HCP Analytics 1

Hortonworks Cybersecurity Platform

Date of Publish: 2018-07-15

http://docs.hortonworks.com

Contents

Introduction to HCP Analytics

Data Scientists are frequently responsible for performing data science life cycle activities, including training, evaluating, and scoring analytical models. HCP provides the ability to create profiles and models, analyze data using statistical and mathematical functions and Apache Zeppelin and create runbooks for SOC analysts and investigators.

Creating Profiles

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.

Any field contained within a message can be used to generate a profile. A profile can even be produced by combining fields that originate in different data sources. You can transform the data used in a profile by leveraging the Stellar language.

The Profiler is automatically installed and started when you install HCP through Ambari install.

Configuring the Profiler

The configuration for the Profiler topology is stored in ZooKeeper at /metron/topology/profiler. These properties also exist in the default installation of HCP at $METRON_HOME/config/zookeeper/profiler.json. You can change these values two ways: with Ambari or on disk and then uploaded to ZooKeeper using $METRON_HOME/bin/zk_load_configs.sh. The following task uses Ambari to configure the Profiler.

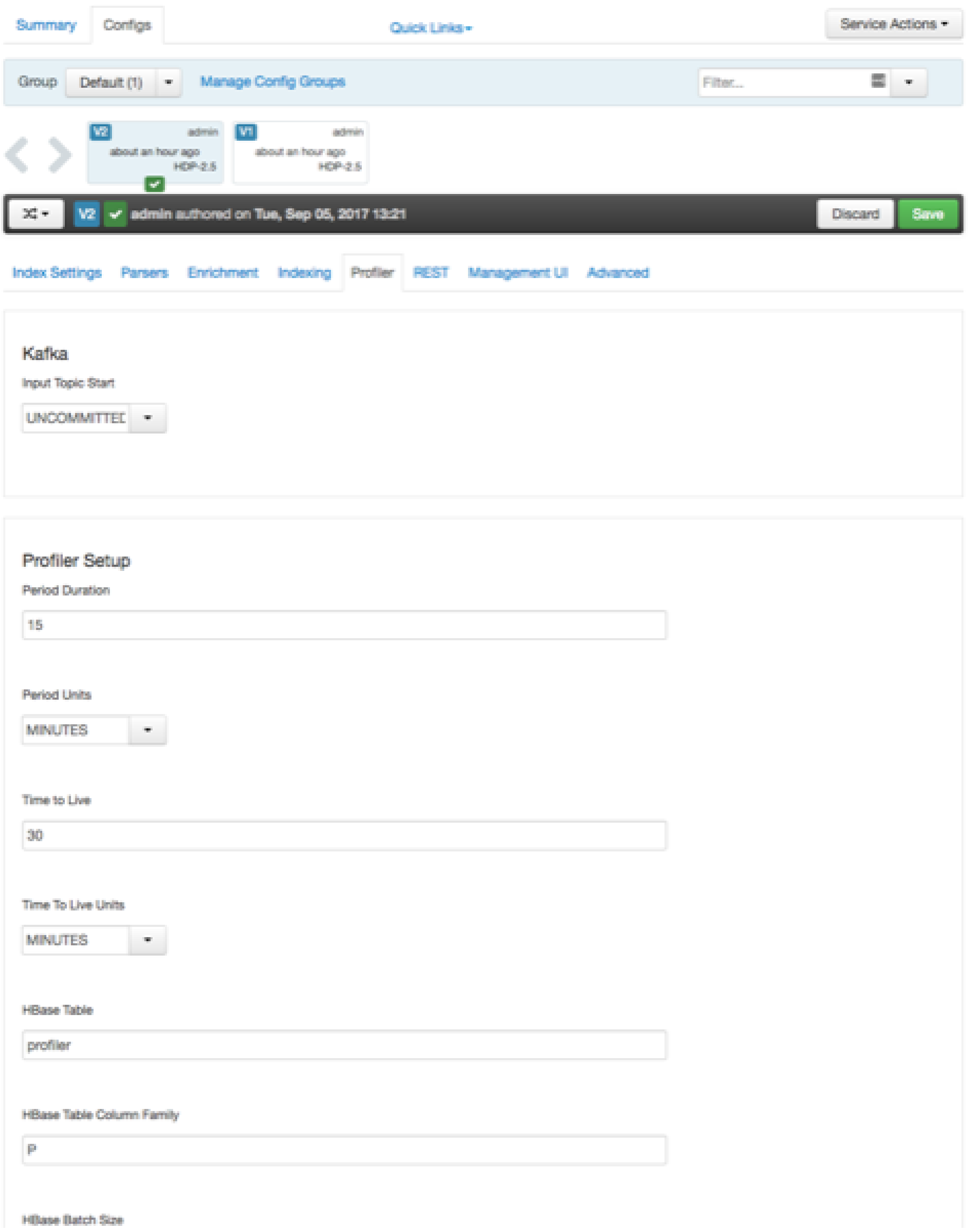
Procedure

1. Display the Ambari user interface and click the **Services** tab.

   Note: You might need to work with your Platform Engineer to modify Profiler values.
2. Click **Metron** in the list of services, then click the **Configs** tab.
3. Click the **Profiler** tab.

   Ambari displays a list of Profiler properties that you can use to configure Profiler.

   Profiler Properties in Ambari



Kafka

Input Topic Start: UNCOMMITTED

Profiler Setup

Period Duration: 15

Period Units: MINUTES

Time to Live: 30

Time To Live Units: MINUTES

HBase Table: profiler

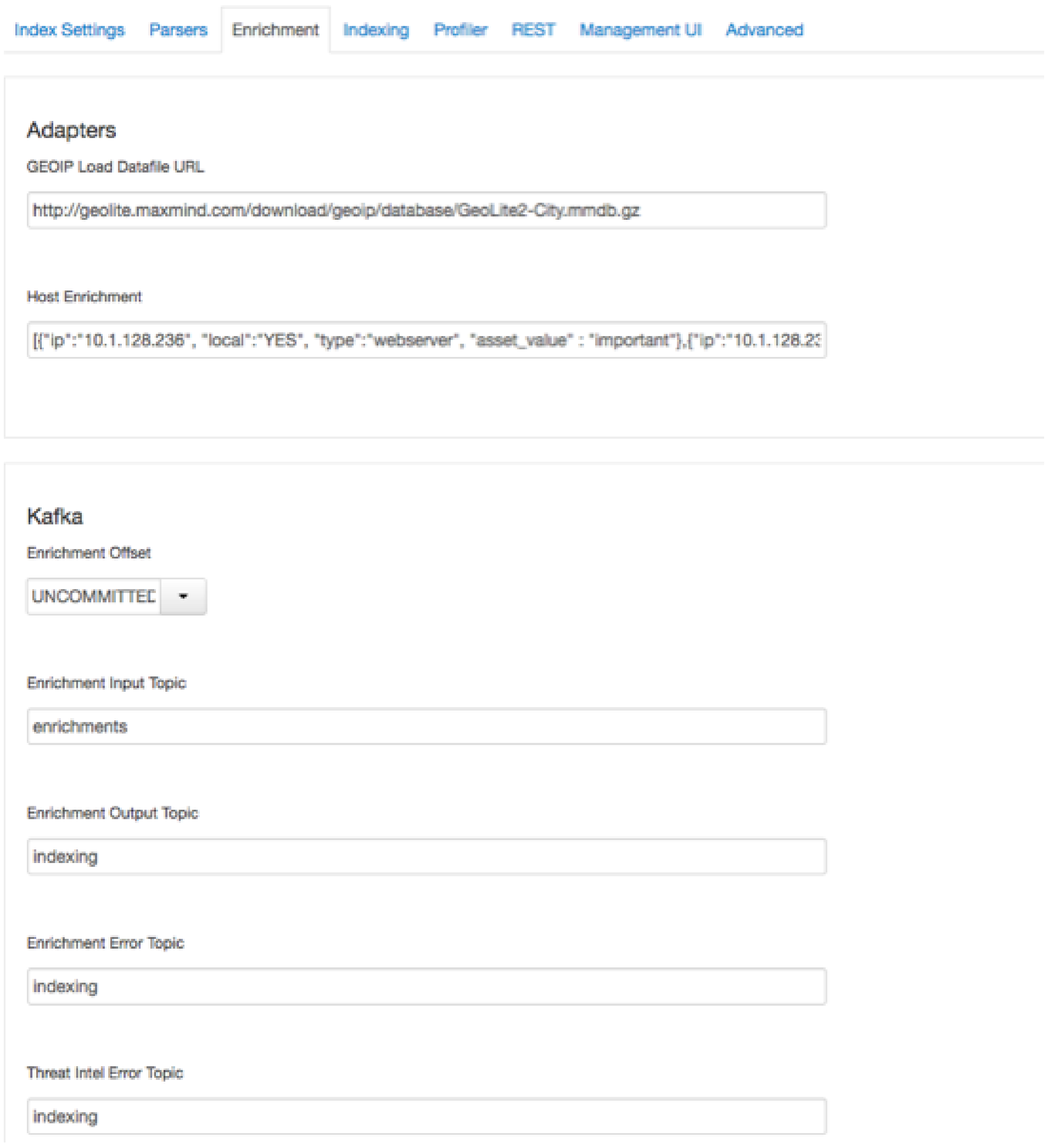
HBase Table Column Family: P

HBase Batch Size

4. Use these properties to configure the Profiler, then click the Save button near the top of the window.

   The profiler input topic is bound to the enrichment output topic. If that enrichment output topic is changed, then the profiler will restart as well as the enrichment topology.

   Enrichment Output Topic



Adapters

GEOIP Load Datafile URL

http://geolite.maxmind.com/download/geoip/database/GeoLite2-City.mmdb.gz

Host Enrichment

[{"ip":"10.1.128.236", "local":"YES", "type":"webserver", "asset_value" : "important"},{"ip":"10.1.128.23

Kafka

Enrichment Offset

UNCOMMITTEL

Enrichment Input Topic

enrichments

Enrichment Output Topic

indexing

Enrichment Error Topic

indexing

Threat Intel Error Topic

indexing

Profiler Properties

Use the Profiler Properties to configure the Profiler.

Table 1: Profiler Properties

Ambari Configs Field.	Settings.	Description
Kafka Input Topic Start	profiler.kafka.start	One of EARLIEST, LATEST, UNCOMMITTED_EARLIEST, UNCOMMITTED_LATEST

Profiler Setup Period Duration	profiler.period.duration	The duration of each profile period. This value should be defined along with profiler.period.duration.units.
Period Units	profiler.period.units	The units is used to specify the profiler.period.duration. This value should be define along with profiler.period.duration.
Time to Live	profiler.ttl	If a message has not been applied to a Profile in this period of time, the Profile will be terminated and its resources will be cleaned up. This value should be defined along with profiler.ttl.units. This time-to-live does not affect the persisted Profile data in HBase. It only affects the state stored in memory during the execution of the latest profile period. This state will be deleted if the time.to.live is exceeded.
Time to Live Units	profiler.ttl.units	The units used to specify the profiler.ttl.
HBase Table	profiler.hbase.table	The name of the HBase table the profiler is written to. The profiler expects that the table exists and is writable.
HBase Table Column Family	profiler.hbase.cf	The column family used to store profile data in HBase.
HBase Batch Size	profiler.hbase.batch	The number of puts that are written to HBase in a single batch.
HBase Flush Interval	profiler.hbase.flush.interval	The maximum number of seconds between batch writes to HBase.
Storm topology.worker.childopts	profiler.topology.worker.childopts	Extra topology child opts for the storm opts.
Number of Workers	profiler.topology.workers	The profiler storm topology storm workers
Number of Acker Executors	profiler.acker.executors	The profiler storm topology acker execuors
N/A	profiler.writer.batchSize	The size of the batch that is written to Kafka at once. Defaults to 15 (size of 1 disables batching).
N/A	profiler.writer.batchTimeout	The timeout after which a batch will be flushed even if batchSize has not been met. Optional. If unspecified or set to 0, it defaults to a system-determined duration which is a fraction of the Storm parameter topology.message.timeout.secs. Ignored if batchsize is 1 because this disables batching.

Starting and Stopping the Profiler

The Profiler is automatically started when you install HCP. However, you can also manually stop or restart the Profiler.

Procedure

1. Display the Ambari user interface.
2. Select the **Services** tab, then select **Metron** from the list of services.

   Ambari displays a list of Metron components.
3. Select **Metron Profiler**.

   Metron displays a complete list of Metron components.
4. Select the pull down menu next to **Metron Profiler / Metron** and select the appropriate status:

   - Restart

- Stop
- Turn on Maintenance Mode

Developing 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.

About this task

Follow these steps in the Stellar REPL to see how it can be used to help create profiles.

Procedure

1. Take a first pass at defining your profile.

   As an 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.

   As an 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.

   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.

   ```
   [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}]
   ```

   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 use the profile against your live, streaming data in a Metron cluster.

Profile Examples

You can use the Profile examples to better understand the functionality provided by the Profiler. Each example shows the configuration that would be required to generate the profile.

These examples assume a fictitious input message stream that looks something like the following:

```
{
  "ip_src_addr": "10.0.0.1",
  "protocol": "HTTPS",
  "length": "10",
  "bytes_in": "234"
},
{
  "ip_src_addr": "10.0.0.2",
  "protocol": "HTTP",
  "length": "20",
  "bytes_in": "390"
},
{
  "ip_src_addr": "10.0.0.3",
  "protocol": "DNS",
  "length": "30",
  "bytes_in": "560"
}
```

Example 1

The total number of bytes of HTTP data for each host. The following configuration would be used to generate this profile.

```
{
  "profiles": [
    {
      "profile": "example1",
      "foreach": "ip_src_addr",
      "onlyif": "protocol == 'HTTP'",
      "init": {
        "total_bytes": 0.0
      },
```

```
      "update": {
        "total_bytes": "total_bytes + bytes_in"
      },
      "result": "total_bytes",
      "expires": 30
    }
  ]
}
```

This creates a profile with the following parameters:

- Named ‘example1’
- That for each IP source address
- Only if the 'protocol' field equals 'HTTP'
- Initializes a counter ‘total_bytes’ to zero
- Adds to ‘total_bytes’ the value of the message's ‘bytes_in’ field
- Returns ‘total_bytes’ as the result
- The profile data will expire in 30 days

Example 2

The ratio of DNS traffic to HTTP traffic for each host. The following configuration would be used to generate this profile.

```
{
  "profiles": [
    {
      "profile": "example2",
      "foreach": "ip_src_addr",
      "onlyif": "protocol == 'DNS' or protocol == 'HTTP'",
      "init": {
        "num_dns": 1.0,
        "num_http": 1.0
      },
      "update": {
        "num_dns": "num_dns + (if protocol == 'DNS' then 1 else 0)",
        "num_http": "num_http + (if protocol == 'HTTP' then 1 else 0)"
      },
      "result": "num_dns / num_http"
    }
  ]
}
```

This creates a profile with the following parameters:

- Named ‘example2’
- That for each IP source address
- Only if the 'protocol' field equals 'HTTP' or 'DNS'
- Accumulates the number of DNS requests
- Accumulates the number of HTTP requests
- Returns the ratio of these as the result

Example 3

The average of the length field of HTTP traffic. The following configuration would be used to generate this profile.

```
{
  "profiles": [
    {
      "profile": "example3",
      "foreach": "ip_src_addr",
      "onlyif": "protocol == 'HTTP'",
      "update": { "s": "STATS_ADD(s, length)" },
      "result": "STATS_MEAN(s)"
    }
  ]
}
```

This creates a profile with the following parameters:

- Named ‘example3’
- That for each IP source address
- Only if the 'protocol' field is 'HTTP'
- Adds the length field from each message
- Calculates the average as the result

Example 4

It is important to note that the Profiler can persist any serializable Object, not just numeric values. An alternative to the previous example could take advantage of this.

