How to use Load Testing Framework

# How it works?

At the heart Load Testing Framework provides 3 things, all driven by the parameters/info specified in the test configuration–

1. Ability to deploy the silo and client binaries specified in the **Deployment** section to specified list of machines specified in **Cluster** section of config, start specified processes on those machines. (Also watch for process crashes if any).
2. Parse the output of running clients and extract the metrics from the output based on grammar specified in the **QuickParser** section of the config, convert these metrics to other derived metrics based on **MetricCollector** section of the config.
3. Save log files and send summary email.

More details are as follows

1. Load testing framework deploys silos and client application to a number of machines as specified in the “Cluster” Section of the test configuration.
   1. The silos are started in parallel, waiting until each silo starts writing to log file. The log file of the silo is watched continuously.
   2. Next the clients are started in parallel waiting until each client outputs enough output to be considered “WarmedUp”. The console output of each client is stored in a file.
2. Log files and output are constantly monitored and parsed using the grammar/patterns defined in “QuickParser” section of the test configuration.
   1. QuickParser grammar is used to extract various metric data from the files. This happens as soon as new lines are written to the logs/console.
   2. In addition to predefined states “Initialized”, “Started”, “WarmedUp” , “Stable” and “Finished” you can define your own states and transition rules.
   3. Regular expression are used to lex the lines and extract information. This defines *primary metric* that are used in the next steps
   4. When a file is being watched the state of the parser is “Initialized” which automatically transitions to “Started” state as soon as it is being written at least once. Now onwards the transitions are driven by user provided patterns.
3. The metrics extracted from various files in step 2 is fed into metric collector which uses the “MetricCollector” section.
   1. Various derivative metrics like average, moving average , percentile etc. are specified in metric definition. These secondary metrics use primary metric data parsed by QuickParser as the basis. The metric is calculated as the new data points become available.
   2. The metric assets are used to monitor that metrics are within the expected range.
   3. All the primary and calculated metrics are stored to a file.
4. At the end of a test all log files are stored in folder so that it can be used for post facto analysis.

# How to specify Load Test Configuration?

## Cluster Section.

Example:

<Cluster Name="nightly\_build\_cluster" AllowMachineReuse="false">

<Servers Prefix="17xcg17" Start="1" End="30" Skip="3,7,24,27"/>

<Clients Prefix="17xcg17" Start="31" End="44" Skip="32,33"/>

<Networking Subnet="10.68" StartPort="11000" GatewayPort="30000"/>

</Cluster>

1. Name the section so that it can be used in the test.
2. Specify set of machines using <Servers> and <Clients> tags.
   1. The range of machines is specified using Prefix, Start and End attributes.
   2. The configuration file template is specified using ConfigTemplate.
   3. To specify non-contiguous set of machines use multiple tags or use Skip
3. Specify if multiple instances of silos/clients can be run by specifying AllowMachineReuse
4. Specify details for networking using Networking section.

## Deployment Section.

Example:

<Deployment Name="nightly\_build">

<Packages>

<Package Name="SDK Drop"

Path="Deployment\Orleans"

Type="SDK"/>

<Package Name="PresenceConsole"

Path="$(Configuration)\TestInput\LoadTests\Client\PresenceConsoleTest.exe"

Type="ClientApp"/>

<Package Name="UserLoadTest"

Path="$(Configuration)\TestInput\LoadTests\Client\UserLoadTest.exe"

Type="ClientApp"/>

<Package Name="Reach Presence"

Path="$(Configuration)\TestInput\LoadTests\Grains"

Type="Application"/>

</Packages>

<Packages Root="\\17xcg1701\C$\">

<Package Name="output" Path="TestResults\SavedLogs" Type="Logs"/>

</Packages>

</Deployment>

1. Specify multiple paths to for deployment using multiple packages sections under Deployment section
2. All paths are relative to the explicitly specified **root** or “Binaries” folder under TFS
3. For **Type** attribute, following values are supported.
   1. SDK – path for SDK drop
   2. ClientApp – path for client application
   3. Application – Directory where grains are stored.
   4. Logs – where logs are stored.

