Anatomy and classification of load testing tools

by Andrei Pokhilko, JMeter-Plugins.org

My Experience Building Tools

- JMeter-Plugins.org
- Loadosophia
- Yandex.Tank
- Blazemeter Taurus
- Encarno

BlazeMeter

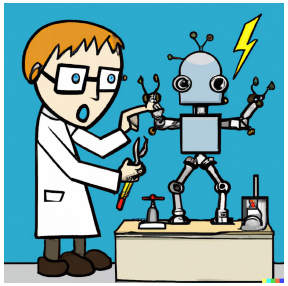
Taurus

Why to Learn Tools Internals

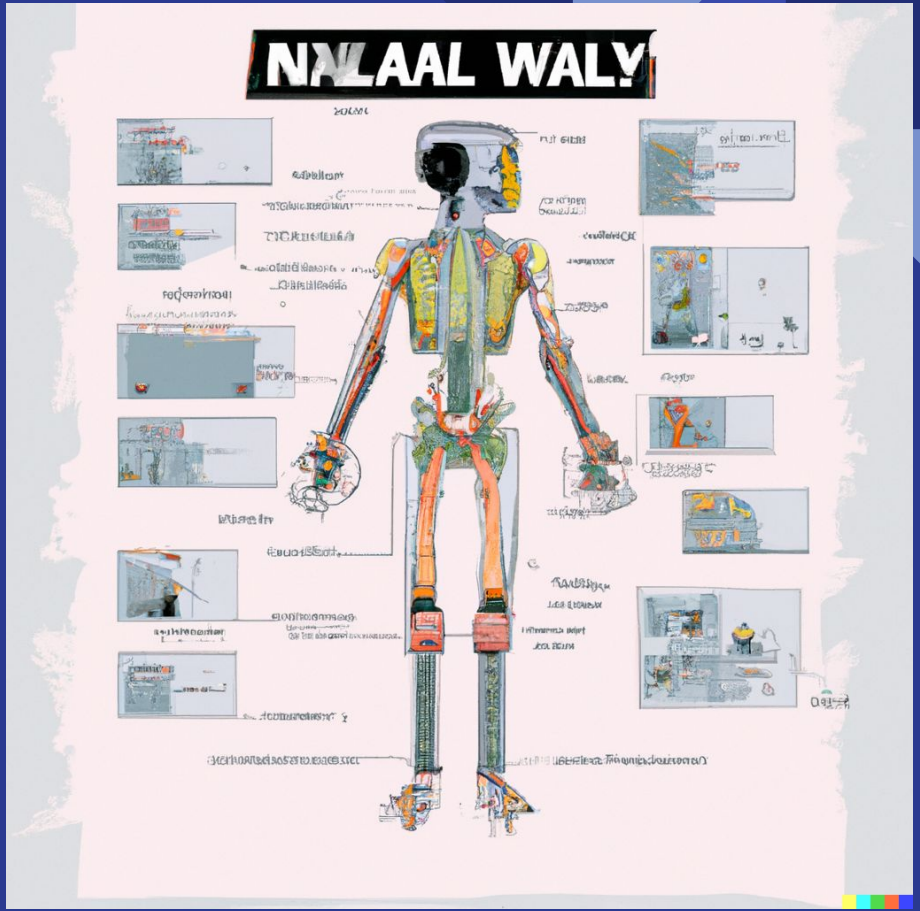
Choosing a tool

Use your tool better

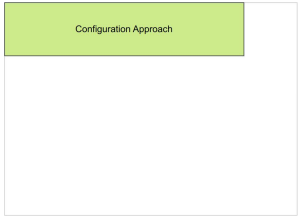
Develop your own



Anatomy

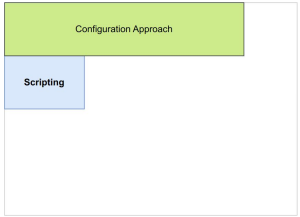


Configuration Interface

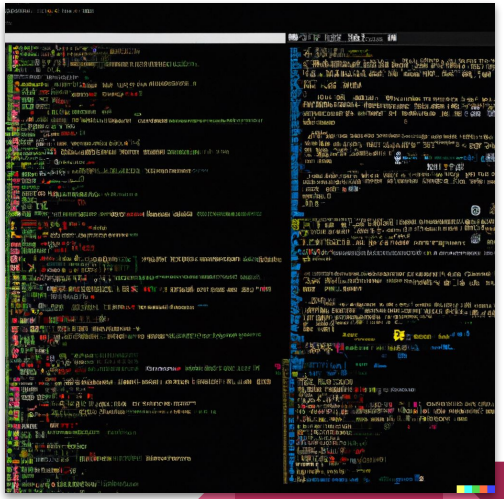


- Command-line parameters
- Environment variables
- Configuration files
- GUI
- Programmatic configuration (function calls etc.)