Instead of storing the mean of the length, the profile could store a more generic summary of the length. This summary can then be used at a later time to calculate the mean, min, max, percentiles, or any other sensible metric. This provides a much greater degree of flexibility.

```
{
  "profiles": [
    {
      "profile": "example4",
      "foreach": "ip_src_addr",
      "onlyif": "protocol == 'HTTP'",
      "update": { "s": "STATS_ADD(s, length)" },
      "result": "s"
    }
  ]
}
```

The following Stellar REPL session shows how you might use this summary to calculate different metrics with the same underlying profile data.

Retrieve the last 30 minutes of profile measurements for a specific host.

```
$ bin/stellar -z node1:2181

[Stellar]>>> stats := PROFILE_GET("example4", "10.0.0.1", PROFILE_FIXED(30,
 "MINUTES"))
[Stellar]>>> stats
[org.apache.metron.common.math.stats.OnlineStatisticsProvider@79fe4ab9, ...]
```

Calculate different metrics with the same profile data.

```
[Stellar]>>> STATS_MEAN( GET_FIRST( stats))
15979.0625

[Stellar]>>> STATS_PERCENTILE( GET_FIRST(stats), 90)
30310.958
```

Merge all of the profile measurements over the past 30 minutes into a single summary and calculate the 90th percentile.

```
[Stellar]>>> merged := STATS_MERGE( stats)
[Stellar]>>> STATS_PERCENTILE(merged, 90)
```

```
29810.992
```

Accessing Profiles

You can use a client API to access the profiles generated by the HCP Profiler to use for model scoring. HCP provides a Stellar API to access the profile data but this section provides only instructions for using the Stellar client API. You can use this API in conjunction with other Stellar functions such as MAAS_MODEL_APPLY to perform model scoring on streaming data.

Selecting Profile Measurements

The PROFILE_GET command allows you to select all of the profile measurements written.

This command takes the following arguments:

REQUIRED::

profile	The name of the profile
entity	The name of the entity
periods	The list of profile periods to grab. These are ProfilePeriod objects. This field is generally the output of another Stellar function which defines the times to include.

OPTIONAL:

groups_list

List (in square brackets) of groupBy values used to filter the profile. Default is an empty list, which means that groupBy was not used when creating the profile. This list must correspond to the 'groupBy' list used in profile creation.

The groups_list argument in the client must exactly correspond to the groupBy configuration in the profile definition. If groupBy was not used in the profile, groups_list must be empty in the client. If groupBy was used in the profile, then the client groups_list is not optional; it must be the same length as the groupBy list, and specify exactly one selected group value for each groupBy criterion, in the same order. For example:

```
If in Profile, the groupBy
 criteria are:  [ “DAY_OF_WEEK()”,
 “URL_TO_PORT()” ]
Then in PROFILE_GET, an allowed
 groups value would be:  [ “3”,
 “8080” ]
which will select only records from
 Tuesdays with port number 8080.
```

config_overrides

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.

Note:

There is an older calling format where groups_list is specified as a sequence of group names, "varargs" style, instead of a List object. This format is still supported for backward compatibility, but it is deprecated, and it is disallowed if the optional config_overrides argument is used.

By default, the Profiler creates profiles with a period duration of 15 minutes. This means that data is accumulated, summarized, and flushed every 15 minutes. The Client API must also have knowledge of this duration to correctly retrieve the profile data. If the Client is expecting 15 minute periods, it will not be able to read data generated by a Profiler that was configured for 1 hour periods, and will return zero results.

Similarly, all six Client configuration parameters listed in the table below must match the Profiler configuration parameter settings from the time the profile was created. The period duration and other configuration parameters from the Profiler topology are stored in a local file system at $METRON_HOME/config/profiler.properties. The Stellar Client API can be configured correspondingly by setting the following properties in HCP's global configuration, on a local file system at $METRON_HOME/config/zookeeper/global.json, then uploaded to ZooKeeper (at /metron/topology/global) by using zk_load_configs.sh:

```
                              ```
$ cd $METRON_HOME
$ bin/zk_load_configs.sh -m PUSH -i
 config/zookeeper/ -z node1:2181
```
```

Any of these six Client configuration parameters may be overridden at run time using the config_overrides Map argument in PROFILE_GET. The primary use case for overriding the client configuration parameters is when historical profiles have been created with a different Profiler configuration than is currently configured, and the analyst, needing to access them, does not want to change the global Client configuration so as not to disrupt the work of other analysts working with current profiles.

Table 2: Profiler Client Configuration Parameters

Key	Description	Required	Default
profiler.client.period.duration	The duration of each profile period. This value should be defined along with profiler.client.period.duration.units.	Optional	15
profiler.client.period.duration.units	The units used to specify the profile period duration. This value should be defined along with profiler.client.period.duration.	Optional	MINUTES
profiler.client.hbase.table	The name of the HBase table used to store profile data.	Optional	profiler
profiler.client.hbase.column.family	The name of the HBase column family used to store profile data.	Optional	P
profiler.client.salt.divisor	The salt divisor used to store profile data.	Optional	1000
hbase.provider.impl	The name of the HBaseTableProvider implementation class.	Optional	

Related Information

Specifying Profile Time

Specifying Profile Time and Duration

The third required argument for PROFILE_GET is a list of ProfilePeriod objects. These objects allow you to specify the timing, frequency, and duration of the PROFILE_GET. This list is produced by another Stellar function. There are two options available: PROFILE_FIXED and PROFILE_WINDOW.

PROFILE_FIXED	PROFILE_FIXED specifies a fixed period to look back at the profiler data starting from now. These are ProfilePeriod objects.	
REQUIRED:	**durationAgo**	How long ago should values be retrieved from?
	units	The units of 'durationAgo'.
OPTIONAL:	**config_overrides**	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. For example, to retrieve all the profiles for the last 5 hours: PROFILE_GET('profile', 'entity', PROFILE_FIXED(5, 'HOURS'))
PROFILE WINDOW	PROFILE_WINDOW provides a finer-level of control over selecting windows for profiles. This profile selector allows you to specify the exact time, duration, and frequency for the profile. It does this by a domain specific language that mimics natural language that defines the excluded windows. You can use PROFILE_WINDOW to specify: • Windows relative to the data timestamp (see the optional now parameter below) • Non-contiguous windows to better handle seasonal data (for example, the last hour for every day for the last month) • Profile output excluding holidays • Only profile output on a specific day of the week	
REQUIRED:	**windowSelector**	The statement specifying the window to select.
	now	Optional - The timestamp to use for now.
OPTIONAL:	**config_overrides**	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.

For example, to retrieve all the measurements written for 'profile' and 'entity' for the last hour on the same weekday excluding weekends and US holidays across the last 14 days:

```
PROFILE_GET('profile', 'entity', PROFILE_WINDOW('1 hour window every 24
 hours starting from 14 days ago including the current day of the week
 excluding weekends, holidays:us'))
```

Note: The config_overrides parameter operates exactly as the config_overrides argument in PROFILE_GET. The only available parameters for override are:

- profiler.client.period.duration
- profiler.client.period.duration.units

Profile Selector Language

The domain specific language for the profile selector can be broken into a series of clauses, some of which are optional.

Total Temporal Duration	The total range of time in which windows may be specified
Temporal Window Width	The size of each temporal window
Skip distance	(optional) How far to skip between when one window starts and when the next begins
Inclusion/Exclusion specifiers	(optional) The set of specifiers to further filter the window

You must specify either a total temporal duration or a temporal window width. The remaining clauses are optional.

From a high level, the domain specific language fits the following three forms, which are composed of the clauses above:

- time_interval Window (INCLUDING specifier list) (EXCLUDING specifier list)

  temporal window width inclusion specifiers exclusion specifier
- time_interval WINDOW EVERY time_interval FROM time_interval (TO time_interval) (INCLUDING specifier_list) (EXCLUDING specifier list)

  temporal window width skip distance total temporal duration inclusion specifiers exclusion specifier
- FROM time_interval (TO time_interval)

  total temporal duration total temporal duration

Total Temporal Duration

Total temporal duration is specified by a phrase: FROM time_interval AGO TO time_interval AGO. This indicates the beginning and ending of a time interval. This is an inclusive duration.

FROM	Can be the words "from" or "starting from".
time_interval	A time amount followed by a unit (for example, 1 hour). Fractional amounts are not supported. The unit may be "minute", "day", "hour" with any pluralization.
TO	Can be the words "until" or "to".
AGO	(optional) The word "ago"

The TO time_interval AGO portion is optional. If this portion is unspecified then it is expected that the time interval ends now.

Due to the vagaries of the English language, the from and the to portions, if both are specified, are interchangeable with regard to which one specifies the start and which specifies the end. In other words "starting from 1 hour ago to 30 minutes ago" and "starting from 30 minutes ago to 1 hour ago" specify the same temporal duration.

Total Temporal Duration Examples

The domain specific language allows for some flexibility on how to specify a duration. The following are examples of various ways you can specify the same duration.

- A duration starting 1 hour ago and ending now:
  - from 1 hour ago
  - from 1 hour

  - starting from 1 hour ago
  - starting from 1 hour
- A duration starting 1 hour ago and ending 30 minutes ago:
  - from 1 hour ago until 30 minutes ago
  - from 30 minutes ago until 1 hour ago
  - starting from 1 hour ago to 30 minutes ago
  - starting from 1 hour to 30 minutes

Temporal Window Width

Temporal window width is the specification of a window. A window may either repeat within a total temporal duration or it may fill the total temporal duration. This is an inclusive window. A temporal window width is specified by the phrase: time_interval WINDOW.

time_interval	A time amount followed by a unit (for example, 1 hour). Fractional amounts are not supported. The unit may be "minute", day", or "hour" with any pluralization.
WINDOW	(optional) The word "window".

Temporal Window Width Examples

- A fixed window starting 2 hours ago and going until now
  - 2 hour
  - 2 hours
  - 2 hours window
- A repeating 30 minute window starting 2 hours ago and repeating every hour until now. This would result in 2 30-minute wide windows: 2 hours ago and 1 hour ago
  - 30 minute window every 1 hour starting from 2 hours ago

    temporal window width skip distance total temporal duration
  - 30 minute windows every 1 hour from 2 hours ago

    temporal window width skip distance total temporal duration

Skip Distance

Skip distance is the amount of time between when one temporal window begins and the next window starts. It is, in effect, the window period. It is specified by the phrase EVERY time_interval.

time_interval	A time amount followed by a unit (for example, 1 hour). Fractional amounts are not supported. The unit may be "minute", "day", or "hour" with any pluralization.
EVERY	The word/phrase "every" or "for every".

Skip Distance Examples

- A repeating 30 minute window starting 2 hours ago and repeating every hour until now. This would result in 2 30-minute wide windows: 2 hours ago and 1 hour ago
  - 30 minute window every 1 hour starting from 2 hours ago

    temporal window width skip distance total temporal duration
  - 30 minutes window every 1 hour from 2 hours ago

    temporal window width skip distance total temporal duration
- A repeating 30 minute window starting 2 hours ago and repeating every hour until 30 minutes ago. This would result in 2 30-minute wide windows: 2 hours ago and 1 hour ago

- 30 minute window every 1 hour starting from 2 hours ago until 30 minutes ago

  temporal window width skip distance total temporal duration
- 30 minutes window every 1 hour from 2 hours ago to 30 minutes ago

  temporal window width skip distance total temporal duration
- 30 minutes window for every 1 hour from 30 minutes ago to 2 hours ago

  temporal window width skip distance total temporal duration

Inclusion/Exclusion Specifiers

Inclusion and Exclusion specifiers operate as filters on the set of windows. They operate on the window beginning timestamp.

For inclusion specifiers, windows that are passed by any of the set of inclusion specifiers are included. Similarly, windows that are passed by any of the set of exclusion specifiers are excluded. Exclusion specifiers trump inclusion specifiers.

Specifiers follow one of the following formats depending on if it is an inclusion or exclusion specifier:

- INCLUSION specifier, specifier, ...

  INCLUSION can be "include", "includes" or "including"
- EXCLUSION specifier, specifier, ...

  EXCLUSION can be "exclude", "excludes" or "excluding"

The specifiers are a set of fixed specifiers available as part of the language:

- Fixed day of week-based specifiers - includes or excludes if the window is on the specified day of the week
  - "monday" or "mondays"
  - "tuesday" or "tuesdays"
  - "wednesday" or "wednesdays"
  - "thursday" or "thursdays"
  - "friday" or "fridays"
  - "saturday" or "saturdays"
  - "sunday" or "sundays"
  - "weekday" or "weekdays"
  - "weekend" or ""weekends"
- Relative day of week-based specifiers - includes or excludes based on the day of week relative to now
  - "current day of the week"
  - "current day of week"
  - "this day of the week"
  - "this day of week"
- Specified date - includes or excludes based on the specified date
  - "date" - Takes up to 2 arguments
    - The day in yyyy/MM/dd format if no second argument is provided

      Example: date:2017/12/25 would include or exclude December 25, 2017
    - (optional) The format in which to specify the first argument

      Example: date:20171225:yyyyMMdd would include or exclude December 25, 2017
- Holidays - includes or excludes based on if the window starts during a holiday
  - "holiday" or "holidays"
    - Arguments form the jollyday hierarchy of holidays. For example, "us:nyc" would be holidays for New York City, USA

Countries supported are those supported in jollyday

Example: holiday:us:nyc would be the holidays of New York City, USA

- If none is specified, it will choose based on locale.

  Example: holiday:hu would be the holidays of Hungary

Inclusion/Exclusion Specifiers Examples

The following are inclusion/exclusion specifier examples and identify the various clauses used in these examples.

Assume the following examples are executed at noon.

- A 1 hour window for the past 8 'current day of the week'
  - 1 hour window every 24 hours from 56 days ago including this day of the week

    temporal window width skip distance total temporal duration inclusion/exclusion specifiers
- A 1 hour window for the past 8 tuesdays
  - 1 hour window every 24 hours from 56 days ago including tuesdays

    temporal window width skip distance total temporal duration inclusion/exclusion specifiers
- A 30 minute window every tuesday at noon starting 14 days ago until now
  - 30 minute window every 24 hours from 14 days ago including tuesdays

    temporal window width skip distance total temporal duration inclusion/exclusion specifiers
- A 30 minute window every day except holidays and weekends at noon starting 14 days ago until now
  - 30 minutes every 24 hours from 14 days ago excluding holidays:us, weekends

    30 minutes every 24 hours from 14 days ago including weekdays excluding holidays:us, weekends

    temporal window width skip distance total temporal duration inclusion/exclusion specifiers
- A 30 minute window at noon every day from 7 days ago including saturdays and excluding weekends. Because exclusions trump inclusions, the following will never yield any windows
  - 30 minute window every 24 hours from 7 days ago including saturdays excluding weekends

    temporal window width skip distance total temporal duration inclusion/exclusion specifiers

Related Information

Jollyday

Client Profile Example

The following are usage examples that show how the Stellar API can be used to read profiles generated by the Metron Profiler. This API would be used in conjunction with other Stellar functions like MAAS_MODEL_APPLY to perform model scoring on streaming data.

These examples assume a profile has been defined called ‘snort-alerts’ that tracks the number of Snort alerts associated with an IP address over time. The profile definition might look similar to the following.

