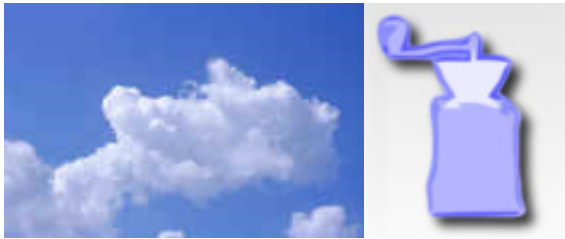


Grinder in the Cloud

Get Loaded!



Contents

| | |
|---|----|
| Contents | 2 |
| Changes | 3 |
| This Document..... | 3 |
| Intended Audience..... | 3 |
| Prerequisites..... | 3 |
| The Solution..... | 4 |
| Architectural Overview | 4 |
| Benefits..... | 6 |
| Costs | 6 |
| Tutorial: Putting your test in the cloud..... | 6 |
| Best Practice..... | 17 |
| Parameter lists | 17 |
| Distribute userdata over agents | 17 |
| Log properties | 18 |
| Support | 19 |
| Conclusion..... | 19 |



Changes

- 11/27/2009 Included link to current AMI
- 05/07/2009 More detailed monitoring by patching MRTG
- 04/30/2009 Agent monitoring with MRTG/snmp
- 04/14/2009 Added section on security group
Clarified RDP connection step
- 04/08/2009 Updated document to include Grinder 3.2 AMI

This Document

This document describes an Open Source based load test approach that uses cloud computing facilities offered by amazon web services (cf. <http://aws.amazon.com/>).

Intended Audience

The intended audience for this document consists of Test Managers, people responsible for test infrastructure and tool selection, System Administrators and people responsible for the development and execution of load tests.

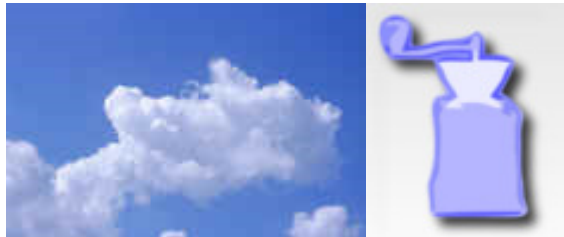
Prerequisites

To understand the concepts and benefits of The Grinder In The Cloud load test approach you will need the following:

- Common sense

To use Grinder In The Cloud load test approach you will need the following:

- Amazon Web Services account
- Basic knowledge of Amazon Web Services, especially ec2



- Basic knowledge of The Grinder

The Solution

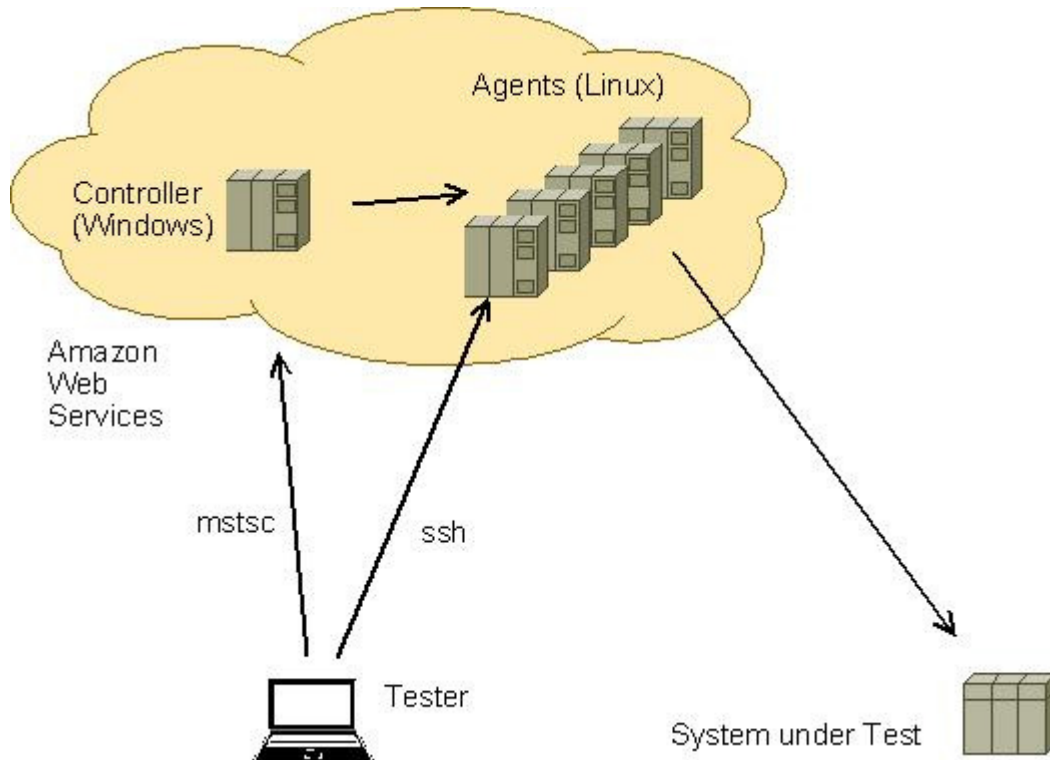
The solution is built upon the open source load test framework The Grinder (cf. <http://grinder.sourceforge.net/>). The Grinder is – next to Apache JMeter – one of the most popular open source load tools.

