SmartSense Configuration 1

Configuring SmartSense

Date of Publish: 2018-11-15

http://docs.hortonworks.com

Contents

Configuring bundle upload

SmartSense gateway is automatically configured with HTTPS so you don't normally need to perform this configuration. However, if a specific custom configuration is required by your corporate network firewall policies, you can use these instructions to configure SmartSense gateway to upload bundles by using HTTPS.

Configure the HST gateway to use HTTPS

You can configure the gateway to use HTTPS to upload bundles to Hortonworks by using the connectivity and configuration details available in Configuring the SmartSense gateway for automatic bundle submission. To view this article, you need a valid Hortonworks support account. To use an authenticated HTTP or HTTPS proxy to upload bundles to Hortonworks, follow these steps:

Procedure

1. If you would like to set up HTTPS proxy, you must contact Hortonworks Support.

   On the SmartSense gateway host, edit the /etc/hst/conf/gateway/hst-gateway.ini file and supply the appropriate values for your environment:

```
; All proxy configurations are applicable only for HTTPS provider type
;#set to true#to#set up#a#proxy#between#gateway#and#SmartSense#environment
;default:false
provider.https.proxy.enabled=true
;#fully#qualified#proxy#hostname
provider.https.proxy.hostname=your.proxy.host
;#proxy#port#that#will#be#used#by#gateway#for#outbound#access
provider.https.proxy.port=3128
;#supported proxy#types#:#HTTP#/#HTTPS#[default:HTTP]
provider.https.proxy.type=HTTP
; supported proxy authentication
 #types#:#NONE#/#BASIC#/#DIGEST#[default:NONE]
provider.https.proxy.auth.type=BASIC
;#proxy#username#for#identified#auth.type
provider.https.proxy.auth.username=proxyuser
;#proxy#password#for#identified#auth.type
provider.https.proxy.auth.password=proxypassword
;#[optional]#any#additional#proxy#setup#parameters
; use#"|" to#separate#multiple#parameters
;#for example:#digest#requires#setting#parameters#such as
;#realm=default|nonce=12GHtqeZA!7Ke43
provider.https.proxy.auth.parameters=
```

2. After you update the configuration file, restart the SmartSense gateway:

```
hst gateway restart
```

Related Information

Configuring the SmartSense gateway for automatic bundle submission (Salesforce article)

Configuring data anonymization rules

As data is captured, specific types of data are automatically anonymized. By default, IP addresses and the domain component of host names are anonymized. To customize these anonymization rules, follow the steps and use the reference information related to fields used for defining anonymization rules and anonymization rule types.

Anonymization rule types

Anonymization rules define regular expressions to anonymize sensitive data (like IP addresses, and so on). Each rule uses JSON format to define what to match and the value to replace.

You can define the following types of anonymization rules:

- Pattern-based - Anonymize data by pattern, using the extract field to match and extract content to anonymize.
- Property-based - Anonymize structured content. The supported formats are: XML, property, ini, and YAML files.
- Xpath-based - Anonymize XML data using XPATH.
- JSONPath-based - Anonymize JSON data using JSONPATH.

In addition, there are domain-based rules that can be used to anonymize domain names. They are a special case of pattern rules where the anonymization pattern is build from local host FQDN. The domain-based rules cannot be customized.

Note:

Anonymization rule formats vary between different SmartSense versions. Make sure that you consult the documentation that matches your SmartSense version.

Pattern-based anonymization rules

Write pattern-based rules to anonymize data by pattern, using the extract pattern to extract content to anonymize.

