographies.
- GitHub: This uses Elasticsearch to index over 8 million code repositories, and index multiple events across the platform, hence providing real-time search capabilities across it.

Some of the key features of Elasticsearch are:

- It is an open source distributed, scalable, and highly available real-time document store
- It provides real-time search and analysis capabilities
- It provides a sophisticated REST API to play around with lookup, and various features, such as multilingual search, geolocation, autocomplete, contextual did-you-mean suggestions, and result snippets
- It can be scaled horizontally easily and provides easy integrations with cloud-based infrastructures, such as AWS and others

4.19.2.2 Logstash

Logstash is a data pipeline that helps collect, parse, and analyze a large variety of structured and unstructured data and events generated across various systems. It provided plugins to connect to

various types of input sources and platforms, and is designed to efficiently process logs, events, and unstructured data sources for distribution into a variety of outputs with the use of its output logins, namely file, stdout (as output on a console running Logstash), or Elasticsearch.

It has the following key features:

- Centralized data processing: Logstash helps build a data pipeline that can centralize data processing. With the use of a variety of plugins for input and output, it can convert a lot of different input sources to a single common format.
- Support for custom log formats: Logs written by different applications often have particular formats specific to the application. Logstash helps parse and process custom formats on a large scale. It provides support to write your own filters for tokenization and also provides ready-to-use filters.
- Plugin development: Custom plugins can be developed and published, and there is a variety of custom developed plugins already available.

4.19.2.3 Kibana

Kibana is an open source Apache 2 licensed data visualization platform that helps in visualizing any kind of structured and unstructured data stored in Elasticsearch indexes. Kibana is entirely written in HTML and JavaScript. It uses the powerful search and indexing capabilities of Elasticsearch exposed through its REST API to display powerful graphics for the end users. From basic business intelligence to real-time debugging, Kibana plays its role through exposing data through histograms, geomaps, pie charts, graphs, tables, and so on.

Kibana makes it easy to understand large volumes of data. Its simple browser-based interface enables you to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time.

Some of the key features of Kibana are as follows:

- It provides flexible analytics and a visualization platform for business intelligence.
- It provides real-time analysis, summarization, charting, and debugging capabilities.
- It provides an intuitive and user friendly interface, which is highly customizable through some drag and drop features and alignments as and when needed.
- It allows saving the dashboard, and managing more than one dashboard. Dashboards can be easily shared and embedded within different systems.
- It allows sharing snapshots of logs that you have already searched though, and isolates multiple problem transactions.

4.19.2.4 ELK Data Pipeline

A typical ELK stack data pipeline looks something like this:

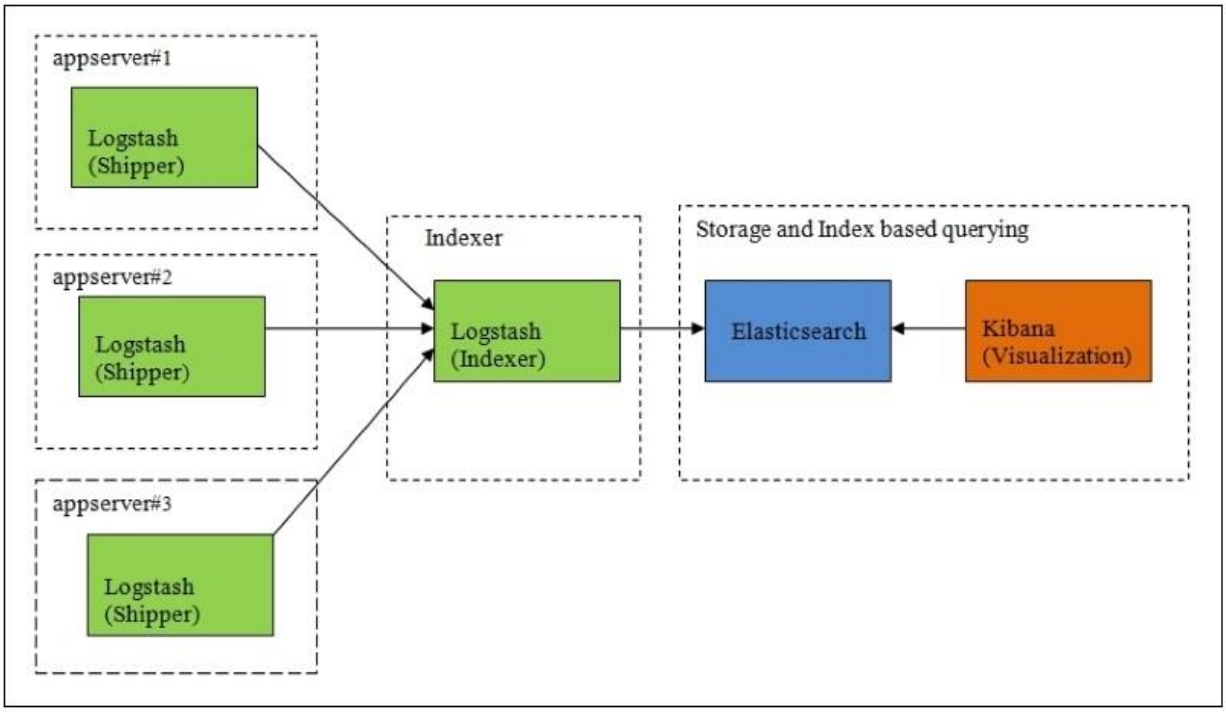


Figure 48 ELK Data Pipeline

In a typical ELK Stack data pipeline, logs from multiple applications servers are shipped through Logstash shippers to a centralized Logstash indexer. The Logstash indexer will output data to an Elasticsearch cluster, which will be queried by Kibana to display great visualizations and build dashboards over the log data.

4.19.3 Beats

Beats is the open source platform for single-purpose data shippers. They install as lightweight agents on the servers to sends different types of operational data to Logstash or Elasticsearch. Beats are great for gathering data. They sit on servers and centralize data in Elasticsearch. If more processing power is needed, Beats can also ship to Logstash which can be used to parse and transform data.

Packetbeat, Filebeat, Metricbeat, and Winlogbeat are a few examples of Beats. Packetbeat is a network packet analyzer that ships information about the transactions exchanged between your application servers. Filebeat ships log files from your servers. Metricbeat is a server monitoring agent that periodically collects metrics from the operating systems and services running on your servers. And Winlogbeat ships Windows event logs.

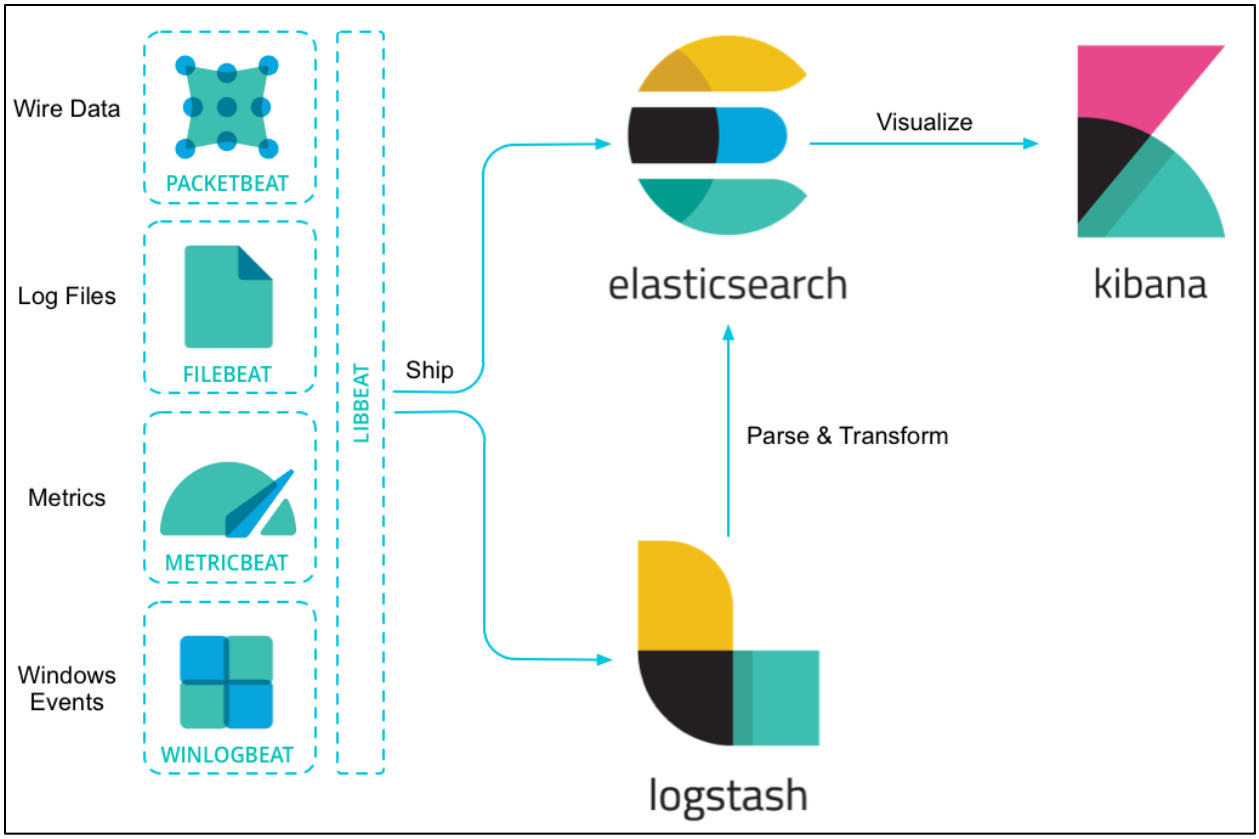


Figure 49 Beats

Log collection is facilitated by a set of log collection agents (one installed on each Ingress and Egress Dataflow Management Framework CI Node) identified as Ingress and Egress DataFlow Management Framework CI: Beats. The log collection agents send logs to a log collection service as described in the ACS System Management Framework CI SRS.

4.20 ZOOKEEPER

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them which makes them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

ZooKeeper aims at distilling the essence of these different services into a very simple interface to a centralized coordination service. The service itself is distributed and highly reliable. Consensus, group management, and presence protocols will be implemented by the service so that the applications do not need to implement them on their own. Application specific uses of these will consist of a mixture of specific components of Zoo Keeper and application specific conventions.

For example, the Ingress and Egress Dataflow Management Framework CI is configured to operate as a cluster of NiFi nodes to support scalability in a zero-master manner as described by the following figure. Each node in a NiFi cluster performs the same tasks on the data, but each operates on a different set of data. ZooKeeper elects a single node as the Cluster Coordinator, and failover is handled automatically by ZooKeeper. All cluster nodes report heartbeat and status information to the Cluster Coordinator. The Cluster Coordinator is responsible for disconnecting and connecting nodes. Additionally, every cluster has one Primary Node, also elected by ZooKeeper. ACS customer DataFlow managers may interact with the NiFi cluster through the UI of any node; any changes made are replicated to all nodes in the cluster.

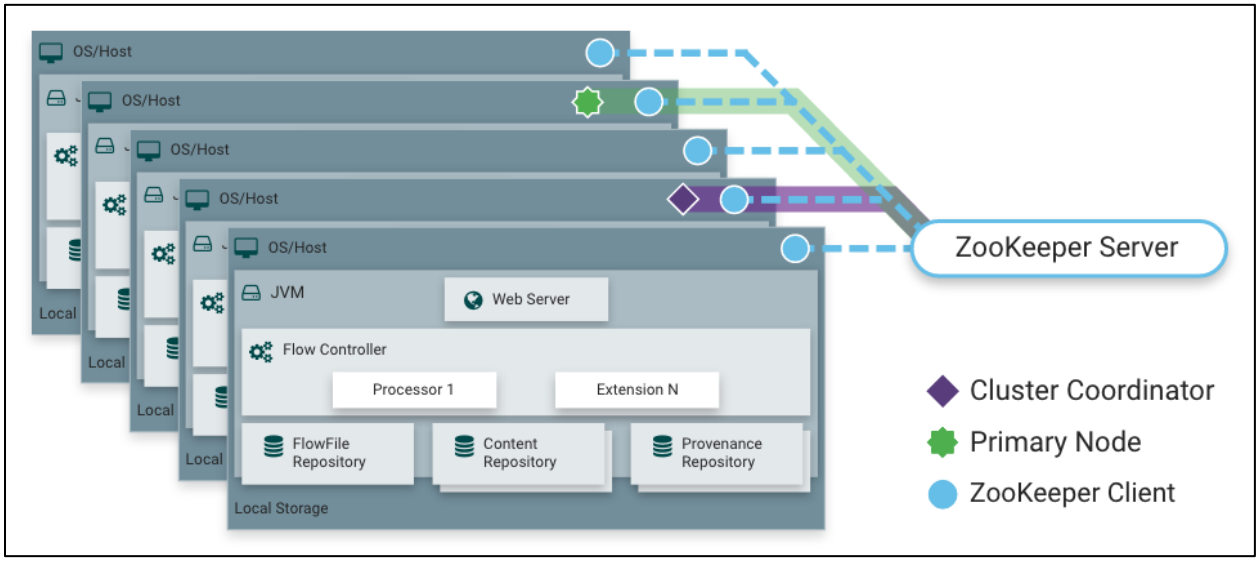


Figure 50 ZooKeeper Server of NiFi Cluster

5. DATA STORES

5.1 DATA ELEMENTS

Within the TCRI, there are essentially four large categories of data persisted in the system:

1. Artifacts of raw data: documents, messages, databases, etc. – raw data sources that have been ingested into the system
2. Extracted or enriched data elements: Data that has been generated from the raw artifact. Examples of this include structured data extractions from these Artifacts, enriched or extracted data generated through application or analytic processing, or assertions of an analyst based on messages. For example, that John Doe in a specific document is the name of a person or that 40°42′51″ N is a specific latitude.
3. Provenance information: information detailing where a data element or artifact originated
4. Semantics of Artifacts and structured data – data models, ontologies, and mappings between models and ontology.

Extracted/enriched data can be represented using a variety of different data models including:

1. Semi-structured data: highly structured data that may or may not have a fixed schema. This type of data model can have a high temporal element to it (for example event data), or it may have no temporal element associated. This type of data often consists mainly of entities with associated properties.
2. A linked graph: Linked data that is best represented as linkages between entities or nodes. This type of data encompasses tree or hierarchical data, directed graphs, and is best suited for representing relationships between entities within a data set.

5.2 DATA MODELS

5.2.1 Artifact Models

Artifact models are used to store specific sets of data in a pre-set template. So let's say you have a "Sentence" model. Whenever a sentence is input into storm, the different words are taken apart and placed in that model: "The Jeep is Green" is scanned and turned into:

```
<sentence>
<id>01</id>
<starter>"The"</starter>
<noun>"Jeep"</noun>
<verb>"is"</verb>
<adj>"Green"</adj>
</sentence>
```

Whatever sentence is pushed into the model, a <sentence> artifact is created with the specific information from that sentence.

This also means that any sentence not following the pattern will need a new model created for it. Something like "I love to type words" will need a new artifact model to fit the data into.

5.2.2 Domain Models

Domain Models are cool. They allow many different kinds of data to be compared and analyzed by cutting out the things that don't matter. Back to the sentence example. Say we have 3 kinds of sentences, with a different artifact model for each one. The original sentences were:

- The Jeep is Green
- I love to type words
- I will get you next time, Gadget!

Each of these would be put into its own artifact model, with different forms and pieces. But say we wanted to process all of the noun-verb combinations of multiple sentences. That is when we take our artifact models and squeeze them into domain models. For example, here's the domain model for the last sentence:

```
<noun_verb>
<id>03</id>
<noun>"I"</noun>
<verb>"get"</verb>
</noun_verb>
```

Each of the 3 sentences can be fit into this domain model, and afterword we'll have 3 new objects that are much easier to organize and compare for *this specific domain*. If we want to compare, say, adjectives and Starters, then we will need to make a new domain model.

5.3 DATA STORES

Within the TCRI, there are essentially four classes of data stores:

1. Artifact store: a store that primarily stores raw artifacts and their associated metadata.
2. Linked graph store: A store that primarily stores linked data (relationships between entities in extracted/enriched data)
3. Semi-structured data store: A store that primarily stored semi-structured data. Due to the variety of different use cases that are satisfied at least partially with semi-structured data, it may be necessary to have several structured data stores targeted towards specific access or ingest patterns.
4. Semantic Store: A store that primarily stores model information. While each store within the data space may store model information, the semantic store exposes model information to end users in a consistent manner and also provides a means of relating models across the different data stores.

In the following sections, these different data stores and their purposes within the data space will be further described.

5.3.1 Artifact Store

Let us consider two data sources: a database and a message. Artifacts represent sources of information. These sources can be each identified by a unique identifier and described by metadata (e.g. date of creation, MIME type, etc.). Data sources fall into two categories: tangible and intangible. Tangible artifacts include text documents, videos, images, etc. Intangible artifacts

include users, processes, etc. Regardless of type, each artifact is described using a GUID and a set of metadata (e.g. date of creation, MIME type, etc.). Additionally, for tangible artifacts, there is associated raw data that can be exposed via the artifact store.

The artifact store can also store representations of associations between artifacts (such as versions, relevance, similarity, etc.). These associations between artifacts allow for a user to represent things like the provenance of a raw artifact.

5.3.2 Linked Graph Store

Within a linked graph store, there are essentially two key constructs:

- Entity Terms: Nodes within a graph
- Statements: Relationships between nodes in a graph

Entity Terms can be characterized as a Node with an associated type and an associated literal ("Sign" or significant identifier). The significant identifier or sign of an entity term is a string that represents some fragments of Artifacts, e.g. ‘Bill’ for a table cell and ‘William’ for a part of the message. The significant identifier should be an abstractions from the source data and may not necessarily be derived from fragments of the Artifacts. The associated type of the node can reference a concept stored within the semantic store.

A Statement is an abstraction of a directed graph assertion for describing linked graph data. A statement is represented as a triple: <Subject Entity Term, Predicate, Object Entity Term>.

Both Terms and Statements can have metadata associated with them. This metadata contains information about when an assertion was made, who it was made by and what artifact(s) the information was taken from.

5.3.3 Semi-Structured Data Store

Semi-structured data stores are intended to store data that is highly structured and has some sort of associated schema. While the schema may be flexible, it will mainly consist of a list of properties and their associated values for a particular entity type. The semi-structured data store in general will need to support high velocity input, and should be able to support forward and reverse look ups based off of property type values. Also, this class of store should also be able to support a generic "roll off" policy for data management, where data is purged either following some sort of an age off policy or some sort of query based methodology.

There are two main categories of semi-structured data:

1. Temporal event data: Semi-structured data that has a high temporal element to it and is typically accessed primarily through time based patterns
2. Non-temporal entity data: Semi-structured data that will not be access through primarily temporally oriented access patterns. While this data may have a temporal aspect to it, time is not its main distinguishing identifier, and most queries may not involve time at all.

To support these two main categories, it is necessary to have two types of semi-structured data stores:

1. A time sharded event store
2. A non-time sharded object/document store

5.3.3.1 Time Based Event Store

The core data structure persisted within the time based event store is an Event, which contains the following elements:

- **Id:** Unique identifier for the Event
- **Timestamp:** Timestamp of the Event
- **Type:** Datatype of the Event
- **Event Fields:** Parameters and metadata associated with the Event
  - **Key:** Parameter field key
  - **Value:** Parameter field value
  - **Visibility:** Parameter field visibility (cell-level security)
  - **Metadata:** Event field metadata used for ingest indexing and normalization
- **Content:** Attached content associated with an event
  - **Bytes:** Raw content bytes
  - **Media Type:** Identifier to indicate the type of data (e.g. application/pdf)
  - **Visibility:** Content visibility
  - **Text Content:** (available on IndexableContent) ASCII representation of content
- **Metadata:** Event specific metadata

Event Types describe what each event field is intended to contain. Since all data is assumed to be time based within the event store, data should be automatically "rolled off" or deleted after specified periods of time.

5.3.3.2 Non-time sharded Document/Object Store

The core data structure within the non-time sharded object store is a structured object that contains the following elements:

- Key: a unique identifier for an individual structured object
- Properties: zero or more field value pairings. Properties are by default a multimap, supporting multiples values for a single field.

The key structure for a structured object is:

```
<key> === <namespace><ancestor_path><kind><id>
```

```
The inclusion of a namespace in the key allows for multi-tenancy where
different applications can essentially create sandboxes of data through
different namespaces (for instance if an analytic needs a temporary working
space for intermediate results). There will be a single namespace that holds
all shared mission data.
```

```
The kind indicates that the object is a member of a category of structured
objects. The <id> is a kind-specific identifier. The ancestor path allows for
an implicit hierarchy in the definition of an object, where a single parent
(the ancestor structured object) can contain several children structured
objects. This implicit relationship can be utilized to optimize retrieval for
common association like queries ("give me all the children of a parent",
```

"give me all of the track reports associated with a particular ship") if performance issues arise. This implicit relationship is analogous to the nested object relationship used with User-Centered Design (UCD) Objects.

A Structured Object's Properties are defined in terms of a label (field name), associated datatype, and the value associated with that property. The set of datatypes supported by structured objects has yet to be fully enumerated, but would include primitive types, strings, byte arrays, geospatial types, etc.

To support associated access patterns, a structured object data store will typically contain forward and reverse sorted look ups based on property value.

5.3.4 Semantic Store

The semantic store stores model information across the dataspace.

There are two levels of model information stored within the semantic store:

1. Common Model information: Shared semantics or the common model that underlies all of data

2. Datastore Specific (Native) model information: For information that is meant to be shared/part of the common dataspace, platform specific models can be shared using the semantic store.

The semantic store stores a reference to the native store's URI (a unique identifier) and the native stores id. Mappings between the store specific models and the common model are also stored within the semantic store. The mappings (Model Associations) can be used to indicate relative hierarchy/taxonomy information, as well as equivalence. In the future additional model associations may be implemented to indicate additional semantics.

To standardize the semantic within the dataspace, the semantic store contains the following data structures:

In an effort to unify terminology across the stores, the following data structures were created:

Individual - An identifiable datum - Individual identifier examples: drif sign, structured object key, depot event id, etc. - Primitive values are also individuals because they are uniquely identified by value

Type - A named set of Individuals - Equivalent to a Data Type, Kind, or Concept

Property - Relationship types or attributes/fields attached to an Individual - Predicate identifier examples : drif predicate, structured object property id, depot event field key, catalog type label - There are two types of properties -- Object Properties (relationships between Named Individuals) and Data Properties (attributes or fields associated with an Individual) - Object Properties can be defined as having a domain/range (which are analogous to the matching terminology in rdf) - Data Properties can have a domain and an associated range data type

Model Association - An instance of a relationship between either two types or two properties - Currently three types of associations are allowed:

1. Equivalence: Two types or properties are equivalent
2. Super: One type is a hypernym of the other
3. Sub: One type is a hyponym of the other

5.4 DATA STORES AND SERVICE APIS

5.4.1 Cohesive Data and Knowledge Store Strategy

ACS provides a set of stores to house data throughout its various processing phases. At one end of the spectrum, raw artifacts contain the pristine original content. At the other end of the spectrum, fully processed (potentially semantic) data can be used for reasoning and inferencing. Employing multiple data stores allows ACS to handle most mission use cases out-of-the-box without the need to develop custom stores or APIs.

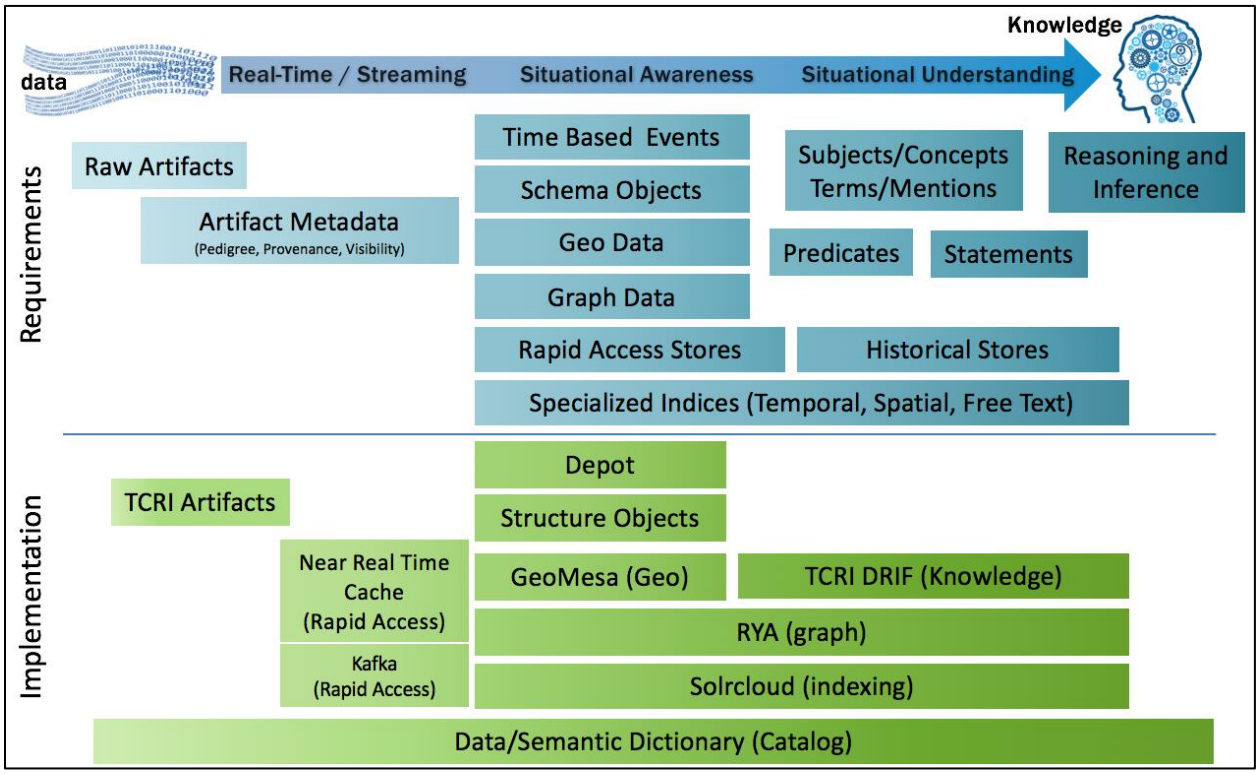


Figure 51 Cohesive Data and Knowledge Store Strategy

5.4.2 Data Store Listing

In general, each ACS data stores can be categorized as either a Primary store or a Special Purpose store. The Primary stores are implemented in Accumulo (or straight HDFS), have a rigorous security model, and align with industry standards such as ICITE and JIE. The Special Purpose stores fill niche requirements and leverage a custom security model to fulfill their purposes. See ACS Data Stores for details on each store.

Table 4 Data Store Listing

Type	Name	Purpose
Primary (Accumulo / HDFS)	Artifact	Original unmodified content
	Provenance	Contributing Agents to an artifact
	Content Zone	Large file storage in HDFS
	Structured Objects	Object / Document representation of structured data
	GeoMesa Historic	Geo-temporal feature storage (long term)
	Depot	Time-sharded events (to be replaced by Kronos)
	RYA	Graph store for RDF triples
	DRIF	Historic / semantic knowledge store
	SSR	Shared Semantic Repository (data model storage)
Special Purpose	GeoMesa NRT	Geo-temporal features for near-realtime access
	SOLR	Indexing for Artifact and DRIF stores
	ElasticSearch	Provenance Events and Audit Events

5.4.3 Data Store Comparison

A given datum may be represented in zero, one or multiple stores depending on the specific requirements of a programs. Through configuration and modeling, a program must specify which stores are used for which data. Criteria such as access (query) patterns and ingest performance should be considered when choosing data stores.

Table 5 Data Store Comparison

Store	Core Data Structures	Strength	Weakness	APIs
Artifact / CZ	Source artifact and metadata	Base raw artifact storage and retrieval	Very small discrete datum sizes	
Provenance	Contributing Agents Provenance Events Audit Events	ACS data store APIs have baked-in usage of Provenance APIs	Non-ACS data stores must use supplied decorators	

Store	Core Data Structures	Strength	Weakness	APIs
DRIF	Terms and Statements (linked graph data)	Semantically meaningful, linked graph data	High throughput, low semantically meaningful data	
Structured Objects	Structured Object (entity with attributes)	High throughput, highly structured data	Linked graph data	
Depot (Kronos)	Event	High throughput, time focused event data	Non-time based data, linked data	
RYA	RDF Triple	Linked data	Non-linked data	SAIL (Java) RYA Web (REST)
GeoMesa	Simple Feature	Geospatial and temporal focused data	Linked graph data	

5.4.4 API Types

Note that each of the ACS data stores (above) expose multiple APIs available to consumers. A listing of the various API types are listed in the table below, but some general guidelines are:

- To ingest data from external systems (outside the system boundary), DPF should be configured for ingestion.
- To persist data by trusted application and analytics (within the system boundary), use the Adhoc (aka Analytic Write) APIs.
- Queries by external systems (outside the system boundary) must use the REST APIs.
- Streaming applications (within the system boundary) can read or write from Kafka topics (offramps or onramps)

Table 6 API Types

Type	Use Case	Access	Benefits
Java	Use within the system boundary by Java consumers	JAR files, published in Nexus, must be integrated by app developers	Efficient native usage. Note that the "Analytic Write" or "Adhoc" APIs are implemented as Java APIs.
Thrift	Use within the system boundary by non-Java consumers	Thrift clients, published per API, must be integrated by app developers	Efficient support to multiple programming languages
REST	Available outside the system boundary for secure access	URLs and parameters, published per API, can be invoked at runtime	Secured with SSL and require PKI certs so that external system access is controlled.
Kafka - Offramp	Use within the system boundary for streaming consumers	Topic names and parameters, published in Confluence, can be read at runtime	Applications within the system boundary can acquire data during the ingress process
Kafka - Onramp	Use within the system boundary for streaming producers	Topic names and parameters, published in Confluence, can be written to at runtime	Applications within the system boundary can inject data into the ingress process

6. CLUSTER HOSTS

6.1 ANSIBLE HOST NAMES

Throughout this document, there will be references to different servers such as <basebox>, <storm_master>, <spark_master>, etc. These are names of individual servers that resides on the ACS cluster. To find out which nodes the servers runs on the ansible host file can be viewed. The ansible host file is located on the basebox node of the cluster in the /etc/ansible directory. A portion of the host file is shown below.

```
basebox         ansible_ssh_host=10.10.6.100            ansible_connection=local
node1           ansible_ssh_host=10.10.6.101
node2           ansible_ssh_host=10.10.6.102
node3           ansible_ssh_host=10.10.6.103
...
...
[basebox]
basebox

[storm_master]
node1

[accumulo_master]
node1
node3
```

In the ansible host file, the hostname and the corresponding IP's of the cluster are listed on top of the file. The lines following are the available servers (names with brackets around them) and the name of the nodes that run that server are listed under. So for the example above, there is the Accumulo Master server and the nodes named node1 and node3 has that server running on them.

6.2 WEB URLS

Table 7 ACS URLs

Component	URL	Description
Accumulo	http://<accumulo_master>:50095	Accumulo UI
ArcGIS	http://<geoevent_server>:6143/geoevent/manager/index.html	
	http://<arcgis_server>/portal/home	ArcGIS Portal home
ELK	https://<elk_server>:5643/app/kibana#/	Kibana
	https://<elk-sever>:5643/app/kibana#/dashboard/Topbeat-Dashboard	Topbeat dashboard

Component	URL	Description
	https://<elk_server>:9200/_cluster/health?pretty	ELK cluster health
Hadoop	http://<namenode>:50070	Primary NameNode server - overview of the state of the cluster's HDFS
	http://<failover_namenode>:50070	Secondary NameNode server
	http://<resourcemgr>:23188	Hadoop Resource manager
	http://<failover_resourcemgr>:23188	Failover Hadoop Resource manager
	http://<history_server>:19888	Hadoop Job history
IntellEx	https://<intellex_web>:8443/ConfigUI	Configuration UI
	https://<intellex_web>:8443/IXDesigner	Designer (starting ACS 3.1.0.0 release)
NiFi	http://<nifi>:6434/nifi	NiFi UI
	http://<nifi>:6434/nifi-api	NiFi REST API
SOQL	https://<tcri_so_web_server>:8443/sol-ws/query.html	SOQL
	https://<tcri_so_web_server>/tcri-structured-store-ws/sol-ws/query.html	SOQL (starting ACS 3.0.4.0 release)
OWF	https://[owf_web]:8443/owf	OZONE Widget Framework
Spark	http://<spark_master>:18080	Spark master
	http://<history_server>:18090	Spark History server
Storm	http://<storm_master>:8080	Storm UI

7. DATAFLOW

The Apache NiFi supports the automation and management of DataFlows (defined as the automated and managed flow of information between systems) into ACS from external data sources and systems. External data sources may include, but are not limited to, data contained in a filesystem, database, message bus, or external system.

7.1 TERMINOLOGY

Here are some key terms that readers should be familiar with when working with NiFi. For a comprehensive list, refer to https://nifi.apache.org/docs/nifi-docs/.

FlowFile: Data that the user brings into NiFi for processing and distribution is referred to as a FlowFile. A FlowFile is made up of two components: Attributes and Content. Content is the User Data itself and Attributes are the key-value pairs that are associated with the User Data. All FlowFiles have the following Standard Attributes:

- **uuid**: A unique identifier for the FlowFile
- **filename**: A filename that may be used when storing the data to disk or in an external service
- **path**: A path that can be used when storing data to disk or an external service.

Processor: The Processor is the NiFi component that is responsible for creating, sending, receiving, transforming, routing, splitting, merging, and processing FlowFiles. It is the most important building block to build a DataFlow.

Relationship: Each Processor has a set of defined Relationships that it can send data to. When a Processor finishes handling a FlowFile, it transfers the FlowFile to one of the Relationships; thus, user can configure how to handle FlowFiles based on the result of Processing. For example, many Processors define two Relationships: *success* and *failure*. Users are then able to configure data to be routed through the flow one way if the Processor successfully processes the data or another if fails to process the data.

Connection: A Connection allows data to be routed from one Processor to another. Each connection houses a FlowFile queue to which a FlowFile resides when it is being transferred to another Relationship.

Provenance: Provenance is a record of what has happened to the FlowFile.

7.2 NIFI SERVICE

ACS installed NiFi as a service on each of the NiFi hosts. The NiFi service can be started or stopped or its running status can be checked using the **systemctl** command on a NiFi host as follows:

```
# systemctl stop nifi
# systemctl start nifi
# systemctl status nifi
```

7.3 NIFI UI

When NiFi service is started, user is able to navigate to the UI by opening the URL `http://<nifi>:6434/nifi` in a web browser. The port can be changed by editing the **nifi.web.http.port property** in the **/opt/nifi/default/conf/nifi.properties** config file on the NiFi host (be sure to restart the NiFi service after modifying any properties).

When navigate to the UI for the first time, a blank canvas is provided on which a DataFlow can be built:

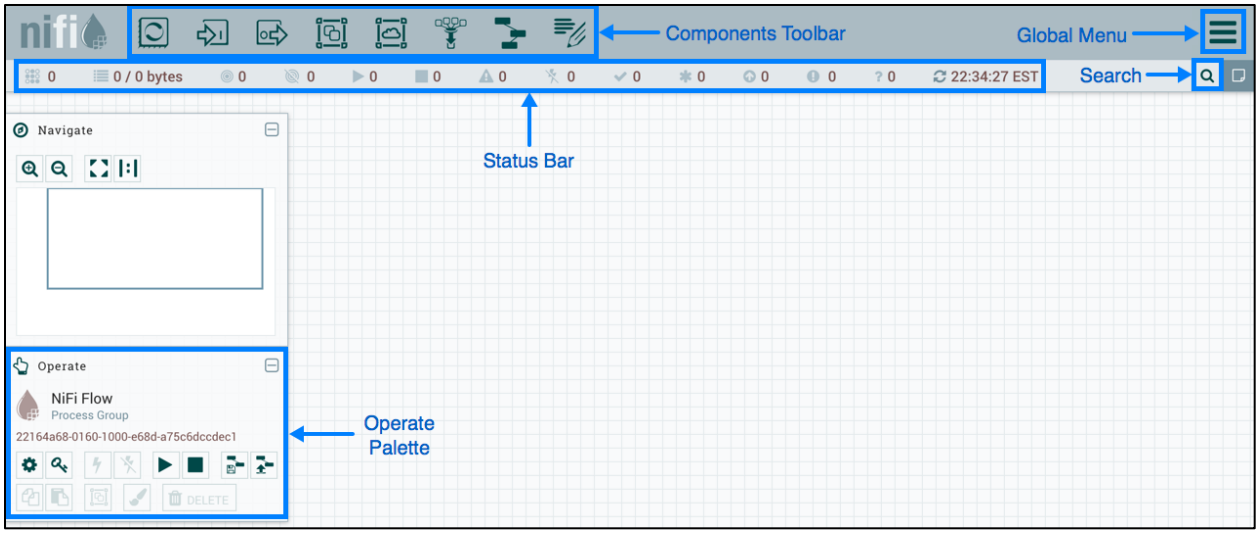


Figure 52 DataFlow NiFi Canvas

As an example to build a DataFlow and demonstrate how it works, we will create a DataFlow that reads flight data from a public HTTPS website, wraps it in a thin wrapper for serialization and to map the data to an appropriate data model, and then sends it to a data pipeline as seen in the following flow diagram. The HTTPS web site is https://adsbexchange.com/ that has the world's unfiltered flight data. In this example, data will be restricted to flights within 100km of Phoenix, AZ airport.

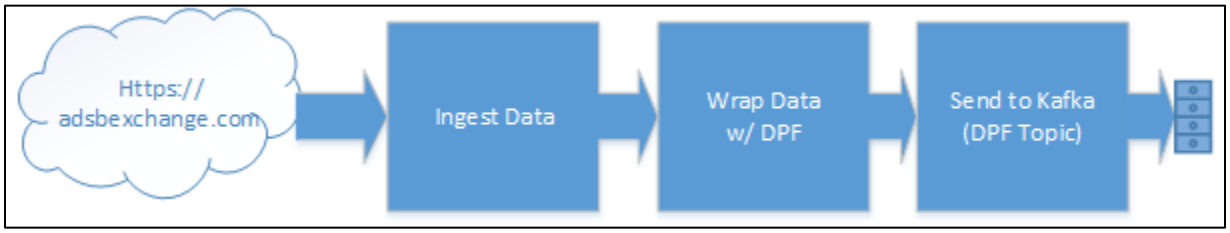


Figure 53 Data Flow Example

7.4 CREATING DATAFLOW

Three NiFi processors need to be created for the DataFlow in this example:

1. **GetHTTP**: continuously feeds data from an HTTPS endpoint
2. **DPFInputWrapper**: creates the data wrapper

3. **PublishKafka**: sends data to data pipeline (Kafka topic: dpf)

7.4.1 GetHTTP Processor

The **GetHTTP** Processor connects to an HTTPS endpoint and processes the response as a FlowFile. The **GetHTTP** Processor needs to be configured with the target host that we want to connect to and SSL protocol information.

7.4.1.1 Create GetHTTP Processor

Create a new processor by dragging the Processor icon ( ) from the Component toolbar onto the canvas to open a dialog to choose which type of Processor to use. Select the **GetHTTP** Processor by searching for http in the upper-right search bar. As seen in the following figure, the **GetHTTP** Processor is for fetching data from an HTTP or HTTPS URL. In this example, the URL to feed data from is https://public-api.adsbexchange.com/VirtualRadar/AircraftList.json?lat=33.433638&lng=-112.008113&fDstL=0&fDstU=100, which queries flights within 100km of Phoenix, AZ airport. The **GetHTTP** Process will remember the ETag and Last-Modified Date in order to ensure that data is not continually ingested.

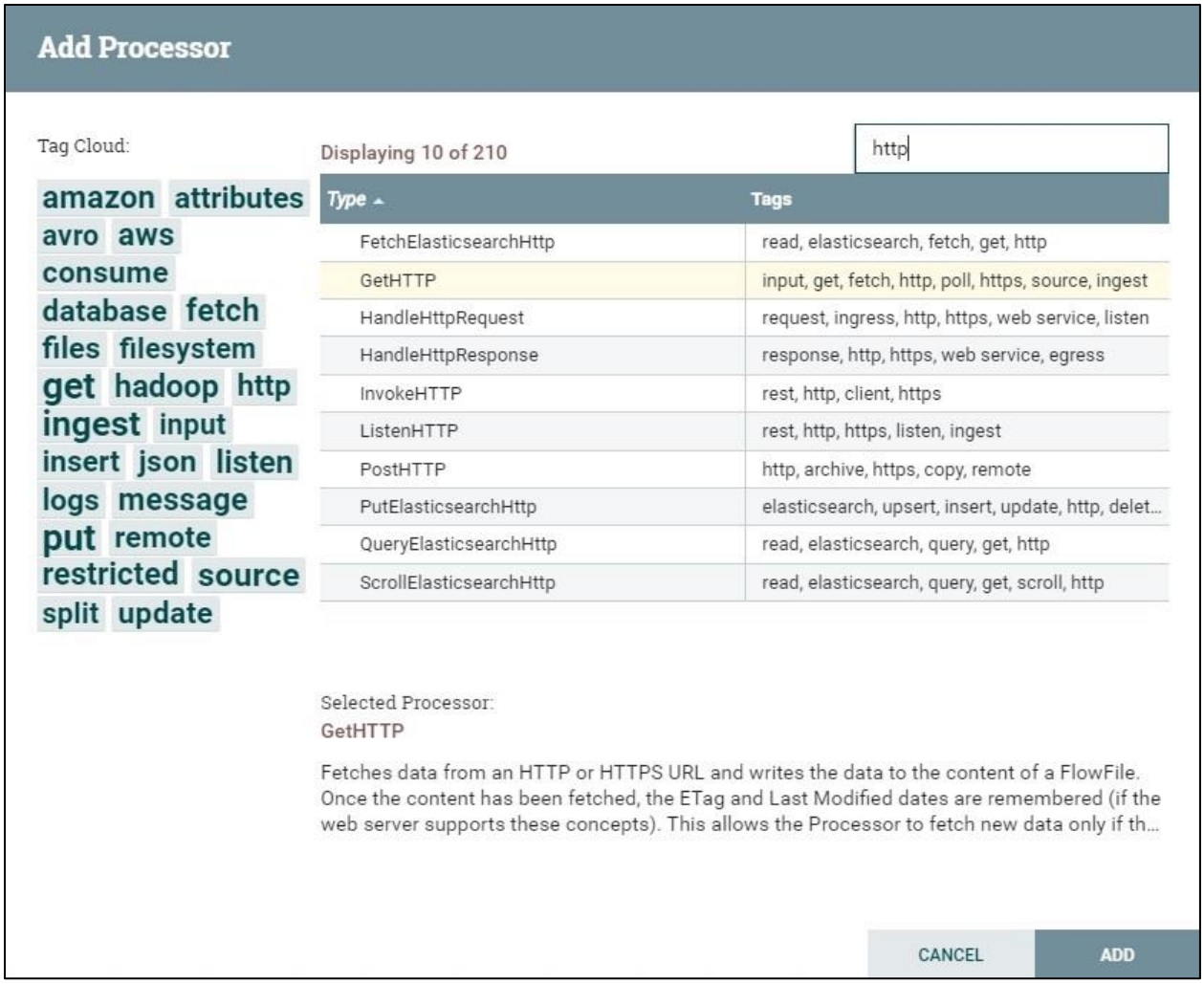


Figure 54 Add GetHTTP Processor

Highlight **GetHTTP** then click the **Add** button or double-click on the **GetHTTP** Processor name to add the **GetHTTP** Processor to the canvas at the location that it was dropped.

7.4.1.2 Configuring GetHTTP Processor

Now that the **GetHTTP** Processor has been added, the Processor needs to be configured with the target host that we want to connect to by right-clicking on the **GetHTTP** Processor and choosing **Configure** menu item from the context menu:

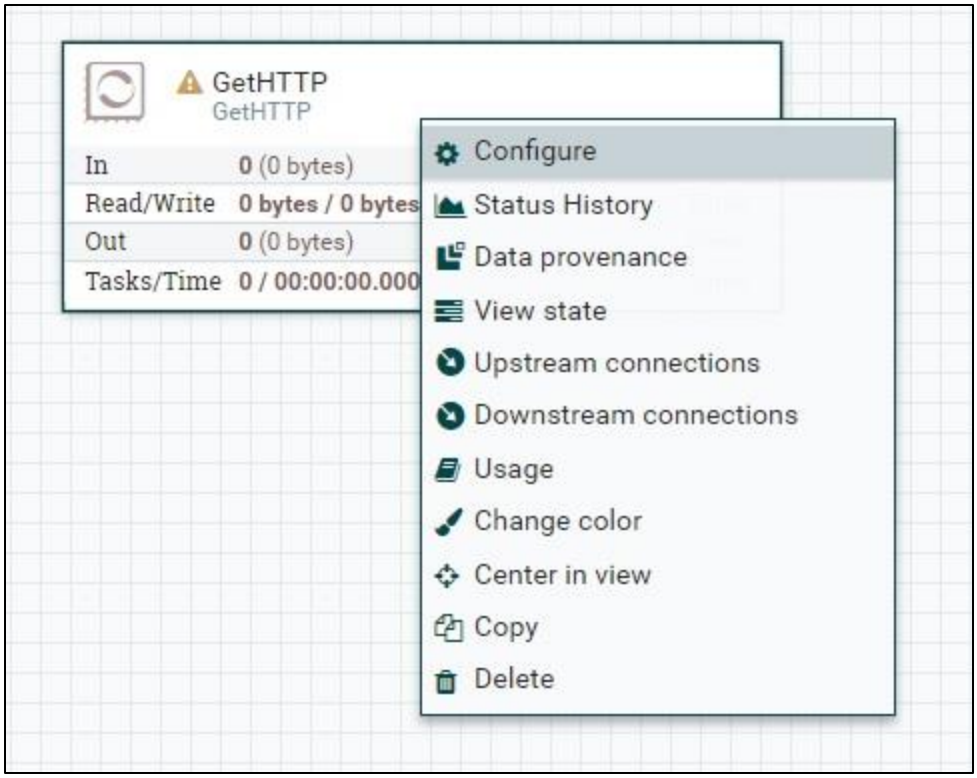


Figure 55 Configure GetHTTP Processor

Navigate to the **Properties** tab then click on the value cell corresponding to each of the following fields and set its property as follows:

- URL: https://public-api.adsbexchange.com/VirtualRadar/AircraftList.json?lat=33.433638&lng=-112.008113&fDstL=0&fDstU=100
- Filename: **flights.json**

7.4.1.3 Adding SSL Service

Because the URL that feeds flight data is HTTPS, keystore and truststore need to be configured to communicate with SSL protocols.

Click on **SSL Context Service** value cell on the **Properties** tab of the **GetHTTP Configure Processor** dialog and select **Create New Service** from the dropdown menu:

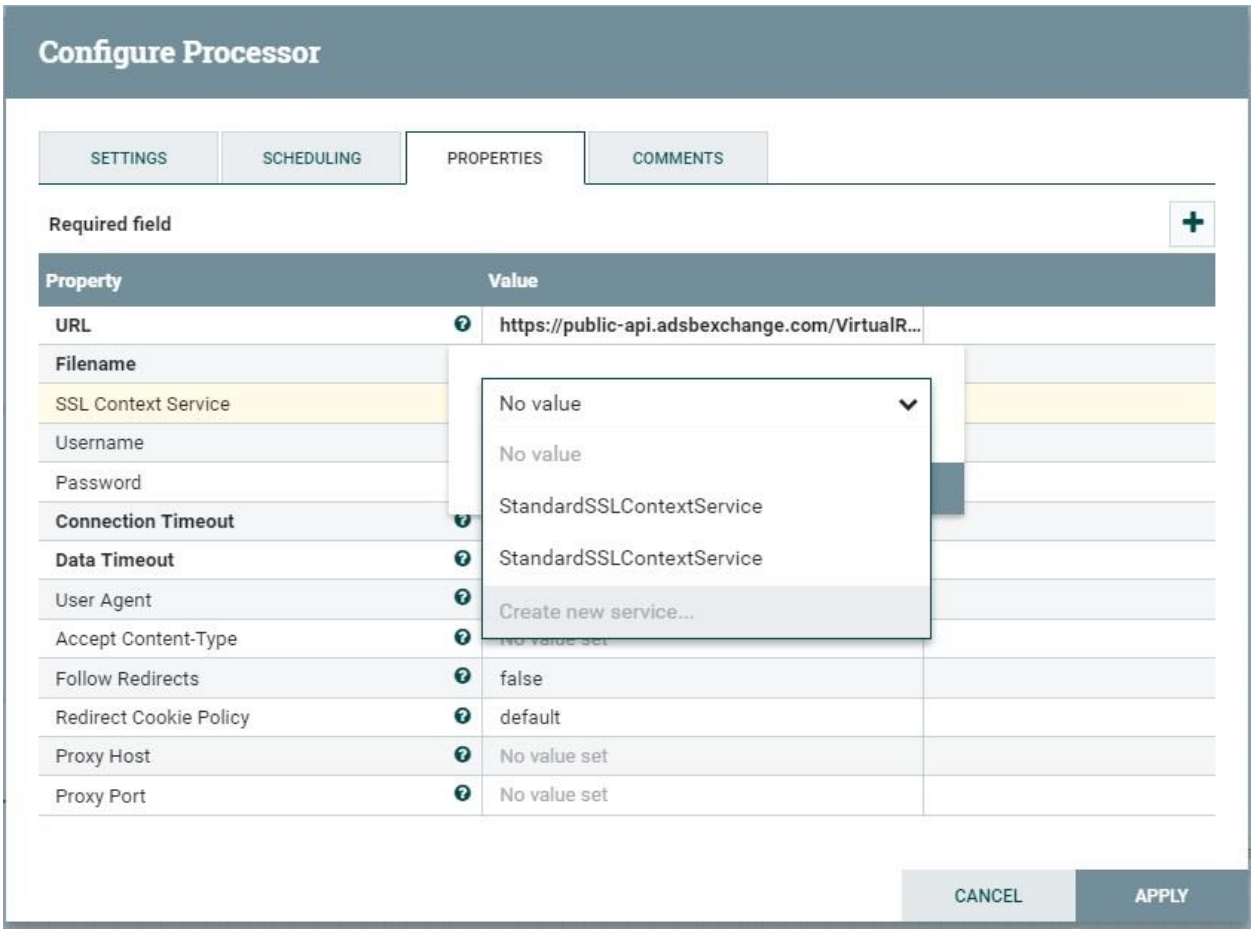


Figure 56 Create New SSL Service

Select **StandardSSLContextService** from the **Controller Service** dropdown menu, then click the **Create** button. As described in the following figure, Standard SSL Context Service provides the ability to configure keystore and/or truststore properties once and reuses that configuration throughout the application. This service can be used to communicate with both legacy and modern systems. If only need to communicate with non-legacy systems, the StandardRestrictedSSLContextService is recommended as it only allows a specific set of SSL protocols to be chosen.

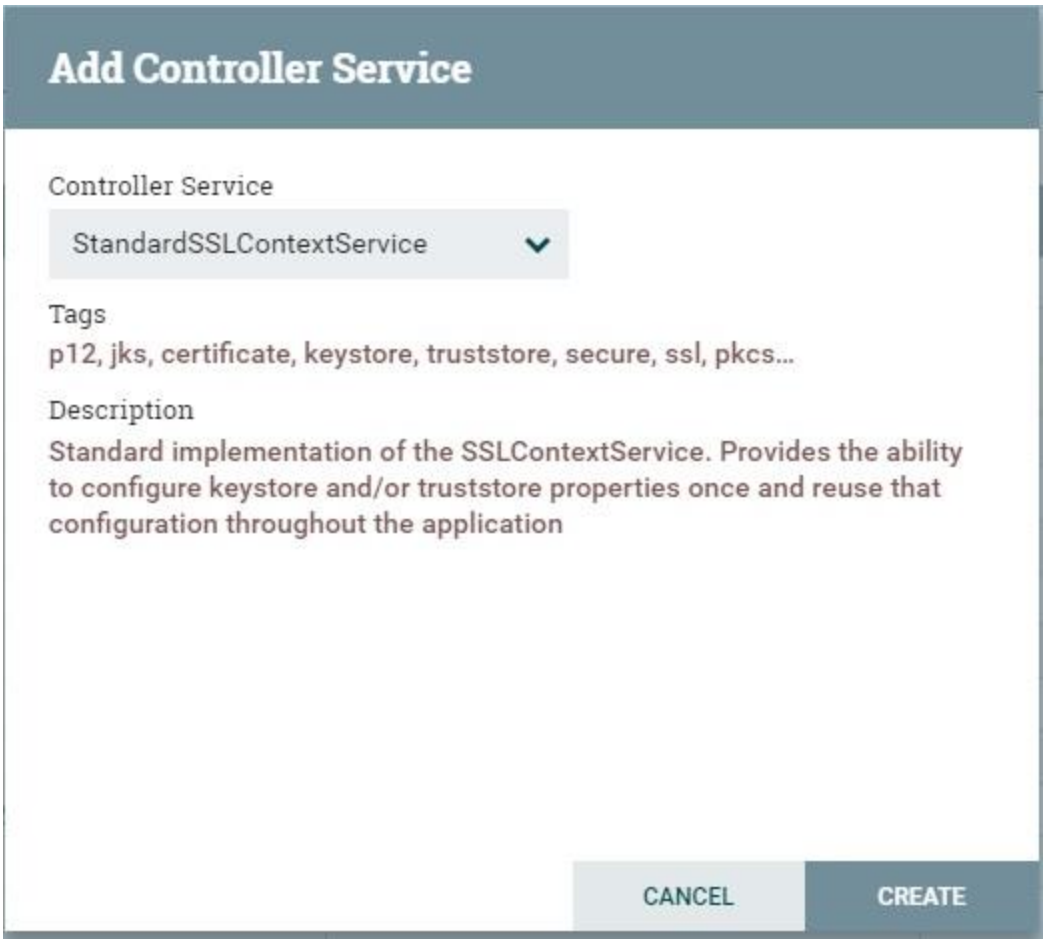


Figure 57 Add Standard SSL Context Service

Click the Go To icon (→) next to the SSL Context Service's value cell. When prompted to save configuration changes, click **yes**.

Click on the pencil ( ) icon to edit the SSL Context Service properties.

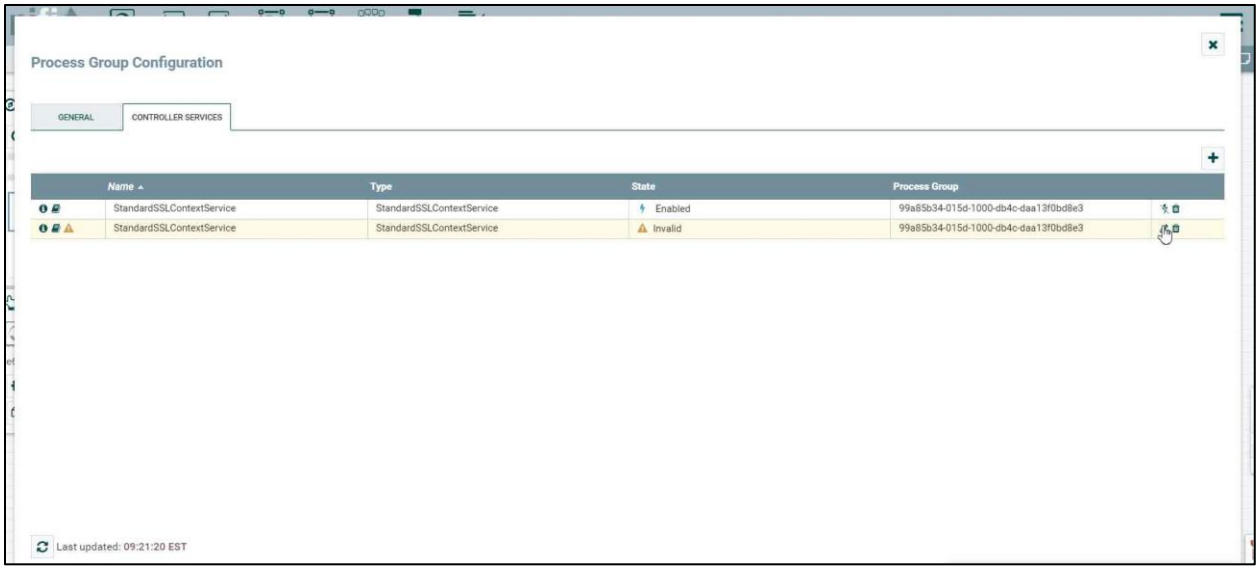


Figure 58 Edit SSL Context Service

Set the following properties:

- Truststore Filename: **/usr/lib/jvm/java-openjdk/jre/lib/security/cacerts**
- Truststore Password: ######## (refer to truststore_pass value in tcri.yml config file on the basebox)
- Truststore Type: **JKS**
- Truststore Protocol: **TLS**

Click the **Apply** button.

> The **/usr/lib/jvm/java-openjdk/jre/lib/security/cacerts** truststore is the standard store included as part of the installation of the JRE. It contains a common set of commercial CA certificates that are useful or required for applications working in the commercial world on the internet. However, in real DoD production environments, this file may not be present in ACS. Rather, another truststore is created that contains the DoD CA certificates and is meant to be used for "real" mission applications. This store is found at **/etc/pki/java/truststore.jks**. For the purposes of this demo, the former will be used.

Click the lightning bolt ( ) icon to enable the **SSL Context Service**.

Under **Scope**, select **Service** only, then click **Enable** button. Click **Close** button once the service is enabled.

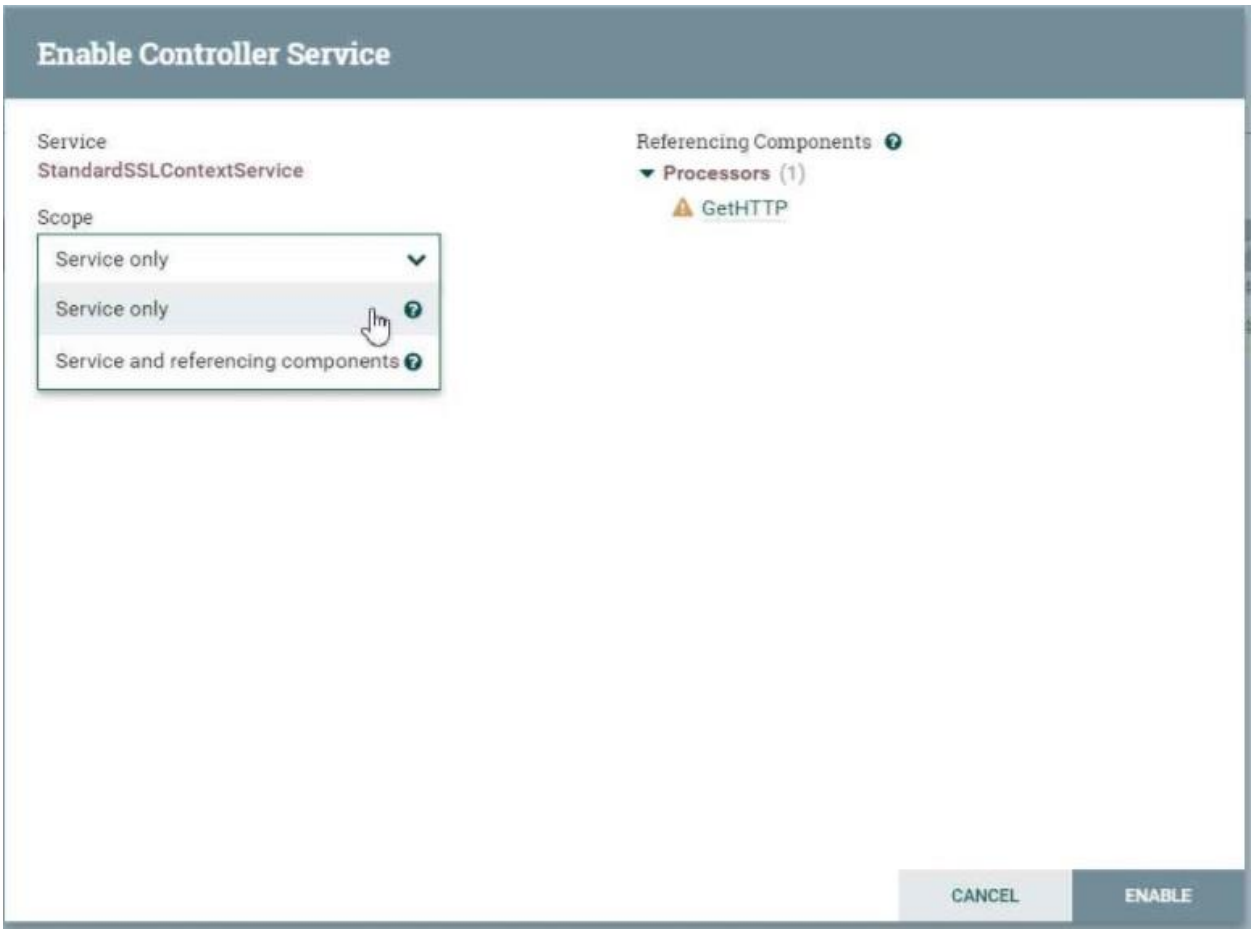


Figure 59 Enable Controller Service

Now, the **GetHTTP** Processor with **SSL Context Service** has been created.

7.4.2 DPFInputWrapper Processor

Next we ill create a processor that will wrap the flight data payload to prepare it for ingest by the Data Processing Framework (DPF) Storm topology. The purpose of this wrapper is to enable serialization of the payload from NiFi to Kafka to Storm. It also puts it into a common format that the DPF topology expects. Finally, the wrapper includes some metadata about the payload, such as the id of the artifact model that should be used for reading and mapping the data. This id is set using the "DPF Model Lookup Key" in the DPFInputWrapper Processor. The key corresponding to the artifact model for the flight data is **demo-flight-geo**.

Follow the similar steps above to create a new Processor, **DPFInputWrapper,** with the following properties:

- DPF Model Lookup Key: **demo-flight-geo**

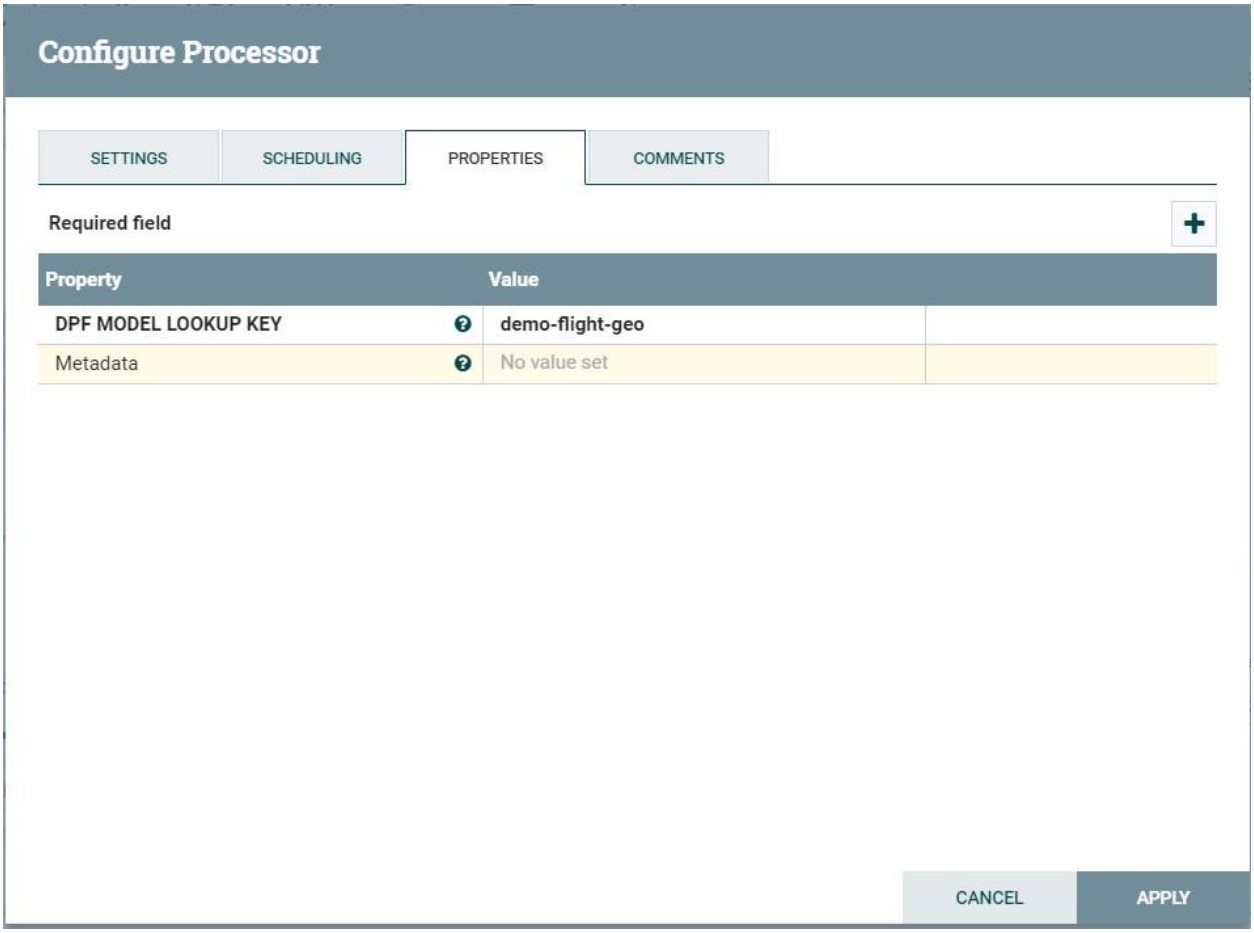


Figure 60 Create DPFInputWrapper Processor

7.4.3 PublishKafka Processor

Lastly, we will send the flight data content as a message to the Apache Kafka servers (default port is 9092) under the topic name '**dpf**'.