As most serious load tools The Grinder implements an Agent/Controller Architecture. Within the Grinder framework the Controller is called “Console”. The Console process coordinates several Agent processes. Each Agent process controls several Worker processes. In order to generate load Console and Agent processes run on different machines in the network. The number of Agent processes – and thereby the number of machines necessary to run the tests – increases with the amount of load specified for the tests.

Console and Agent processes are implemented as Amazon Machine Images (cf. <http://aws.amazon.com/ec2/#os>). The Grinder has been tweaked to allow for the transparent start up of as many agents as necessary. On start up the Agents connect automatically with the Console. The logs from the Agents are harvested automatically. The complexity of running load tests on a distributed cluster of agent machines is completely hidden from the user. The agents are monitored using MRTG (cf. <http://oss.oetiker.ch/mrtg/>). After the agents become available a working MRTG configuration is built behind the scenes and becomes active. Agents are polled once every minute.

Architectural Overview

The overall Architecture of the solution is shown below.



The tester talks from his machine to an instance of the controller amazon machine image (AMI) containing all the necessary software. The controller is accessed using the remote desktop application (MSTSC). For more information on the Windows remote desktop application, please see:

<http://www.microsoft.com/windowsXp/using/mobility/getstarted/Remoteintro.mspx#EQG>. The controller instance starts up as many agents as requested. The agents are implemented as AMIs and live in the cloud. If required the agents can be accessed using commonly available secure shell (ssh) tools. The agents inject the load into the system under test (SuT). Results and Logs are reported and maintained on the controller. After the test results and logs are stored and analyzed on the tester's machine.



The tester needs to start up and stop the controller AMI. This can be done using free software (firefox plugins, HTML pages). The software lives in a browser and is operating system independent.

Important: the instances run on real hardware. Each instance consists of a real machine!

Benefits

This test approach has several benefits for you. To name but a few:

- Endless firepower at your fingertips
- Endless firepower at will
- Virtually no hardware required on your side
- No license fees
- Up and running in minutes
- Pay for real usage and not for future use

Costs

The architecture comes at a very competitive price:

Controller: 0,10 USD/h

Each agent: 0,085 USD/h

That's it!