Required and Optional Fields

- name
- description (optional)
- rule_id (should be set to PATTERN)
- patterns
- extract (optional)
- include_files (optional)
- exclude_files (optional)
- action (optional, default value is ANONYMIZE)
- replace_value (optional, applicable only when action=REPLACE)
- shared (optional, default value is true)
- enabled (optional, default value is true)

Rule Definition Example (without extract)

```
{
  "name": "EMAIL",
  "rule_id": "Pattern",
  "patterns": ["(?<![a-z0-9._%+-])[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,6}(?!
[a-z0-9._%+-])$?",
  "shared": false
}
```

The content of the input file version.txt is:

```
Hadoop 2.7.3.2.5.0.0-1245
Subversion git@github.com:hortonworks/hadoop.git -r
 cb6e514b14fb60e9995e5ad9543315cd404b4e59
Compiled by jenkins on 2016-08-26T00:55Z
```

The content of the output file version.txt, with anonymized email address, is:

```
Hadoop 2.7.3.2.5.0.0-1245
```

```
Subversion ‡qpe@unqfay.mjp‡:hortonworks/hadoop.git -r
 cb6e514b14fb60e9995e5ad9543315cd404b4e59
Compiled by jenkins on 2016-08-26T00:55Z
```

Rule Definition Example (with extract)

```
{
  "name": "KEYSTORE",
  "rule_id": "Pattern",
  "patterns": ["oozie.https.keystore.pass=([^\\s]*)",
 "OOZIE_HTTPS_KEYSTORE_PASS=([^\\s]*)"],
  "extract": "=([^\\s]*)",
  "include_files": ["java_process.txt", "pid.txt", "ambari-agent.log",
 "java_process.txt", "oozie-env.cmd"],
  "shared": false
}
```

The content of the input file oozie-env.cmd is:

```
oozie.https.keystore.pass=abcde
set OOZIE_HTTPS_KEYSTORE_PASS=12345
```

To anonymize the content of the input file, the following anonymization patterns configured in the rule will be used:

```
"oozie.https.keystore.pass=([^\\s]*)", "OOZIE_HTTPS_KEYSTORE_PASS=([^\\s]*)"
```

oozie.https.keystore.pass=([^\\s]*) and OOZIE_HTTPS_KEYSTORE_PASS=([^\\s]*) match with oozie.https.keystore.pass=abcde and OOZIE_HTTPS_KEYSTORE_PASS=12345 respectively.

Next, the extract pattern "=([^\\s]*) is used to identify 12345 and abcde, which are the values to be anonymized.

The content of the output file oozie-env.cmd is:

```
oozie.https.keystore.pass=‡vvdwa‡
set OOZIE_HTTPS_KEYSTORE_PASS=‡zdowg‡
```

The values of oozie.https.keystore.pass and OOZIE_HTTPS_KEYSTORE_PASS have been anonymized.

More Examples

Example 1: Mask by pattern across all log files, without extract pattern

To mask all email addresses in all log files, use the following rule definition:

```
{
  "name": "EMAIL",
  "rule_id": "Pattern",
  "patterns": ["(?<![a-z0-9._%+-])[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,6}(?!
[a-z0-9._%+-])"],
  "include_files": ["*.log*"],
  "shared": false
}
```

Example 2: Mask by pattern across all log files, with extract pattern

To mask encryption keys, logged in the following format Key=12.. with a value consisting of 64 hexadecimal characters, use the following rule definition:

```
{
  "name": "ENC_KEYS",
  "rule_id": "Pattern",
  "patterns": ["Key=[a-f\\d]{64}\\s"],
  "extract": "=([a-f\\d]{64})",
```

```
  "include_files": ["*.log*"],
  "shared": false
}
```

Input data, test.log is:

```
encryption
 key=1234567890adc1234567aaabc1234567890adc1234567aaabc12345678901234 for
 keystore
derby.system.home=null
```

Output data, test.log, with the encryption keys anonymized, is:

```
encryption
 key=‡8697685738fnx1736987qigyx7611731027yds0096404hlsph91727138403654‡ for
 keystore
derby.system.home=null
```