## QuickParser Section.

<QuickParser Name="ClientLogForReliability">

<Transition Name="Rule1"

From="Started" To="Printing"

Pattern="Current TPS:" Count="1"/>

<Transition Name="Rule4"

From="Printing" To="Stable"

Pattern="Current TPS:" Count="30" />

<Transition Name="Rule5"

From="Unstable" To="Stable"

Pattern="Current TPS:" Count="5" />

<Lexer Name="Pattern1" State="Stable" Pattern="Current TPS:(?{tps} \d+.\d+)">

<Variable Name="tps"/>

</Lexer>

</QuickParser>

When a file is being watched the state of the parser is “Initialized” which automatically transitions to “Started” state as soon as it is being written at least once. Now onwards the transitions are driven by user provided patterns.

1. The transition is defined using <Transition> tag.
   1. When the pattern specified in Pattern attribute is encountered number of times specified in Count then the parser transitions from From state to To state. Set Consecutive to restrict to case when matching lines are consecutive.
   2. You can also specify the Priority in case there are multiple matching rules for a state.
2. The lexing or metric extraction pattern is specified using <Lexer> tag.
   1. The applicable state is provided by State attribute.
   2. Pattern provides the .net regular expression syntax.
   3. To avoid using angled brackets the variables are enclosed in {}. The name of the variables is provided in <Variable> child elements. *These variables names are considered primary metric.*
   4. *When* AutoVariables="true" the line is assumed to be commas separated with pattern Varibale : 0000.00 unit,

## Metric Collector Section.

Example:

<MetricCollector Name="MetricDefinition2" ExitEarly="true">

<Metric Type="AverageMetric"

Name="TPS Moving Average"

BasedOn="tps"

WindowSize="20"

Scope="Moving"/>

<Metric Type="AggregateMetric"

Name="Aggregate of TPS"

BasedOn="tps"

WindowSize="20"

Scope="Global"/>

<!-- each client needs to see TPS above 2000 -->

<MetricAssert Type="MetricWatermarkAssert"

BasedOn="tps"

LowWatermark="0.5" HighWatermark="10"

IsGlobal ="false" WindowSize="5" Strict="true" ScaleBy="ScaleFactorPerClient"/>

</MetricCollector>

The metrics are defined using following attributes

All values extracted using a parser are treated as primary metrics

1. The type of Metric – following types are provided.
   1. Aggregate
   2. Average
   3. Count
   4. Min
   5. Max
   6. Percentile
2. Scope of metric
   1. Moving – the metric is calculated on a moving window
   2. Global – metric is calculated across all clients.
   3. Period – the window jumps for each period

How does scope work ?

Global : one datapoint is taken from each sender, and metric calculated across them

Moving : The window moves one item at a time

so for window size of 3 we have

[1,2,3,4,5,6] => [1,2,3] -> [2,3,4] -> [3,4,5]

Period : The window jumps

so for window size of 3 we have

[1,2,3,4,5,6] => [1,2,3] -> [4,5,6]

1. Window size
2. BasedOn and Pass – the metrics can be based on values of other metrics or parsed values.
3. Name – name of the metric

Metric asserts are based on metrics values calculated and the value must be within the range provided. Strict="true" the all values must fall within the range.

# Paths and unique ids.

* Each test run gets a unique\_run\_id : <UserName>-<MachineName>-d-MMM-yyyy-HH-mm.
* All files are copied under C:\TestResults\unique\_run\_id.
* Silos and Clients are named as follows Silo0, Silo1…SiloNN and Client0,Client1…ClientNN.
* The binaries for silo and clients are copied to C:\TestResults\unique\_run\_id\SiloNN or C:\TestResults\unique\_run\_id\ClientNN.
* The output of the client is saved in Test\_<date>clientNN.log file.
* All the generated configs are saved under GeneratedConfigs folder.