```
{
  "profiles": [
    {
      "profile": "snort-alerts",
      "foreach": "ip_src_addr",
      "onlyif":  "source.type == 'snort'",
      "update":  { "s": "STATS_ADD(s, 1)" },
      "result":  "STATS_MEAN(s)"
    }
  ]
}
```

During model scoring, the entity being scored, in this case a particular IP address, will be known. The following examples shows how this profile data might be retrieved. Retrieve all values of ‘snort-alerts’ from ‘10.0.0.1’ over the past 4 hours.

```
PROFILE_GET('snort-alerts', '10.0.0.1', PROFILE_FIXED(4, 'HOURS'))
```

Retrieve all values of ‘snort-alerts’ from ‘10.0.0.1’ over the past 2 days.

```
PROFILE_GET('snort-alerts', '10.0.0.1', PROFILE_FIXED(2, 'DAYS'))
```

If the profile had been defined to group the data by weekday versus weekend, then the following example would apply:

Retrieve all values of ‘snort-alerts’ from ‘10.0.0.1’ that occurred on ‘weekdays’ over the past 30 days.

```
PROFILE_GET('snort-alerts', '10.0.0.1', PROFILE_FIXED(30, 'DAYS'),
 ['weekdays'] )
```

The client may need to use a configuration different from the current Client configuration settings. For example, perhaps you are on a cluster shared with other analysts, and need to access a profile that was constructed 2 months ago using different period duration, while they are accessing more recent profiles constructed with the currently configured period duration. For this situation, you may use the config_overrides argument:

Retrieve all values of ‘snort-alerts’ from ‘10.0.0.1’ over the past 2 days, with no groupBy, and overriding the usual global client configuration parameters for window duration.

```
PROFILE_GET('profile1', 'entity1', PROFILE_FIXED(2,
 'DAYS', {'profiler.client.period.duration' : '2',
 'profiler.client.period.duration.units' : 'MINUTES'}), [])
```

Retrieve all values of ‘snort-alerts’ from ‘10.0.0.1’ that occurred on ‘weekdays’ over the past 30 days, overriding the usual global client configuration parameters for window duration.

```
PROFILE_GET('profile1', 'entity1', PROFILE_FIXED(30,
 'DAYS', {'profiler.client.period.duration' : '2',
 'profiler.client.period.duration.units' : 'MINUTES'}), ['weekdays'] )
```

Creating Models

One of the enhancements to cybersecurity most frequently requested is the ability to augment the threat intelligence and enrichment processes with insights derived from machine learning and statistical models. While valuable, this model management infrastructure has significant challenges.

- Applying the model management infrastructure might be both computationally and resource intensive and could require load balancing and multiple versions of models.
- Models require frequent training or updating to react to growing threats and new patterns that emerge.
- Models should be language and environment agnostic as much as possible. So, models should include small-data and big-data libraries and languages.

To support these requirements, Hortonworks Cybersecurity Platform (HCP) powered by Metron provides the following components:

- A YARN application that listens for model deployment requests and upon execution, registers their endpoints in ZooKeeper.
- A command line deployment client that localizes the model payload onto HDFS and submits a model request.
- A Java client that interacts with ZooKeeper and receives updates about model state changes (for example, new deployments and removals).

- A series of Stellar functions for interacting with models deployed by the Model as a Service infrastructure.

Install the YARN Application

The YARN application listens for model deployment requests. Models are exposed as REST microservices that expose your model application as an endpoint. The YARN application takes the submitted request that specifies the model payload that includes a shell script and other model collateral which will start the microservice. Upon execution of the shell script that starts the model, the YARN application registers the endpoints in ZooKeeper.

About this task

If you are using or depending an API library in your model such as Flask and Jinja2, the library must be installed on every data node. This is because the model is executed by a shell script which must be able to run successfully on every node.

In order to know on which port that the REST service is listening, the model must create a file in the current working directory which indicates the URL for the model. Because you might have more than one copy of the model, it is a good idea to find an open port and bind to that. An example of how to do that in Python is as follows:

```
  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('localhost', 0))
port = sock.getsockname()[1]
sock.close()
with open("endpoint.dat", "w") as text_file:
  text_file.write("{\"url\" : \"http://0.0.0.0:%d\"}" % port)
```

Procedure

1. As root, log into the host from which you run Metron.
2. Create a directory called "sample" in the root user's home directory where you will put a very simple model.
3. Now, you can create a simple shell script that will expose a REST endpoint called "echo" that will echo back the arguments passed to it. Create a file in the "sample" directory named "echo.sh", and copy the following information into the file.

   Note: In this simple REST service, we are always binding to port 1500. In a real REST service which would expose your model, we would be more intelligent about the choice of the port.

```
#!/bin/bash
rm -f out
mkfifo out
trap "rm -f out" EXIT
echo "{ \"url\" : \"http://localhost:1500\", \"functions\" : { \"apply\" :
 \"echo\" } }" > endpoint.dat
while true
do
  cat out | nc -l 0.0.0.0 1500 > >( # parse the netcat output, to build
 the answer redirected to the pipe "out".
    export REQUEST=
    while read line
    do
      line=$(echo "$line" | tr -d '[\r\n]')

      if echo "$line" | grep -qE '^GET /' # if line starts with "GET /"
      then
        REQUEST=$(echo "$line" | cut -d ' ' -f2) # extract the request
      elif [ "x$line" = x ] # empty line / end of request
      then
        HTTP_200="HTTP/1.1 200 OK"
        HTTP_LOCATION="Location:"
        HTTP_404="HTTP/1.1 404 Not Found"
        # call a script here
        # Note: REQUEST is exported, so the script can parse it (to answer
 200/403/404 status code + content)
        if echo $REQUEST | grep -qE '^/echo/'
        then
            printf "%s\n%s %s\n\n%s\n" "$HTTP_200" "$HTTP_LOCATION"
 $REQUEST ${REQUEST#"/echo/"} > out
        else
            printf "%s\n%s %s\n\n%s\n" "$HTTP_404" "$HTTP_LOCATION"
 $REQUEST "Resource $REQUEST NOT FOUND!" > out
        fi
      fi
    done
  )
done
```

4. Change directories to $METRON_HOME.

```
cd $METRON_HOME
```

5. Start the MaaS service in bin/maas_service.sh -zq node1:2181.

```
bash bin/maas_service.sh -zq node1:2181
```

where

-c, --create	Flag to indicate whether to create the domain specified with -domain.
-d,--domain <arg>	ID of the time line domain where the time line entities will be put
-e,--shell_env <arg>	Environment for shell script. Specified as env_key=env_val pairs.
-h,--help	The help screen
-j,--jar <arg>	Jar file containing the application master
-l,--log4j <arg>	The log4j properties file to load
-ma,--modify_acls <arg>	Users and groups that allowed to modify the time line entities in the given domain
-ma,--master_vcores <arg>	Amount of virtual cores to be requested to run the application master
-mm,--master_memory	Amount of memory in MB to be requested to run the application master
-nle,--node_label_expression <arg>	Node label expression to determine the nodes where all the containers of this application will be allocated, "" means containers can be allocated anywhere, if you don't specify the option, default node_label_expression of queue will be used.
-q,--queue <arg>	RM Queue in which this application is to be submitted

-t,--timeout <arg>	Application timeout in milliseconds
-va,--view_acls <arg>	Users and groups that allowed to view the time line entities in the given domain
-zq,--zk_quorum <arg>	ZooKeeper Quorum
-zr,--zk_root <arg>	ZooKeeper Root

6. Test the configuration to ensure that the MaaS service is running correctly.

For example, you would enter the following:

a) Start one instance of a sample echo service (named 'sample' version '1.0') in a container of 500m:

```
bin/maas_deploy.sh -lmp ~/sample -hmp /user/root/maas/sample -m 500 -mo
 ADD -n sample -ni 1 -v 1.0 -zq node1:2181
```

b) Wait a couple seconds and then ensure that the service started by running the following command:

```
curl -i http://localhost:1500/echo/foobar
```

You should see a response foobar.

c) List the active models and ensure that you see the sample model in the output.

```
bin/maas_deploy.sh -mo LIST -n sample -zq node1:2181
```

d) Remove one instance of the sample model.

```
bin/maas_deploy.sh -mo REMOVE -n sample -ni 1 -v 1.0 -zq node1:2181
```

e) After a couple seconds ensure that you cannot access the sample model any longer:

```
curl -i http://localhost:1500/echo/foobar
```

Deploying Models

After creating a model, you need to deploy the model onto HDFS and submit a request for one or more instances of the model.

Procedure

1. Create a simple sample python model.

Let's say that you have a model, exposed as a REST microservice called "mock_dga" that takes as an input argument "host" which represents an internet domain name and returns a field called "is_malicious" which is either "malicious" if the domain is thought to be malicious or "legit" if the domain is not thought to be malicious. The following is a very simple example service that thinks that the only legitimate domains are "yahoo.com" and "amazon.com":

```
from flask import Flask
from flask import request,jsonify
import socket
app = Flask(__name__)

@app.route("/apply", methods=['GET'])
def predict():
        h = request.args.get('host')
        r = {}
        if h == 'yahoo.com' or h == 'amazon.com':
                r['is_malicious'] = 'legit'
        else:
                r['is_malicious'] = 'malicious'
        return jsonify(r)

if __name__ == "__main__":
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind(('localhost', 0))
        port = sock.getsockname()[1]
        sock.close()
        with open("endpoint.dat", "w") as text_file:
                text_file.write("{\"url\" : \"http://0.0.0.0:%d\"}" %
 port)
        app.run(threaded=True, host="0.0.0.0", port=port)
```

2. Store this python model in a directory called /root/mock_dga as dga.py and an accompanying shell script called rest.sh which starts the model:

```
#!/bin/bash
python dga.py
```

3. If you have not already done so, start MaaS:

```
$METRON_HOME/bin/maas_service.sh -zq node1:2181
```

4. Start one or more instances of the model, calling it "dga" and assigning an amount of memory to each instance:

   Because you have placed the model in the /root/mock_dga directory, enter the following:

```
$METRON_HOME/bin/maas_deploy.sh -zq node1:2181 -lmp /root/mock_dga -hmp /
user/root/models -mo ADD -m 512 -n dga -v 1.0 -ni 1
```

   where

-h, --h	A list of functions for maas_deploy.sh
-hmp, --hdfs_model_path <arg>	Model path (HDFS)
-lmp, --local_model_path <arg>	Model path (local)
-m, --memory <arg>	Memory for container
-mo, --mode <arg>	ADD, LIST, or REMOVE
-n, --name <arg>	Model name
-ni, --num_instances <arg>	Number of model instances
-v, --version <arg>	Model version
-zq, --zk_quorum <arg>	ZooKeeper quorum
-zr, --zk_root <arg>	ZooKeeper root

Adding the MaaS Stellar Function to the Sensor Configuration

After deploying a model, you need to add the Stellar function for MaaS to the configuration file for the sensor on which you want to run the model.

Procedure

1. Edit the sensor configuration at $METRON_HOME/config/zookeeper/parsers/$PARSER.json to include a new FieldTransformation to indicate a threat alert based on the model.

```
{
  "parserClassName": "org.apache.metron.parsers.GrokParser",
  "sensorTopic": "squid",
  "parserConfig": {
    "grokPath": "/patterns/squid",
    "patternLabel": "SQUID_DELIMITED",
    "timestampField": "timestamp"
  },
  "fieldTransformations" : [
    {
      "transformation" : "STELLAR"
    ,"output" : [ "full_hostname", "domain_without_subdomains",
 "is_malicious", "is_alert" ]
    ,"config" : {
      "full_hostname" : "URL_TO_HOST(url)"
      ,"domain_without_subdomains" :
 "DOMAIN_REMOVE_SUBDOMAINS(full_hostname)"
      ,"is_malicious" : "MAP_GET('is_malicious',
 MAAS_MODEL_APPLY(MAAS_GET_ENDPOINT('dga'), {'host' :
 domain_without_subdomains}))"
      ,"is_alert" : "if is_malicious == 'malicious' then 'true' else null"
                }
    }
                           ]
}
```

where

transformation	Enter 'STELLAR' to indicate this is a Stellar field transformation.
output	The information the transformation will output. This typically contains full_host, domain_without_subdomains, is_malicious, and is_alert.
full_hostname	The domain component of the "url" field.
domain_without_subdomains	The domain of the "url" field without subdomains.
is_malicious	The output of the "mock_dga" model as deployed earlier. In this case, it will be "malicious" or "legit", because those are the values that our model returns.
is_alert	Set to "true" if and only if the model indicates the hostname is malicious.

2. Edit the sensor enrichment configuration at $METRON_HOME/config/zookeeper/parsers/PARSER.json to adjust the threat triage level of risk based on the model output:

```
{
  "index": "$PARSER_NAME",
  "batchSize": 1,
  "enrichment" : {
    "fieldMap": {}
  },
  "threatIntel" : {
    "fieldMap":{},
    "triageConfig" : {
      "riskLevelRules" : {
        "is_malicious == 'malicious'" : 100
      },
      "aggregator" : "MAX"
    }
  }
}
```

3. Upload the new configurations to $METRON_HOME/bin/zk_load_configs.sh --mode PUSH -i $METRON_HOME/config/zookeeper -z node1:2181.
4. If this is a new sensor and it does not have a Kafka topic associated with it, then we must create a new sensor topic in Kafka.