Create a new Processor, **PublishKafka,** with the following properties:

- Kafka Brokers: **<kafka_hostname>:9092**
- Topic Name: **dpf**

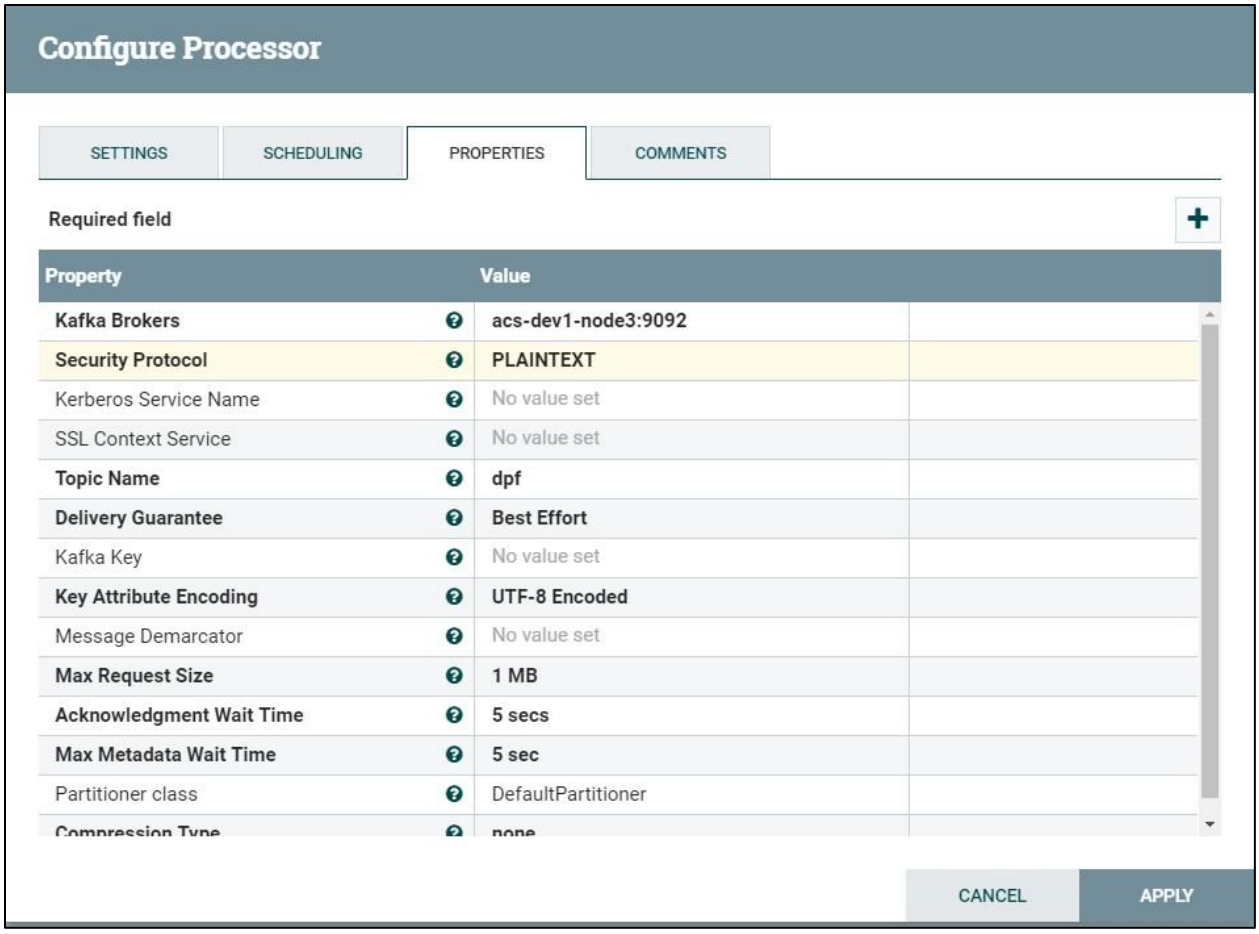


Figure 61 Create PublishKafka Processor

7.4.4 Connecting Processors

Once the three processors have been added to the canvas and configured, we will connect the processors as described for this example. This is accomplished by create a Connection between each component. When user hovers the mouse over a processor component, a Connection icon ( ) appears. User drags the Connection bubble from one component to another until the second component is highlighted then release the mouse. A 'Create Connection' dialog appears with two tabs: 'Details' and 'Settings'. On the Details Tab, user can configure whether the connection is for FlowFile that was successfully processed.

Create the following connections for this example:

- Success connection from **GetHTTP** to **DPFInputWrapper**
- Success connection from **DPFInputWrapper** to**PublishKafka**
- Failure connection from **PublishKafka** to itself

7.4.5 Auto-Terminate Relationship

For a Processor to be valid and able to run, each Relationship defined by the processor must be either connected to a downstream component or auto-terminated. If a Relationship is auto-terminated, any FlowFile that is routed to that Relationship will be removed from the flow and its processing considered complete. Any Relationship that is already connected to a downstream component cannot be auto-terminated.

Auto-Terminate Relationship can be specified by modifying the Settings tab on the Configure Processor dialog. For this example, the following Auto-Terminate Relationships need to be added:

- DPFInputWrapper: Auto-Terminate on failure
- PublishKafka: Auto-Terminate on success

The DataFlow for this example will now look like the following diagram.

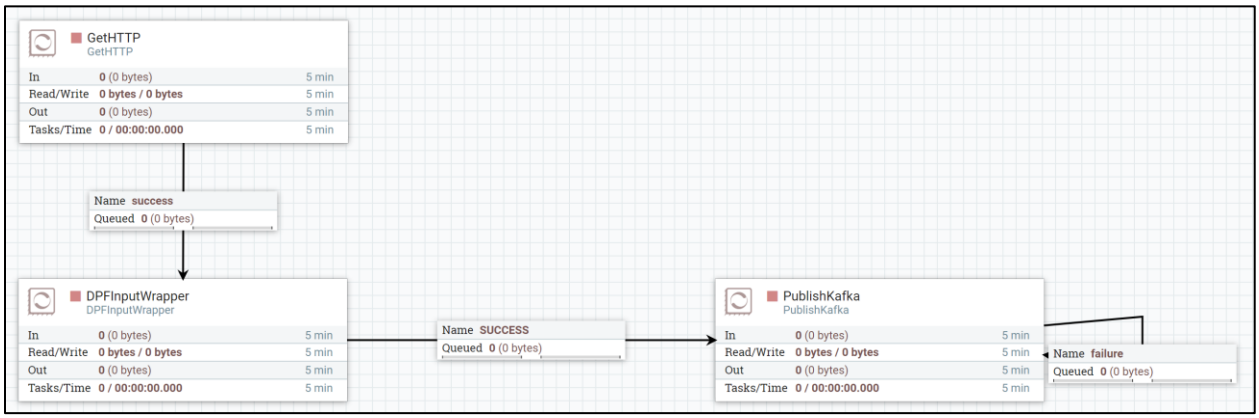


Figure 62 Auto-Terminate Example DataFlow

7.5 TESTING DATAFLOW

7.5.1 Processor Validation

Before starting a Processor, the Processor's configuration must be valid. A status indicator is shown in the top-left of the Processor.

An Alert icon ( ) indicates an invalid Processor, i.e. there is a problem with the Processor configuration. In this case, mouse over the indicator icon will provide a tooltip showing all the validation errors for the Processor.

Once, all the validation errors have been corrected, the status indicator will change to a Stopped icon ( ), indicating the Processor is valid and ready to be started but currently not running.

If the Processor is started and running, the status indicator will change to a Running icon,

7.5.2 Starting and Stopping Processors

A single Processor can be started by right-clicking on the Processor and choose the **Start** menu item. Alternatively, a group of Processors can be started by Shift-selecting the Processors then clicking the Start icon in the Operation pallet.

Once started, the Processors' status indicators will change to the Running icon. The Processor or Processors can then be stopped by selecting the Processor(s) and clicking the Stop icon in the Operate pallet.

Start the Processors in the example for a few seconds then stop them and observe that some (six as in the following figure) FlowFiles have moved between the Processors.

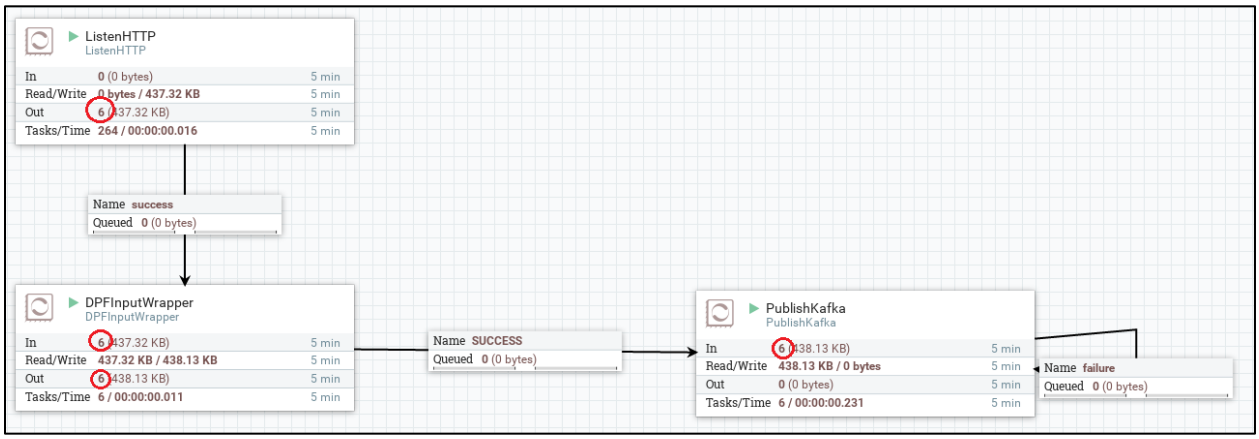


Figure 63 Start/Stop Example DataFlow

7.5.3 Data Provenance

While monitoring a DataFlow, user may need to determine what happened to a particular FlowFile. Data Provenance provides that information. A Processor's Data Provenance can be accessed by right-clicking on the Processor and selecting the **Data Provenance** menu item.

In the above example, we expect DPFInputWrapper Processor wrapping DPF information around the flight data. Let's examine the data on the DPFInputWrapper Processor. Bring up the Data Provenance dialog then click the View Details icon ( ) on one of the events to bring up the Provenance Event dialog window with three tabs: Details, Attributes, and Content. The Content tab shows information about the FlowFile's content, such as its location in the Content Repository and its size. User may click the **Download** button to download a copy of the FlowFile's content as it existed at this point in the flow or click the **View** button to examine the FlowFile's content.

Since the **DPFInputWrapper** Processor modified the FlowFile's content, user will see the 'before' and 'after' content claims.

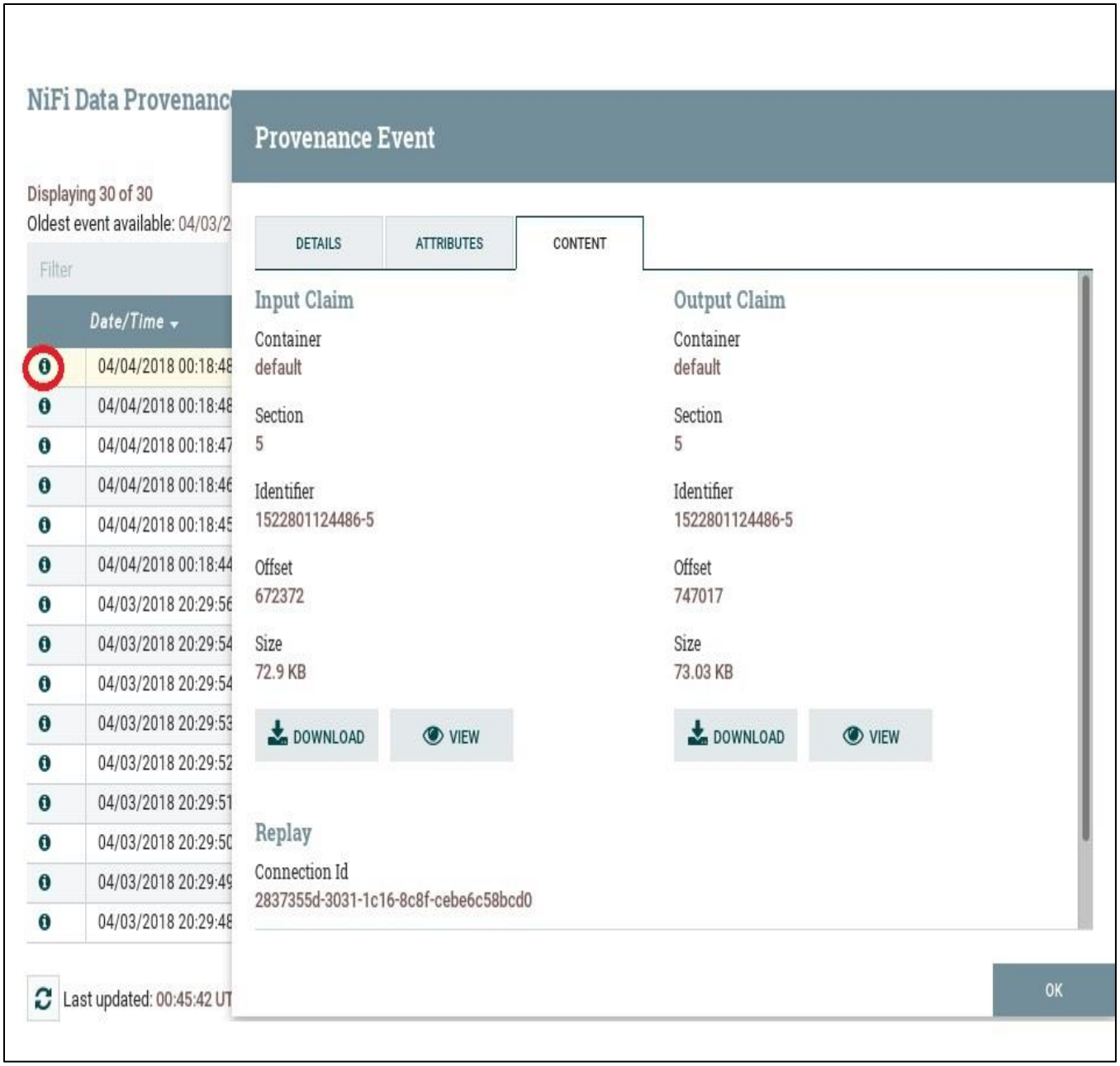


Figure 64 Data Provenance

Additionally, there is a **Replay** button that allows the user to re-insert the FlowFile into the flow and re-process it from the exactly the point at which the event happened. If the results are not as expected, the flow can be modified and re-processes the FlowFile again.

7.5.4 Lineage Graph

In addition to viewing the details of a Provenance event, user can view the path a FlowFile took within the dataflow by clicking on the **Show Lineage** icon ( ) on the far-right column of the Data Provenance table. This opens a graph displaying the FlowFile and various processing events that happened to the FlowFile as it traversed the flow.

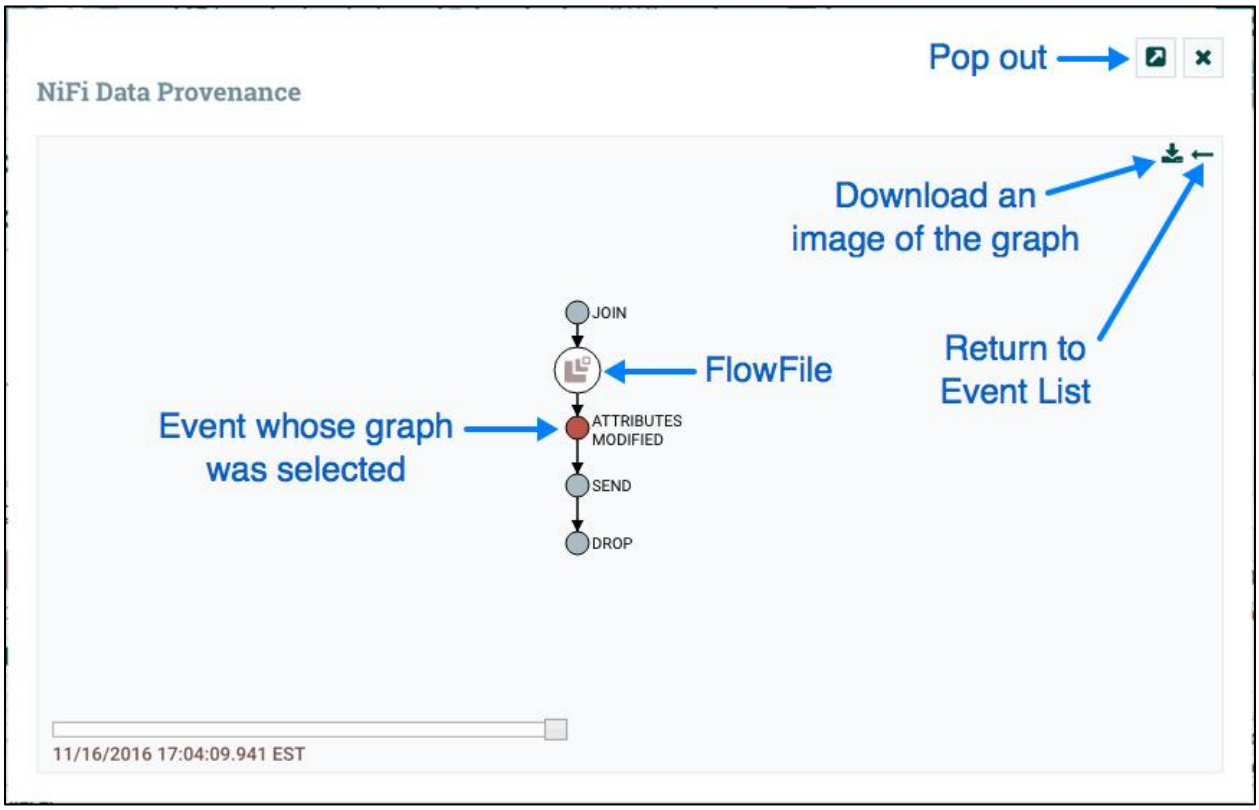


Figure 65 Data Lineage

From here, user can right-click on any events and click **View Details** menu item to view the event details.

7.6 REST API

The NiFi REST API provides programmatic access to command and control a NiFi instance in real time. REST APIs can be used to start and stop processors, monitor queues, query provenance data, and more.

The URL to access the NiFi API in ACS is http://<nifi>:6434/nifi-api. The REST endpoints are described in https://nifi.apache.org/docs/nifi-docs/rest-api/index.html. Followings are some examples of NiFi REST APIs.

7.6.1 System Diagnostics

The REST endpoint to get the diagnostics for the system NiFi is running on is **/system-diagnostics**.


```
{     "systemDiagnostics":{
      "aggregateSnapshot":{
         "totalNonHeap":"545.99 MB",
```

```
        "totalNonHeapBytes":572514304,
        "usedNonHeap":"515.76 MB",
        "usedNonHeapBytes":540817088,
        "freeNonHeap":"30.23 MB",
        "freeNonHeapBytes":31697216,
        "maxNonHeap":"-3 bytes",
        "maxNonHeapBytes":-3,
        "nonHeapUtilization":"-8.47367083E8%",
        "totalHeap":"3 GB",
        "totalHeapBytes":3221225472,
        "usedHeap":"1.61 GB",
        "usedHeapBytes":1724158424,
        "freeHeap":"1.39 GB",
        "freeHeapBytes":1497067048,
        "maxHeap":"6 GB",
        "maxHeapBytes":6442450944,
        "heapUtilization":"27.0%",
        "availableProcessors":6,
        "processorLoadAverage":0.35,
        "totalThreads":348,
        "daemonThreads":198,
        "flowFileRepositoryStorageUsage":{
           "freeSpace":"87.45 GB",
           "totalSpace":"149.93 GB",
           "usedSpace":"62.48 GB",
           "freeSpaceBytes":93898190848,
           "totalSpaceBytes":160982630400,
           "usedSpaceBytes":67084439552,
           "utilization":"42.0%"
        },
        "contentRepositoryStorageUsage":[
           {
              "identifier":"default",
              "freeSpace":"87.45 GB",
              "totalSpace":"149.93 GB",
              "usedSpace":"62.48 GB",
              "freeSpaceBytes":93898190848,
              "totalSpaceBytes":160982630400,
              "usedSpaceBytes":67084439552,
              "utilization":"42.0%"
           }
        ],
        "garbageCollection":[
           {
              "name":"G1 Old Generation",
              "collectionCount":0,
              "collectionTime":"00:00:00.000",
              "collectionMillis":0
           },
           {
              "name":"G1 Young Generation",
              "collectionCount":17054,
              "collectionTime":"00:04:14.868",
              "collectionMillis":254868
           }
        ],
        "statsLastRefreshed":"21:11:51 UTC",
        "versionInfo":{
           "javaVendor":"Oracle Corporation",
           "javaVersion":"1.8.0_161",
           "osName":"Linux",
```

```
            "osVersion":"3.10.0-693.21.1.el7.x86_64",
            "osArchitecture":"amd64",
            "buildTag":"nifi-1.1.2-RC1",
            "buildRevision":"",
            "buildBranch":"",
            "buildTimestamp":"04/07/2017 15:36:33 UTC",
            "niFiVersion":"nifi-1.1.2"
         }
      }
   }
}
```


7.6.2 Flow Status

Current status of this NiFi can be queried using the endpoint **/flow/status** and the result may look like the following:


```
{
   "controllerStatus":{
      "activeThreadCount":0,
      "queued":"0 (0 bytes)",
      "flowFilesQueued":0,
      "bytesQueued":0,
      "runningCount":3,
      "stoppedCount":0,
      "invalidCount":0,
      "disabledCount":0,
      "activeRemotePortCount":0,
      "inactiveRemotePortCount":0
   }
}
```


7.6.3 Querying Processor

A Processor running status can be queried using the REST endpoint **/flow/processors/{processor-id}/status** where the processor id can be obtained from the **Settings** tab of the **Configure Processor** dialog window. The data returned from the query is shown below indicating that the **DPFInputWrapper** Processor is in Running state.


```
{
   "processorStatus":{
      "groupId":"40213b5d-0162-1000-ac77-9cd5fbecc859",
      "id":"265f3012-8d6e-13d3-95d7-8bbc157fd780",
      "name":"DPFInputWrapper",
      "statsLastRefreshed":"21:53:59 UTC",
      "aggregateSnapshot":{
         "id":"265f3012-8d6e-13d3-95d7-8bbc157fd780",
         "groupId":"40213b5d-0162-1000-ac77-9cd5fbecc859",
         "name":"DPFInputWrapper",
         "type":"DPFInputWrapper",
         "runStatus":"Running",
         "bytesRead":0,
         "bytesWritten":0,
```

```
            "read":"0 bytes",
            "written":"0 bytes",
            "flowFilesIn":0,
            "bytesIn":0,
            "input":"0 (0 bytes)",
            "flowFilesOut":0,
            "bytesOut":0,
            "output":"0 (0 bytes)",
            "taskCount":0,
            "tasksDurationNanos":0,
            "tasks":"0",
            "tasksDuration":"00:00:00.000",
            "activeThreadCount":0
        }
    },
    "canRead":true
}
```

8. KAFKA MESSAGING

Apache Kafka provides a streaming platform with three key capabilities:

- Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system
- Store streams of records in a fault-tolerant durable way
- Process streams of records as they occur

8.1 KAFKA SERVICE

Like NiFi, Kafka was installed in ACS as a service on each of the Kafka servers. The Kafka service can be started or stopped or its running status can be checked using the **systemctl** command on a Kafka server as follows:

```
# systemctl stop kafka-server
# systemctl start kafka-server
# systemctl status kafka-server
```

8.2 KAFKA TOPIC

Kafka cluster stores streams of records in categories called topics. Each record consists of a key, a value, and a timestamp.

8.2.1 Listing Topics

ACS installation created a few Kafka topics. To view the list of current topics, execute the **kafka-topics.sh** command from the **/opt/kafka/kafka-current/bin** directory of a Kafka server.

```
# ./kafka-topics.sh --list --zookeeper localhost
GEO_ALL
RYA_ALL
RYA_POST_PERSIST
SO_PERSISTED
UCD
__consumer_offsets
attachment
c3.ingest.events
depotEventTopic
directingest
dpf
drif-event
geomesa-ds-kafka-AIS_MSG
geomesa-ds-kafka-ais_oth
geomesa-ds-kafka-ais_oth_so
geomesa-ds-kafka-dwell-sft
geomesa-ds-kafka-exif_so
geomesa-ds-kafka-twitter
mil.army.inscom.ucd.streams.All
mil.ucd_plus_artifact
```

```
monitor-ais-oth
nlp_ut_body
prov_all
provenance
so-pre-persist
ua
ucd_plus_artifact
```

The **--zookeeper** option is required and specifies the connection string for the zookeeper connection in the form host:port. Default port will be used if not specified. Multiple URLs separated by commas (,) can be given to allow fail-over. For example if node3 and node4 are the name of the zookeeper hosts, the option will be **--zookeeper node3,node4**.

8.2.2 Viewing a Topic

Notice **dpf** is one of the active topics and it is also the topic to which we sent flight data at the end of the previous DataFlow example. To list flight data messages, execute the following command.

```
# ./kafka-console-consumer.sh --zookeeper localhost --topic dpf --from-beginning
```

User can skip the **--from-beginning** option if they are only interested in new messages. Note that each of the flight data messages is wrapped with DPF information as it was produced by the **DPFInputWrapper** Processor above.

8.2.3 Creating a Topic

A new topic can be created, for example, to create a topic named "test" with a single partition and only one replica as follow:

```
# ./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --
partitions 1 --topic test
```

9. STORM TOPOLOGY

9.1 OVERVIEW

Apache Storm is an open source & distributed streaming processing computation framework. Storm makes it easy to reliably process unbounded streams of data in a fault-tolerant and horizontal scalable method. It is a streaming data framework that has the capability of high ingestion rates. Though Storm is stateless, it manages its distributed environment and cluster state via Apache ZooKeeper.

9.1.1 Storm Architecture

One of the main highlights of Apache Storm is that it is a distributed application that is fast and fault-tolerant, without a single point of failure. A Storm cluster is comprised of a set of Storm nodes which can be expanded to meet increased demand of applications.

There are two kinds of nodes in a Storm cluster: a master node and worker nodes.

Master Node (Nimbus)

The master node runs a daemon called the Nimbus. The Nimbus is responsible for distributing code around the cluster, assigning tasks to machines, and monitoring for failures.

The Nimbus service relies on Apache ZooKeeper to monitor the message processing task as all the workers nodes update their tasks status in the Apache ZooKeeper service.

Worker Nodes (Supervisor)

Each worker node runs a daemon called the Supervisor. The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what the Nimbus has assigned to it. Each worker process executes a subset of a topology; a running topology consists of many worker process spread across many machines.

All coordination between the Nimbus and the Supervisors is done through a ZooKeeper cluster. Additionally, the Nimbus daemon and Supervisor daemons are fail-fast and stateless. Even though the stateless nature has its own disadvantages, it helps Storm process real-time data in the best possible and quickest way.

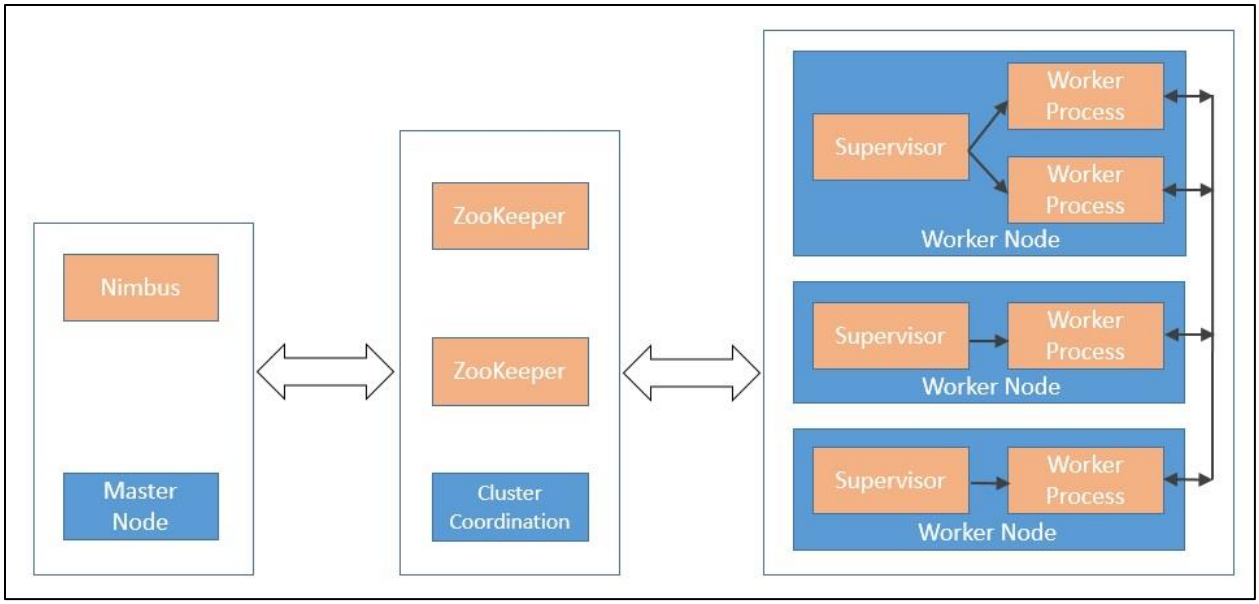


Figure 66 Storm Cluster

9.1.2 Topologies

Topologies are created in Storm to do real-time computations. Each node in a topology contains processing logic (bolts) and links between the nodes indicate how the data should be pass around the nodes (streams). In simple words, a topology is a directed graph where vertices are computation and edges are streams of data.

A simple Storm topology starts with spouts. Spouts emits data to one or more bolts. A bolt represents a node in the topology having the smallest processing logic and the output of a bolt can be emitted into another bolt as an input. The main data structure in Storm is called a tuples. A stream of tuples flows from spout to bolts or in between bolts. In Storm there are various streaming grouping techniques that defines how data should flow in a topology like global grouping, etc.

When a topology is submitted to a Storm cluster, the Nimbus service on the master node consults the supervisor services on different worker nodes and submits the topology. Each supervisor will then create one or more worker processes called the Executors.

The following diagram shows the basic data flow through a typical Storm topology.

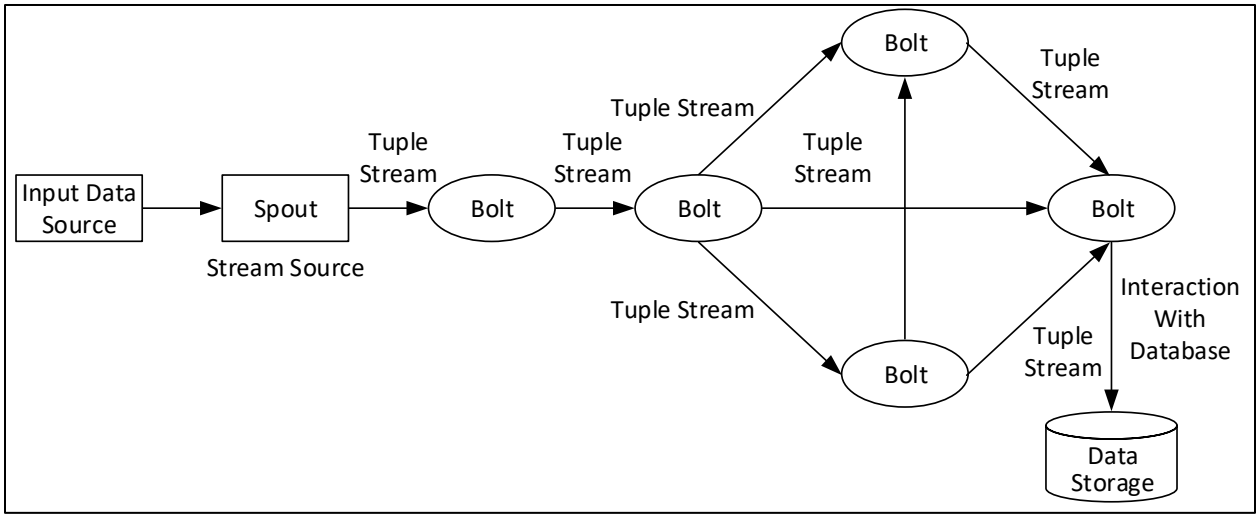


Figure 67 Data Flow through a typical Storm topology

Table 8 Component Description

Components	Description
Tuple	Tuple is the main data structure in Storm. It is a list of ordered elements. By default, a Tuple supports all data types. Generally, it is modelled as a set of comma separated values and passed to a Storm cluster.
Stream	Stream is an unordered sequence of tuples.
Spouts	Source of stream. Generally, Storm accepts input data from raw data sources like Twitter Streaming API, Apache Kafka queue, Kestrel queue, etc. Otherwise you can write spouts to read data from datasources.
Bolts	Bolts are logical processing units. Spouts pass data to bolts and bolts process and produce a new output stream. Bolts can perform the operations of filtering, aggregation, joining, interacting with data sources and databases. Bolt receives data and emits to one or more bolts.

Topology Breakdown

Storm distinguishes between the following three main entities that are used to actually run a topology in a Storm cluster:

1. Worker processes
2. Executors (threads)
3. Tasks

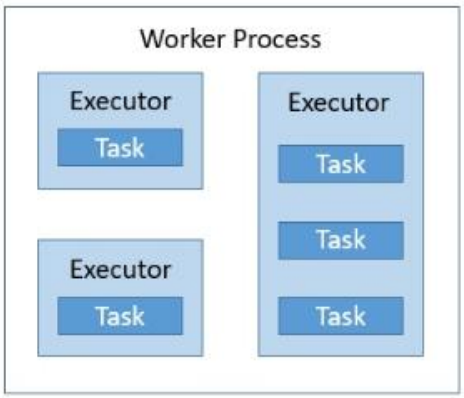


Figure 68 Worker Process

Here is a simple illustration of their relationships:

A worker process executes a subset of topology. A worker process belongs to a specific topology and may run one or more executors for one or more components (spouts and bolts) of this topology. A running topology consists of many such processes running on many machines within a Storm cluster.

An executor is a thread that is spawned by a worker process. It may run one or more tasks for the same component (spout or bolt).

A task performs the actual data processing - each spout or bolt that you implement in code executes as many tasks across the cluster. The number or tasks for a component is always the same throughout the lifetime of a topology, but the number of executors (threads) for a component can change over time.

By default, the number of tasks is set to be the same as the number executors, i.e. Storm will run one task per thread.

9.2 STORM UI

The Storm UI depicts some vital and important aspects of the Storm cluster and topology. Some of the information provided by Storm UI can be used for optimizing performance of the cluster. The sections below provides a short explanation of the parameters in the Storm UI.

The Storm UI can be reached at http://<storm_master_node>:8080/index.html.

9.2.1 Storm UI Landing Page

This is the landing page for storm. If you make an http GET through the browser to the storm http you will get this page. This gives you a summarized view of the cluster.

Cluster Summary

Cluster Summary

Version	Supervisors	Used slots	Free slots	Total slots	Executors	Tasks
1.0.2	2	7	1	8	86	86

Figure 69 Cluster Summary

- Version - The version of storm installed on the UI node.
- Supervisors - The number of nodes in the cluster currently.
- Used slots - Number of workers that are occupied.
- Free slots - Number of workers which are free.
- Total slots - Number of workers in the cluster [You have to define the number of slots per machine and its most of the time same as number of cores in the machine]
- Executors - Number of threads. These reside in the worker processes. You can define how many tasks are assigned to an executor.
- Tasks - A Task is an instance of a Bolt or Spout.

Topology Summary

Topology Summary

Name	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)	Scheduler Info
drif-storm-topology	storm	ACTIVE	8d 19h 36m 52s	1	11	11	1	832	

Figure 70 Topology Summary

- Name - The name given to the topology by when it was submitted. Click this name to view the Topology's information.
- Status - This denotes the state of the topology, which could be *active* for a live topology, *killed* when a topology is killed using the UI or CLI, *inactive* for a deactivated topology, and *rebalancing* for a topology where the rebalance command is executed wherein the number of workers allocated to the topology is increased or decreased.
- Uptime - The time since the Topology was submitted.
- Num workers - Number of workers used in the current topology. This is similar to what is defined in topology configuration.
- Num executors - Number of executors used in the current topology.
- Num tasks - Number of tasks used in the current topology.

Supervisor Summary

Supervisor Summary

Host	Id	Uptime	Slots	Used slots	Used Mem (MB)	Version
node2.sd.spawar.navy.mil	44ce8c1f-b779-4071-a6b2-b3c66be76f1f	3h 21m 40s	4	4	6656	1.0.2
node3.sd.spawar.navy.mil	83705759-f9c0-42a0-a935-d0bf5baf612e	3h 21m 15s	4	3	5824	1.0.2

Figure 71 Supervisor Summary

- Host - The hostname reported by the remote host. (Note that this hostname is not the result of a reverse look-up at the Nimbus node.)
- Id - A unique identifier given to a Supervisor when it joins the cluster.
- Up-time - The length of time a Supervisor has been registered to the cluster.
- Slots - Number of workers in the subject host. Normally storm evenly distributes the workers among all the hosts (nodes).

- Used Slots - Number of workers that are occupied in the subject host

Nimbus Configuration

- Here the nimbus configuration for the configuration is displayed.

9.2.2 Storm UI Topology Home

Topology Summary

Topology summary

Name	Id	Owner	Status	Uptime	Num workers	Num executors	Num tasks	Replication count	Assigned Mem (MB)
Streaming-Data-Processing-Framework	Streaming-Data-Processing-Framework-9-1524145503	storm	ACTIVE	7d 1h 30m 51s	1	9	9	1	832

Figure 72 Storm UI Topology Summary

This is similar to the Topology summary table on the Storm Landing Page.

- Name - The name given to the topology by when it was submitted. Click this name to view the Topology's information.
- id - This is the Storm-generated unique ID that is a combination of the topology name, timestamp, and ID, which is used by Storm to identify and differentiate the topology.
- Status - This denotes the state of the topology, which could be *active* for a live topology, *killed* when a topology is killed using the UI or CLI, *inactive* for a deactivated topology, and *rebalancing* for a topology where the rebalance command is executed wherein the number of workers allocated to the topology is increased or decreased.
- Uptime - The time since the Topology was submitted.
- Num workers - Number of workers used in the current topology. This is similar to what is defined in topology configuration.
- Num executors - Number of executors used in the current topology.
- Num tasks - Number of tasks used in the current topology.

Topology Actions

Figure 73 Topology Actions

- Activate - The UI provides a facility to revive or reactivate a topology that has been suspended earlier. Once activated, it can again start consuming the messages from the spout and process them.
- Deactivate - When this action is executed, the topology immediately turns off the spout, that is, no new messages are read from the spout and pushed downstream to the DAG. Existing messages that are already being processed in various bolts are processed completely.

- Re-balance - This action is executed when the worker allocation to a live topology is altered.
- Kill - This is used to send a termination signal for the topology to the Storm framework. It's always advisable to provide a reasonable kill time so that the topology drains completely and is able to clean the pipelined events before it terminates.

Topology Stats

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
10m 0s	0	0	0		
3h 0m 0s	0	0	0		
1d 0h 0m 0s	0	0	0		
All time	0	0	0		

Figure 74 Topology Stats

- Window - This field specifies the time duration in the following segments: last for 10 minutes, last 3 hours, the past day, or all the time. The topology's progress is captured against these time sections.
- Emitted - This captures the number of tuples emitted by the spout at various time segments.
- Transferred - This specifies the number of tuples sent to other components in the topology. Please note that the number of emitted tuples may or may not be equal to the number of transferred tuples as the former is the exact number of times the emit method of the spout is executed, while the latter is the number transferred based on the grouping used; for example, if we have bound a spout to a bolt that has the parallelism of two tuples using all grouping, then for every `x` tuples emitted by the spout, `2x` tuples will be transferred.
- Complete latency - This is the average total time taken by a tuple to execute throughout the topology.
- Ack - This holds the number of events acked that are successfully processed.
- Failed - This is the number of events that failed to process successfully.

Spouts (All time)

Spouts (All time)

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
00.Filesystem-Path-Selector.hdfs	1	1	0	0	0.000	0	0				

Figure 75 Spouts All Time

- Id - The ID assigned to a Component by the Topology. Click on the name to view the Component’s page.
- Executors - Number of executors assigned for spout.
- Tasks - Number of tasks assigned for spout.
- Emitted - The number of Tuples emitted.
- Transferred - The number of Tuples Transferred.
- Complete latency - The average time a Tuple “tree” takes to be completely processed by the Topology. A value of 0 is expected if no acking is done.

- Acked - The number of Tuple “trees” successfully processed. A value of 0 is expected if no acking is done.
- Failed - The number of Tuple “trees” that were explicitly failed or timed out before acking was completed. A value of 0 is expected if no acking is done.

This section of the page is good to monitor the number of tuples emitted and transferred. As more tuples are sent and acknowledged, the number shown under the emitted and transferred should increase.

Bolts (All time)

Bolts (All time)

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error
00.Filesystem-File-Loader	1	1	0	0	0.000	0.000	0	0.000	0	0			

Figure 76 Bolts All Time

- Id - The ID assigned to a Component by the Topology. Click on the name to view the Component’s page.
- Executors - Number of executors assigned for the bolt.
- Tasks - Number of tasks assigned for the bolt.
- Emitted - The number of Tuples emitted from this bolt.
- Transferred - The number of Tuples Transferred from this bolt.
- Capacity - If this is around 1.0, the corresponding Bolt is running as fast as it can, so you may want to increase the Bolt’s parallelism. This is (number executed * average execute latency) / measurement time.
- Execute latency - The average time a Tuple spends in the execute method. The execute method may complete without sending an Ack for the tuple.
- Executed - The number of incoming Tuples processed.
- Process latency - The average time it takes to Ack a Tuple after it is first received. Bolts that join, aggregate or batch may not Ack a tuple until a number of other Tuples have been received.
- Acked - The number of Tuples acknowledged by this Bolt.
- Failed - The number of tuples Failed by this Bolt.

This is also another section of the page to monitor the number of tuples emitted and transferred. Again, as more tuples are sent and acknowledged, the number shown under the emitted and transferred should increase.

Topology Visualization

In the Storm UI, click the Show Visualization button to display a visual representation of your topology and to find the data bottleneck in the Storm application. The descriptions given describes the diagram below.

- The blue component (circle) represents the first component in the topology, such as the DPF spout below from the Data Processing Framework topology.
- The colors of the other topology components indicates whether the component is exceeding cluster capacity: red components denote a data bottleneck and green components indicate components are operating within capacity.
- The lines in between components represents data flow. The thicker lines between components denote a larger data flow while slimmer lines denotes a smaller data flow.

With the topology visualization mapping, data can be visually shown as data continuously gets ingested passing through the Kafka-Reader.dpf spout, Processing bolt, and GeoSOPersistenceBolt. The thicker lines in between the spout and bolts show that data is flowing between them.

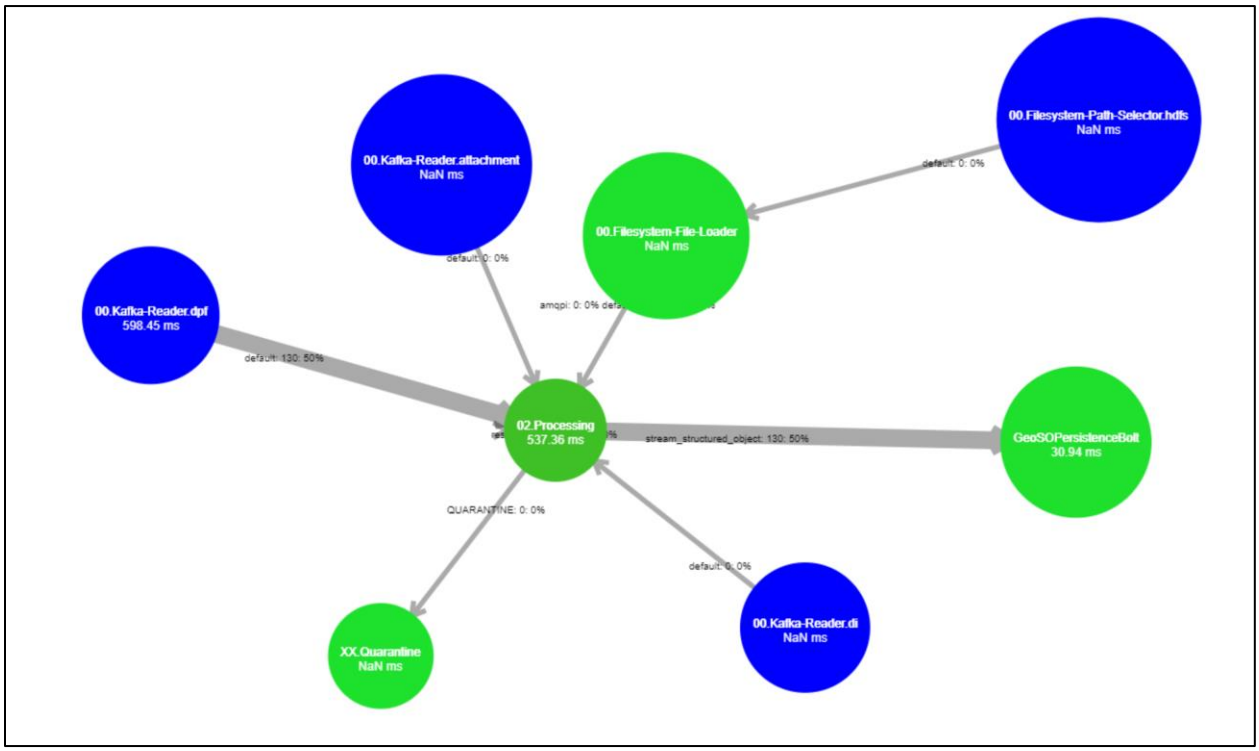


Figure 77 Topology Visualization

10. ARTIFACT AND DOMAIN MODEL

10.1 ARTIFACT MODEL

The artifact model represents the mapping instructions for extracting data from ingested sources and populating entities. Artifact models contain instructions to the DPF topology on how to map ingested sources to structured object data. Artifact models can reference XML, JSON, or DSV files.

10.1.1 Artifact Model File Requirements

The four required files needed to process an artifact model are:

- Artifact Model file - .am.xml
- Artifact Model Lookup file - .lookup
- Artifact Model Repo file - .repo
- Artifact Model Type Discovery file - .td.xml

A short description of the required artifact model files are detailed below.

10.1.2 Artifact Model File

The components of the artifact model file are listed.

- properties
- artifact
- security
- metadata
- content
- attachments
- domainMapping
- structuredObjectMapping

A short description of the components are explained below.

properties

The properties section provides information on the Streaming DPF topology regarding document format. At its simplest, the properties element describes the source document as a XML format and provides information on the character set to be used in parsing the document content:

`<properties>`
` <document format="xml">`
` <charset>utf-8</charset>`
` <xmlns prefix="sec"`
`uri="urn:us:gov:ic:sec"/>`
` </document>`
` </properties>`
	`<properties>`
	` <document format="json">`
	` <charset>utf-8</charset>`
	` <xmlns prefix="sec"`
	`uri="urn:us:gov:ic:sec"/>`
	` </document>`
	` </properties>`

A more complex example of a properties element describes a document as a delimited format, and designates the expressions used as column delimiters and line/row delimiters:

```
<properties>
    <document format="dsv">
        <dsv trimWhitespace="true">
            <delimiter>
                <column xml:space="preserve" type="literal">,</column>
                <row xml:space="preserve" type="regex">&#xD;?&#xA;</row>
            </delimiter>
            <headerRow>0</headerRow>
            <quote>
                <quoteString>&quot;</quoteString>
                <quoteEscape>&quot;</quoteEscape>
            </quote>
        </dsv>
    </document>
</properties>
```

The properties section of the model also allows contextual variable to be set for later use in the data mapping sections of the document:

```
<properties>
    <document format="dsv">
        <dsv trimWhitespace="true">
            <delimiter>
                <column xml:space="preserve" type="literal">,</column>
                <row xml:space="preserve" type="regex">&#xD;?&#xA;</row>
            </delimiter>
            <headerRow>0</headerRow>
            <quote>
                <quoteString>&quot;</quoteString>
                <quoteEscape>&quot;</quoteEscape>
            </quote>
        </dsv>
    </document>
</properties>
```

artifact

The artifact section of the model describes how data should be mapped to the metadata and data sections.

security

The security section describes how marking information should be derived for the artifact.

```
<security>
    <classification eval="el">
         var class = xpath:getText("/Message/SEC/Classification");
         var info = xpath:getText("/Message/SEC/AmplifyInfo");
         return class + "//" + info;
    </classification>
</security>
```

A simpler example would be to assignment of a document-wide literal classification:

```
<security>
    <classification eval="literal">Unclassified</classification>
</security>
```

metadata

The metadata section provides for control over how select metadata fields are populated. The meta data describe source data attributes that are not necessarily directly related to source data content, such as pedigree information (source), content type and format, source language, and message size (automatically derived).

```
<metadata>
    <language eval="literal">en_US</language>
    <contentType>
        <primaryType eval="literal">text</primaryType>
        <subType eval="literal">json</subType>
    </contentType>
    <contentUri eval="literal">abc:def:ingest-artifact-model</contentUri>
</metadata>
```

structuredData

It is possible to ingest data without mapping it to Domain Model elements by mapping to content/structuredData section in the artifact model. The structuredData section provides the means to map data to a simple map of key-value pairs. Since the key and value elements of the structuredData section are also mappings, it is possible to map data to this section using the same evaluator type used in mapping data to terms with the domainMapping section (xpath, el, literal...): Mapping to structured data is desirable in instances where query for artifacts based on known artifact content is advantageous, or where the over of the mapping source data to Domain Model elements too costly for a given data type.

```
<structuredData>
    <dataDescriptor eval="literal">tcri:prov</dataDescriptor>
    <properties>
        <property>
            <key eval="literal">Model</key>
            <value eval="literal">tcri:demo-flight-geo</value>
        </property>
        <property>
            <key eval="literal">icao</key>
            <value eval="jsonPath" required="false">$..Icao</value>
        </property>
        <property>
            <key eval="literal">latitude</key>
            <value eval="jsonPath" required="false">$..Lat</value>
        </property>
        <property>
            <key eval="literal">longitude</key>
            <value eval="jsonPath" required="false">$..Long</value>
        </property>
    </properties>
</structuredData>
```

attachments

The attachments section describes the information needed for retrieval and serialization of embedded or external data to which the source artifact refers. Commonly encountered examples attachments might include video clips, imagery, audio clips, or human-readable documents. Often these formats are serialized in line with their metadata (Base64 encoded, for example), or referenced via hyperlink or filesystem location.

```
    <attachments>
        <external>
            <path eval="el">context:get("contentUri")</path>
            <artifact>
                <metadata>
                    <fileName eval="el">context:get("contentUri")</fileName>
                    <contentType>
                        <primaryType eval="literal">video</primaryType>
                        <subType eval="literal">mpeg</subType>
                    </contentType>
                </metadata>
            </artifact>
        </external>
    </attachments>
```

The example above illustrates an externally accessible attachment residing on a filesystem at a location within the source metadata artifact. Other attachment types include embedded (inline encoded) and external (accessible via URI).

domainMapping

There is only one domainMapping section allowed within an artifact model. The domainMapping tag contains a linkage attribute that identifies the domain model with which the current Artifact Model should be associated (the domain model describing the majority of the mapped Terms). In the domaingMapping element, it is possible to define one or more instances. Each instance can define a linkage to a domain model.

```
    <domainMapping linkage="tcri:demo-flight-geo">
        <instance linkage="tcri:demo-flight-geo">
            <scope eval="jsonPath">$.acList[*]</scope>
            <term handle="artifact_key">
                <primitive eval="el" type="string" required="false">
                    return context:getArtifactKey();
                </primitive>
            </term>
            <term handle="latitude">
              <primitive eval="jsonPath" type="double"
required="false">$..Lat</primitive>
            </term>
            <term handle="longitude">
              <primitive eval="jsonPath" type="double"
required="false">$..Long</primitive>
            </term>
            <term handle="icao">
              <primitive eval="jsonPath" type="string"
required="false">$..Icao</primitive>
            </term>
        </instance>
```

```
    </domainMapping>
```

Each instance is assigned a scope to identify containers for potential nested repeating elements within the raw data. Instances may also be nested within the domainMapping section, to reflect the hierarchical nature of some source formats.

The additional three files needed to process an artifact model are mostly pointer files which points to specific Data Processing Framework lookup key and repository files needed.

10.1.3 Artifact Model Lookup File

The lookup file points to the lookup key that should be used on the on the command line when an ingest job and also the name of the type discovery file which will be described below.

```
inputs:
-  lookupKey: 'demo-flight-geo'
   typeDiscovery: 'demo-flight-geo.td.xml'
```

10.1.4 Artifact Model Repository File

The repo file points to the artifact model am.xml file created. Both sub-type and label fields should be left as '*'. The type should be equal to the type specified in the type discovery td.xml file.

```
artifactModelFile: 'demo-flight-geo.am.xml'
typeAssociations:
-  type: 'demo-flight-geo'
   subtype: '*'
   label: '*'
```

10.1.5 Artifact Model Type Discovery File

The type discovery section has a properties section which provides information on the Streaming DPF topology regarding document format and also a typeMetadata section

```
<?xml version="1.0"?>
<typeDiscovery xmlns="urn:dpf:ingest-artifact-type-discovery">

    <properties>
        <document format="json"/>
    </properties>

    <typeMetadata>
        <type eval="literal">demo-flight-geo</type>
        <subtype eval='literal'>JSON</subtype>
        <label eval="literal">demo-flight-geo</label>
    </typeMetadata>
</typeDiscovery>
```

10.2 DOMAIN MODEL

The process of building structured objects in the Data Processing Framework (DPF) involves extracting data from a document and making relationships. The extracted data is a representation of some piece of knowledge. The extracted data along with an associated metadata are mapped in the artifact model which was explained in the previous section. The relationships and associations of these Terms are defined in one or more domain models.

10.2.1 Domain Model File Requirements

The four required files needed to process an artifact model are:

- Domain Model file - .dm.xml
- Domain Model Concept Definition file - .cd.xml
- Domain Model Predicates Definition file - .pd.xml
- Domain Model Repository file - .repo

A short description of the required domain model files are detailed below.

10.2.2 Domain Model File

The components of the domain model file are listed.

- name
- term
- statement
- structuredObject

A short description of the components are explained below.

name

The name section contains the name of the domain model.

```
<name>tcri:demo-flight-geo</name>
```

term

In the term section, the domainMapping section listed in the artifact model will need to be included in the key terms in the term definition section of the domain model. This section is the domain model where the key term originates is listed and the dataformat (type) is also given.

For example, if the domainMapping section listed in the artifact model is the following:

```
<domainMapping linkage="tcri:demo-flight-geo">
    <instance linkage="tcri:demo-flight-geo">
        <scope eval="jsonPath">$.acList[*]</scope>
        <term handle="artifact_key">
            <primitive eval="el" type="string" required="false">
                return context:getArtifactKey();
            </primitive>
```

```
            </term>
            <term handle="latitude">
              <primitive eval="jsonPath" type="double"
required="false">$..Lat</primitive>
            </term>
            <term handle="longitude">
              <primitive eval="jsonPath" type="double"
required="false">$..Long</primitive>
            </term>
        </instance>
    </domainMapping>
```

The term section of the domain model will look like the following:

```
    <term handle="artifact_key">
        <concept label="ARTIFACT_KEY" model="tcri:demo-flight-geo"
dataformat="STRING"/>
    </term>
    <term handle="latitude">
        <concept label="LATITUDE" model="tcri:demo-flight-geo"
dataformat="DOUBLE"/>
    </term>
    <term handle="longitude">
        <concept label="LONGITUDE" model="tcri:demo-flight-geo"
dataformat="DOUBLE"/>
    </term>
```

structuredObject

The structuredObject section of the domain model defines how the structured object will look. There will be times where some of the structured objects will be nested within others, and it should be noted the structured objects should be defined here individually and with a flat structure. The handle listed must be a unique reference to the structured objected. Although not required, it is good practice to have the handle match the kind. For each property key, a unique property key handle and text value should be defined. The key handle will be used for referencing the property within the models and the text value should be a human-readable name. For the property value, the type is defined here. The type given can be one of the following: double, long, integer, wkt, string, and boolean. Each property element can also include an optional "required" attribute. By default it is set to false, but if it is set to true, it will cause the processing of the structured object to fail if this property is not mapped to a value in the artifact model.

```
    <structuredObject handle="demo_flight_geo">
        <kind>demo_flight_geo</kind>
        <properties>
            <property required="true">
                <key handle="artifact_key">artifact_key</key>
                <value type="string"/>
            </property>
            <property>
                <key handle="latitude">latitude</key>
                <value type="double"/>
            </property>
```

```
            <property>
                <key handle="longitude">longitude</key>
                <value type="double"/>
            </property>
        </properties>
    </structuredObject>
```

The additional three files needed to process a domain model are mostly pointer files which points to repository files needed and a concept definition file.

10.2.3 Domain Model Concept Definition file

The concept definition file defines a list of concepts that may be referenced in mappings to a domain model. The concept is identified by a key, consisting of the domain model name it pertains to, the concept name, and the dataformat. An optional description element provides for the insertion of documentation describing the concept. Since concepts within a model have the potential to describe sensitive system characteristics, a security field allows the concept to be flagged, in terms of its classification. An optional origin field allows for providing a reference to external documentation or a link to an external source. And finally, the associations element allows the model to define zero or more associations between the concept being described and concepts in the current or other domain models.

```
<?xml version="1.0" encoding="utf-8"?>
<concepts xmlns="urn:dpf:concept-predicate-model">
    <concept>
        <key label="ARTIFACT_KEY" model="tcri:demo-flight-geo"
dataformat="STRING"/>

        <description>The message identifier</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="LATITUDE" model="tcri:demo-flight-geo" dataformat="DOUBLE"/>

        <description>The aircraft's latitude over the ground</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="LONGITUDE" model="tcri:demo-flight-geo" dataformat="DOUBLE"/>

        <description>The aircraft's longitude over the ground</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
</concepts>
```

10.2.4 Domain Model Predicates Definition file

The predicates definition file is used to build up statements in various domains. In ACS, the domain model predicates definition file is mostly empty, like the listing below.

```
<predicates xmlns="urn:dpf:concept-predicate-model">
</predicates>
```

10.2.5 Domain Model Repository file

The repository file points to the domain model am.xml, concept definitions, and predicate definitions files created.

```
domainModelFiles:
- 'demo-flight-geo.dm.xml'

conceptDefinitions:
- 'demo-flight-geo.cd.xml'

predicateDefinitions:
- 'demo-flight-geo.pd.xml'
```

11. HADOOP / HDFS

The HDFS (Hadoop Distributed File System) is a distributed, scalable, and portable file system written in Java for the Hadoop framework. HDFS stores large files (typically in the gigabytes to terabytes range) across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence theoretically does not require RAID (Redundant Array of Independent Disks) storage on hosts. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and the DataNodes that store the actual data.

11.1 HADOOP SERVICES

ACS installed 4 daemons for Hadoop:

- **hadoop-hdfs-namenode**: runs on <namenode> and <failover_namenode> for HDFS
- **hadoop-hdfs-datanode**: runs on slave <datanodes> for HDFS
- **hadoop-yarn-resourcemanager**: runs <resourcemgr> and <failover_resourcemgr> for Yarn
- **hadoop-yarn-nodemanager**: runs on slave <nodemgrs> for Yarn

11.2 HDFS OVERVIEW

11.2.1 NameNode and DataNodes

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes which manages storage attaches to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be store in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNodes executes file system namespace operations like opening, closing, and renaming files and directories, and it also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instructions form the NameNode.

11.2.2 Web Interface

The HDFS NameNode and DataNode each run an internal web server to display basic information about the current status of the cluster. With the default configuration, the NameNode's web interface is at http://namenode:50070/. It lists the DataNodes in the cluster and the basic statistics of the cluster. The web interface can also be used to browse the file system (using the "Browse the file system" link under the "Utilities" tab on the NameNode front page).

11.2.3 Shell Commands

Hadoop includes various shell-like command that directly interact with HDFS and other file systems that Hadoop support. The command "hdfs dfs -help" lists the commands support by the Hadoop shell. Furthermore, the command "bin/hdfs dfs -help command_name" displays more detailed help for a command. These commands support most of the normal files systems

operations like copying files, changing file permissions, etc. It can also support a few HDFS specific operations like changing replication of files, etc. A full list of HDFS commands can be found at https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-hdfs/HDFSCommands.html.

A list of common HDFS command options for users of ACS are below:

dfs

```
Usage: hdfs dfs [[COMMAND [COMMAND_OPTIONS]]
```

Run a filesystem command on the file system supported in Hadoop. The more common COMMANDs used in ACS are found below.

ls

```
Usage: hdfs dfs -ls [-d] [-h] [-R] [-t] [-S] [-r] [u] <args>
```

Options:

- -d: Directories are listed as plain files.
- -h: Format file sizes in a humun readable fashion(eg 64.0m instead 67108864).
- -R: Recursively list subdirectories encountered.
- -t: Sort output by modification time (most recent first).
- -S: Sort output by file size.
- -r: Reverse the sort order.
- -u: Use access time rather than modification time for display and sorting.

mkdir

```
Usage: hdfs dfs -mkdir [-p] <paths>
```

Options:

- The -p option behavior is much like Unix mkdir -p, creating parent directories along the path.

Example:

- hdfs dfs -mkdir /user/hadoop/dir1 /user/hadoop/dir2

chown

```
Usage: hdfs dfs -chown [-R] [OWNER][:[GROUP]] URI [URI ]
```

Change the owner of files. The user must be a super-user.

Options:

- The -R option will make the change recursively through the directory structure.

put

```
Usage: hdfs dfs -put <localsrc> ... <dst>
```

Copy single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and writes to destination file system.

Example:

- hdfs dfs -put localfile /user/hadoop/hadoopfile

rm

```
Usage: hdfs dfs -rm [-f] [-r | -R] [-skiptrash] URI [URI ...]
```

Delete files specified as args.

Options:

- The -f option will not display a diagnostic message or modify the exit status to reflect an error if the file does not exist.
- The -R option deletes the directory and any content under it recursively.
- The -r option is equivalent to -R.
- The -skipTrash option will bypass trash, if enabled, and delete the specified file(s) immediately. This can be useful when it is necessary to delete files from an over quota directory.

Example:

- hdfs dfs -rm /user/hadoop/directory

11.2.4 Hadoop Configuration Settings

The configuration files for Hadoop are located under the /etc/hadoop/conf directory. A brief explanation of the core-site.xml and hdfs-site.xml configuration is below.

The core-site.xml file information the Hadoop daemon where NameNode runs in the cluster. It contains the configuration settings for Hadoop Core such as I/O settings that are common to HDFS.

The hdfs-site.xml file contains the configuration setting for the HDFS deamons; the NameNode, the Secondary NameNode, and the DataNodes. Default block replication and permission check on HDFS can be configured through the hdfs-site.xml configuration file. Fields of interest from the hdfs-site.xml are described below.

Table 9 HDFS Fields of Interest

Property	Value	Description
dfs.namenode.name.dir	<value>file:///data00/hadoop/hdfs/name</value>	Determines where on the local filesystem the DFS name node should store the name table. If this is comma-delimited list of directories then the name table is replicated in all of the directories for redundancy.
dfs.datanode.data.dir	<value>file:///data01/hadoop/hdfs/data</value>	Determines where on the local filesystem a DFS data node should store its blocks. If this is a comma-delimited list of directories, then data will be stored in all named directories, typically on different devices. Directories that do not exist are ignored.

HDFS: Uploading to HDFS and Installing Custom Artifact / Domain Models

In this section, the steps needed to upload and install custom artifact and domain models will be explained. The artifact and domain models provide the underlying schema mapping to persist the flight data to both the structured objects (SO) datastore and the CJMTK/ArcGIS datastore. More explanation on artifact and domain models will be provided in the Storm section.

To install a custom artifact and domain, the Streaming Data Processing Framework will first have to be stopped through the Storm Web UI. The Storm Web UI can be reached through a web browser through web browser at http://<storm_master_node>:8080/index.html. On the Storm UI main page, a summary of the Storm cluster, numbus, and topology are given. Under the Topology Summary section lists the available topologies through Storm, click on the Streaming-Data-Processing-Framework topology link to enter the Streaming Data Processing Framework topology page.

