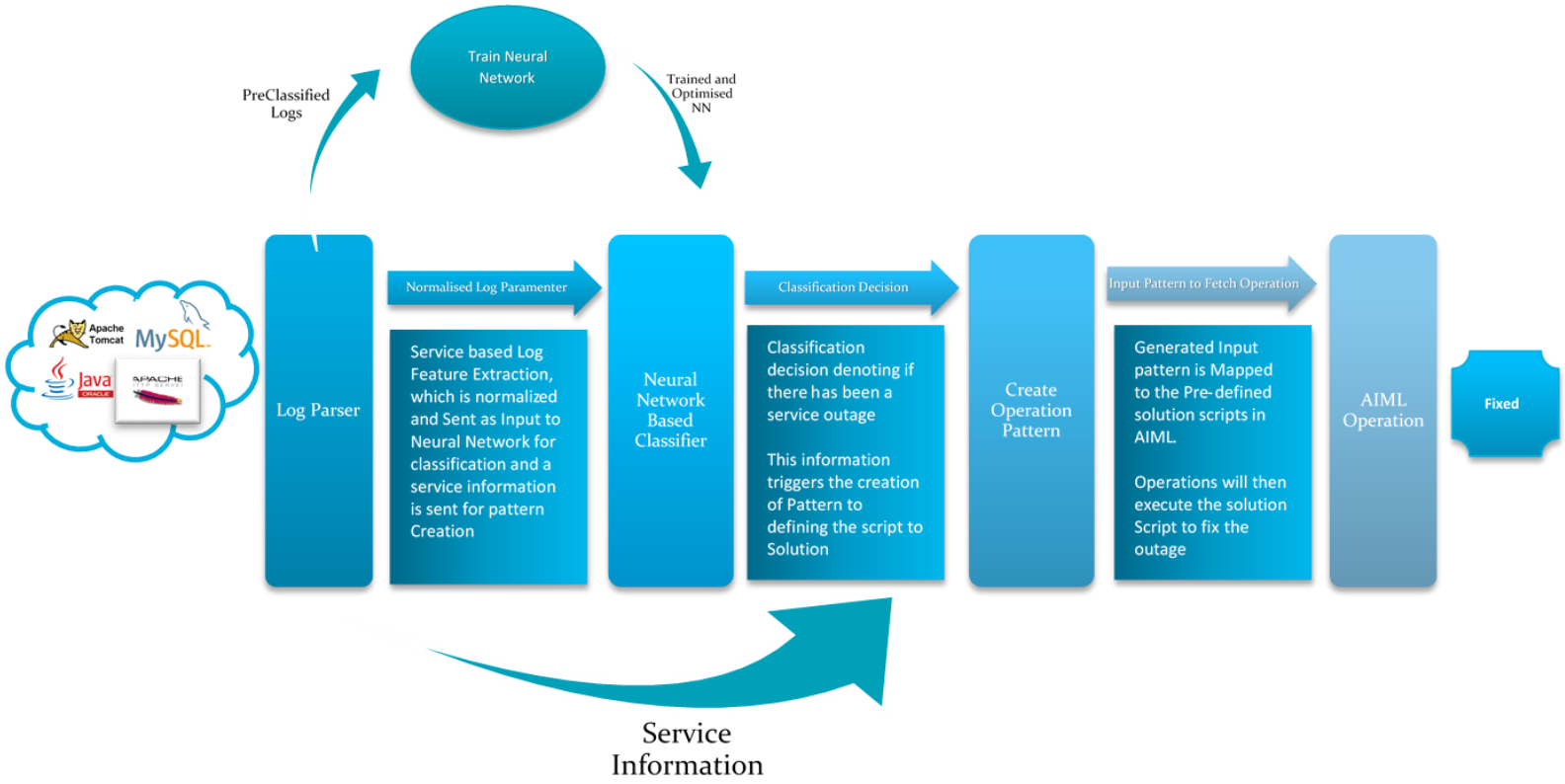

# Automation Engine

PHASE 0.1 DOCUMENTATION

Devops Technology Inc | 11th October 2016

# System Architecture Diagram.




## INITIAL REQUIREMENTS

All the following are based in Linux/Unix Environment. Ubuntu 14.04 LTS and above.

The pre-requisites for the build are

- Python
- Python packages:
    - Numpy
    - Scipy
    - PIP - for Installation
    - Regular expression Parser Packages
    - AIML – Artificial Intelligence Markup Language
    - Java

Instruction for installation of pre-requisites:

## LOG PARSER

The process starts with parsing the Service error log or system log into various defining parameters. Each service has a logging format. In this phase of the automation engine deals only with One Service: Apache HTTP Server. The log being analyzed is the Error Log of Apache.