```
/usr/hdp/current/kafka-broker/bin/kafka-topics.sh --zookeeper node1:2181
 --create --topic $PARSER_NAME --partitions 1 --replication-factor 1
```

Starting Topologies and Sending Data

The final step in setting up Model as a Service, is to start the topologies and send some data to test the model.

Procedure

1. Start the sensor upon which the Model as a Service will run:

```
$METRON_HOME/bin/start_parser_topology.sh -k node1:6667 -z node1:2181 -s
 $PARSER_NAME
```

2. Generate some legitimate data and some malicious data on the sensor.

   For example:

```
#Legitimate example:
squidclient http://yahoo.com
#Malicious example:
squidclient http://cnn.com
```

3. Send the data to Kafka:

```
cat /var/log/squid/access.log | /usr/hdp/current/kafka-broker/bin/kafka-
console-producer.sh --broker-list node1:6667 --topic squid
```

4. Browse the data in Elasticsearch at http://node1:9100/_plugin/head to verify that it contains the appropriate documents.

   For the current example, you would see the following:

   - One from yahoo.com which does not have is_alert set and does have is_malicious set to legit.
   - One from cnn.com which does have is_alert set to true, is_malicious set to malicious, and threat:triage:level set to 100.

Modifying a Model

You can remove a number of instances of the model by executing maax_deploy.sh with remove as the -mo argument.

Procedure

1. For example, the following removes one instance of the dga model, version 1.0:

    ```
    $METRON_HOME/bin/maas_deploy.sh -zq node1:2181 -mo REMOVE -m 512 -n dga -v
     1.0 -ni 1
    ```

2. If you need to modify a model, you need to modify the model itself and deploy a new version, then remove the old version instances afterward.

Analyzing Enriched Data Using Apache Zeppelin

Apache Zeppelin is a web-based notebook that supports interactive data exploration, visualization, sharing and collaboration. HCP users will use Zeppelin at two levels.

- Senior analysts and data scientists can use Zeppelin to produce workbooks to analyze data and to create recreatable investigations or runbooks for junior analysts.
- Junior analysts can use recreatable investigations or runbooks in Zeppelin to discover cybersecurity issues much like they do with the Metron Dashboard. However, Zeppelin can perform more complex calculations and handle larger groups of data.

Setting up Zeppelin to Run with HCP

You can import the Zeppelin Notebook using Ambari or manually. To complete your set up you'll need to use Zeppelin interpretors and load the telemetry information.

Setting up Zeppelin is very simple. To access Zeppelin, go to http://$ZEPPELIN_HOST:9995.

In addition to this documentation, there are three other sources for Zeppelin information.

- The Zeppelin installation for HCP provides a couple sample notes including tutorials specific to Metron. These notes are listed on the left side of the **Welcome** screen and in the **Notebook** menu.
- Zeppelin documentation provides information on launching and using Zeppelin, and you can refer to the following links for this information:
    - Launching Zeppelin
    - Working with Notes
- Apache Zeppelin documentation provides information on Zeppelin basic features, supported interpreters, and more.

Related Information

Importing Zeppelin Notebook Using Ambari

Importing Zeppelin Notebook Manually

Using Zeppelin Interpreters

Loading Telemetry Information into Zeppelin

Apache Zeppelin 0.7.0

Using Zeppelin Interpreters

When you install Zeppelin on HCP the installation includes the interpreter for Spark. Spark is the main backend processing engine for Zeppelin. Spark is also a front end for Python, Scale, and SQL and you can use any of these languages to analyze the telemetry data.

Loading Telemetry Information into Zeppelin

Before you can analyze telemetry information in Zeppelin, you must first download it from Metron. Metron archives the fully parsed, enriched, and triaged telemetry for each sensor in HDFS. This archived telemetry information is simply raw JSON files which makes it simple to parse and analyze the information with Zeppelin.

The following is an example of some Bro telemetry information.

```
%sh

hdfs dfs -ls -C -R /apps/metron/indexing/indexed/bro
/apps/metron/indexing/indexed/bro/enrichment-null-0-0-1484124296101.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-0-1484128332104.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-0-1484131460758.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-1-1484217861096.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-10-1484995461039.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-11-1485081861043.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-12-1485168261040.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-13-1485254661040.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-14-1485341061047.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-15-1485427461040.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-16-1485513861039.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-17-1485600261045.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-18-1485686661035.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-19-1485773061037.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-2-1484304261042.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-20-1485859461037.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-21-1485945861039.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-22-1486032261036.json
```

You can use Spark to load the archived information from HDFS into Zeppelin.

For example if you are loading information received from Bro, your command would look like the following:

```
%spark
sqlContext.read.json("hdfs:///apps/metron/indexing/indexed/
bro").cache().registerTempTable("bro")
```

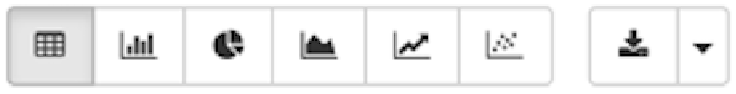
Working with Zeppelin

The Zeppelin user interface consists of notes that are divided into paragraphs. Each paragraph consists of two sections: the code section where you put your source code and the result section where you can see the result of the code execution.

To use the Spark interpreter, you must specify the interpreter directive at the beginning of each paragraph, using the format % [INTERPRETER_NAME}. When you use the Spark interpreter, you can enter source code in Python, Scala, or SQL. So, the interpreter directive could be: %spark.sql.

When you run the code, Zeppelin sends the code to a backend processor such as Spark. The processor or service then returns results; you can then use Zeppelin to review and visualize results in the browser using the Settings toolbar:

Zeppelin Settings Toolbar



Related Information

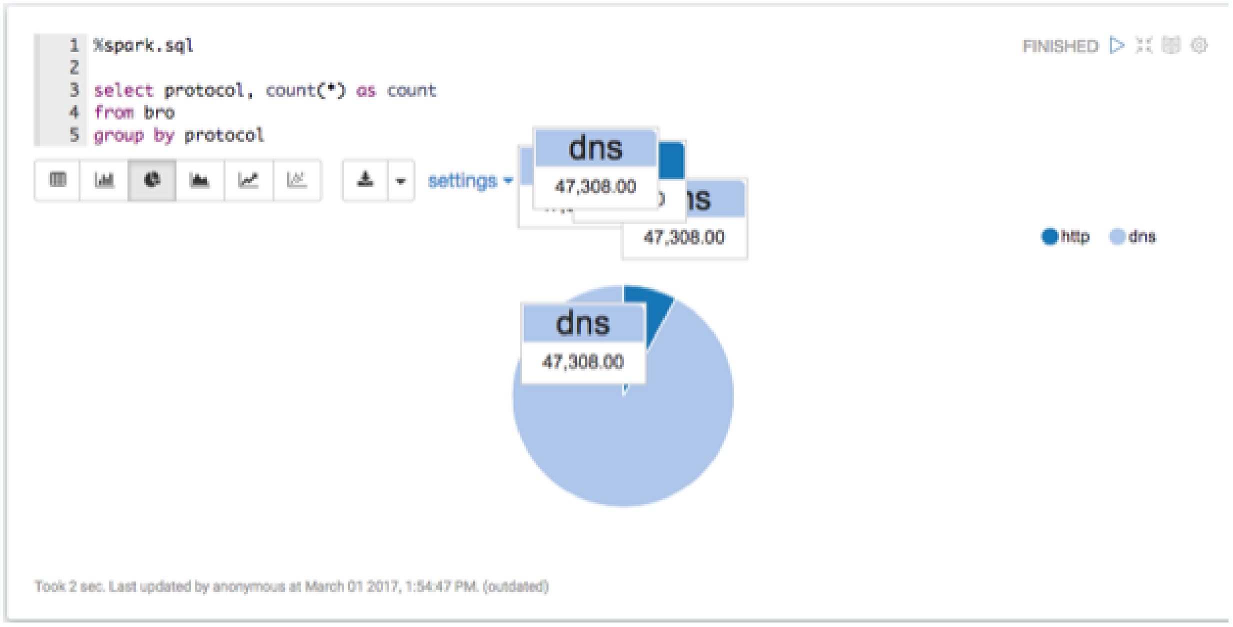
Working with Notes

Using Zeppelin to Analyze Data

Zeppelin enables you to analyze the enriched telemetry information Metron archives in HDFS.

Procedure

1. Bro produces data about a number of different network protocols. View which types of protocols exist in the data.



```
%spark.sql

select protocol, count(*) as count
from bro
group by protocol
```

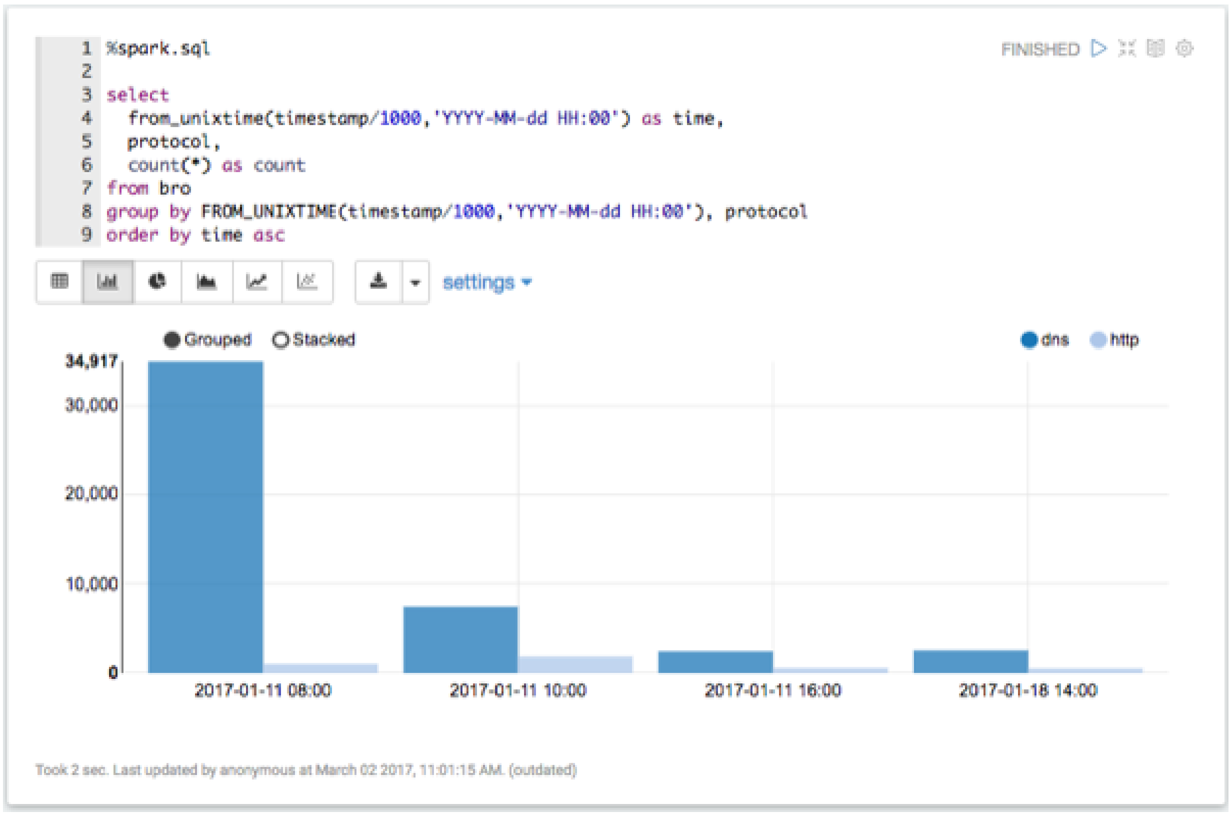
Took 2 sec. Last updated by anonymous at March 01 2017, 1:54:47 PM. (outdated)

2. View when Bro telemetry information was received.

   You can check for any odd gaps and fluctuating periods of high and low activity.

   > **Note:** If there is not enough data for this visualization to be interesting, let Metron consume more data before continuing.

   Bro Telemetry Received

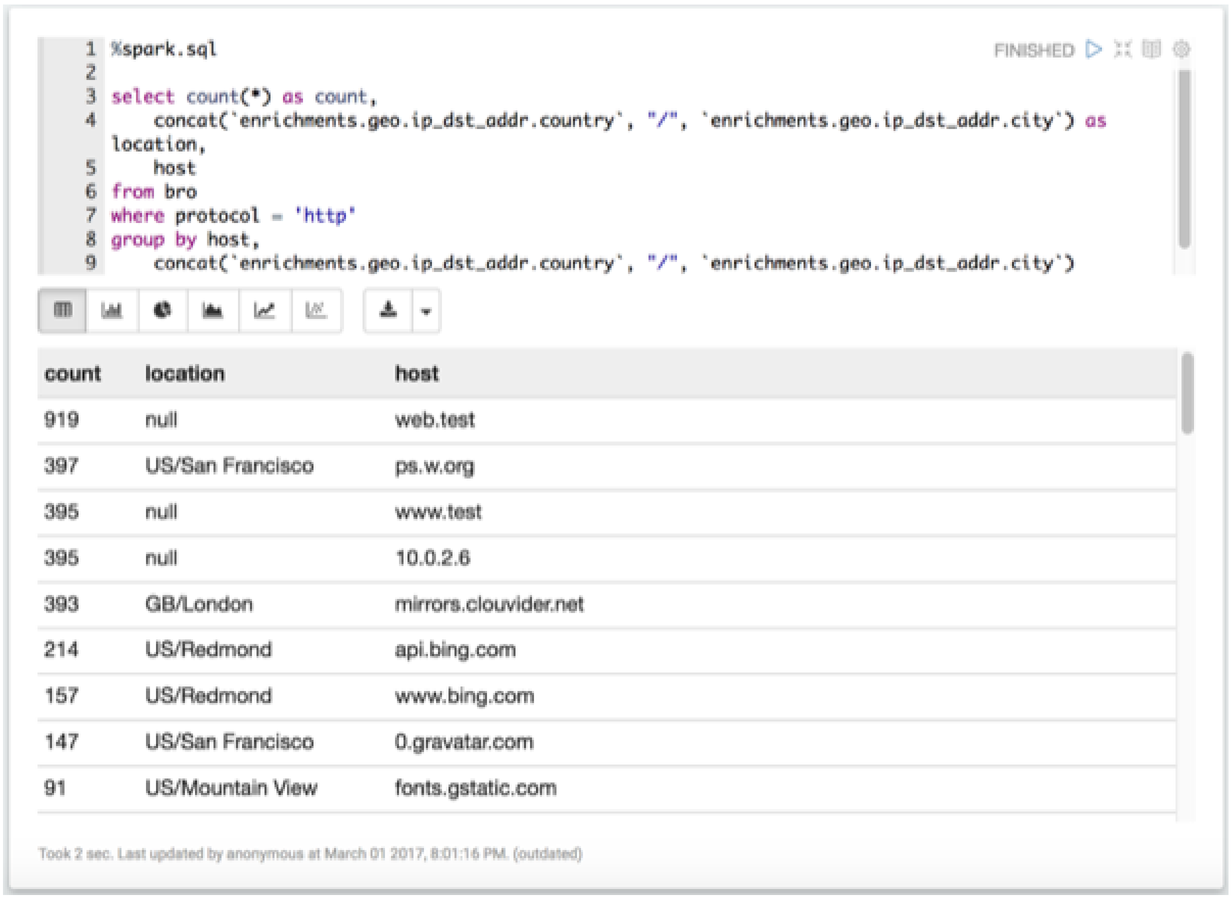