Tutorial: Putting your test in the cloud

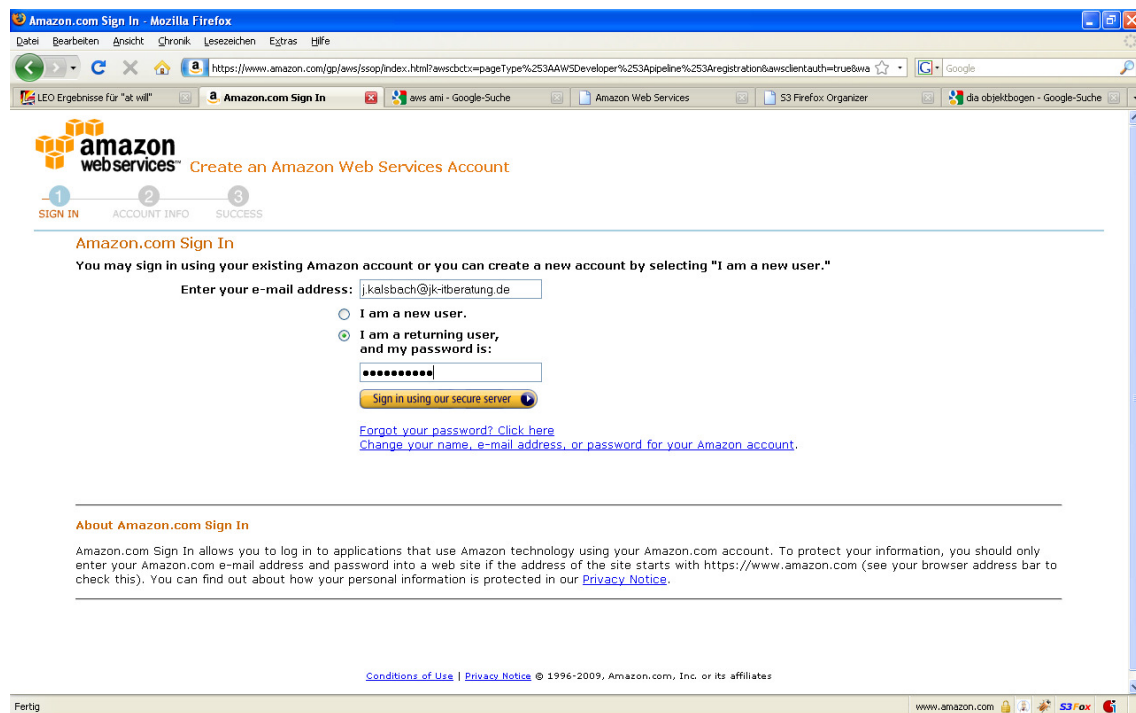
Seeing is believing! To give you a better idea a few screenshots demonstrate



the ease of use.

Putting a load test in the cloud requires the following steps:

Sign in at amazon.



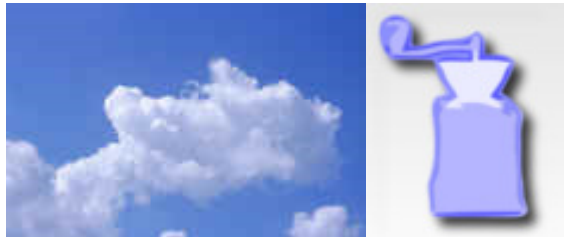
Create a security group named “All Incoming”. The security group needs to open tcp ports 22, 80, 3389, 6372 and udp port 161.

Use your favourite EC2 Client (e.g. Elasticfox

<http://developer.amazonwebservices.com/connect/entry.jspa?externalID=609>)

to launch an instance of the grinder_in_the_cloud AMI. You can always find the current AMI at





<http://developer.amazonwebservices.com/connect/entry.jspa?externalID=2055>
or by filtering the available AMIs by itbjk.

Launch new instance(s) ✕

AMI ID:

AMI Manifest:

AKI ID:

ARI ID:

Instance Type:

Minimum number of instances:

Maximum number of instances:

KeyPair:

Availability Zone:

Additional Info:

Security Groups

Available Groups:

Launch in:

User Data



Note: your security group needs to open ports 22, 80, 3389 and 6372 for tcp and 161 for udp.

Note: this windows AMI takes approx. 10' to boot. Work is ongoing to cut boot time.