Example 3: Mask by pattern across all files, except a few files

To mask email addresses in all files, except hdfs-site.xml and .property files, use the following rule definition:

```
{
  "name": "EMAIL",
  "rule_id": "Pattern",
  "patterns": ["(?<![a-z0-9._%+-])[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,6}(?!
[a-z0-9._%+-])"],
  "exclude_files" : ["*.properties", "hdfs-site.xml"],
  "shared": false
}
```

Input data, version.txt, is:

```
Hadoop 2.7.3.2.5.0.0-1245
Subversion git@github.com :hortonworks/hadoop.git -r
 cb6e514b14fb60e9995e5ad9543315cd404b4e59
Compiled by jenkins on 2016-08-26T00:55Z
```

Output file version.txt, with an anonymized email address, is:

```
Hadoop 2.7.3.2.5.0.0-1245
Subversion ‡qpe@unqfay.mjp‡ :hortonworks/hadoop.git -r
 cb6e514b14fb60e9995e5ad9543315cd404b4e59
Compiled by jenkins on 2016-08-26T00:55Z
```

Related Information

Fields used for defining anonymization rules

Property-based anonymization rules

Property-based rules anonymize structured content. The supported formats are: XML, property, ini, and YAML files.

Required and Optional Fields

- name
- description (optional)
- rule_id (should be set to PROPERTY)
- properties
- parentNode (optional, applicable only for XML, default value is "property")
- include_files

- exclude_files (optional)
- action (optional, default value is ANONYMIZE)
- replace_value (optional, applicable only when action=REPLACE)
- shared (optional, default value is true)
- enabled (optional, default value is true)

Rule Definition Example

```
{
  "name": "PASSWORDS",
  "rule_id": "Property",
  "properties": [".*password.*", ".*awsAccessKeyId.*"],
  "include_files": ["*.xml", "*.properties", "*.yaml", "*.ini"],
  "exclude_files" : ["capacity-scheduler.xml"],
  "action" : "REPLACE",
  "replace_value": "Hidden"
}
```

The following examples show how the rule defined above anonymizes specific password-related properties in XML, property, ini, and YAML files.

- XML file content:

  ```
  <property>
    <name>fs.s3a.proxy.password</name>
    <value>Abc7j*4$aTh</value>
    <description>Password for authenticating with proxy server.</
  description>
  </property>
  ```

  The XML file content, with password value anonymized:

  ```
  <property>
    <name>fs.s3a.proxy.password</name>
    <value>Hidden</value>
    <description>Password for authenticating with proxy server.</
  description>
  </property>
  ```

- Property file content:

  ```
  javax.jdo.option.ConnectionPassword=pswd
  ```

  The property file content, with password value anonymized:

  ```
  javax.jdo.option.ConnectionPassword=Hidden
  ```

- Ini file content:

  ```
  connection_password=pswd
  ```

  The ini file content, with password value anonymized:

  ```
  connection_password=Hidden
  ```

- YAML file content:

  ```
  "metrics_collector:\n" +
                  "  truststore.path : \"/etc/security/clientKeys/all.jks
  \"\n" +
                  "  truststore.type : \"jks\"\n" +
  ```

```
                "  truststore.password : \"bigdata\"\n"
```

The YAML file content, with password value anonymized:

```
"metrics_collector:\n" +
                "  truststore.path : \"/etc/security/clientKeys/all.jks
\"\n" +
                "  truststore.type : \"jks\"\n" +
                "  truststore.password : Hidden\n"
```

More Examples

Example 1: Mask one configuration parameter in multiple files

Rule definition example:

```
{
  "name": "JPA_PASSWORD",
  "rule_id": "Property",
  "properties": ["oozie.service.JPAService.jdbc.password"],
  "include_files": ["oozie-site.xml", "sqoop-site.xml"],
  "action" : "REPLACE",
  "replace_value": "Hidden"
}
```

This rule anonymizes the value of oozie.service.JPAService.jdbc.password in oozie-site.xml and sqoop-site.xml:

Input data, sqoop-site.xml:

```
<property>
  <name>oozie.service.JPAService.jdbc.px</name>
  <value>at@!_*rue</value>
</property>
```

Output data, sqoop-site.xml, with anonymized oozie.service.JPAService.jdbc.px parameter value:

```
<property>
  <name>oozie.service.JPAService.jdbc.px</name>
  <value>Hidden</value>
</property>
```

Example 2: Mask multiple configuration parameters in multiple files

Rule definition example:

```
{
  "name": "JDBC_JPA_PASSWORDS",
  "rule_id": "Property",
  "properties": ["oozie.service.JPAService.jdbc.password",
 "javax.jdo.option.ConnectionPassword"],
  "include_files": ["oozie-site.xml", "sqoop-site.xml", "hive-site.xml"],
  "action" : "REPLACE",
  "replace_value": "Hidden"
}
```

Example 3: Mask a configuration that matches a pattern

Rule definition example:

```
{
  "name": "GLOBAL_JDBC_PASSWORDS",
  "rule_id": "Property",
  "properties": [".*password"],
```

```
  "include_files": ["*.xml"],
  "action" : "REPLACE",
  "replace_value": "Hidden"
}
```

Input data:

ssl-server.xml

```
<property>
  <name>ssl.server.keystore.keypassword</name>
  <value>big123!*</value>
</property>
```

ssl-client.xml

```
<property>
  <name>ssl.client.keystore.password</name>
  <value>NBg7j*4$aTh</value>
</property>
```

Output data:

Anonymized ssl-server.xml

```
<property>
  <name>ssl.server.keystore.keypassword</name>
  <value>Hidden</value>
</property>
```

Anonymized ssl-client.xml

```
<property>
  <name>ssl.client.keystore.password</name>
  <value>Hidden</value>
</property>
```

Related Information

Fields used for defining anonymization rules

XPath-based anonymization rules

XPath-based rules anonymize XML data using XPath.

Required and Optional Fields

- name
- description (optional)
- rule_id (should be set to XPATH)
- paths
- include_files
- exclude_files (optional)
- action (optional, default value is ANONYMIZE)
- replace_value (optional, applicable only when action=REPLACE)
- shared (optional, default value is true)
- enabled (optional, default value is true)

Rule Definition Example

```
{
```

```
  "name": "XPATH_RULE",
  "rule_id": "XPATH",
  "paths": ["/data/record[1]/value"],
  "include_files": ["*test_config.xml"],
  "shared": true
}
```

Sample Input XML Data

```
<data>
    <record>
        <name>password</name>
        <value>valueToAnonymize</value>
    </record>
    <record>
        <name>name</name>
        <value>value</value>
    </record>
</data>
```

Sample Output XML Data (After Anonymization)

```
<data>
    <record>
        <name>password</name>
        <value>¶smfz923swc¶</value>
    </record>
    <record>
        <name>name</name>
        <value>value</value>
    </record>
</data>
```

More Examples

Example 1: Rule with nested XML structure

Rule definition example:

```
{
  "name": "NESTED_XPATH_RULE",
  "rule_id": "XPATH",
  "paths": ["/configs/properties/passwd"],
  "include_files": ["*config.xml"],
  "shared": true
}
```

Input data:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configs>
    <properties>
        <user>abc</user>
        <passwd>1234</passwd>
    </properties>
</configs>
```

Output data (after anonymization):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configs>
    <properties>
```

```
        <user>abc</user>
        <passwd>¶9165¶</passwd>
    </properties>
</configs>
```

Example 2: Rule with XML array structure

Rule definition example:

```
{
  "name": "ARRAY_XPATH_RULE",
  "rule_id": "XPATH",
  "paths": ["/configs/properties[2]/passwd"],
  "include_files": ["*config.xml"],
  "shared": true
}
```