```
%spark.sql

select
  from_unixtime(timestamp/1000,'YYYY-MM-dd HH:00') as time,
  protocol,
  count(*) as count
from bro
group by FROM_UNIXTIME(timestamp/1000,'YYYY-MM-dd HH:00'), protocol
order by time asc
```

Took 2 sec. Last updated by anonymous at March 02 2017, 11:01:15 AM. (outdated)

3. List the most active Bro hosts.

Most Active Hosts



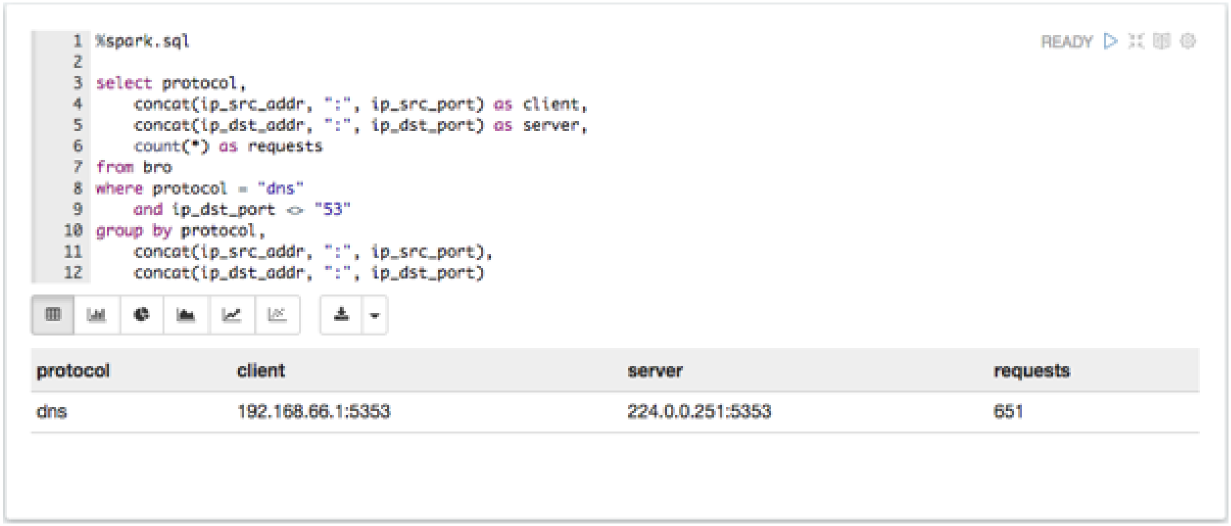
```
%spark.sql

select count(*) as count,
    concat(`enrichments.geo.ip_dst_addr.country`, "/", `enrichments.geo.ip_dst_addr.city`) as
location,
    host
from bro
where protocol = 'http'
group by host,
    concat(`enrichments.geo.ip_dst_addr.country`, "/", `enrichments.geo.ip_dst_addr.city`)
```

count	location	host
919	null	web.test
397	US/San Francisco	ps.w.org
395	null	www.test
395	null	10.0.2.6
393	GB/London	mirrors.clouvider.net
214	US/Redmond	api.bing.com
157	US/Redmond	www.bing.com
147	US/San Francisco	0.gravatar.com
91	US/Mountain View	fonts.gstatic.com

Took 2 sec. Last updated by anonymous at March 01 2017, 8:01:16 PM. (outdated)

4. List any DNS servers running on non-standard ports.

DNS Servers

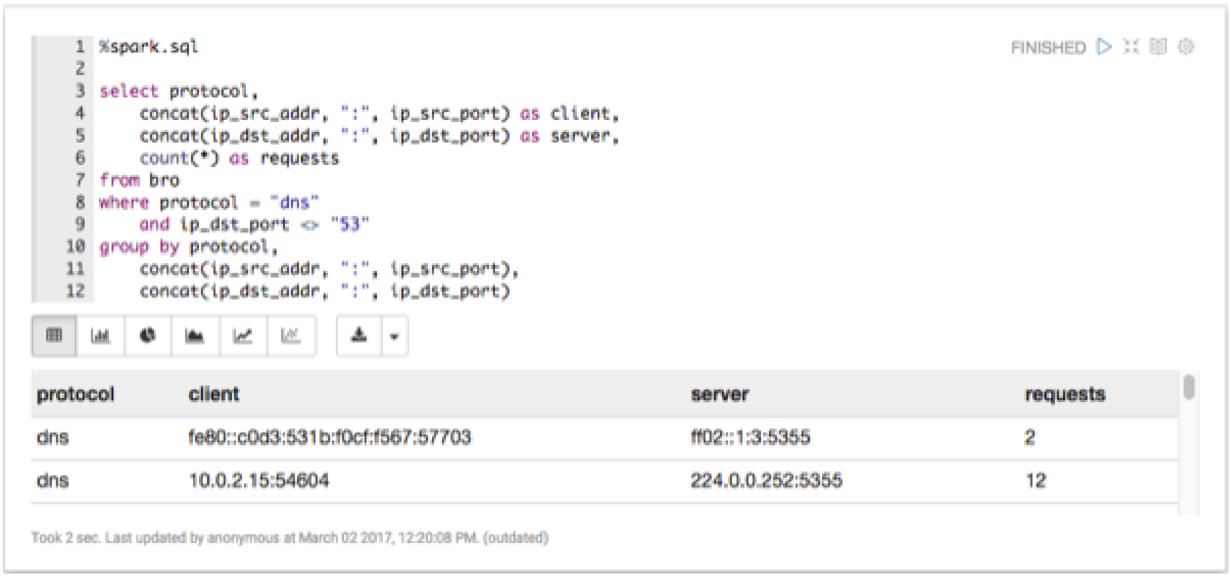


```
%spark.sql

select protocol,
    concat(ip_src_addr, ":", ip_src_port) as client,
    concat(ip_dst_addr, ":", ip_dst_port) as server,
    count(*) as requests
from bro
where protocol = "dns"
    and ip_dst_port <> "53"
group by protocol,
    concat(ip_src_addr, ":", ip_src_port),
    concat(ip_dst_addr, ":", ip_dst_port)
```

protocol	client	server	requests
dns	192.168.66.1:5353	224.0.0.251:5353	651

5. List any mime types that could be concerning.

Mime Types



```
%spark.sql

select protocol,
    concat(ip_src_addr, ":", ip_src_port) as client,
    concat(ip_dst_addr, ":", ip_dst_port) as server,
    count(*) as requests
from bro
where protocol = "dns"
    and ip_dst_port <> "53"
group by protocol,
    concat(ip_src_addr, ":", ip_src_port),
    concat(ip_dst_addr, ":", ip_dst_port)
```

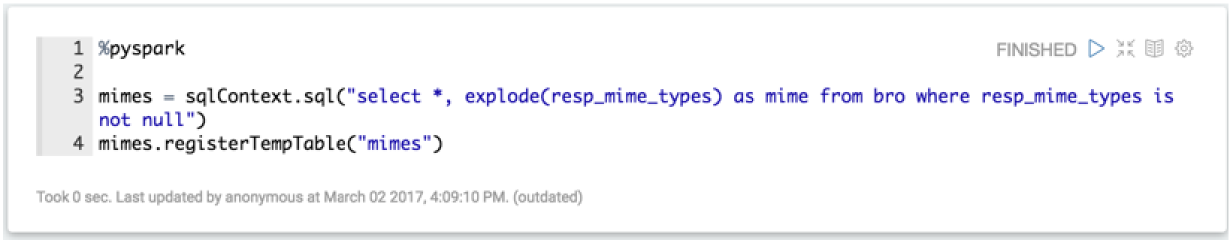
protocol	client	server	requests
dns	fe80::c0d3:531b:f0cf:f567:57703	ff02::1:3:5355	2
dns	10.0.2.15:54604	224.0.0.252:5355	12

Took 2 sec. Last updated by anonymous at March 02 2017, 12:20:08 PM. (outdated)

6. Explode the HTTP records.

Each HTTP record can contain multiple mime types. These need to be 'exploded' to work with them properly.

Exploded HTTP Records

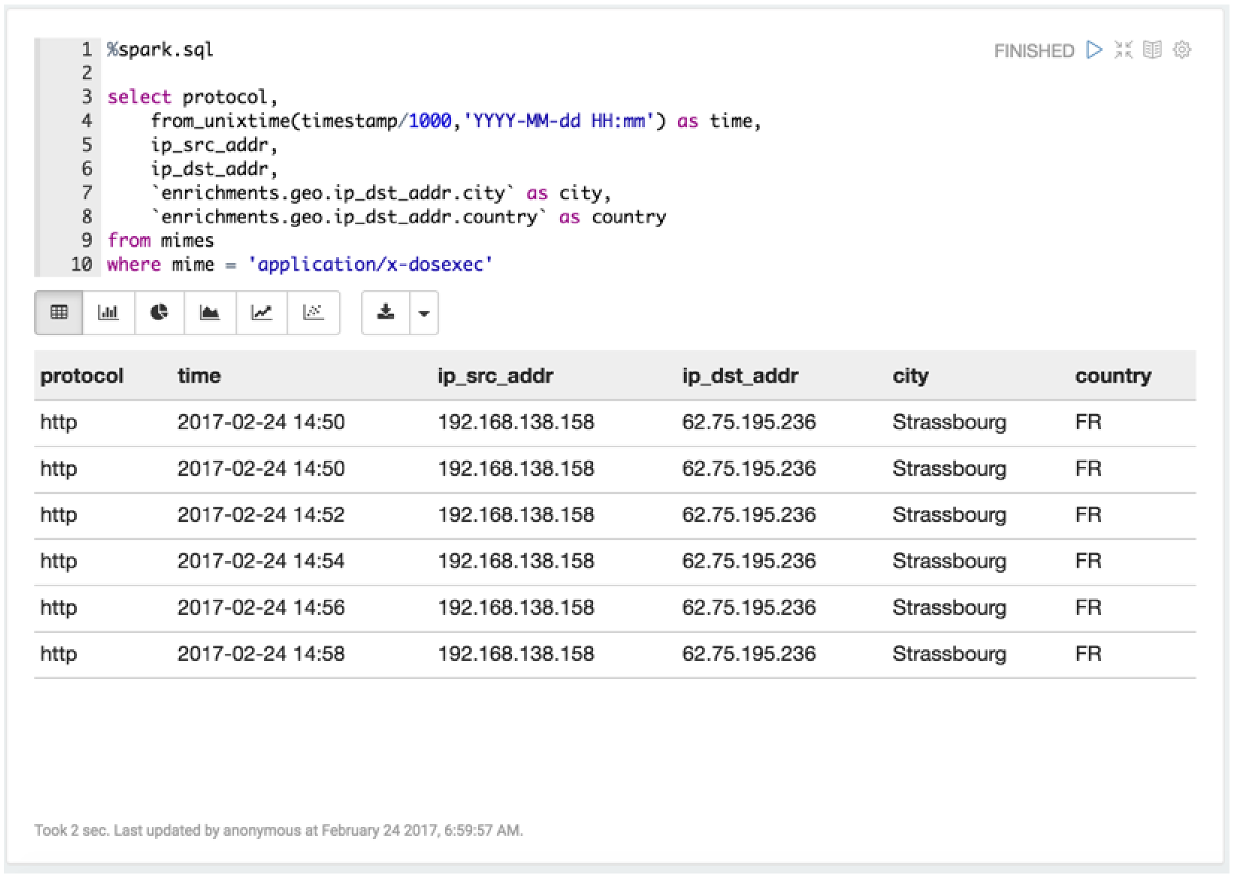


```
%pyspark

mimes = sqlContext.sql("select *, explode(resp_mime_types) as mime from bro where resp_mime_types is
not null")
mimes.registerTempTable("mimes")
```

Took 0 sec. Last updated by anonymous at March 02 2017, 4:09:10 PM. (outdated)

7. Determine where application/x-dosexec originated.

Suspicious xdosexc



FINISHED

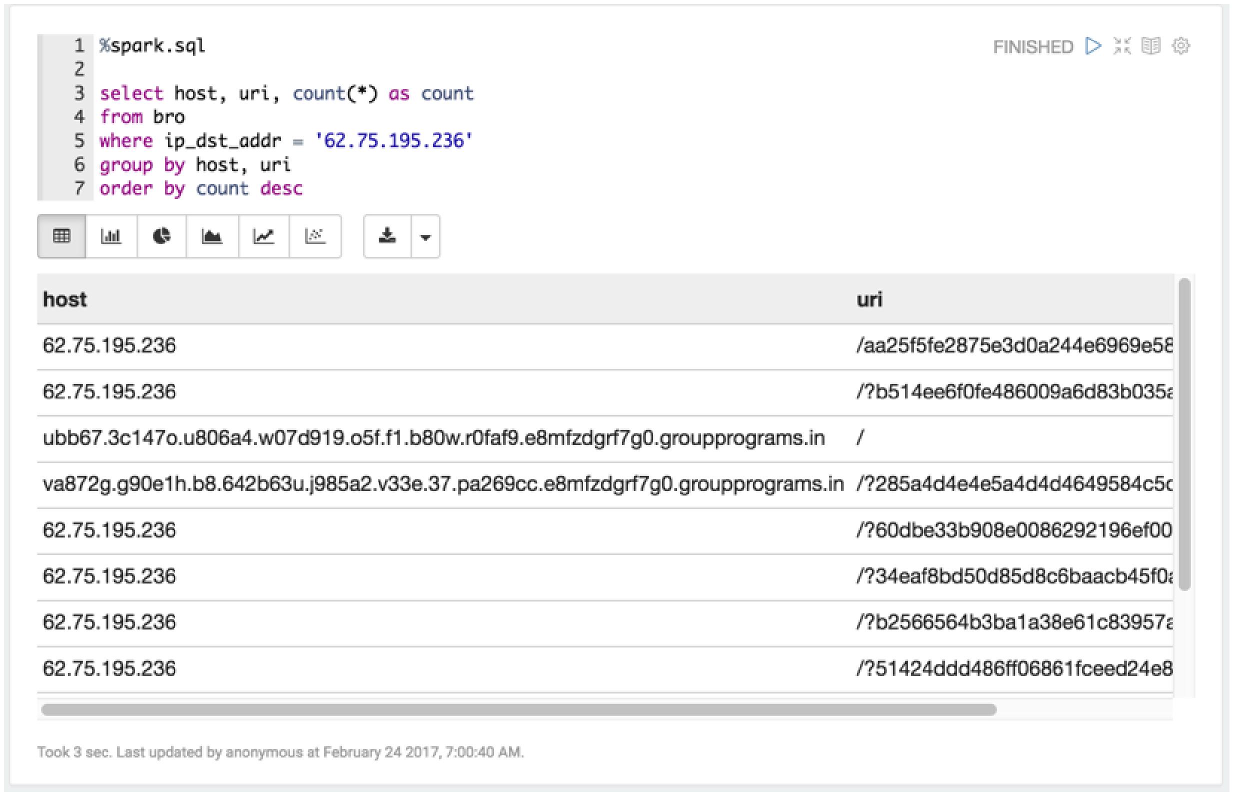
```
%spark.sql