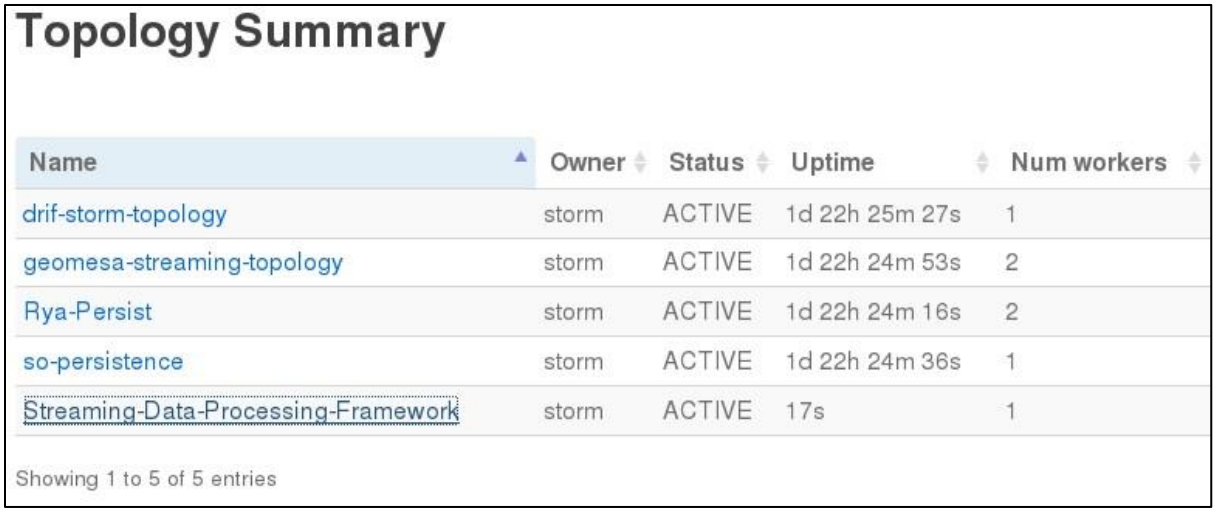


Figure 78 Storm UI Topology Summary

From the Streaming Data Processing Framework topology page, to kill the Streaming-Data-Processing-Framework topology click the 'Kill' button located under the Topology action section of the page.

Figure 79 Storm Topology Kill Button

When prompted to kill the topology and specify a wait time in seconds, enter 1 second and click 'Ok'.

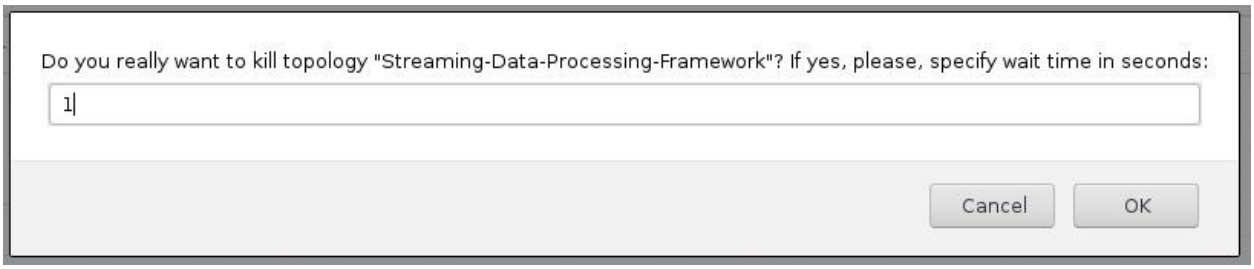


Figure 80 Storm Kill Topology

Once the Streaming-Data-Processing-Framework topology has been killed, the steps to install a custom artifact and domain model can be started. The basics steps to installing a custom model is to login as storm, create the respective artifact and domain files and directories in the /ingest/repository/ directory through HDFS, and restart the streaming DPF topology to process the models. The steps are detailed below.

First SSH into <storm_master>.

Create a new folder named 'demo-flight-geo-art' in the /opt/common/models/tcri/ directory and copy the following files to the newly created directory (The text for these files can be found in Appendix A of this guide). These are the artifact model files which would be installed by Storm once the Storm Topologies are restarted.

- demo-flight-geo.am.xml
- demo-flight-geo.lookup
- demo-flight-geo.repo
- demo-flight-geo.td.xml

Create a new folder named 'demo-flight-geo-dom' in the /opt/common/model/tcri/ directory and copy the following files to the new created directory (The text for these files can be found in Appendix A of this guide). These are the domain model files which would be installed by Storm once the Storm Topologies are restarted.

- demo-flight-geo.cd.xml
- demo-flight-geo.depo
- demo-flight-geo.dm.xml
- demo-flight-geo.pd.xml

Once all the demo-flight-geo directories are created and model are files copied over, a change of ownership to storm: storm is needed for the new directories and files. A change of ownership is needed because the following steps will be run under "storm" user and the user will need access to these artifact and domain files.

```
sudo chown -R storm:storm /opt/common/models/tcri/demo-flight-geo-art/
/opt/common/models/tcri/demo-flight-geo-dom/
```

Become the "storm" user.

```
[root@acs-dev1-node1 tcri]# su - storm
-bash-4.2$
```

Create a demo-flight-geo directories for both flight data artifact and domain files in HDFS. The directory for the artifact model files should be created in the /ingest/repository/artifact/ directory and the domain model files should be created in the /ingest/repository/domain/ directory. These are the directories which will be ingested and the corresponding artifact and domain models will be installed once the Streaming Data Processing Framework topology is restarted. It should also be noted that the HDFS commands are run as the "storm" user because the /ingest/ directory is owned by storm and all artifact and domain model files needs to be owned by storm to process them correctly once the topology is restarted.

```
-bash-4.2$ hdfs dfs -mkdir /ingest/repository/artifact/demo-flight-geo/
-bash-4.2$ hdfs dfs -mkdir /ingest/repository/domain/demo-flight-geo/
```

The newly copied over artifact and domain model files from the local node needs to be transferred over their respective HDFS directories. Use the following HDFS put command to copy over the files.

```
-bash-4.2$ hdfs dfs -put -f /opt/common/models/tcri/demo-flight-geo-art/*
/ingest/repository/artifact/demo-flight-geo
-bash-4.2$ hdfs dfs -put -f /opt/common/models/tcri/demo-flight-geo-dom/*
/ingest/repository/domain/demo-flight-geo
```

Once all artifact and domain models has been copied over to their respective HDFS directories, from the command line the Streaming-Data-Processing-Framework topology can be started by running the start-topology.sh script.

```
bash-4.2$ /opt/storm/topologies/tcri/streaming-dpf/bin/start-topology.sh
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/storm/storm-1.0.2/lib/log4j-slf4j-impl-
2.1.jar!/org/slf4j/impl/StaticLoggerBinder.class]
...
...
4324 [main] INFO  o.a.s.StormSubmitter - Finished submitting topology: Streaming-
Data-Processing-Framework
```

Once the "Finished submitting topology: Stream-Data-Processing-Framework" is display, the newly built demo-flight-geo models has been installed and the Streaming-Data-Processing-Framework topology will show on the Topology Summary section of the Storm UI main page.

Removing Models Manually

There are cases where individual artifact and domain models would have to be removed. The steps required are similar to adding models above but the directories created for the model will be removed instead. The actual commands for removing models manually are shown below. Be sure to stop the Streaming-Data-Processing-Framework topology through Storm UI before executing the commands and restart the topology afterwards to successfully remove the model similar to the steps above.

```
[root@acs-dev1-node1 tcri]# su - storm
-bash-4.2$ hdfs dfs -rm -r -f /ingest/repository/artifact/(the model to remove)/*
-bash-4.2$ hdfs dfs -rm -r -f /ingest/repository/domain/(the model to remove)/*
```

12. STRUCTURED OBJECT DATASTORE

12.1 OVERVIEW

The Structured Object Datastore is an abstraction layer that sits on top of Accumulo and is designed to provide a low level API for CRUD (create, read, update, delete) operations. The Structured Object Store is designed to make it possible to implement ACID (atomicity, consistency, isolation, durability) transactions over specific groupings of objects. Each instance of Structured Objects has the following tables: instance.table.core, instance.table.kind, instance.table.composite, instance.table.ascending, instance.table.descending. The last four tables are indexes for the Structured Objects.

12.2 DATA STRUCTURE

Structured Object data contains one or more properties, with each property able to have one or more values. Properties have a name and datatype similar to how fields are defined in object oriented languages. A unique key identifies each object.

```
<key> : <property>
      : <property>
      : <property>
```

Figure 81 Data Structure

12.2.1 Key Structure

Applications can retrieve an individual Structured Object using the key. The key consists of the following parts: namespace, ancestor path, kind, and id. The namespace allows for multitenancy. The ancestor path locates the object within the hierarchy of objects and consists of a kind, id pair. The kind indicates that the object is a member of a category of structured objects. The id is a kind-specific identifier.

```
<key> === <namespace><ancestor_path><kind><id>
```

Figure 82 Key Structure

12.2.1.1 Key URI Scheme

There is a custom URI (Aug 2015) for encoding the components of a key. The authority segment of the URI contains an identifier of the instance, if the instance identifier is not present it is assumed to be the local instance in the current application context.

```
Absolute
so://instance/namespace/kind;id/kind;id/...

Absolute (local instance)
so:///namespace/kind;id/...
so:///namespace/kind

Relative
/namespace/kind;id/...
kind;id/...
```

Figure 83 Key URI Scheme

12.2.2 Property Structure

Structured Objects contain zero or more "property" fields, each consisting of a label, datatype, and value. If the key of the Structured Object contains a certain "kind" the properties may be controlled by a schema. The "kind" field in the key should be allowed to reserve labels that start and end with a single underscore. An example of the datatypes is presented below.

Value Type	Java Type	Sort Order
Text String (short)	`java.lang.String`	Unicode
Text String (long)	`mil.tcri.structured.store.data.Text`	N/A (not indexed)
Byte String (short)	`mil.tcri.structured.store.data.ShortBlob`	Byte Order
Byte String (long)	`mil.tcri.structured.store.data.Blob`	N/A (not indexed)
Integer	`java.lang.Integer`	Numeric
Long	`java.lang.Long`	Numeric
Floating Point	`java.lang.Float`	Numeric
Floating Point	`java.lang.Double`	Numeric
Boolean	`java.lang.Boolean`	False `<` True
Temporal Instant	`java.util.Date`	Chronological
Temporal Interval	`mil.tcri.structured.store.data.DateInterval`	N/A (not indexed)
Well-Known Text	`mil.tcri.structured.store.data.GeoWKT`	N/A (not indexed)
Well-Known Binary	`mil.tcri.structured.store.data.GeoWKB`	N/A (not indexed)
IPv4 Address	`mil.tcri.structured.store.data.Ipv4`	Unicode
IPv6 Address	`mil.tcri.structured.store.data.Ipv6`	Unicode
Property Key	`mil.tcri.structured.store.datastore.PropertyKey`	Unicode (by label)
Structured Object Key	`mil.tcri.structured.store.datastore.StructuredObjectKey`	Unicode (by key)

Figure 84 Property Structure

12.2.3 Attribute-Based Access Control (Visibility)

This field controls the access of the entire Structured Object, with each attribute having the same visibility. The access control is designed this way because the whole object is retrieved and there is no way to guarantee that the redaction of attributes lowers the overall visibility of the object. The suggested solution is to create multiple versions of the object that have different visibility.

12.2.4 Ancestor Paths

Structured Objects have a hierarchical organization where each object may have a single parent. The parent and child relationship does not mean that the child was derived from the parent, only that the Structured Object that is the parent contains the child. Ancestry makes it possible to implement efficient transactions for objects that reside in the same group.

12.3 TABLE STRUCTURES

12.3.1 The so_manager Table

The so_manager table manages the datastore instances as well as the index definitions. This table contains global configuration information that has to be synchronized between all clients. Examples of this information are table names, partitioning of indexes, and table schema version. This table is not displayed when using the soadmin tool.

12.3.2 Instance Management Table

The datastore instances are stored in the so_manager table. When an instance is created additional tables are created and configured and global schema data and property data are loaded.

Stores information about instances including configuration values that must be shared amongst all the clients of a datastore instance. All information about the instance is stored in a single Accumulo row. Instance specific configuration information is stored in the "config" column family and composite index definitions are stored within the "index" column family. The index column family stores metadata that is necessary for managing composite index definitions.

Row	Column Family	Column Qualifier	Value
<instance_name>	"config"	"instance.version"	<version> (e.g. 1.0.0)
<instance_name>	"config"	"instance.table.name.core"	<table name> (e.g. so_tcri_core)
<instance_name>	"config"	"instance.table.name.kind"	<table name> (e.g. so_tcri_kind)
<instance_name>	"config"	"instance.table.name.ascending"	<table name> (e.g. so_tcri_ascending)
<instance_name>	"config"	"instance.table.name.descending"	<table name> (e.g. so_tcri_descending)
<instance_name>	"config"	"instance.table.name.composite"	<table name> (e.g. so_tcri_composite)
<instance_name>	"config"	"instance.index.partitions"	<num_partitions>
<instance_name>	"index"	"__seq__"	<index_id> (e.g. 1, 2, 3, ...)
<instance_name>	"index"	"__hash__"<definition_hash>	<namespace><kind><index_id>
<instance_name>	"index"	<namespace><kind><index_id>	<definition_json>
<instance_name>	"persistence"	<namespace><kind><property_key>	<property_value>

Figure 85 Instance Management Table

12.3.3 Core Table

This table is configured with two non-default locality groups in order to isolate journal entries and entries that indicated whether or not an objects exists. Bloom filters and block caching is also enabled for the core table.

Row	Column Family	Column Qualifier	Value
`<key>`	*empty*	`<property>`	*empty*

Figure 86 Core Table

12.3.3.1 Key Encoding

The rowID is the key and it consists of a namespace, an optional ancestor path, kind, and id. The components of the id are separated by the Unicode information-separator-one character (U+001F)

12.3.3.2 Property Encoding

The property key is compactly encoded into a byte array based on the property type. It contains a property label, a separator, a property type, a byte that indicates if the property value is null, and a property value that is encoded to specific types. Below is the property key

```
<property_label><separator><property_type><is_null><property_value>
```

Figure 87 Property Encoding

Property Component	Description
`<property_label>`	A utf-8 encoded string (may not contain U+001F)
`<separator>`	A single byte containing unicode information-separator-one (U+001F)
`<property_type>`	A byte that represents the primitive type of the property's value (string, integer, ...)
`<is_null>`	A byte that identifies whether or not the property's value is null
`<property_value>`	An encoded representation of the property value specific to each type

Figure 88 Property Component Description

The property type integer mapping

Integer	Property Type
0	String
1	Opaque byte array
2	32-bit Signed Integer
3	64-bit Signed Integer
4	64-bit Floating Point
5	Boolean
6	Date
7	Date Interval
8	well-known-text
9	IPv4 address
10	IPv6 address

Figure 89 Property Type Mapping

12.3.4 Indexing

Indexing is configurable at the granularity of a property. Ascending, descending reverse indexes are opt-out, while composite indexing is opt-in. According to the documentation as of January 2016 the feature has not been implemented, but this is unclear.

12.3.4.1 Kind Index

The kind index allows queries that retrieve all objects of a certain kind, regardless of where they exist within the hierarchy of objects. The table will have the name "instanceName_kind".

Row	Column Family	Column Qualifier	Value
`<namespace><kind><key>`	*empty*	*empty*	

Figure 90 Kind Index

12.3.4.2 Ascending Reverse Index

Build to allow queries based on Boolean expressions involving equality filters, inequality filters on a single property, and ascending ordered retrieval based on a single property. This table will have the name "instanceName_ascending"

Row	Column Family	Column Qualifier	Value
<namespace><key_shard><kind><property_name><property_type><asc_property_value>	*empty*	<ancestor_path><id>	*empty*

Figure 91 Ascending Reverse Index

12.3.4.3 Descending Reverse Index

Supports the same types of queries that the ascending index supports but able to satisfy a single property descending order rather than ascending. This table will have the name "instanceName_descending"

Row	Column Family	Column Qualifier	Value
<namespace><key_shard><kind><property_name><property_type><desc_property_value>	*empty*	<ancestor_path><id>	*empty*

Figure 92 Descending Reverse Index

12.3.4.4 Composite Index

The composite index table consists of rows that contain concatenated values based on predefined index definitions. Each definition contains all of the information about the property keys involved in each index.

Row	Column Family	Column Qualifier	Value
<namespace><key_shard><index_id><segment_1><segment_2>...	*empty*	<ancestor_path><id>	*empty*

Figure 93 Composite Index

This table will have the name "instanceName_composite"

12.3.5 Index Definitions

Index definitions are stored in the so_manager table and are associated with a particular kind of structured object in a namespace. An identifier for the index definitions is determined by sequentially incrementing the value of a column in Accumulo. Duplicate index definitions are detected by computing a content-based hash from the namespace, kind, and index segments. The definitions are cached within datastore clients to avoid a lookup for each write, with the cache being refreshed each minute.

13. QUERYING STRUCTURED OBJECT WITH SOQL

13.1 OVERVIEW

The SOQL (pronounced 'SO-COOL') defines a SQL-inspired language for performing queries on a Structured Object datastore. From a developer perspective, a SOQL query is a string that can be parsed into a query object which can in turn be used to scan for matching structured objects.

```
String soql = "SELECT * FROM 'BaseballPlayer' IN 'namespace' WHERE height_inches >
68"
SOQL.Parser parser = new SOQL.Parser().setDefaultNamespace(NAMESPACE);
Query query = parser.parse(soql);
```

The SOQL language supports a restricted subset of SQL features and adds some unique features of its own in order to expose special capabilities of the Structured Object datastore.

Every SOQL query starts with the `SELECT` keyword which is followed by an argument that we call the *scope* section. The *scope* section defines what the query intends to retrieve from the datastore. In most cases, the value for this argument will be a `*` indicating that the entire set of properties for each object matching the query constraints should be returned.

The client can choose to set a default namespace that will be used if the SOQL query does not explicitly specify a namespace. Namespaces can be specified by using an absolute URI for the `kind`.

```
SELECT * FROM 'BaseballPlayer' IN 'namespace' WHERE label = 'value'
```

13.2 SOQL INTRODUCTION

The SOQL language is currently very simple and only borrows the `SELECT` capabilities from the SQL. A `SELECT` query is composed of several sections.

1. A Scope Section (required)
2. A Kind Section (required)
3. A Constraint Section (optional)

The scope section is similar to the SQL projection but expands the concept to utilize the hierarchical relationships between Structured Objects through their keys. The scope section can be satisfied by a comma separated list containing a wildcard `*`, signifying all fields of the SELF scope, as well as additional scopes which signify hierarchically related objects.

The kind section specifies a single `kind` of structured object that the query constraint is applied to.

The constraint section defines a Boolean expression tree of constraints that must be met for each return object. There are some limitations on the types of constraints that can be evaluated with the default ascending and descending indexes.

13.3 SOQL SECTION TO SQL SECTION MAPPING

The intent of this table is to point out the similarities between SQL and SOQL.

Table 10 SOQL Section Mapping

SOQL Query Section	SQL Query Section	SQL Examples	SOQL Examples
Scope	Projection	`SELECT * ...`	`SELECT * ...`, `SELECT PARENTS ...`
Kind	FROM table	`SELECT * FROM People ...`,	`SELECT * FROM 'People'...`
Constraints	WHERE constraints	`SELECT * FROM People WHERE l_name = 'Doe'`	`SELECT * FROM 'BaseballPlayer' WHERE l_name = 'Doe'`

Literal Types

The SOQL Lexer identifies a set of valid literal types by deterministic rules. For numeric types and Boolean, the Lexer abides by Java Specific rules. For the string type SOQL utilizes a single quote for beginning and termination. The remaining types are currently in flux but will be solidified soon.

The SOQL defines the following literal types:

Table 11 SOQL Literals

Type	Notes	Example Value
string	single quote	'test string'
integer		9
long		1L
boolean		true / false
double		1.2
datetime		ISO 8601 date format**

Since the SOQL Lexer only identifies a subset of primitive types, there are limitations of what queries can be specified with SOQL compared to those that can be constructed using the Java API.

Datetime formats conform to ISO 8601 standards. The supported formats are `yyyy-MM-dd`, `yyyy-MM`, `yyyy-MM-dd'T'HH:mm:ssZ`, `yyyy-MM-dd'T'HH:mm:ss+00:00`, `yyyyMMdd'T'HHmmss'Z'`. The timezone for all dates is UTC.

13.4 CONSTRAINTS

13.4.1 WHERE clause

The `WHERE` clause supports complex Boolean expressions through the use of parenthesis, `AND` and `OR` operators. The operator precedence followed by the SOQL parser gives priority to parenthesis enclosed expressions, while giving AND and OR equal precedence from left to right. Queries containing 'AND' and 'OR' operators requires composite indices.

13.4.2 Namespace Constraints (IN clause)

Providing an argument for 'IN' clause specifies a namespace for a query and replaces the SOQL parser's default namespace. Consider the following example:

```
SELECT * FROM 'Person' IN 'tcri'
```

The SOQL parser interprets this query as retrieving StructuredObjects with namespace 'tcri' and kind 'Person'. This SOQL query is equivalent to:

```
Query query = new Query("tcri", "Person");
```

13.4.3 Key Constraints (WITH clause)

'WITH' clause enables users to search for StructuredObjects within a range of StructuredObjectKeys. These key queries should only contain the 'WITH' clause. Like the constraints using 'WHERE' clause, key constraints support complex boolean expressions through the use of logical operators ('AND' or 'OR'). Users can query for an object with a certain key:

```
SELECT SELF WITH KEY = so:///tcri/Person;Joe
```

Users can query for objects within a range of SturcturedObjectKeys.

```
SELECT * WITH KEY < so:///tcri/Person;Joe AND KEY > so:///tcri/Person;Bill
SELECT * WITH ( KEY >= so:///ns/Person;Bill AND KEY <= so:///ns/Person;Joe) OR (
KEY >= so:///ns/Person;Jill AND KEY <= so:///ns/Person;Mary)
```

13.4.3.1 Ancestor Key Constraints (WITH ANCESTOR clause)

'WITH' clause also allows querying for StructuredObjects with an ancestor with a specified key. Queries with ancestor constraint requires a 'FROM' clause to specify the kind of the results. An example query is the following:

```
SELECT * FROM 'Mention' WITH ANCESTOR = 'so:///tcri/Person;Joe'
```

Example 1 - Homogeneous Boolean Constraints

```
SELECT * FROM ‘BaseballPlayer’ WHERE position = 'FB' AND bats = "L" AND throws "L"
```

```
                    AND
                   /   \
                 /       \
               /           \
     position = 'FB'      AND
                         /  \
                       /      \
                     /          \
               bats = 'L'    throws = 'L'
```

Example 2 - Homogeneous Boolean Constraints

Since the OR clause is enclosed in parenthesis, this operator will be given priority over the AND operator.

```
SELECT * FROM 'BaseballPlayer' WHERE team = 'Orioles' AND (position = 'LF' OR
position = 'RF')
```

```
                    AND
                   /   \
                 /       \
               /           \
     team = 'Orioles'      OR
                         /  \
                       /      \
                     /          \
         position = 'LF'       position = 'RF'
```

13.5 QUERY SCOPE

The query scope is specified in the `SELECT` clause and is a unique feature for StructuredObject which can be based on *ancestor paths*. Given a stream of results determined by the `FROM` and `WHERE` clauses, the query scope allows for retrieving structured objects that are ancestrally related to those in the original stream.

Table 12 Query Scope

Scope	Description
*	Includes all fields on object
`<field name>`	Includes the given field in the returned object
ROOT	Emits only the roots of the objects matched by the query
ANCESTORS	Emits all ancestors of the objects matched by the query
PARENTS	Emits all direct ancestor (parents) of the objects matched by the query
SELF	Emits the objects matched by the query
CHILDREN	Emits all direct descendants (children) of the objects matched by the query
DESCENDANTS	Emits all descendants of the objects matched by the query

ROOT Scope

The root scope indicates that only the roots of the ancestor paths present in the stream of results determined by the `FROM` and `WHERE` clauses should be returned. Note that there may be multiple kinds of structured objects returned for this type of query since the kind of structured objects returned is unrelated to the kind specified in the `FROM` clause

Example

Considering the following structured objects (using the URI format of structured object keys).

```
so:///tcri/Root;001/Child;001 a = 1
so:///tcri/Root;001/Child;002 a = 1
so:///tcri/Root;001/Child;003 a = 2
so:///tcri/Root;001/Child;004 a = 2
so:///tcri/Root;001/Child;005 a = 1
so:///tcri/Root;001/Child;006 a = 2
so:///tcri/Root;001/Child;007 a = 2
```

```
The process by which this occurs is as follows (Not yet implemented).

1. The query execution logic searches for all structured object keys that
   meet the constraint a = 1.

2. As these keys are retrieved, the root of the ancestor path is determined.

3. If the root is different than the previous root, its existence is
   verified.

4. If the root exists, it is returned as the next result.
```

The query `SELECT ROOT FROM 'Child' WHERE a = 1` should return the structured object with the following keys (assuming they exist). Note that objects can be referenced in ancestor paths without existing on their own.

```
so:///tcri/Root;001
so:///tcri/Root;002
```

```
The process by which this occurs is as follows (Not yet implemented).
1. The query execution logic searches for all structured object keys that
   meet the constraint a = 1.
2. As these keys are retrieved, the root of the ancestor path is determined.
3. If the root is different than the previous root, its existence is
   verified.
4. If the root exists, it is returned as the next result.
```

13.5.1 ANCESTORS Scope

The ancestors scope indicates that objects contained in the ancestor paths present in the stream of results determined by the `FROM` and `WHERE` clauses should be returned.

Example

Considering the following structured objects.

```
so:///tcri/Root;001/Parent;001/Child;001 a = 1
so:///tcri/Root;001/Parent;001/Child;002 a = 1
so:///tcri/Root;001/Parent;002/Child;003 a = 2
so:///tcri/Root;002/Parent;002/Child;004 a = 2
so:///tcri/Root;002/Parent;002/Child;005 a = 1
so:///tcri/Root;002/Parent;003/Child;006 a = 2
so:///tcri/Root;003/Parent;003/Child;007 a = 2
```

```
The query SELECT ANCESTORS FROM 'Child' WHERE a = 1 should return the
structured object with the following keys (assuming they exist). Note that
objects can be referenced in ancestor paths without existing on their own.
```

```
so:///tcri/Root;001
so:///tcri/Root;001/Parent;001
so:///tcri/Root;002
so:///tcri/Root;002/Parent;002
```

13.5.2 PARENTS Scope

The parents scope indicates that objects which are direct ancestors of the objects present in the stream of results determined by the `FROM` and `WHERE` clauses should be returned.

Example

Considering the following structured objects.

```
so:///tcri/Root;001/Parent;001/Child;001 a = 1
so:///tcri/Root;001/Parent;001/Child;002 a = 1
so:///tcri/Root;001/Parent;002/Child;003 a = 2
so:///tcri/Root;002/Parent;002/Child;004 a = 2
so:///tcri/Root;002/Parent;002/Child;005 a = 1
so:///tcri/Root;002/Parent;003/Child;006 a = 2
so:///tcri/Root;003/Parent;003/Child;007 a = 2
```

```
The query SELECT PARENTS FROM 'Child' WHERE a = 1 should return the
structured object with the following keys (assuming they exist). Note that
objects can be referenced in ancestor paths without existing on their own.
so:///tcri/Root;001/Parent;001
so:///tcri/Root;002/Parent;002
```

13.5.3 SELF Scope

The self scope is equivalent to `*` and indicates that the objects present in the stream of results determined by the `FROM` and `WHERE` clauses should be returned.

13.5.4 CHILDREN Scope

The children scope indicates that objects which are direct descendants of the objects present in the stream of results determined by the `FROM` and `WHERE` clauses should be returned.

13.5.5 DESCENDANTS Scope

The descendants scope indicates that all objects descending from the objects present in the stream of results determined from the `FROM` and `WHERE` clauses should be returned.

13.6 USING SOQL

13.6.1 Web UI

SOQL Web UI can be accessed at the URL: https://<tcri_so_web_server>:8443/sol-ws/query.html. Enter the following query then click submit to retrieve flight data objects:

```
SELECT * FROM 'so:///tcri/demo_flight_geo'
```

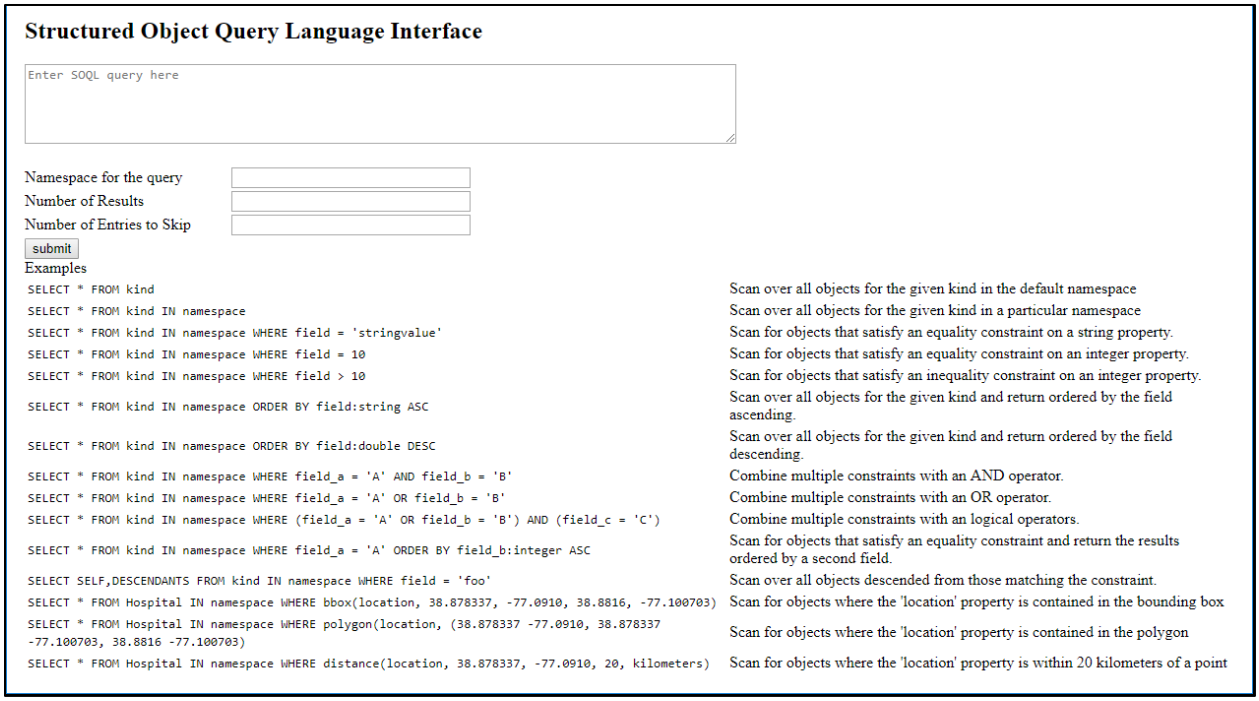


Figure 94 Structured Object Query Language Interface

13.6.2 Command Line Interface

SSH into <tcri_so_web_server> and execute the following command to query for structured objects in flight data example:

```
sudo /opt/sol/cli/bin/soclient query -n tcri -q "SELECT * FROM
'so:///tcri/demo_flight_geo'"
```

Observe the returned query result:

```
21:30:46.742INFO  m.t.s.c.c.QueryCommand - 0results counted in 0.02seconds
21:30:46.750INFO  m.t.s.c.c.QueryCommand - 0.0results per second.
```

Query result of flight data example may look like the following:

```
22:37:45.072 INFO  m.t.s.c.c.QueryCommand - [tcri][demo_flight_geo]:[A047FE]
[Property{key=PropertyKey{label=altitude, type=DbFloat64}, value=1475.0},
Property{key=PropertyKey{label=artifact_key, type=DbString},
value=urn:drif:artifact:7c596965b87253549f3b39abd5420378de33d5f8727f02c7bd65a2ba78
db2a8e}, Property{key=PropertyKey{label=icao, type=DbString}, value=A047FE},
Property{key=PropertyKey{label=latitude, type=DbFloat64}, value=33.2863},
Property{key=PropertyKey{label=longitude, type=DbFloat64}, value=-111.6366},
Property{key=PropertyKey{label=op, type=DbString}, value=U-HAUL CO OF OREGON     -
PORTLAND, OR}, Property{key=PropertyKey{label=trak, type=DbFloat64}, value=133.9}]
22:37:45.072 INFO  m.t.s.c.c.QueryCommand - [tcri][demo_flight_geo]:[A051C2]
[Property{key=PropertyKey{label=altitude, type=DbFloat64}, value=4925.0},
Property{key=PropertyKey{label=artifact_key, type=DbString},
```

```
value=urn:drif:artifact:792c4058365c567290ab5b22553f2d8b13681c8d9bbea238d131e6217f
bfca3a}, Property{key=PropertyKey{label=icao, type=DbString}, value=A051C2},
Property{key=PropertyKey{label=latitude, type=DbFloat64}, value=33.2384},
Property{key=PropertyKey{label=longitude, type=DbFloat64}, value=-112.1826},
Property{key=PropertyKey{label=op, type=DbString}, value=BANSHOHO HOLDINGS LLC
- BRIDGEHAMPTON, NY}, Property{key=PropertyKey{label=trak, type=DbFloat64},
value=52.5}]
22:37:45.072 INFO  m.t.s.c.c.QueryCommand - [tcri][demo_flight_geo]:[A05559]
[Property{key=PropertyKey{label=altitude, type=DbFloat64}, value=25825.0},
Property{key=PropertyKey{label=artifact_key, type=DbString},
value=urn:drif:artifact:e45c26c1f8ec94c9cb7c8819371edf7be89e9379e615ef13a8a9504d33
604472}, Property{key=PropertyKey{label=icao, type=DbString}, value=A05559},
Property{key=PropertyKey{label=latitude, type=DbFloat64}, value=32.676692},
Property{key=PropertyKey{label=longitude, type=DbFloat64}, value=-111.470536},
Property{key=PropertyKey{label=op, type=DbString}, value=SkyWest Airlines},
Property{key=PropertyKey{label=trak, type=DbFloat64}, value=331.0}]
22:37:45.072 INFO  m.t.s.c.c.QueryCommand - 35 results counted in 0.455 seconds
22:37:45.072 INFO  m.t.s.c.c.QueryCommand - 76.92307692307692 results per second.
```

Other options for **soclient** are described below:

```
usage: soclient [command] [command options]
    --admin-pass <arg>        username for the Accumulo instance
    --admin-user <arg>        username for the Accumulo instance
    --auths <arg>             user authorizations for scanning the datastore
    --config <arg>            An alternative way to provide Accumulo connection
                              information.
    --count-frequency <arg>   print the count every k objects
    --instance-name <arg>     name of the Accumulo instance
 -l,--limit <arg>             maximum number of entries to scan
 -n,--name <arg>              name of datastore instance
 -np,--no-pagination          do not limit the result output to a single page
    --print <arg>             whether or not to print the output
 -q,--query <arg>             the query to issue
    --skip <arg>              number of entries to skip
    --zookeepers <arg>        zookeepers for the Accumulo instance
```

Note the query result limit is default to 25 entries, and can be modified by the **-l** or **--limit** option.

14. ARCGIS GEOEVENT SERVER

In this section, we will learn how to display on a map the flight data that we ingested in the above example.

14.1 GEOEVENT DEFINITIONS

Event data in GeoEvent Server has an associated schema that identifies the attribute fields and data types (date, string, integer, and so forth) for the event's data. This schema is referred to as a GeoEvent Definition and can be viewed by browsing to GeoEvent Manager URL: https://<geoevent_node>:6143/geoevent/manager/site.html. Under GeoEvent Definitions, define a new **demo_flight_geo** with the attributes as seen in the following figure.

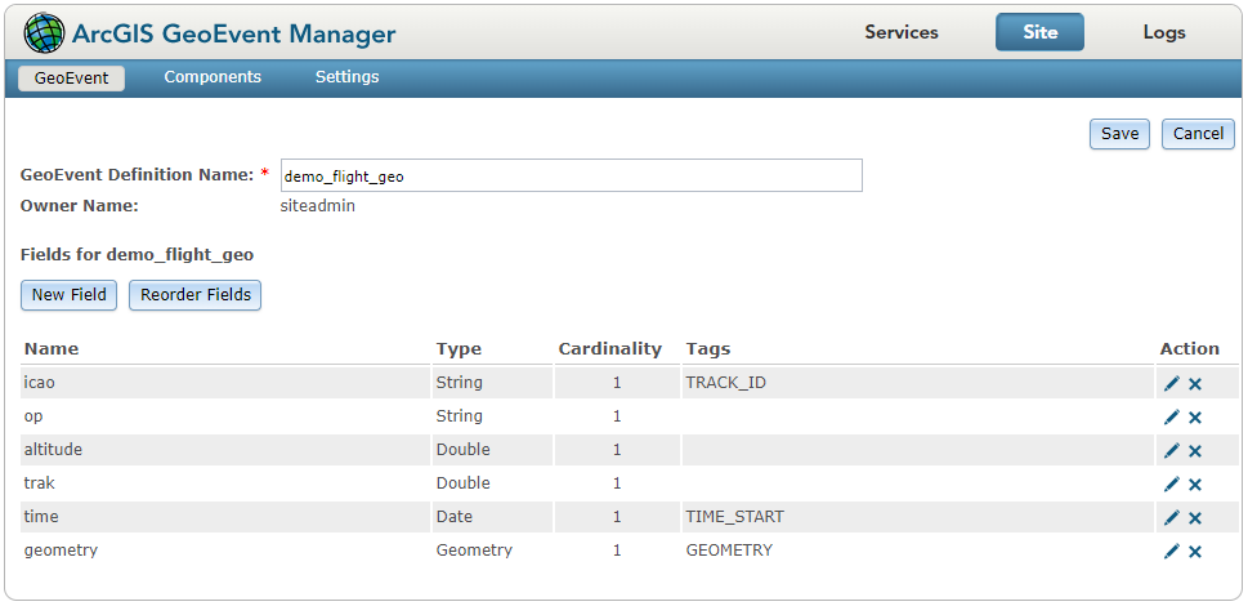


Figure 95 GeoEvent Definitions - demo_flight_geo

A GeoEvent Definition consistent with the data in an event stream must exist in order for an input connector to construct a GeoEvent from the events coming in. Similarly, a GeoEvent Definition consistent with the schema of a designated output must exist in order for an output connector to construct a GeoEvent and send the event data. Some input connectors are able to create event definitions based on data they discover in the events received. Administrators import event definitions to support operations such as updating published feature services. This means the list of known GeoEvent Definitions frequently changes.

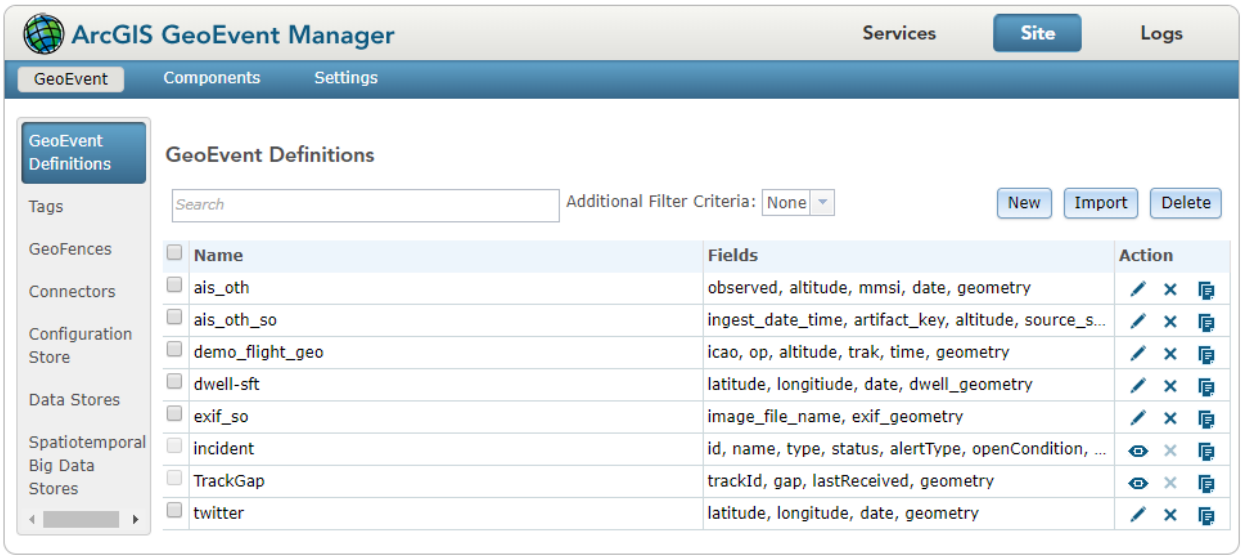


Figure 96 GeoEvent Definitions

14.2 GEOEVENT TAGS

A tag is a label placed on a particular field of a GeoEvent Definition. Tags are similar to field aliases; they support consistency in expressions when different event definitions associate similar data with different field names. Certain built-in tags identify GeoEvent fields containing critical information such as TRACK_ID or GEOMETRY.

Individual fields in a GeoEvent Definition can be tagged with a GeoEvent tag. The same tag can be applied to different fields in different GeoEvent Definitions to indicate that those fields contain similar or compatible data. However, within a given GeoEvent Definition, a tag can be applied to only one field.

Administrators can view available tags by browsing to **Site** > **GeoEvent** > **Tags** in GeoEvent Manager.

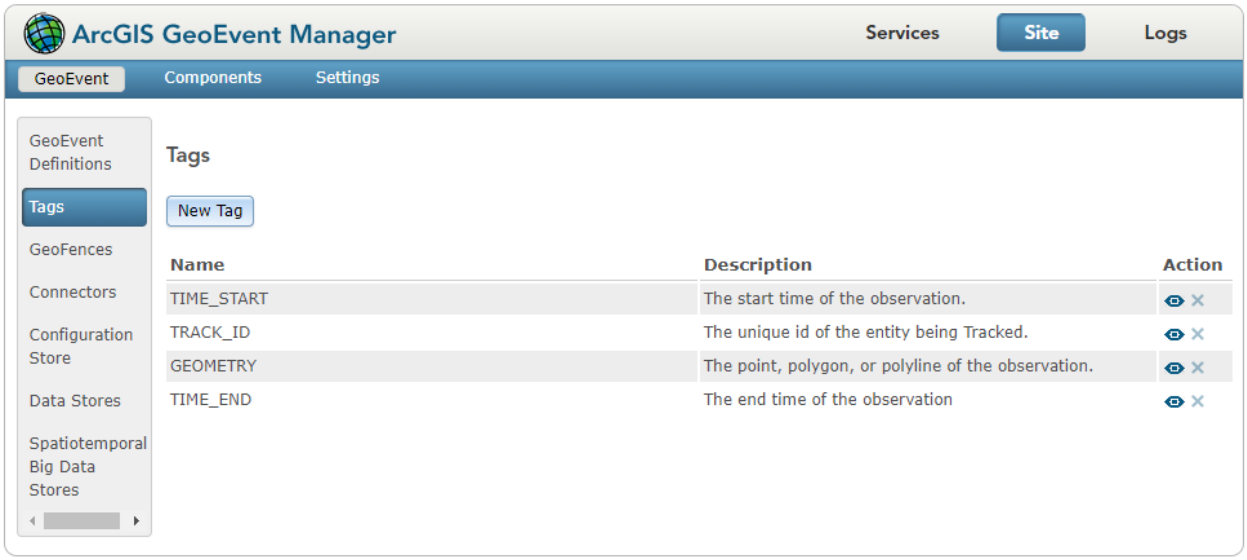


Figure 97 GeoEvent Tags

14.3 GEOEVENT INPUTS

An input connector is the component of a GeoEvent Service responsible for receiving and interpreting events from a data stream. The input connector retrieves attribute values from each event and constructs a GeoEvent, which can be routed through optional filters and processors to an output connector. Each input connector is configured to receive a data stream from a single specific data source. Every GeoEvent Service must include at least one input connector, but a GeoEvent Service may include more than one input connector.

Input connectors are created from connectors listed on the **Site > GeoEvent > Connectors** page in GeoEvent Manager. By clicking the **Add Input** button on the **Services > Inputs** page, a list of connectors available for receiving streaming data is displayed. We will define **kafka-kryo-in** input as in the following figure. This input uses **Receive Features via Kafka** input connectors. For full list of GeoEvent input connectors, refer to http://enterprise.arcgis.com/en/geoevent/latest/process-event-data/input-connectors.htm.

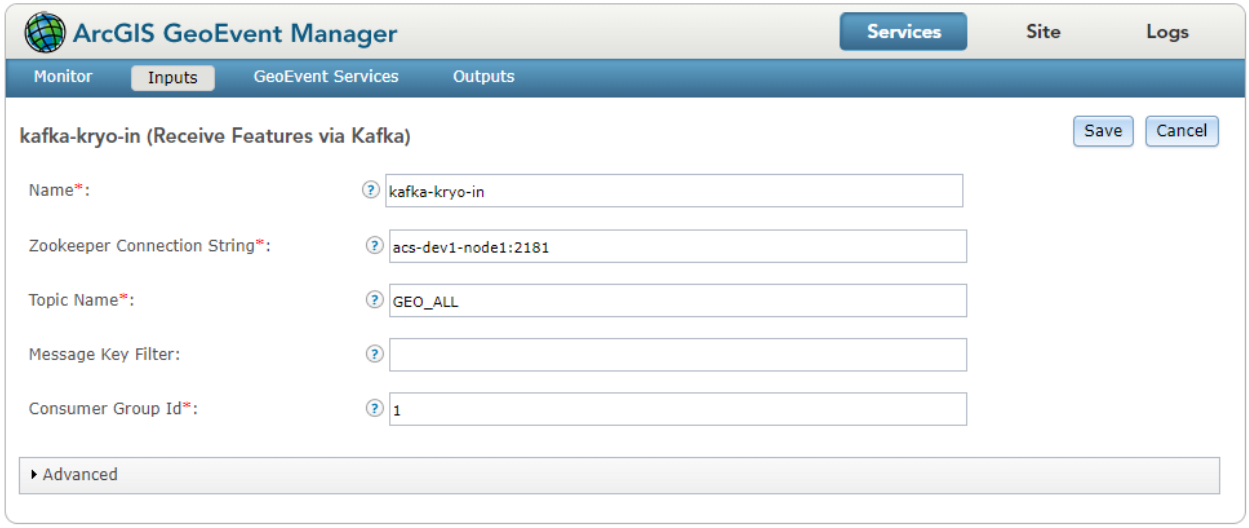


Figure 98 GeoEvent Inputs

14.4 GEOEVENT OUTPUTS

The GeoEvent Server is capable of sending event data it has received to a variety of different destinations. An output connector must know about the communication channel over which it will be sending the data and be able to translate the GeoEvents to an expected format. These tasks are handled by adapters and transports, technical implementations beneath each and every connector.

We will define two outputs, one to send flight data to spatiotemporal big data store and one to a stream service.

A stream service is a type of ArcGIS Server service that emphasizes low latency, real-time data dissemination, for client-server data flows. Clients connecting to a stream service begin receiving data immediately upon subscribing to the service. Clients can specify and reconfigure both spatial and attribute constraints without first unsubscribing and then reconnecting to the service.

Stream services leverage WebSocket technology, which supports full-duplex, bidirectional communication. This enables clients to specify the data they want to receive without having to unsubscribe and reestablish their connection to the service. Clients are able to filter stream service data by specifying either spatial or attribute constraints.

Connecting to a stream service to receive a real-time data feed allows you to separate the basic need of immediate event visualization from the need to persist data in a database. By configuring the GeoEvent Server output to broadcast event data via a stream service, you have the option of storing the latest event data in an enterprise relational data store or spatiotemporal big data store.

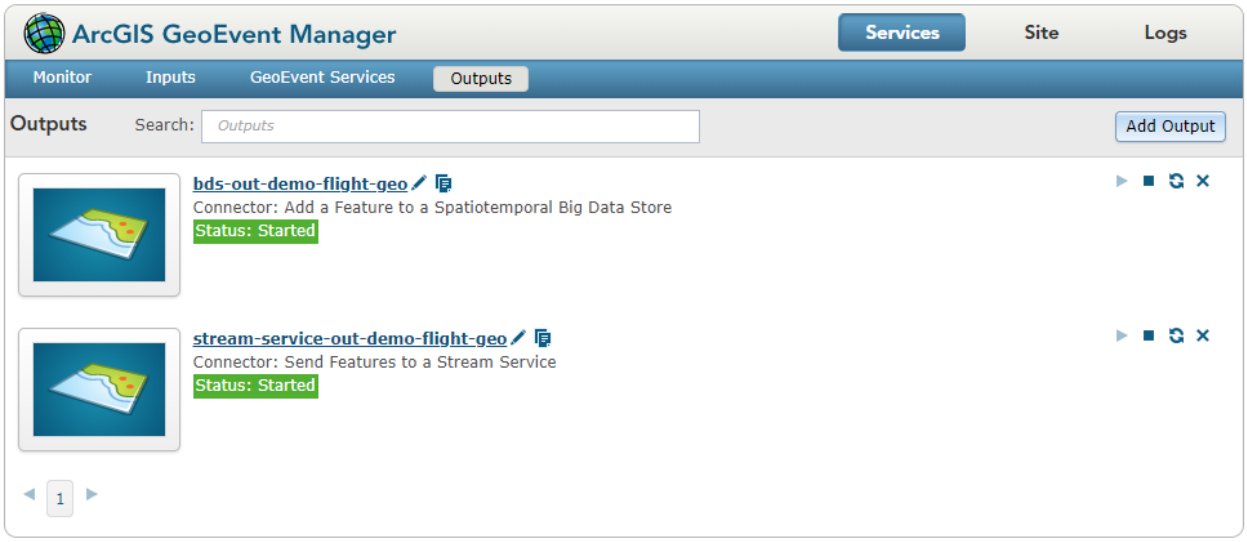


Figure 99 GeoEvent Outputs

Detailed descriptions for **bds-out-demo-flight-geo**:

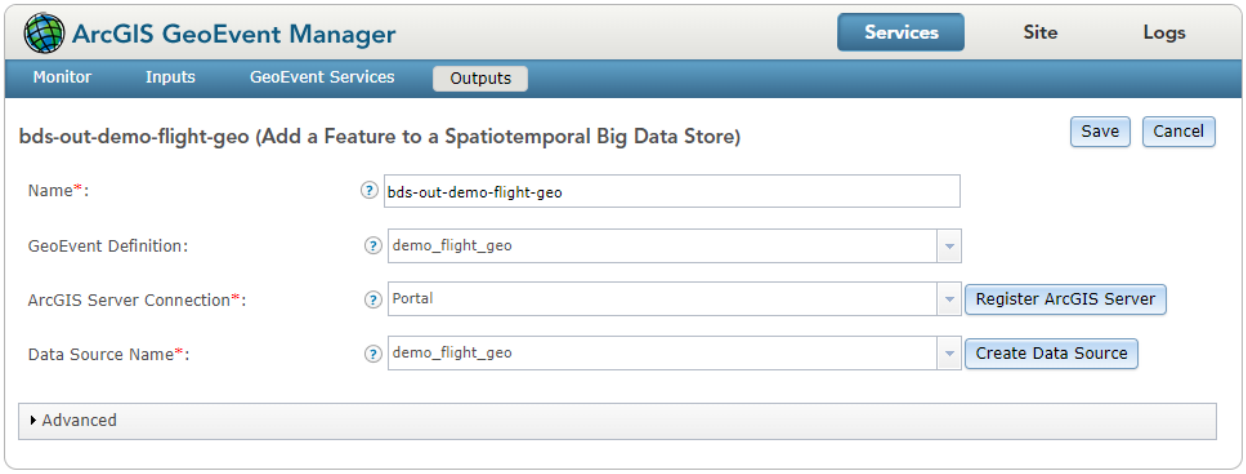


Figure 100 GeoEvent Outputs - bds-out-demo-flight-geo

where **demo_flight_geo** data source is created with the following options:

- Data Retention Option: 1 Hour
- Display Field Name: icao
- Time Info > Time Interval: 1
- Time Info > time Interval Units: Milliseconds
- Time Info > Has Live Data: checked

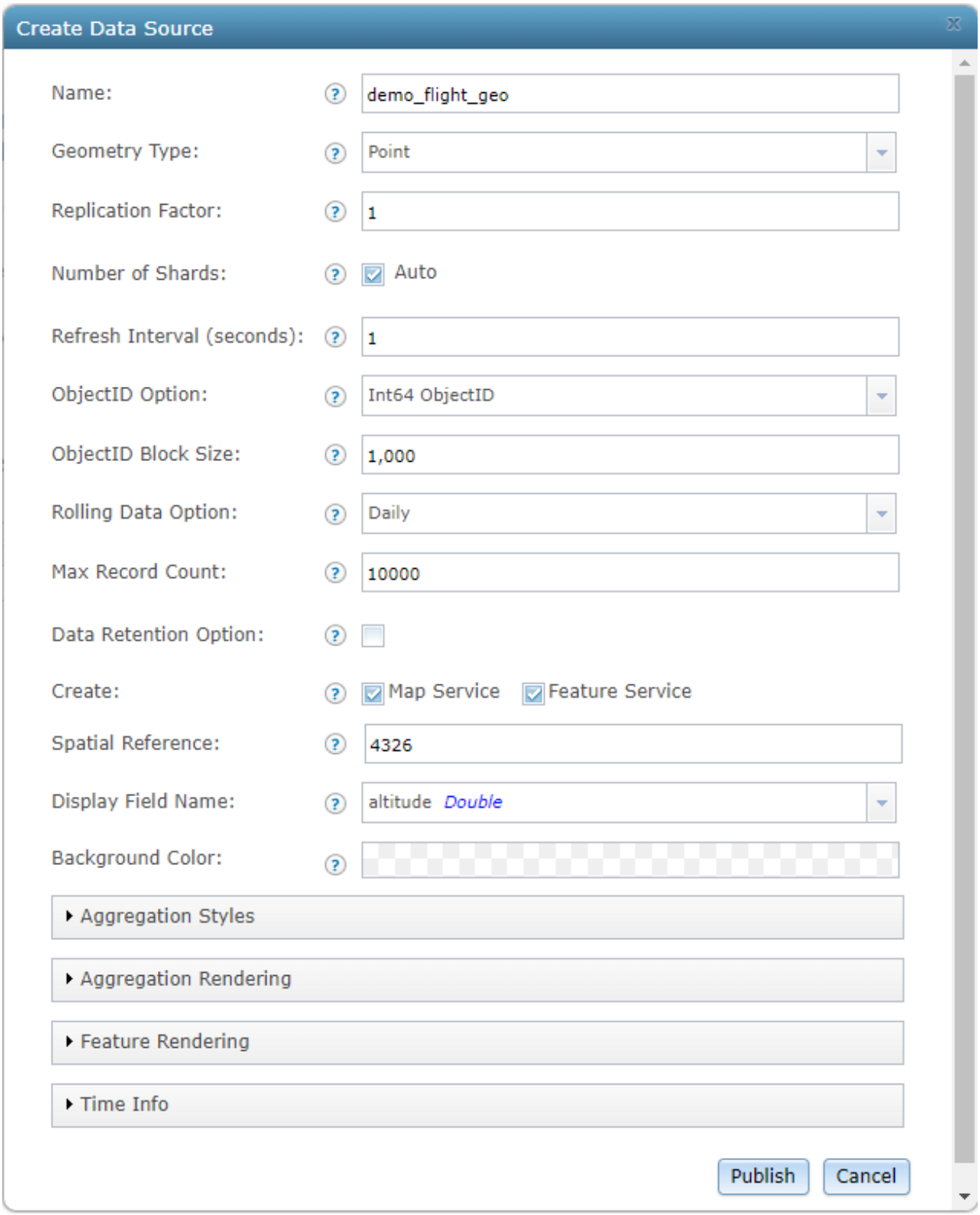


Figure 101 Data Source - demo_flight_geo

Detailed descriptions for **stream-service-out-demo-flight-geo**:

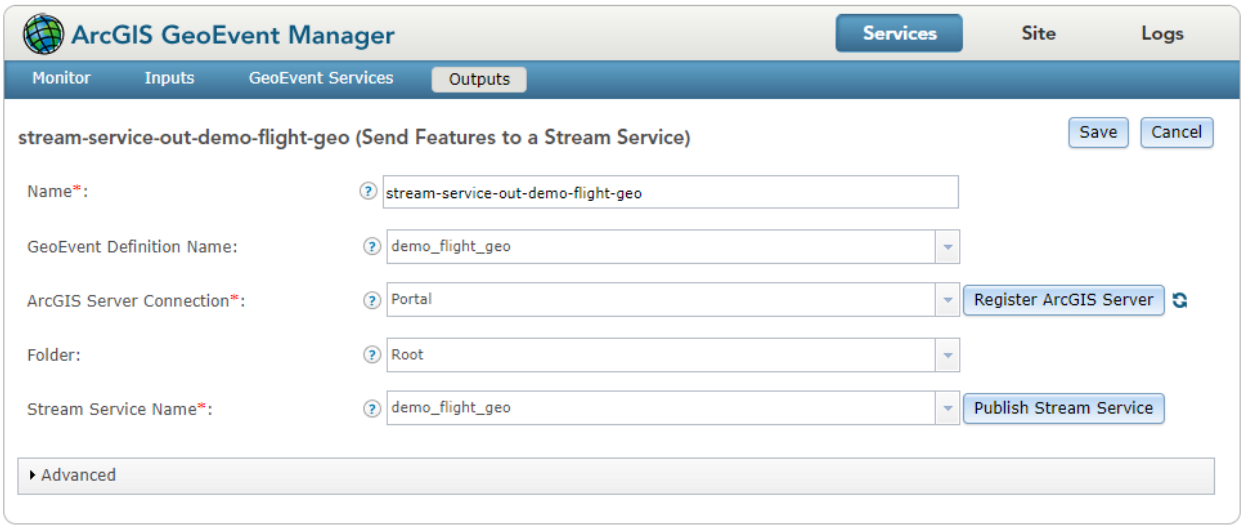


Figure 102 GeoEvent Outputs - stream-service-out-demo-flight-geo

where the stream service is published with the following options:

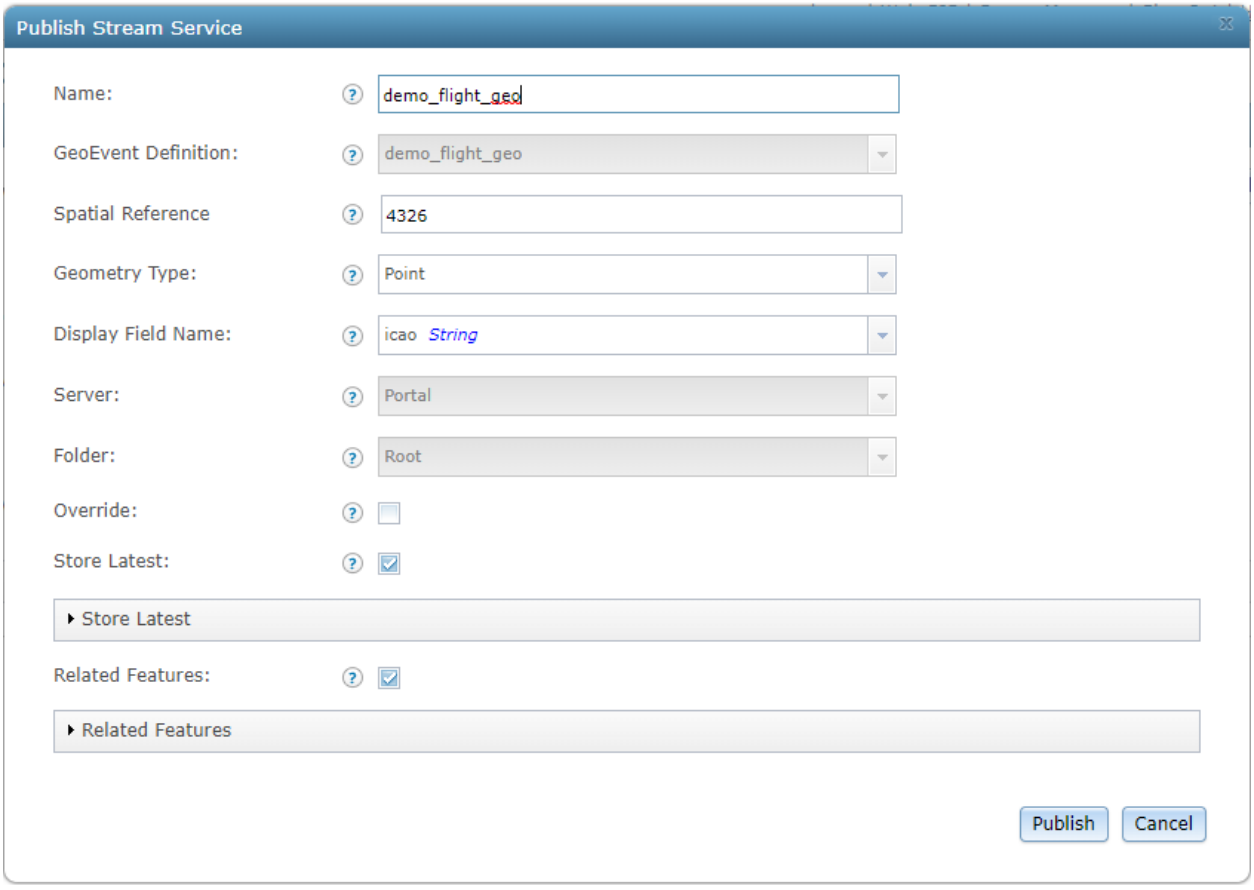


Figure 103 Stream Service Options

and Store Latest and Related Features are configured as follows:

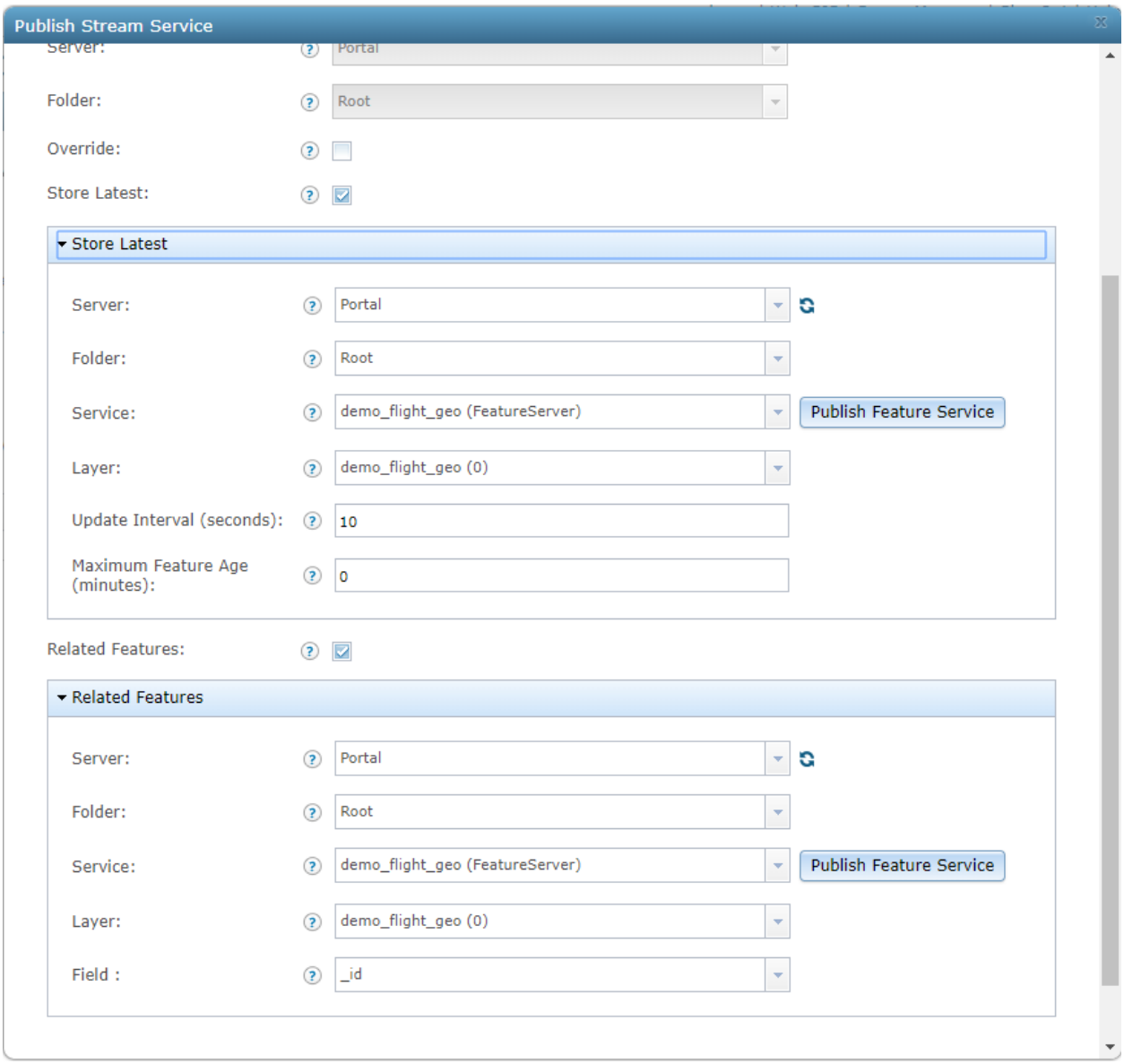


Figure 104 Stream Service Options Expanded

Further details on GeoEvent output connectors can be found at http://enterprise.arcgis.com/en/geoevent/latest/process-event-data/output-connectors.htm.

14.5 GEOEVENT SERVICES

GeoEvent Services allow you to define and apply the real-time filtering and processing of GeoEvents as they flow from input to output in GeoEvent Server. A service designer defines the flow of event data by adding filter and processor elements to the canvas and then connecting these elements to inputs and outputs to create a GeoEvent Service. The final step is to publish the GeoEvent Service; this starts the service and the real-time processing of the event data.

