Stress Test Plan

Prepared by: Filippo Giuffrida

Date: 20/08/2019

Version: 1.0

INTRODUCTION

The Digital Enabler is the Engineering’s Digital Ecosystem Platform. The platform allows collecting, integrating and harmonizing data and, in the end, visualizing them following a 100% self-service approach.

The data can come from devices, open data and/or legacy systems and all of them can be easily connected to the platform without writing a single line of code.

OBJECTIVES AND TASKS

Objectives

The main objective of the test sessions will be to detect potential stability issues of the platform and, in the end, improve the overall quality of service of the platform.

All the tools of the platform should be provided in High Availability so it’s very important to prove that such a requirement is respected and, if not, to detect the action required to obtain it. The resilience, the self-healing capabilities and the ability to handle a high volume of input traffic are only some of the necessary characteristics of a “Highly Available” solution.

In order to test these characteristics the platform will be subject of several and iterative test sessions. Such tests will be of different kind, mainly Chaos Tests and Stress Tests.

Stress Testing is a type of Software Testing that verified the stability & reliability of the system. This test mainly determines the system on its robustness and error handling under extremely heavy load conditions.

Stress Testing is done to make sure that the system would not crash under crunch situations

Tasks

The tasks that for each stress test session must be addressed are:

Test session preparation

This task implies the following preventive activities:

- Compare the upcoming session with the last one and detect any new Test Case that must be implemented
  - Writing the new test scripts
• Check that all the prerequisites of the upcoming session are respected.
• Perform a backup of all the data belonging to the involved tools
• Few minutes before the test session, notifying the community that a test session is going to start and that this could result in a performance degradation. Report also the expected time when the platform will be back 100% working again;

Test session execution

Once the test session starts, the tasks are:
• Running the scripts in the planned order;
• Check that the scripts are working properly;
• Collect the evidences;
• Monitor the time to be sure that the planned timeslot is respected.

Post testing activities

After the test session the following activities must be addressed:
• Be sure that the platform is 100% working again;
  o If necessary, perform whichever activity needed to make the platform working again;
• Be sure that all the evidences are collected properly;
• Notify the community that the test session ended and report the next planned session.

Reporting of the results

The results reporting is maybe the most important part of the testing activities. It implies the formalization of the collected evidences and the writing of the proper “Test Session details” sub-section of this document.

In the report be sure that the following information are explicitly and clearly reported:
• who participated to the test;
• which tools have been involved in the session and make explicitly clear;
• which are the used scripts and which is the load level that they simulate;
• which are the collected evidences;
  o where available put also explanatory charts and tables;
• report all the problems encountered during the session (e.g. misbehaviour, misfunctioning, etc..) and for each of them describe which could be the mitigation actions that will solve it
• In case a problem is caused by a software bug or implies some kind of action by the development team, report the problem on an Issue Tracker making explicit the reference to the Test Session ID during which the issue arose.

At the end of the report preparation, upload the new version of this document on the community.

The entire process can be formalized by the following activity diagram:
**TESTING STRATEGY**

The Digital Enabler team will execute two stress test session per month. One Main Session and a Mitigation Session.

Each **Main session** will be used to:

- test new features,
- extend the testing boundaries
- collect new kind of evidences.

After each Main Session the bugs and required new features will be fixed and developed and then, after 2 weeks (1 sprint), a **Mitigation Session** will be performed.

During such a session, the testing team will focus more on:

- Check that any kind of regression have not been injected in the system
- The expected improvements took place

The tools that the testing team will use are:

- **JMeter** ([https://jmeter.apache.org/](https://jmeter.apache.org/)) to simulate an extreme load from external system (e.g. API invocation, Sensor Measures pushing, etc...)
- **Selenium** ([https://www.seleniumhq.org](https://www.seleniumhq.org)) to simulate the GUI navigation (through Browsers) by the users.
- **Nagios** ([https://www.nagios.org/](https://www.nagios.org/)) to collect evidences at infrastructure level
  - **Check_MK** ([https://mathias-kettner.com/](https://mathias-kettner.com/)) is the GUI that allows to easily navigate the Nagios insights via browser.
- **Jira** ([http://jira.com](http://jira.com)) to report bugs to the Development team
- **Digital Enabler community portal** ([https://issues.digitalenabler.eng.it](https://issues.digitalenabler.eng.it)) to notify the community about the execution of the test session.