Input data:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configs>
    <properties>
        <database>mysql</database>
        <url>user@host:port</url>
    </properties>
    <properties>
        <user>abc</user>
        <passwd>1234</passwd>
    </properties>
</configs>
```

Output data (after anonymization):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configs>
    <properties>
        <database>mysql</database>
        <url>user@host:port</url>
    </properties>
    <properties>
        <user>abc</user>
        <passwd>¶9165¶</passwd>
    </properties>
</configs>
```

Example 3: Rule with XML map structure

Rule definition example:

```
{
  "name": "MAP_XPATH_RULE",
  "rule_id": "XPATH",
  "paths": ["/configs/properties/passwd"],
  "include_files": ["*config.xml"],
  "shared": true
}
```

Input data:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configs>
    <db>mysql</db>
```

```
    <properties>
        <user_name>sa</user_name>
        <passwd>sa_pass</passwd>
    </properties>
    <pooli_size>32</pooli_size>
    <timeout>10</timeout>
</configs>
```

Output data (after anonymization):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><configs>
    <db>mysql</db>
    <properties>
        <user_name>sa</user_name>
        <passwd>¶vm_wtto¶</passwd>
    </properties>
    <pooli_size>32</pooli_size>
    <timeout>10</timeout>
</configs>
```

Example 4: Rule to mask all array elements

Rule definition example:

```
{
  "name": "ALL_FROM_ARRAY_XPATH_RULE",
  "rule_id": "XPATH",
  "paths": ["/configs/properties[*]/passwd"],
  "include_files": ["*config.xml"],
  "shared": true
}
```

Input data:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configs>
    <properties>
        <user>abc1</user>
        <passwd>pass1</passwd>
    </properties>
    <properties>
        <user>abc2</user>
        <passwd>pass2</passwd>
    </properties>
</configs>
```

Output data (after anonymization):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configs>
    <properties>
        <user>abc1</user>
        <passwd>¶smfz7¶</passwd>
    </properties>
    <properties>
        <user>abc2</user>
        <passwd>¶smfz8¶</passwd>
    </properties>
</configs>
```

Example 5: Rule to mask some array elements which have passwd

Rule definition example:

```
{
  "name": "SOME_FROM_ARRAY_XPATH_RULE",
  "rule_id": "XPATH",
  "paths": ["/configs/properties[passwd]/passwd"],
  "include_files": ["*config.xml"],
  "shared": true
}
```

Input data:

```
<?xml version="1.0" encoding="UTF-8" ?>
<configs>
    <properties>
        <user>abc1</user>
        <passwd1>pass1</passwd1>
    </properties>
    <properties>
        <user>abc2</user>
        <passwd2>pass2</passwd2>
    </properties>
    <properties>
        <user>abc3</user>
        <passwd>pass3</passwd>
    </properties>
</configs>
```

Output data (after anonymization):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configs>
    <properties>
        <user>abc1</user>
        <passwd1>pass1</passwd1>
    </properties>
    <properties>
        <user>abc2</user>
        <passwd2>pass2</passwd2>
    </properties>
    <properties>
        <user>abc3</user>
        <passwd>¶smfz9¶</passwd>
    </properties>
</configs>
```

Related Information

Fields used for defining anonymization rules

XPath syntax

JSONPath-based anonymization rules

JSONPath-based rules anonymize JSON data using JSONPath.

Required and Optional Fields

- name
- description (optional)
- rule_id (should be set to JSONPATH)
- paths
- include_files

- exclude_files (optional)
- action (optional, default value is ANONYMIZE)
- replace_value (optional, applicable only when action=REPLACE)
- shared (optional, default value is true)
- enabled (optional, default value is true)

Rule Definition Example

```
{
  "name": "JSONPATH_RULE",
  "rule_id": "JSONPATH",
  "paths": ["$.users[0].password"],
  "include_files": ["*test_config.json"],
  "shared": true
}
```

Sample Input JSON Data

