Lemon Tutorial

Sensor How-To

Miroslav Siket, Dennis Waldron
http://cern.ch/lemon
CERN-IT/FIO-FD

Outline

- Terminology
- Examples of existing sensors
- Considerations
- Live Examples
  - Hello World
  - Service based monitoring
- Do’s and Don’ts

Terminology

- Sensor:
  - A process or script which is connected to the lemon-agent via a bi-directional pipe and collects information on behalf of the agent. Sensors implement,
- Metric Classes:
  - The equivalent to a class in OOP (Object Orientated Programming)
- Metric Instance:
  - Is an instance (an object) of a metric class which has its own configuration data.
- Metric ID:
  - A unique identifier associated with a particular metric instance of a particular metric class.

Existing sensors

- At CERN:
  - Approx 40 active sensors defined, providing 264 metrics and 227 exceptions.
  - Default installation of the Lemon agent comes with three sensors:
    - MSA (builtin) – self monitoring of the agent.
    - Linux – performance, file system and process monitoring.
    - File – file tests e.g. size, mtime, ctime.
  - Together they provide 135 metrics (51% of all CERN metrics)
  - Other officially distributed sensors include:
    - exception – correlation sensor for generating alarms.
    - remote – provides ping and http web server checks.
    - oracle – oracle database statistics monitoring.
    - parselog – log file parsing sensor.
  - All available from the lemon software repository http://linuxsoft.cern.ch/lemon/
  - Other contributing sensors are available from CVS: CVSROOT=:pserver:anonymous@isscvs.cern.ch:/local/reps/elfms/sensors

Considerations

- Question: What is your goal? How do you intend to use the monitoring information you collect?
- Is it for:
  - Pure data collection?
    - OK
  - Graphs displayed on the lemon status pages?
    - Just because you've collected data doesn't give you graphs immediately! This is not automatic!
  - Information to be alarmed?
    - Make sure the structure of the data you collect can be alarmed!
    - Data that cannot be alarmed:
      - Timestamps as strings - NO
      - Timestamps as numbers - NO
      - Parsing of complex strings - NO

Considerations (II) - Use Case

- Grid Certificate Expiry Use Case

  Outline: you wish to be notified or raise an alarm if the Grid Certificate on a machine will expiry in the next two weeks.

- You need 1 metric and 1 exception
  - The metric will record the expiry time of the certificate.
  - The exception will check the metric and decide if it expires in the next two weeks.
- The metric needs to be structured in such a way that the correlation unit of the exception sensor can understand it.
- Can I record the data as a:
  - String e.g. “Sun Oct 8 16:05:47 2006” NO (Cannot be converted to a number)
  - UnixTime e.g. “1160316347” NO (Correlation unit doesn’t understand time, yet!!)
- Solution:
  - Record the number of seconds until the certificate expires.
  - E.g 1814400 seconds (3 wks) can be mathematical alarmed :-

    If metric < 1209600 (2 wks) then raise alarm

Considerations (III)

- Misconception:
  - In Lemon that a metric has to be related to one and only one distinct piece of information (1 to 1 mapping)
- Not true:
  - A metric can be associated with multiple values and have multi rows with each row identified by a unique key.

Considerations (IV) – Use Case

- Recording partition information

  Outline: you would like to know the total size, space used in megabytes, space used as a % and the mount options of all mounted partitions on a machine.

  – Under the idea of a 1 to 1 mapping, that's 4 metrics per partition. An average machine may have 7 partitions (4x7 = 28 metrics in total).

  – Why not:
    - Convert the data into a multi-valued metric?
    - 7 metrics each reporting 4 values. So,
      – Metric 1 total_space
      – Metric 2 space_used_mb
      – Metric 3 space_used_perc
      – Metric 4 mount_options

      Becomes:
      – Metric A total_space space_used_mb space_used_perc mount_options

  – Go one step further:
    - Convert the data into a multi-valued, multi-rowed metric
    - 1 metric reporting the values for all mount points. So,
      – Metric A total_space space_used_mb space_used_perc mount_options

      Becomes:
      – Metric B **mountname1** total_space space_used_mb space_used_perc mount_options
      - Metric B **mountname2** total_space space_used_mb space_used_perc mount_options
      - ….

  - Benefits:
    - Monitoring of new mount points is dynamic, no need for reconfigurations, no need to going through a registration process to get new metric ids.

Example 1 – Hello World

Objective: To create a Perl sensor which records the value “Hello World” into Lemon.

- Simple sensor to demonstrate:
  - The generic build framework for sensors.
  - How to registering your Perl module with the API.
  - How to register metric classes that your modules provides.
  - How to store the text “Hello World” for the machine under which the sensor runs into Lemon.
  - Running and debugging your sensor on the command line.
- Functions used:
  - registerVersion()
  - registerMetric()
  - storeSample01()
- Documented at:

  http://lemon.web.cern.ch/lemon/doc/howto/sensor_tutorial.shtml

Example 2 – Service Monitoring

Objective: To check if a webpage is available on a remote web server and record the HTTP response code under a service name.

- Demonstrates:
  - The basics of on behalf reporting
  - The ability to parse configuration arguments
  - The ability to log messages
- Functions used:
  - registerMetric()
  - getParam()
  - log()
  - storeSample03()

Do's and Don'ts

- Don't:
  - Call die() or exit() from inside your sensor.
  - Open or write to files in locations writeable by non-root users such as /tmp/
  - Read from filehandles (e.g sockets) that may block. This will make your sensor unresponsive to requests from the agent.
  - Never rely on, or have dependencies on files on remote file systems such as AFS (Andrew File System). Your sensor should aim to have as few dependencies as possible
- Do's:
  - Document your sensor. Refer to the sensor tutorial to see how this can be done automatically for you.
  - If you have the ability to use a timeout around calls to databases and services like LSF, use it!!
  - Make your metric classes configurable, avoid hard coded paths to non standard files.
  - Try to make your sensors as generic as possible so that others can benefit from your work.