CCP Setting Up Profiler 2.0.0

Setting Up a Profile

Date of publish: 2017-11-06

https://docs.cloudera.com/

Legal Notice

Contents

Setting Up a Profile Overview

A profile describes the behavior of an entity on a network. An entity can be a server, user, subnet, or application. Once you generate a profile defining what normal behavior looks like, you can build models that identify anomalous behavior.

Install Profiler

Data scientists use the Profiler to describe the behavior of entities on a network. The first step in setting up a Profiler is to install it.

Procedure

1. Build the Metron RPMs (see Building the RPMs).

   You might have already built the Metron RPMs when core Metron was installed.

   ```
   $ find metron-deployment/ -name "metron-profiler*.rpm"
   metron-deployment//packaging/docker/rpm-docker/RPMS/noarch/metron-
   profiler-0.4.1-201707131420.noarch.rpm
   ```

2. Copy the Profiler RPM to the installation host.

   The installation host must be the same host on which CCP was installed. Depending on how you installed CCP, the Profiler RPM might have already been copied to this host with the other CCP RPMs.

   [root@$METRON_HOME ~]# find /localrepo/ -name "metron-profiler*.rpm" /localrepo/metron-profiler-0.4.0-201707112313.noarch.rpm

3. Install the RPM.

   ```
   [root@$METRON_HOME ~]# rpm -ivh metron-profiler-*.noarch.rpm
   Preparing...                ########################################### 
    [100%]
      1:metron-profiler        ########################################### 
    [100%]
   ```

   ```
   [root@$METRON_HOME ~]# rpm -ql metron-profiler
   /usr/metron
   /usr/metron/0.4.1
   /usr/metron/0.4.1/bin
   /usr/metron/0.4.1/bin/start_profiler_topology.sh
   /usr/metron/0.4.1/config
   /usr/metron/0.4.1/config/profiler.properties
   /usr/metron/0.4.1/flux
   /usr/metron/0.4.1/flux/profiler
   /usr/metron/0.4.1/flux/profiler/remote.yaml
   /usr/metron/0.4.1/lib
   /usr/metron/0.4.1/lib/metron-profiler-0.4.0-uber.jar
   ```

4. Create a table within HBase that will store the profile data. By default, the table is named profiler with a column family P.

   ```
   $ /usr/hdp/current/hbase-client/bin/hbase shell
   hbase(main):001:0> create 'profiler', 'P'
   ```

   The table name and column family must match the Profiler's configuration (see Metron Profiler for Storm or Metron Profiler for Spark).

5. Edit the configuration file located at $METRON_HOME/config/profiler.properties.

```
kafka.zk=node1:2181
kafka.broker=node1:6667
```

Change kafka.zk to refer to ZooKeeper in your environment.

Change kafka.broker to refer to a Kafka Broker in your environment.

6. Start the Profiler topology.

```
$ cd $METRON_HOME
$ bin/start_profiler_topology.sh
```

At this point the Profiler is running and consuming telemetry messages. We have not defined any profiles yet, so it is not doing anything very useful. The next section walks you through the steps to create your very first "Hello, World!" profile.

Create a Profile

After you install Profiler, you must define the profile and upload the definition to ZooKeeper.

Procedure

1. Create a table within HBase that will store the profile data.

```
$ /usr/hdp/current/hbase-client/bin/hbase shell
hbase(main):001:0> create 'profiler', 'P'
```

The table name and column family must match the Profiler's configuration.

2. Define the profile in a file located at $METRON_HOME/config/zookeeper/profiler.json.

The following example JSON creates a profile that simply counts the number of messages per ip_src_addr during each sampling interval.

```
{
  "profiles": [
    {
      "profile": "test",
      "foreach": "ip_src_addr",
      "init":    { "count": 0 },
      "update":  { "count": "count + 1" },
      "result":  "count"
    }
  ]
}
```

The following table lists the supported profile elements:

Name	Description	
profile	Required	A unique name identifying the profile. The field is treated as a string.