Once the necessary input connectors and output connectors are created as in the previous sections, you can drag and drop elements onto the canvas and connect them to create a GeoEvent Service. Step-by-step instructions illustrating how to create a GeoEvent Service can be found in the Introduction to GeoEvent Server tutorial. Access the tutorial from GeoEvent Server tutorials. For this example, we define **kafka-kryo-in-bds-out** services as follows:

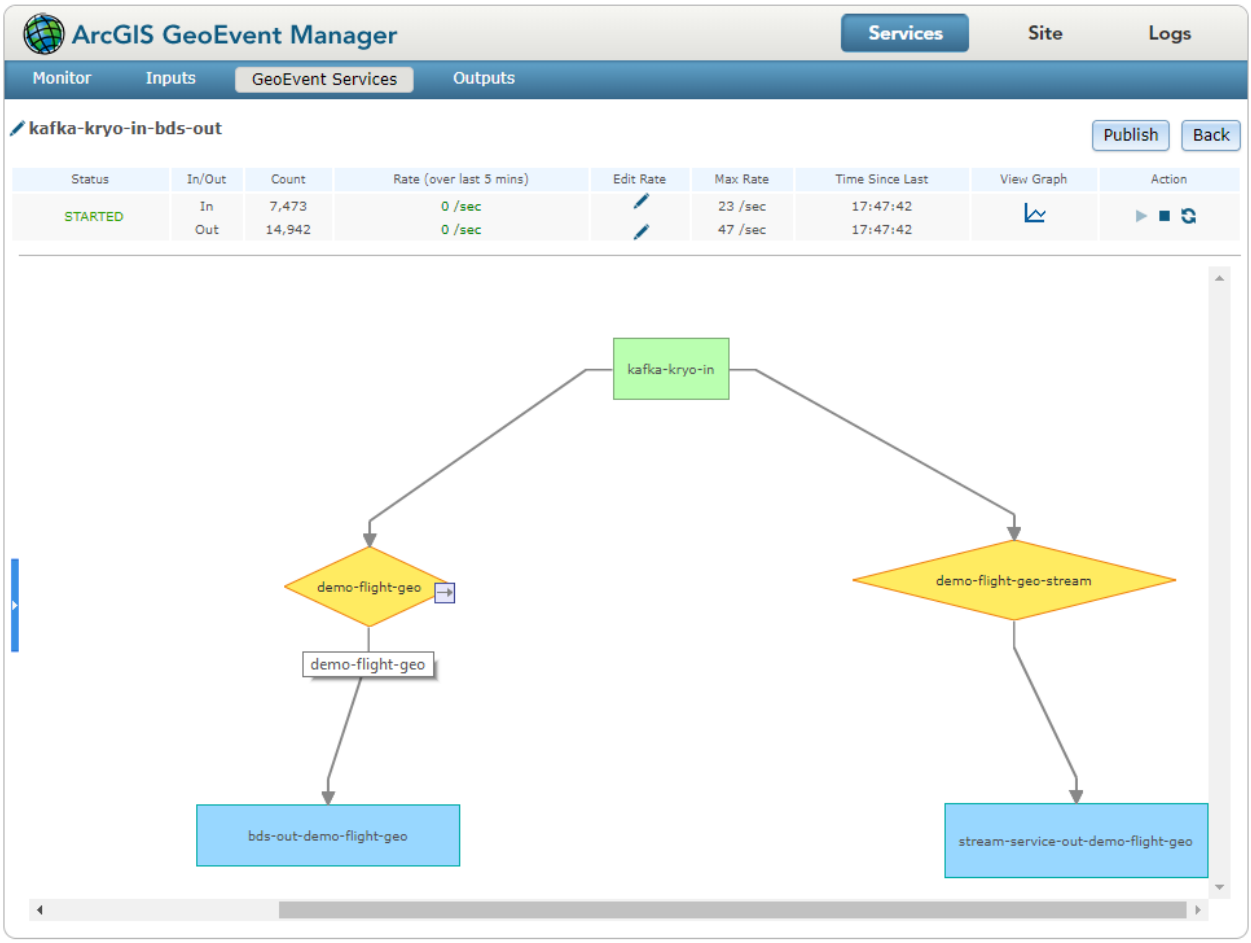


Figure 105 GeoEvent Services

14.5.1 GeoEvent Filters

Filters are configurable components of GeoEvent Services that filter (remove from the streaming data) GeoEvents that do not satisfy specified criteria. Filters are generally attribute filters, spatial filters or a combination of both attribute and spatial. GeoEvent Server product macros such as $DEFINITION_NAME, for example, enable GeoEvent filtering by properties (such as the name of the GeoEvent Definition associated with the current GeoEvent).

To add a filter to a GeoEvent Service, drag and drop a filter element from the element tray onto the canvas or double-click the filter element to add it to the GeoEvent Service—in either case, a properties dialog box will open, allowing you to configure the filter.

14.5.1.1 Attribute Filters

Attribute filters in a GeoEvent Service filter events based on attribute criteria. An attribute filter can be configured with one or more attribute expressions. The logical combination of these expressions represents the criteria a GeoEvent must satisfy to pass through the filter. Some attribute operators include Equal (=), Not Equal (!=), Greater Than (>), Greater Or Equal (>=), Less Than (<), Less Or Equal (<=), IN, MATCHES, EXISTS, and ISNULL.

14.5.1.2 Spatial Filters

Spatial filters in a GeoEvent Service filter GeoEvents based on a spatial relationship with a geofence. The geometries that specify a geofence must be imported into GeoEvent Server from a published feature service before a spatial filter can be configured. Spatial operators include INSIDE, OUTSIDE, ENTER, EXIT with relative to a GeoFence.

14.5.1.3 GeoEvent Property Filters

Every GeoEvent in GeoEvent Server has an associated set of properties such as the GeoEvent Definition used to construct the GeoEvent. A filter can be configured to filter GeoEvents based on their properties such as their GeoEvent Definition.

When configuring a filter expression, the drop-down for the Field parameter displays a list of field names from available GeoEvent Definitions. At the top of this list are several GeoEvent Server properties including:

- $OWNER_ID
- $OWNER_URI
- $DEFINITION_NAME
- $DEFINITION_GUID
- $RECEIVED_TIME

In this example, we construct filter that discards any GeoEvents that do not contain the demo_flight_geo, i.e. GeoEvent Definition name: $DEFINITION_NAME = demo_flight_geo. We define two filters between the input and the two outputs as seen in the previous figure.

14.5.2 Publishing GeoEvent Service

Once a GeoEvent Service contains all of the required elements that provide the necessary filtering, processing, and event routing, the GeoEvent Service can be published.

A GeoEvent Service must be published to either create and start a new GeoEvent Service or stop, update, and restart an existing GeoEvent Service that has been modified. GeoEvent Services are published from within the service designer used to create the GeoEvent Service.

14.5.3 Monitoring GeoEvent Server

GeoEvent Server can be monitored by viewing the Services > Monitor page in GeoEvent Manager. The data rates for all inputs, outputs, and GeoEvent Services can be monitored in real time, allowing you to quickly evaluate the current status as well as identify potential issues. The counts will indicate whether the input is receiving and handling events from the source, whether

the GeoEvent Service is processing the events, and whether the output is sending events to the appropriate output. As flight data are ingested, expect the counts increase.

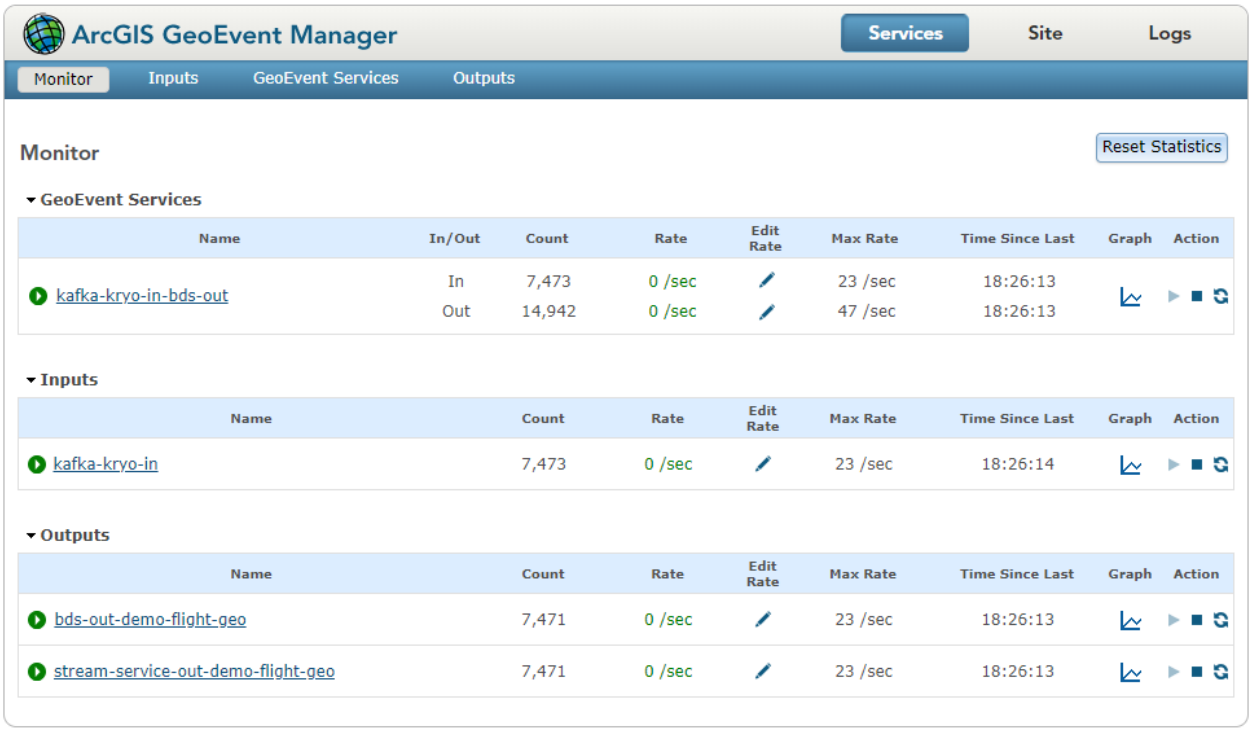


Figure 106 GeoEvent Monitor

14.5.3.1 Setting Rate Thresholds

Rate thresholds are an important feature for monitoring GeoEvent Server. Minimum and maximum rate thresholds can be set for the flow of GeoEvents through specific inputs, outputs, and GeoEvent Services. These thresholds are user defined and intended to reflect the expected behavior of each input, output, or GeoEvent Service under normal operating conditions. Setting appropriate minimum and maximum expected rates allows you to quickly see whether a particular component is behaving as expected. If data rates are within the expected bounds, the rate appears green. If data rates are outside the expected bounds, the rate appears red. This gives you immediate visual feedback when the system is behaving unexpectedly.

14.5.3.2 Max Rate and Time Since Last

The Monitor page includes statistics for the maximum observed rate and the time since the last GeoEvent was handled by each input, output, and service. These statistics help administrators identify unusual spikes or absences of streaming data.

14.5.3.3 Rate (over the last 5 minutes)

The Monitor page includes a rate statistic for the average number of events handled by each component over the last five minutes. This statistic is useful for data streams that have been running for a period of time, since the average rate for an input that has just started may be artificially low.

14.5.3.4 Average rate (over last 5 minutes)

By default, the statistics on the number of events received, sent, or handled by an input, output, or service are not reset and will continue to grow. The statistics can be reset, which will clear all metrics for inputs, outputs, and services (count, average rate, max rate, and time since last). To reset the statistics, click the Reset Statistics button at the top of the Monitor page.

14.5.4 Starting and stopping inputs, outputs, and GeoEvent Services

The current status for all inputs, outputs, and GeoEvent Services can be viewed on the **Services > Monitor** page in GeoEvent Manager. Starting, stopping, and restarting inputs, outputs, and GeoEvent Services.

From the Monitor page, individual components can be started, stopped, and restarted by clicking the **Start**, **Stop**, and **Restart** buttons for any of the listed inputs, outputs, or GeoEvent Services. Additionally, you can click the **Start All** and **Stop All** buttons on the Monitor page to start and stop all of the inputs, outputs, or GeoEvent Services available.

14.6 PORTAL FOR ARCGIS

Portal for ArcGIS is a component of ArcGIS Enterprise that allows you to share maps, scenes, apps, and other geographic information with other people. Portal can be accessed at https://<portal_node>/portal/home/content.html.

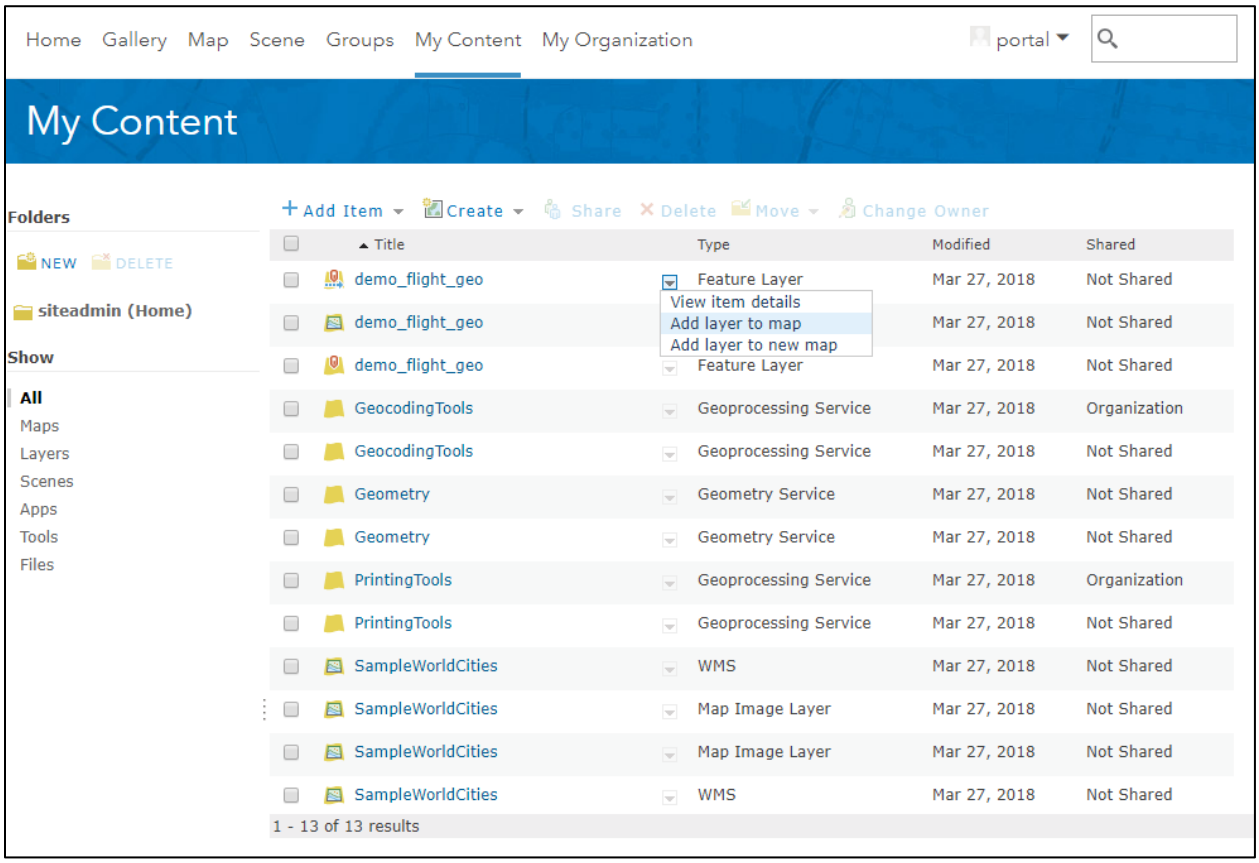


Figure 107 ArcGIS Portal

Click on Drop Down button next to **demo_flight_geo** Feature Layer and select **Add layer to map** to bring up mapping of flight data that have been ingested.

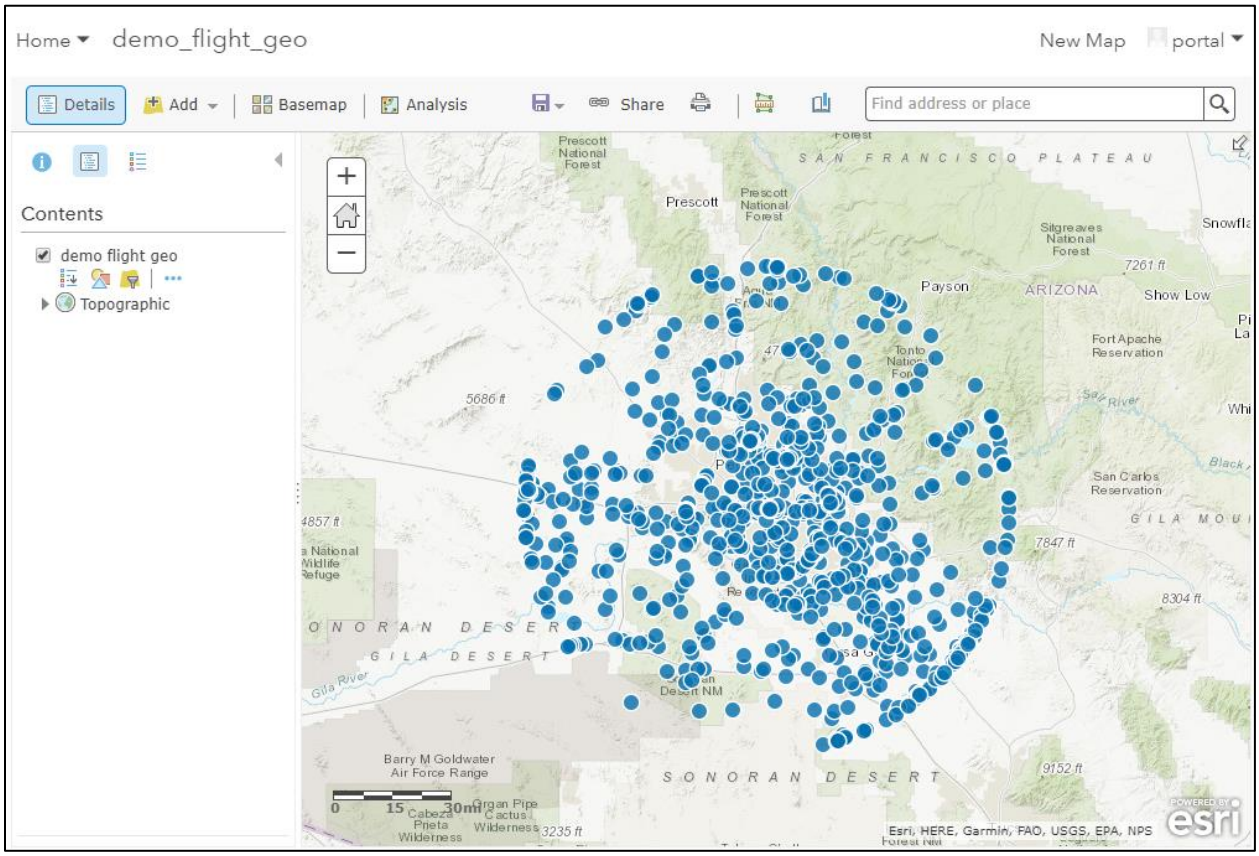


Figure 108 ArcGIS Portal - demo_flight_geo

Map Viewer can be navigated with **Zoom in** button + or **Zoom out** button -, the mouse and scroll wheel, or the arrow keys on the keyboard.

15. DG ANALYTICS WITH SPARK

15.1 ANALYTICS WITH SPARK

- Analytics with Spark
- Introduction
- Creating a simple Spark job
- Creating a Spark job for processing flight data off a Kafka topic
- Troubleshooting

15.2 INTRODUCTION

Spark is an engine with a diverse set of libraries for large-scale data processing. The application is flexible enough to access data across a variety of data sources, such as HDFS, Cassandra, HBase, and Hive. Resilient Distributed Datasets (RDD) are the most prominent building blocks for Spark, in which they serve as the program's fundamental collection of datasets. RDD's are designed to be immutable data structures, which can be expressed in Python, Java, and Scala. Due to Spark's ease of use and the ability to tap into multiple data sources, the application is a good candidate for conducting data processing on flight data flowing into ACS or other potential data to be utilized in the future. This guide will walk through how to build a 'Hello world' equivalent Spark job and will later dive into how to apply custom Spark jobs for processing flight data with Kafka and various datastores.

15.3 CREATING A SIMPLE SPARK JOB

This exercise walks through how to build a simple Spark job within an ACS environment that counts the number of lines that contain the letter 'c' and the word 'the' in the Spark README.md file. The example is based off the exercise on Spark's quick start guide https://spark.apache.org/docs/latest/quick-start.html.

Prerequisites:

- access to a development environment running either an Ubuntu or RHEL variant
- Java installed
- Sbt installed within the development environment (Simple build tool that leverages Java. Sbt install instructions can be found at https://www.scala-sbt.org/release/docs/Installing-sbt-on-Linux.html)

1. On the development environment, create a directory tree structure named my_spark_examples/src/main/scala.

```
sudo mkdir -p my_spark_examples/src/main/scala
```

Spark applications can be written in Java, Scala, and Python. This particular example uses Scala.

2. Create a build.sbt file within the my_spark_examples root directory. This file contains any dependencies that are needed to build the spark application.

```
sudo vim my_spark_examples/build.sbt


name := "Simple Project"

version := "1.0"

scalaVersion := "2.11.8"

libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.0.0"
```

The Spark and Scala versions can both be found on an ACS system by opening the spark-shell program.

Initiate the spark-shell program as the Spark user located on one of the Spark nodes. In this ACS instance, Spark 2.0.0 and Scala 2.11.8 are installed on the system.

```
[root@acs-dev1-node2 sysadmin]# su spark
bash-4.2$ /opt/spark/bin/spark-shell
...
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.0.0
      /_/

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_161)
```

3. Create the SimpleApp.scala application in the my_spark_examples/src/main/scala directory. This is the underlying Scala code that instructs Spark to find the number of lines containing the letter 'c' and the word 'the' in the Spark [README.md](#) file.

```
sudo vim my_spark_examples/src/main/scala/SimpleApp.scala


/* SimpleApp.scala */
import org.apache.spark.sql.SparkSession

object SimpleApp {
   def main(args: Array[String]) {
          val logFile = "/opt/spark/README.md" // Should be some file on
your HDFS system
        val spark = SparkSession.builder.appName("Simple
```

```
Application").getOrCreate()
        val logData = spark.read.textFile(logFile).cache()
        val numCs = logData.filter(line => line.contains("c")).count()
        val numThes = logData.filter(line => line.contains("the")).count()
        println(s"Lines with 'c': $numCs, Lines with 'the': $numThes")
        spark.stop()
    }
}
```

4. Enter the my_spark_examples directory containg the build.sbt file and initialize the build with the 'sbt package' command, which will create a jar file under my_spark_examples/target/scala-2.11/simple-project_2.11-1.0.jar.

```
cd my_spark_examples


sbt package
```

5. After the build finishes, SCP the whole my_spark_examples to the designated ACS spark master node:

```
sudo scp -rp my_spark_examples/ sysadmin@<spark_master_node_ip>:/opt/spark/
```

6. SSH into <spark_master_node>.
7. Recursively change ownership of the my_spark_examples directory to spark.

```
sudo chown -R /opt/spark/my_spark_examples/
```

8. Create a '/opt/spark' directory path in HDFS where Spark can access the README.md file.

```
sudo hdfs dfs -mkdir -p /opt/spark
sudo hdfs dfs -put /opt/spark/README.md /opt/spark/
```

9. Enter the /opt/spark on the local filesystem (not HDFS) and run simple project application with the following parameters.

```
bin/spark-submit --class "SimpleApp" --master local[4]
my_spark_examples/target/scala-2.11/simple-project_2.11-1.0.jar
```

10. As part of the output from executing the simple project application, there should be a line that lists the number of lines containing the letter 'c' and the word 'the'. This indicates that the application ran successfully.

```
Lines with 'c': 51, Lines with 'the': 22
```

15.4 CREATING A SPARK JOB FOR PROCESSING FLIGHT DATA OFF A KAFKA TOPIC

Create a Spark job prototype that can read a specified Kafka topic and return a wordcount throughout the stream.

Build.sbt file:

```
name := "Scala Kafka Project"

version := "1.0"

scalaVersion := "2.11.8"

libraryDependencies += "org.apache.spark" %% "spark-core" % "2.0.0"
libraryDependencies += "org.apache.spark" %% "spark-streaming" % "2.0.0"
libraryDependencies += "org.apache.spark" %% "spark-streaming-kafka-0-10" %
"2.0.0"

// https://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka-0-8-
assembly
libraryDependencies += "org.apache.spark" %% "spark-streaming-kafka-0-8-assembly"
% "2.0.0"
```

Scala file. A stream instance is created with KafkaUtils by pointing to the dpf topic, which is a topic that is utilized by streaming flight data.

```
import org.apache.spark._
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.kafka._
import org.apache.spark.streaming.kafka.KafkaUtils

object WordCount {

     def main( args:Array[String] ){

            val conf = new
SparkConf().setMaster("local[*]").setAppName("KafkaReceiver")
            val ssc = new StreamingContext(conf, Seconds(10))

            val kafkaStream = KafkaUtils.createStream(ssc, "acs-dev1-
node3:2181","console-consumer-87348", Map("dpf" -> 5))

            //need to change the topic name and the port number based on ACS
environment
            val words = kafkaStream.flatMap(x => x._2.split(" "))

            val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)

            kafkaStream.print() //prints the stream of data received
            wordCounts.count().print() //prints the wordcount result of the stream

            ssc.start()
            ssc.awaitTermination()

    }
```

```
}
```

Kafka group id for dpf can be found with the following steps:

List Kafka group ids on a kafka node.

```
[sysadmin@acs-dev1-node3 ~]$ /opt/kafka/kafka-current/bin/kafka-consumer-groups.sh
--list --zookeeper acs-dev1-node1,acs-dev1-node2,acs-dev1-node3
console-consumer-77463
console-consumer-56732
1
```

Verify dpf is in console-consumer-77463.

```
[sysadmin@acs-dev1-node3 ~]$ /opt/kafka/kafka-current/bin/kafka-consumer-groups.sh
--zookeeper acs-dev1-node1,acs-dev1-node2,acs-dev1-node3 --describe --group
console-consumer-77463
GROUP                          TOPIC                          PARTITION  CURRENT-
OFFSET  LOG-END-OFFSET  LAG             OWNER
console-consumer-77463         dpf                            0          10747
0               -10747          none
console-consumer-77463         dpf                            1          10736
0               -10736          none
```

Steps to invoke the Spark job:

1. SSH into <spark_master>.
2. Wget the spark streaming kafka assembly jar from Maven upstream. Place the jar in the Spark home directory at /opt/spark. Even though the assembly jar is listed as a dependency in the build.sbt file, it still needs to be explicitly stated with running spark-submit since Spark may be pointing to the wrong directory at run-time.

   ```
   sudo wget http://central.maven.org/maven2/org/apache/spark/spark-streaming-
   kafka-0-8-assembly_2.11/2.0.0/spark-streaming-kafka-0-8-assembly_2.11-
   2.0.0.jar
   ```

3. Su into spark user
4. Invoke the Spark kafka word count job

   ```
   bin/spark-submit --jars spark-streaming-kafka-0-8-assembly_2.11-2.0.0.jar -
   -class "WordCount" --master local[4]
   demo_examples/word_count_example_2.0/target/scala-2.11/scala-kafka-
   project_2.11-1.0.jar
   ```

5. Assuming that data has not been ingested into the system yet, a zero word count should output within the console after the word count job is invoked

```
-------------------------------------------
Time: 1528481330000 ms
-------------------------------------------
0
```

6. SSH into <storm_master> in another terminal tab
7. Su into storm
8. Ingest some static flight data to populate the DPF topic.

```
bash-4.2$ /opt/common/util/tcri/utility/bin/dpf-utility ingest -s kafka -l
demo-flight-geo /opt/storm/topologies/tcri/streaming-
dpf/sample/flight_data_sample.json
```

9. Switch back to the <spark_master> terminal tab.
10. Word count output should now increment within the console.

```
-------------------------------------------
Time: 1528481410000 ms
-------------------------------------------
1894
```

11. Terminate the streaming Spark job with CTRL+Z.
12. Check Spark history server UI for Kafka word count job output at <spark_history_server>:18090. Ensure that the latest Spark job is logged within the Spark history server by checking the "Last Updated" timestamp.

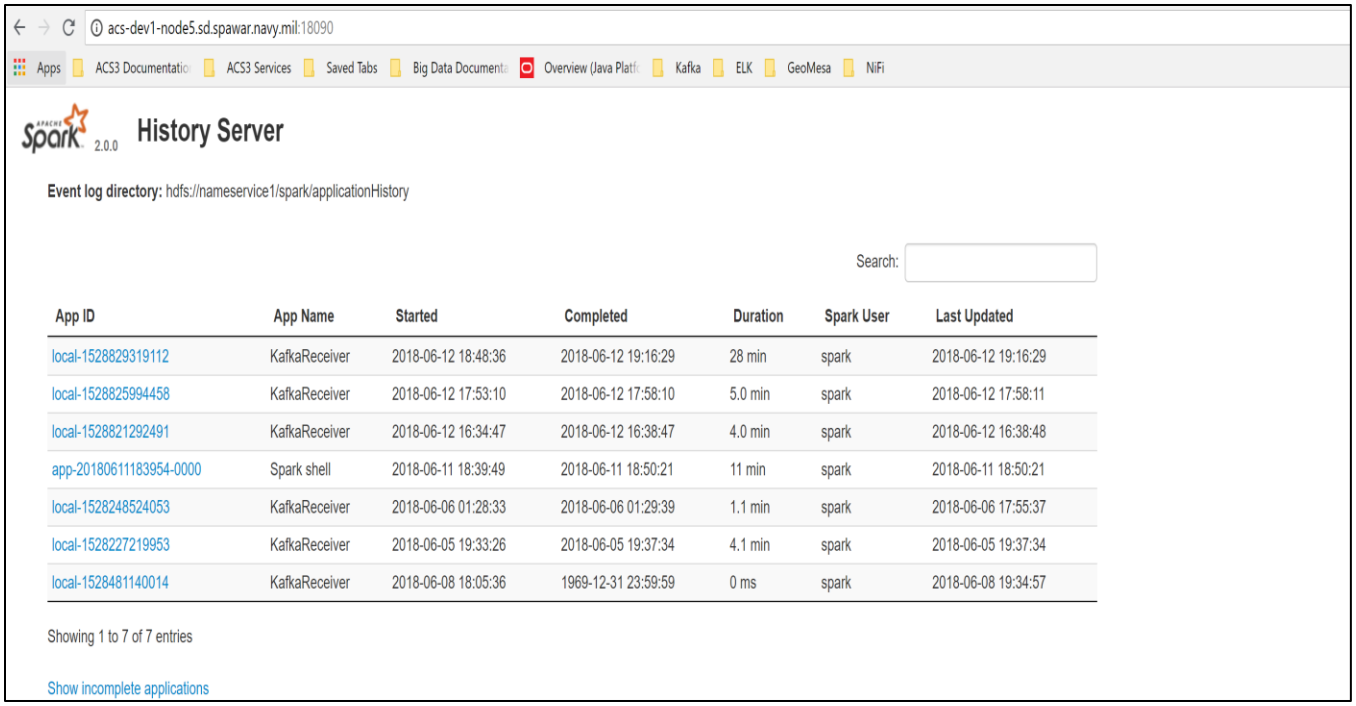


Figure 109 Spark History UI

13. Ensure the Spark workers are alive through the Spark master UI at <spark_master_server>:18080.

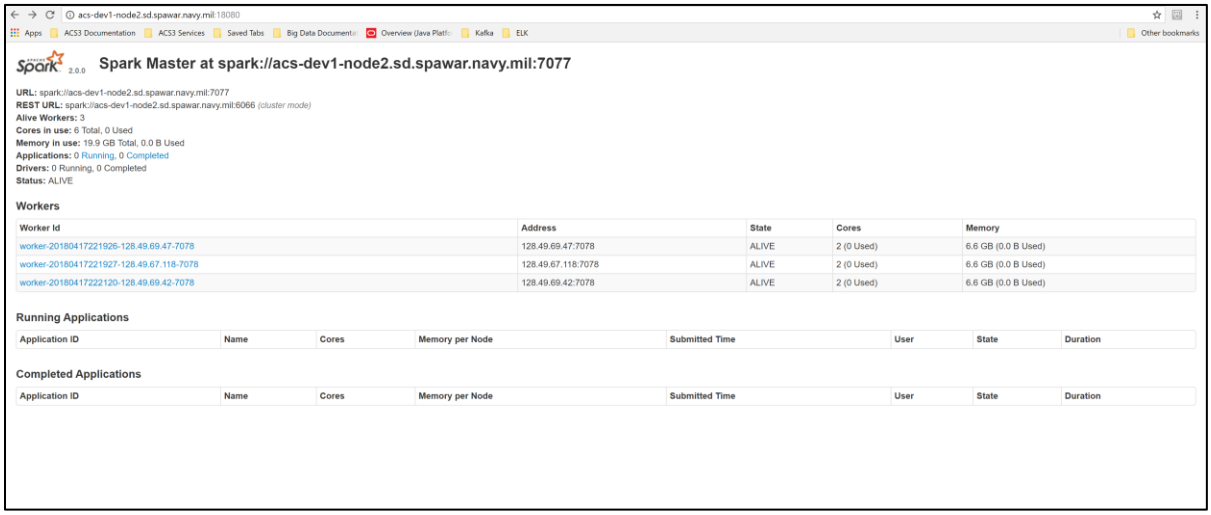


Figure 110 Spark Master UI

15.5 TROUBLESHOOTING

When running Spark jobs or the Spark shell, you might encounter a permissions error that resembles the following message.

```
18/04/30 18:00:18 ERROR storage.DiskBlockManager: Failed to create local dir in
/data00/spark/local. Ignoring this directory.
java.io.IOException: Failed to create a temp directory (under /data00/spark/local)
after 10 attempts!
        at org.apache.spark.util.Utils$.createDirectory(Utils.scala:281)
        at
org.apache.spark.storage.DiskBlockManager$$anonfun$createLocalDirs$1.apply(DiskBlo
ckManager.scala:132)
        at
org.apache.spark.storage.DiskBlockManager$$anonfun$createLocalDirs$1.apply(DiskBlo
ckManager.scala:130)
        at
scala.collection.TraversableLike$$anonfun$flatMap$1.apply(TraversableLike.scala:24
1)
        at
scala.collection.TraversableLike$$anonfun$flatMap$1.apply(TraversableLike.scala:24
1)
        at
scala.collection.IndexedSeqOptimized$class.foreach(IndexedSeqOptimized.scala:33)
        at scala.collection.mutable.ArrayOps$ofRef.foreach(ArrayOps.scala:186)
        at
scala.collection.TraversableLike$class.flatMap(TraversableLike.scala:241)
        at scala.collection.mutable.ArrayOps$ofRef.flatMap(ArrayOps.scala:186)
        at
org.apache.spark.storage.DiskBlockManager.createLocalDirs(DiskBlockManager.scala:1
30)
        at
org.apache.spark.storage.DiskBlockManager.<init>(DiskBlockManager.scala:42)
        at org.apache.spark.storage.BlockManager.<init>(BlockManager.scala:80)
        at org.apache.spark.SparkEnv$.create(SparkEnv.scala:320)
        at org.apache.spark.SparkEnv$.createDriverEnv(SparkEnv.scala:165)
        at org.apache.spark.SparkContext.createSparkEnv(SparkContext.scala:259)
        at org.apache.spark.SparkContext.<init>(SparkContext.scala:423)
        at
org.apache.spark.streaming.StreamingContext$.createNewSparkContext(StreamingContex
t.scala:836)
        at
org.apache.spark.streaming.StreamingContext.<init>(StreamingContext.scala:84)
        at WordCount$.main(KafkaWordCount2.scala:12)
        at WordCount.main(KafkaWordCount2.scala)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at
org.apache.spark.deploy.SparkSubmit$.org$apache$spark$deploy$SparkSubmit$$runMain(
SparkSubmit.scala:729)
        at org.apache.spark.deploy.SparkSubmit$.doRunMain$1(SparkSubmit.scala:185)
        at org.apache.spark.deploy.SparkSubmit$.submit(SparkSubmit.scala:210)
        at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:124)
        at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
```

This error message is most likely the result of not running the Spark job as the Spark user. Ensure that the Spark job is ran as the Spark user. Also, ensure Spark permissions are set for the /data00/spark directory.

```
su spark
sudo chown -R spark:spark /data00/spark
```

16. DG INTELLEX ALERTS

16.1 OBJECTIVE

In this section, we will set up an IntellEx Interest Model that receives flight data through IntellEx Simple Topic and generates alerts for flights flying below 2,000ft. Flight data will be deposited from NiFi into Intellex POST REST endpoint.

The data flow is depicted below:

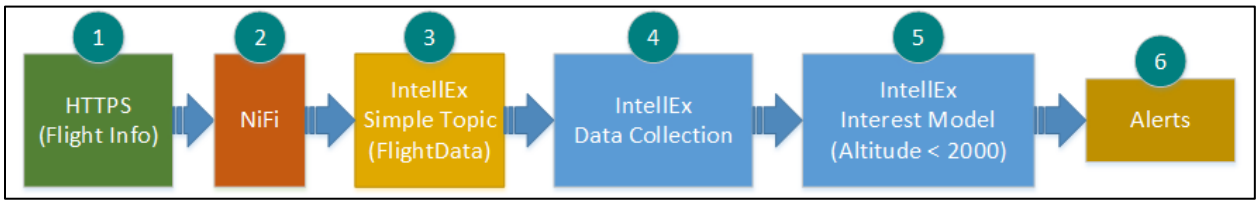


Figure 111 IntellEx Alert Data Flow

1. Real time position of flights around the world can be queried from https://www.adsbexchange.com/data/. In this example, we will use the NiFi set up for flight data example in earlier sections
2. NiFi will post flight data into IntellEx REST POST endpoint under "FlightData" topic
3. IntellEx receives flight data via its Simple Topic source
4. IntellEx data collection extracts interested information and feeds into its knowledge base
5. Interest models looks for flights flying below 2,000ft
6. Create alerts for these low-altitude flights

16.2 PREPARATION

This example requires the NiFi GetHTTP processor already setup to acquire flight data from https://www.adsbexchange.com/data/ as in the earlier sections.

For convenience, the following windows can be opened and ready for the configuration:

- NiFi UI: http://<nifi>:6434/nifi
- IntellEx Config UI: http://<intellex_web>:8443/IntellexConfigUI
- Terminal session to <intellex_web> node (to watch log during configuration at /opt/jboss/jboss-current/standalone/log/server.log)

16.3 NIFI SETUP

NiFi PostHTTP processor will be required to post the flight data to a location where IntellEx can process and that location is the IntellEx HTTP REST POST endpoint, namely http://<intellex_web>:8080/workflow/postdata, followed by a topic name. For this example, the topic name will be FlightData. Hence, the NiFi PostHTTP processor will need to be defined with the following properties:

- URL: http://<intellex_web>:8080/workflow/postdata/FlightData
- Send as FlowFile: false
- Use Chunked Encoding: false
- Compression Level: 0
- Automatically Terminate Relationships: on failure and success (on Settings Tab)

Next, GetHTTP processor needs to be connected to PostHTTP processor on success as seen on the figure below.

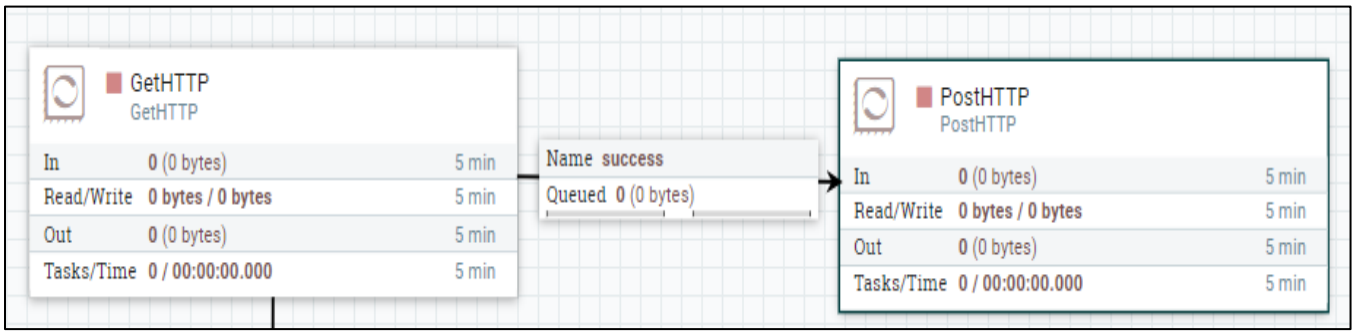


Figure 112 Configure NiFi PostHTTP Processor

16.4 INTELLEX SETUP

All IntellEx configurations in the following sections will be done on IntellEx ConfigUI at http://<intellex_web>:8443/IntellexConfigUI. During configuration, Jboss log, located at /opt/jboss/jboss-current/standalone/log/server.log on intellex_web node could be monitored for any configuration errors.

16.4.1 Workspace

Default-Workspace will be used. Other workspace can be created and utilized if interested.

16.4.2 Data Source

intellEx Data Sources define endpoints representing external sources of data. They may take the form of an endpoint URL, may point to a publish/subscribe service such as Kafka via a topic name, or may point to other types of sources, such as a database table. These sources are dynamically configurable within IntellEx at run time.

Data sources contain configuration information necessary for a data handler to create a connection to the endpoint and poll or listen for data. In this example, incoming data will be deposited to the http://<intellex_web>:8080/workflow/postdata/<topic> where topic is FlightData as stated before. This source type is call SIMPLE_TOPIC in IntellEx.

The following configuration items are defined for data source:

- Name: FlightDataSimpleTopicSource
- SourceType: SIMPLE_TOPIC
- Enabled: true

- Parameters:

```
{
 "topicName" : "FlightData"
}
```

The data source configuration will look similar to the following figure.

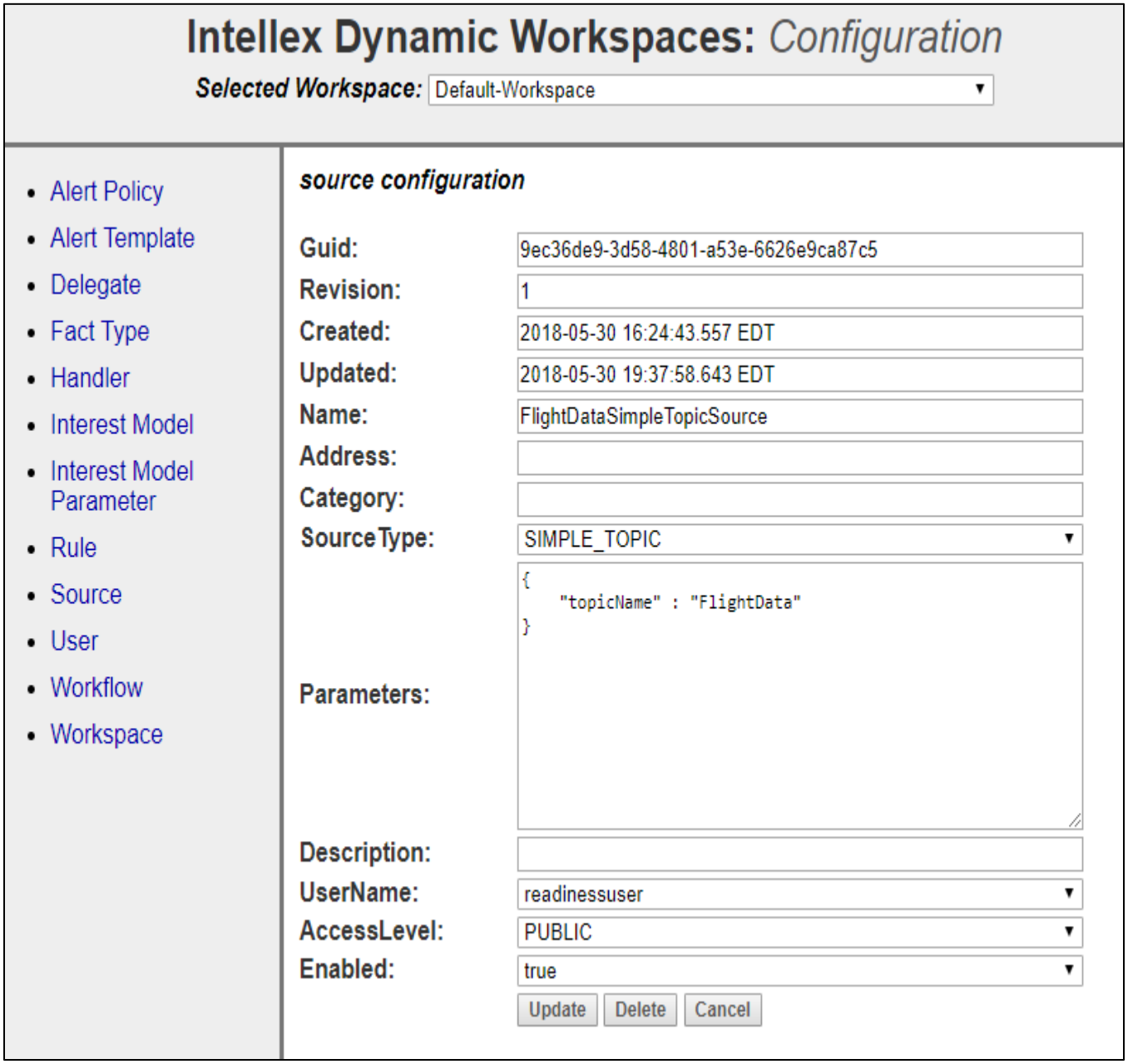


Figure 113 Configure IntellEx Source

16.4.3 Fact Type

The IntellEx Knowledge Base consists of data objects called Facts. A single Fact is of a single Fact Type, which defines available Fields, Field Types, and Relations to other Fact Types. Fact Types are analogous to Class definitions in Object Oriented programming, while Facts are analogous to Class instances. A key differentiation is the ability for Fact Types to be defined and manipulated at run time through configuration and available user interfaces. This is a key concept when integrating IntellEx with new data sources - no compiled changes to the IntellEx platform are necessary.

In this example, we will separate flight data into two facts: an Aircraft fact that contains such aircraft attributes as its model, manufacturer, operating airline, etc. and a Flight Fact that contains information about the current flight like speed, altitude, geo-position, etc.

16.4.3.1 Aircraft Fact Type

An Aircraft Fact Type will be added with the following attributes:

- Name: Aircraft
- Package Name: intellex.fact.flightdata
- Definition:

```
<factDefinition name="Aircraft" key="icao">
  <field name="icao" type="String" required="true"/>
  <field name="airline" type="String"/>
  <field name="model" type="String"/>
  <field name="manufacturer" type="String"/>
  <field name="year" type="Integer"/>
  <field name="country" type="String"/>
</factDefinition>
```

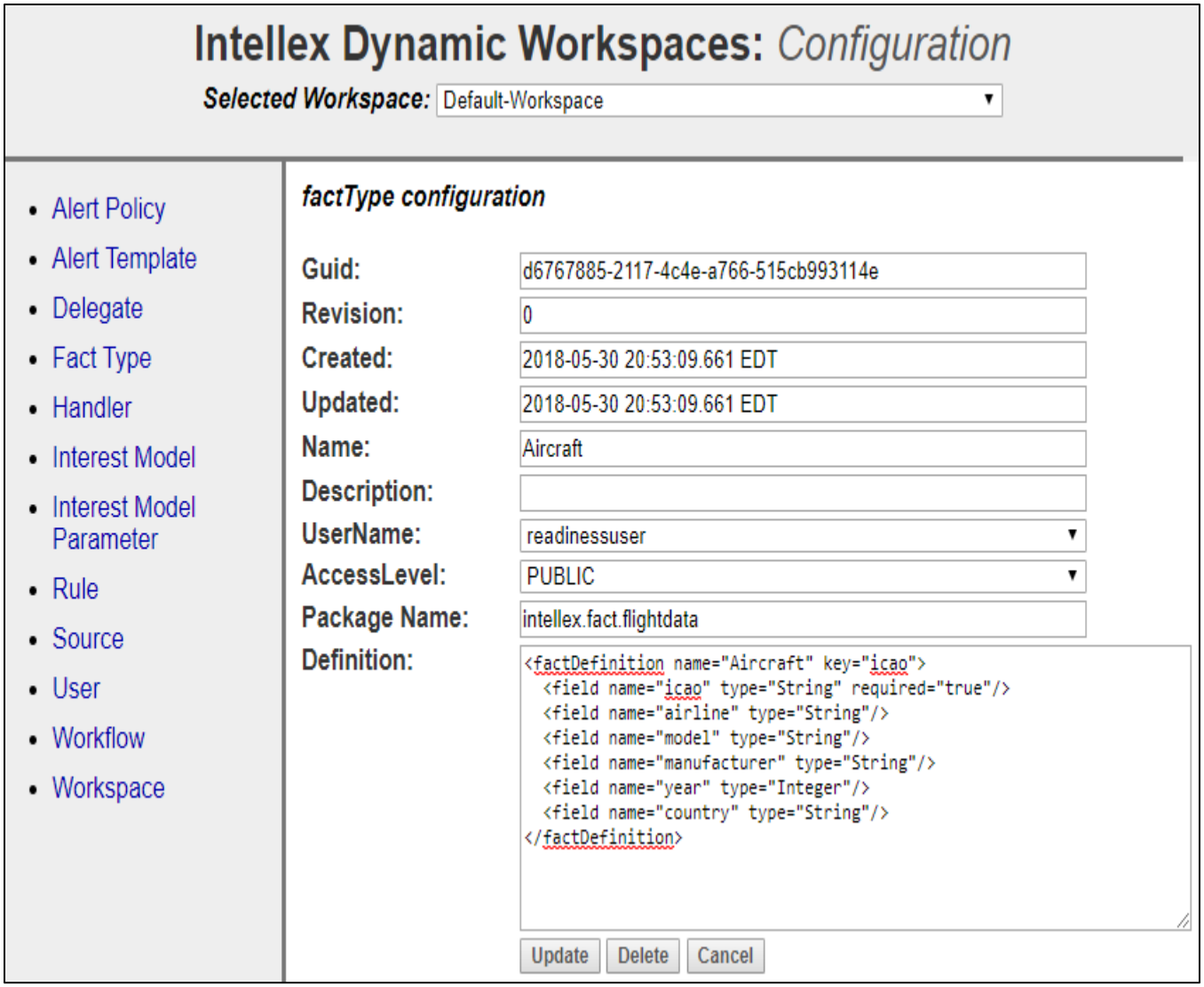


Figure 114 Configure Aircraft Fact Type

16.4.3.2 Flight Fact Type

A Flight Fact Type will be added with the following attributes:

- Name: Flight
- Package Name: intellex.fact.flightdata
- Definition:

```
<factDefinition name="Flight" key="icao">
  <field name="icao" type="String" required="true"/>
  <field name="from" type="String"/>
  <field name="to" type="String"/>
  <field name="altitude" type="Integer"/>
  <field name="latitude" type="Double"/>
  <field name="longitude" type="Double"/>
```

```
  <field name="speed" type="Double"/>
  <field name="speedtype" type="Integer"/>
  <field name="angle" type="Double"/>
</factDefinition>
```

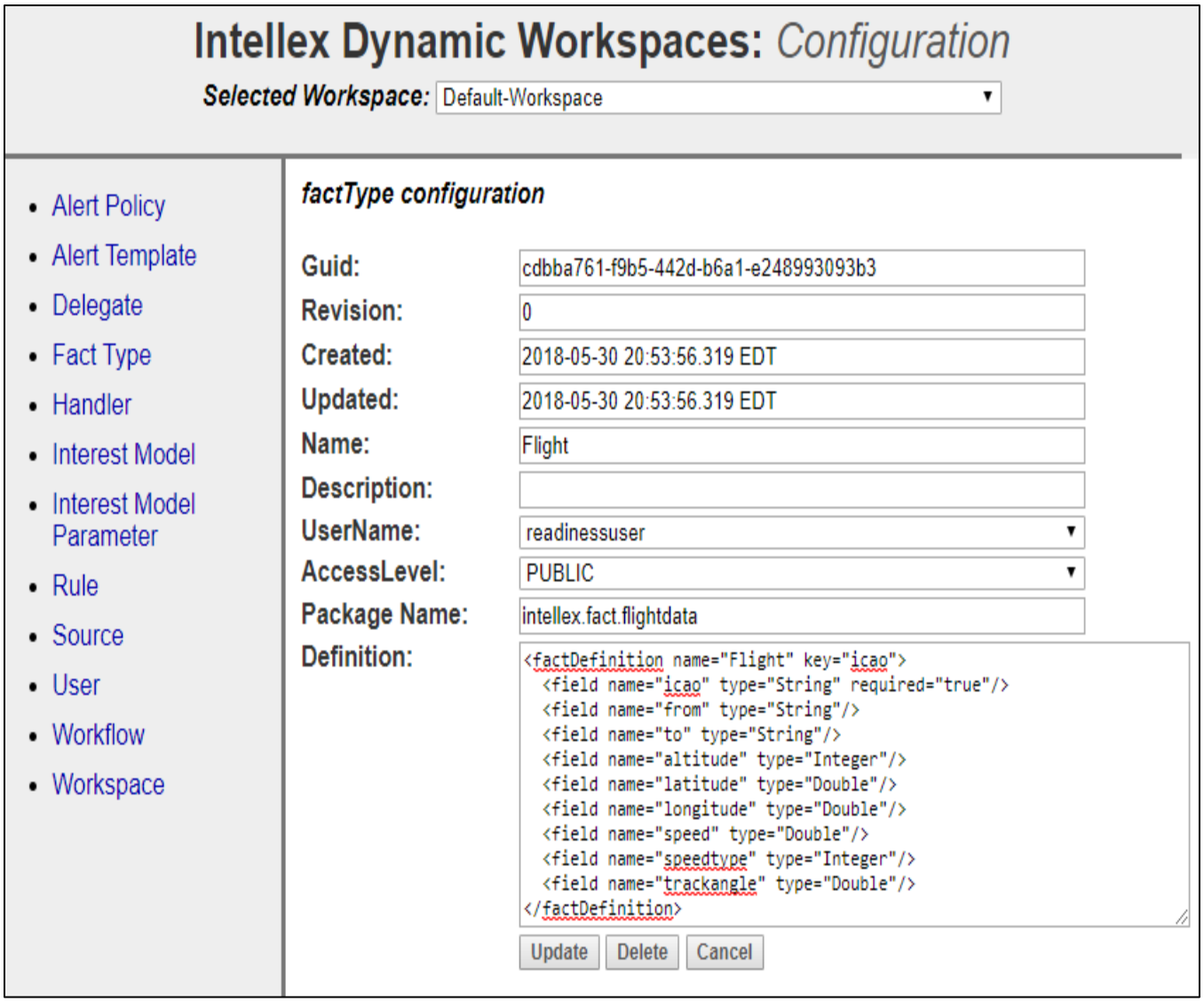


Figure 115 Configure Flight Fact Type

IntellEx ConfigUI may show error when adding second fact type to the effect that the application needs restart. Application restart can be performed at an<intellex_web> node by executing the following command at directory: /opt/jboss/jboss-current/standalone/deployments:

```
$ mv IntellexDelegatePackage.ear.deployed IntellexDelegatePackage.ear.dodeploy
```

IntellexDelegatePackage will be re-deployed, and its extension will be changing from dodeploy to isdeploying and then deployed for the application restart to be successful.

16.4.4 Data Handler

Data Handlers create connections to external data sources and ingest data into the intellEx knowledge base. Data handlers can be synchronous pollers as in REST calls or asynchronous listeners as in JMS. A handler connects to a single data source using configuration parameters, receive raw data from the source on one or more calls, and activates processing chains consisting of workflow steps that implement data synchronization activities with the knowledge base.

16.4.4.1 JSON Parsing Script

In this example, we will use a Java script to parse flight data and populate corresponding Aircraft and Flight objects.

A file, namely JsonFlightDataExtractor.bsh, will need to be created on <intellex_web> node at the location /opt/jboss/jboss-current/standalone/configuration/intellex/scripts/. The file content is as follows:

```
package intellex.extractor.acs;

import org.cubrc.intellex.workspace.data.StructuredData;
import org.cubrc.intellex.workspace.data.action.ActionChainContext;
import org.cubrc.intellex.workspace.data.extractor.DataExtractor;
import org.cubrc.intellex.workspace.data.extractor.ExtractionResult;
import org.cubrc.reasoning.fact.FactTypeIdentifier;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.DeserializationFeature;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


public class JsonFlightDataExtractor implements DataExtractor {

    public final static Logger log =
LoggerFactory.getLogger(JsonFlightDataExtractor.class);
    public final static FactTypeIdentifier AIRCRAFT = new
FactTypeIdentifier("intellex.fact.flightdata", "Aircraft");
    public final static FactTypeIdentifier FLIGHT = new
FactTypeIdentifier("intellex.fact.flightdata", "Flight");

    public ExtractionResult extract(ActionChainContext actionChainContext) {

        List structuredDataList = new ArrayList();

        Object payload = actionChainContext.getPayload();

        try {
            String jsonString = (String)payload;

            ObjectMapper objectMapper = new ObjectMapper();
            JsonNode root = objectMapper.readTree(jsonString);
```

```
            JsonNode acList = root.get("acList");
            if (acList.isArray()){
                log.info("Processing " + acList.size() + " aircrafts");

                for (int i = 0; i < acList.size(); i++ ) {

                    JsonNode aircraft = acList.get(i);
                    if (aircraft == null) continue;

                    String icao = aircraft.get("Icao")==null ? "" :
aircraft.get("Icao").asText();
                    String airline = aircraft.get("Op")==null ? "" :
aircraft.get("Op").asText();
                    Integer altitude = aircraft.get("Alt")==null ? 0 :
aircraft.get("Alt").asInt();
                    Double latitude = aircraft.get("Lat")==null ? 0.0 :
aircraft.get("Lat").asDouble();
                    Double longitude = aircraft.get("Long")==null ? 0.0 :
aircraft.get("Long").asDouble();

                    String from = aircraft.get("From")==null ? "" :
aircraft.get("From").asText();
                    String to = aircraft.get("To")==null ? "" :
aircraft.get("To").asText();
                    String model = aircraft.get("Mdl")==null ? "" :
aircraft.get("Mdl").asText();
                    String manufacturer = aircraft.get("Man")==null ? "" :
aircraft.get("Man").asText();
                    String country = aircraft.get("Cou")==null ? "" :
aircraft.get("Cou").asText();

                    Double speed = aircraft.get("Spd")==null ? 0.0 :
aircraft.get("Spd").asDouble();
                    Integer speedtype = aircraft.get("SpdTyp")==null ? 0 :
aircraft.get("SpdTyp").asInt();
                    Double angle = aircraft.get("Trak")==null ? 0.0 :
aircraft.get("Trak").asDouble();

                    Integer year = aircraft.get("Year")==null ? 0 :
aircraft.get("Year").asInt();

                    Map aircraftMap = new HashMap();
                    aircraftMap.put("icao", icao);
                    aircraftMap.put("airline", airline);
                    aircraftMap.put("model", model);
                    aircraftMap.put("manufacturer", manufacturer);
                    aircraftMap.put("year", year);
                    aircraftMap.put("country", country);

                    Map flightMap = new HashMap();
                    flightMap.put("icao", icao);
                    flightMap.put("from", from);
                    flightMap.put("to", to);
                    flightMap.put("altitude", altitude);
                    flightMap.put("latitude", latitude);
                    flightMap.put("longitude", longitude);
                    flightMap.put("speed", speed);
                    flightMap.put("speedtype", speedtype);
                    flightMap.put("angle", angle);
```

```
                    StructuredData flightStructuredData = new
StructuredData(FLIGHT, (String) null, flightMap);
                    StructuredData aircraftStructuredData = new
StructuredData(AIRCRAFT, (String) null, aircraftMap);

                    structuredDataList.add(flightStructuredData);
                    structuredDataList.add(aircraftStructuredData);
                }
            }
        } catch( Exception e ) {
            e.printStackTrace();
        }

        return new ExtractionResult(structuredDataList);
    }
}
```

16.4.4.2 Data Handler

16.4.4.2.1 Data Processing Chains

A Flight data handler will listen to the REST POST endpoint for incoming flight data and will be processed by the following processing chain steps:

- Pre-processing chain - an optional chain that executes once before each request cycle (before data arrive)
- Action chain - a required chain that executes once for each incoming payload
- Post processing chain - an optional chain that executes once after each request cycle

16.4.4.2.2 Chain Nodes

A chain is constructed by combining one or more Action Nodes. These Action Nodes function as atomic operations within the defined workflow and fall into one of five categories:

Data Transformers

A Data Transformer transforms an Action Chain payload in some fashion to another payload. Transformer performs tasks such as filtering out unwanted data, generating facts from extracted data, and finding related facts for a set of data. Specific examples include:

- StringToDocumentTransformer - transforms a string payload into an XML document
- FactGeneratingTransformer - transforms structured data outputs from an extractor into fact objects
- FactLocatingTransformer - finds facts instances from the reasoner that match a structured data input
- StateChangeFilter - removes data from a collection of structured data that has previously been inserted as facts into the IntellEx reasoner, and has not changed from its previous state. This node usually follows an extractor node in an Action chain and serves to improve efficiency of the system by reducing the number of structured data items that need to be processed by the FactGeneratingTransformer

Data Extractors

A Data Extractor node take a processing chain payload and generates structured data objects which can then be transformed into fact objects by the FactGeneratingTransformer. The output payload of the node will always be a list of structured data objects. Extractors generally occur near or at the beginning of an action chain and take a raw data input payload.

Some examples of Data Extractors are follows:

- Beanshell Data Extractor - uses Java Beanshell script to extract structured data from a raw payload
- XPath Data Extractor - extracts simple fact type accross a single XML document
- JSON Path Data Extractor - extracts simple fact type accross a single JSON string

Data Processors

A Data Processor operates on an Action Chain payload but passes through the original input payload to output without modifying it, thus no changes will be applied to the payload from the processor.

Some examples of the Data Processors are follows:

- Payload Printing Procesor - writes payload contents to the log file and optionally to the file system (great for debugging)
- Fact Retracting Processor - removes facts from the system
- Reference Persisting Processor - provides a mechanism where raw data can be saved for later retrieval from the database
- Embedded Call Processor - invokes a child data handler during processing of a data item in the parent handler

Control Structures

A Control Structure node controls how a chain is being executed, without any dependencies on the input payload of the node. The common example of this is the Splitting Action Node, that splits a chain into two concurrent pathways. These nodes pass through the original input payload to output without modifying it, so the node can be placed anywhere in the chain.

Actions

Action nodes perform an action for the chain that is not specifically dependent on the input payload. Action nodes pass throught the original input payload to output without modifying it, so an action node can be placed anywhere in the chain. Common examples of Action nodes are follows:

- Reasoning Action Node - indicates to the system that new information is available for rule processing
- Begin Transaction Action Node - start an alert transaction with respect to the data that the handler is producing
- Commit Transaction Action Node - completes a transaction and releases all alert generator deferrals for the handler

16.4.4.2.3 Flight Data Handler

The diagram below illustrates the workflow for the example.

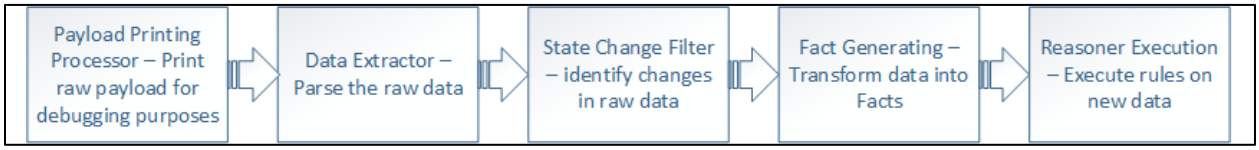


Figure 116 Flight Data Workflow

Flight data handler will be defined as follows:

- Name: FlightDataTopicHandler
- Enable: true
- Schedule Type: LISTENER
- Startup Type: AUTOMATIC
- Pre Processing Chain Definition Type: NONE
- Action Chain Definition:

```
<DataActionConfiguration
   xmlns="urn:org.cubrc.intellex.workspace.data.action"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <Node xsi:type="ProcessingDataActionNodeType">
        <Processor xsi:type="PayloadPrintingDataProcessorType"/>
        <OutputDirectory>/tmp/intellex/payload</OutputDirectory>
    </Node>
    <Node xsi:type="ExtractingDataActionNodeType">
        <Extractor xsi:type="BeanShellExtractorType">

<ScriptPath>file:${jboss.server.config.dir}/intellex/scripts/JsonFlightDataExtract
or.bsh</ScriptPath>
        </Extractor>
    </Node>
    <Node xsi:type="TransformingDataActionNodeType">
        <Transformer xsi:type="StateChangeFilterTransformerType"/>
    </Node>
    <Node xsi:type="TransformingDataActionNodeType">
        <Transformer xsi:type="FactGeneratingTransformerType"/>
    </Node>
    <Node xsi:type="ReasoningDataActionNodeType"/>
</DataActionConfiguration>
```

- Action Chain Definition Type: XML
- Post Processing Chain Definition Type: NONE
- Data Source Identifier: FlightDataSimpleTopicSource
- Workflow Id: Default-Service-Workflow
- Channel Type: POST

16.4.5 Interest Model Rules

Interest Model Rules express the business logic necessary to generate the desired information products, and are executed by the reasoning engine. Rules seek sets of facts matching specific criteria and perform actions when those criteria are met. Actions can range from simply tagging facts to be included in the report to complex computation resulting in the creation of new Facts. It is the responsibility of the rules to examine the Facts and determine what data should be reported. Through intermediate Facts, rules can be chained together to invoke other rules allowing for data-driven and change-driven analytics of arbitrary complexity and resolution.