Once each test session finishes, the Digital Enabler team will compile a report and then publish it in the Digital Enabler community ([https://community.digitalenabler.eng.it/documents/2](https://community.digitalenabler.eng.it/documents/2)).

**TESTING TEAM**

The testing team is composed by the following people:

<table>
<thead>
<tr>
<th>NAME AND SURNAMER</th>
<th>EMAIL</th>
<th>ROLE</th>
<th>RESPONSIBILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antonino Sirchia</td>
<td><a href="mailto:antonino.sirchia@eng.it">antonino.sirchia@eng.it</a></td>
<td>Team Leader</td>
<td>To define the testing scenarios; coordinate the activities; writing the report; report bugs.</td>
</tr>
<tr>
<td>Filippo Giuffrida</td>
<td><a href="mailto:filippo.giuffrida@eng.it">filippo.giuffrida@eng.it</a></td>
<td>Script writer and executor</td>
<td>Write the scripts, run the scripts; collect evidences; contributing in writing the report; report bugs.</td>
</tr>
<tr>
<td>Ivan Ligotino</td>
<td><a href="mailto:ivan.ligotino@gmail.com">ivan.ligotino@gmail.com</a></td>
<td>Script writer and executor</td>
<td>Write the scripts, run the scripts; collect evidences; contributing in writing the report; report bugs.</td>
</tr>
</tbody>
</table>
This session has been coordinated by Filippo Giuffrida.

**RISKS AND ASSUMPTIONS**

*The main risks that must be foreseen for the sessions are:*

- **Irreversible data corruption in production**

In case the test session will result in an irreversible data corruption the test team MUST have a very recent backup ready to be restored just after the test session.

- **Delay in delivery a mitigation action**

In case a mitigation would not be delivered in time, the next planned mitigation session should not be delayed. It must be anyway executed to test the rest of the involved tools and the already completed mitigation activities. A further session (more focused on the still missing mitigation actions) must be planned BEFORE the next planned Main session.

**SESSIONS SCHEDULE**

<table>
<thead>
<tr>
<th>Test Session ID</th>
<th>Test Session Type</th>
<th>Test Session Date</th>
<th>Test Session Time Slot</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>MAIN</td>
<td>12/06/2019</td>
<td>09:30 – 10:30</td>
</tr>
<tr>
<td>#2</td>
<td>MITIGATION</td>
<td>29/07/2019</td>
<td>11:00 – 13:00, 14:00 – 14:30</td>
</tr>
<tr>
<td>#3</td>
<td>MAIN</td>
<td>20/08/2019</td>
<td>12:21 – 13:21</td>
</tr>
<tr>
<td>#4</td>
<td>MITIGATION</td>
<td>29/08/2019</td>
<td>09:30 – 10:30</td>
</tr>
<tr>
<td>#5</td>
<td>MAIN</td>
<td>09/09/2019</td>
<td>09:30 – 10:30</td>
</tr>
<tr>
<td>#6</td>
<td>MITIGATION</td>
<td>26/09/2019</td>
<td>09:30 – 10:30</td>
</tr>
</tbody>
</table>

Session planning was rescheduled due to some unexpected events.

**TEST SESSIONS DETAILS**

**Session #3 – 20/08/2019**

The session #3 tested the ability of the platform to deal with an increasing amount of incoming traffic.

The main expected output of this session are:

- Horizontal scaling rules efficiency
- Impact of the input load focused on the platform uptime
- Impact of the input load focused on the Web GUI User Experience
- Correlation between the input load and the resource consumption (CPU, RAM) for each of the involved components (horizontal scaling enabled)
The intervention plans emerged in stress test session #1 and #2 were provided as input for session 3 interventions: Replication of reverse proxy (nginx) over 2 instances, decomposition of the Data Mashup Editor in 2 components (GUI and APIs in order to monitor each intervention individually).