```
{
  "users": [
    {
      "name": "Logsearch Admin",
      "username": "admin",
      "password": "testdata"
    },
    {
      "name": "Admin",
      "username": "admin",
      "password": "test data"
    }
  ]
}
```

Sample Output JSON Data (After Anonymization)

```
{
  "users": [
    {
      "name": "Logsearch Admin",
      "username": "admin",
      "password": "¶smfvvcz9¶"
    },
    {
      "name": "Admin",
      "username": "admin",
      "password": "test data"
    }
  ]
}
```

More Examples

Example 1: Rule with nested JSON elements

Rule definition example:

```
{
  "name": "NESTED_JSONPATH_RULE_1",
  "rule_id": "JSONPATH",
  "paths": ["$.configs.properties.passwd"],
  "include_files": ["*config.json"],
  "shared": true
```

```
}
```

Input data:

```
{
  "configs": {
      "properties":
         {
            "user": "abc",
            "passwd": "12!@"
         }
    }
}
```

Output data (after anonymizarion):

```
{
  "configs": {
    "properties": {
      "user": "abc",
      "passwd": "¶91!@¶"
    }
  }
}
```

Example 2: Rule with indexed JSON array objects

Rule definition example:

```
{
  "name": "ARRAY_JSONPATH_RULE",
  "rule_id": "JSONPATH",
  "paths": ["$.configs.properties[1].passwd"],
  "include_files": ["config.json"],
  "shared": true
}
```

Input data:

```
{
  "configs": {
    "properties": [
       {
          "database": "mysql",
          "url": "user@host:port"
       },
       {
          "user": "abc",
          "passwd": "12!@"
       }
     ]
  }
}
```

Output data (after anonymization):

```
{
  "configs": {
    "properties": [
      {
        "database": "mysql",
        "url": "user@host:port"
```

```
      },
      {
        "user": "abc",
        "passwd": "¶91!@¶"
      }
    ]
  }
}
```

Example 3: Rule with JSON map

Rule definition example:

```
{
  "name": "MAP_JSONPATH_RULE",
  "rule_id": "JSONPATH",
  "paths": ["$.properties.passwd"],
  "include_files": ["*config.json"],
  "shared": true
}
```

Input data:

```
{
  "db":"mysql",
  "properties":{
    "user_name":"sa",
    "passwd":"sa_pass"
  },
  "pooli_size":32,
  "timeout":10
}
```

Output data (after anonymization):

```
{
  "db": "mysql",
  "properties": {
    "user_name": "sa",
    "passwd": "¶vm_wtto¶"
  },
  "pooli_size": 32,
  "timeout": 10
}
```

Example 4: Rule to mask all JSON objects from list

Rule definition example:

```
 {
  "name": "ALL_FROM_ARRAY_JSONPATH_RULE",
  "rule_id": "JSONPATH",
  "paths": ["$.configs.properties[*].passwd"],
  "include_files": ["*config.json"],
  "shared": true
 }
```

Input data:

```
{
 "configs": {
   "properties": [
```

```
      {
        "user": "abc1",
        "passwd": "pass1"
      },
      {
        "user": "abc2",
        "passwd": "pass2"
      }
    ]
  }
}
```

Output data (after anonymization):