Name	Description	
foreach	Required	A separate profile is maintained 'for each' of these. This is effectively the entity that the profile is describing. The field is expected to contain a Stellar expression whose result is the entity name. For example, if ip_src_addr then a separate profile would be maintained for each unique IP source address in the data; 10.0.0.1, 10.0.0.2, etc.
onlyif	Optional	An expression that determines if a message should be applied to the profile. A Stellar expression that returns a Boolean is expected. A message is only applied to a profile if this expression is true. This allows a profile to filter the messages that get applied to it.
groupBy	Optional	One or more Stellar expressions used to group the profile measurements when persisted. This is intended to sort the Profile data to allow for a contiguous scan when accessing subsets of the data. The 'groupBy' expressions can refer to any field within aorg.apache.metron.profiler.ProfileMeasurement. A common use case would be grouping by day of week. This allows a contiguous scan to access all profile data for Mondays only. Using the following definition would achieve this. `"groupBy": [ "DAY_OF_WEEK()" ]`
init	Optional	One or more expressions executed at the start of a window period. A map is expected where the key is the variable name and the value is a Stellar expression. The map can contain 0 or more variables/expressions. At the start of each window period the expression is executed once and stored in a variable with the given name. `"init": { "var1": "0", "var2": "1" }`

Name	Description	
update	Required	One or more expressions executed when a message is applied to the profile. A map is expected where the key is the variable name and the value is a Stellar expression. The map can include 0 or more variables/expressions. When each message is applied to the profile, the expression is executed and stored in a variable with the given name. `"update": { "var1": "var1 + 1", "var2": "var2 + 1" }`
result	Required	A Stellar expression that is executed when the window period expires. The expression is expected to summarize the messages that were applied to the profile over the window period. The expression must result in a numeric value such as a Double, Long, Float, Short, or Integer. For more advanced use cases, a profile can generate two types of results. A profile can define one or both of these result types at the same time. • profile: A required expression that defines a value that is persisted for later retrieval. • triage: An optional expression that defines values that are accessible within the Threat Triage process.
expires	Optional	A numeric value that defines how many days the profile data is retained. After this time, the data expires and is no longer accessible. If no value is defined, the data does not expire.

3. Upload the profile definition to ZooKeeper:

```
$ cd /$METRON_HOME/
$ bin/zk_load_configs.sh -m PUSH -i config/zookeeper/ -z
 $ZOOKEEPER_HOST:2181
```

4. Start the Profiler topology:

```
$ bin/start_profiler_topology.sh
```

5. Ensure that test messages are being sent to the Profiler's input topic in Kafka.

   The Profiler will consume messages from the inputTopic defined in the Profiler's configuration.

6. Check the HBase table to validate that the Profiler is writing the profile.

```
$ /usr/hdp/current/hbase-client/bin/hbase shell
hbase(main):001:0> count 'profiler'
*** Output Information ***
```

   Remember that the Profiler is flushing the profile every 15 minutes. You will need to wait at least this long to start seeing profile data in HBase.

7. Use the Profiler Client to read the profile data.

The following example PROFILE_GET command reads data written by the sample profile given above, if 10.0.0.1 is one of the input values for ip_src_addr. For more information on using the API client, refer to Accessing Profiles.

```
$ bin/stellar -z $ZOOKEEPER_HOST:2181

[Stellar]>>> PROFILE_GET( "test", "10.0.0.1", PROFILE_FIXED(30,
 "MINUTES"))
[451, 448]
```

Profiler Configuration Settings

The Profiler is installed during the Cloudera Cybersecurity Platform (CCP) installation and runs as an independent Storm topology. The configuration for the Profiler topology is stored in ZooKeeper at /metron/topology/profiler. These properties also exist in the default installation of CCP at $METRON_HOME/config/zookeeper/profiler.json. The profiler values can be changed on disk and then uploaded to ZooKeeper using $METRON_HOME/bin/zk_load_configs.sh.

You might need to work with your Platform Engineer to modify or tune the Profiler values.

Settings.	Description
profiler.workers	The number of worker processes to create for the topology.
profiler.executors	The number of executors to spawn per component.
profiler.input.topic	The name of the Kafka topic from which to consume data.
profiler.output.topic	The name of the Kafka topic to which profile data is written. Only used with profiles that use the triage result field.
profiler.period.duration	The duration of each profile period. This value should be define along with profiler.period.duration.units.
profiler.period.duration.units	The units used to specify the profile period duration. This value should be defined along with profiler.period.duration.
profiler.ttl	If a message has not been applied to a Profile in this period of time, the Profile will be forgotten and its resources will be cleaned up. This value should be defined along with profiler.ttl.units.
profiler.ttl.units	The units used to specify the profiler.ttl.
profiler.hbase.salt.divisor	A salt is prepended to the row key to help prevent hotspotting. This constant is used to generate the sale. Ideally, this constant should be roughly equal to the number of nodes in the HBase cluster.
profiler.hbase.table	The name of the HBase table that profiles are written to.
profiler.hbase.column.family	The column family used to store profiles.
profiler.hbase.batch	The number of puts that are written in a single batch.
profiler.hbase.flush.interval.seconds	The maximum number of seconds between batch writes to HBase.

Start the Profiler

After you install and configure the Profiler, you can start the profiler.