16.4.5.1 Rule Type

Rules can be authored in either the native Drools or intellEx IXRL languages. IntellEx provides the IXRL rule language for authoring custom rules. IXRL uses Java syntax for rule actions, and a proprietary filtering language for the rule conditions capable of leveraging relations established in fact definitions. Drools rule language is described at https://docs.jboss.org/drools/release/5.2.0.Final/drools-expert-docs/html/ch05.html.

16.4.5.2 Rule Package

Rules exist within packages. it is an intellEx convention for a package to contain only a single rule. This allows for rules to be inserted and removed from the Knowledge Base individually.

For this example, we will define a Low Altitude rule that looks for flights that flying at below 2,000 ft. The rule is defined as followed:

- Name: LowAltitudeRules
- Rule Type: NATIVE
- Package Name: intellex.rule.flightdata
- Defiinition:

```
package intellex.rule.flightdata

import org.cubrc.reasoning.alert.AlertGenerator
import org.cubrc.reasoning.alert.Disposition

import intellex.fact.flightdata.Aircraft
import intellex.fact.flightdata.Flight

rule "LowAltitudeRule1"

   when
      $aircraft : Aircraft()
      $flight : Flight(icao == $aircraft.icao, altitude < 2000)
   then
      try {
         log.info("Executing LowAltitudeRules Rules  ");
         log.trace("Aircraft: {}", $aircraft);
         log.trace("Flight: {}", $flight);

         // Send alerts
         AlertGenerator alertGeneratorInstance =
```

```
alertGenerator.getInstance("LowFlyingAircraftModel");
         alertGeneratorInstance.addData("Aircraft", $aircraft,
Disposition.TERMINAL);
         alertGeneratorInstance.addData($aircraft, "Flight", $flight,
Disposition.TERMINAL);

      } catch (Exception e) {
         log.error("LowFlyingAircraftModel: {}", e.toString());
      }
end
```

16.4.6 Interest Model Product (Alert) Generation

Alert generation takes place at the end of reasoner execution. During reasoner execution, rules expose Facts within hierarchical collection structures to be included in serialized Interest model products. The rules thus generate a hierarchy of report data, which can then be formatted into an interest Model products (alert) through the use of a template.

Rules add interest model data to this hierarchical data structure through the set of interest model Functions described above. These functions place Facts within named collections and relate collections to other Facts (thus establishing hierarchical relationships between Facts).

16.4.6.1 Alert Template

An alert template is an XML document describing how the resulting Interest model Alert should be formatted. The template schema provides means for iterating over collections, checking existence of collections or objects, binding objects to variables, accessing object properties, and a number of built in functions.

16.4.6.1.1 Persisting Alerts

The IntellEx alert generation processes and the IntellEx Alert Service both have capabilities to automatically persist alerts in the database. While it is desirable to persist alerts for most interest model use cases, ther may be times when an alert history is not needed. To control this, the "persist" flag can be set on the "Alert" element to provide guidance to above processes on how to handle persistence of new alerts. This flag would be set on the "Alert" element in the template and would subsequently pass through to the actual generated alerts.

Persist flag takes on the following values:

- ALL or true: means that new alerts will always be saved as new records in the database. This is default settings if "persist" flag was not specified
- NEVER or false: means that new alerts will not be saved to the database
- LATEST-ONLY: means that new alerts will replace the latest alert of the same name and interest model combination in the database.

Example usage:

<Alert id="abc123">: Not specifying the "persist" flag means the same thing as "ALL".

<Alert id="abc123" persist="ALL">

16.4.6.2 forEach

The <forEach> element is used to iterate over a collection, biding each entry to a given variable name. This is useful for repeating segments of XML for multiple entries. <forEach> element has to following attributes:

- collection - (required) the name to the collection to iterate over
- bind - (required) the name of the variable to bind each object to
- dirtyOnly - (optional) true/false, true to show dirty collection members and their children and false to show all collection members and their children

16.4.6.3 exists and notExists

The <exists> and <notExist> elements are used to check whether a collection or objects is present in the data. This is useful for ormiting a segment of XML when no data exists to populate it, or to provide an alternate section when data is missing.

For this example, we will generate alerts by listing the aircraft properties, such as operator or country, and its flight properties like altitudes. The template is defined bellow.

- Name: LowAltitudeFlightAlertTemplate
- Definiton:

```
<as:Alert xmlns:as="urn:org.cubrc.intellex.alert"
          xmlns:T="http://www.cubrc.org/intellex/alert/template"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          id="${randomGuid()}">
   <as:Name>LowAltitudeFlights</as:Name>
   <as:Source>http://${serverUrl()}</as:Source>
   <as:Destination>
      <as:Channel>NONE</as:Channel>
   </as:Destination>
   <as:GenerationTime>${now()}</as:GenerationTime>
   <as:Type/>
   <as:Description/>
   <as:Payload>
      <Flights xmlns:acs="urn:intellex.acs.flightdata">
         <T:forEach collection="Aircraft" bind="aircraft">
            <Flight id="${aircraft.icao}">
               <Operator>${aircraft.airline}</Operator>
               <Country>${aircraft.country}</Country>
               <T:forEach collection="aircraft.Flight" bind="flight">
                  <Origination>${flight.from}</Origination>
                  <Destination>${flight.to}</Destination>
                  <Altitude>${flight.altitude}</Altitude>
               </T:forEach>
            </Flight>
         </T:forEach>
      </Flights>
   </as:Payload>
</as:Alert>
```

16.4.7 Alert Distribution Policy

Following Alert Generation, the automated distribution of alerts is governed by Distribution Policies. Each policy establishes a set of distribution channels, and specifies which Interest Model alerts should be distributed over each channel. Distribution policies, defined independently from Alert Templates and Interest models, allow the same interest Model product to be routed via multiple protocols to possibly many consumer endpoints and in possibly many different formats. For example, an Interest Model alert may be published over an AMQP broker topic, as well as sent via SMTP to a recipient email list. Distribution Policies are specified in XML format.

16.4.7.1 Channels

Support is provided for broker-based like Java Message Service (JMS), file based (file system, FTP, etc.), email (SMTP), and HTTP POST channel. Every channel can have an optional transformer to convert the alert XML to another format.

16.4.7.2 Decimation

Decimation can be applied to a Distribution Channel to restrict or further control the frequency of alert updates which are published via asynchronous protocols such as SMTP, JMS,etc. The nature of data-driven analytics in IntellEx can potentially lead to alerts being updated at a much higher rate than consumers wish to receive those updates. Decimation settings on Distribution Channels can be used in this case to limit how often updates are sent out on a per channel and Interest model basis.

The Alert Policy for this example is defined as followed:

- Name: AlertServicePolicy
- Definition:

```
<Policy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="urn:org.cubrc.intellex.alert.dissemination"
        id="AlertServicePolicyId"
        name="AlertServicePolicy">

  <!-- This ensures the Alert Service receives and caches alerts -->
  <Channel name="AlertServiceChannel" xsi:type="JmsChannel">
    <BrokerUrl>localhost:1099</BrokerUrl>
    <Topics>
      <Topic>/topic/ixClientResponseTopic</Topic>
    </Topics>
    <ConnectionFactoryName>java:/ConnectionFactory</ConnectionFactoryName>
    <JndiProperties>
      <Property name="java.naming.factory.initial"
value="org.jboss.as.naming.InitialContextFactory" />
      <Property name="java.naming.factory.url.pkgs"
value="org.jboss.naming:org.jnp.interfaces" />
    </JndiProperties>
    <MessageMetadata>
      <Property name="command" value="newAlert" />
    </MessageMetadata>
    <ReconnectWaitSec></ReconnectWaitSec>
    <!--Decimation type="dm:SynchronizedDecimation">
    </Decimation-->
```

```
  </Channel>

  <!-- This sends alerts to the Table Generation Service. Needs configuration -->
  <Channel name="TableGenerationChannel" xsi:type="JmsChannel">
    <Transformer>

<Script>${jboss.server.config.dir}/intellex/scripts/AlertMessageWrapperTransform.b
sh</Script>
      <ClassName>AlertMessageWrapperTransform</ClassName>
    </Transformer>
    <BrokerUrl>localhost:1099</BrokerUrl>
    <Topics>
      <Topic>/topic/ixClientResponseTopic</Topic>
    </Topics>
    <ConnectionFactoryName>java:/ConnectionFactory</ConnectionFactoryName>
    <JndiProperties>
      <Property name="java.naming.factory.initial"
value="org.jboss.as.naming.InitialContextFactory" />
      <Property name="java.naming.factory.url.pkgs"
value="org.jboss.naming:org.jnp.interfaces" />
    </JndiProperties>
    <MessageMetadata>
      <Property name="command" value="generateCachedTables"/>
    </MessageMetadata>
    <ReconnectWaitSec>60</ReconnectWaitSec>
    <!--Decimation type="dm:SynchronizedDecimation">
    </Decimation-->
  </Channel>

  <!-- All channels need to be configured like this to actually work -->
  <InterestModel>
    <InterestModelId>*</InterestModelId>
    <Channels>
      <Channel>AlertServiceChannel</Channel>
      <Channel>TableGenerationChannel</Channel>
    </Channels>
  </InterestModel>

</Policy>
```

16.4.8 Interest Model

Finally, we define Interest Model to bind rules and alert templates. Interest Model may have one of the following types: CHANGE_BASED or STATE_BASED. This type affects the output of an alert template in the following ways:

- Change-based alerts by default will only show data that is marked dirty, i.e. has changed since the previous alert output
- State-based alerts by default will show all data in every alert output

The template attribute "dirtyOnly" provides a way to override the change/state-based character of the interest model.

The Interest Model for the example is defined below:

- Name: LowFlyingAircraftModel
- Enabled: true

- AlertType: STATE_BASED
- Alert Templates: LowAltitudeFlightAlertTemplate (select)
- Rules: LowAltitudeRules (select)
- Workflow Id: Default-Service-Workflow

16.5 GENERATING ALERTS

From NiFi UI, start PutHTTP and PostHTTP processor to ingest flight data and stop them. If everything goes right, alerts should be generated at the URL: http://<intellex_web>:20580/alert/latest?name=LowAltitudeFlights and look like the following.

```
▼<as:ResultSet xmlns:as="urn:org.cubrc.intellex.alert">
  ▼<as:Alert id="961fadbf-1fb8-412f-91e7-e5ace2b86d36">
     <as:Name>LowAltitudeFlights</as:Name>
     <as:InterestModelID>LowFlyingAircraftModel</as:InterestModelID>
     <as:Source>http://aws2-node5.us-gov-west-1.compute.internal</as:Source>
    ▼<as:Destination>
       <as:Channel>NONE</as:Channel>
     </as:Destination>
     <as:GenerationTime>2018-05-31T17:25:01.000-04:00</as:GenerationTime>
     <as:Type/>
     <as:Description/>
    ▼<as:Payload>
     ▼<Flights>
      ▼<Flight id="A54D87">
         <Operator>BANK OF UTAH TRUSTEE - SALT LAKE CITY, UT</Operator>
         <Country>United States</Country>
         <Origination/>
         <Destination/>
         <Altitude>1575</Altitude>
       </Flight>
      ▼<Flight id="A8BBBC">
         <Operator>American Airlines</Operator>
         <Country>United States</Country>
        ▼<Origination>
           KDTW Detroit Metropolitan Wayne County, United States
         </Origination>
         <Destination>KSJC Norman Y. Mineta San Jose, United States</Destination>
         <Altitude>1300</Altitude>
       </Flight>
      ▼<Flight id="A52898">
         <Operator>United Airlines</Operator>
         <Country>United States</Country>
         <Origination>MMCZ Cozumel, Mexico</Origination>
         <Destination>KORD Chicago O'Hare, United States</Destination>
         <Altitude>1275</Altitude>
       </Flight>
      ▼<Flight id="A54D6C">
         <Operator>BANK OF UTAH TRUSTEE - SALT LAKE CITY, UT</Operator>
         <Country>United States</Country>
         <Origination/>
         <Destination/>
         <Altitude>1550</Altitude>
       </Flight>
```


Figure 117 Low Altitude Flights Alert

The generated alert lists all flights that currently are flying at lower than 2,000 ft.

17. ELK LOGGING

17.1 OVERVIEW

The ELK platform is a complete log analytics solution, built on a combination of three open source tools - Elasticsearch, Logstash, and Kibana. ELK utilizes the open source stack of Elasticsearch for deep search and data analytics; Logstash for centralized logging management, which includes shipping and forwarding the logs from multiple servers, log enrichment, and parsing; and finally, Kibana for powerful and beautiful data visualizations. In this section we will mainly cover Kibana, the visualization engine of ELK and the beats that are used in ACS.

In ELK, beats are lightweight data shippers that transport data from different sources such as applications, machines, or networks. It is lightweight because it has a small installation footprint, use limited system resources, and have no runtime dependencies. Beats are installed on servers to capture all sorts of operations data which include logs, metrics, network packet data, etc. The Beats sends operational data to Elasticsearch, either directly or via Logstash, so it can be visualized with Kibana.

The Beats that are currently installed on ACS are the following:

- **Filebeat** is a log data shipper for local files. Installed as an agent on your servers, Filebeat monitors the log directories or specific log files, tails the files, and forwards them to either Elasticsearch or Logstash for indexing.
- **Topbeat** monitors system data and metrics for system processes on the network. It collects data on CPU usage, memory, process statistics, and other system-related metrics that when shipped into the ELK Stack for indexing an analysis, can be used for real-time monitoring of the infrastructure.
- **Packetbeat** sniffs packets over the network wire for specific protocols such as MySQL and HTTP. It grabs all the fundamental metrics that will be used to monitor the protocol in question. For example, in the case of HTTP, the beats will get the request/response, wrap it into a document, and index it into Elasticsearch

17.2 CONFIGURING BEATS

Filebeat, Topbeat, and Packetbeat are installed by default on an ACS installation. The beats on ACS are already pre-configured to log information from certain log files, monitor certain system processes, etc. The basic configuration of beats are given below.

17.2.1 Configuring Filebeat

To configure Filebeat, log onto the ACS node where additional log monitoring is needed and edit the configuration file located at /etc/filebeat/filebeat.yml. The configuration file will consists of two key items: the prospectors and the output. The prospector section is a list of logs to be monitored and the location on the system where Filebeat can find these logs. The other section is the output section which states where Filebeat is going send each log message to and in ACS it is configured to send to the Logstash node on port 5044. This section does not need to be modified unless additional Logstash nodes are available for load balancing.

The current Filebeat configuration for ACS log monitoring is quite simple. An example is given below:

Code Block 1 Filebeat Configuration Sample

```
filebeat:
  prospectors:
    -
      paths:
        - "/var/log/audit/audit.log"
      document_type: "audit"


output:
  logstash:
  hosts : ["acs-node11:5044"]
```

In this configuration, the prospector is scanning the audit log located in the /var/log/audit path on the node it resides on. The document_type is a short descriptor of the types of messages for the given path being monitored. The document_type will later be displayed as _type when the message log is displayed in Kibana, and it will also be used by Logstash filter configuration to choose the right parsing filter if parsing is to be done for the log.

To add additional log files to be monitored by Filebeat just add the additional locations to the prospector section of the filebeat.yml file.

- Add additional paths to files for Filebeat to monitor in the prospector section of filebeat configuration file.

Code Block 2 Filebeat Configuration Additions

```
filebeat:
  prospectors:
    -
      paths:
        - "/var/log/audit/audit.log"
      document_type: "audit"
  #ADD ADDITIONAL FILES TO BE MONITORED HERE
    -
      paths:
        - "/opt/app1/*.log"
      document_type: "app1_logs"
```

Be sure to stop the Filebeat service before editing the configuration file and restart the service after any edits has been made. The Filebeat service is named filebeat.service.

17.2.1.1 Optional Configuration

There are many extra options that can be used in the prospector section of the filebeat configuration file. A whole list of filebeat prospector configuration options can be found at Elastic's Filebeat Option Configuration page. A couple of prospector configuration options of interest that can be of use during log monitoring is listed below.

- exclude_lines:
  This prospector option is used by Filebeat to exclude any lines in the log file that match a regular expression in the list. By default, no lines in the logs are dropped.
  The example given below configures Filebeat to exclude any lines that start with "DBG".

Code Block 3 Prospector exclude_lines

```
filebeat:
  prospectors:
  - paths:
      - /var/log/myapp/*.log
    exclude_lines: ['^DBG']
```

- include_lines:
  This prospector option is used by Filebeat to include any lines in the log file that match a regular expression in the list. By default, all lines in the logs are exported.
  The example given below configures Filebeat to export any lines that start with "ERR" or "WARN".

Code Block 4 Prospector include_lines

```
filebeat:
  prospectors:
  - paths:
      - /var/log/myapp/*.log
    include_lines: ['^ERR', '^WARN']
```

- multiline:
  There are times when multiline log events are generated and the multiline option is used to handle the multiline log events before sending the event to Logstash. This prospector option is used by Filebeat to deal with log message that span multiple lines.
  The example given below configures Filebeat to handle a multiline message where the first line of the message begins with a bracket ([).

Code Block 5 Prospector multiline

```
multiline.pattern: '^\['
multiline.negate: true
multiline.match: after
```

With these multiline option, Filebeat takes all lines that do not start with a [and combines them with the previous lines that does start with a [. With the configuration above, the example below can be used to join the lines of the multiline message into a single event.

Code Block 6 Prospector multiline sample log

```
[beat-logstash-some-name-832-2015.11.28] IndexNotFoundException[no
such index]
    at
org.elasticsearch.cluster.metadata.IndexNameExpressionResolver$Wildc
ardExpressionResolver.resolve(IndexNameExpressionResolver.java:566)
    at
org.elasticsearch.cluster.metadata.IndexNameExpressionResolver.concr
eteIndices(IndexNameExpressionResolver.java:133)
    at
org.elasticsearch.cluster.metadata.IndexNameExpressionResolver.concr
eteIndices(IndexNameExpressionResolver.java:77)
    at
org.elasticsearch.action.admin.indices.delete.TransportDeleteIndexAc
tion.checkBlock(TransportDeleteIndexAction.java:75)
```

The settings for multiline that controls how Filebeat combines the lines in the log messages are the following:

- pattern: specifies the regular expression pattern to match.
- negate: defines whether the pattern is negated. The default is false.
- match: specifies how Filebeat combines matching lines into an event. The setting are after or before.
- max_lines: the maximum number of liens that can be combines into on event. If the multiline message contains more than max_line, any additional lines are discarded. The default is 500.
- timeout: after the specified timeout, Filebeat sends that multiline event even if no new pattern is found to start a new event. The default is 5s.

These are only a couple of Filebeat prospector options that can be used. There are a lot more option available and they are documented on the Filebeat Prospector Options page on the Elastic site. Also another good page to reference is the Regular Expression Support page which provides documentation regular expressions used for pattern matching.

17.2.2 Configuring Topbeat

The configuration file for Topbeat is located at /etc/topbeat/topbeat.yml on each ACS node. Topbeat on ACS is currently configured to use the predefined default values.

Code Block 7 Topbeat default values

```
input:
  period: 10
  procs: [".*"]
```

To configure Topbeat, set the following options under the input section to specify how often Topbeat reads statistics and specify which process it monitors.

- period: defines how often to read system-wide and per-process statistics from the servers. The interval is in seconds and by default it is set to 10 seconds.
- procs: defines a list of regular expressions to match all the processes that needs to be monitored. By default, all the running processes are monitored.
  If you are not interested in collecting per-process statistics, so the following can be used:

```
input:
  stats:
    process: false
```

By default, ACS is set to send output to the Logstash node on port 5044. If the output of Topbeat needs to be routed to Elasticsearch, set the IP address and port where Topbeat can find the Elasticsearch installation:

Code Block 8 Topbeat Elasticsearch configuration

```
# Configure what outputs to use when sending the data collected by the beat.
# Multiple outputs may be used.
output:
  ### Elasticsearch as output
  elasticsearch:
    # Array of hosts to connect to.
     hosts: ["192.168.1.11:9200"]
```

Be sure to stop the Topbeat service before editing the configuration file and restart the service after any edits has been made. The Topbeat service is named topbeat.service.

17.2.3 Configuring Packetbeat

The configuration file for Packetbeat is located at /etc/packetbeat/packetbeat.yml on each ACS node. Packetbeat on ACS is currently configured to use the predefined default values.

To configure Packetbeat, set the following:

- Select the network interface from which to capture the traffic. Packetbeat support capturing all messages sent or received by the server on which it is installed on. For this, use any as the device:

Code Block 9 Packetbeat interfaces configuration

```
interfaces:
  device: any
```

- In the protocol section, configure the ports on which Packetbeat can find each protocol. If any non-standard ports are used, add them in this section. Otherwise, the default values should do fine.

Code Block 10 Packetbeat protocol configuration

```
protocols:
- type: dns
  ports: [53]

  include_authorities: true
  include_additionals: true

- type: http
  ports: [80, 8080, 8081, 5000, 8002]

- type: memcache
  ports: [11211]
...
- type: mongodb
  ports: [27017]
```

- By default, ACS is set to send output to the Logstash node on port 5044. If the output of Packetbeat needs to be routed to Elasticsearch, set the IP address and port where Topbeat can find the Elasticsearch installation:

Code Block 11 Packetbeat Elasticsearch configuration

```
output:
  elasticsearch:
    hosts: ["192.168.1.11:9200"]
```

Be sure to stop the Packetbeat service before editing the configuration file and restart the service after any edits has been made. The Packetbeat service is named packetbeat.service.

These are some basic option used to configure Packetbeat. A more extensive list of configuration options can be found at Elastic's Configuring Packetbeat page.

17.3 KIBANA

Kibana is an analytics and visualization platform that builds on Elasticsearch to give you a better understanding of the data logged. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster. Kibana's web frontend will be where all the user interface action will happen. Below each of Kibana's core component will briefly be explained.

Kibana's user interface can be found at https://[elk_server]:5643/app/kibana#/

17.3.1 User Interface

17.3.1.1 Discover Section

Kibana's Discover section is the first place you will go once you have indexed some data in Elasticsearch.Error rendering macro 'scroll-office-title' : com.atlassian.renderer.v2.macro.MacroException: Parameter named 'title' is missing

Kibana screenshot above shows the Discover section of the page. The page includes the following sections.

1. The index pattern selected. The filebeat-* index is shown in the diagram but other indexes such has topbeat and packetbeat are available from the dropdown.
2. The fields that exist within your index. Clicking on the filed name, provides a current view of the top values in that field. For example, to get a view of the field breakdown you can click on the _type field in the available fields section.Error rendering macro 'scroll-office-title' : com.atlassian.renderer.v2.macro.MacroException: Parameter named 'title' is missing
3. The number of event time in the form of a data histogram. The period is set with the time picket in the top-right corner of the page.
4. The indexed document list. The indexed document can be expanded and a detailed description of the document would be displayed.Error rendering macro 'scroll-office-title' : com.atlassian.renderer.v2.macro.MacroException: Parameter named 'title' is missing

17.3.1.2 Visualize Section

Kibana's Visualize section is the place to build visualizations on top of the data you have indexed through Elasticsearch. Dashboards can be built to display related visualizations. A variety of visualization tools including area charts, data tables, line charts, and more are available to build help build your visualizations. This section also includes a list of saved visualizations.Error rendering macro 'scroll-office-title' : com.atlassian.renderer.v2.macro.MacroException: Parameter named 'title' is missing

17.3.1.3 Dashboard Section

Kibana's Dashboard section is the place where a collection of saved visualizations are displayed. The dashboards can be edited so visualizations can be arranged and resized as needed and the modified dashboard can be saved so they can be reloaded and shared at a later time.Error rendering macro 'scroll-office-title' : com.atlassian.renderer.v2.macro.MacroException: Parameter named 'title' is missing

17.3.2 How to View Infrastructure Metrics with Topbeat

Topbeat, which is one of the several "Beats" data shipper that helps send various types of server data to an Elasticsearch instance, allows you to gather information about the CPU, memory, and process activity on your servers. In this section, we will show you how to use the ELK stack to gather and visualize infrastructure metrics using Topbeat. It is assumed that Topbeat has been installed and configured properly on each client server it is running on.

1. First connect to **Kibana**. In the web browser, enter the URL: https://[elk_server]:5643/app/kibana#/ and if it loaded correctly you should see the Kibana Discover page.

2. Select the **topbeat-*** from the Index Patterns menu (left side) to view the Topbeat data in the Discover view. Here you can search and drill down on various Topbeat entries

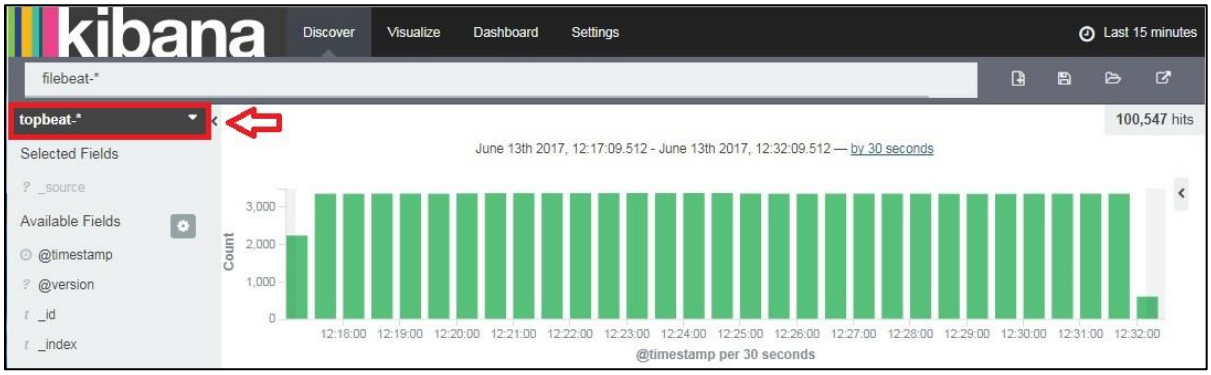


Figure 118 Kibana Discover Page

3. To load the Topbeat dashboard, click on **Dashboard** to get to the Dashboard page.

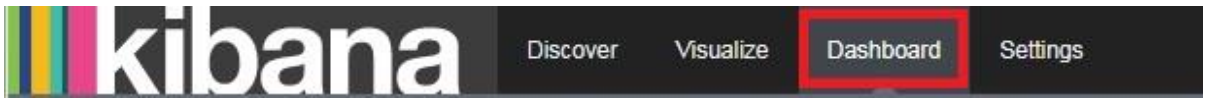


Figure 119 Kibana Dashboard

4. Next, to load the Topbeat dashboard click the **Load Saved Dashboard** icon on the upper right side of the Kibana page.

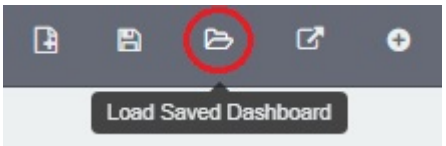


Figure 120 Load Saved Dashboard

5. Now, navigate to the second page of the saved dashboards then click on **Topbeat-Dashboard.**

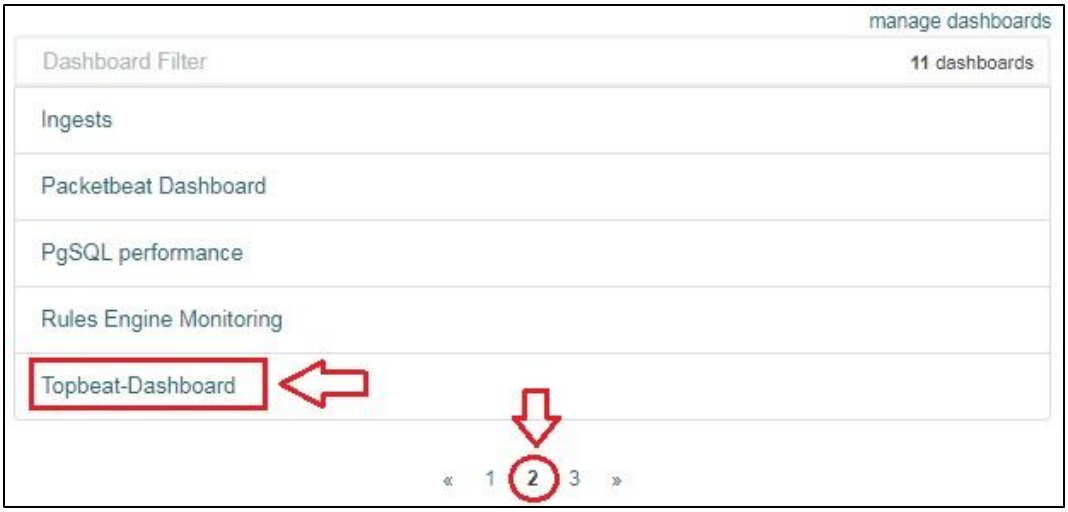


Figure 121 Topbeat Dashboard

6. Once the Topbeat-Dashboard is loaded, you will see a variety of metrics that were gathered from your client servers that you installed Topbeat on.

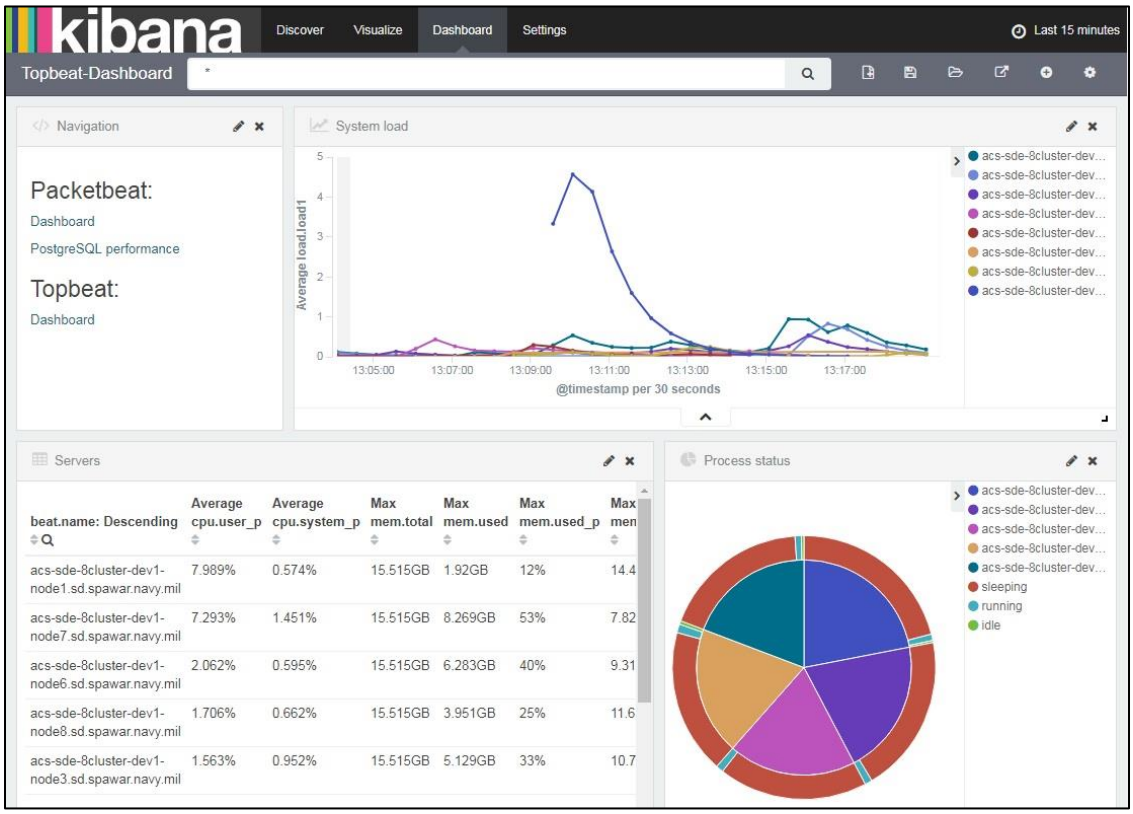


Figure 122 Topbeat Metrics

17.3.3 How to View Network Analytics with Packetbeat

Packetbeat is a data shipper and analyzer for network packets that is integrated into the ELK Stack. Packetbeat provides real-time monitoring metrics on web, database, and other network

protocols by monitoring the actual packets being transferred across the wire. With Packetbeat a user-friendly dashboard with useful statistics can be used to monitor data packets to detect unusual levels of network traffic, identify packet sources and destinations, and more. In this section, we will show you how to access the Packetbeat dashboard. It is assumed that Packetbeat has been installed and configured properly.

1. First connect to **Kibana**. In the web browser, enter the URL: https://[elk_server]:5643/app/kibana#/ and if it loaded correctly you should see the Kibana Discover page
2. Select the **packetbeat-*** from the Index Patterns menu (left side) to view the Packetbeat data in the Discover view. Here you can search and drill down on various Packetbeat entries.

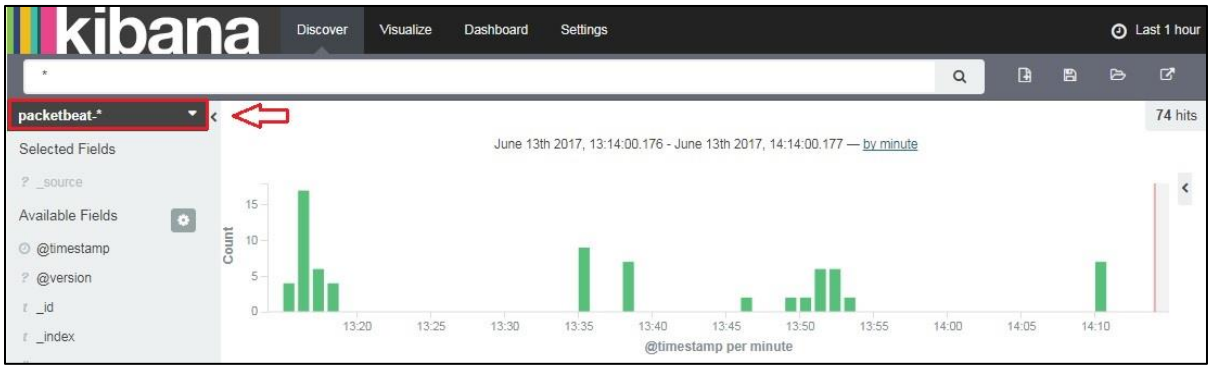


Figure 123 Kibana Discover Page

3. To load the Packetbeat dashboard, click on **Dashboard** to get to the Dashboard page.

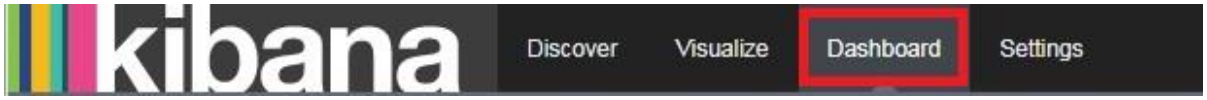


Figure 124 Kibana Dashboard

4. Next, to load the Packetbeat dashboard click the **Load Saved Dashboard** icon on the upper right side of the Kibana page

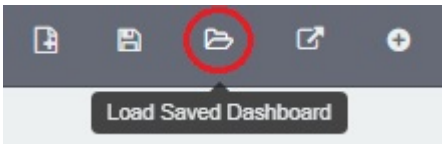


Figure 125 Load Saved Dashboard

5. Now, navigate to the second page of the saved dashboards then click on **Packetbeat Dashboard.**

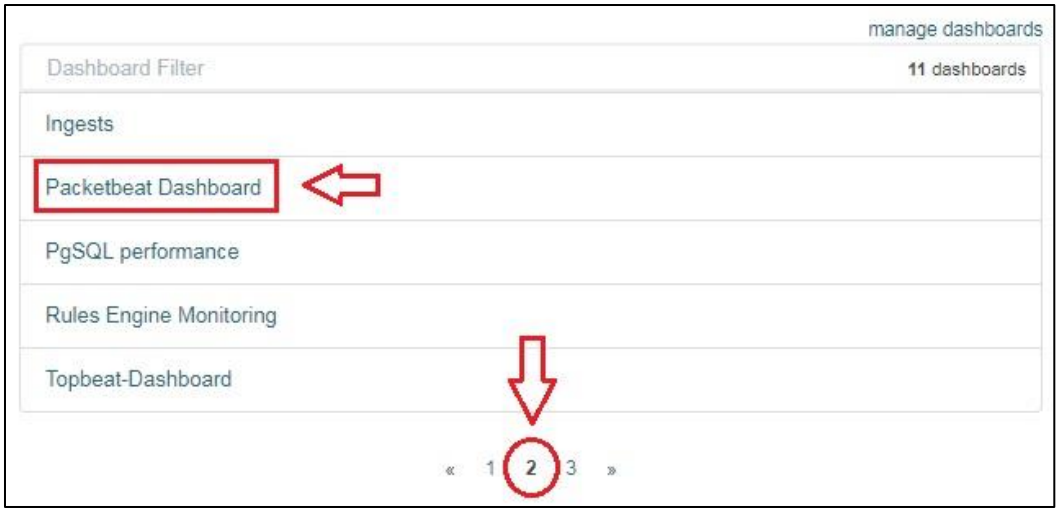


Figure 126 Load the Packetbeat dashboard

6. Once the Packetbeat Dashboard is loaded, you will see a variety of metrics that were gathered from your client servers that you installed Packetbeat on.

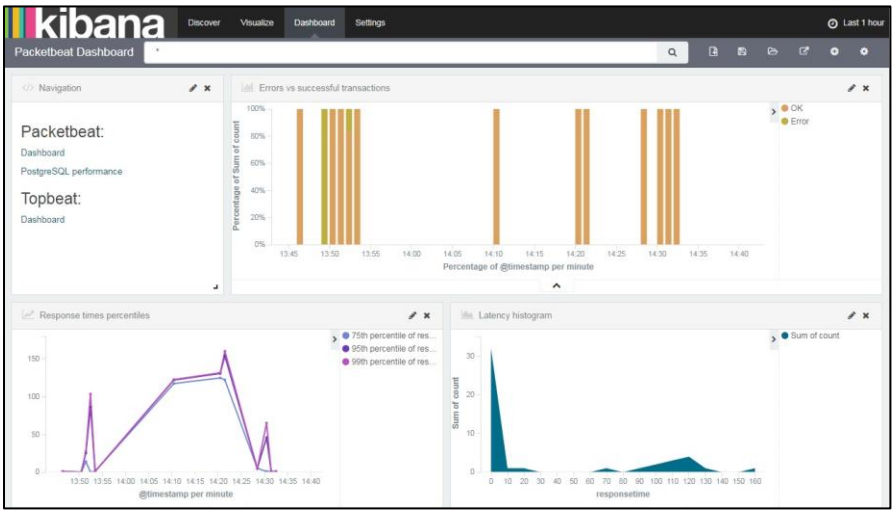


Figure 127 Packetbeat Metrics

17.3.4 How to View Log Data with Filebeat

Filebeat is a log data shipper for local files. Filebeat monitors the log directories or specific log files, tails the files, and forwards them either to Logstash for parsing or directly to Elasticsearch indexing. We will be using Kibana to search and visualize the logs. It is assumed that Filebeat has been installed and configured properly.

1. First connect to **Kibana**. In the web browser, enter the URL: https://[elk_server]:5643/app/kibana#/ and if it loaded correctly you should see the Kibana Discover page.
2. Select the **filebeat-*** from the Index Patterns menu (left side) to view the Packetbeat data in the Discover view. Here you can search and drill down on various filebeat entries

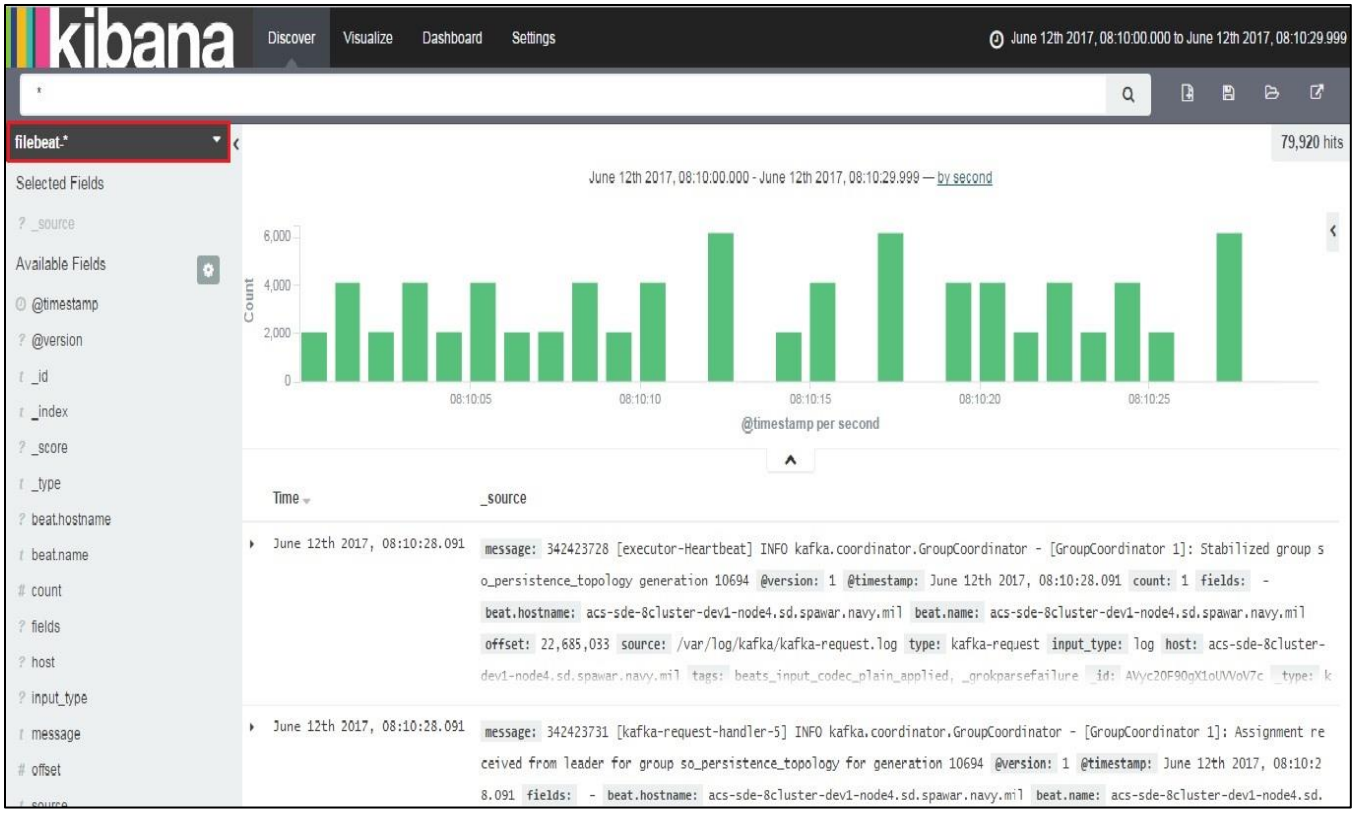


Figure 128 Filebeat Logs

3. Since Filebeat does not currently have a Dashboard page, the only viewable filebeat entries are located on the Discover page. With the filebeat-* selected, all the log entries from all the log files listed in the prospector section of the filebeat.yml file will be displayed. These log entries displayed on Kibana are from all the nodes where the filebeat service is running. Sample log information is shown below. Here log information including message, hostname, type, timestamp, and more are displayed for the entries captured by Filebeat.

Time	_source
May 16th 2018, 13:29:02.080	message: 2018-05-16 20:28:59,033 [myid:1] - INFO [Thread-4579:NIOServerCnxn$StatCommand@655] - Stat command output @version: 1 @timestamp: May 16th 2018, 13:29:02.080 type: zookeeper count: 1 fields: - source: /var/log/zookeeper/zookeeper.log offset: 101,510,866 input_type: log beat.hostname: acs-dev1-node1.sd.spawar.navy.mil beat.name: acs-dev1-node1.sd.spawar.navy.mil host: acs-dev1-node1.sd.spawar.navy.mil tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWNqpFSYegjKhfmGkGdK _type: zookeeper _index: filebeat-2018.05.16 _score:
May 16th 2018, 13:29:02.080	message: 2018-05-16 20:28:59,032 [myid:1] - INFO [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:NIOServerCnxn@821] - Processing stat command from /128.49.69.39:59784 @version: 1 @timestamp: May 16th 2018, 13:29:02.080 input_type: log fields: - beat.hostname: acs-dev1-node1.sd.spawar.navy.mil beat.name: acs-dev1-node1.sd.spawar.navy.mil offset: 101,510,712 type: zookeeper source: /var/log/zookeeper/zookeeper.log count: 1 host: acs-dev1-node1.sd.spawar.navy.mil tags: beats_input_codec_plain_applied, _grokparsefailure _id: AWNqpFSYegjKhfmGkGdJ _type: zookeeper _index: file

Figure 129 Sample Log Information

4. To see a more detailed view of the log entry, click on the triangle besides the timestamp next to the log entries to expand the entry.

Table JSON Link to /filebeat-2018.05.16/zookeeper/AWNqpFSYegjKhfmGkGdK

@timestamp	May 16th 2018, 13:29:02.080
@version	1
_id	AWNqpFSYegjKhfmGkGdK
_index	filebeat-2018.05.16
_score	
_type	zookeeper
beat.hostname	acs-dev1-node1.sd.spawar.navy.mil
beat.name	acs-dev1-node1.sd.spawar.navy.mil
count	1
fields	-
host	acs-dev1-node1.sd.spawar.navy.mil
input_type	log
message	2018-05-16 20:28:59,033 [myid:1] - INFO [Thread-4579:NIOServerCnxn$StatCommand@655] - Stat command output
offset	101,510,866
source	/var/log/zookeeper/zookeeper.log
tags	beats_input_codec_plain_applied, _grokparsefailure
type	zookeeper

Figure 130 Expanded Log Entries

A possible use-case for using Kibana to analyze the log entries captured by Filebeat is to query for failed login attempts for a specific node. In ACS, Filebeat is already configured to monitor the /etc/audit/audit.log on all nodes which captures system events including login information. Since audit information is being captured by Filebeat all entries are going to be displayed in

Kibana. From here, we can use the search capabilities in Kibana to filter out all log entries beside the node of interest and filter out all message types beside the message type of interest.

For example, if we are interested in viewing the log entries captured by Filebeat for the host acs-dev1-node10.sd.spawar.navy.mil. In the Kibana search bar the following can be entered:

Code Block 12 Kibana Filebeat Search Example

```
beat.hostname: acs-dev1-node10.sd.spawar.navy.mil
```

acs-dev1-node10.sd.spawar.navy.mil	type=CRED_DISP msg=audit(1526571001.125:34710): pid=24786 uid=0 auid=0 ses=3761 msg='op=PAM:setcred grantors=pam_rootok acct="root" exe="/usr/bin/su" hostname=? addr=? terminal=? res=success'	audit
acs-dev1-node10.sd.spawar.navy.mil	type=CRED_ACQ msg=audit(1526571001.030:34697): pid=24769 uid=0 auid=4294967295 ses=4294967295 msg='op=PAM:setcred grantors=pam_env,pam_localuser,pam_unix acct="root" exe="/usr/sbin/crond" hostname=? addr=? terminal=cron res=success'	audit
acs-dev1-node10.sd.spawar.navy.mil	May 17 15:30:01 acs-dev1-node10 systemd: Started Session 3761 of user root.	message
acs-dev1-node10.sd.spawar.navy.mil	May 17 15:30:01 acs-dev1-node10 systemd: Created slice User Slice of root.	message

Figure 131 Log Entries Captured by Filebeat

To filter out all message types besides the message type of audit for that node the following can be entered:

```
beat.hostname: acs-dev1-node10.sd.spawar.navy.mil AND _type: audit
```

acs-dev1-node10.sd.spawar.navy.mil	type=USER_ACCT msg=audit(1526571001.028:34693): pid=24768 uid=0 auid=4294967295 ses=4294967295 msg='op=PAM:accounting grantors=pam_access,pam_unix,pam_localuser acct="root" exe="/usr/sbin/crond" hostname=? addr=? terminal=cron res=success'	audit
acs-dev1-node10.sd.spawar.navy.mil	type=LOGIN msg=audit(1526571001.029:34695): pid=24768 uid=0 old-auid=4294967295 auid=0 tty=(none) old-ses=4294967295 ses=3760 res=1	audit
acs-dev1-node10.sd.spawar.navy.mil	type=CRED_REFR msg=audit(1526571001.056:34702): pid=24768 uid=0 auid=0 ses=3760 msg='op=PAM:setcred grantors=pam_env,pam_localuser,pam_unix acct="root" exe="/usr/sbin/crond" hostname=? addr=? terminal=cron res=success'	audit
acs-dev1-node10.sd.spawar.navy.mil	type=USER_AUTH msg=audit(1526571001.105:34705): pid=24786 uid=0 auid=0 ses=3761 msg='op=PAM:authentication grantors=pam_rootok acct="root" exe="/usr/bin/su" hostname=? addr=? terminal=? res=success'	audit

Figure 132 Filtered Audit Log Entries

And to filter out all the audit messages besides user login and failed login attempt audit messages the following can be entered:

```
beat.hostname: acs-dev1-node10.sd.spawar.navy.mil AND _type: audit AND message:
user_login AND message: failed
```

```
acs-dev1-node10.sd.spawar.navy.mil    type=USER_LOGIN msg=audit(1526570066.330:34685): pid=23972 uid=0 auid=4294967295 ses=4294967295    audit
msg='op=login acct="root" exe="/usr/sbin/sshd" hostname=? addr=128.49.66.169 terminal=ssh
res=failed'
```

Figure 133 Failed User Login Attempts

The above screen captures of the Kibana queries had the beatname, message, and message type displayed. The fields being displayed in Kibana are fully configurable by the users. To add or remove fields to be displayed in Kibana, select the interested fields on the left side of the Kibana as shown below.

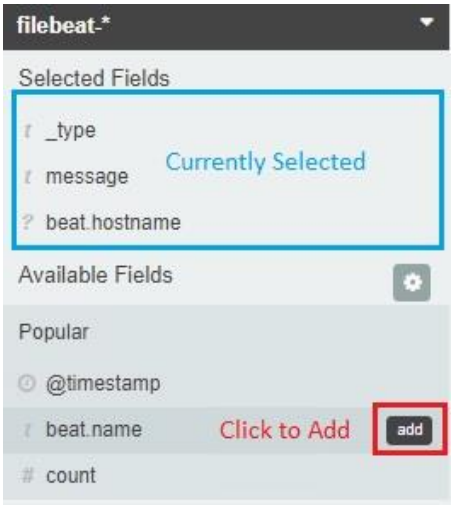


Figure 134 Add or Remove Fields

18. ORGANIZING WEB APPLICATIONS WITH OWF

In this section, we will explore how to use Ozone Widget Framework (OWF) to organize and display Web applications in a single browser window. OWF consists of an environment and a set of tools for this purpose.

18.1 OVERVIEW

OWF App Components (previously known as OWF Widgets) are lightweight Web application wrapped with a metadata definition that provide a description to the framework of how the component should load. The app component metadata definition contains a number of fields including a URL, name, height and width properties.

OWF provides a suite of APIs that enable the developer to extend their Web application through the use of inter-widget communication, user preferences and internationalization. Each API is written in JavaScript so that app components can be built in a large variety of Web technologies.

OWF supports Internet Explorer 7 and higher and Firefox 17 and higher and Chrome. OWF is tested against the following browsers:

Table 13 OWF - Tested Browsers

Browsers	Versions
Internet Explorer	7 & 9
Firefox	17
Chrome	33

18.2 OWF IN ACS

OWF is deployed in ACS as **owf.war** folder which contains both HTML and JavaScript files. These files make up the OWF user interface and the server backend which handles persistence of the user, user preferences, widget definition and dashboards.

Users must be authenticated by the system before they can access OWF services. The user certificates must be installed in a user's browser. These certificates can be found under **/etc/certs/client** on the basebox.

OWF can be launched by navigating to `https://<owf_web>:8443/owf`.

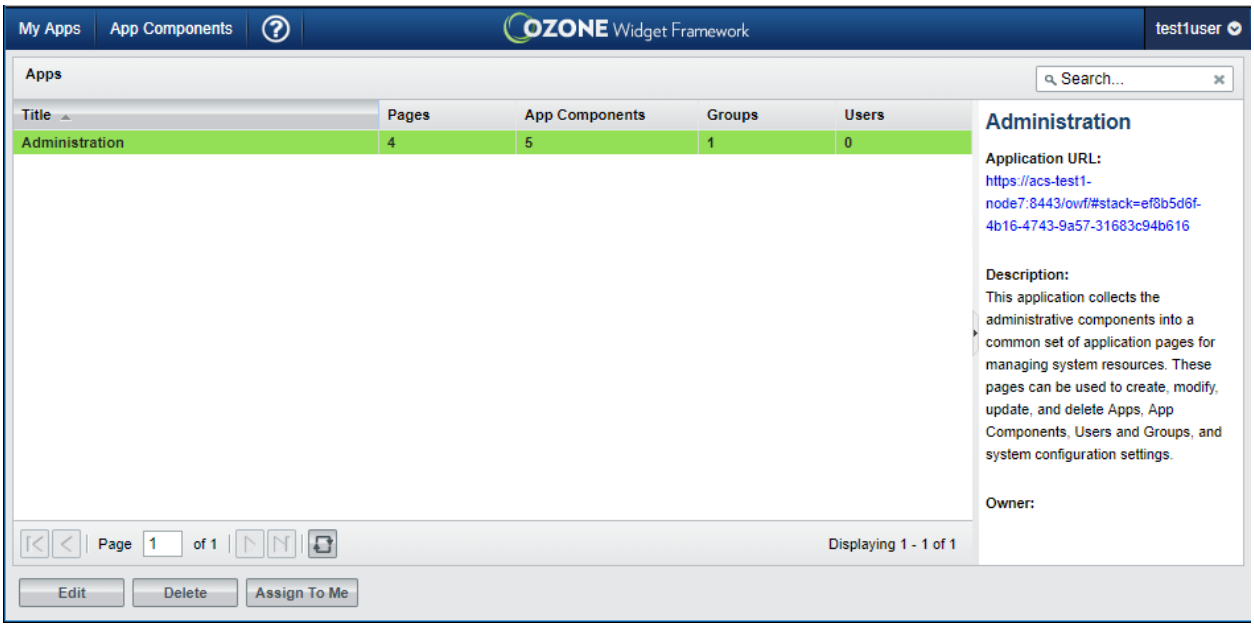


Figure 135 OWF - Initial Screen

18.3 OWF UI COMPONENTS

18.3.1 OWF Toolbar

Figure 136 OWF - Initial Screen

The following buttons are found on the toolbar:

- **My Apps:** Opens the My Apps Menu, used to find, start and manage apps. Use the menu to get details about an app and app pages.
- **App Components:** The App Components window displays all available app components. Use this feature to start or add components to an app.
- **Online Help**: Repository of instructional guides and video tutorials.

18.3.2 Dropdown User Menu

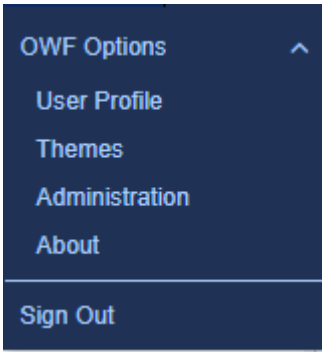


Figure 137 OWF Dropdown User Menu

Access the drop-down User Menu by clicking on the username in the upper-right of the page. The following elements reside in the OWF User Menu:

- **User Profile –** User identification including the username, full name, e-mail and list of groups of which the user is a member. Under the User Preferences heading, the user can also enable OWF animations by clicking the "Enable animations" checkbox.
- **Themes** – Opens the Theme Settings window.
- **About –** Displays customizable information about the product.
- **Sign out –** Used to sign out of the framework in accordance with the security protocols that were used for the initial sign-in.

18.4 OWF APP COMPONENTS

An app (app) component is a lightweight, single-purpose app that offers a summary or limited view of a larger app. In OWF, an app component is a global description for a piece of Web content that can be configured by the user and displayed within an app. OWF app component was previously referred to as widget.

18.4.1 App Component Types

18.4.1.1 Singleton App Components

Singleton app components allow only one instance of the app component to open in an app. (Users can open multiple instances of regular app components on each app.) If a Singleton app component is open on an app and a user tries to open another instance of the app component, the open instance will move to the forefront of the screen. Administrators may make an app component a Singleton for numerous reasons. For example, preventing users from opening multiple instances per app may reduce confusion, increase performance (if the app component uses a substantial amount of memory), or address another need.

18.4.1.2 Background App Components

Background app components run but do not appear on a user's app. They often serve as caching and logging tools that do not have a user interface. Background app components can be obtained from a Store server or configured by an OWF administrator. Most users will not be aware that Background app components are running in their instance of OWF.

18.4.2 App Component Menu

Users access their app components from the App Components Menu by clicking the (

) button in the toolbar. Once open, the App Components Menu appears as a carousel.

Figure 138 OWF App Components Carousel

The App Components Menu displays the app component's title and icon. For more information about the app component, hover over the app component icon to reveal the Information link. Clicking the opens the Information Window which provides a description of the app component and a Remove link.

18.4.3 Create an App Component

18.4.3.1 Link to Another Web App

To create an app component that opens another Web app, follow the following steps:

- Click on "My Apps" button on OWF Toolbar to bring up My Apps. Click Administration then App Components button to open App Component Manager

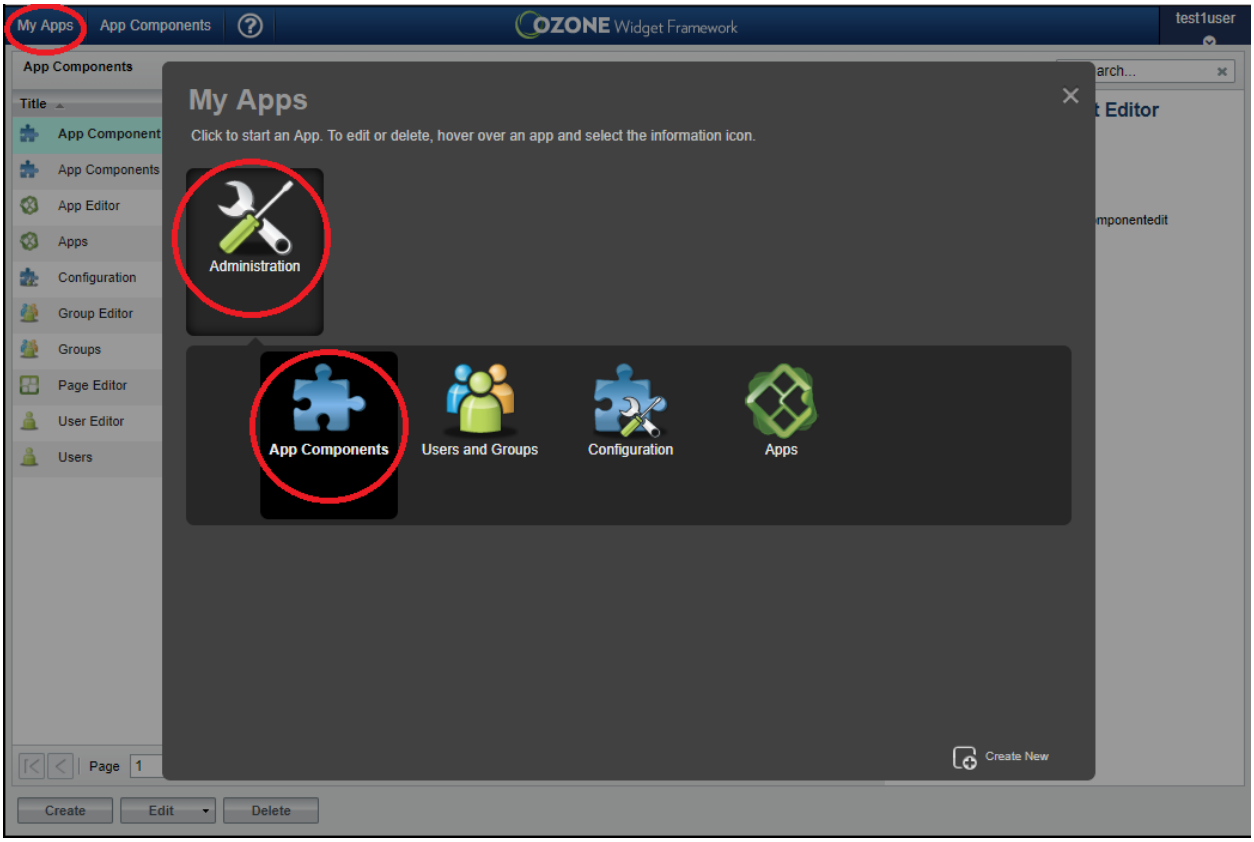


Figure 139 OWF My Apps

- Click Create button to open App Component Editor then click on 'Don't have a descriptor URL?' link to enter the App Component description

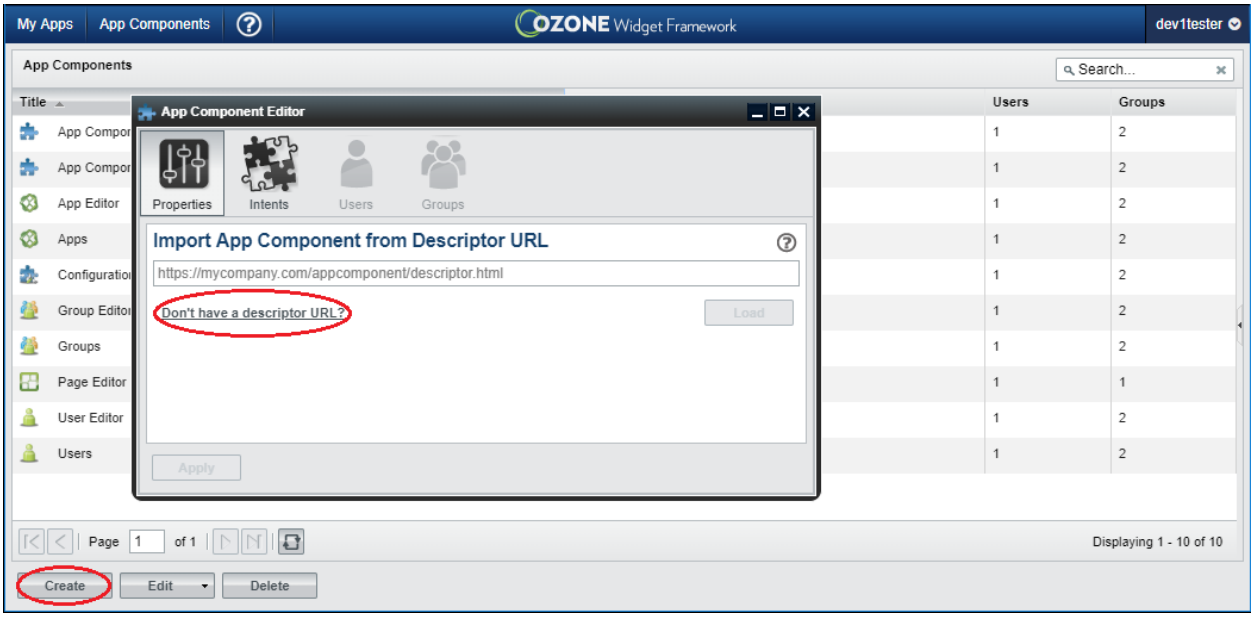


Figure 140 OWF App Components Manager

- As an example, let populate the fields as follows to create an App Component to IntellEx Config UI then click Apply button:
  - Name: IntellEx Config
  - URL: https://acs-dev1-node5:8443/IntellexConfigUI/
  - Small Icon URL: themes/common/images/adm-tools/Stacks24.png
  - Medium Icon URL: themes/common/images/adm-tools/Stacks64.png

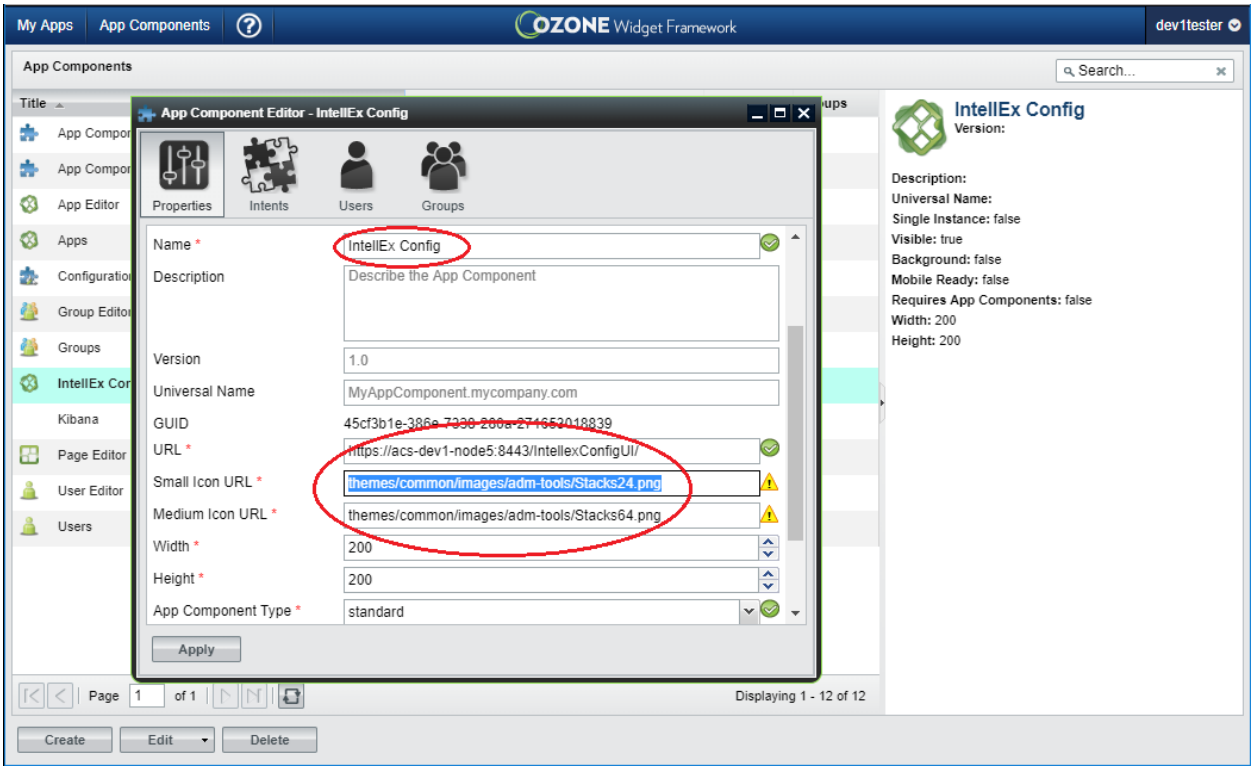


Figure 141 OWF App Components Editor

- From the App Component Editor, click User button to add the new App Component to a user. Similarly, click Groups button to add the new App Component to a group.