```
{
 "configs": {
   "properties": [
     {
       "user": "abc1",
       "passwd": "¶smfz7¶"
     },
     {
       "user": "abc2",
       "passwd": "¶smfz8¶"
     }
   ]
 }
}
```

Related Information

Fields used for defining anonymization rules

JsonPath syntax

Fields used for defining anonymization rules

To define anonymization rules, use the following fields:

Table 1: Fields used for defining anonymization rules

Field	Description
name	Provides a descriptive name for data anonymized by the rule. It has to be unique across all rules.
description	Provides a description for the rule.
rule_id	Defines the class of rules the current rule belongs to. The supported rule IDs are: PATTERN, PROPERTY, XPATH, JSONPATH. This parameter is case-insensitive.
patterns	Defines a list of data patterns to be anonymized. It is applicable only to Pattern rules, where rule_id=PATTERN. These patterns are matched in a case-insensitive manner, which means that the following pattern keystore.pass=([^\\s]*) matches with any of the following values: • keystore.pass=123 • KeyStore.Pass=123 • KEYSTORE.PASS=123

Field	Description
extract	Specifies a pattern to extract data matched through the list of patterns. The extract pattern is matched in a case-insensitive manner. For example, in order to anonymize the oozie.https.keystore.pass password, the following pattern and extract values are used: "patterns": ["oozie.https.keystore.pass=([^\\s]*)"] "extract": "=([^\\s]*)", This pattern is matched with values such as oozie.https.keystore.pass=1234. The extract pattern is used to extract and anonymize only the values after the = (which in this example is 1234). The [^\\s]* denotes all non-whitespace characters, and the capturing group () is used the exclude = from the anonymized value. If the extract pattern is not configured, the entire value matched with the pattern is anonymized (which in this example is oozie.https.keystore.pass=1234), regardless of capturing groups used in the patterns.
properties	Specifies a list of property name patterns to anonymize; these are case-insensitively matched. It is applicable only to Property rules.

parentNode	This field is applicable to property anonymization in XML files. It allows you to define the parent node of the property that you want to anonymize. By default, parentNode is set to "parentNode": "property", because typically the XML block to anonymize has the parent node property, like in the following example:
	`<property>`
	`  <name>fs.s3a.proxy.password</name>`
	`  <value>Abc7j*4$aTh</value>`
	`  <description>Password for`
	` authenticating with proxy server.</`
	`description>`
	`</property>`
	For example, you can anonymize main.ldapRealm.contextFactory.systemPassword in the following XML block that has a parent node called param by setting "parentNode": "param" in the anonymization rule:
	`<param>`
	` <name>main.ldapRealm.contextFactory.systemPassw`
	`name>`
	`  <value>pass</value>`
	`</param>`
	The rule to anonymize the above content configures param as the root tag "parentNode": "param":
	`  {`
	`    "name": "KNOX LDAP Password",`
	`    "rule_id": "Property",`
	`    "properties":`
	` ["main.ldapRealm.contextFactory.systemPassword"`
	`    "include_files": ["topologies/`
	`*.xml"],`
	`    "action" : "REPLACE",`
	`    "parentNode": "param",`
	`    "replace_value": "Hidden"`
	`  }`

Field	Description
action	The supported actions are: ANONYMIZE, DELETE, REPLACE. The action value is not case sensitive, so Anonymize or delete are also accepted values. ANONYMIZE action encrypts the data using the key indicated by shared flag, DELETE deletes the data, and REPLACE replaces the data with a predefined value, which can be customized using replace_value.
replace_value	This field is used by the REPLACE action to specify a replacement for the data to anonymize. The default value is Hidden.
shared	Indicates which key to use for anonymization (shared or private). This value is used when the anonymization action is set to ANONYMIZE. It is a boolean type property (true/false). If set to true - the Hortonworks support team can unmask data if needed for diagnostic purposes; for example, host names and IP addresses for resolving issues on specific hosts or communication between hosts. Note that unmasked data is not stored in Hortonworks repositories; it is discarded as soon as the analysis finishes. The default value is true. Rules configured with shared = false cannot be unmasked by Hortonworks (and in some cases might become a roadblock for support case analysis.)
include_files	Specifies a list of glob file patterns for which the rule applies. If not configured, the rule is applicable to all files.
exclude_files	Specifies a list of glob file patterns which are excluded from anonymization. If not configured, no file is excluded from the rule application.
enabled	A flag (true/false) which specifies if the rule is enabled to be executed. By default, it is set to true.

Configure data anonymization rules

As data is captured, specific types of data are automatically anonymized. By default, IP addresses and the domain component of host names are anonymized. To customize these anonymization rules, follow these steps:

Procedure

1. Navigate to the Ambari Dashboard and click the SmartSense service.
2. Click the Config tab.
3. Navigate to the Data Capture section.
4. Add the new anonymization rule (or change the existing rule) by following the details provided in the documentation for anonymization rule types.

Related Information

Anonymization rule types

Anonymization rule types

Anonymization rules define regular expressions to anonymize sensitive data (like IP addresses, and so on). Each rule uses JSON format to define what to match and the value to replace.

You can define the following types of anonymization rules:

- Pattern-based - Anonymize data by pattern, using the extract field to match and extract content to anonymize.
- Property-based - Anonymize structured content. The supported formats are: XML, property, ini, and YAML files.

- Xpath-based - Anonymize XML data using XPATH.
- JSONPath-based - Anonymize JSON data using JSONPATH.

In addition, there are domain-based rules that can be used to anonymize domain names. They are a special case of pattern rules where the anonymization pattern is build from local host FQDN. The domain-based rules cannot be customized.

Note:

Anonymization rule formats vary between different SmartSense versions. Make sure that you consult the documentation that matches your SmartSense version.

Improving SmartSense performance

This section contains tips for achieving optimal performance for your cluster.

Tuning JVM memory settings

To achieve optimal performance for your cluster size, you may need to increase the JVM memory settings.

The default setting, 2048 MB, is appropriate for a cluster with up to 100 nodes. For each additional 100 nodes, increase this setting by 0.5 GB to improve performance.

To adjust the setting, in the Ambari Web UI, navigate to the SmartSense service's Config section > Advanced > Advanced hst-server-conf where you will find the Server max heap size configuration property.

Cleaning up old bundles

HST server has a background process which periodically deletes bundles older than 30 days. Additionally, bundle "purge" commands can be used to trigger this process for purging bundles older than specified number of days or for purging a particular bundle. There are two ways to purge bundles:

- Purge: The bundle file is removed from the storage but associated records such as recommendations from the HST DB are retained.
- Hard purge: All bundle data and its associated records such as recommendations from the HST DB are completely removed.

When using hst purge, soft purging is used unless the hard purging option is specified.

Syntax