Scripting Engine

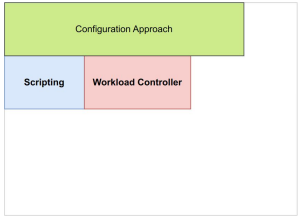


- Protocol adapter is part of it
- Can be as simple as single request
- All the way to full programming language
- Log replay as useful DevOps alternative

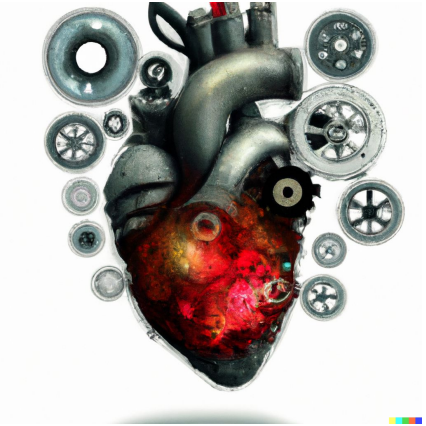


Familiarity of scripting language
often guides (and misleads) the tool choice.

Workload Controller

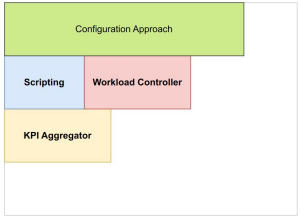


- It’s a heart of load testing tool
- Runs the script according to workload profile
- Rare tool gets beyond parallel workers
- Throughput, arrivals, open workload model



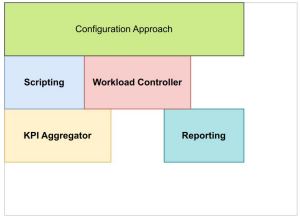
The problem of workload realism is still there.

KPI Aggregator

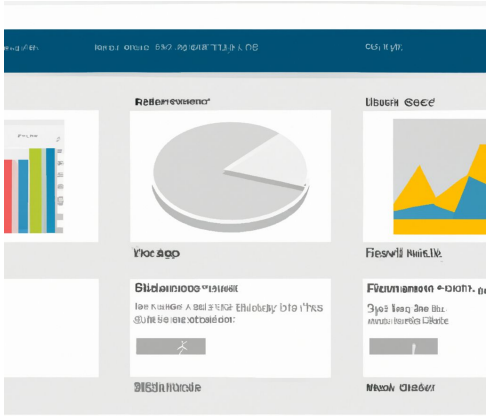


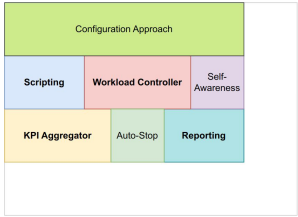
- Accumulates and processes results of individual requests
- Averages, percentiles and detailed statistics
- Takes some dedication to make it right
- Distributed tests require hyper-aggregator

Reporting



- Live feedback is crucial to save time
- Can be as subtle as exporter into DB
- Or as advanced as UI with dashboards and comparison
- The trend is to bring LT results into monitoring systems
- Static report is what many people need





Self-Awareness

- Detect self-overload
- Report measurement accuracy

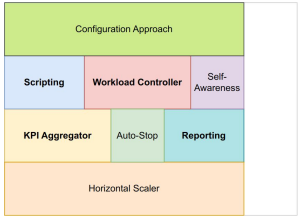
Can we trust results or not?

