**le-mon v0.1**

Probe Development Instructions

Teemu Kanstrén

Contents

[Introduction 3](#_Toc335748577)

[What is a probe? 3](#_Toc335748578)

[Integrating your probes with le-mon 3](#_Toc335748579)

# Introduction

This document describes how to integrate your probes with le-mon.

# What is a probe?

The basic architecture of a deployed le-mon system is shown in Figure XX.

Examples of probes include:

* Data from a Nagios server
* Results from running an SSH script
* Custom code reading custom sensors

# Integrating your probes with le-mon

To integrate with le-mon, two different basic options are provided. One is a push-based solution, while the other is a pull-based solution. The difference with the two is that in a push-based solution, the probe pushes the data to le-mon at its own will. In a pull-based solution, le-mon will query the probe for data at predefined intervals.

**Pull-Based**

You need to implement the fi.vtt.lemon.probe.shared.Probe interface for you probe agent.

An example of a pull-based probe-agent can be found in the class fi.vtt.lemon.probes.ssh.SSHProbeAgent. Let’s take a look at it next.

public class SSHProbeAgent extends BaseProbeAgent {

…

public String getMeasureURI() {

return measureURI;

}

public int getPrecision() {

return precision;

}

public String measure() {

try {

String result = executeScript();

log.debug("measurement result:" + result);

return result;

} catch (Exception e) {

throw new RuntimeException("Failed to perform measure for " + measureURI + ", ", e);

}

}

//executes the configured shell script on the configured target

private String executeScript() throws Exception { …

}

}

This partial code sample shows the measure() function implementation of the SSHProbeAgent. It also shows the getMeasureURI() and getPrecision() functions that the MeasurementProvider uses to capture information about the data the probe provides. You then start the le-mon measurement provider with the following code.

MeasurementProvider mp = new MeasurementProvider(new ServerClient("::1"), 5, 10);

mp.setInterval(1);

mp.startMeasuring(this);

This results in creation of a MeasurementProvider object with a thread pool of size 5 and timeout value of 10 seconds. The measurement interval is 1 second, meaning the MeasurementProvider will ask the probe agent to provide a new measurement every one second. The thread pool configuration defines how many threads will be created to run measurements for this probe-agent. The timeout defines how long the framework waits for a single measurement task to complete before aborting it. The number of threads defines how many tasks it can start for different measurements. You can add as many probes as you like to a MeasurementProvider object with the .startMeasuring() function.

The ServerClient takes as an argument the address of the RabbitMQ broker (here the IPv6 localhost).

The abortion feature is implemented to deal with the observed issue that in some cases errors in probe implementation can hang the probe execution, which would otherwise block all probes from ever executing.

**Push-Based**

In this case, you are free to implement the probe-agent as you best see fit. If we look at the example of the fi.vtt.lemon.probes.http.HTTPProbeAgent, we will see that all you need to do is to send data through the le-mon components.

public class HTTPProbeAgent implements Filter {

private final static Logger log = new Logger(HTTPProbeAgent.class);

private final ServerClient server;

private final String measureURI;

private final int precision;

public HTTPProbeAgent(ServerClient server, String measureURI, int precision) {

this.server = server;

this.measureURI = measureURI;

this.precision = precision;

}

…

public void doFilter(ServletRequest servletRequest, ServletResponse resp, FilterChain chain) throws IOException, ServletException {

…

server.measurement(measureURI, precision, content);

…

}

}

Here the ServerClient object provides an interface to deliver measurements to the server. It is up to you to decide how often to call it to provide new measurements etc.