At this point, the new App Component, namely IntellEx Config has been created. It should be listed on the App Component Manager; however, the browser page needs to be refreshed in order to load the new App Component into the App Component menu.

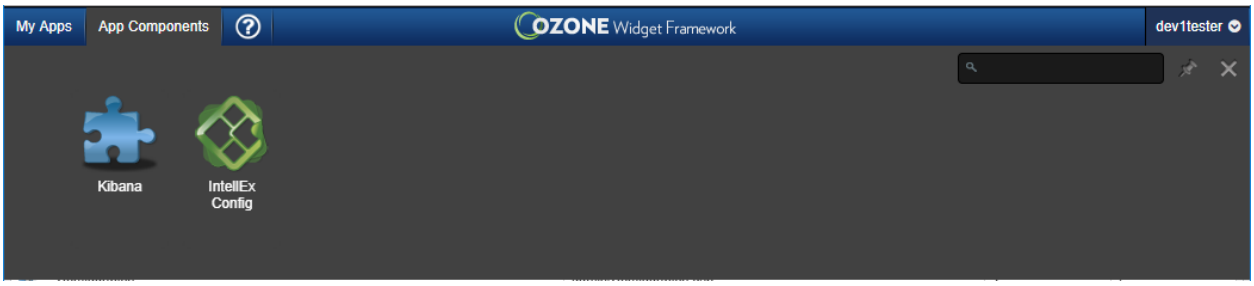


Figure 142 OWF App Components Carousel

18.5 OWF APP

An OWF app is a screen where a user can dictate (for the most part) which application components to load, which layouts to use and the arrangement of the app components within the specified layouts.

Each time a saved application loads, the screen and app component layout maintain the same look and feel as the last time the application was accessed by the user. Users can receive applications by the following methods:

- Create their own
- Add from the Store
- Assign to individual user by an administrator
- Assign to a group by an administrator

Group assigned applications provide identical applications for each member of a group. Each group member can customize their instance of a pre-configured application. Applications that have not been created by the user can be restored to their default states.

18.5.1 My Apps Menu

As seen in the previous sections, My Apps menu lists all of the user's applications. Applications included here are OZONE Apps, either created in OWF or obtained from the Store, and Web apps. From the My Apps Menu, users can start an application, create new ones or search for applications in the Store. Applications can be shared, restored, edited or deleted through the manager buttons in the My Apps Menu.

To open the My Apps menu, click **My Apps** in the OZONE Toolbar.

18.5.2 Opening an App

Applications can contain one or more layouts called Pages. If an application has only one page, then clicking its icon in the My Apps Menu will start it.

To open an application:

- Click the **My Apps** button in the toolbar to open the My Apps Menu.
- Click an application, it will automatically open.

If the application has more than one page, then clicking the application’s icon will open a carousel displaying all the application’s pages.

18.5.3 Create an App

OWF includes an App Builder where user can design and customize apps.

To create a new app:

- Click My Apps button in the OWF Toolbar to open My Apps menu, then click Create New at the bottom of the screen to open the Create New App window

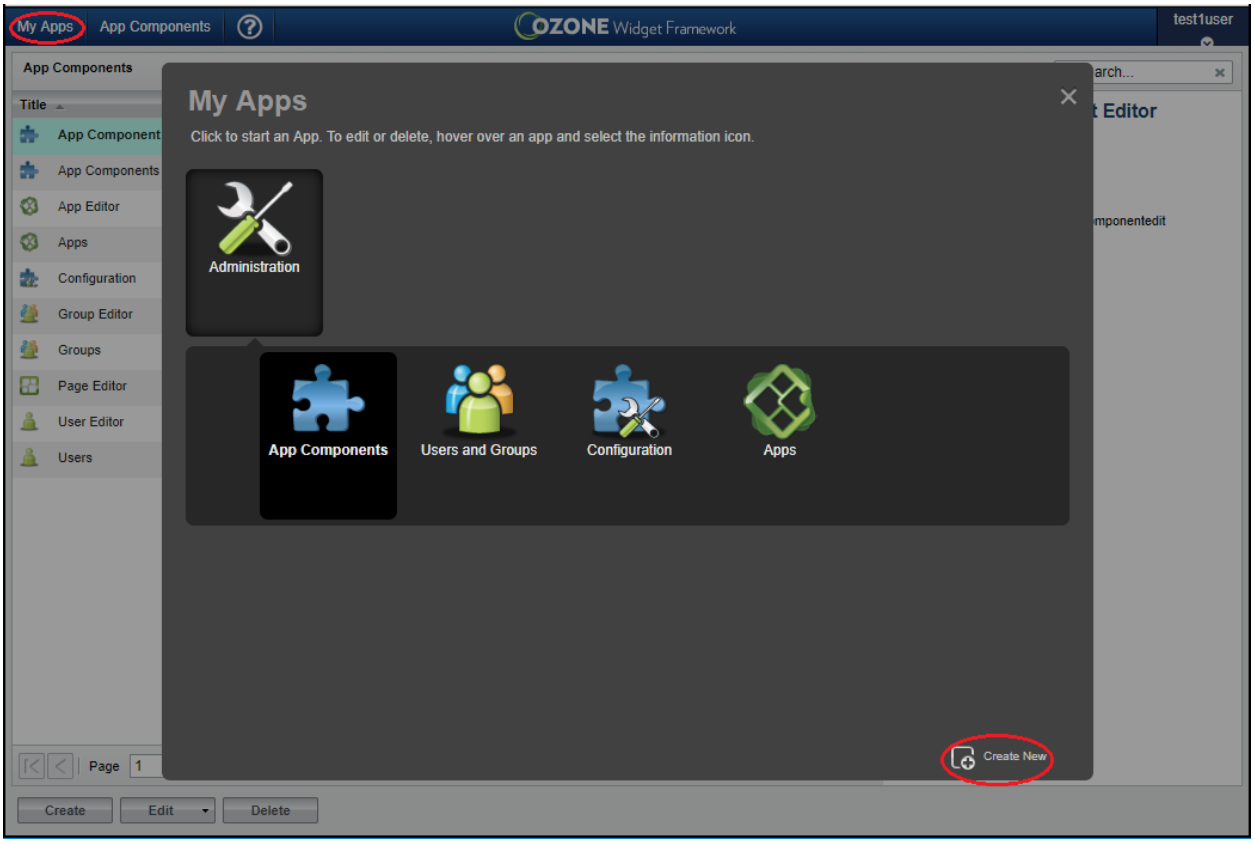


Figure 143 OWF Create New App

- Give the app a title (required), an image URL (optional) and a description (optional), choose a layout for the app, then click OK

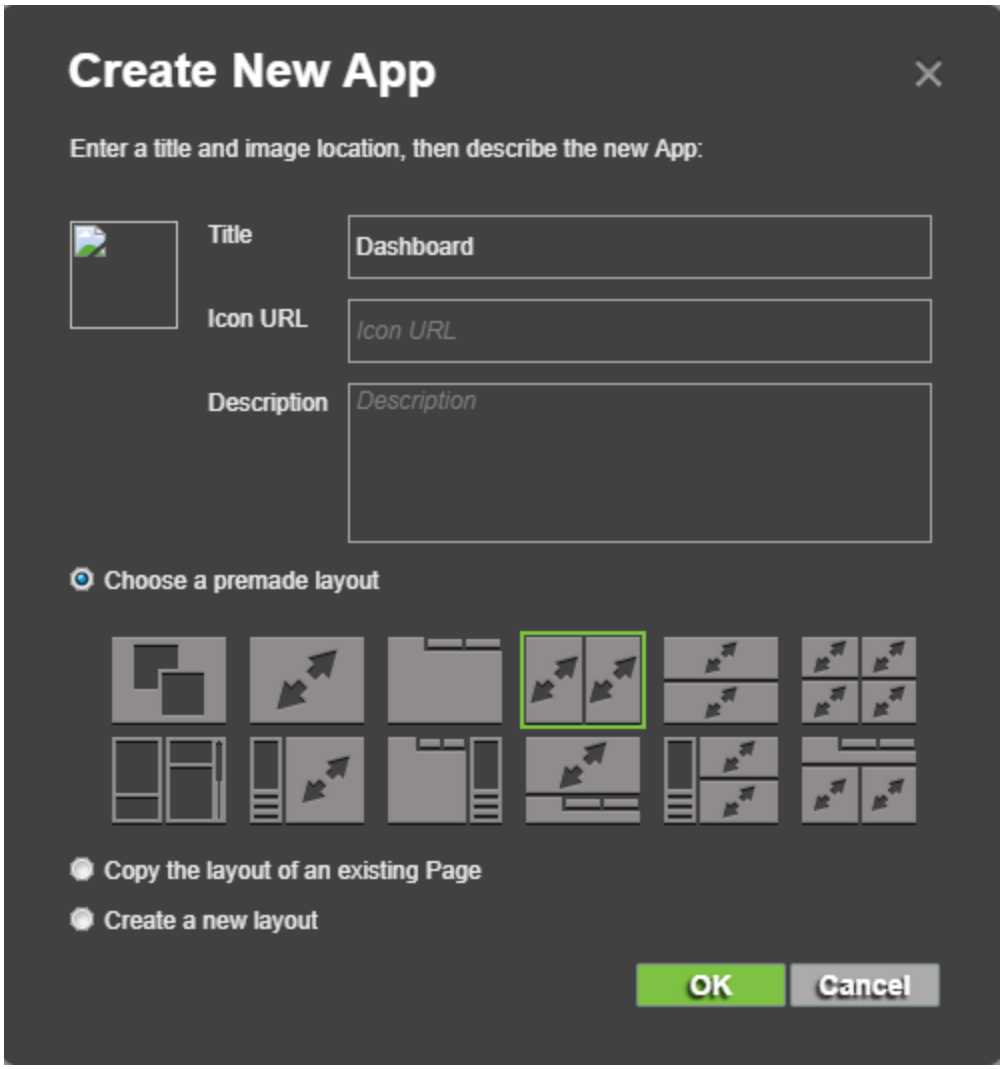


Figure 144 OWF Create New App

The app will automatically open unless user selected "Create a new layout."

Now, user can add App Components to the app by clicking the App Component button on the Toolbar to show all the App Components and select components by clicking them.

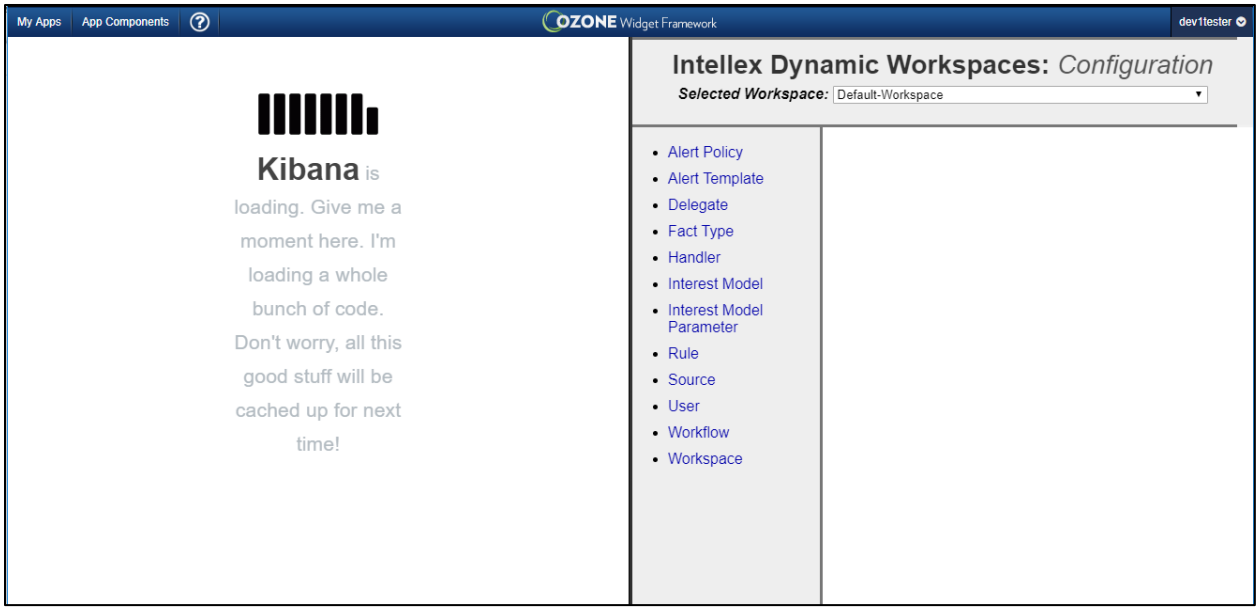


Figure 145 OWF Create New App

Next time the My Apps menu is opened, it will contain this app.

19. OPENSHIFT

19.1 ARCHITECTURE

OpenShift by default is configured with 1 master/etcd/registry server, and 3 nodes available for pods. This is a non HA cluster, so a failing master will bring the cluster down. In the future, a multi-master system will be available.

19.2 MANAGING USERS

19.2.1 Adding a User

After new users log in to OpenShift Enterprise, an account is created for that user per the identity provider configured on the master. The cluster administrator can manage the access level of each user.

19.2.2 Viewing User and Identity Lists

OpenShift Enterprise user configuration is stored in several locations within OpenShift Enterprise. Regardless of the identity provider, OpenShift Enterprise internally stores details like role-based access control (RBAC) information and group membership. To completely remove user information, this data must be removed in addition to the user account.

In OpenShift Enterprise, two object types contain user data outside the identification provider: `user` and `identity`.

To get the current list of users:

```
$ oc get user
NAME      UID                                     FULL NAME   IDENTITIES
demo     75e4b80c-dbf1-11e5-8dc6-0e81e52cc949               htpasswd_auth:demo
```

To get the current list of identities:

```
$ oc get identity
NAME                  IDP NAME        IDP USER NAME   USER NAME   USER UID
htpasswd_auth:demo    htpasswd_auth   demo            demo        75e4b80c-dbf1-
11e5-8dc6-0e81e52cc949
```

Note the matching UID between the two object types. If you attempt to change the authentication provider after starting to use OpenShift Enterprise, the user names that overlap will not work because of the entries in the identity list, which will still point to the old authentication method.

19.2.3 Managing User and Group Labels

Labels are identifying metadata consisting of key/value pairs attached to resources. Labels are used to add identifying attributes to objects that are relevant to users and can be used to reflect architectural or organization concepts. Labels can be used in conjunction with labels selectors to uniquely identify individual resources or groups of resources.

To add a label to a user or group:

```
$ oc label user/<user_name> <label_name>
```

For example, if the user name is **theuser** and the label is **level=gold**:

```
$ oc label user/theuser level=gold
```

To remove the label:

```
$ oc label user/<user_name> <label_name>-
```

To show labels for a user or group:

```
$ oc describe user/<user_name>
```

19.2.4 Deleting a User

To delete a user:

1. Delete the user record:

   ```
   $ oc delete user demo
   user "demo" deleted
   ```

2. Delete the user identity.

   The identity of the user is related to the identification provider you use. Get the provider name from the user record in `oc get user`.

   In this example, the identity provider name is **htpasswd_auth**. The command is:

   ```
   # oc delete identity htpasswd_auth:demo
   identity "htpasswd_auth:demo" deleted
   ```

   If you skip this step, the user will not be able to log in again.

After you complete these steps, a new account will be created in OpenShift Enterprise when the user logs in again.

If your intention is to prevent the user from being able to log in again (for example, if an employee has left the company and you want to permanently delete the account), you can also remove the user from your authentication back end (like **htpasswd**, **kerberos**, or others) for the configured identity provider.

For example, if you are using **htpasswd**, delete the entry in the ***htpasswd*** file that is configured for OpenShift Enterprise with the user name and password.

For external identification management like Lightweight Directory Access Protocol (LDAP) or Internet Download Manager (IDM), use the user management tools to remove the user entry.

19.3 CREATING AND INSTALLING IMAGES

This document serves to describe a workflow for building a basic image using ansible container. Ansible Container allows us to create an image with little or no modification to our exiting roles. Reusing the roles will still allow us to deploy via OpenShift, VM, or bare metal. The basic workflow for creating an image is as follows:

1. Create an ansible container project
2. modify a pom.xml for use in creating and deploying an rpm of the image
3. deploy the image to a host and load into the OpenShift registry

19.3.1 Installation

1. Install ansible-container

```
$ pip install ansible-container[docker,k8s]
```

2. install docker

```
$ yum install docker-1.13.1
$ systemctl restart docker
$ pip install docker==2.7.0
$ sudo sed -i "s/'run', 'secrets'/'tmp', 'secrets'/g"
/usr/lib/python2.7/site-packages/container/docker/engine.py
```

3. add user 'theuser' to docker

```
$ usermod -a -G docker theuser
```

19.3.2 Creating the project and image rpm

1. Create Ansible base

```
$ create-image-project -p acs-hello-world
```

2. Modify ansible.cfg to point to your roles

```
ex:  roles-path: /opt/tcri/ansible/playbooks/roles
```

3. Build the image

```
$ mvn clean compile
```

4. Create an image to import

```
$ mvn package
```

19.3.3 Ansible Container basics

19.3.3.1 container.yml

- defines the structure and configuration for the image
- the conductor is an image used to orchestrate the building of the deployed image
- Ansible container has the ability to push images to OpenShift, but for our purposes we will be prebuilding images and deploying via an RPM
- the services section is where images are defined that will be built

```
        web:
          from: "centos:7"
          roles:
           - hello
          ports:
            - "80:80"
          command: ["/usr/bin/dumb-init", "/usr/sbin/apache2ctl", "-D",
     "FOREGROUND"]
          dev_overrides:
            environment:
              - "DEBUG=1"
```

- multiple images/services can be defined in this section
- owf-web and owf-db can be defined and run as isolated images but as dependent images. in this scenario the pom will need to be updated to pull all sevices and build tar and rpms

```
     ...
                        <execution>
                           <id>create-tar</id>
                          <phase>package</phase>
```

```
                        <configuration>
                          <target>
                            <exec executable="docker">
                              <arg value="save"/>
                              <arg value="-o"/>
                              <arg
value="target/${project.artifactId}.tar"/>
                              <arg value="${project.artifactId}-web"/>
                            </exec>
                          </target>
                        </configuration>
                        <goals>
                          <goal>run</goal>
                        </goals>
                      </execution>

...


                                <sources>
                                    <source>

<location>target/${project.artifactId}.tar</location>
                                        <excludes>
                                          <exclude>pom.xml</exclude>
                                        </excludes>
                                    </source>
                                </sources>


...


                          <script>
                             docker load -i
/opt/acs/${project.artifactId}/${project.artifactId}.tar
                          </script>
```

- basic commands
  - ansible-container init - Creates the default project structure
  - ansible-container build - Create and image
  - ansible-container run - will run on a local docker

19.3.3.2 Example deployment:

```
---
- hosts: localhost
  tasks:

  - name: create acs project
    shell: oc new-project acs  --description="acs 3" --display-name="acs"
    ignore_errors: true

  - name: delete route
    shell: oc delete route dc/acs-openam
    ignore_errors: true

  - pause: seconds="5"

  - name: build image
    shell: ansible-container build
    args:
      chdir: ../image_builds/openam

  - name: login to openshift
    shell: echo "oc login --insecure-skip-tls-verify=true  -u admin -p <password>
https://{{ groups['masters'][0] }}:8443 "


  - name: get token
    shell: oc whoami -t
    register: token

  - name: push to openshift
    shell: "ansible-container --engine openshift push --push-to openshift --
username admin --password  {{ token.stdout }} --tag latest"
    args:
      chdir: ../image_builds/openam

  - name: create application
    shell: oc new-app --docker-image=docker-registry.default.svc:5000/acs/acs-
openam:latest -n acs -f ../image_builds/openam/openam-template.yaml --allow-
missing-images
    ignore_errors: true

  - name: create edge route
    shell: oc create route edge --service=acs-openam --port=8080 -n acs
    ignore_errors: true
```

19.4 BASIC CLI OPERATIONS

19.4.1 Test the default service and login

The **oc login** command is the best way to initially set up the OpenShift CLI, and it serves as the entry point for most users. The interactive flow establishes a session to an OpenShift server with the provided credntials. The information is automatically saved in a CLI configuration file that is then used for subsequent commands.

The following command logs in and switches to the **default** project.

```
$ oc login -u system:admin -n default
```

Validate Result:

- project list should include the following projects

```
Logged into "https://cisv56.sd.spawar.navy.mil:8443" as "system:admin" using
existing credentials.
You have access to the following projects and can switch between them with 'oc
project <projectname>':
* default
kube-public
kube-service-catalog
kube-system
logging
management-infra
openshift
openshift-ansible-service-broker
openshift-infra
openshift-node
openshift-template-service-broker
openshift-web-console
```

19.4.2 Using project "default"

oc get command return a list of objects for the specified object type. If the optional object name is included in the request, then the list of results is filtered by that value.

```
oc get all
```

Validate Result:

- ensure docker-registry, registry-console, and router are running
- ensure that routers are running for docker-registry, registry-console

```
NAME DOCKER REPO TAGS UPDATED
imagestreams/registry-console docker-registry.default.svc:5000/default/registry-
console v3.7
NAME REVISION DESIRED CURRENT TRIGGERED BY
deploymentconfigs/docker-registry 1 1 1 config
deploymentconfigs/registry-console 1 1 1 config
deploymentconfigs/router 4 1 1 config
NAME HOST/PORT PATH SERVICES PORT TERMINATION WILDCARD
routes/docker-registry docker-registry-default.apps.sd.spawar.navy.mil docker-
registry <all> passthrough None
routes/registry-console registry-console-default.apps.sd.spawar.navy.mil registry-
console <all> passthrough None
NAME READY STATUS RESTARTS AGE
po/docker-registry-1-bpd6t 1/1 Running 0 17d
po/registry-console-1-kn44z 1/1 Running 0 17d
```

```
po/router-4-xbzt8 1/1 Running 0 8d
NAME DESIRED CURRENT READY AGE
rc/docker-registry-1 1 1 1 17d
rc/registry-console-1 1 1 1 17d
rc/router-1 0 0 0 17d
rc/router-2 0 0 0 16d
rc/router-3 0 0 0 8d
rc/router-4 1 1 1 8d
NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE
svc/docker-registry 172.30.74.210 <none> 5000/TCP 17d
svc/kubernetes 172.30.0.1 <none> 443/TCP,53/UDP,53/TCP 17d
svc/registry-console 172.30.242.213 <none> 9000/TCP 17d
svc/router 172.30.123.146 <none> 80/TCP,443/TCP,1936/TCP 17d
```

19.4.3 List Project

oc project command displays the project currently in use. If there are multiple projects, specifying the project name with the **oc project** command switches to the specified project.

```
oc project openshift
oc get all
```

Validate Results

- Ensure OpenShift images are present

```
NAME                                                DOCKER REPO
TAGS                      UPDATED
imagestreams/dotnet                                 docker-
registry.default.svc:5000/openshift/dotnet
1.0,1.1,2.0
imagestreams/dotnet-runtime                         docker-
registry.default.svc:5000/openshift/dotnet-runtime                        2.0
imagestreams/fis-java-openshift
basebox50.sd.spawar.navy.mil:5000/jboss-fuse-6/fis-java-openshift
imagestreams/fis-karaf-openshift
basebox50.sd.spawar.navy.mil:5000/jboss-fuse-6/fis-karaf-openshift
imagestreams/httpd                                  docker-
registry.default.svc:5000/openshift/httpd                                 2.4
imagestreams/jboss-amq-62                           docker-
registry.default.svc:5000/openshift/jboss-amq-62
1.1,1.2,1.3 + 3 more...
imagestreams/jboss-amq-63                           docker-
registry.default.svc:5000/openshift/jboss-amq-63
1.0,1.1,1.2
imagestreams/jboss-datagrid65-client-openshift      docker-
registry.default.svc:5000/openshift/jboss-datagrid65-client-openshift     1.0,1.1
imagestreams/jboss-datagrid65-openshift             docker-
registry.default.svc:5000/openshift/jboss-datagrid65-openshift
1.2,1.3,1.4 + 1 more...
imagestreams/jboss-datagrid71-client-openshift      docker-
registry.default.svc:5000/openshift/jboss-datagrid71-client-openshift     1.0
imagestreams/jboss-datagrid71-openshift             docker-
registry.default.svc:5000/openshift/jboss-datagrid71-openshift            1.0,1.1
imagestreams/jboss-datavirt63-driver-openshift      docker-
```

```
registry.default.svc:5000/openshift/jboss-datavirt63-driver-openshift     1.0,1.1
imagestreams/jboss-datavirt63-openshift            docker-
registry.default.svc:5000/openshift/jboss-datavirt63-openshift
1.1,1.2,1.3 + 1 more...
imagestreams/jboss-decisionserver62-openshift      docker-
registry.default.svc:5000/openshift/jboss-decisionserver62-openshift      1.2
imagestreams/jboss-decisionserver63-openshift      docker-
registry.default.svc:5000/openshift/jboss-decisionserver63-openshift      1.3,1.4
imagestreams/jboss-decisionserver64-openshift      docker-
registry.default.svc:5000/openshift/jboss-decisionserver64-openshift      1.0,1.1
imagestreams/jboss-eap64-openshift                 docker-
registry.default.svc:5000/openshift/jboss-eap64-openshift
1.2,1.3,1.4 + 3 more...
imagestreams/jboss-eap70-openshift                 docker-
registry.default.svc:5000/openshift/jboss-eap70-openshift
1.3,1.4,1.5 + 1 more...
imagestreams/jboss-eap71-openshift                 docker-
registry.default.svc:5000/openshift/jboss-eap71-openshift                  1.0-TP
imagestreams/jboss-processserver63-openshift       docker-
registry.default.svc:5000/openshift/jboss-processserver63-openshift       1.3,1.4
imagestreams/jboss-processserver64-openshift       docker-
registry.default.svc:5000/openshift/jboss-processserver64-openshift       1.0,1.1
imagestreams/jboss-webserver30-tomcat7-openshift   docker-
registry.default.svc:5000/openshift/jboss-webserver30-tomcat7-openshift
1.2,1.3,1.1
imagestreams/jboss-webserver30-tomcat8-openshift   docker-
registry.default.svc:5000/openshift/jboss-webserver30-tomcat8-openshift
1.1,1.2,1.3
imagestreams/jboss-webserver31-tomcat7-openshift   docker-
registry.default.svc:5000/openshift/jboss-webserver31-tomcat7-openshift   1.0,1.1
imagestreams/jboss-webserver31-tomcat8-openshift   docker-
registry.default.svc:5000/openshift/jboss-webserver31-tomcat8-openshift   1.0,1.1
imagestreams/jenkins                               docker-
registry.default.svc:5000/openshift/jenkins                                1,2
imagestreams/mariadb                               docker-
registry.default.svc:5000/openshift/mariadb                                10.1
imagestreams/mongodb                               docker-
registry.default.svc:5000/openshift/mongodb
3.2,2.4,2.6
imagestreams/mysql                                 docker-
registry.default.svc:5000/openshift/mysql
5.5,5.6,5.7
imagestreams/nodejs                                docker-
registry.default.svc:5000/openshift/nodejs                                 6,0.10,4
imagestreams/perl                                  docker-
registry.default.svc:5000/openshift/perl
5.16,5.20,5.24
imagestreams/php                                   docker-
registry.default.svc:5000/openshift/php
5.5,5.6,7.0
imagestreams/postgresql                            docker-
registry.default.svc:5000/openshift/postgresql
9.2,9.4,9.5
imagestreams/python                                docker-
registry.default.svc:5000/openshift/python
2.7,3.3,3.4 + 1 more...
imagestreams/redhat-openjdk18-openshift            docker-
registry.default.svc:5000/openshift/redhat-openjdk18-openshift
1.0,1.1,1.2
imagestreams/redhat-sso70-openshift                docker-
registry.default.svc:5000/openshift/redhat-sso70-openshift                 1.3,1.4
```

```
imagestreams/redhat-sso71-openshift                docker-
registry.default.svc:5000/openshift/redhat-sso71-openshift
1.0,1.1,1.2
imagestreams/redis                                 docker-
registry.default.svc:5000/openshift/redis                                    3.2
imagestreams/ruby                                  docker-
registry.default.svc:5000/openshift/ruby
```

19.4.4 Validate user can login

```
$ oc login -u <user name> -p <user password>
```

```
Login successful.
```

19.4.5 Validate docker login

```
oc login -u admin -p spawarinstall
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

    acsdevelopertestproject
  * default
    kube-public
    kube-service-catalog
    kube-system
    logging
    management-infra
    openshift
    openshift-ansible-service-broker
    openshift-infra
    openshift-node
    openshift-template-service-broker
    openshift-web-console

Using project "default".
[root@cisv56 ~]# oc whoami -t
wsy0Xt5pd4jdbOogEoCmJTxki6Mct8YSCU1P1SMiz3A
[root@cisv56 ~]# docker login -u admin -p
wsy0Xt5pd4jdbOogEoCmJTxki6Mct8YSCU1P1SMiz3A
Login Succeeded
[root@cisv56 ~]#
```

19.4.6 Create New Project

```
$ oc new-project acsdevelopertestproject
```

Result: **This needs to be extended to an actual test app

```
Now using project "acsdevelopertestproject" on server
"https://cisv56.sd.spawar.navy.mil:8443".

You can add applications to this project with the 'new-app' command. For example,
try:

    oc new-app centos/ruby-22-centos7~https://github.com/openshift/ruby-ex.git

to build a new example application in Ruby.
```

19.5 WEB UI

OpenShift Web UI can be accessed at URL: https://<openshift_master>:8443. The login page is followed:

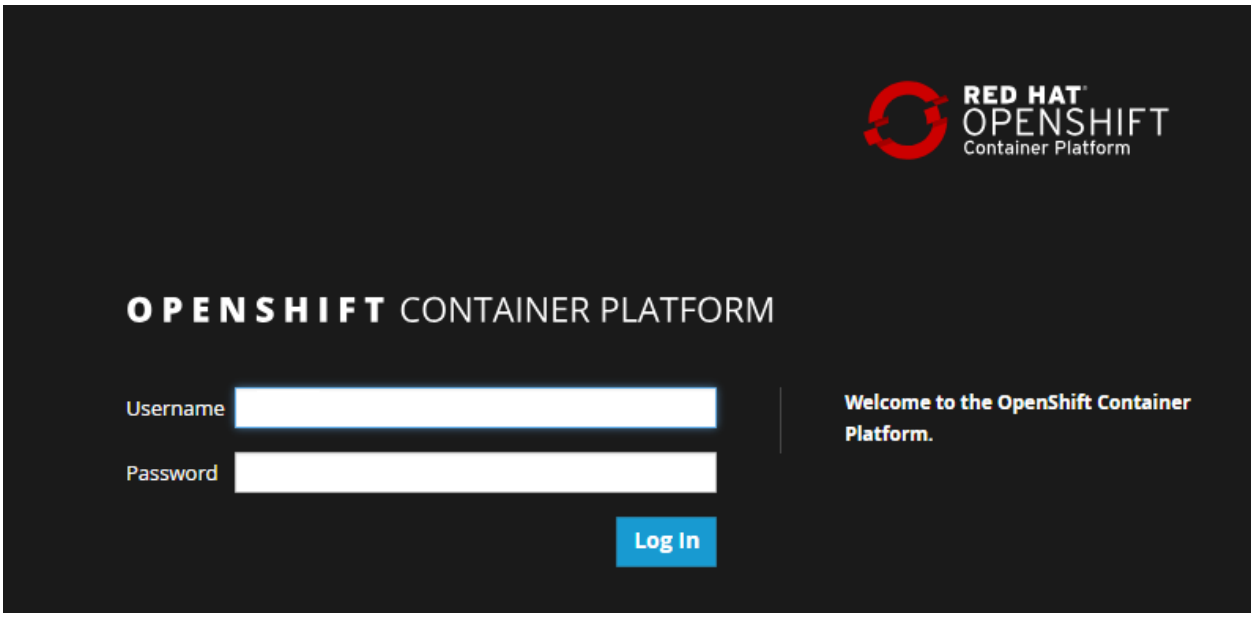


Figure 146 OpenShift Logon

Once, the login credentials are provided, user can navigate the web console.

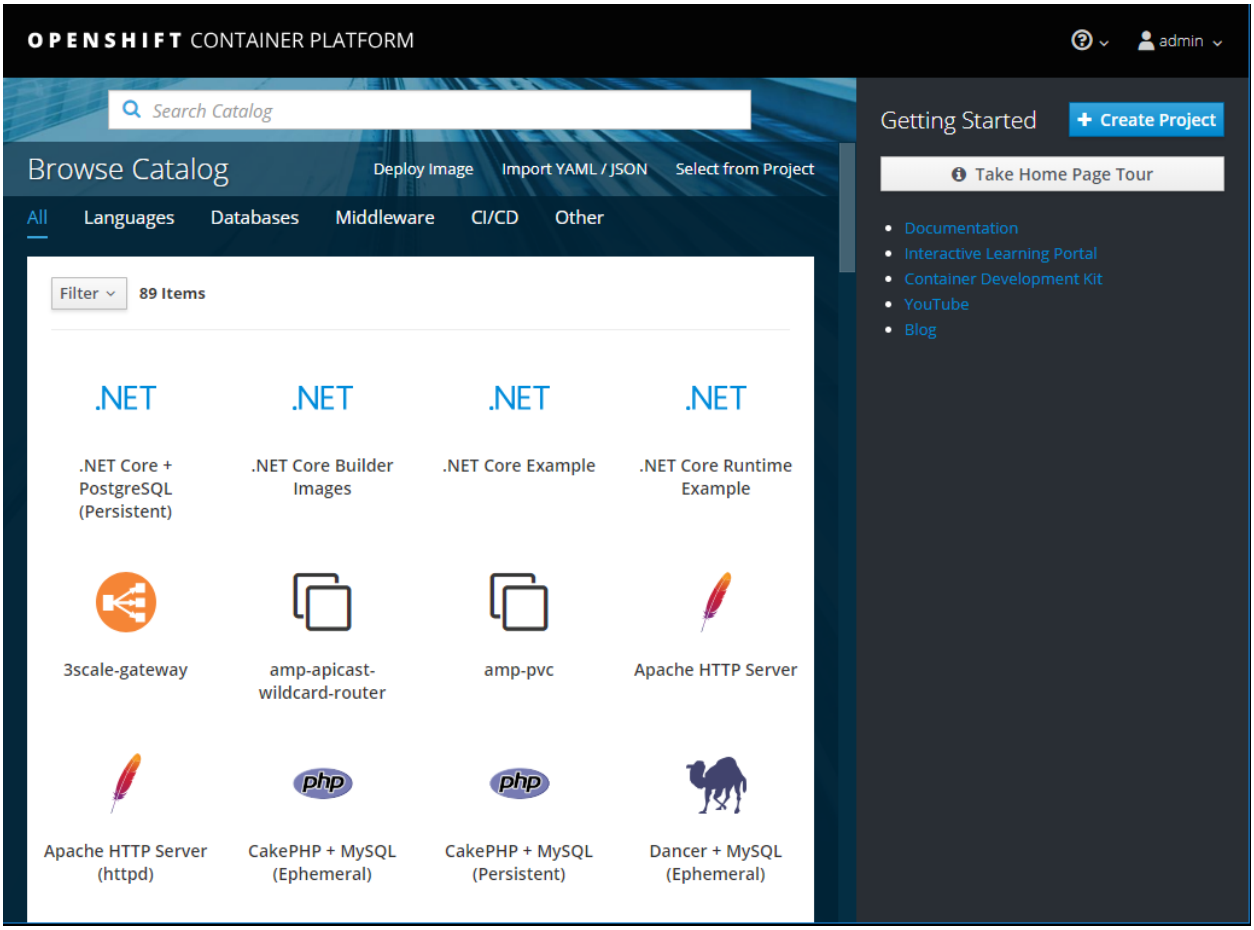


Figure 147 OpenShift Project

19.6 HELLO WORLD!

The following steps create a Hello World app.

SSH to an OpenShift node, master or client, then log in to the OpenShift server:

```
$ oc login -u <user name> -p <password>
```

Create a new project, named **test**, for the user:

```
$ oc new-project test
```

This allows for containers to run as root:

```
$ oc adm policy add-scc-to-user anyuid -z default
```

This will pull the image from the docker registry and deploy into a pod:

```
$ oc new-app --docker-image=private-registry:5000/httpd:latest --insecure-
registry=true --name=httpd
```

Open the app to the world:

```
$ oc expose svc/httpd
```

Look for the project name which is the url to access the GUI:

```
$ oc status
```

```
In project apache on server https://node12.dev12.acs.sd.spawar.navy.mil:8443

http://httpd-apache.apps.dev12.acs.sd.spawar.navy.mil to pod port 80-tcp (svc/httpd)
  dc/httpd deploys istag/httpd:latest
    deployment #1 deployed about an hour ago - 1 pod

View details with 'oc describe <resource>/<name>' or list everything with 'oc get all'.
[root@node15 ~]#
```

Access the URL:

Figure 148 OpenShift Project

Delete the project **test** when done:

```
$ oc delete project test
```

20. APP DEVELOPMENT BEST PRACTICES

20.1 TWELVE-FACTOR APPS

The twelve-factor app is a methodology for building Software as a Service (SaaS) apps (also called *web apps*) suitable for deployment on a modern cloud platform, minimizing divergence between development and production to enable continuous deployment. These can scale up without significant changes to tooling, architecture, or development practices. This methodology is not dependent on the choice of programming language, the database, etc., for the app.

This section summarizes the twelve factors at a high level in the subsections below; readers who wish to learn more about this topic should visit web sites such as https://12factor.net/ for additional details.

20.1.1 Codebase

A version control system always tracks a twelve-factor app, such as Git. A copy of the revision-tracking database is called the code repository (repo). A codebase is any single repo or any set of repos who share a root commit.

Each app will have just a single codebase, but there will be many deploys of the app. A deploy is a running instance of the app.

The codebase will be the same across all deploys, but each deploy could have a different version active. For example, a developer might have the latest app version running in their local environment, but staging will only have the latest version to pass through all of the automated testing while production many be several revisions behind while it undergoes user acceptance testing.

20.1.2 Dependencies

A twelve-factor app declares all dependencies via a dependency declaration manifest instead of assuming that it can rely on system-wide packages. A dependency isolation tool prevents implicit dependencies from “leaking in” from the surrounding system.

One major benefit of not assuming that system tools will be available is that it prevents the app from failing if moved to a system without the assumed tools; by explicitly declaring the version of each tool used, it prevents compatibility issues with the different versions of each tool that may be present on any system.

Another major benefit is that new developers can immediately begin work on an app because they can simply check out the app’s codebase and setup the environment with a build command, instead of having to install the proper version of each tool themselves.

20.1.3 Config

Twelve-factor apps are required to separate their config from their code. This is essential because the config can have significant variation across deploys, while the code does not. This is because the config contains all of the information that varies between deploys.

Examples of config values include credentials to external services (Aegis Weapon System (AWS), Consolidated Afloat Networks and Enterprise Services (CANES), etc.) and resource handles to the backing services.

Twelve-factor apps store the config in environment variables, which are easy to change between deploys without modifying any code. These environment variables are language and OS agnostic and are not checked into the code repo.

20.1.4 Backing Services

A backing service is any service that the app consumes over the network as part of its normal operation. Example backing services include databases, messaging/queueing systems, and caching systems. Backing services are locally managed, or provided and managed by third parties such as Amazon.

Systems administrators who manage the app's runtime deploy traditionally also manage the local backing services.

Twelve-factor app code does *not* distinguish between local and third party services; they are simply resources available to the app. This enables a deploy to switch backing services such as going from a local database to a third party database without any code changes, only the resource handle in the config would be changed.

Being able to attach and detach resources at will can have significant advantages — for example, if an app's database is failing due to a hardware issue, the administrator can switch the app to a different database instantiation without any code change.

20.1.5 Build, Release, Run

The twelve-factor app uses strict separation between the build, release, and run stages. The codebase goes through these three stages for each deploy beyond the development stage. The build stage converts the code repo into an executable bundle called a build. The build stage gathers vendors' dependencies and compiles binaries and assets for the code version for the specified commit in the deployment process.

The release stage combines the build created by the build stage with the deploy's config. The release is ready for immediate execution in the execution environment.

The run stage runs the app in the execution environment. Every release should have a unique release identifier; once created, a release cannot be mutated. This enables deployment tools to easily rollback to any previous release if needed.

20.1.6 Processes

A twelve-factor app executes as one or more stateless processes. Twelve-factor processes are stateless and share-nothing. All data that needs to persist must be stored in a backing service (normally a database), since no container data will persist when the container is destroyed and regenerated from the base image.

The app should never assume that anything cached in memory or on disk is available on a future request or job, since the local state (memory and file system) can easily be changed or wiped.

It is important that the app and database are separated because there will be no persistent storage (outside of a database separate from the app) in the PaaS.

Note that web apps commonly rely on cached data in memory of the app's process, assuming that future requests from the same visitor route to the same process. These "sticky sessions" are a violation of twelve-factor and should never be used or relied upon.

Best practice for developers is to use a datastore like Redis for session state data.

20.1.7 Port Binding

The twelve-factor app is completely self-contained and exports Hypertext Transfer Protocol (HTTP) as a service by binding to a port and listening to requests coming in on that port. This is unlike how some web apps execute inside a webserver container (such as a Java app running inside Tomcat). A developer will typically visit an URL with localhost in their local development environment to interact with the service exported by their app. In deployment, the routing layer handles routing requests from a public-facing hostname to the port-bound web processes.

Nearly any kind of server software can be exported by port binding — port binding is not limited to HTTP. Developers should note that this port-binding approach means that one app can be a backing service for another app.

20.1.8 Concurrency

Unlike PHP and Java processes where the running processes are only minimally visible to app developers and concurrency is managed internally via threads, developers have control over twelve-factor processes. The developer is able to architect an app to handle diverse workloads by assigning process types.

Individual VMs have a maximum vertical scaling, so applications must be able to span multiple processes on multiple physical machines for efficient scaling. The twelve-factor process model scales efficiently because its share-nothing, horizontally partitionable nature means that adding more concurrency is a simple and reliable operation.

Twelve-factor developers should rely on the OS's process manager and should never daemon-ize or write Process Identification (PID) number files.

20.1.9 Disposability

Twelve-factor app processes can start or stop at a moment's notice. This disposability provides fast elastic scaling, rapid deployment of code or config changes, and robustness of production deploys. Processes should minimize startup time to provide more agility for the release process, scaling, and robustness benefits. Processes should shut down gracefully, refusing all new requests but allowing any current requests to finish before exiting.

Developers should ensure that their processes are robust against unexpected, non-graceful terminations. One way this can happen is during a sudden hardware failure.

A best practice approach is to use a robust queueing back end that returns jobs to the queue when clients disconnect or time out.

20.1.10 Development/Production Parity

Traditionally, legacy applications had significant differences between development (when developers make changes to the code in their local dev environment) and production (when end

users can access the app). In addition to significant time and tool differences between development and production, ops engineers deploy the production code while developers deploy their code locally.

It is important for the continuous deployment of twelve-factor apps to minimize the gap between development and production. Software pipelines that are not following the twelve-factor guidelines often use a lightweight backing service in local environments, while using a much heavier and robust backing service in production.

Best practice for twelve-factor development is to use the same backing service in development and production. This avoids any incompatibilities or differences that can cause code to pass tests in staging but fail in production. These types of errors can have an extremely high cost over the application's lifetime, at a time when modern tools allow developers to run local environments that closely approximate production environments with minimal cost and effort over using lightweight local services.

20.1.11 Logs

Logs contain the stream of events collected from the output streams of all running processes and backing services. A twelve-factor app *never* worries about the routing or storage of its output stream.

Best practice is to have each process write its event stream, unbuffered, to stdout. Twelve-factor apps do not write to or manage logfiles — among the many flaws with containers using logfiles is that the logfiles will be lost when the container is destroyed, which is often the most critical time for the logged events to undergo analysis.

The execution environment captures each process's stream in staging and production, routing to destinations for viewing and long-term archival. Log indexing and analysis systems or data warehousing systems can receive event streams, which enable analysis of the app's operation over time (e.g., data trends, alerts based on user-defined heuristics).

20.1.12 Admin Process

Developers will often want to do specific administrative or maintenance tasks for the app such as database migrations or running one-time scripts committed into the app's repo. Best practices requires that one-off admin processes be run in an identical environment as the regular long-running processes of the app, running against a release using the same codebase and config as any process run against the release. Admin code must ship with application code for synchronization purposes.

The twelve-factor admin process strongly favors languages that provide a Read-Eval-Print Loop (REPL) shell out of the box that make it easy to run one-off scripts. Use the same dependency isolation techniques discussed earlier on all process types.

20.2 SECRETS MANAGEMENT

Credentials are environment dependent configurations that need to be secret and read only by subjects with a need-to-know. Hard coding credentials such as passwords into containers are insecure because the same password would replicate in each of the container's deploy environments.

The secret object type provides a mechanism to hold sensitive information such as passwords, OpenShift client config files, dockercfg files, etc. Secrets decouple sensitive content from the pods that use it and can be mounted into containers using a volume plug-in or used by the system to perform actions on behalf of a pod.

A system can directly manage secrets. There are a variety of open source and pay tools to help address secrets management. For example, HashiCorp's Vault offering includes a free offering that is sufficient for most users. They also offer Pro and Premium Enterprise versions with additional features used by companies such as Adobe, SAP Ariba, Citadel LLC, New Relic and Spaceflight.

These secrets management tools secure, store, and tightly control access to tokens, passwords, certificates, Application Program Interface (API) keys, and other modern computing secrets. Also provided are leasing, key revocation, key rolling, and auditing.

20.3 COMPUTE AND STORAGE LIMITATION FOR APPS

If the target production environment is an afloat Navy ship, it will not provide unlimited compute and storage capacity for the apps. In this case, app developers should verify their compute and storage requirements that the platform can satisfy these requirements.

20.4 TRUSTED BASE IMAGES

Build container images from trusted base images. If trusted DoD base images are not available, the next best option are base images provided by Red Hat.

The web interface for the Red Hat container catalog is located at https://access.redhat.com/containers/. In order to use Red Hat's container images, you will need to have a valid RHEL subscription for the machine that is pulling the images.

If the developer does not have access to trusted DoD base images or base images from Red Hat, the next best viable source is official base images on Docker Hub. Although Docker Hub is a public Docker registry and should not be trusted in general, the official Ubuntu base image provided by Canonical or the official CentOS base image provided by the CentOS project are more trustworthy than other images, because anyone in the world is permitted to upload images there.

Another option is to create your own base images from scratch. Docker Inc. provides information on how to do this at https://docs.docker.com/engine/userguide/eng-image/baseimages/.

At some point there will need to be a DoD agency or group responsible for creating secure base images and making them available to developers. Otherwise, each program or project will have to reinvent the wheel individually, which is very inefficient. However, there is currently no entity charged with this important task.

It is possible that at some point base images from Dockerhub — even official ones — will be assumed to be untrustworthy and banned from use by DoD policy. Developers should consider this while deciding the best option for their program.

21. PERFORMANCE TUNING

21.1 ACCUMULO

21.1.1 Pre-splitting

The most important factor in improving Accumulo write performance is pre-splitting the tables with a split file containing a list of split points. This will boost ingest performance tremendously, as pre-splitting will evenly distribute writes across all tablet servers instead of overloading a single tablet server.

To implement this change:

1. On Accumulo master, create a splits.txt file with appropriate split points on each newline, depending on the data in the **so_tcri_core** Accumulo table.

For example, if the row ID's start with numbers in sequential order, a good set of split points would look like:

```
1
2
3
4
5
7
8
9
```

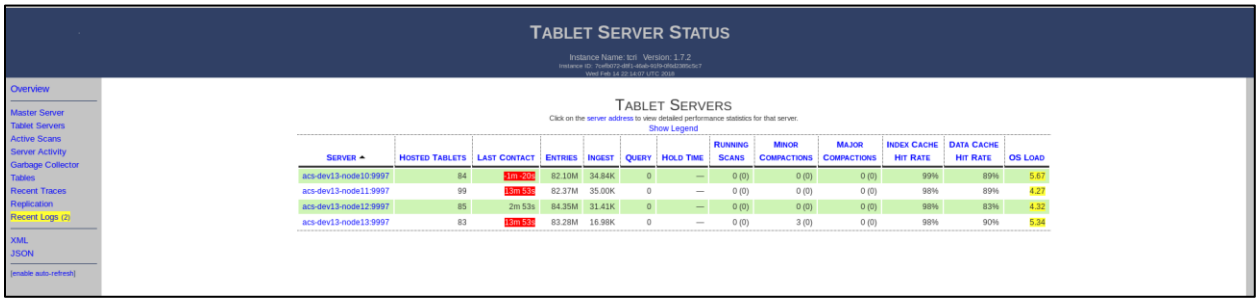
2. On the **accumulo_master** node, run:

```
/opt/sol/cli/bin/soadmin addsplits -n tcri -sf splits.txt
```

3. Ingest the data through the **so-persistence** topology.

4. Go to the Accumulo UI on **accumulo_master**, and click on Tablet Servers on the left menu.

5. Verify that the ingest and OS Load column values for every tablet server are relatively close, like the screenshot below.



Server	Hosted Tablets	Last Contact	Entries	Ingest	Query	Hold Time	Running Scans	Minor Compactions	Major Compactions	Index Cache Hit Rate	Data Cache Hit Rate	OS Load
acs-dev13-node10:9997	84	1m 20s	82.10M	34.84K	0	—	0 (0)	0 (0)	0 (0)	99%	89%	5.67
acs-dev13-node11:9997	99	1m 53s	82.37M	35.00K	0	—	0 (0)	0 (0)	0 (0)	98%	89%	4.27
acs-dev13-node12:9997	85	2m 53s	84.35M	31.41K	0	—	0 (0)	0 (0)	0 (0)	98%	83%	4.32
acs-dev13-node13:9997	83	1m 53s	83.28M	16.98K	0	—	0 (0)	3 (0)	0 (0)	98%	90%	5.34

Figure 149 Accumulo Tablet Server Status

There are no tradeoffs. This change is solely beneficial with no side effects or negative consequences.

21.1.2 Change table.file.compress.type to Snappy

Change table.file.compress.type to snappy to increase write speed.

To implement this change:

1. In Accumulo shell on **accumulo_master**:

```
config -s table.file.compress.type=snappy
```

The tradeoff is that snappy compression increases write speed at the expense of using more disk space.

21.1.3 Give Accumulo More Memory

Since Accumulo ingest can be memory intensive, dedicating more memory to the tablet servers will improve performance. When a tablet server runs low on memory, the garbage collector runs to free up memory. During garbage collection, there are long ingest pauses that slow down performance.

The ACS Services team provisioned 256 GB of RAM to all Accumulo tablet servers and Accumulo master nodes in VMware vCenter Server. A user must also proportionally increase every memory option value in **/opt/accumulo/accumulo-current/conf/accumulo-env.sh** on **accumulo-master** to reflect the increased amount of memory. For example, if memory was increased from 8 GB to 256 GB, then all the memory values should be multiplied by 32 in **accumulo-env.sh**. Next, Accumulo should be restarted. On **accumulo_master**:

```
sudo -s
su - accumulo
cd /opt/accumulo/accumulo-current/bin
./stop-all.sh
./start-all.sh
```

A user must pre-split to the Accumulo tables to fully utilize the additional memory available to improve ingest performance.

The Geomesa page, https://geomesa.atlassian.net/wiki/spaces/GEOMESA/pages/26312706/Tuning+Accumulo#TuningAccumulo-ServerSideParameters, has some good suggestions to fine-tune Accumulo write performance. Avoid modifying **tserver.total.mutation.queue.max** and **tserver.mutation.queue.max**. The official Accumulo documentation provides more detailed descriptions of each of these table-level properties at https://accumulo.apache.org/docs/2.0/administration/properties#table_balancer.

21.1.4 Create Additional Accumulo Tablet Servers and HDFS Datanodes

Scaling out and increasing the number of Accumulo tablets and datanodes also improves write performance, since more tablet servers and datanodes means more data can be written at a time. The inventory file can be modified to create additional Accumulo tablets simply by adding more nodes under the accumulo_tablets and datanodes roles and rerunning tcri-master.yaml. There is no need to increase the overall number of nodes in the cluster. A user must pre-split to the

Accumulo tables to fully utilize the additional tablet servers available to improve ingest performance.

21.1.5 Results from Accumulo Tuning

Table 14 Results From Accumulo Tuning

Number of Tablet Servers	Pre-Tuning Ingest Rate (Tuples emitted/second)	Post-Tuning Ingest Rate (Tuples emitted/second)
1	1200	1800
2	1400	2000
4	1700	2300
6	1800	2500
8	2000	2700
10	2200	2800

22. TOOLS

22.1 TEST HARNESS

22.1.1 Purpose

The test-harness-tcri is a data generation tool used by the ACS community. The purpose of this page is to provide a high level overview of how the test-harness is used. This page will rely on provided scripts and configurations that come with the test-harness. If you need additional capabilities/features, please read the project's documentation as it covers many additional features that are not included in this tutorial.

There are a couple of key concepts that you should understand before using the test-harness.

1. The test-harness is a data generation tool and not a simulator. Most data is assigned a random value from a set of preconfigured values. The test-harness can simulate straight-line movement to produce non-random location reports but this is not a high fidelity simulator.
2. The test-harness produces messages based on a single template message. For instance, you can use a single ais-oth and, with a simple json script, have the test-harness produce thousands of unique messages.
3. The test-harness is designed to generate load on ACS, but does not measure ACS performance.
4. There may be one or more ways to increase production rates.
5. Like all systems there are limitations to the test-harness. The main limiting factor is the number of CPUs on the machine running the test-harness. If you are not achieving your configured production rates, look at your system's resources while paying attention to CPU performance. You may be maxing out your CPUs.

22.1.2 Ingest TRIG Files

Before we start producing track reports, we need to ingest a trig file with details about each track. Tracks contain a specific MMSI and trig files contain additional details about each MMSI. Run the following to ingest the trig file.

```
bin/test-harness-tcri scripts/ingest/trig.json
```

22.1.3 Producing AIS-OTH Data

The test-harness provides many scripts but we're going to focus on scripts/ingest/performance/dpf/ais-oth.json which uses ACS' ais-oth artifact model. We'll walk you through how to increase the production rate in a second but, first, let's talk about the data.

This script produces ais-oth.xml and is preconfigured to update each outgoing message with unique values to give you some variability in the data. Considering we know this outputs track reports, we use the SimulateMovement event that calculates straight-line movement to give you somewhat realistic movement. Each message records the current time so that it appears that the

messages were created in real-time. Messages are also linked to a specific MMSI. MMSIs can be found in the trig files that you have already ingested above.

There are couple of ways to increase the production rate of this script.

First, you can link to more MMSIs. You can do this by adding more MMSIs to scripts/mmsi.txt. scripts/ingest/performance/dpf/ais-oth.json will produce a series of track reports based on how many lines are in scripts/ingest/performance/mmsi.txt. Every second an AIS report for each MMSI in that file is produced. Adding more MMSIs will get more messages per second. You can ensure that those MMSIs are in the trig by running `grep mmsi trigs/simulated_vessels_corrected.trig | awk -F\" '{ print $2 }' | head -n$NUM > scripts/ingest/performance/mmsi.txt` where NUM is the number of MMSIs that you'd like to simulate. Now, there are some limitations to this. You may not have enough MMSIs to achieve the output rate you want. simulated_vessel_corrected.trig only has 1000 MMSIs so if you want more than 1000 messages per second and don't want to create random MMSIs not contained in the trig file, follow the next step.

Second, you can increase the number of track reports for each MMSI. By default, only a single report will be generated for each MMSI. You can increase this by adjusting the producer's count. Count:2 will produce two reports per mmsi, Count:10 will produce ten reports per mmsi. It's basically just a multiplying factor to the number of lines in mmsi.txt.

Third, you can send each report at a shorter frequency. It is important to note that 99% of the time you can use the previous two methods to increase/decrease production rates. At the bottom of the script there is an UpdateProductionFrequency event. Adjust its frequency to whatever is desired. Frequency:"500ms" will send a new report every 500 milliseconds while Frequency:"10s" will send a report every 10 seconds.

Even if you follow all of these instructions perfectly, you may not be able to achieve your desired output rate due to limited resources. In testing, a single test-harness instance was able to produce about 3000 ais-oth messages per second. You may need a higher rate than that though. To achieve this you will need to run a separate instance of the test-harness in your cluster. Simply follow the previous steps on a separately deployed instance and run both at the same time.

Now that you've configured your desired output rate, you need to configure how long you want the test-harness to run. To do this, adjust the Apocalypse event to your desired run length. Delay:"2hr" will run for two hours, Delay:"10min" will run for ten minutes, etc.

22.1.3.1 Example - 100 messages per second

Let's do this in an example. First, we want to produce 100 messages per second for 25 minutes.

```
grep mmsi trigs/simulated_vessels_corrected.trig | awk -F\" '{ print $2 }' | head
-n100 > scripts/ingest/performance/mmsi.txt
```

Edit scripts/ingest/performance/dpf/ais-oth.json and make sure Apocalypse is set appropriately

```
...
{ "type":"Apocalypse", "delay":"25min" }
...
```

Now re-run

```
bin/test-harness-tcri scripts/ingest/performance/dpf/ais-oth.json
```

22.1.3.2 Example - 1500 messages per second

Now let's produce 1500 messages per second for 1 hour

```
grep mmsi trigs/simulated_vessels_corrected.trig | awk -F\" '{ print $2 }' | head
-n750 > script/ingest/performance/mmsi.txt
```

Edit the script and make sure producer count is set to 2 and Apocalypse are set appropriately

```
{
     "producers":[{
    "type":"batch",
    "name":"producer",
    "count":2,
...
{ "type":"Apocalypse", "delay":"1hr" }
...
```

Now run it....

```
bin/test-harness-tcri scripts/ingest/performance/dpf/ais-oth.json
```

22.1.4 Troubleshooting

The below describes the few common issues that can appear for ACS and how to resolve them.

22.1.4.1 Data Not Going to Kafka

The test-harness is specific to a particular ingest pipeline. There are many different output formats that it supports (e.g., console, log4j, rabbitmq). To ingest through TCRI's DPF, make sure that the transmitter is set to `dpf-kafka` in the producer's definition

```
     "producers":[{
    ...
    "transmitter":{
      "type":"dpf-kafka",
      "topic":"dpf",
      "dpf-lookup-key":"ais-oth"
    }
  }
```

22.2 DPF UTILITY

22.2.1 Background

The core components included with the UCD ingest and query capabilities allow querying for terms and statements by sign value, as long as the Concept referenced by the sign is part of the core models. Unfortunately, many knowledge domains include concepts that are not part of the UCD core models. In order to query by sign (value) against Concepts that are not part of the core models, the Concepts must first be registered with the system, such that they are present in the UCD tables within Accumulo before an ingest occurs which references those Concepts.

22.2.2 Using the DPF Utility to Register Concepts

DPF Utility may be used to register custom concepts and predicates for use in indexing and querying data via the UCD APIs. The DPF Utility operates by examining the currently published models (located in the /ingest/repository/domain directory of the HDFS distributed file system), looking for Concept and Predicate Definition documents. The utility compares the Concepts and Predicates included in the definition documents with those currently published to the **ucd_concept** and **ucd_predicate** tables. Missing concepts and predicates are registered with the system by publishing them to the Accumulo tables.

For the DPF Utility to have any effect, there must be at least one domain model deployed to the HDFS ingest repository that includes Concept and/or Predicate Definitions (as evidenced by the presence of <model name>.cd.xml and <model name>.pd.xml files in /ingest/repository/domain/<model name>.

Confirm this with the following command on **storm_master**:

```
su - hdfs
hdfs dfs -ls -R /ingest/repository/domain
```

To run the tool and register custom Concepts and / or Predicates, execute the utility with the **preingest** argumemt on **storm_master**:

```
/opt/common/util/tcri/utility/bin/dpf-utility preingest
```

DPF Utility should complete with a line similar to "INFO m.a.i.d.u.p.PreIngestCommand - Found 494 unique concepts, predicates or associations defined in the repository."

22.2.3 Known Limitations

The utility does not associate registered concepts and predicates with a domain model (as an Artifact).

23. TROUBLESHOOTING

23.1 CLEAR ACCUMULO TABLES

23.1.1 Step-by-Step Guide

1. Drop Accumulo tables, rebuild the tables, set table authorizations, and run preingest. On the basebox:

```
ansible-playbook -M library -k -vv -u root dfntc-reset.yml --tags=accumulo
```

23.2 CLEAR KAFKA

23.2.1 Step-by-Step Guide

1. Stop all topologies, reset kafka, and start topologies. On the basebox:

```
ansible-playbook -k -vv -b dfntc-reset.yml --tags=kafka
```

By default, this removes all kafka topics and recreates the kafka topics listed in group_vars/all/kafka.yml. You can alter the variables to change the number of partitions and replications for your environment.

23.3 OPENSHIFT

23.3.1 acs-extras is needed for the install/Flannel not found

```
---

- hosts: tcri_nodes

  tasks:

  - command: rpm -Uvh http://10.10.10.4/pub/katello-ca-consumer-latest.noarch.rpm
--force
  - command: subscription-manager register --serverurl=https://acs-
repo.sd.spawar.navy.mil --activationkey=acs-rhel7-virt --activationkey=acs-extras
--org='Default_Organization' --force
```

TASK [openshift_node : Install Node package, sdn-ovs, conntrack packages]
****
****

task path: /opt/acs/acs-openshift/roles/openshift_node/tasks/install.yml:2

Wednesday 23 May 2018 17:15:06 +0000 (0:00:00.468) 0:16:42.508 *********

FAILED - RETRYING: Install Node package, sdn-ovs, conntrack packages (3 retries left).

failed: [node13.dev9.acs.sd.spawar.navy.mil] (item={u'name': u'atomic-openshift-node-3.7.42'}) => {"attempts": 3, "changed": false, "item": {"name": "atomic-openshift-node-3.7.42"}, "msg": "https://acs-repo.sd.spawar.navy.mil/pulp/repos/Default_Organization/Library/custom/Cloudera/CDH_5_8/repodata/repomd.xml: [Errno 14] curl#6 - \"Could not resolve host: acs-repo.sd.spawar.navy.mil; Unknown error\"\nTrying other mirror.\nhttps://acs-repo.sd.spawar.navy.mil/pulp/repos/Default_Organization/Library/custom/EPEL/EPEL/repodata/repomd.xml: [Errno 14] curl#6 - \"Could not resolve host: acs-repo.sd.spawar.navy.mil; Unknown error\"\nTrying other mirror.\nhttps://acs-repo.sd.spawar.navy.mil/pulp/repos/Default_Organization/Library/custom/Elastic/Bea

```
ansible -a "systemctl restart dbus dnsmasq" nodes -vvk
```

Flannel is missing

```
ansible -vvka "subscription-manager register --serverurl=https://acs-
repo.sd.spawar.navy.mi --activationkey='acs-openshift-backup' --activationkey=acs-
rhel7-virt --activationkey=acs-extras --org='Default_Organization' --force" nodes
```

APPENDIX A FLIGHT DATA DEMO

INTRODUCTION

This demonstration leverages the geospatial mapping capabilities of CJMTK and Accumulo persistence through the structured objects (SO) datastore. Artifact and domain models that contain both structured objects mapping and simple features mapping for plotting geospatial points are needed. The guide first focuses on verifying what is expected when flight data is ingested without any artifact models installed for a before and after comparison. The guide then dives into verifying any expected indicators when both batch flight data and streaming flight data are ingested. The end of the guide walks through how to plot the flight data points on an ArcGIS map by creating a stream service layer through the CJMTK Portal.