select protocol,
    from_unixtime(timestamp/1000,'YYYY-MM-dd HH:mm') as time,
    ip_src_addr,
    ip_dst_addr,
    `enrichments.geo.ip_dst_addr.city` as city,
    `enrichments.geo.ip_dst_addr.country` as country
from mimes
where mime = 'application/x-dosexec'
```

protocol	time	ip_src_addr	ip_dst_addr	city	country
http	2017-02-24 14:50	192.168.138.158	62.75.195.236	Strassbourg	FR
http	2017-02-24 14:50	192.168.138.158	62.75.195.236	Strassbourg	FR
http	2017-02-24 14:52	192.168.138.158	62.75.195.236	Strassbourg	FR
http	2017-02-24 14:54	192.168.138.158	62.75.195.236	Strassbourg	FR
http	2017-02-24 14:56	192.168.138.158	62.75.195.236	Strassbourg	FR
http	2017-02-24 14:58	192.168.138.158	62.75.195.236	Strassbourg	FR

Took 2 sec. Last updated by anonymous at February 24 2017, 6:59:57 AM.

8. Take a look at the requests for x-dosexc.

x-dosexc Requests



FINISHED

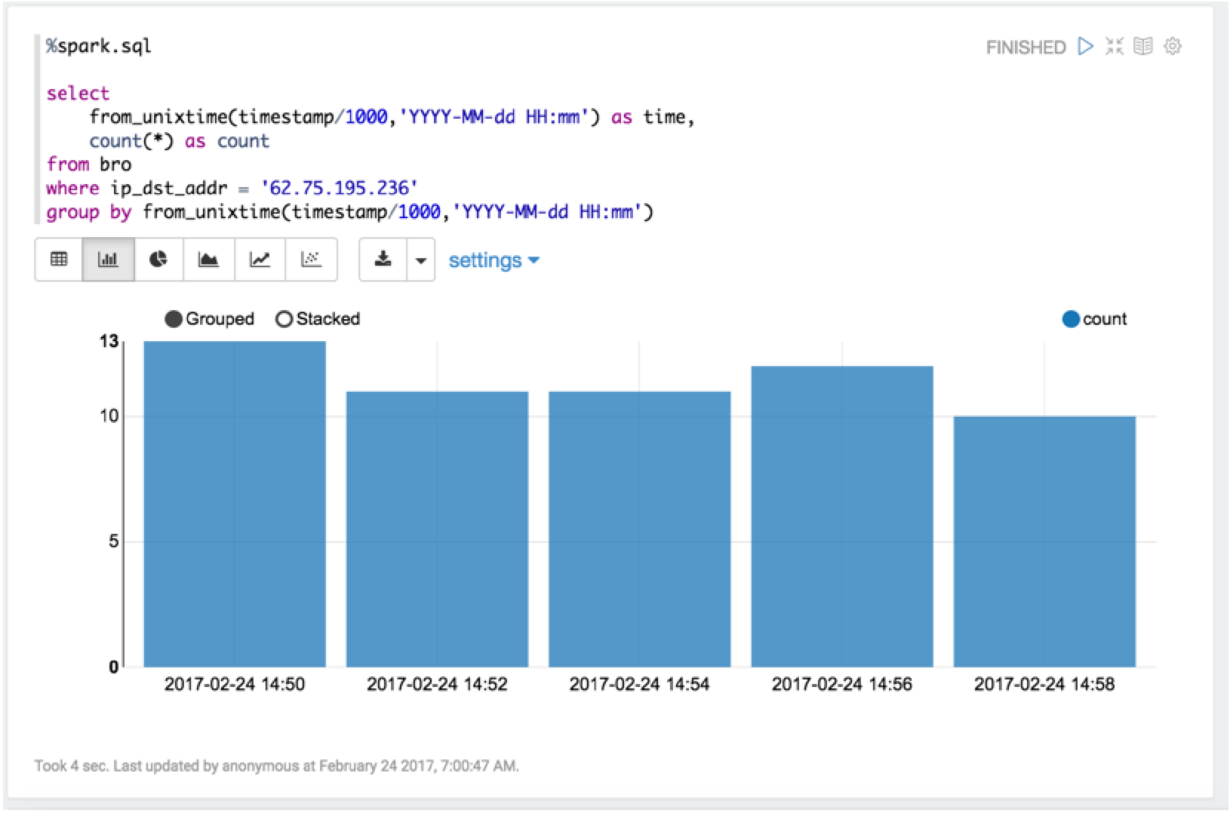
```
%spark.sql

select host, uri, count(*) as count
from bro
where ip_dst_addr = '62.75.195.236'
group by host, uri
order by count desc
```

host	uri
62.75.195.236	/aa25f5fe2875e3d0a244e6969e58
62.75.195.236	/?b514ee6f0fe486009a6d83b035a
ubb67.3c147o.u806a4.w07d919.o5f.f1.b80w.r0faf9.e8mfzdgrf7g0.groupprograms.in	/
va872g.g90e1h.b8.642b63u.j985a2.v33e.37.pa269cc.e8mfzdgrf7g0.groupprograms.in	/?285a4d4e4e5a4d4d4649584c5c
62.75.195.236	/?60dbe33b908e0086292196ef00
62.75.195.236	/?34eaf8bd50d85d8c6baacb45f0a
62.75.195.236	/?b2566564b3ba1a38e61c83957a
62.75.195.236	/?51424ddd486ff06861fceed24e8

Took 3 sec. Last updated by anonymous at February 24 2017, 7:00:40 AM.

9. Determine when the interactions with the suspicious host are occurring.

When Interactions Occur



```
%spark.sql

select
    from_unixtime(timestamp/1000,'YYYY-MM-dd HH:mm') as time,
    count(*) as count
from bro
where ip_dst_addr = '62.75.195.236'
group by from_unixtime(timestamp/1000,'YYYY-MM-dd HH:mm')
```

Took 4 sec. Last updated by anonymous at February 24 2017, 7:00:47 AM.

10. Create an IP report in Zeppelin using the Metron IP Report notebook.

For a given IP address, this notebook produces a report of:

- Most frequent connections (YAF defaults to 24 hours)
- Recent connections (Yaf, defaults to 1 hours)
- Top DNS queries (Bro, defaults to 24 hours)
- All ports used (Yaf, defaults to 24 hours)
- HTTP user agents (Bro, defaults to 24 hours)

11. Create traffic connection request report using the Connection Volume Report notebook.

This notebook lets the user get connection counts filtered by a CIDR block. This notebook can be used for Bro, Yaf, and Snort.

What to do next

Through this brief analysis we uncovered something that looks suspicious. So far we have leveraged only the geo-enriched Bro telemetry. From here, we can start to explore other sources of telemetry to better understand the scope and overall exposure. Continue to investigate our suspicions with the other sources of telemetry available in Metron.

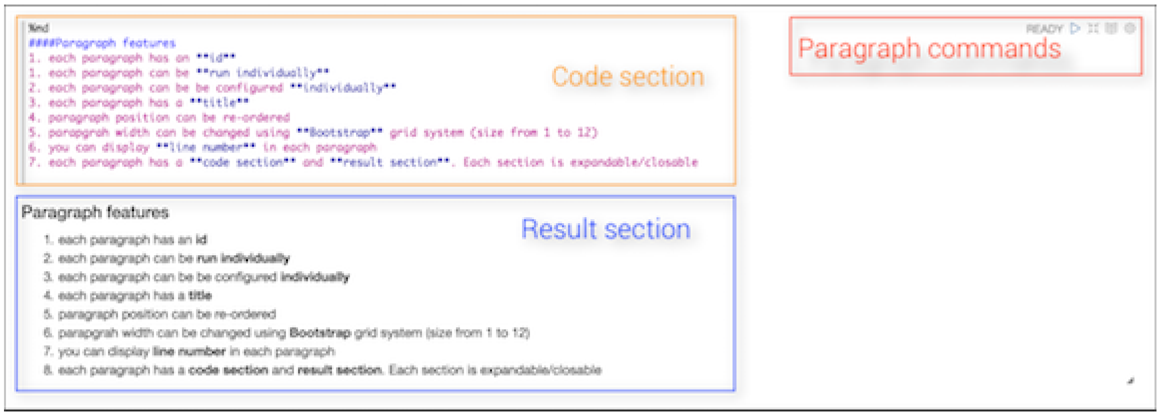
- Try loading the Snort data and see if any alerts were triggered.
- Load the flow telemetry and see what other internal assets have been exposed to this suspicious actor.
- If an internal asset has been compromised, investigate the compromised asset's activity to uncover signs of internal reconnaissance or lateral movement.

Working with Zeppelin Notes

This section provides an introduction to Apache Zeppelin notes.

An Apache Zeppelin note consists of one or more paragraphs of code, which you can use to define and run snippets of code in a flexible manner.

A paragraph contains code to access services, run jobs, and display results. A paragraph consists of two main sections: an interactive box for code, and a box that displays results. To the right is a set of paragraph commands. The following graphic shows paragraph layout.

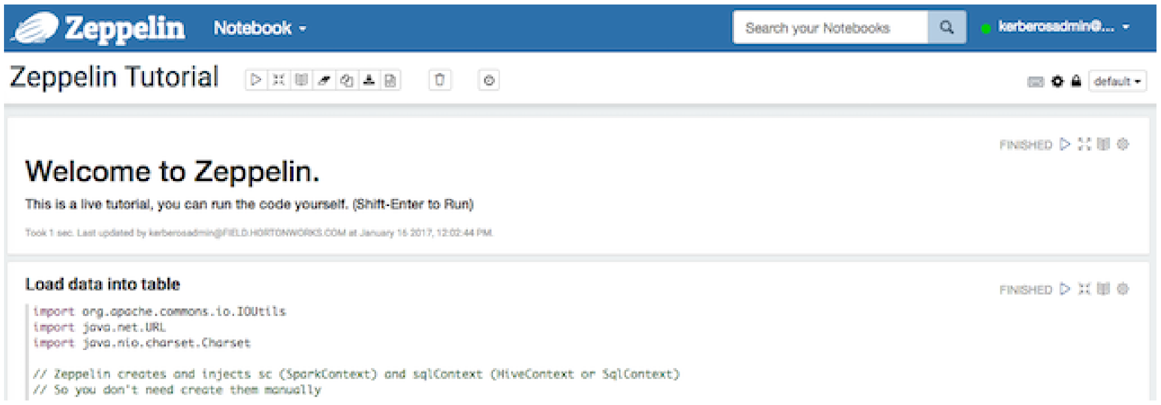


```
%md
####Paragraph features
1. each paragraph has an **id**
1. each paragraph can be **run individually**
2. each paragraph can be be configured **individually**
3. each paragraph has a **title**
4. paragraph position can be re-ordered
5. parapgrah width can be changed using **Bootstrap** grid system (size from 1 to 12)
6. you can display **line number** in each paragraph
7. each paragraph has a **code section** and **result section**. Each section is expandable/closable
```

Paragraph features

1. each paragraph has an **id**
2. each paragraph can be **run individually**
3. each paragraph can be be configured **individually**
4. each paragraph has a **title**
5. paragraph position can be re-ordered
6. parapgrah width can be changed using **Bootstrap** grid system (size from 1 to 12)
7. you can display **line number** in each paragraph
8. each paragraph has a **code section** and **result section**. Each section is expandable/closable

Zeppelin ships with several sample notes, including tutorials that demonstrate how to run Spark scala code, Spark SQL code, and create visualizations.



Zeppelin Tutorial

Welcome to Zeppelin.

This is a live tutorial, you can run the code yourself. (Shift-Enter to Run)

Took 1 sec. Last updated by kerberosadmin@FIELD.HORTONWORKS.COM at January 16 2017, 12:02:44 PM.

Load data into table

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset

// Zeppelin creates and injects sc (SparkContext) and sqlContext (HiveContext or SqlContext)
// So you don't need create them manually
```

To run a tutorial:

1. Navigate to the tutorial: click one of the Zeppelin tutorial links on the left side of the welcome page, or use the Notebook pull-down menu.
2. Zeppelin presents the tutorial, a sequence of paragraphs prepopulated with code and text.
3. Starting with the first paragraph, click the triangle button at the upper right of the paragraph. The status changes to PENDING, RUNNING, and then FINISHED when done.
4. When the first cell finishes execution, results appear in the box underneath your code. Review the results.
5. Step through each cell, running the code and reviewing results.

Create and Run a Note

Use the following steps to create and run an Apache Zeppelin note.

To create a note:

1. Click "Create new note" on the welcome page, or click the "Notebook" menu and choose "+ Create new note."
2. Type your commands into the blank paragraph in the new note.

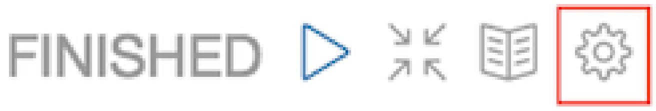
When you create a note, it appears in the list of notes on the left side of the home page and in the Notebook menu. By default, Zeppelin stores notes in the $ZEPPELIN_HOME/notebook folder.

To run your code:

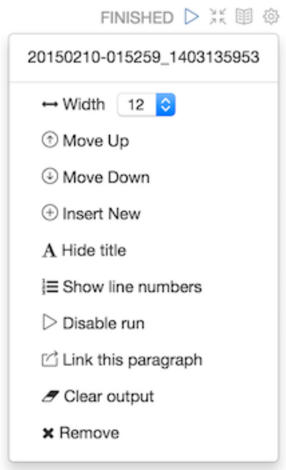
1. Click the triangle button in the cell that contains your code:

2. Zeppelin displays status near the triangle button: PENDING, RUNNING, ERROR, or FINISHED.
3. When finished, results appear in the result section below your code.

The settings icon (outlined in red) offers several additional commands:



These commands allow you to perform several note operations, such as showing and hiding line numbers, clearing the results section, and deleting the paragraph.



Import a Note

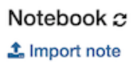
Use the following steps to import an Apache Zeppelin note.

About this task

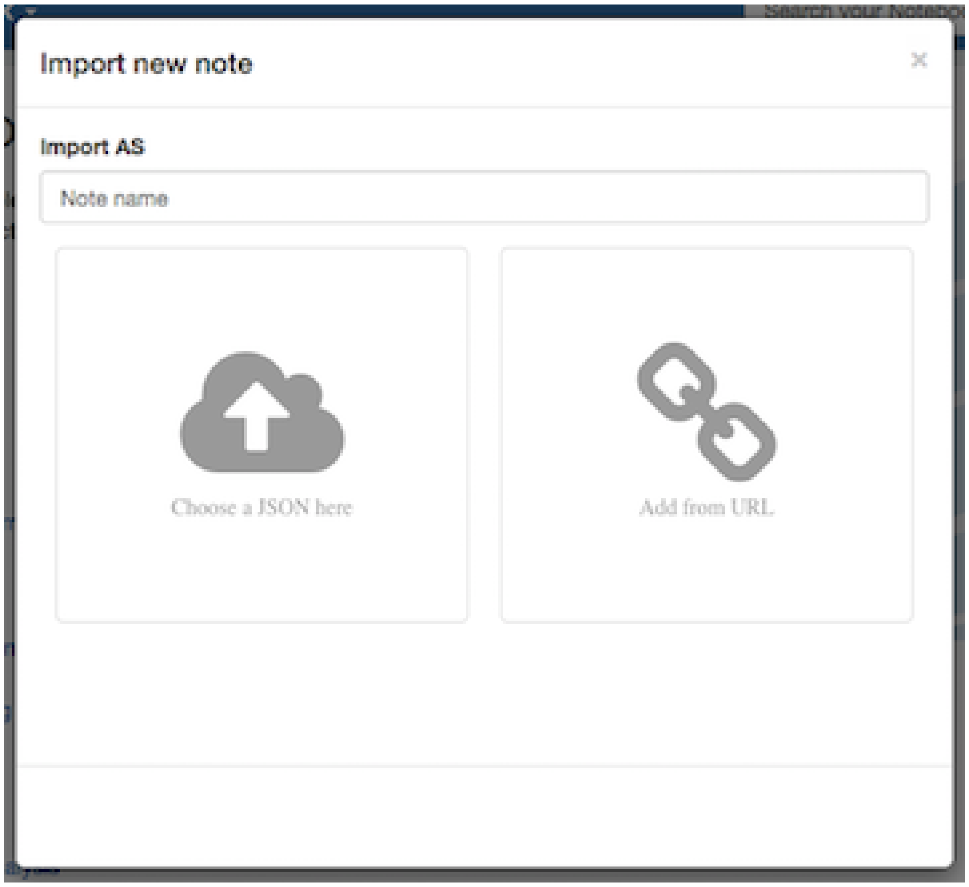
To import a note from a URL or from a JSON file in your local file system:

Procedure

1. Click "Import note" on the Zeppelin home page:



2. Zeppelin displays an import dialog box:



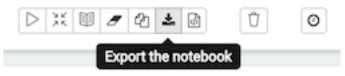
3. To upload the file or specify the URL, click the associated box.