Auto-Stop

- Wrong test config or dead target
- Enough statistics accumulated
- Target is over the brink

Save time and resources

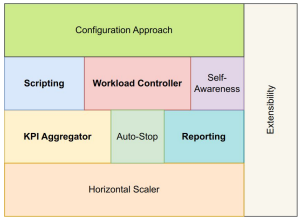
Horizontal Scaler



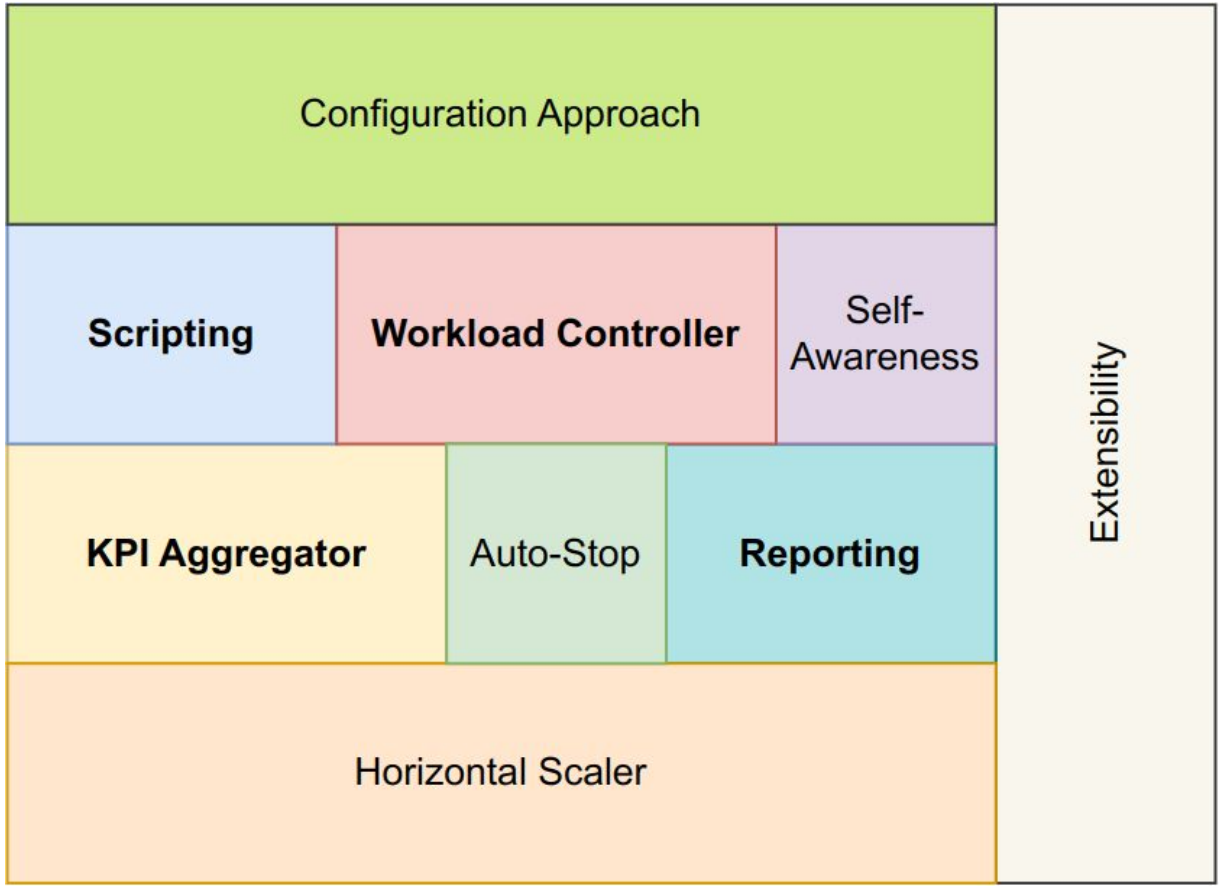
- Distributing the load across multiple generators
- Doing hyper-aggregation of results
- Control cloud provisioning and orchestration

Implementing it is a significant challenge.