Due to MQTT disconnection problems highlighted in session #2 we have decided to not test MQTT devices during this test, in order to have some evidence about HTTP protocol.

**Lab Environment**

The tools involved in the stress test session are:

- Gateway
- FIWARE IoT Agents JSON
- FIWARE IoT Agents UL2.0
- FIWARE Orion Context Broker
- Mashup Execution Engine

These tools are distributed over the machines *xerus, beaver, xerus-1, xerus-1-2, xerus-7 and xerus-7-2* hosts as follows:
The flavors of each host machine are the followings:

- **xerus**: m1.small
- **beaver**: m1.large
- **xerus-1**: m1.large
- **xerus-1-2**: m1.large
- **xerus-7**: m1.big
- **xerus-7-2**: m1.big

Where each flavour has the following resources:

<table>
<thead>
<tr>
<th>FlavorID</th>
<th>RAM</th>
<th>vCPU</th>
<th>DISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>2GB</td>
<td>1</td>
<td>20GB</td>
</tr>
<tr>
<td>m1.medium</td>
<td>4GB</td>
<td>2</td>
<td>40GB</td>
</tr>
<tr>
<td>m1.large</td>
<td>8GB</td>
<td>4</td>
<td>80GB</td>
</tr>
<tr>
<td>m1.big</td>
<td>16GB</td>
<td>4</td>
<td>80GB</td>
</tr>
</tbody>
</table>

**Prerequisites:**

- The following tools must be replicated by default and able to scale on, at least, two different VMs:
  - Nginx
  - IoT Agents JSON
  - IoT Agents UL2.0
  - Orion Context Broker
  - Mashup Execution Engine

- The following monitoring tools are correctly up and running:
  - cAdvisor\(^1\) (v0.32.0)
  - Prometheus\(^2\) (v2.10.0)

- The autoscaling tool\(^3\) (v0.0.2) is up and running

- The following autoscaling rules are enabled:

\(^1\) [https://hub.docker.com/r/google/cadvisor](https://hub.docker.com/r/google/cadvisor)
\(^2\) [https://hub.docker.com/r/prom/prometheus](https://hub.docker.com/r/prom/prometheus)
\(^3\) [https://hub.docker.com/r/sahaisoft/docker-swarm-service-autoscaler](https://hub.docker.com/r/sahaisoft/docker-swarm-service-autoscaler)
In order to enable the automatic horizontal scaling for the IoT Agent JSON and IoT Agent UL2.0

- There are at least:
  - 1 HTTP device for JSON format
  - 1 HTTP device for UL2.0 format

- The **JMeter** tool is ready to simulate a load for the four devices with the following payloads:

```
{
  "temperature": 12.5,
  "humidity": 67.3,
  "ObservationDateTime": "2019-01-08T18:34:43.00Z"
}
```

- All the involved devices are connected to a Mashup that is in charge of harmonizing the payload of the measure to the WeatherObserved FIWARE Datamodel

- The mashup publishes the harmonized data onto the Orion Context Broker
Execution:

The session has been executed on 20/08/2019 from 12:21 am to 13:21 am.

The session consists in the simulation of an increasing amount of measures sent by a fixed number of IoT Devices.

The simulated load, for each device, increases in time accordingly with the curve shown in the following picture:

- **Step 1:**
  - Duration of load increase: 3 min and 20 sec
  - Active virtual users: **100**
  - Sending frequency for each virtual user: 1 measure/sec.
  - Total input to the platform: 100 measures/sec.
  - Full duration: 10 min.

- **Step 2:**
  - Duration of load increase: 3 min and 20 sec
  - Active virtual users: **200**
  - Sending frequency for each virtual user: 1 measure/sec.
  - Total input to the platform: 200 measures/sec.
  - Full duration: 10 min.