   By default, the name of the imported note is the same as the original note. You can rename it by providing a new name in the "Import AS" field.

Export a Note

Use the following steps to export an Apache Zeppelin note.

To export a note to a local JSON file, use the export note icon in the note toolbar:



Zeppelin downloads the note to the local file system.

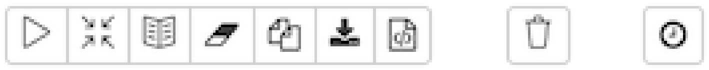
Note: Zeppelin exports code and results sections in all paragraphs. If you have a lot of data in your results sections, consider trimming results before exporting them.

Using the Note Toolbar

This section describes how to use the Apache Zeppelin Note toolbar.

At the top of each note there is a toolbar with buttons for running code in paragraphs and for setting configuration, security, and display options:

There are several buttons in the middle of the toolbar:



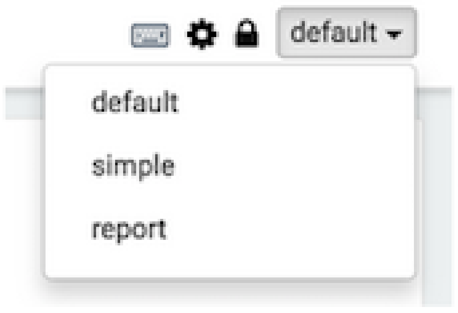
These buttons perform the following operations:

- Execute all paragraphs in the note sequentially, in the order in which they are displayed in the note.

- Hide or show the code section of all paragraphs.
- Hide or show the result sections in all paragraphs.
- Clear the result section in all paragraphs.
- Clone the current note.
- Export the current note to a JSON file.

  Note that the code and result sections in all paragraphs are exported. If you have extra data in some of your result sections, trim the data before exporting it.
- Commit the current note content.
- Delete the note.
- Schedule the execution of all paragraphs using CRON syntax. This feature is not currently operational. If you need to schedule Spark jobs, consider using Oozie Spark action.

There are additional buttons on the right side of the toolbar:



These buttons perform the following operations (from left to right):

- Display all keyboard shortcuts.
- Configure interpreters that are bound to the current note.
- Configure note permissions.
- Switch display mode:
  - Default: the notebook can be shared with (and edited by) anyone who has access to the notebook.
  - Simple: similar to default, with available options shown only when your cursor is over the cell.
  - Report: only your results are visible, and are read-only (no editing).

  Note: Zeppelin on HDP does not support sharing a note by sharing its URL, due to lack of proper access control over who and how a note can be shared.

Creating Runbooks Using Apache Zeppelin

Apache Zeppelin is a web-based notebook that supports interactive data exploration, visualization, sharing and collaboration.

HCP users will use Zeppelin at two levels.

- Senior analysts and data scientists can use Zeppelin to produce notebooks to analyze data and to create recreatable investigations or runbooks for junior analysts.
- Junior analysts can use recreatable investigations or runbooks in Zeppelin to discover cybersecurity issues much like they do with the Metron Dashboard. However, Zeppelin can handle larger groups of data.

Setting up Zeppelin to Run with HCP

After installing Zeppelin, you need to set up Zeppelin to run with HCP.

Procedure

To access Zeppelin, go to http://$ZEPPELIN_HOST:9995.

What to do next

To complete your set up, see the following sections:

- Using Zeppelin Interpreters
- Loading Telemetry Information into Zeppelin

Using Zeppelin Interpreters

When you install Zeppelin on HCP the installation includes the interpreter for Spark. Spark is the main backend processing engine for Zeppelin. Spark is also a front end for Python, Scale, and SQL and you can use any of these languages to analyze the telemetry data.

Loading Telemetry Information into Zeppelin

Before you can analyze telemetry information in Zeppelin, you must first download it from Metron. Metron archives the fully parsed, enriched, and triaged telemetry for each sensor in HDFS. This archived telemetry information is simply raw JSON files which makes it simple to parse and analyze the information with Zeppelin.

The following is an example of some Bro telemetry information.

```
%sh

hdfs dfs -ls -C -R /apps/metron/indexing/indexed/bro
/apps/metron/indexing/indexed/bro/enrichment-null-0-0-1484124296101.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-0-1484128332104.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-0-1484131460758.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-1-1484217861096.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-10-1484995461039.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-11-1485081861043.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-12-1485168261040.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-13-1485254661040.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-14-1485341061047.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-15-1485427461040.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-16-1485513861039.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-17-1485600261045.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-18-1485686661035.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-19-1485773061037.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-2-1484304261042.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-20-1485859461037.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-21-1485945861039.json
/apps/metron/indexing/indexed/bro/enrichment-null-0-22-1486032261036.json
```

You can use Spark to load the archived information from HDFS into Zeppelin.

For example if you are loading information received from Bro, your command would like the following:

```
%spark
sqlContext.read.json("hdfs:///apps/metron/indexing/indexed/
bro").cache().registerTempTable("bro")
```

Working with Zeppelin

The Zeppelin user interface consists of notes that are divided into paragraphs. Each paragraph consists of two sections: the code section where you put your source code and the result section where you can see the result of the code execution. When you use the Spark interpreter, you can enter source code in Python, Scala, or SQL. When you run the code from the browser, Zeppelin sends the code to a backend processor such as Spark. The processor or service then returns results; you can then use Zeppelin to review and visualize results in the browser.

For more information on using notes, see Working with Notes.

Related Information

Working with Notes

Using Zeppelin to Create Runbooks

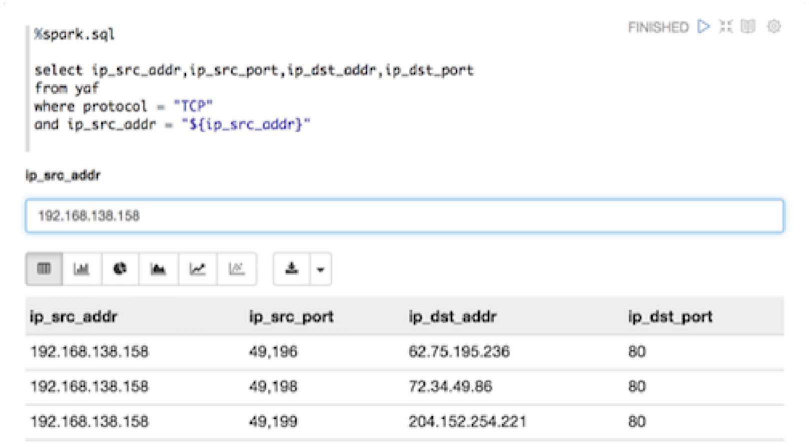
Zeppelin enables data scientists and senior analysts to create workbooks for junior analysts that can be used as runbooks for recreatable investigations. These runbooks can be static, which require no input from the junior analyst, or dynamic, which require the junior analyst to enter or choose information. You can see an example of a static type of notebook in the Metron - YAF Telemetry note. This section provides instructions for creating both kinds of runbooks.

Procedure

1. Click **Create new note** on the welcome page, or click the **Notebook** menu and choose **+ Create new note**.
2. Type your commands into the blank paragraph in the new note.

   To make your runbook dynamic, use one or more of the dynamic forms that Zeppelin supports:

   - Text input form



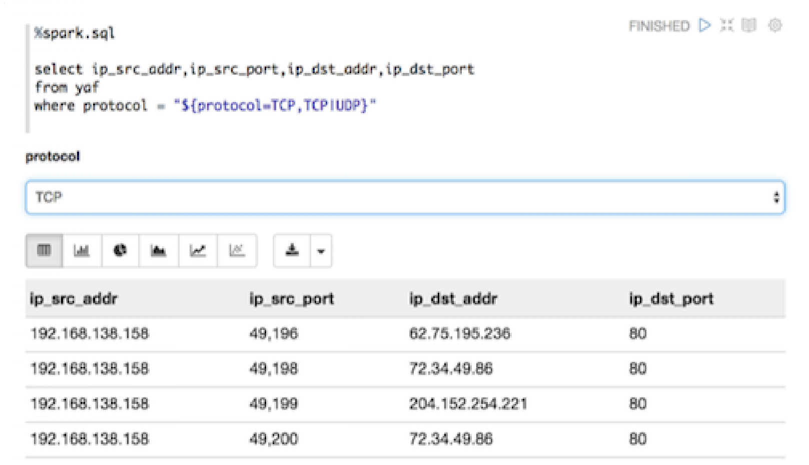
```
%spark.sql

select ip_src_addr,ip_src_port,ip_dst_addr,ip_dst_port
from yaf
where protocol = "TCP"
and ip_src_addr = "${ip_src_addr}"
```

ip_src_addr

192.168.138.158

ip_src_addr	ip_src_port	ip_dst_addr	ip_dst_port
192.168.138.158	49,196	62.75.195.236	80
192.168.138.158	49,198	72.34.49.86	80
192.168.138.158	49,199	204.152.254.221	80

   - Text input form with default value
   - Select form



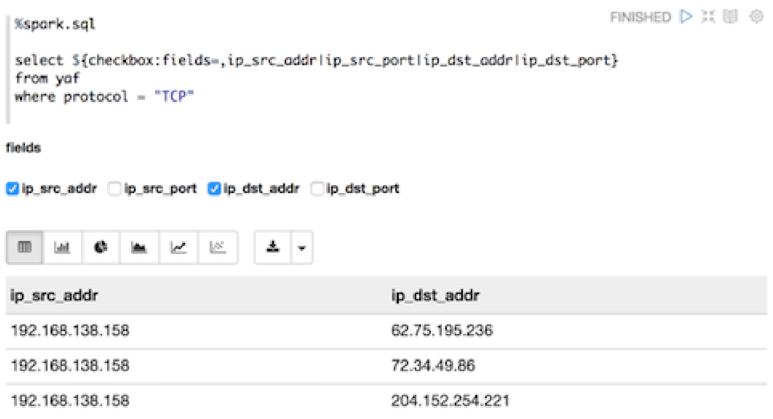
```
%spark.sql

select ip_src_addr,ip_src_port,ip_dst_addr,ip_dst_port
from yaf
where protocol = "${protocol=TCP,TCP|UDP}"
```

protocol

TCP

ip_src_addr	ip_src_port	ip_dst_addr	ip_dst_port
192.168.138.158	49,196	62.75.195.236	80
192.168.138.158	49,198	72.34.49.86	80
192.168.138.158	49,199	204.152.254.221	80
192.168.138.158	49,200	72.34.49.86	80

   - Checkbox form



```
%spark.sql

select ${checkbox:fields=,ip_src_addr|ip_src_port|ip_dst_addr|ip_dst_port}
from yaf
where protocol = "TCP"
```

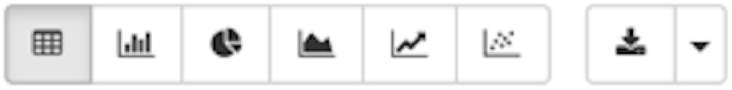
ip_src_addr	ip_dst_addr
192.168.138.158	62.75.195.236
192.168.138.158	72.34.49.86
192.168.138.158	204.152.254.221

When you create a note, it appears in the list of notes on the left side of the home page and in the **Notebook** menu. By default, Zeppelin stores notes in the $ZEPPELIN_HOME/notebook folder.

3. Run your new code by clicking the triangle button in the cell that contains your code.

   Zeppelin attempts to run the code and displays the status near the triangle button: PENDING, RUNNING, ERROR, or FINISHED. Zeppelin also displays another empty paragraph so you can add another command.
4. Continue adding commands until you've completed the runbook.
5. Choose the appropriate type of visualization for your code results from the settings toolbar below the code section of the paragraph.

   Zeppelin Settings Toolbar



6. If appropriate, notify the junior analyst about the runbook that he can clone and use.

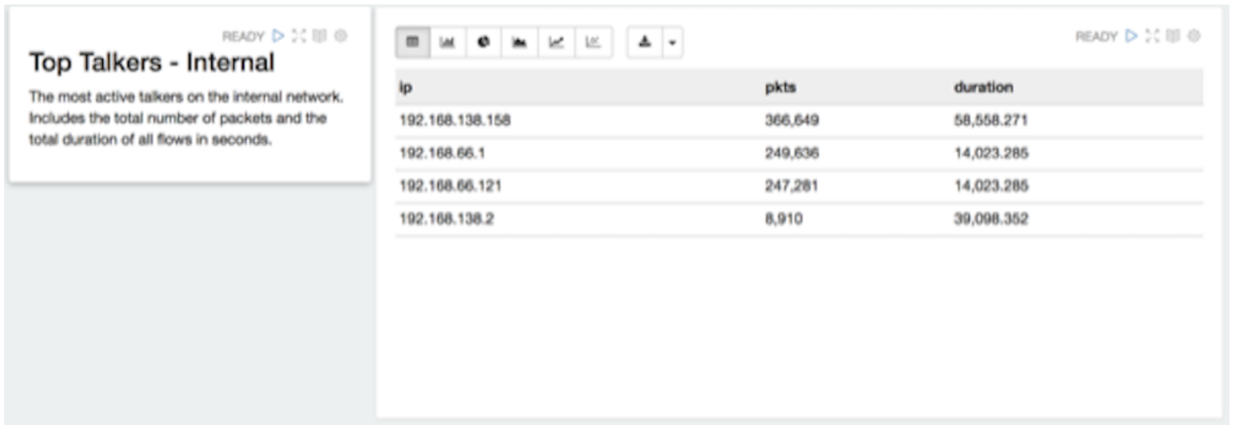
What to do next

The following examples provide sample paragraphs you might want to include in a runbook:

- Top Talkers - Internal

  This paragraph is static and requires no input from the user.