PROCEDURES

1. Set up NiFi workflow. The NiFi web interface can be accessed through http://<nifi>:6434/nifi. By creating a NiFi processor to constantly fetch data from a URL, it is possible to stream flight data from the ADS-B Exchange (https://ADSBexchange.com) feed in real time and persist it to structured object. This NiFi dataflow requires three processors: GetHTTP, DPFInputWrapper, and PublishKafka
    a. Create a new processor

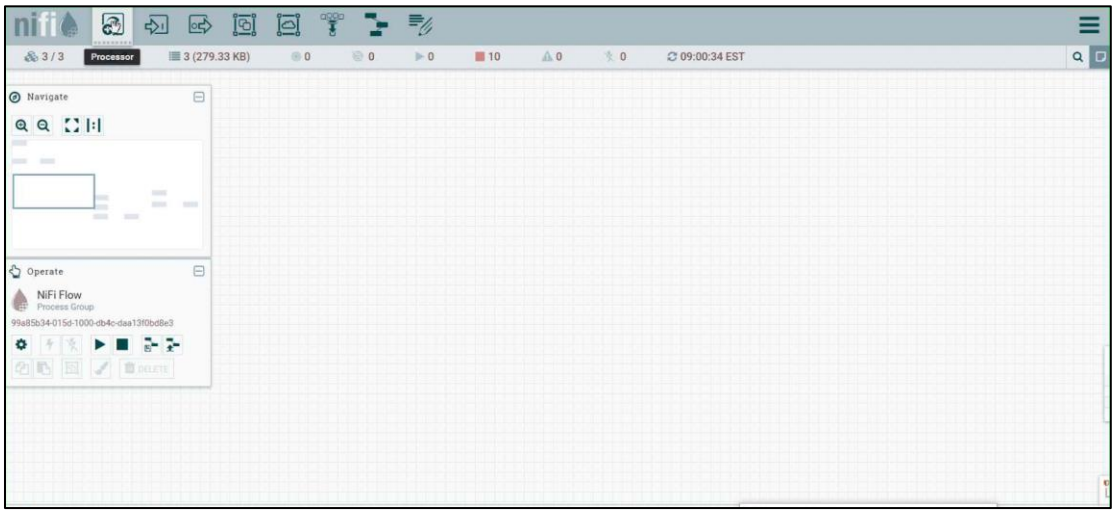


Figure 150 Create a New Processor

    b. Drag it to the grid workspace

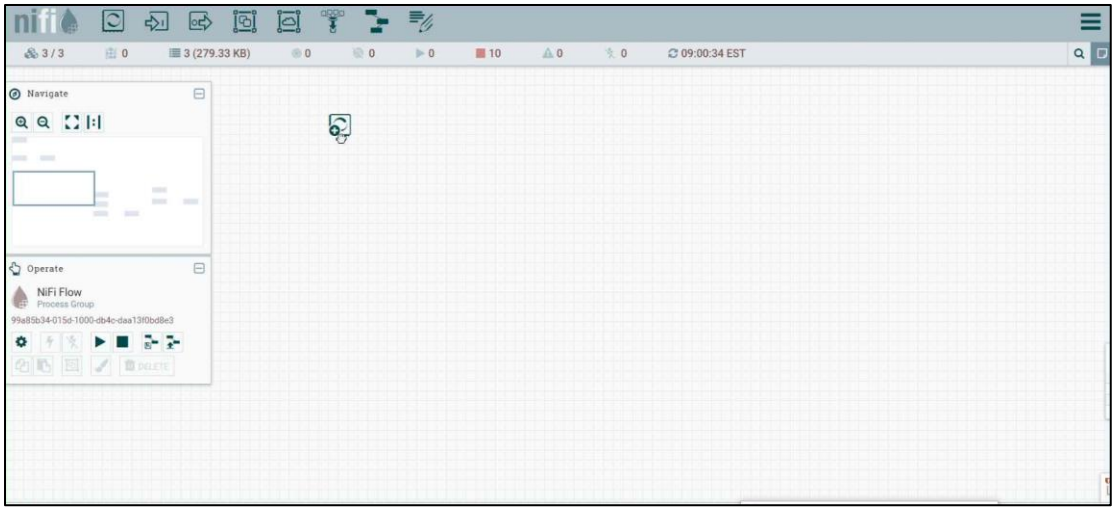


Figure 151 Workspace

c. Select the GetHTTP processor by searching for http in the upper right search bar

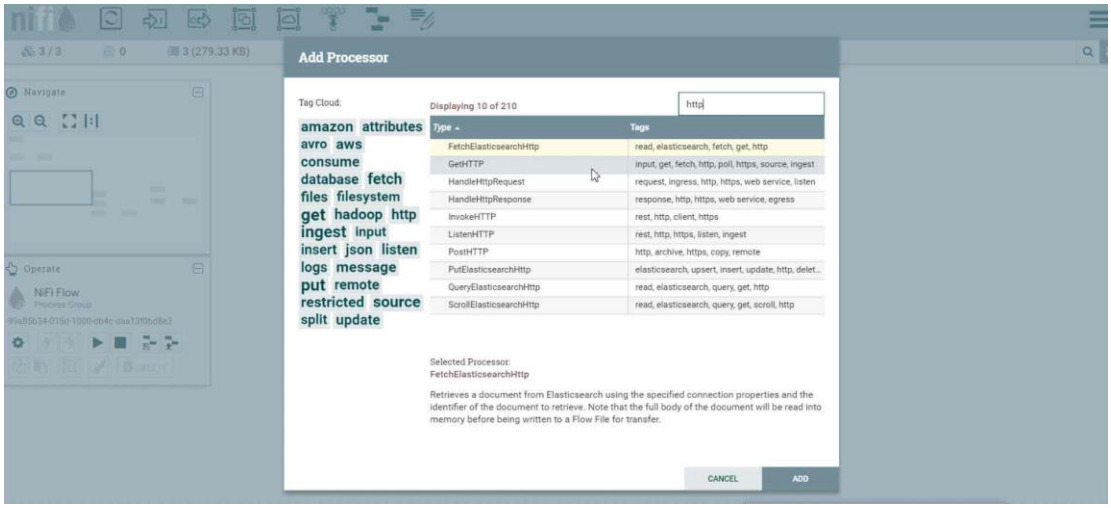


Figure 152 GetHTTP Processor

d. Right click on the GetHTTP box and click Configure

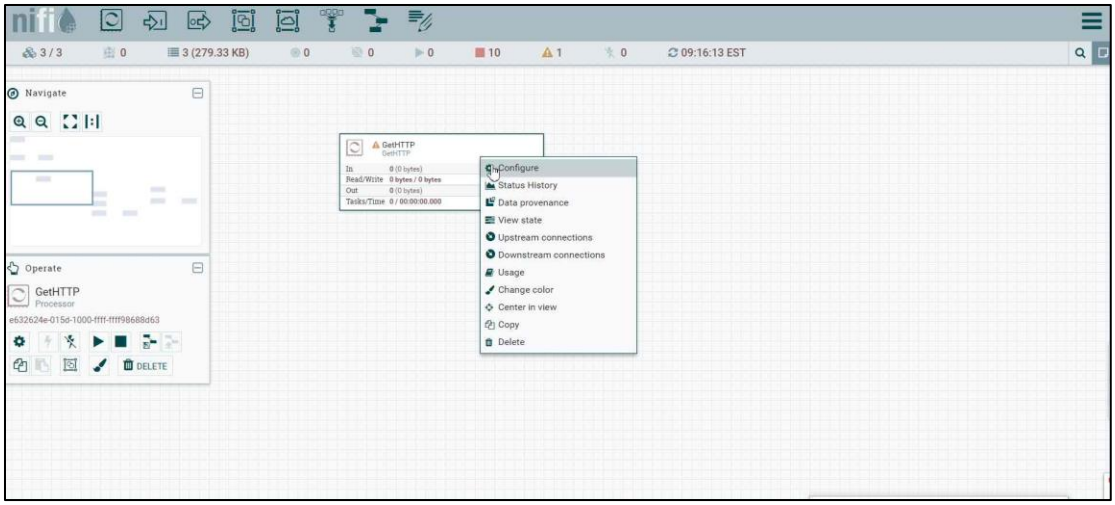


Figure 153 Configure

e. Set GetHTTP's URL to https://public-api.adsbexchange.com/VirtualRadar/AircraftList.json?lat=33.433638&lng=-112.008113&fDstL=0&fDstU=100

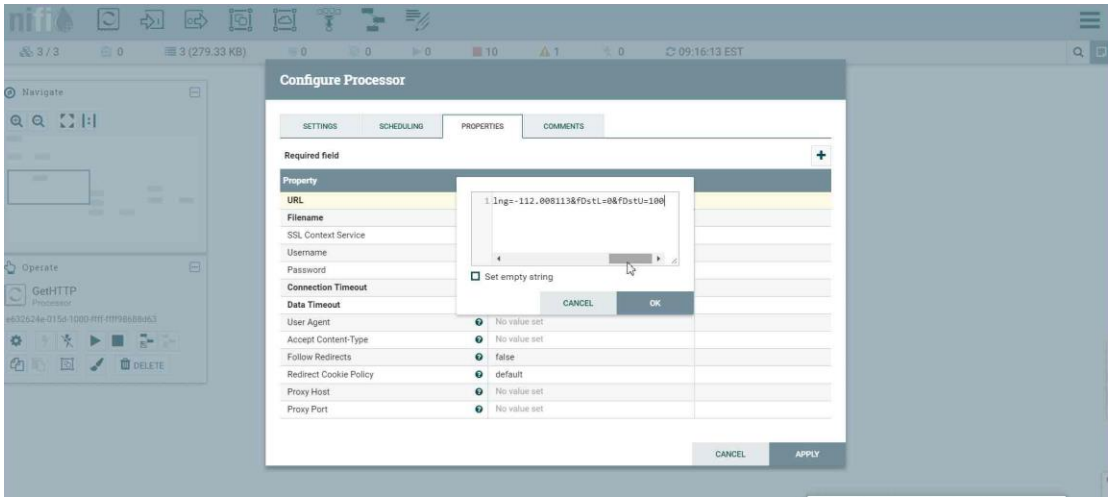


Figure 154 Set GetHTTP's URL

f. Set Filename to flights.json

g. Click on SSL Context Service

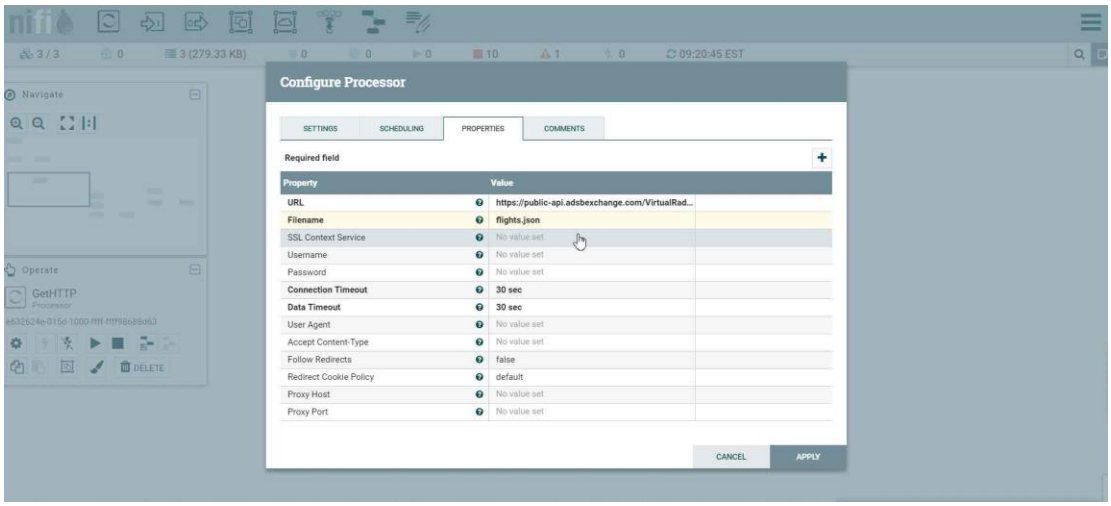


Figure 155 SSL Context Service

h. In the dropdown menu, click Create New Service

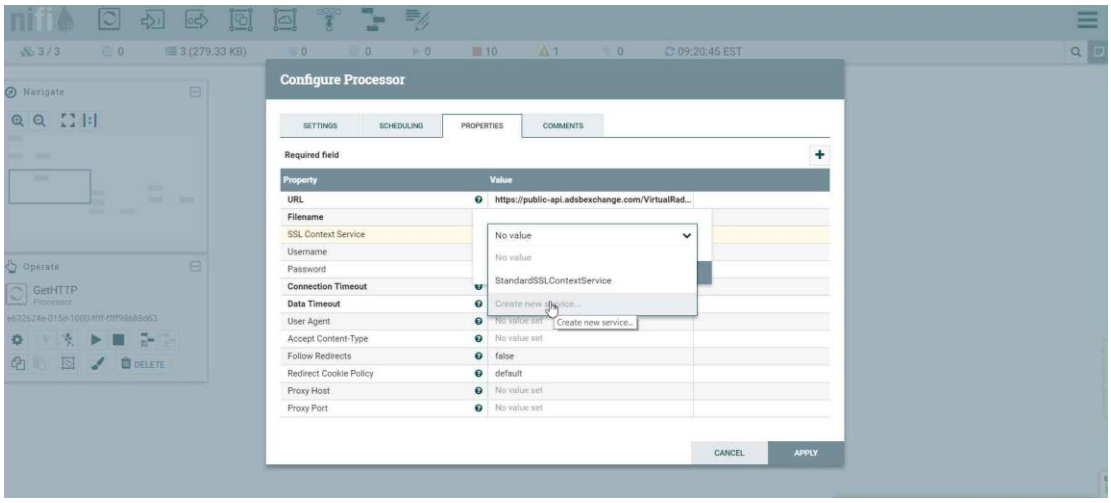


Figure 156 Create New Service

i. Make sure the StandardSSLContextService choice is selected under Controller Service, then click Create

j. Click the right arrow next to SSL Context Service's value field

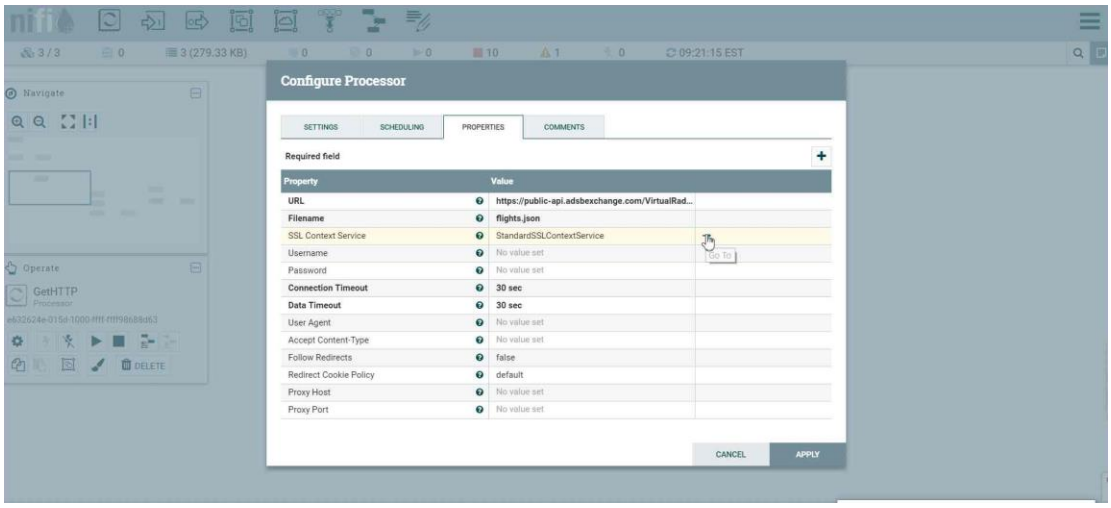


Figure 157 Context Service's Value Field

k. When it prompts you to save changes, click yes

l. Click the pencil icon to edit

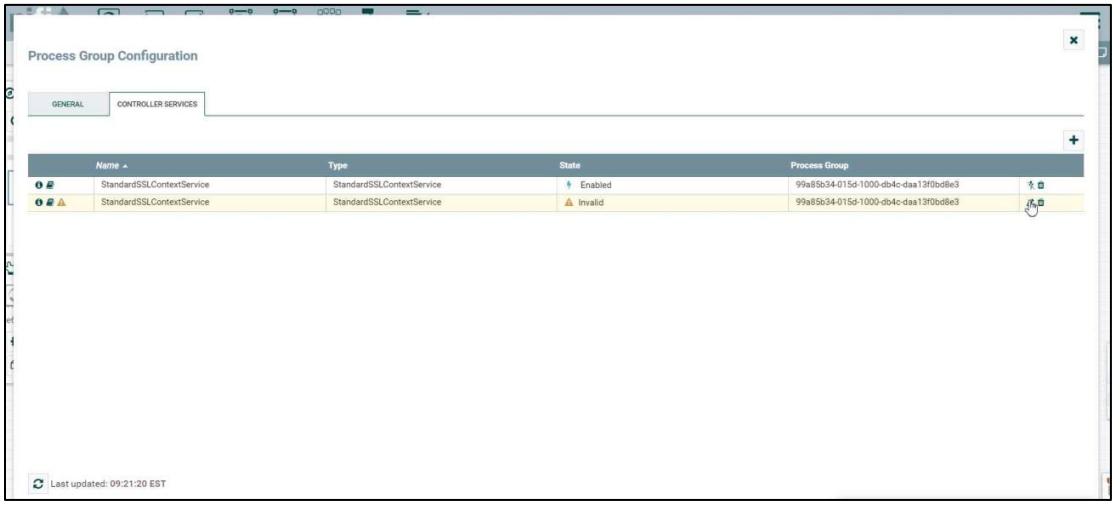


Figure 158 Edit

m. Set Truststore Filename to: /usr/lib/jvm/java-openjdk/jre/lib/security/cacerts

Set Truststore Password to ######## (refer to truststore_pass value in tcri.yml config file on basebox)

Set Truststore Type to: JKS

SSL Protocol should be TLS

We need to do this because the source URL is HTTPS. See https://community.hortonworks.com/questions/9509/connecting-to-datasift-https-api-using-nifi.html for why an SSLContextService is required

Click Apply

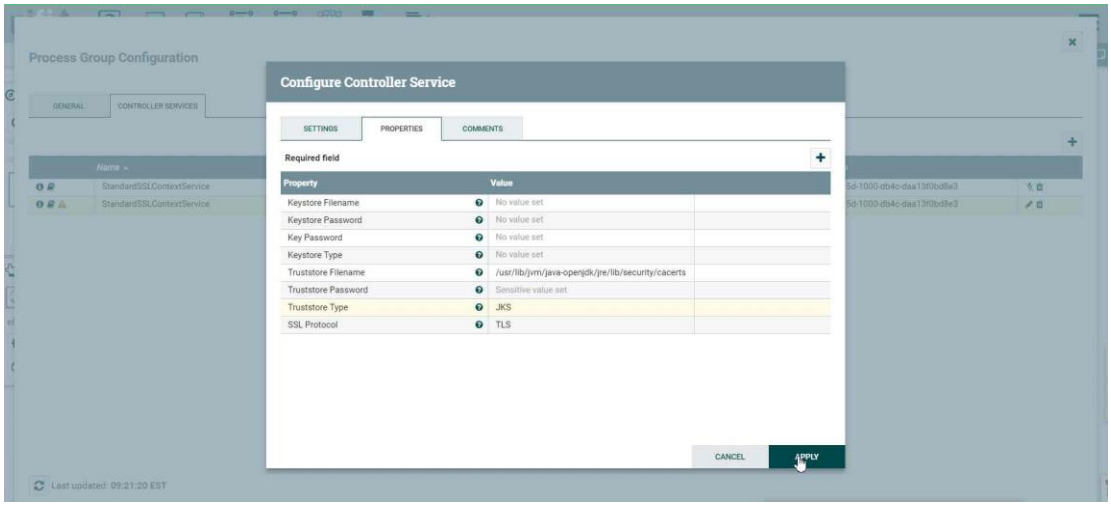


Figure 159 Setup Truststore

n. Click the lightning bolt to enable the SSL Context Service.

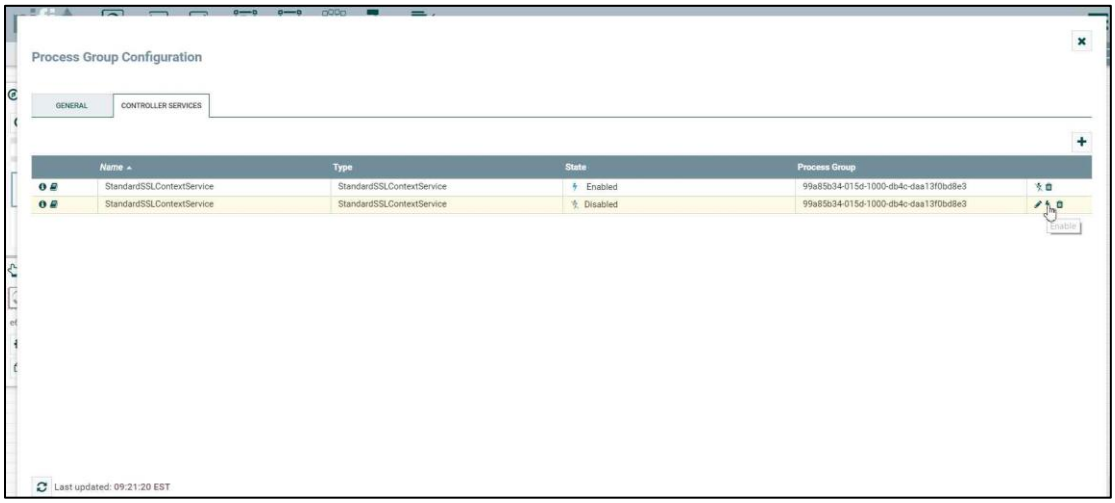


Figure 160 Enable the SSL Context Service.

o. Under Scope, select Service Only. Then click Enable. Click the Close button once it's enabled.

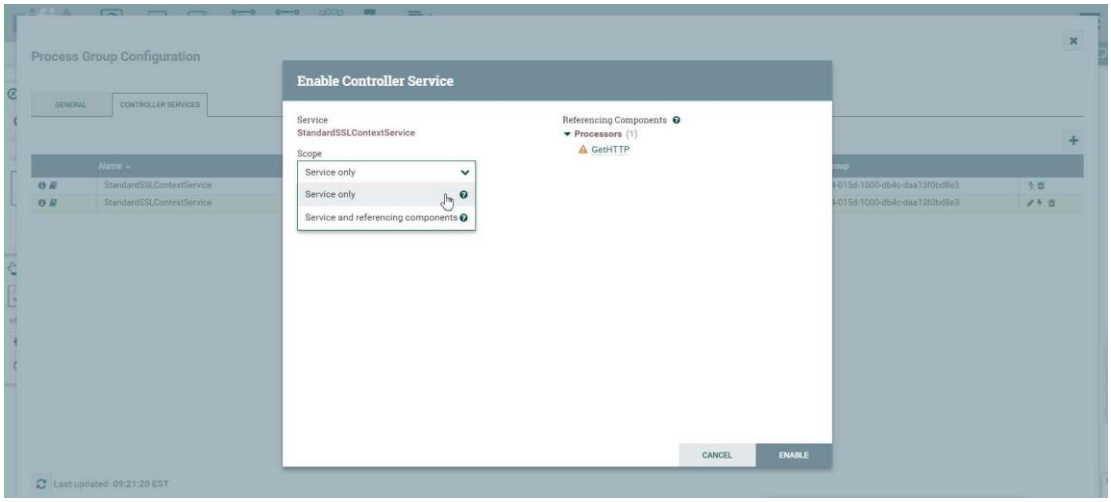


Figure 161 Enable Controller Service

p. Click and drag a new Processor, and select DPFInputWrapper. This must point to the DPF lookup key corresponding to the artifact model intended to extract flight data. To do this, right click on DPFInputWrapper, click Configure, and under the Properties tab, set DPF MODEL LOOKUP KEY to 'demo-flight-geo'

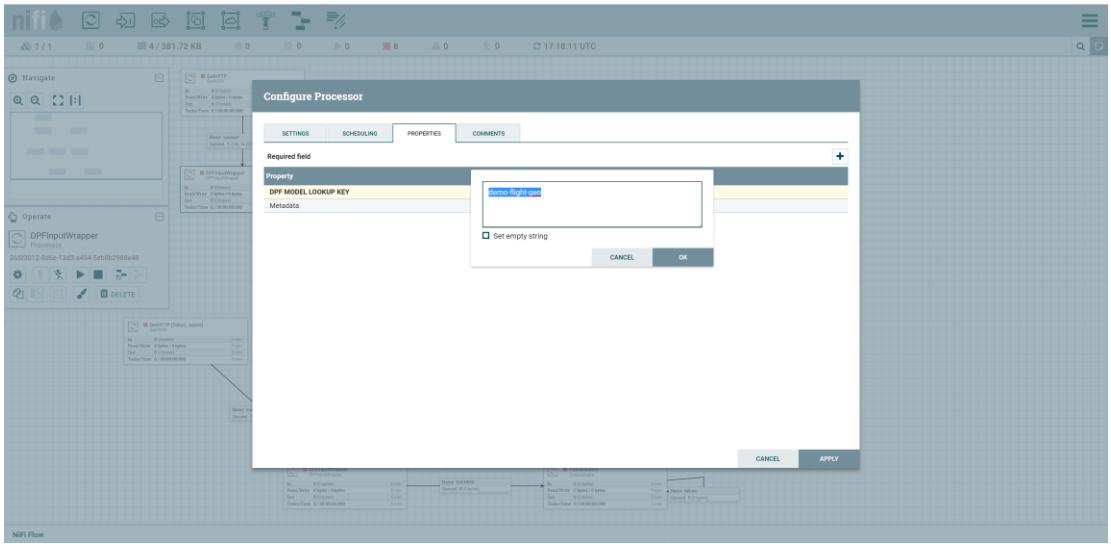


Figure 162 New Processor

q. Click and drag a new Processor, and select PublishKafka. Right click configure, and under Properties, set Kafka Brokers to <kafka_servers>:9092 (ie. 'acs-dev3-node03:9092') and Topic Name to 'dpf'.

r. Draw success arrows from GetHTTP to DPFInputWrapper and DPFInputWrapper to PublishKafka, and a failure arrow from PublishKafka to itself. Arrows can be drawn by clicking within the desired source, so that a diagonal arrow button pops up, and dragging the arrow to the destination.

s. Right click configure DPFInputWrapper, and under Settings, check the box for FAILURE under Automatically Terminate Relationships. Right click configure

PublishKafka, and under Settings, check the box for success under Automatically Terminate Relationships. The NiFi workflow should look like the picture below.

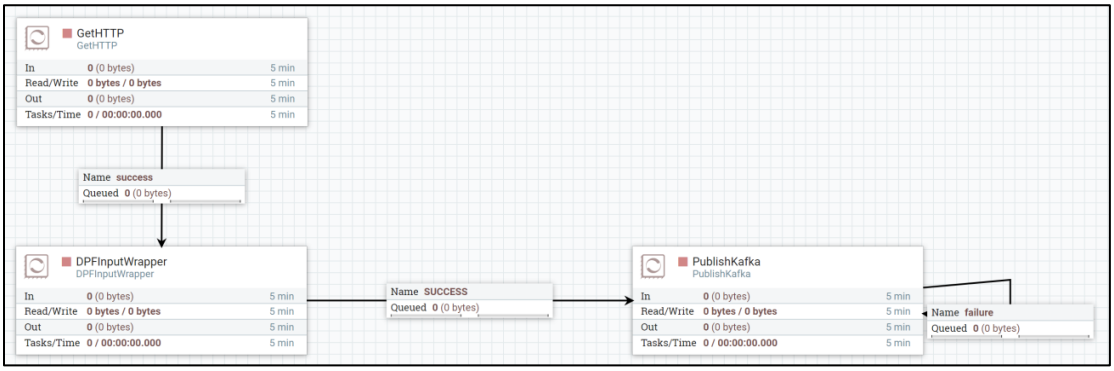


Figure 163 Configure DPFInputWrapper

2. Ingest data - see that it is quarantined (shows that it doesn't work without custom models)
    a. Shift select all three processors in the demo_flight_geo NiFi dataflow template and click on the 'Start' icon
    b. Observe that the following values increment:
        i. GetHTTP processor's Out-to
        ii. DPFInputWrapper processor's In/Out-to
        iii. PublishKafka processor's In-to
    c. Browse to the Storm Web UI at <storm_master_node>:8080/index.html
    d. Click on the 'Streaming-Data-Processing-Framework' topology link under 'Topology Summary'
    e. Observe incrementing values on edges between Kafka-Reader.dpf spout, Processing bolt, and XX.Quarantine bolt

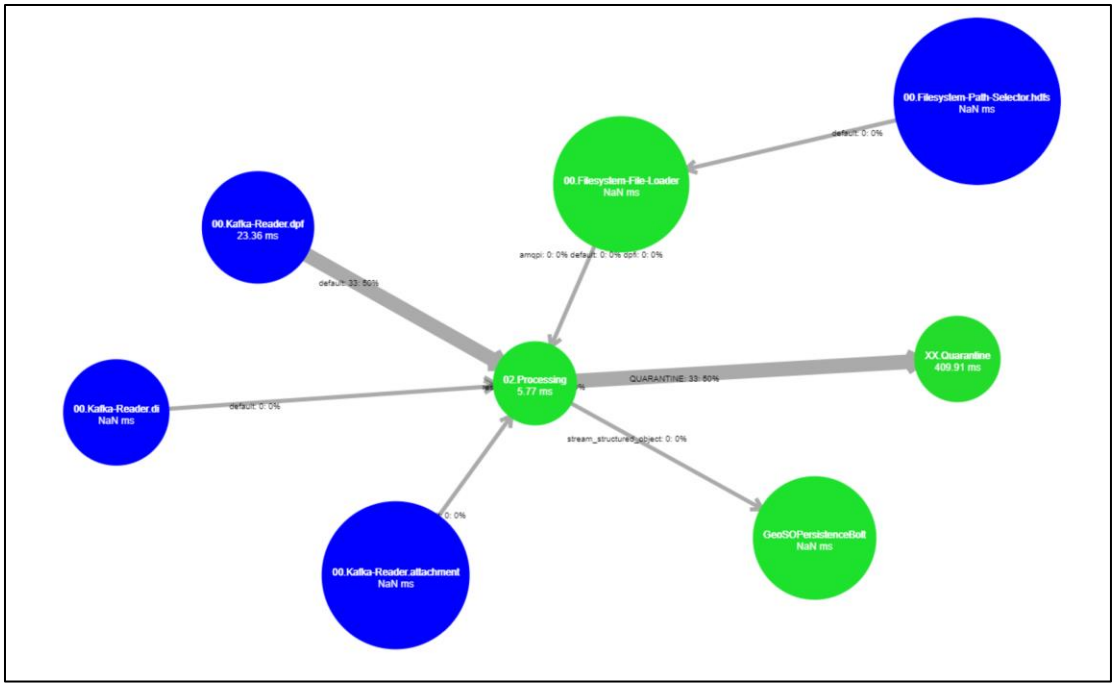


Figure 164 Ingest Data

f. Shift select all three processors in the demo_flight_geo NiFi dataflow template and click on the 'Stop' button

3. Run SOQL query on SOQL web interface or SO client. Observe no items matched
   a. With SOQL web interface:
      i. Browse to the SOQL Web UI at https://<so_web_server>:8443/sol-ws/query.html
      ii. Type the following into the query box

```
SELECT * FROM 'so:///tcri/demo_flight_geo'
```

      iii. Click on 'Submit'
      iv. Observe zero returned queries
   b. With SO client:
      i. SSH into node 1
      ii. Execute the following command

```
sudo /opt/sol/cli/bin/soclient query -n tcri -q
"SELECT * FROM 'so:///tcri/demo_flight_geo'"
```

      iii. Observe zero returned queries

```
21:30:46.742 INFO  m.t.s.c.c.QueryCommand - 0
results counted in 0.02 seconds
21:30:46.750 INFO  m.t.s.c.c.QueryCommand - 0.0
results per second.
```

4. Install custom artifact / domain models by uploading to HDFS. The artifact and domain models provide the underlying schema mapping to persist the flight data to both the structured objects (SO) datastore and the CJMTK/ArcGIS datastore
    a. Browse to the Storm web UI at <storm_master_node>:8080/index.html
    b. Click on the Streaming-Data-Processing-Framework topology under the Topology Summary section
    c. Kill the Streaming-Data-Processing-Framework topology by clicking the 'Kill' button located near the top of the page
    d. Change the default time to 1 second when prompted and click 'Ok'
    e. SSH into <storm_master>
    f. Create the following artifact model files in a folder named 'demo-flight-geo-art' in the /opt/common/models/tcri/ directory
        i. demo-flight-geo.am.xml

```
<?xml version="1.0"?>
<artifactModel xmlns="urn:dpf:ingest-artifact-
model"
               xmlns:version="1.3.2">
    <properties>
        <document format="json">
            <charset>utf-8</charset>
        </document>
    </properties>

    <artifact>
        <security>
            <classification
eval="literal">UNCLASSIFIED</classification>
        </security>

        <metadata>
            <language
eval="literal">en_US</language>
            <contentType>
                <primaryType
eval="literal">text</primaryType>
                <subType
eval="literal">json</subType>
            </contentType>
            <contentUri
eval="literal">abc:def:ingest-artifact-
model</contentUri>
        </metadata>
        <content>
            <structuredData>
                     <dataDescriptor
```

```
eval="literal">tcri:prov</dataDescriptor>
                <properties>

                    <property>
                        <key
eval="literal">Model</key>
                        <value
eval="literal">tcri:demo-flight-geo</value>
                    </property>
                      <property>
                        <key
eval="literal">icao</key>
                        <value eval="jsonPath"
required="false">$..Icao</value>
                    </property>
                    <property>
                        <key
eval="literal">latitude</key>
                        <value eval="jsonPath"
required="false">$..Lat</value>
                    </property>
                    <property>
                        <key
eval="literal">longitude</key>
                        <value eval="jsonPath"
required="false">$..Long</value>
                    </property>
                    <property>
                        <key
eval="literal">altitude</key>
                        <value eval="jsonPath"
required="false">$..Alt</value>
                    </property>
                    <property>
                        <key
eval="literal">trak</key>
                        <value eval="jsonPath"
required="false">$..Trak</value>
                    </property>
                    <property>
                        <key
eval="literal">op</key>
                        <value eval="jsonPath"
required="false">$..Op</value>
                    </property>
                </properties>
            </structuredData>
        </content>
    </artifact>
    <attachments/>

    <domainMapping linkage="tcri:demo-flight-
geo">

        <instance linkage="tcri:demo-flight-
geo">

            <scope
eval="jsonPath">$.acList[*]</scope>

            <term handle="artifact_key">
```

```
                <primitive eval="el"
type="string" required="false">
                    return
context:getArtifactKey();
                </primitive>
            </term>
            <term handle="latitude">
              <primitive eval="jsonPath"
type="double"
required="false">$..Lat</primitive>
            </term>
            <term handle="longitude">
              <primitive eval="jsonPath"
type="double"
required="false">$..Long</primitive>
            </term>
            <term handle="altitude">
              <primitive eval="jsonPath"
type="double"
required="false">$..Alt</primitive>
            </term>
            <term handle="trak">
              <primitive eval="jsonPath"
type="double"
required="false">$..Trak</primitive>
            </term>
            <term handle="op">
              <primitive eval="jsonPath"
type="string" required="false">$..Op</primitive>
            </term>
           <term handle="icao">
              <primitive eval="jsonPath"
type="string"
required="false">$..Icao</primitive>
            </term>
        </instance>
    </domainMapping>

    <!-- ######################### Structured
Object Mapping  ######################### -->

    <structuredObjectMapping>

        <structuredObject
handle="demo_flight_geo">
            <scope
eval="jsonPath">$.acList[*]</scope>
            <kind>demo_flight_geo</kind>

        <id eval="jsonPath">$..Icao</id>

            <security>
                <classification
eval="literal">UNCLASSIFIED</classification>
            </security>

            <properties>
                <property label="artifact_key"
type="string">
                    <primitive eval="el"
required="false">
```

```
                        return
context:getArtifactKey();
                    </primitive>
                </property>
                  <property label="latitude"
type="double">
                    <primitive eval="jsonPath"
required="false">$..Lat</primitive>
                </property>
                <property label="longitude"
type="double">
                    <primitive eval="jsonPath"
required="false">$..Long</primitive>
                </property>
                <property label="altitude"
type="double">
                    <primitive eval="jsonPath"
required="false">$..Alt</primitive>
                </property>
                <property label="trak"
type="double">
                    <primitive eval="jsonPath"
required="false">$..Trak</primitive>
                </property>
                <property label="op"
type="string">
                    <primitive eval="jsonPath"
required="false">$..Op</primitive>
                </property>
                <property label="icao"
type="string">
                    <primitive eval="jsonPath"
required="false">$..Icao</primitive>
                </property>
            </properties>

        </structuredObject>
    </structuredObjectMapping>

    <simpleFeatureMapping>
    <simpleFeatureType>
        <scope
eval="jsonPath">$.acList[*]</scope>
        <name>demo_flight_geo</name>
        <security>
            <classification
eval="literal">UNCLASSIFIED</classification>
        </security>
        <attributes>
            <attribute label="time" type="date">
                <date>
                    <long eval="el">
                            return
cp:forName('java.util.Date').newInstance().getTi
me();
                    </long>
                </date>
            </attribute>
            <attribute label="geometry"
type="point">
                <primitive eval="el">
```

```
                    var lat =
jsonPath:getText("$..Lat")
                    var long =
jsonPath:getText("$..Long")
                    return "POINT(" + long + " "
+ lat + ")";
                </primitive>
            </attribute>
            <attribute label="icao"
type="string">
                <primitive
eval="jsonPath">$..Icao</primitive>
            </attribute>
            <attribute label="op" type="string">
                <primitive
eval="jsonPath">$..Op</primitive>
            </attribute>
            <attribute label="altitude"
type="double">
                <primitive
eval="jsonPath">$..Alt</primitive>
            </attribute>
            <attribute label="trak"
type="double">
                <primitive
eval="jsonPath">$..Trak</primitive>
            </attribute>
        </attributes>
    </simpleFeatureType>
    </simpleFeatureMapping>

</artifactModel>
```

Note:

Artifact model scope

Code Block 13 Scope from artifact model XML file

```
<scope eval="jsonPath">$.acList[*]</scope>
```

The scope path points to a set of array elements where each element is individually mapped to its own structured object. It was necessary to set the scope to $.acList[*], since this path expression instructs the model to iterate over each element within the acList array that is in the flight data sample file. The [*] is needed to recognize each array element as an individual section.

DPF limitation

The current limitation with DPF is that it can only map a single value to a mapping expression. In this case, that single value is the first element of the acList array. Many of the existing artifact models within ACS just map to the root document ($ for Json syntax). Due to the DPF limitation, mapping to the root document would not work for the demo since the model must pull multiple values and only the first element of the acList array would persist to structured objects.

Artifact model id

Code Block 14 Id from artifact model XML file

```
<id eval="jsonPath">$..Icao</id>
```

Artifact model id must be any data field for each array element that corresponds to a unique data value. The Icao field was chosen to represent the artifact model id, since the Icao is a unique six-digit hexadecimal identifier broadcast from aircraft in order to identify themselves. Icao for the most part stays the same compared to the other data fields leveraged from the flight data API. The only scenario where the Icao would change is if the aircraft registration number changes, which can sometimes happen when aircraft is sold to an owner within another country.

ii. demo-flight-geo.lookup

```
inputs:
-  lookupKey: 'demo-flight-geo'
   typeDiscovery: 'demo-flight-geo.td.xml'
```

iii. demo-flight-geo.repo

```
artifactModelFile: 'demo-flight-geo.am.xml'
```

```
typeAssociations:
-  type: 'demo-flight-geo'
   subtype: '*'
   label: '*'
```

iv. demo-flight-geo.td.xml

```
<?xml version="1.0"?>
<typeDiscovery xmlns="urn:dpf:ingest-artifact-
type-discovery">

    <properties>
        <document format="json"/>
    </properties>

    <typeMetadata>
        <type eval="literal">demo-flight-
geo</type>
        <subtype eval='literal'>JSON</subtype>
        <label eval="literal">demo-flight-
geo</label>
    </typeMetadata>
</typeDiscovery>
```

g. Create the following domain model files in a folder named 'demo-flight-geo-dom' in the /opt/common/models/tcri/ directory

i. demo-flight-geo.cd.xml

```
<?xml version="1.0" encoding="utf-8"?>
<concepts xmlns="urn:dpf:concept-predicate-
model">

    <concept>
        <key label="ARTIFACT_KEY"
model="tcri:demo-flight-geo"
dataformat="STRING"/>

        <description>The message
identifier</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="LATITUDE" model="tcri:demo-
flight-geo" dataformat="DOUBLE"/>

        <description>The aircraft's latitude
over the ground</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="LONGITUDE" model="tcri:demo-
flight-geo" dataformat="DOUBLE"/>
```

```
        <description>The aircraft's longitude
over the ground</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="ALTITUDE" model="tcri:demo-
flight-geo" dataformat="DOUBLE"/>

        <description>The aircraft's broadcasted
latitude in feet at standard
pressure</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="TRAK" model="tcri:demo-
flight-geo" dataformat="DOUBLE"/>

        <description>The aircraft's track angle
across the ground clockwish from 0 degrees
north</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="OP" model="tcri:demo-flight-
geo" dataformat="STRING"/>

        <description>The aircraft's operator
name</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>
    <concept>
        <key label="ICAO" model="tcri:demo-
flight-geo" dataformat="STRING"/>

        <description>The aircraft's broadcasted
six-digit hexadecimal unique
identifier</description>
        <security>U</security>
        <conceptType>ATTRIBUTE</conceptType>
    </concept>

</concepts>
```

ii. demo-flight-geo.depo

```
domainModelFiles:
- 'demo-flight-geo.dm.xml'

conceptDefinitions:
- 'demo-flight-geo.cd.xml'

predicateDefinitions:
- 'demo-flight-geo.pd.xml'
```

iii. demo-flight-geo.dm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<domainModel xmlns="urn:dpf:domain-model">

    <name>tcri:demo-flight-geo</name>

    <term handle="artifact_key">
        <concept label="ARTIFACT_KEY"
model="tcri:demo-flight-geo"
dataformat="STRING"/>
    </term>
    <term handle="latitude">
        <concept label="LATITUDE"
model="tcri:demo-flight-geo"
dataformat="DOUBLE"/>
    </term>
    <term handle="longitude">
        <concept label="LONGITUDE"
model="tcri:demo-flight-geo"
dataformat="DOUBLE"/>
    </term>
    <term handle="altitude">
        <concept label="ALTITUDE"
model="tcri:demo-flight-geo"
dataformat="DOUBLE"/>
    </term>
    <term handle="trak">
        <concept label="TRAK" model="tcri:demo-
flight-geo" dataformat="DOUBLE"/>
    </term>
    <term handle="op">
        <concept label="OP" model="tcri:demo-
flight-geo" dataformat="STRING"/>
    </term>
    <term handle="icao">
        <concept label="ICAO" model="tcri:demo-
flight-geo" dataformat="STRING"/>
    </term>

    <structuredObject handle="demo_flight_geo">
        <kind>demo_flight_geo</kind>
        <properties>
            <property>
                <key
handle="artifact_key">artifact_key</key>
                <value type="string"/>
            </property>
            <property>
                <key
handle="latitude">latitude</key>
                <value type="double"/>
            </property>
            <property>
                <key
handle="longitude">longitude</key>
                <value type="double"/>
            </property>
            <property>
                <key
handle="altitude">altitude</key>
```

```
                        <value type="double"/>
                    </property>
                    <property>
                        <key handle="trak">trak</key>
                        <value type="double"/>
                    </property>
                    <property>
                        <key handle="trak">op</key>
                        <value type="string"/>
                    </property>
                    <property>
                        <key handle="icao">icao</key>
                        <value type="string"/>
                    </property>
                </properties>
            </structuredObject>
        <!--
           <geoHints>
                <structuredObjectGeospatialHints>
                    <kind>demo_flight_geo</kind>
                    <hints>
                        <defaultGeometry
        type="point">wkt_geometry</defaultGeometry>

        <defaultDTG>valid_date_time</defaultDTG>

        <nearRealTimeCache>true</nearRealTimeCache>
                    </hints>
                </structuredObjectGeospatialHints>
            </geoHints>
        -->
            <simpleFeatureType>
                <name>demo_flight_geo</name>
                <nrtCache>true</nrtCache>
                <dateAttribute
        index="join">time</dateAttribute>
                <geometryAttribute
        type="point">geometry</geometryAttribute>
                <attributes>
                    <attribute
        type="string">icao</attribute>
                    <attribute
        type="string">op</attribute>
                    <attribute
        type="double">altitude</attribute>
                    <attribute
        type="double">trak</attribute>
                </attributes>
            </simpleFeatureType>

        </domainModel>
```

iv. demo-flight-geo.pd.xml

```
        <predicates xmlns="urn:dpf:concept-predicate-
        model">

        </predicates>
```

h. Change ownership of the demo-flight-geo model directories and their containing files to storm:storm

```
sudo chown -R storm:storm /opt/common/models/tcri/demo-flight-
geo-art/ /opt/common/models/tcri/demo-flight-geo-dom/
```

i. Become the "storm" user

```
[root@acs-dev1-node1 tcri]# su - storm
-bash-4.2$
```

j. Create directories in HDFS, for the flight data artifact and domain files

```
-bash-4.2$ hdfs dfs -mkdir /ingest/repository/artifact/demo-
flight-geo/
-bash-4.2$ hdfs dfs -mkdir /ingest/repository/domain/demo-
flight-geo/
```

k. Put the artifact and domain model files into their respective HDFS directories

```
-bash-4.2$ hdfs dfs -put -f /opt/common/models/tcri/demo-
flight-geo-art/* /ingest/repository/artifact/demo-flight-geo
-bash-4.2$ hdfs dfs -put -f /opt/common/models/tcri/demo-
flight-geo-dom/* /ingest/repository/domain/demo-flight-geo
```

l. Start the Streaming-Data-Processing-Framework Topology from the command line while still logged in as the Storm user

```
bash-4.2$ /opt/storm/topologies/tcri/streaming-dpf/bin/start-
topology.sh
```

5. Set up GeoEvent configuration - use GeoEvent UI, CLI scripts (every new definition)
   a. Set up GeoEvent Definitions:
      i. Run a preingest on <storm_master> to load the simple feature attributes into GeoEvent definitions

```
sudo /opt/common/util/tcri/utility/bin/dpf-
utility preingest
```

      ii. Browse to https://<geoevent_node>:6143/geoevent/manager/site.html
      iii. Under the 'GeoEvent Definitions' tab, there should be demo_flight_geo among the list of definitions
      iv. Click on the 'Edit' icon next to the demo_flight_geo definition
      v. Edit 'icao' and ensure it is set to the 'TRACK_ID' tag

vi. Edit 'time' and ensure it is set to the 'TIME_START' tag
vii. Edit 'geometry' and ensure it is set to the 'GEOMETRY' tag
viii. Click on 'Save'

b. Set up GeoEvent Inputs:
   i. Browse to https://<geoevent_node>:6143/geoevent/manager/inputs.html
   ii. Click on 'Add Input'
   iii. Search for 'Receive Features via Kafka' and click on the 'Select' button
   iv. Set 'Name' to 'kafka-kryo-in'
   v. Set 'Zookeeper Connection String' to <zookeeper>:2181
   vi. Set 'Topic Name' to 'GEO_ALL'
   vii. Set 'Consumer Group Id' to '1'
   viii. Click on 'Save'

c. Set up GeoEvent Outputs
   i. Set up data source output:
      1. Browse to https://<geoevent_node>:6143/geoevent/manager/outputs.html
      2. Click on 'Add Output'
      3. Search for 'Add a Feature to a Spatiotemporal Big Data Store' and click on 'Select'
      4. Set 'Name' to 'bds-out-demo-flight-geo'
      5. Set 'GeoEvent Definition' to 'demo_flight_geo' in the drop-down menu
      6. Set 'ArcGIS Server Connection' to 'Portal' in the drop-down menu
      7. For 'Data Source Name', click on 'Create Data Source' to configure
         a. Set 'Name' to 'demo_flight_geo'
         b. Check mark 'Data Retention Option' and set to '1 Hours'
         c. Set 'Display Field Name' to 'icao'
         d. Expand the 'Time Info' tab
         e. Set 'Time Interval' to '1'
         f. Set 'Time Interval Units' to 'Milliseconds'
         g. Check mark 'Has Live Data'
         h. Leave the rest of the configurations to their default
         i. Click on 'Publish'
      8. Click on 'Save'
   ii. Set up stream service output:
      1. Click on 'Add Output'
      2. Search for 'Send Features to a Stream Service' and click on 'Select'
      3. Set 'Name' to 'stream-service-out-demo-flight-geo'

4. Set 'GeoEvent Definition Name' to 'demo_flight_geo' in the drop-down menu
5. Set 'ArcGIS Server Connection' to 'Portal' in the drop-down menu
6. Set 'Folder' to 'Root' in the drop-down menu
7. For 'Stream Service Name', click on 'Publish Stream Service' to configure
    a. Set 'Name' to 'demo_flight_geo'
    b. Set 'Display' Field Name' to 'icao'
    c. Check mark 'Store Latest' and expand the tab
    d. Set 'Server' to 'Portal' in the drop-down menu
    e. Set 'Folder' to 'Root' in the drop-down menu
    f. Set 'Service to 'demo_flight_geo (FeatureServer)' in the drop-down menu
    g. Set 'Layer' to 'demo_flight_geo (0)' in the drop-down menu
    h. Set 'Update Interval (seconds)' to '1'
    i. Set 'Maximum Feature Age (minutes)' to '0'
    j. Check mark Related Features' and expand the tab
    k. Set 'Server' to 'Portal' in the drop-down menu
    l. Set 'Folder' to 'Root' in the drop-down menu
    m. Set 'Service to 'demo_flight_geo (FeatureServer)' in the drop-down menu
    n. Set 'Layer' to 'demo_flight_geo (0)' in the drop-down menu
    o. Set 'Field' to 'icao' in the drop-down menu
    p. Click on 'Publish'
8. Click on 'Save'

d. Set up GeoEvent Services
    i. Browse to https://<geoevent_node>:6143/geoevent/manager/models.html
    ii. Click on 'Add Service'
    iii. Set 'Service Name' to 'kafka-kryo-in-bds-out
    iv. Click on 'Create', which opens a model creation board
    v. Under the 'Inputs' box, drag the 'kafka-kryo-in' input to the board
    vi. Under the 'Outputs' box, drag the 'bds-out-demo-flight-geo' and 'stream-service-out-demo-flight-geo' outputs to the board
    vii. Under the 'New Elements' box, drag a 'Filter' between the 'kafka-kryo-in' input and 'bds-out-demo-flight-geo' output
        1. For Filter Properties, set 'Name' to 'demo-flight-geo'
        2. Click on the green plus sign to add expression

3. Set 'Definition' to 'demo_flight_geo' in the drop-down menu
4. Set 'Field' to '$DEFINTION_NAME' in the drop-down menu
5. Set 'Operator' to '=' in the drop-down menu
6. Set 'Definition Name' to 'demo_flight_geo' in the drop-down menu
7. Click on 'Ok'

viii. Under the 'New Elements' box, drag another 'Filter' between the 'kafka-kryo-in' input and 'stream-service-out-demo-flight-geo' output

1. For Filter Properties, set 'Name' to 'demo-flight-geo-stream'
2. Click on the green plus sign to add expression
3. Set 'Definition' to 'demo_flight_geo' in the drop-down menu
4. Set 'Field' to '$DEFINTION_NAME' in the drop-down menu
5. Set 'Operator' to '=' in the drop-down menu
6. Set 'Definition Name' to 'demo_flight_geo' in the drop-down menu
7. Click on 'Ok'

ix. Connect an arrow between the 'kafka-kryo-in' input and 'demo-flight-geo' filter

x. Connect an arrow between the 'demo-flight-geo' filter and 'bds-out-demo-flight-geo' output

xi. Connect an arrow between the 'kafka-kryo-in' input and 'demo-flight-geo-stream' filter

xii. Connect an arrow between the 'demo-flight-geo-stream' filter and stream-service-out-demo-flight-geo' output

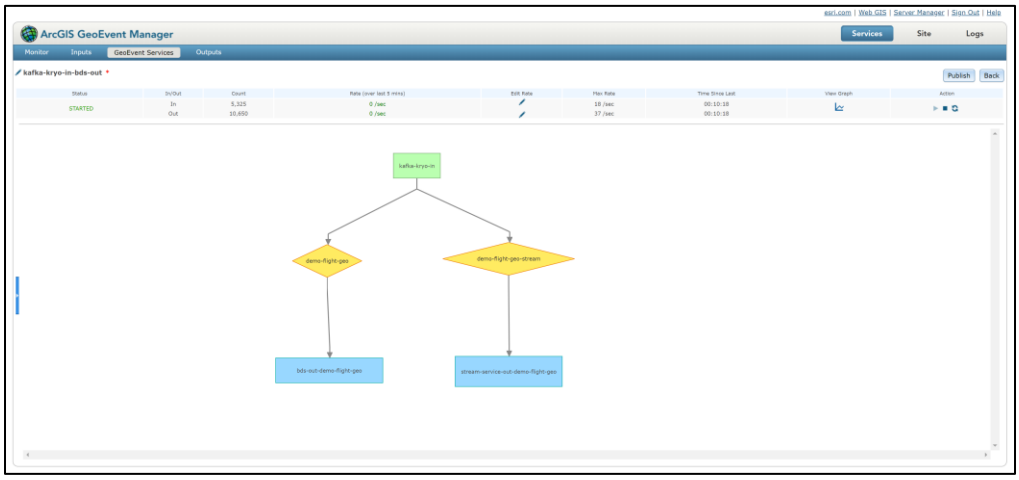


Figure 165 ArcGIS GeoEvent Manager

xiii. Click on 'Publish'

e. Verify GeoEvent configuration

i. Browse to the ArcGIS GeoEvent Manager at https://<geoevent_node>:6143/geoevent/manager/index.html

ii. Verify that the 'kafka-kryo-in-bds-out' service, 'kafka-kryo-in' input, 'bds-out-demo-flight-geo' and 'stream-service-out-demo-flight-geo' outputs are turned on (indicated by a green arrow to the left of each row)

6. Ingest batch data (DPF performs mapping of data to models) by running DPF utility script with batch data
   a. SSH to node 1
   b. Change directories to /opt/storm/topologies/tcri/streaming-dpf/

```
cd /opt/storm/topologies/tcri/streaming-dpf/
```

   c. Make a directory named 'sample'

```
sudo mkdir sample
```

   d. Change the 'sample' directory's ownership to storm:storm

```
sudo chown storm:storm sample/
```

   e. Download the flight data sample file locally, which contains 79 flight elements: https://confluence.di2e.net/download/attachments/241572456/flight_data_sample.json?api=v2
   f. SCP the 'flight_data_sample.json' file to the <storm_master>:/opt/storm/topologies/tcri/streaming-dpf/sample directory
   g. Change ownership of the 'flight_data_sample.json' file to storm:storm and privileges to '755'

```
sudo chown storm:storm sample/flight_data_sample.json && sudo
chmod 755 sample/flight_data_sample.json
```

   h. Become the storm user
   i. Ingest flight batch data with dpf-utility script

```
-bash-4.2$ /opt/common/util/tcri/utility/bin/dpf-utility
ingest -s kafka -l demo-flight-geo
/opt/storm/topologies/tcri/streaming-
dpf/sample/flight_data_sample.json
```

   j. Expect a successfully processed document as part of the batch ingest command line output

```
INFO  m.a.i.d.u.i.IngestCommand - Successfully processed 1
document(s) in 1002 millisecond(s)
```

```
Successfully processed 1 document(s) in 1002 millisecond(s)
```

7. Verify that the number of items ingested by GeoEvent matches the expected number of items from the batch job
    a. Browse to the ArcGIS GeoEvent Manager at https://<geoevent_node>:6143/geoevent/manager/index.html
    b. Observe counts to increment for GeoEvent services, inputs, and outputs
8. Verify that flight data publishes to the GEO_ALL Kafka topic for each batch ingest
    a. SSH to Kafka broker
    b. Use the kafka-console-consumer script to browse published data to the Kafka GEO_ALL topic

```
/opt/kafka/kafka-current/bin/kafka-console-consumer.sh --
zookeeper <node1>,<node2>,<node3> --topic GEO_ALL
```

    c. Observe published data on the GEO_ALL topic for each batch ingest
9. Verify that Storm topologies (Streaming DPF, SO_Persistence, etc.) have bolts that interact with the databases during batch ingest
    a. Browse to the Storm web UI at <storm_master_node>:8080/index.html
    b. Click on the 'Streaming-Data-Processing-Framework' topology link under 'Topology Visualization'
    c. Observe that numbers associated with the edges between the Kafka-Reader.dpf spout, Processing bolt, and GeoSOPersistenceBolt increment

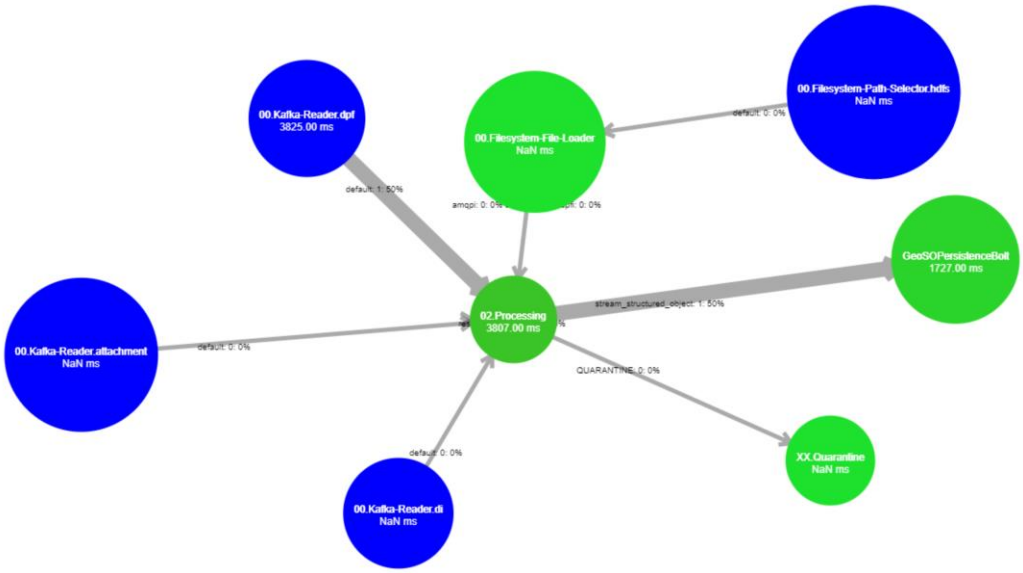


Figure 166 Kafka-Reader.dpf spout

d. Observe that the numbers associates with the "XX.Quarantine" bolt's incoming edge is not incrementing

10. Run SOQL query on SOQL web interface or SO client to verify SO persistence during batch ingest
    a. With SOQL web interface:
        i. Browse to the SOQL Web UI at https://<so_web_server>:8443/sol-ws/query.html
        ii. Type the following into the query box

```
SELECT * FROM 'so:///tcri/demo_flight_geo'
```

        iii. Set 'Number of Results' to '100'
        iv. Click on 'Submit'
        v. Search for string 'icao' with Ctrl+F
        vi. Expect 79 search results
    b. With SO client:
        i. SSH into node 1
        ii. Execute the following command

```
sudo /opt/sol/cli/bin/soclient query -n tcri -q
"SELECT * FROM 'so:///tcri/demo_flight_geo'"
```

        iii. Expect 25 queried results (soclient cap)

```
19:43:00.185 INFO  m.t.s.c.c.QueryCommand - 25
results counted in 0.399 seconds
19:43:00.185 INFO  m.t.s.c.c.QueryCommand -
62.65664160401002 results per second.
```

11. Ingest streaming data (DPF performs mapping of data to models) by turning on the NiFi workflow
    a. Browse to the NiFi Web UI at <nifi_node>:6434/nifi
    b. Shift select all three processors in the demo_flight_geo NiFi dataflow template and click on the 'Start' icon
    c. Expect the GetHTTP processor's Out to increment up, DPFInputWrapper processor's In/Out to increment up, and PublishKafka processor's In to increment up
    d. Leave stream on for verification steps below
12. Verify that the number of items ingested are continuously incrementing in GeoEvent
    a. Browse to the ArcGIS GeoEvent Manager at https://<geoevent_node>:6143/geoevent/manager/index.html
    b. Expect counts to increment for GeoEvent services, inputs, and outputs

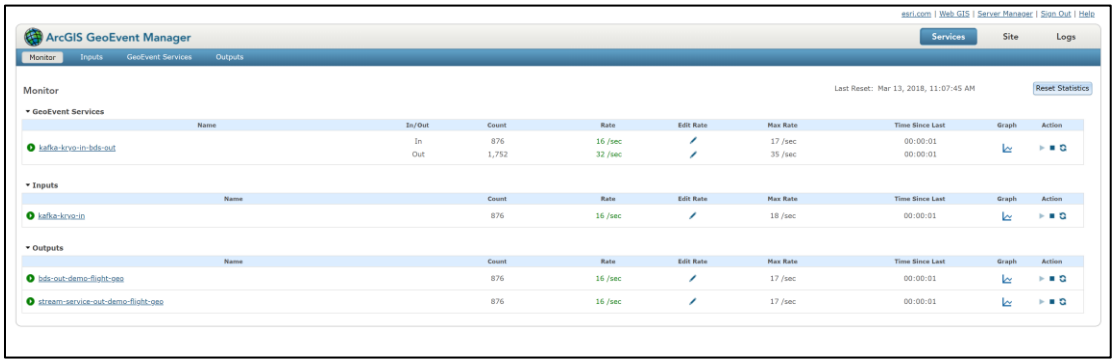


Figure 167 GeoEvent services, inputs, and outputs

13. Verify that flight data continuously publishes to the GEO_ALL Kafka topic while the stream runs
    a. SSH to node 3 Kafka broker
    b. Use the kafka-console-consumer script to browse published data to the Kafka GEO_ALL topic

```
/opt/kafka/kafka-current/bin/kafka-console-consumer.sh --
zookeeper <node1>,<node2>,<node3> --topic GEO_ALL
```

    c. Expect continuously published data on the GEO_ALL topic for the duration of the stream
14. Verify that Storm topologies (Streaming DPF, SO_Persistence, etc.) have bolts that interact with the databases during stream ingest

a. Browse to the Storm Web UI at <storm_master_node>:8080/index.html
b. Click on the 'Streaming-Data-Processing-Framework' topology link under 'Topology Summary'
c. Expect data only continuously passes through the Kafka-Reader.dpf spout, Processing bolt, and GeoSOPersistenceBolt

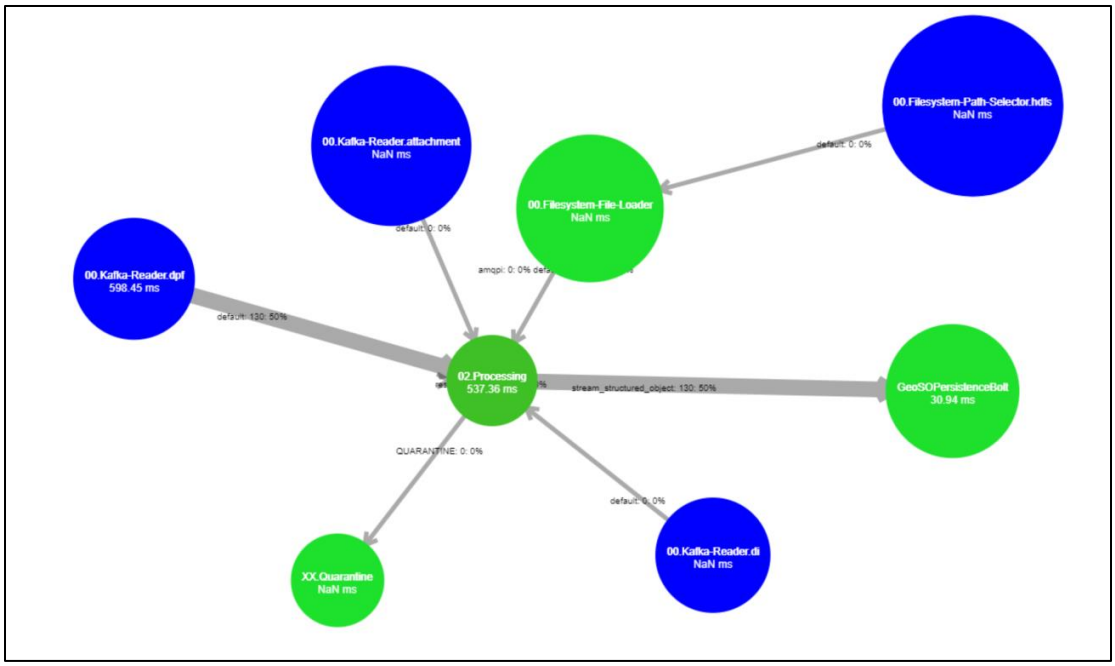


Figure 168 Kafka-Reader.dpf spout, Processing bolt, and GeoSOPersistenceBolt

15. Run SOQL query with SOQL web interface or SO client to verify SO persistence during stream ingest
    a. With SOQL web interface:
        i. Browse to the SOQL Web UI at https://<so_web_server>:8443/sol-ws/query.html
        ii. Type the following into the query box

```
SELECT * FROM 'so:///tcri/demo_flight_geo'
```

        iii. Set 'Number of Results' to '500'
        iv. Click on 'Submit'
        v. Search for string 'icao' with Ctrl+F
        vi. Expect many search results
    b. With SO client:
        i. SSH into node 1
        ii. Execute the following command

```
sudo /opt/sol/cli/bin/soclient query -n tcri -q
"SELECT * FROM 'so:///tcri/demo_flight_geo'"
```

iii. Expect 25 queried results (soclient cap)