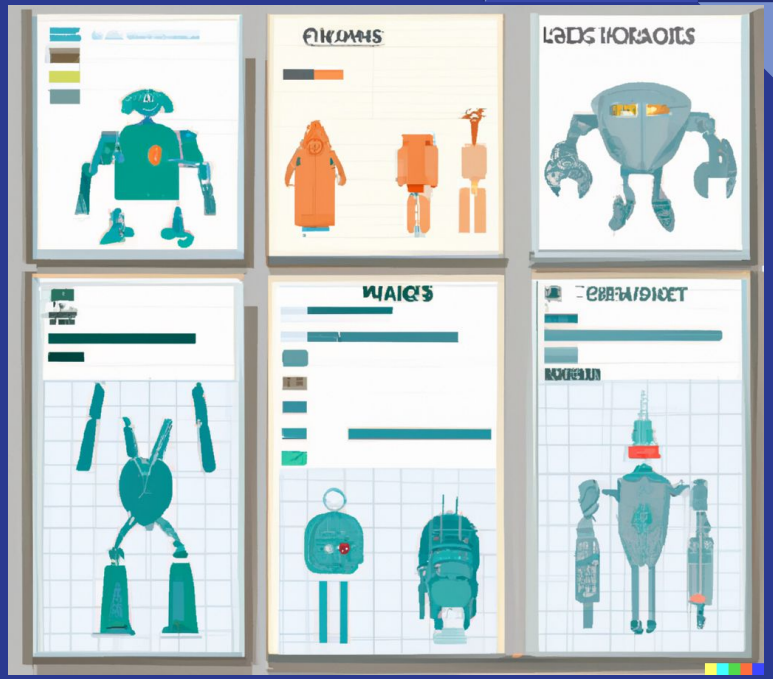
Extensibility



- Expand protocol coverage, load profile, reporting
- Turns the tool into potential ecosystem
- Very strategic feature for choosing a tool


Configuration Approach
Scripting
Workload Controller
Self-Awareness
KPI Aggregator
Auto-Stop
Reporting
Horizontal Scaler
Extensibility

Classification



Quick Benchmarks

- No scripting, one URL and go
- Usually with subtle reporting
- Popularity through simplicity

Examples: ab, wrk, hey, vegeta

Load Generators

- Can do all “quick benchmark” does
- Complex workload profiles and scripts
- Some with decent built-in reports
- More difficult to master

Examples: JMeter, Locust, Gatling, k6, LoadRunner, drill, encarno

Distributed Scaling Controllers

- Take load generator and run it on multiple machines, maybe in the cloud.
- Collect and aggregate results.

Examples:
beeswithmachineguns, artillery, JMeter, k6, LoadRunner, Gatling, Locust, Autometer and whole bunch of solutions on GH

Automation Frameworks

- Many tools claim they’re CI-friendly, but don’t offer much for that
- Features around configuration, export formats, automatic test shutdown
- Ability to write smaller load generator tools and reuse reporting/configuration

Examples: Yandex.Tank and Taurus

Hosted Solutions

All-you-can-eat, off the web, cloud-provisioned and high scale

Examples: BlazeMeter and clones, NeoLoad, HP StormRunner, Gatling FrontRunner, WebLOAD, LoadComplete

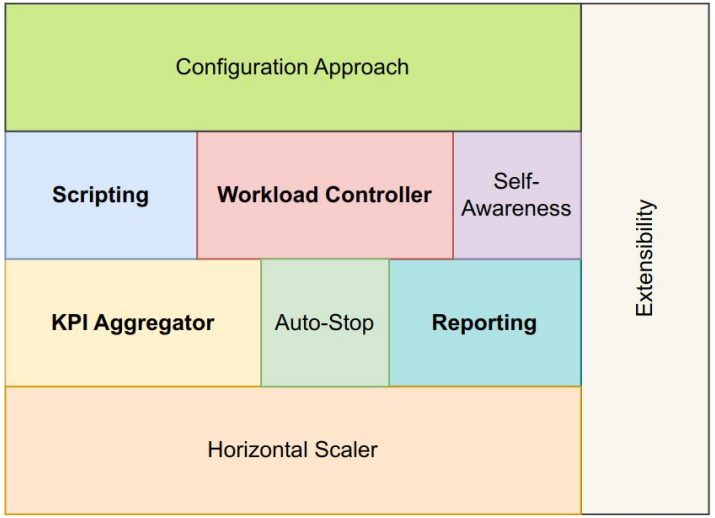
Code-integrated solutions (DSLs)

- Focused on staying inside IDE / familiar programming language
- Appeals to developers, shift-left & continuous testing
- Molotov, Gatling, Locust, k6 are close, but valueprop is different

Examples: JMeter Java DSL, NBomber

In Conclusion

- We learned LT anatomy
- We applied classification



- Choosing, using and creating own is easier and better

What's the next step of evolution?

Thank you!

andrei.pokhilko@gmail.com