- **Step 3:**
  - Duration of load increase: 10 min.
  - Active virtual users: **400**
  - Sending frequency for each virtual user: 1 measure/sec.
  - Total input to the platform: 400 measures/sec.
  - Full duration: 10 min.

- **Step 4:**
  - Duration of load increase: 10 min.
Active virtual users: 800
Sending frequency for each virtual user: 1 measure/sec.
Total input to the platform: 800 measures/sec.
Full duration: 10 min.

Step 5:
Duration of load increase: 6 min. 40 sec.
Active virtual users: 1000
Sending frequency for each virtual user: 1 measure/sec.
Total input to the platform: 1000 measures/sec.
Full duration: 10 min.

Since the number of virtual devices involved in the simulation is 2, the load will go from a baseline of 200 measures to maximum of 2000 measures.

Results reporting

The total duration of the session has been 60 minutes. Unlike previous sessions, the test has been performed in its entirety, since the Reverse Proxy discreetly worked until the end.

At the end of the session the platform came back 100% working again.

The session didn’t caused any data corruption.

Collected evidences:

From an infrastructural point of view, the availability of the platform has not been affected at all. As the following screenshot shows, the availability of the involved machines during the entire test session timeframe remained at 100%.
Nevertheless, some application unavailability have been experienced during the session.

In particular, although it certainly performed better than the previous sessions, the Reverse Proxy has not been able to manage all the incoming requests:

In the first step (first 20 minutes), when we execute 100 measures/second for each agent, with a total of 200 measures/second, we have had no errors.

In the second step (started at 20 minutes from the begin), when we execute 200 measures/second for each agent, with a total of 400 measures/second, some errors have begun to appear.

In the next steps the number of errors increased, but this however remained low and the system continued to work.

In the following image it is shown the analysis result on a set of jmeter connections for HTTP format/agent.

The image shows that the number of errors increased advancing the steps and the number of measures/second:

1. 0.83% → 200 measures/second
2. 1.67 → 400 measures/second
3. 3.77% → 800 measures/second
4. 8.68% → 1600 measures/second
5. 18.89% → 2000 measures/second

But the important thing is that in general the number of errors was low: 3.04%.

The mashup execution engine correctly worked until the end of the test session: the high number of task for the engine has generated a small delay in the execution and publication of the harmonized measures, the proof of this is that the last update of the involved entities is after to the end time of the session.

The time shown in the screenshot is server time (2 hours earlier).

It is important to note that the Mashup Editor (Web GUI) has been unavailable starting from the begin of the test session. Checking the Mashup Editor console, this error appears:
org.postgresql.util.PSQLException: FATAL: sorry, too many clients already

This means that also if the APIs have been separated from the GUI, the access to the same database gives problems. These problems are evident on the GUI, less on the APIs, where since the big number of operations, we were not able to understand if any operation has been lost.

From Orion point of view, we noticed that the notifications from Orion to Mashup APIs were failed and that the last success was when the process begun.

```
"lastFailure": "2019-08-20T11:21:04.00Z",
"lastSuccess": "2019-08-20T10:21:07.00Z"
```

Since the Mashup were executed and the harmonized measures published, we supposed that this just a bug on the “lastFailure” and “lastSuccess” date update, but the process worked fine.

**Conclusions:**

The involved IoT Agents have been able to manage the load.

No big problems arose about the Orion Context Broker.

The Mashup EMML Engines worked properly, the Mashup GUI was down for the whole test time.

The development tasks emerged by this test session was the followig:

- Improve the database connection management in the Mashup components.

The improvements will be tested during the next test session.

**Intervention plan:**

Comparing this session with the previous ones, the problems with the Reverse Proxy have been greatly reduced.

In order to mitigate the issues that this session highlighted, the team intends to investigate in order to find a better configuration for the database connections in the Mashup components.

The other task emerged in the session #2 remains valid, in particular the disconnection problem with the MQTT protocol. The team intend to investigate in order to find a better configuration to manage this MQTT connection.

This mitigation will have a high priority in the Digital Enabler’s backlog and will be available by the next mitigation session.