Procedure

Start the Profiler using the following command:

```
$METRON_HOME/bin/start_profiler_topology.sh
```

Develop Profiles

Troubleshooting issues when programming against a live stream of data can be difficult. The Stellar REPL (an interactive top level or language shell) is a powerful tool to help work out the kinds of enrichments and transformations that are needed. The Stellar REPL can also be used to help when developing profiles for the Profiler.

Procedure

1. Take a first pass at defining your profile.

   For example, in the editor copy/paste the basic Hello, World profile below.

```
[Stellar]>>> conf := SHELL_EDIT()
[Stellar]>>> conf
{
  "profiles": [
    {
      "profile": "hello-world",
      "onlyif":  "exists(ip_src_addr)",
      "foreach": "ip_src_addr",
      "init":    { "count": "0" },
      "update":  { "count": "count + 1" },
      "result":  "count"
    }
  ]
}
```

2. Initialize the Profiler.

```
[Stellar]>>> profiler := PROFILER_INIT(conf)
[Stellar]>>> profiler
org.apache.metron.profiler.StandAloneProfiler@4f8ef473
```

3. Create a message to simulate the type of telemetry that you expect to be profiled.

   For example, in the editor copy/paste the JSON below.

```
[Stellar]>>> message := SHELL_EDIT()
[Stellar]>>> message
{
   "ip_src_addr": "10.0.0.1",
   "protocol": "HTTPS",
   "length": "10",
   "bytes_in": "234"
}
```

4. Apply some telemetry messages to your profiles. The following applies the same message 3 times.

```
[Stellar]>>> PROFILER_APPLY(message, profiler)
org.apache.metron.profiler.StandAloneProfiler@4f8ef473

[Stellar]>>> PROFILER_APPLY(message, profiler)
org.apache.metron.profiler.StandAloneProfiler@4f8ef473

[Stellar]>>> PROFILER_APPLY(message, profiler)
```

```
org.apache.metron.profiler.StandAloneProfiler@4f8ef473
```

5. Flush the Profiler to see what has been calculated.

```
[Stellar]>>> values := PROFILER_FLUSH(profiler)
[Stellar]>>> values
[{period={duration=900000, period=1669628, start=1502665200000,
 end=1502666100000},
   profile=hello-world, groups=[], value=3, entity=10.0.0.1}]
```

A flush is what occurs at the end of each 15 minute period in the Profiler. The result is a list of profile measurements. Each measurement is a map containing detailed information about the profile data that has been generated.

This profile counts the number of messages by IP source address. Notice that the value is '3' for the entity '10.0.0.1' as we applied 3 messages with an 'ip_src_addr' of '10.0.0.1'. There will always be one measurement for each [profile, entity] pair.

6. If you are unhappy with the data that has been generated, then 'wash, rinse and repeat' this process. After you are satisfied with the data being generated by the profile, then follow the Getting Started guide to use the profile against your live, streaming data in a Metron cluster.

Testing

After you install, configure, and start Profiler, you should validate that the Profiler is working correctly.

Procedure

1. Login to the server hosting Metron.
2. Enter the following command:

```
[root@node1 0.3.0]# bin/stellar -z $ZOOKEEPER_HOST:2181
Stellar, Go!
Please note that functions are loading lazily in the background and will
 be unavailable until loaded fully.
{es.clustername=metron, es.ip=node1, es.port=9300,
 es.date.format=yyyy.MM.dd.HH}

[Stellar]>>> ?PROFILE_GET
Functions loaded, you may refer to functions now...
PROFILE_GET
Description: Retrieves a series of values from a stored profile.

Arguments:
    profile - The name of the profile.
    entity - The name of the entity.
    durationAgo - How long ago should values be retrieved from?
    periods - The list of profile periods to grab. These are ProfilePeriod
 objects.
    groups - Optional, must correspond to the 'groupBy' list used in
 profile creation - List (in square brackets) of groupBy values used to
 filter the profile. Default is the empty list, meaning groupBy was not
 used when creating the profile.
    config_overrides - Optional - Map (in curly braces) of name:value
 pairs, each overriding the global config parameter of the same name.
 Default is the empty Map, meaning no overrides.
Returns: The profile measurements.

[Stellar]>>> PROFILE_GET('test','192.168.138.158', 1, 'HOURS') [12078.0,
 8921.0, 12131.0]
```

In the preceeding example, we use the Stellar Shell to replicate the execution environment of Stellar running in a Storm topology, like Metron's Parser or Enrichment topology. Replace 'node1:2181' with the URL to a ZooKeeper Broker.

The client API call above retrieves the past hour of the test profile for the entity 192.168.138.158.