  Zeppelin Top Talkers



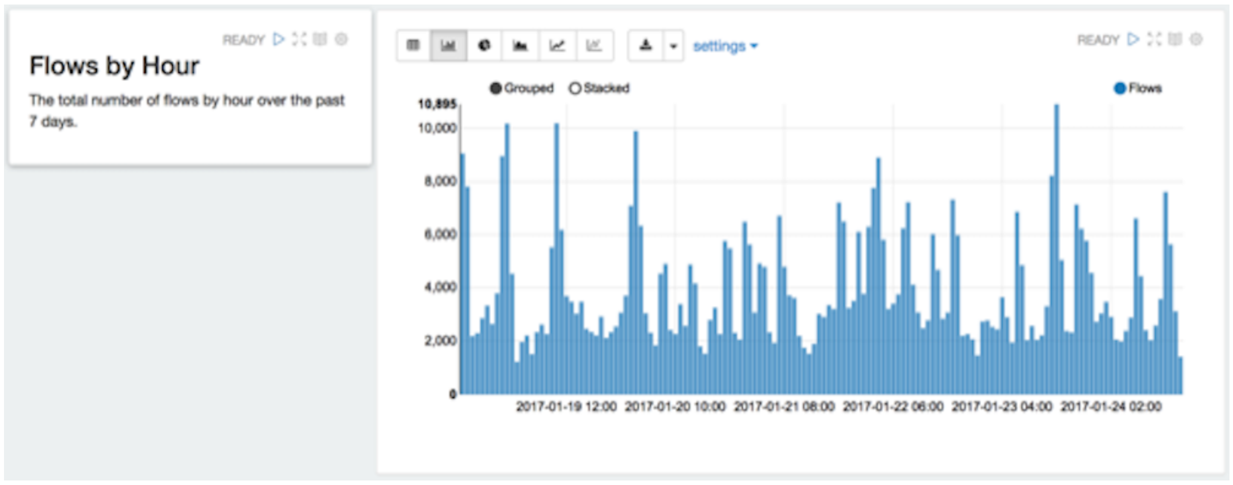
The most active talkers on the internal network. Includes the total number of packets and the total duration of all flows in seconds.

ip	pkts	duration
192.168.138.158	366,649	58,558.271
192.168.66.1	249,636	14,023.285
192.168.66.121	247,281	14,023.285
192.168.138.2	8,910	39,098.352

- Flows by hour

  This paragraph is static and requires no input from the user.

  Zeppelin Flows By Hour



Related Information

Working with Notes

Analyzing Data Using Statistical and Mathematical Functions

HCP provides a variety of advanced analytics that use statistics and advanced mathematical functions. Capturing the statistical snapshots in a scalable way can open up doors for more advanced analytics such as outlier analysis. These analytics provide a robust set of statistical functions and statistical-based algorithms in the form of Stellar functions. These functions can be used from everywhere where Stellar is used.

Approximation Statistics

Data scientists frequently want to find anomalies in numerical data. To that end, HCP has some simple approximation statistics.

Table 3: Approximation Statistics

Function	Description	Input	Returns
HLLP_ADD	Add value to the HyperLogLogPlus estimator set.	• hyperLogLogPlus - The hllp estimator to add a value to • value+ - Value to add to the set. Takes a single item or a list.	The HyperLogLogPlus set with a new value added
HLLP_CARDINALITY	Returns HyperLogLogPlus-estimated cardinality for this set.	• hyperLogLogPlus - The hllp set	Long value representing the cardinality for this set
HLLP_INIT	Initializes the HyperLogLogPlus estimator set. p must be a value between 4 and sp and sp must be less than 32 and greater than 4.	• p - The precision value for the normal set • sp - The precision value for the sparse set. If p is set, but sp is 0 or not specified, the sparse set will be disabled.	A new HyperLogLogPlus set
HLLP_MERGE	Merge hllp sets together. The resulting estimator is initialized with p and sp precision values from the first provided hllp estimator set.	• hllp - List of hllp estimators to merge. Takes a single hllp set or a list.	True if the filter might contain the value and false otherwise

Related Information

HLLP README

Mathematical Functions

Data scientists frequently want to find anomalies in numerical data. To that end, HCP has some mathematical functions.

Table 4: Mathematical Functions

Function	Description	Input	Returns
ABS	Returns the absolute value of a number.	• number - The number to take the absolute value of	The absolute value of the number passed in
BIN	Computes the bin that the value is in given a set of bounds	• value -the value to bin • bounds - A list of value bounds (excluding min and max) in sorted order	Which bin N the value falls in such that bound(N-1) < value <= bound(N). No min and max bounds are provided, so values smaller than the 0'th bound go in the 0'th bin, and values greater than the last bound go in the M'th bin.

Distributional Statistics

Data scientists frequently want to find anomalies in numerical data. To that end, HCP has some simple distributional statistics.

Table 5: Distributional Statistics

Function	Description	Input	Returns
STATS_ADD	Adds one or more input values to those that are used to calculate the summary statistics.	• stats - The Stellar statistics object. If null, then a new one is initialized. • value+ - One or more numbers to add	A Stellar statistics object
STATS_BIN	Computes the bin that the value is in based on the statistical distribution.	• stats - The Stellar statistics object • value - The value to bin • bounds? - A list of percentile bin bounds (excluding min and max) or a string representing a known and common set of bins. For convenience, we have provided QUARTILE, QUINTILE, and DECILE which you can pass in as a string arg. If this argument is omitted, then we assume a Quartile bin split.	Which bin N the value falls in such that bound(N-1) < value <= bound(N). No min and max bounds are provided, so values smaller than the 0'th bound go in the 0'th bin, and values greater than the last bound go in the M'th bin.
STATS_COUNT	Calculates the count of the values accumulated (or in the window if a window is used).	• stats - The Stellar statistics object	The count of the values in the window or NaN if the statistics object is null
STATS_GEOMETRIC_MEAN	Calculates the geometric mean of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The geometric mean of the values in the window or NaN if the statistics object is null.

Function	**Description**	**Input**	**Returns**
STATS_INIT	Initializes a statistics object	• window size - The number of input data values to maintain in a rolling window in memory. If window_size is equal to 0, then no rolling window is maintained. Using no rolling window is less memory intensive, but cannot calculate certain statistics like percentiles and kurtosis.	A Stellar statistics object
STATS_KURTOSIS	Calculates the kurtosis of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The kurtosis of the values in the window or NaN if the statistics object is null
STATS_MAX	Calculates the maximum of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The maximum of the accumulated values in the window or NaN if the statistics object is null.
STATS_MEAN	Calculates the mean of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The mean of the values in the window or NaN if the statistics object is null.
STATS_MERGE	Merges statistics objects	• statistics -A list of statistics objects	A Stellar statistics object
STATS_MIN	Calculates the minimum of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The minimum of the accumulated values in the window or NaN if the statistics object is null.
STATS_PERCENTILE	Computes the p'th percentile of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object • p - A double where 0 <= p < 1 representing the percentile	The p'th percentile of the data or NaN if the statistics object is null
STATS_POPULATION_VARIANCE	Calculates the population variance of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The population variance of the values in the window or NaN if the statistics object is null.
STATS_QUADRATIC_MEAN	Calculates the quadratic mean of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The quadratic mean of the values in the window or NaN if the statistics object is null.
STATS_SD	Calculates the standard deviation of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The standard deviation of the values in the window or NaN if the statistics object is null.
STATS_SKEWNESS	Calculates the skewness of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The skewness of the values in the window or NaN if the statistics object is null.
STATS_SUM	Calculates the sum of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The sum of the values in the window or NaN if the statistics object is null.
STATS_SUM_LOGS	Calculates the sum of the (natural) log of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The sum of the (natural) log of the values in the window or NaN if the statistics object is null.
STATS_SUM_SQUARES	Calculates the sum of the squares of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The sum of the squares of the values in the window or NaN if the statistics object is null.
STATS_VARIANCE	Calculates the variance of the accumulated values (or in the window if a window is used).	• stats - The Stellar statistics object	The variance of the values in the window or NaN if the statistics object is null.

Related Information

Descriptive Statistics

Alternatives to the Median Absolute Deviation

Statistical Outlier Detection

Data scientists frequently want to find anomalies in numerical data. To that end, HCP has some simple statistical outlier detectors.

Table 6: Statistical Outlier Detection

Function	Description	Input	Returns
OUTLIER_MAD_STATE_MERGE	Update the statistical state required to compute the Median Absolute Deviation.	• state - A list of Median Absolute Deviation States to merge. Generally these are states across time. • currentState? - The current state (optional)	The Median Absolute Deviation state
OUTLIER_MAD_ADD	Add a piece of data to the state	• state -The MAD state • value - The numeric value to add	The MAD state
OUTLIER_MAD_SCORE	Get the modified z-score normalized by the MAD: scale * | x_i - median(X) | / MAD.	• state - The MAD state • value - The numeric value to score • scale? -Optionally the scale to use when computing the modified z-score. Default is 0.6745.	The modified z-score

Outlier Analysis

Data scientists frequently want to find anomalies in numerical data. To that end, HCP has some simple statistical anomaly detectors.

Median Absolution Deviation

Much has been written about this robust estimator. See the first page of *Alternatives to the Median Absolute Deviation* for coverage of the good and the bad of median absolution deviation (MAD). The usage, however is fairly straightforward.

- Gather the statistical state required to compute the MAD.
  - The distribution of the values of a univariate random variable over time.
  - The distribution of the absolute deviations of the values from the median.
- Use this statistical state to score unseen values. The higher the score, the more unlike the previously seen data the value is.

There are a couple of issues which make MAD hard to compute. First, the statistical state requires computing median, which can be computationally expensive to compute exactly. To get around this, we use the OnlineStatisticalProvider to compute a sketch rather than the exact median. Secondly, the statistical state for seasonal data should be limited to a fixed, trailing window. We do this by ensuring that the MAD state is mergeable and able to be queried from within the Profiler.

Related Information

Alternatives to the Median Absolute Deviation

Example

To illustrate how to use the MAD functionality we will create a dummy data stream of Gaussian noise. This example will also explain how to use the profiler to tag messages as outliers or not.

Data Generator

We can create a simple python script to generate a stream of Gaussian noise at the frequency of one message per second as a python script which should be saved at ~/rand_gen.py.

```
#!/usr/bin/python
import random
import sys
import time
def main():
  mu = float(sys.argv[1])
  sigma = float(sys.argv[2])
  freq_s = int(sys.argv[3])
  while True:
    print str(random.gauss(mu, sigma))
    sys.stdout.flush()
    time.sleep(freq_s)

if __name__ == '__main__':
  main()
```

This script will take the following as arguments:

- The mean of the data generated
- The standard deviation of the data generated
- The frequency (in seconds) of the data generated

If, however, you'd like to test a longer tailed distribution, like the student t-distribution and have numpy installed, you can use the following as ~/rand_gen.py:

```
#!/usr/bin/python
import random
import sys
import time
import numpy as np

def main():
  df = float(sys.argv[1])
  freq_s = int(sys.argv[2])
  while True:
    print str(np.random.standard_t(df))
    sys.stdout.flush()
    time.sleep(freq_s)

if __name__ == '__main__':
  main()
```

This script will take the following as arguments:

- The degrees of freedom for the distribution
- The frequency (in seconds of the data generated

The Parser

We will create a parser that will take the single numbers in and create a message with a field called value in them using the CSVParser.

Add the following file to $METRON_HOME/config/zookeeper/parsers/mad.json:

```
{
  "parserClassName" : "org.apache.metron.parsers.csv.CSVParser"
 ,"sensorTopic" : "mad"
 ,"parserConfig" : {
    "columns" : {
      "value_str" : 0
                }
                   }
 ,"fieldTransformations" : [
    {
    "transformation" : "STELLAR"
   ,"output" : [ "value" ]
   ,"config" : {
      "value" : "TO_DOUBLE(value_str)"
               }
    }
                           ]
}
```

Enrichment and Threat Intelligence

We will set a threat triage level of 10 if a message generates a outlier score of more than 3.5. This cutoff will depend on your data and should be adjusted based on the assumed underlying distribution. Note that under the assumptions of normality, MAD will act as a robust estimator of the standard deviation, so the cutoff should be considered the number of standard deviations away.

For other distributions, there are other interpretations which will make sense in the context of measuring the "degree different".

Create the following in $METRON_HOME/config/zookeeper/enrichments/mad.json:

```
{
  "enrichment": {
    "fieldMap": {
      "stellar" : {
        "config" : {
          "parser_score" : "OUTLIER_MAD_SCORE(OUTLIER_MAD_STATE_MERGE(
PROFILE_GET( 'sketchy_mad', 'global', PROFILE_FIXED(10, 'MINUTES')) ),
 value)"
         ,"is_alert" : "if parser_score > 3.5 then true else is_alert"
        }
      }
    }
  ,"fieldToTypeMap": { }
  },
  "threatIntel": {
    "fieldMap": { },
    "fieldToTypeMap": { },
    "triageConfig" : {
      "riskLevelRules" : [
        {
          "rule" : "parser_score > 3.5",
          "score" : 10
        }
      ],
      "aggregator" : "MAX"
    }
  }
}
```

Related Information

Using the Median Absolute Deviation to Find Outliers

Index

We also need an indexing configuration.

Create the following in $METRON_HOME/config/zookeeper/enrichments/mad.json:

```
{
  "hdfs" : {
    "index": "mad",
    "batchSize": 1,
    "enabled" : true
  },
  "elasticsearch" : {
    "index": "mad",
    "batchSize": 1,
    "enabled" : true
  }
}
```

The Profiler

We can set up the profiler to track the MAD statistical state required to compute MAD. For the purposes of this demonstration, we will configure the profiler to capture statistics on the minute mark. We will capture a global statistical state for the value field and we will look back for a 5 minute window when computing the median.

Create the following file at $METRON_HOME/config/zookeeper/profiler.json:

```
{
  "profiles": [
    {
      "profile": "sketchy_mad",
      "foreach": "'global'",
      "onlyif": "true",
      "init" : {
        "s": "OUTLIER_MAD_STATE_MERGE(PROFILE_GET('sketchy_mad',
'global', PROFILE_FIXED(5, 'MINUTES')))"
                 },
      "update": {
        "s": "OUTLIER_MAD_ADD(s, value)"
                  },
      "result": "s"
    }
  ]
}
```

Adjust $METRON_HOME/config/zookeeper/global.json to adjust the capture duration:

```
 "profiler.client.period.duration" : "1",
 "profiler.client.period.duration.units" : "MINUTES"
```

Adjust $METRON_HOME/config/profiler.properties to adjust the capture duration by changing profiler.period.duration=15to profiler.period.duration=1

Execute the Flow

When you have finished configuring the MAD functionality, you will need to execute the flow.

Procedure

1. Install the elasticsearch head plugin by executing: /usr/share/elasticsearch/bin/plugin install mobz/elasticsearch-head.

2. Stop all other parser topologies via monit.
3. Create the mad Kafka topic by executing: /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --zookeeper node1:2181 --create --topic mad --partitions 1 --replication-factor 1.
4. Push the modified configs by executing: $METRON_HOME/bin/zk_load_configs.sh --mode PUSH -z node1:2181 -i $METRON_HOME/config/zookeeper/.
5. Start the profiler by executing: $METRON_HOME/bin/start_profiler_topology.sh.
6. Start the parser topology by executing: $METRON_HOME/bin/start_parser_topology.sh -k node1:6667 -z node1:2181 -s mad.
7. Ensure that the enrichment and indexing topologies are started. If not, then start those via monit or by hand.
8. Generate data into kafka by executing the following for at least 10 minutes: ~/rand_gen.py 0 1 1 | /usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --broker-list node1:6667 --topic mad. Note: If you chose to use the t-distribution script above, you would adjust the parameters of the rand_gen.py script accordingly.
9. Stop the above with ctrl-c and send an obvious outlier into kafka: echo "1000" | /usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --broker-list node1:6667 --topic mad.

   You should be able to find the outlier via the elasticsearch head plugin by searching for the messages where is_alert is true.