A typical Log of apache Looks like:

[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1] client denied by server configuration: /export/home/live/ap/htdocs/test

In a typical server the Apache Error Log is stored in **/var/apache2/error.log** file or the path defined in the **apache.conf** (Apache Configuration File) by the directive:

```
ErrorLog "/var/log/httpd/error_log"
```

Format for the error Log can be specified in the **apache.conf** file using the directive

```
ErrorLogFormat "[%t] [%l] [pid %P] %F: %E: [client %a] %M"
```

The Table below gives you the format strings and their description that can be used to define a Apache Log format:

| Format String | Description |
|---|---|
| %% | The percent sign |
| %a | Client IP address and port of the request |
| %{c}a | Underlying peer IP address and port of the connection (see the mod_remoteip module) |
| %A | Local IP-address and port |
| %{*name*}e | Request environment variable *name* |
| %E | APR/OS error status code and string |
| %F | Source file name and line number of the log call |
| %{*name*} i | Request header *name* |
| %k | Number of keep-alive requests on this connection |
| %l | Loglevel of the message |

| | |
|---|---|
| %L | Log ID of the request |
| %{c}L | Log ID of the connection |
| %{C}L | Log ID of the connection if used in connection scope, empty otherwise |
| %m | Name of the module logging the message |
| %M | The actual log message |
| %{*name*}n | Request note *name* |
| %P | Process ID of current process |
| %T | Thread ID of current thread |
| %{g}T | System unique thread ID of current thread (the same ID as displayed by e.g. top; currently Linux only) |
| %t | The current time |
| %{u}t | The current time including micro-seconds |
| %{cu}t | The current time in compact ISO 8601 format, including micro-seconds |
| %v | The canonical ServerName of the current server. |
| %V | The server name of the server serving the request according to the UseCanonicalName setting. |
| \ (backslash space) | Non-field delimiting space |
| % (percent space) | Field delimiter (no output) |

Few Useful Links Include:

http://httpd.apache.org/docs/current/mod/core.html#errorlog

https://httpd.apache.org/docs/1.3/logs.html

http://httpd.apache.org/docs/2.2/logs.html#errorlog

The log parser in this system parses the incoming log line using regular expression defined to extract information is groups. The regular expression and function to return the parsed output of the log line in form or groups are defines in **LogParser.py**

In the Main.py program the parseFile() function reads the log file from the lastUpdate time of the log defining the previous parse on the log Data.

```
parserGroups=logObj.parseLog(line)
```

this call the parseLog() function defined in LogParser.py which return the parse output in form of groups

```
('[Fri Oct 07 10:45:44 2016]', 'Fri Oct 07 10:45:44 2016', '[notice]',
'notice',    '[pid    10820]',    '10820',    'prefork.c(1146):    ',
'prefork.c(1146)', 'client AH00169: ', 'client AH00169', None, None,
'caught SIGTERM, shutting down')
```

The Parsing happen by extectuing the following command to parse the logline against the defined regular expression.

```
parserGroups=re.match(self.regex,line)
```

each group can be accessed using the array access representation like

```
parserGroups[4] or parserGroups.groups() to print the output
```

## NEURAL NETWORK CLASSIFICATION ENGINE

The Neural Network Classification Engine is built to take in parameters got form the logs and classify them into critical error state or non-critical state. Neural in this system is configured and trained to take 5 parameters which are normalized and produce a near 0/1 output denoting the state of the service.

The output from the Neural Network classifier triggers a Pattern Generator module on output State 1 to create a service error pattern including the name of the service causing the down state of the system and include fee details which is parsed to get a possible solution script form AIML parser.

## PATTERN GENERATOR

The pattern Generator module is trigger only on 1(classification state of error) – output State of the Neural Network classifier.

In this system, Pattern of error defines the solution script.

```
self.getCommand('Apache2')
```

in this 'Apache2' the pattern defines the service causing the problem.

Various Pattern Definition can be employed in this system the definition will be like:

[<**service**>,<**problem**>] Listing.

This pattern is parsed using AIML Parser to get the solution script. Pattern generator takes the service information from the triggering service to formulate a pattern and returns the AIML parse able pattern.

The model named **patternGenerator.py** in the Package defines the method to formulate the pattern.

## AIML

AIML stands for artificial intelligence markup language. AIML is an XML type language primary created to match patterns to templates in Chat bot environments.

Classic definition of AIML script looks like:

```
<aiml version="1.0.1" encoding="UTF-8">
<!-- basicOps.aiml -->
    <category>
        <pattern>APACHE2</pattern>
        <template>
            sudo service apache2 restart
        </template>
    </category>
</aiml>
```

Where each tag is defined by

## PROGRAM FLOW