```
# hst purge -h
Usage: hst purge [-r][-b][-H][-q] arg
Triggers bundle purge job

Options:
-h, --help show this help message and exit
-r RETENTIONDAYS, --retentionDays=RETENTIONDAYS number of days to retain a
 bundle before purging
-H, --hard flag to indicate hard purge
-b BUNDLEID, --bundleId=BUNDLEID purge a particular bundle Id
-q, --quiet flag to purge quietly
```

Examples

1. Purge bundles older than 5 days:

```
# hst purge -r 5
```

```
Do you want to continue purging bundles older than 5 days ? y/n (default:
 n): y
Bundles purge job triggered successfully.
```

2. Hard purge bundles older than 20 days:

```
# hst purge -r 20 -H
Do you want to continue hard purging bundles older than 20 days ? y/n
 (default: n): y
Bundles purge job triggered successfully.
```

3. Manually trigger default purge process:

```
# hst purge
Do you want to continue purging bundles older than 30 days ? y/n (default:
 n): n
```

4. Hard purge a particular bundle:

```
# hst purge -b a-xxxxxxxx-c-xxxxxxxx_c6nr_0_2017-05-07_02-00-02 -H
Do you want to continue hard purging bundle : a-xxxxxxxx-c-
xxxxxxxx_c6nr_0_2017-05-07_02-00-02 ? y/n (default: n): y
Bundle purged successfully.
```

5. Purge a particular bundle:

```
# hst purge -b a-xxxxxxxx-c-xxxxxxxx_c6nr_0_2017-05-05_08-32-09
Do you want to continue purging bundle : a-xxxxxxxx-c-
xxxxxxxx_c6nr_0_2017-05-05_08-32-09 ? y/n (default: n): y
Bundle purged successfully.
```