```
19:43:00.185 INFO  m.t.s.c.c.QueryCommand - 25
results counted in 0.399 seconds
19:43:00.185 INFO  m.t.s.c.c.QueryCommand -
62.65664160401002 results per second.
```

16. Visualize the streaming data in an ArcGIS geospatial map
    a. Go to Portal My Content page at http://<portal_node>.sd.spawar.navy.mil/portal/home/content.html
    b. Click on the down arrow next to demo_flight_geo Feature Layer with the Stream Service icon (contains a blue arrow)
    c. Click on Add layer to new map
    d. There should be real-time streaming flight coordinates around 100 km of the Phoenix, AZ airport. Each geometric point should be mapped to a unique icao, signaling that each point is a unique flight

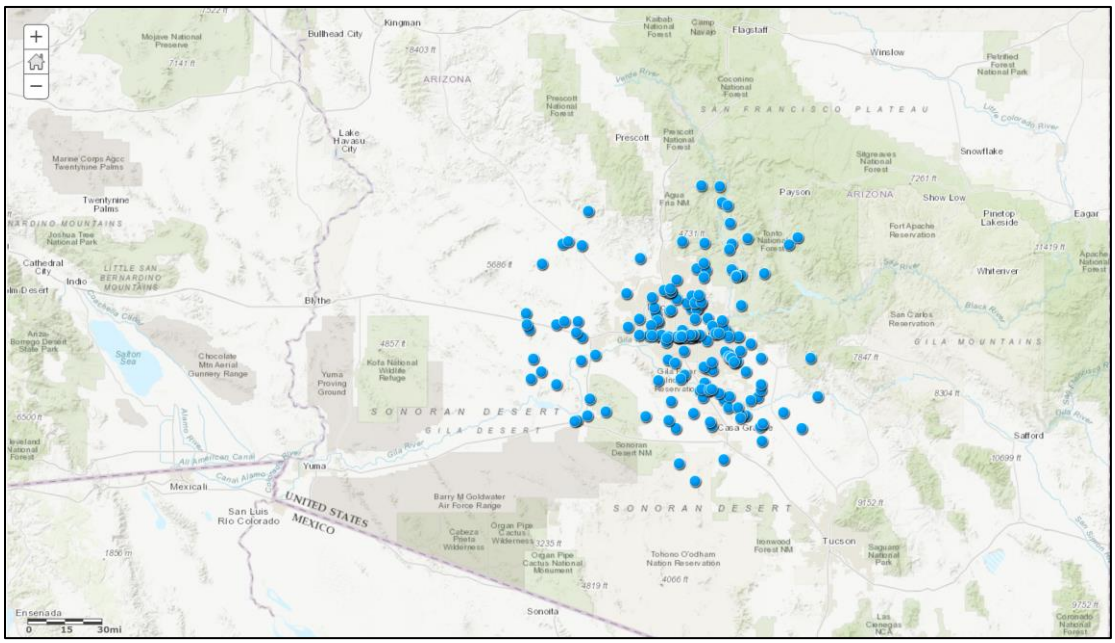


Figure 169 Phoenix, AZ Airport

APPENDIX B ACS TWITTER DEMO

OVERVIEW

> **NiFi upgrade**
> This demo has been adjusted for the NiFi 1.1.2 upgrade.

This demo was initially created by Viral Bhalodia, with the goal of showcasing ACS's basic data ingest functionality.

The main tools that will be used:

- NiFi: Used for importing the data and automating the data workflow in its web interface.
- Hadoop HDFS: Used to store the data distributively, where the data can be accessed by Yarn and MapReduce jobs for further data analysis

In this case, a sample of Twitter data will be used with some keyboard parameters to search for. ACS will save the data in JSON format and will store the data locally (/hadoop/myPutFiles) and on hdfs (/nifi/tweets).

This demo was conducted on the NiFi node within an 8 node ACS cluster (acs-8cluster-dev1-node1).

IMPORT TWITTER DATA INTO ACS

This demo assumes that the Twitter XML template has already been downloaded using the Twitter API. The Twitter XML template can be downloaded at https://confluence.di2e.net/download/attachments/241572456/Demo_Twitter_Files_System_Plus_HDFS.xml?api=v2. Download the Twitter XML template to any location that is easy to find, such as the desktop.

Navigate to the NiFi web interface on your browser, which should be the first node on your cluster. Example: http://acs-8cluster-dev1-node1:8081/nifi/.

Load the template into NiFi by clicking on the **Upload Template** button inside the **Operate** box, located on the left-hand side.

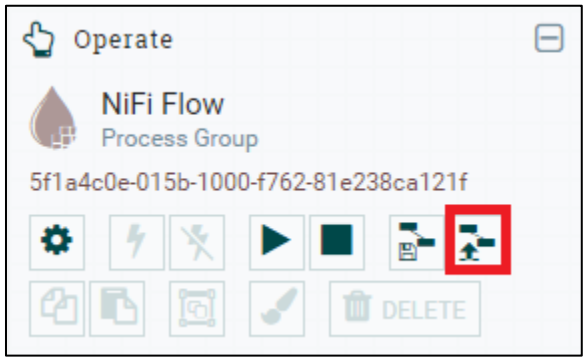


Figure 170 NiFi Flow

Browse for the template where it is saved and click **Upload**.

Once the template imports into NiFi, click on the **Template** button on the upper left-hand side and drag it down to the grid space below.

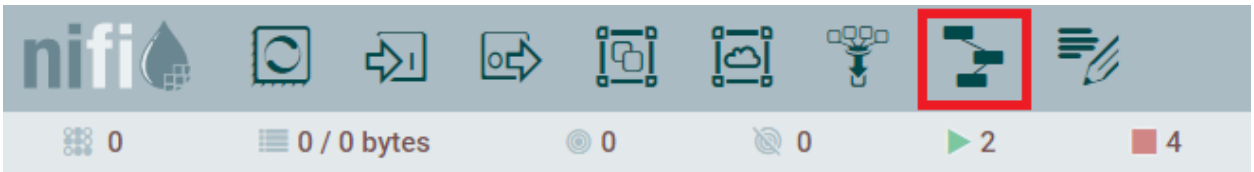


Figure 171 Import into NiFi

There should now be a flow diagram that can be configured.

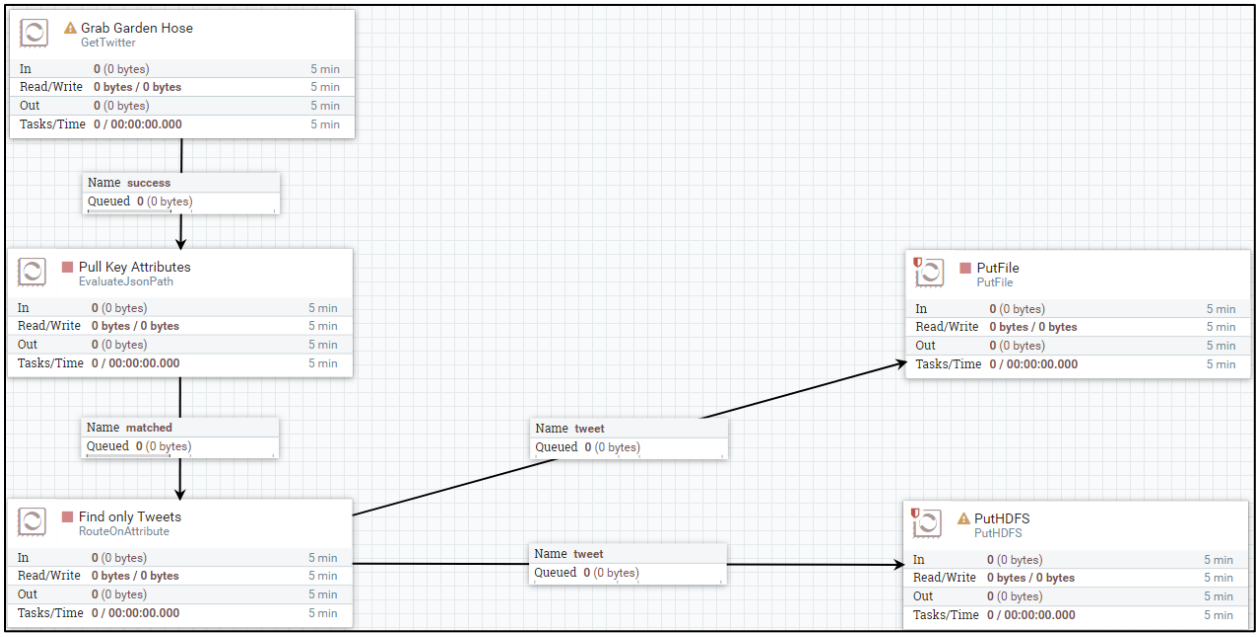


Figure 172 Flow Diagram

CONFIGURE DATA WORKFLOW

Configure Twitter Garden Hose by right-clicking on the **Grab Garden Hose** box. Navigate to configure and then the properties tab.

Fill in the consumer secret and access token secrets, which are created when setting up a Twitter Developers account. If you don't have a Twitter Developers account, use the pre-existing keys in the ACS3-722 ticket.

Consumer secret: F5f3Td73AYXQuu6L4tQtt1Dbe75YGlFkn8IkOHqWmXFoSdTk6U

Access token secret: qeZ9xEduzA6K2gYL21TrfvRZ6Q79z02whaZHpTjClc3Hq

Fill in **Terms to Filer On** with the words that Twitter will query. For example, we can search for all Twitter post instances of "submarine" and "military".

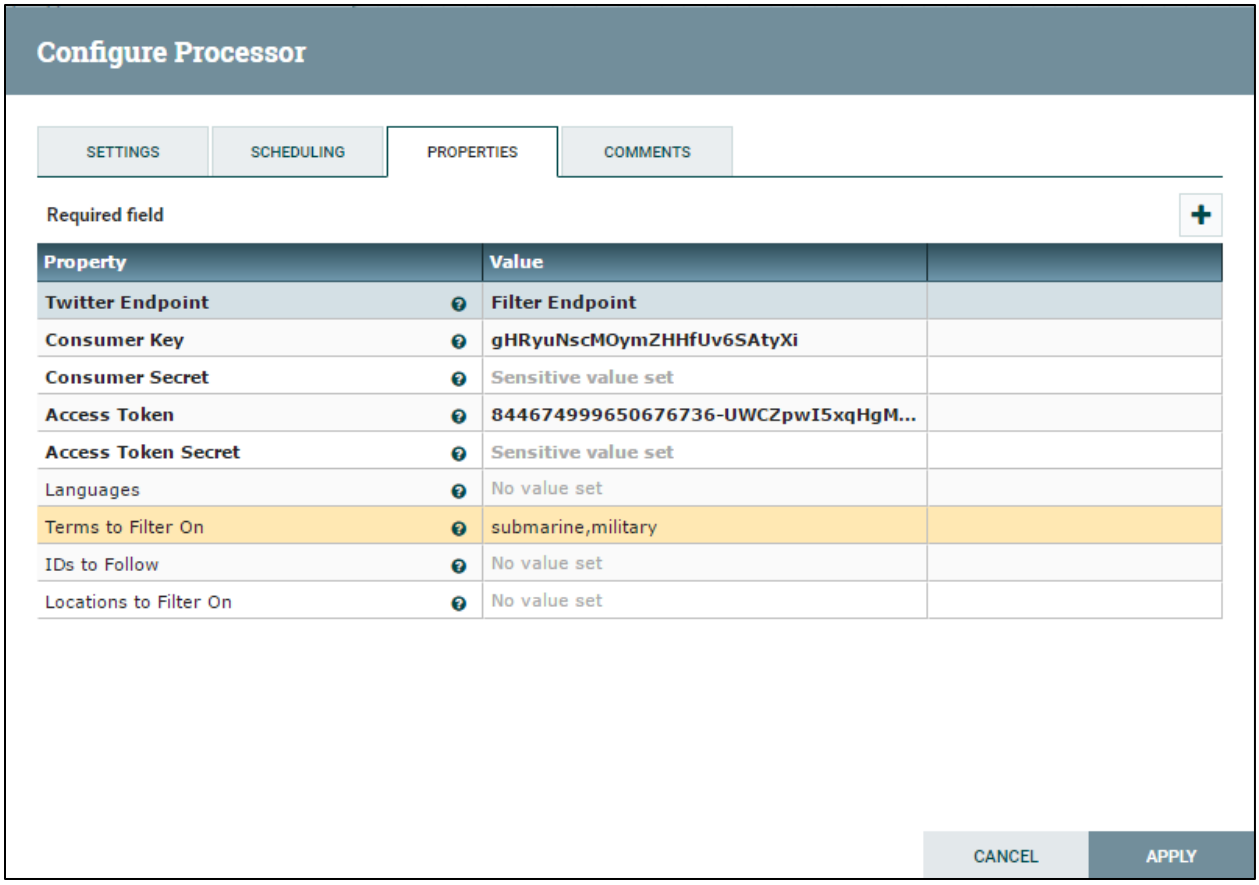


Figure 173 Twitter post instances

HDFS processor

Configure the HDFS processor by right-clicking on the **putHDFS** box. Navigate to configure and then the properties tab.

Under the **Properties** tab, point **Hadoop Configuration Resources** to hdfs-site.xml and core-site.xml on node 1. Separate the file locations under **Hadoop Configuration Resources** with a comma. The files were located under /etc/hadoop/conf on acs-8cluster-dev1-node1.

Hadoop Configuration Resources: /etc/hadoop/conf/hdfs-site.xml,/etc/hadoop/conf/core-site.xml

hdfs-site.xml vs core-site.xml

hdfs-site.xml: Contains the configurations for all the HDFS daemons. These daemons include the NameNode, secondary NameNode, and DataNodes.

core-site.xml: Tells Hadoop where NameNode will run on the cluster. The core-site.xml file also contains the configuration settings for Hadoop Core, such as I/O settings.

Code Block 15 /etc/hadoop/conf/

```
[root@acs-8cluster-dev1-node1 conf]# ls | grep site.xml
core-site.xml
hdfs-site.xml
mapred-site.xml
mapred-site.xml.template
yarn-site.xml
```

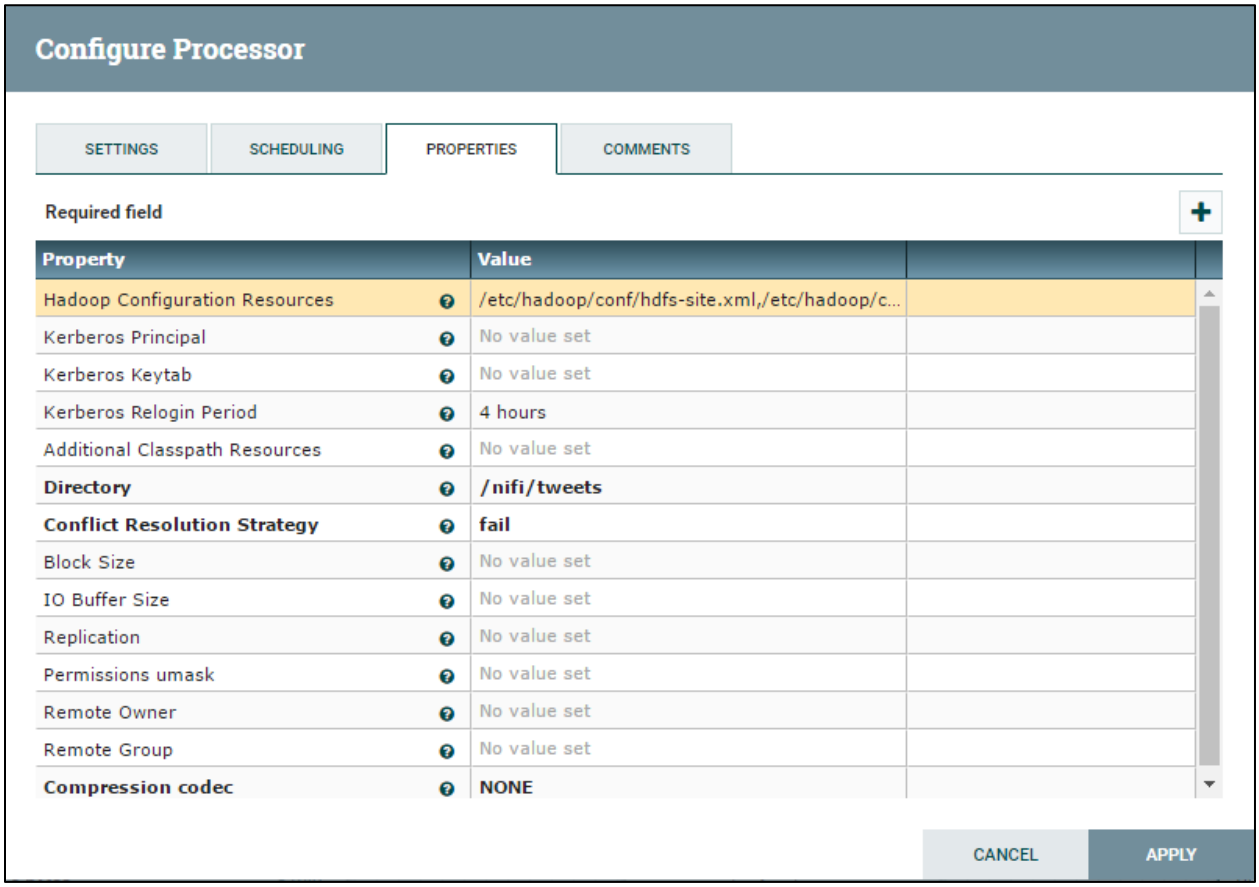


Figure 174 Configure Processor

Data Storage Directory in HDFS

Configure the data storage directory in HDFS where all the Twitter JSON data will be dumped to. For the purpose of this demo, we name the directory /nifi/tweets/.

Log into the hdfs user on node 1.

```
[root@acs-8cluster-dev1-node1 conf]# su - hdfs
Last login: Mon Apr 17 14:43:09 EDT 2017 on pts/0
-bash-4.2$
```

Create the /nifi/tweets/ directory and assign full permissions on the directory for demo purposes. Stricter permission settings may be used for more realistic data sets in a production environment.

```
-bash-4.2$ hadoop fs -mkdir -p /nifi/tweets
-bash-4.2$ hadoop fs -chmod 777 /nifi/tweets
```

VERIFY TWITTER DATA IS INGESTED LOCALLY AND ON HDFS

Within NiFi, right-click on each processor box and hit the green **Start** arrow in the following order:

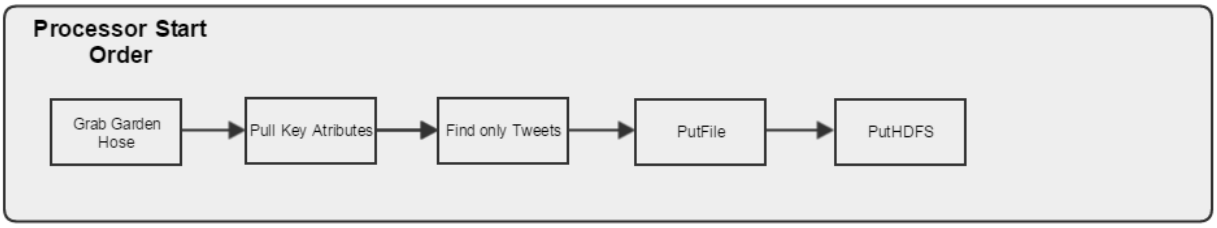


Figure 175 Verify Twitter data

After starting each processor in NiFi, right-click outside of the processor diagram and click on **Refresh status**.

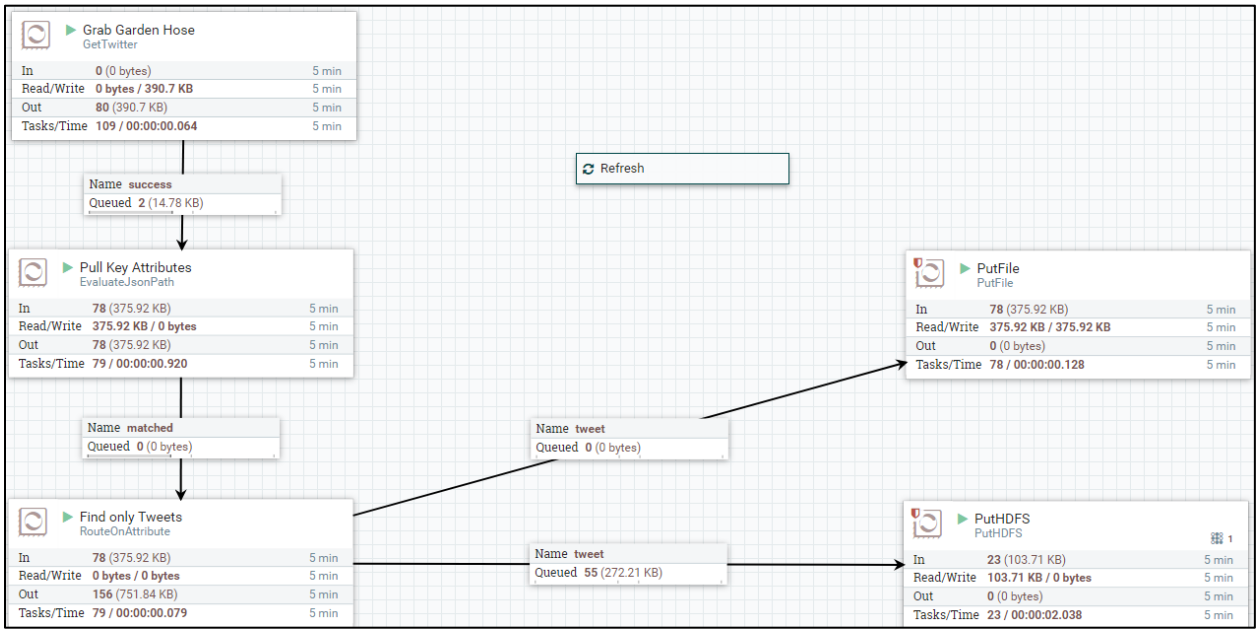


Figure 176 Start NiFi Processors

Once all the processors are initialized in NiFi, you should see Twitter JSON data saved under /hadoop/myPutsFiles on node 1's local system and under /nifi/tweets/ in hdfs. Depending on the Twitter search query, there should be many JSON files stored under these directories once all the processors are executed, but for visual sake we can just list the first 10 data files in these directories with the head command.

Code Block 16 local

```
[root@acs-8cluster-dev1-node1 conf]# ls /hadoop/myPutFiles/ | head -10
279371777981992.json
279371812402981.json
279371813512653.json
279371820265754.json
279372822295906.json
279372823374774.json
279372824243649.json
279372825444792.json
279372826275326.json
279373827303410.json
```

Code Block 17 hdfs

```
[root@acs-8cluster-dev1-node1 conf]# hdfs dfs -ls /nifi/tweets/ | head -10
Found 6778 items
-rw-r--r--   3 root hadoop       7041 2017-04-13 16:41
/nifi/tweets/279371777981992.json
-rw-r--r--   3 root hadoop       6891 2017-04-13 16:41
/nifi/tweets/279371812402981.json
-rw-r--r--   3 root hadoop       5081 2017-04-13 16:41
/nifi/tweets/279371813512653.json
-rw-r--r--   3 root hadoop       3395 2017-04-13 16:41
/nifi/tweets/279371820265754.json
-rw-r--r--   3 root hadoop       2565 2017-04-13 16:41
/nifi/tweets/279372822295906.json
-rw-r--r--   3 root hadoop       2207 2017-04-13 16:41
/nifi/tweets/279372823374774.json
-rw-r--r--   3 root hadoop       2574 2017-04-13 16:41
/nifi/tweets/279372824243649.json
-rw-r--r--   3 root hadoop       5273 2017-04-13 16:41
/nifi/tweets/279372825444792.json
-rw-r--r--   3 root hadoop       3627 2017-04-13 16:41
/nifi/tweets/279372826275326.json
```

Once a sufficient amount of data is grabbed, right-click on each processor box in NiFi and hit the **Stop** button.

APPENDIX C CREATE A CUSTOM NIFI PROCESSOR

OVERVIEW

Although NiFi comes with an extensive list of pre-built processors, there is still a need to build custom processors for unique situations. These situations may range from interacting with a specific REST API or converting non-standard file formats. NiFi processors are built by Maven as a NAR file, which is a zipped container file that consists of XML metadata and other bundled dependencies. All NiFi processors are written in Java and rely on dependencies that are configured in a top level pom.xml file. Bundled NAR files are placed in the nifi/lib directory for deployment.

PREREQUISITES

- Java
- Latest Maven build
- Access to a dev environment that is preferably independent from an ACS cluster, but can SCP files over to an ACS NiFi node for deployment

Build the binary version of Maven instead of the yum install, as the old Maven version from yum has missing dependencies. Instructions on how to install the binary version of Maven can be found at https://maven.apache.org/install.html.

JSON PROCESSOR BUILD

This basic processor is designed to contain some minimal functionality when interacting with json data. The json processor is designed to filter out json string fields from an incoming flowfile when passed a json path as input.

Use Maven Archetype for NiFi

Input the following maven command within the command line and fill in the information below when prompted.

```
mvn archetype:generate -DarchetypeGroupId=org.apache.nifi -
DarchetypeArtifactId=nifi-processor-bundle-archetype -DarchetypeVersion=1.1.2 -
DnifiVersion=1.1.2
```

```
Define value for property 'groupId': acs.nifi
Define value for property 'artifactId': nifi-examples-bundle
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'artifactBaseName': examples
```

```
Define value for property 'package' rocks.nifi.processors.examples: :
[INFO] Using property: nifiVersion = 1.1.2
```

Pom.xml

Take note to implement version dependencies that align with the versions used within your ACS instance. Dependency versions can be found at https://mvnrepository.com/artifact/org.apache.nifi. Also, set maven.compiler.source and maven.compiler.target under the properties section to the java version running on the system.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  Licensed to the Apache Software Foundation (ASF) under one or more
  contributor license agreements. See the NOTICE file distributed with
  this work for additional information regarding copyright ownership.
  The ASF licenses this file to You under the Apache License, Version 2.0
  (the "License"); you may not use this file except in compliance with
  the License. You may obtain a copy of the License at
  http://www.apache.org/licenses/LICENSE-2.0
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.apache.nifi</groupId>
        <artifactId>nifi-nar-bundles</artifactId>
        <version>1.1.2</version>
    </parent>

    <groupId>acs.nifi</groupId>
    <artifactId>nifi-examples-bundle</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>nar</packaging>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.7</maven.compiler.source>
        <maven.compiler.target>1.7</maven.compiler.target>
        <nifi.version>1.1.2</nifi.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.nifi</groupId>
            <artifactId>nifi-api</artifactId>
            <version>${nifi.version}</version>
        </dependency>
```

```
        <dependency>
            <groupId>org.apache.nifi</groupId>
            <artifactId>nifi-utils</artifactId>
            <version>${nifi.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.nifi</groupId>
            <artifactId>nifi-processor-utils</artifactId>
            <version>${nifi.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-io</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>com.jayway.jsonpath</groupId>
            <artifactId>json-path</artifactId>
            <version>1.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.nifi</groupId>
            <artifactId>nifi-mock</artifactId>
            <version>${nifi.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.nifi</groupId>
                <artifactId>nifi-nar-maven-plugin</artifactId>
                <version>1.0.0-incubating</version>
                <extensions>true</extensions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.15</version>
            </plugin>
        </plugins>
    </build>

</project>
```

Point NiFi to Correct Class

This step tells NiFi which class to load. Navigate to /src/main/resources/META-INF/services/org.apache.nifi.processor.Processor and configure the processor to match the custom class name, which is ACSJsonPath in this case.

```
acs.nifi.processors.examples.ACSJsonPath
```

Configure Custom Processor Java File

This is the actual processor file that we will configure. Example path: nifi-examples-bundle/nifi-examples-processors/src/main/java/acs/nifi/processors/examples/ACSJsonPath.java

Below is an ACSJsonPath.java code sample.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package acs.nifi.examples.processors;

import com.jayway.jsonpath.JsonPath;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.util.concurrent.atomic.AtomicReference;
import org.apache.commons.io.IOUtils;
import org.apache.nifi.components.PropertyDescriptor;
import org.apache.nifi.flowfile.FlowFile;
import org.apache.nifi.processor.AbstractProcessor;
import org.apache.nifi.processor.ProcessContext;
import org.apache.nifi.processor.ProcessSession;
import org.apache.nifi.processor.ProcessorInitializationContext;
import org.apache.nifi.processor.Relationship;
import org.apache.nifi.annotation.behavior.SideEffectFree;
import org.apache.nifi.annotation.documentation.CapabilityDescription;
import org.apache.nifi.annotation.documentation.Tags;
import org.apache.nifi.processor.exception.ProcessException;
import org.apache.nifi.processor.io.InputStreamCallback;
import org.apache.nifi.processor.io.OutputStreamCallback;
import org.apache.nifi.processor.util.StandardValidators;

@SideEffectFree
@Tags({"JSON", "ACS", "path"})
@CapabilityDescription("Fetch value from json path.")
public class ACSJsonPath extends AbstractProcessor {

    private List<PropertyDescriptor> properties;
    private Set<Relationship> relationships;

    public static final String MATCH_ATTR = "match";

    public static final PropertyDescriptor JSON_PATH = new
PropertyDescriptor.Builder()
            .name("Json Path")
            .required(true)
            .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
```

```
            .build();

    public static final Relationship SUCCESS = new Relationship.Builder()
            .name("SUCCESS")
            .description("Succes relationship")
            .build();

    @Override
    public void init(final ProcessorInitializationContext context){
        List<PropertyDescriptor> properties = new ArrayList<>();
        properties.add(JSON_PATH);
        this.properties = Collections.unmodifiableList(properties);

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

    @Override
    public void onTrigger(final ProcessContext context, final ProcessSession
session) throws ProcessException {

        final AtomicReference<String> value = new AtomicReference<>();

        FlowFile flowfile = session.get();

        session.read(flowfile, new InputStreamCallback() {
            @Override
            public void process(InputStream in) throws IOException {
                try{
                    String json = IOUtils.toString(in);
                    String result = JsonPath.read(json,
context.getProperty(JSON_PATH).getValue()).toString();
                    value.set(result);
                }catch(Exception ex){
                    ex.printStackTrace();
                    getLogger().error("Failed to read json string.");
                }
            }
        });

        // Write the results to an attribute
        String results = value.get();
        if(results != null && !results.isEmpty()){
            flowfile = session.putAttribute(flowfile, "match", results);
            getLogger().error("ACSDEBUG: found a match");
        }

        // To write the results back out ot flow file
        flowfile = session.write(flowfile, new OutputStreamCallback() {

            @Override
            public void process(OutputStream out) throws IOException {
                try{
                    out.write(value.get().getBytes());
                }catch(Exception ex){
                    ex.printStackTrace();
                    getLogger().error("ACSDEBUG: write to flow file");
                }
            }
        });
```

```
        session.transfer(flowfile, SUCCESS);
    }

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

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

}
```

Below are some explanations that break down the NiFi Processor API and how it's implemented in the ACSJsonPath processor. More information on the NiFi Processor API can be found at https://nifi.apache.org/developer-guide.html.

Abstract Processor Class

ACSJsonPath is an extension of the AbstractProcessor. The AbstractProcessor, according to the Processor API, is the base class that is implemented by a majority of standard and custom processors. the AbstractProcessor class provides the framework that allows developers to build custom processors conveniently.

```
public class ACSJsonPath extends AbstractProcessor
```

Tag and Capability Documentation

The org.apache.nifi.annotations.documentation package provides the Java annotations functionality for documentating components. The @Tags and @CapabilityDescription are implemented directly in the NiFi processor UI and can be used when working with the processor in the NiFi web UI. @Tags is an array of strings for values that can be searched for when attempting to search for the ACSJsonPath processor. For instance, a NiFi developer or sysadmin can search for the words "JSON" or "ACS" in the NiFi processor search bar, and the ACSJsonPath processor will appear in the UI. The @CapabilityDescription value shows when searching for the ACSJsonPath processor as well. The value is intended to be a short description of the processor's functionality and the purpose it fulfills.

```
@SideEffectFree
@Tags({"JSON", "ACS", "path"})
@CapabilityDescription("Fetch value from json path.")
```

Documenting Properties

The property descriptor allows the developer to provide processor documentation directly within the NiFi UI when the processor is loaded into the grid area for creating data flows. The property

descriptor "Json Path" can be accessed by a drop-down field within the processor UI. Even though this particular processor does not implement property descriptions, a descriptions field can be added to provide additional information on how each property descriptor is implemented.

```
public static final PropertyDescriptor JSON_PATH = new
PropertyDescriptor.Builder()
            .name("Json Path")
            .required(true)
            .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
            .build();
```

Documenting Relationships

Calling relationships is similar to the way properties are called. Descriptions for "SUCCESS" and/or "FAILURE" relationships are implemented here and are useful when connecting the ACSJsonPath processor to other processors within a data flow. For instance, ACSJsonPath is connected to a GetHTTP and PutFile processor with "SUCCESS" relationships within a data flow for flight data, indicating that json-formatted data successfully filtered by a specified json path is routed throughout the whole data flow.

```
public static final Relationship SUCCESS = new Relationship.Builder()
            .name("SUCCESS")
            .description("Succes relationship")
            .build();
```

Property Validation

The return value for these methods is a collection of objects that spot any problems during processor validation. In terms of the ACSJsonPath processor, the property and relationships must have unmodified lists and sets in order to pass code validation.

```
public void init(final ProcessorInitializationContext context){
        List<PropertyDescriptor> properties = new ArrayList<>();
        properties.add(JSON_PATH);
        this.properties = Collections.unmodifiableList(properties);

        Set<Relationship> relationships = new HashSet<>();
        relationships.add(SUCCESS);
        this.relationships = Collections.unmodifiableSet(relationships);
```

Performing Work When Triggered

This is the section where work is processed, or the logic needed to successfully search a data field from an incoming json data flow. Work is only scheduled when calling the onTrigger method and when a flowfile exists for work to be acted upon. onTrigger takes the ProcessContext and ProcessSession arguments. The first step of the onTrigger method is to fetch a flowfile if it exists by calling the get method of the ProcessSession argument. In practice, the onTrigger method is only invoked when the ACSJsonPath processor is initialized, or scheduled to run. If a connection is made, the processor will attempt to convert whichever json path is

specified to string format, regardless of what data type is applied. If the processor is successful, it will log "ACSDEBUG: found a match". Otherwise, the processor will catch any processing errors and will log "Failed to read json string.".

```
public void onTrigger(final ProcessContext context, final ProcessSession session)
throws ProcessException {

        final AtomicReference<String> value = new AtomicReference<>();

        FlowFile flowfile = session.get();

        session.read(flowfile, new InputStreamCallback() {
            @Override
            public void process(InputStream in) throws IOException {
                try{
                    String json = IOUtils.toString(in);
                    String result = JsonPath.read(json,
context.getProperty(JSON_PATH).getValue()).toString();
                    value.set(result);
                }catch(Exception ex){
                    ex.printStackTrace();
                    getLogger().error("Failed to read json string.");
                }
            }
        });


        // Write the results to an attribute
        String results = value.get();
        if(results != null && !results.isEmpty()){
            flowfile = session.putAttribute(flowfile, "match", results);
            getLogger().error("ACSDEBUG: found a match");
        }

        // To write the results back out ot flow file
        flowfile = session.write(flowfile, new OutputStreamCallback() {

            @Override
            public void process(OutputStream out) throws IOException {
                try{
                    out.write(value.get().getBytes());
                }catch(Exception ex){
                    ex.printStackTrace();
                    getLogger().error("ACSDEBUG: write to flow file");
                }
            }
        });

        session.transfer(flowfile, SUCCESS);
    }
```

Configure Test Processor Java File

This test file conducts unit testing on the processor configuration and ensures that the processor can build correctly. Example path: nifi-examples-bundle/nifi-examples-processors/src/test/java/acs/nifi/processors/examples/ACSJsonPathTest.java

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package acs.nifi.examples.processors;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;
import org.apache.commons.io.IOUtils;
import org.apache.nifi.util.MockFlowFile;
import org.apache.nifi.util.TestRunner;
import org.apache.nifi.util.TestRunners;
import static org.junit.Assert.*;

public class ACSJsonPathTest {

    /**
     * Test of onTrigger method, of class ACSJsonPath.
     */
    @org.junit.Test
    public void testOnTrigger() throws IOException {
        // Content to be mock a json file
        InputStream content = new ByteArrayInputStream("{\"hello\":\"This is an
ACS unit test\"}".getBytes());

        // Generate a test runner to mock a processor in a flow
        TestRunner runner = TestRunners.newTestRunner(new ACSJsonPath());

        // Add properties
        runner.setProperty(ACSJsonPath.JSON_PATH, "$.hello");

        // Add the content to the runner
        runner.enqueue(content);

        // Run the content in queue, it also takes an int = number of contents
queued
        runner.run(1);

        // All results were processed without failure
        runner.assertQueueEmpty();

        // If you need to read or do additional tests on results, you can access
the content
        List<MockFlowFile> results =
runner.getFlowFilesForRelationship(ACSJsonPath.SUCCESS);
        assertTrue("1 match", results.size() == 1);
        MockFlowFile result = results.get(0);
        String resultValue = new String(runner.getContentAsByteArray(result));
        System.out.println("Match: " +
IOUtils.toString(runner.getContentAsByteArray(result)));

        // Test attributes and content
        result.assertAttributeEquals(ACSJsonPath.MATCH_ATTR, "This is an ACS unit
test");
        result.assertContentEquals("This is an ACS unit test");

    }
```

```
}
```

NiFi eases the process of creating unit tests by leveraging its own unit test framework with JUnit and nifi-mock packages. In the ACSJsonPathTest class, a ByteArrayInputStream is used to mock a a json file. Subsequently, a test runner is generated to mock the processor that is to be unit tested. Some assertion statements are added towards the end of the unit test to verify that an input stream with given attributes run successfully through the mock processor. These dependencies are called within the pom.xml, as listed below.

```
<dependency>
    <groupId>org.apache.nifi</groupId>
    <artifactId>nifi-mock</artifactId>
    <version>${nifi.version}</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.10</version>
    <scope>test</scope>
</dependency>
```

Build NAR File with Maven

Build the NAR file in the same directory with the top level pom.xml file.

```
mvn clean install
```

Deploy NAR File

SCP over the nar file into NiFi's lib directory. If NiFi is running as a cluster, copy the nar file to NiFi's lib directory for each NiFi node.

```
sudo scp nifi-helloworld-nar/target/nifi-examples-bundle-1.0-SNAPSHOT.nar
root@<nifi_node>:/opt/nifi/default/lib
```

SSH into the NiFi node(s). Assign execute permissions on the nar file for the user and change ownership of the nar file to nifi:nifi.

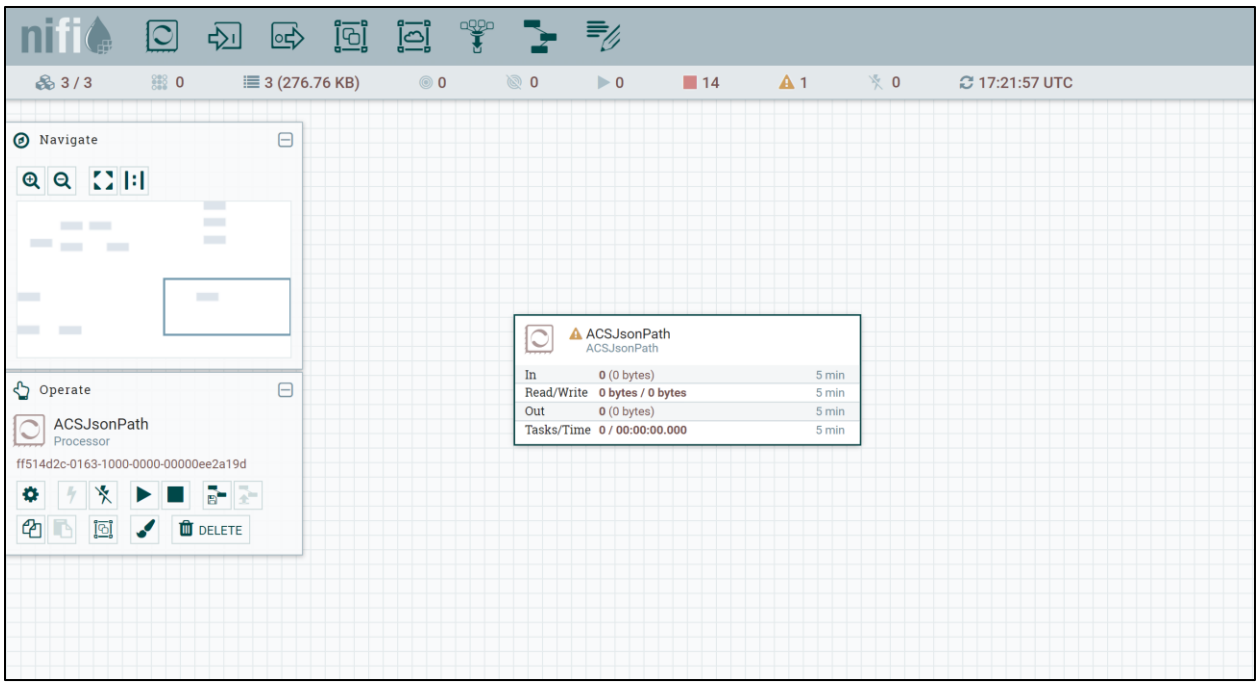
```
sudo chmod u+x /opt/nifi/default/lib/nifi-examples-bundle-1.0-SNAPSHOT.nar
sudo chown nifi:nifi /opt/nifi/default/lib/nifi-examples-bundle-1.0-SNAPSHOT.nar
```

SSH into the ACS basebox node. Restart NiFi with Ansible in order for NiFi to load the new processor. NiFi may take a few minutes to fully reload.

```
sudo ansible -k -a "systemctl restart nifi" nifi
```

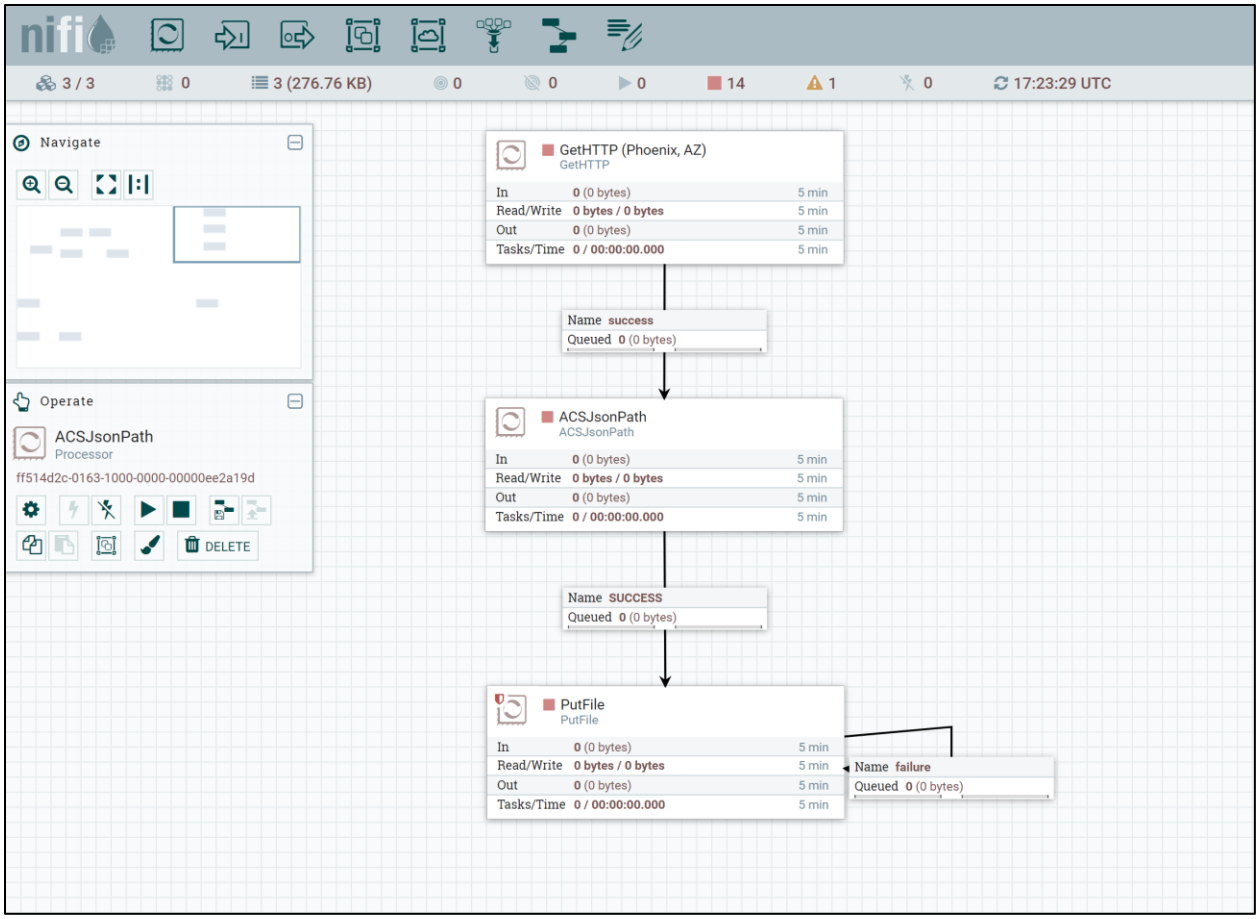
Verify the Custom Processor is Loaded into NiFi

Log into the NiFi web interface and drag the Processor icon down to the grid area. Search for ACSJsonPath and verify that the processor successfully loads in the NiFi UI. The search takes "ACS", "path", and "JSON" as keywords.

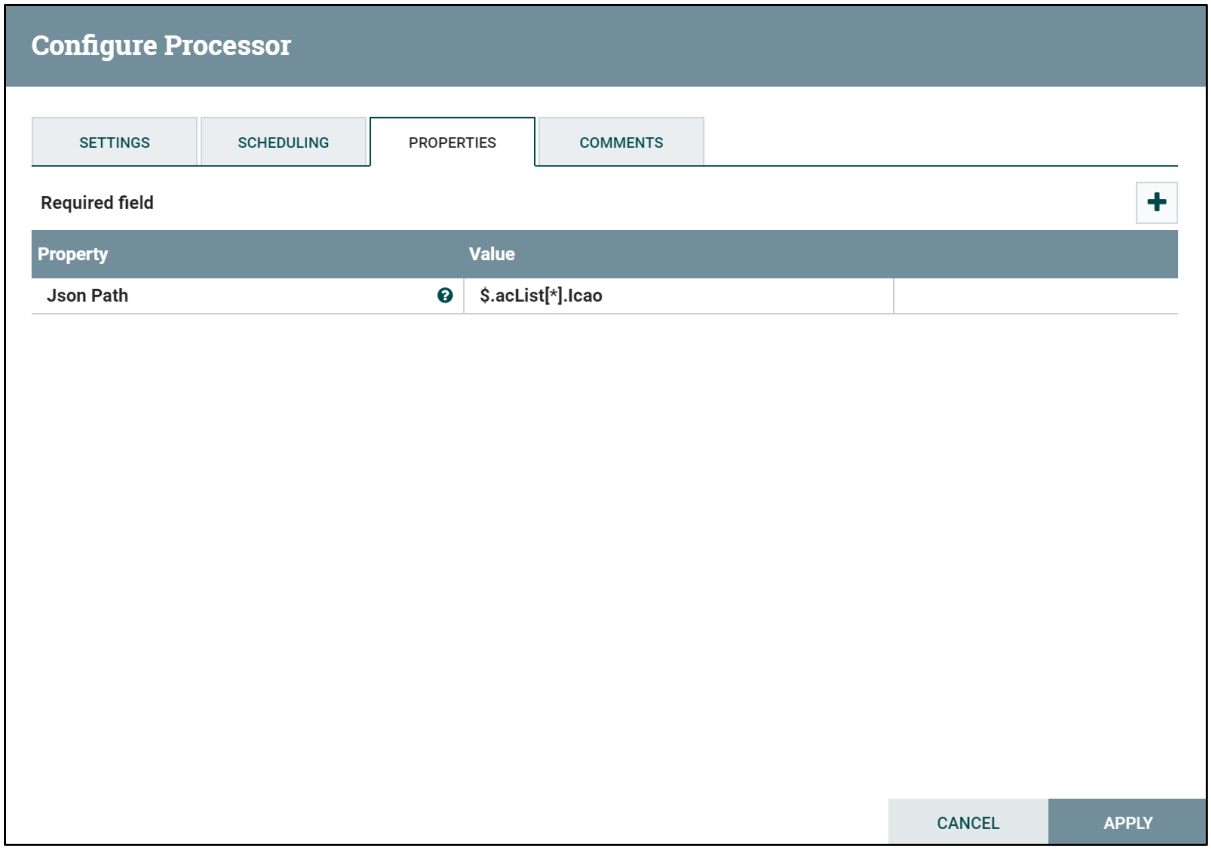


Apply Custom Processor to Data Flow

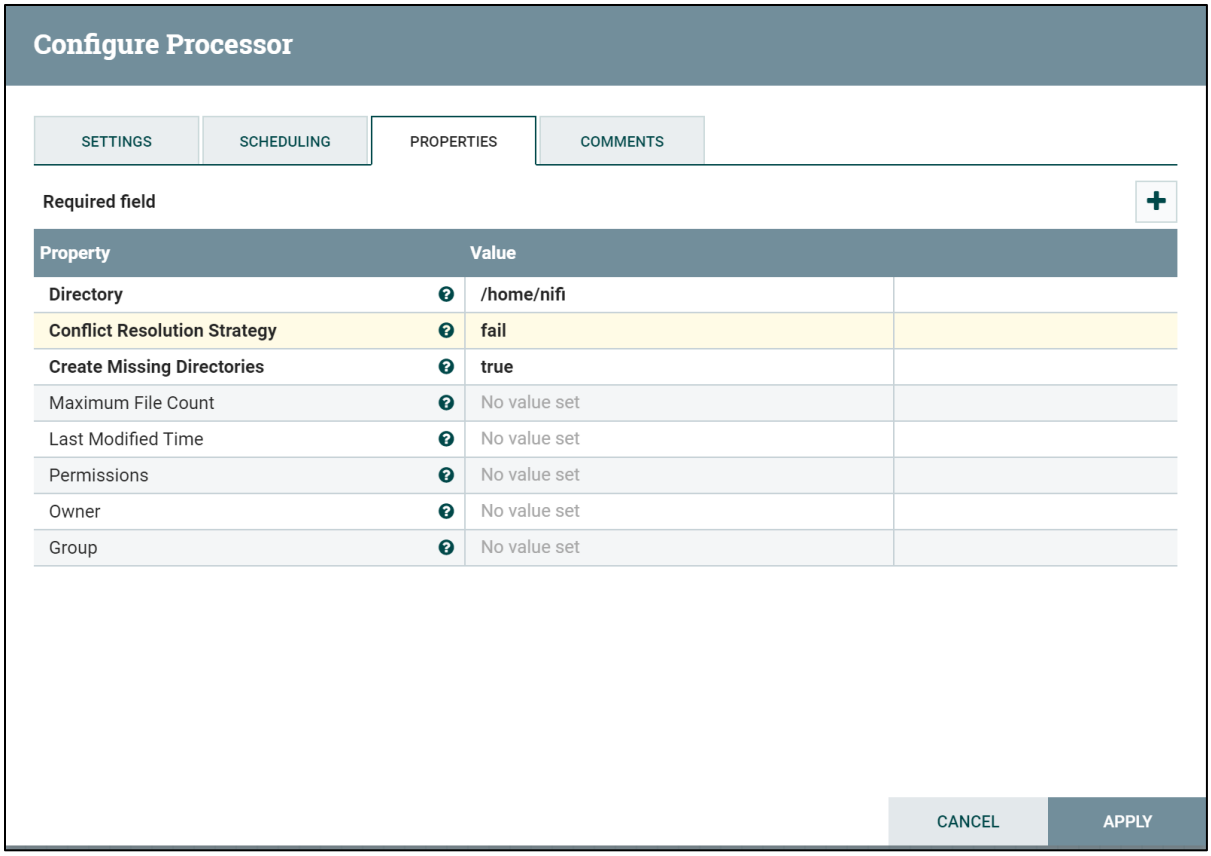
If ACSJsonPath loads into the NiFi UI successfully, the processor can then integrate into a data flow. This example pulls incoming flight data in real time, filters out the 'Icao' data field from the data stream by specifying a json path, and outputs the result to a file located in NiFi's home directory.



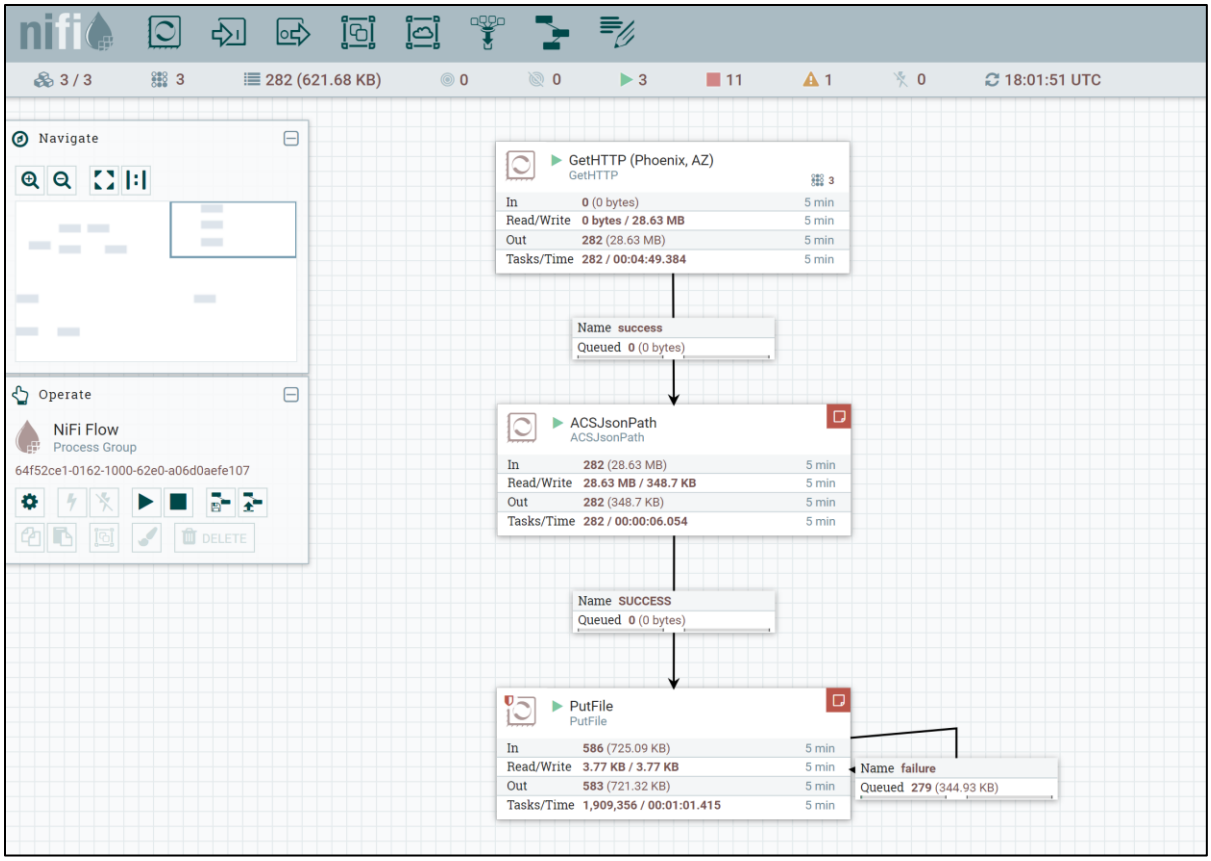
1. Browse to http://<nifi_node>:6434/nifi/
2. Refer to Appendix A Flight Data Demo step 1 on how to configure the GetHTTP processor
3. Drag down the processor icon located at the top left-hand corner. Search for the ACSJsonPath processor and add it to the grid area
4. Right click the ACSJsonPath processor and select Configure
5. Under the Properties tab, set the Json Path to $.acList[*].Icao and click on Apply. This value will filter out all Icao identifiers within the acList array from the incoming data stream



6. Drag down the processor icon located at the top left-hand corner. Search for the PutFile processor and add it to the grid area
7. Right click the PutFile processor and select Configure
8. Under the Properties tab, set the Directory property to /home/nifi and click on Apply. Leave the rest of the properties set to the their default values



9. Create a connection between GetHTTP and ACSJsonPath and ensure that the success relationship is checked
10. Create a connection between ACSJsonPath and PutFile and ensure that the success relationship is checked
11. Create a connection that's only connected to PutFile and ensure that the failure relationship is checked. This connection will catch any failures and loop it through the PutFile processor for re-processing
12. Initiate the data flow by shift selecting all three processors and clicking on the Start icon located in the Operate box on the left-hand side



13. With the mouse pointer, hover over the red box within the ACSJsonPath processor. If it reads "ACSDEBUG: found a match", then the processor should have found the Icao json path within the acList array from the data stream. Otherwise if the processor was unsuccessful, then the red box would read "Failed to read json string."
14. In a terminal, SSH to the same NiFi node used to access the NiFi GUI
15. Become the nifi user
16. With a text editor such as vim, open the /home/nifi/flights_icao.json file created from the data flow
17. If the data flow ran correctly, all the Icao's from the data stream should be filtered out in the flights_icao.json file. Now we can further experiment with the custom processor by specifying different json paths from the flight data stream

```
[
    "AB3C58",
    "A70726",
    "A16665",
    "ABB089",
    "ACD5BA",
    "A13E59",
    "AA8152",
    "AD0562",
    "A05353",
    "AAE038",
    "AB9A44",
```

```
    "A9D2E0",
    "AD2DFF",
    "A1ADDB",
    "A19B48",
    "AD08EF",
    "AAA107",
    "A7D74B",
    "A7E627",
    "AAEA83",
...
]
```

APPENDIX D ACRONYMS

Acronym	Full Name
ABAC	Attribute Based Access Control
ABL	Allocated Baseline
ACL	Access Control List
ACS	Agile Core Services
ACS-C	Agile Core Services Common Services
ACS-DA	Agile Core Services Data Analytics
ADM	Acquisition Decision Memorandum
ALSP	Acquisition Logistics Support Plan
AMQP	Advanced Message Queuing Protocol
AN	Afloat Networks
AO	Authorizing Official
AoA	Analysis of Alternatives
AP	Acquisition Plan
APB	Acquisition Program Baseline
API	Application Programming Interface
APM	Assistant Program Manager
APM-L	Assistant Program Manager - Logistics
APPP	Abbreviated Program Protection Plan
ARWG	ACS Requirements Working Group
ATO	Authority to Operate
AWS	Amazon Web Services
BCA	Business Case Analysis
BTR	Build Technical Review
C&A	Certification & Accreditation
C2S	Commercial Cloud Services
C4I	Command, Control, Communications, Computers and Intelligence
C4ISR	C4I, Surveillance & Reconnaissance
CANES	Consolidated Afloat Networks and Enterprise Services

Acronym	Full Name
CCB	Configuration Change Board
CDD	Capability Development Document
CI	Configuration Item
CJMTK	Commercial Joint Mapping Toolkit
CL	Command Line
CLI	Command Line Interface
CM	Configuration Management
CMP	Configuration Management Plan
COI	Communities of Interest
COR	Contracting Officer Representative
COTS	Commercial-off-the-Shelf
CRL	Certificate Revocation List
CS	Common Services
CSCI	Computer Software Configuration Item
DAF	Data Analytics Framework
DASN	Deputy Assistant Secretary of the Navy
DCGS-N	Distributed Common Ground System - Navy
DCID	Director of Central Intelligence Directive
DCMA	Defense Contract Management Agency
D-DIL	Denied-Disconnected, Intermittent, Low
DFARS	Defense Federal Acquisition Regulation Supplement
DIACAP	DoD IA C&A Process
DI2E	Defense Intelligence Information Enterprise
DMSMS	Diminishing Manufacturing Sources and Material Shortages
DNS	Domain Name System
DoD	Department of Defense
DoDAF	Department of Defense Architectural Framework
DoDD	Department of Defense Directive
DoDI	Department of Defense Instruction
DoN	Department of the Navy

Acronym	Full Name
DPF	Data Processing Framework
DT	Developmental Test
DT&E	Developmental Test and Evaluation
DTIC	Defense Technical Information Center
E2C	Engineering Enterprise and Certification
EA	Economic Analysis
ERB	Engineering Review Board
FD	Fielding Decision
FEA	Federal Enterprise Architecture
FMB 10	Financial Management Budget 2010
GCCS-M	Global Command and Control System - Maritime
GEOINT	Geospatial Intelligence
GSS-API	Generic Security Services Application Program Interface
GENSER	General Service
GFI	Government Furnished Information
GIG	Global Information Grid
GIS	Geospatial Information Services
GOTS	Government-off-the-Shelf
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
HW	Hardware
IA	Information Assurance
IATT	Interim Authority to Test
IC	Intelligence Community
ICD	Initial Capabilities Document
IdAM	Identity and Access Management
IDIQ	Indefinite Delivery, Indefinite Quantity
ILA	Integrated Logistics Assessment
ILS	Integrated Logistics Support
IMP	Integrated Master Plan

Acronym	Full Name
IMS	Integrated Master Schedule
INFOCON	Information Operations Condition
IO	Information Operations
IOT&E	Initial Operational Test & Evaluation
IP	Internet Protocol
IPBL	Initial Product Baseline
IPPD	Integrated Product and Process Development
IPT	Integrated Product Team
IRS	Interface Requirements Specification
ISR	Intelligence, Surveillance, Reconnaissance
IT	Information Technology
JIE/IC-ITE/DI2E	Joint Information Environment/Intelligence Community-Information Technology Enterprise/Defense Intelligence Information Enterprise
JMAP	JSON Meta Application Protocol
JMS	Java Message Service
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KPP	Key Performance Parameter
KSA	Key System Attribute
LAN	Local Area Network
LCCB	Local Configuration Control Board
LCL	Life Cycle Logistics
LCSP	Life Cycle Sustainment Plan
LDAP	Lightweight Directory Access Protocol
LDAPS	Lightweight Directory Access Protocol over SSL
LE	Lead Engineer
LSE	Lead Systems Engineer
M2M	Machine-to-Machine
MDA	Milestone Decision Authority
MIL-HDBK	Military Handbook

Acronym	Full Name
MNS	Mission Need Statement
MOA	Memorandum of Agreement
MOSA	Modular Open Systems Architecture
MS	Milestone
MTC2	Maritime Tactical Command and Control
NCAAS	National Maritime Intelligence Center (NMIC) Centralized Authorization & Authentication Service
NETC	Naval Education and Training Command
NOC	Network Operations Center
NR-KPP	Net-Ready Key Performance Parameters
NSA	National Security Administration
NSS	National Security System
NTAF	Navy Tactical Analysis Framework
NTC	Navy Tactical Cloud
NTCSS	Naval Tactical Command Support System
NTP	Network Time Protocol
OA	Operational Assessment
OB2	Objective Baseline
OCSP	Online Certificate Status Protocol
OGC	Open Geospatial Consortium
OIPT	Overarching IPT
ONR	Office of Naval Research
OPN	Other Procurement, Navy
OPNAV	Office of the Chief of Naval Operations
ORD	Operational Requirements Document
OSD	Office of the Secretary of Defense
OSGi	Open Service Gateway Initiative
OSS	Open Source Software
OT	Operational Testing
OT&E	Operational Test and Evaluation

Acronym	Full Name
OWF	Ozone Widget Framework
PaaS	Platform as a Service
PBL	Performance Based Logistics
PBAC	Policy Based Access Control
PDA	Personal Digital Assistant
PDR	Preliminary Design Review
PEO	Program Executive Office
PIPT	Program-level Integrated Product Team
PKI	public key infrastructure
PLCCE	Program Life Cycle Cost Estimate
PM	Program Manager
PMR	Program Management Review
POR	Program of Record
PMW 160	Program Management, Warfare
POM	Program Objective Memorandum
PoPS	Probability of Program Success
R&D	Research and Development
R&I	Risks and Issues
R&M	Reliability and Maintainability
RDBMS	Relational Database Management System
RDD	Resilient Distributed Dataset
RDF	Resource Description Framework
RDT&E	Research, Development, Test and Evaluation
REST	Representational State Transfer
RHEL	Red Hat Enterprise Linux
RI	Reference Implementation
RMF	Risk Management Framework
RPC	Remote Procedure Call
RPM	RPM Package Manager
RQMP	Requirements Management Plan

Acronym	Full Name
RQRB	Requirements Review Board
SAML	Security Assertion Markup Language
SASL	Simple Authentication and Security Layer
SCRAM	Salted Challenge Response Authentication Mechanism
SDD	Software Design Description
S&T	Science and Technology
SE	Systems Engineering
SEP	Systems Engineering Plan
SER	System Engineering Review
SETR	Systems Engineering Technical Review
SHA	Secure Hash Algorithm
SFR	System Functional Review
SLD	Styled Layer Descriptor
SoS	System of Systems
SPARQL	SPARQL Protocol and RDF Query Language
SPAWAR	Space and Naval Warfare Systems Command
SRR	System Requirements Review
SSC PAC	SPAWAR Systems Center Pacific
SSH	Secure Socket Shell
SSL	Secure Socket Layer
SQL	Structured Query Language
SRS	Software Requirements Specification
SSO	Single Sign-On
STIGs	Security Technical Implementation Guides
SW	Software
STOMP	Simple (or Streaming) Text Orientated Messaging Protocol
TA	Technical Authority
T&E	Test & Evaluation
TDA	Technical Decision Authority
TEMP	Test & Evaluation Master Plan

Acronym	Full Name
TCRI	Tactical Cloud Reference Implementation
TRL	Technology Readiness Level
TS	Top Secret
TTA	Technology Transition Agreement
UCD	User-Centered Design
UCDMO	Unified Cross Domain Management Office
ULSS	User's Logistics Support Summary
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
U.S.C.	United States Code
USD(AT&L)	Under Secretary of Defense for Acquisition Technology & Logistics
USW-DSS	Undersea Warfare - Decision Support System
WAN	Wide Area Network
WCS	Web Coverage Service
WFS	Web Feature Service
WMS	Web Map Service
XACML	eXtensible Access Control Markup Language
YAML	YAML Ain't Markup Language
YARN	Yet Another Resource Negotiator