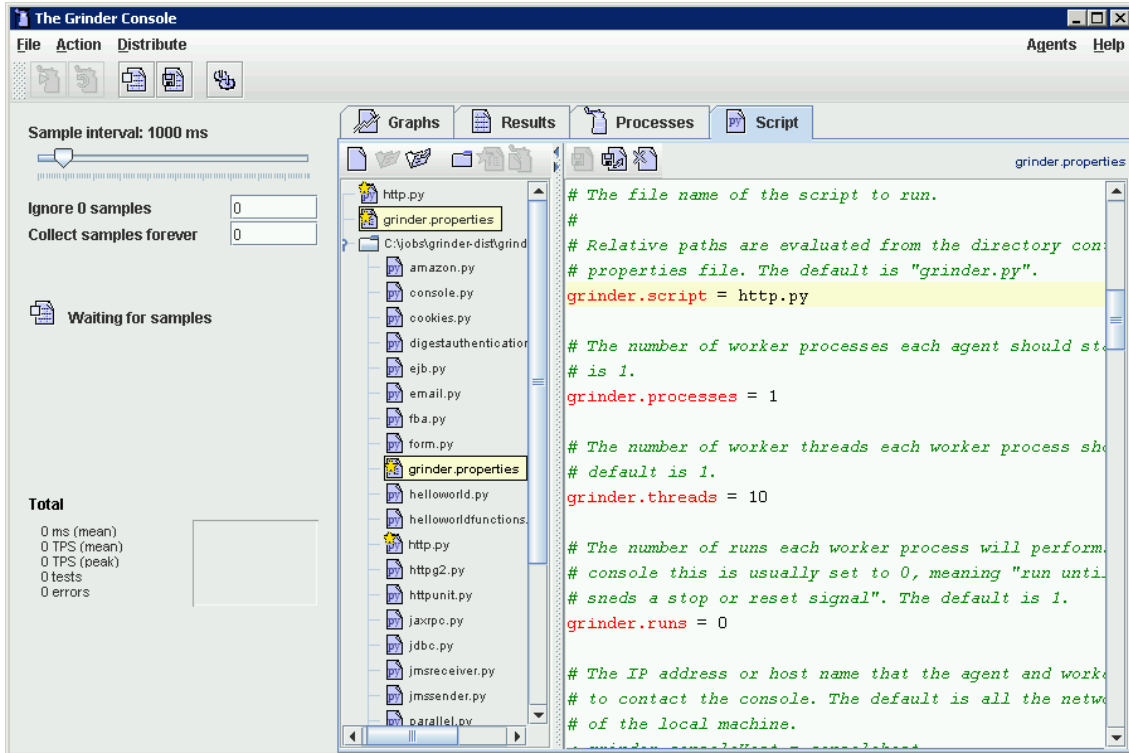
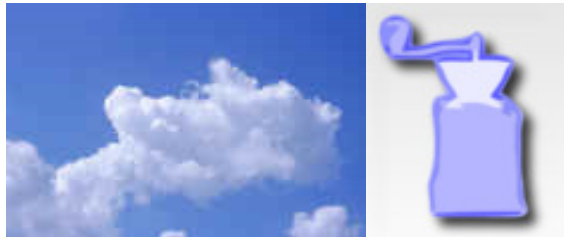
Once your controller instance is up and running copy the public DNS name and connect to the instance. Connect to instance using remote desktop (mstsc) with username: Administrator and password: NmD1M2czuV



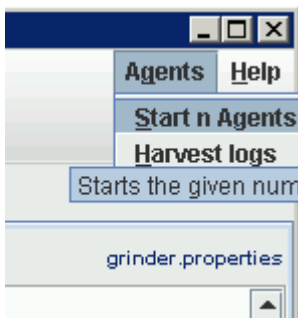
Note: The domain settings will be done automatically behind the scenes by mstsc.

Note: The Grinder starts automatically.

Deploy and configure your test scripts as usual (this demo uses standard example scripts).



Start up agents.



Enter number of instances and AWS Credentials.




| Start up Agent Instances | |
|--------------------------|------------|
| Number of Instances: | 3 |
| AWS Access Key: | Key |
| AWS Secret Access Key: | Secret Key |
| Ok | |
| Cancel | |

Wait until the agents connect to your controller (approx 5' as linux instances boot quite fast).

Note: agents will be shut down automatically on termination of the console process. If the console is terminated by ctrl-c it has no chance to shut down the agents. In this case you will have to shut down the agents manually.

Note: In the background MRTG will be configured to monitor your agents.

| Message | |
|---|---|
|  | Instances have been started. Please wait until they connect. See process tab. |
| OK | |

The instances connect to your controller.



Agents Help

Graphs Results Processes Script

Process status

| Process | Type | State |
|---------------------------|-------|--------------------|
| domU-12-31-39-02-60-C8 | Agent | Connected |
| domU-12-31-39-02-F5-D6 | Agent | Connected |
| 0 worker processes | | 0/0 threads |

Start your tests as usual. Run your test session as usual.

The Grinder Console Agents Help

File Action Distribute

Sample interval: 1000 ms

Ignore 0 samples

Collect samples forever

Collecting samples: 93

123 TPS

Total
 195 ms (mean)
 6.52 TPS (mean)
 124 TPS (peak)
 600 tests
 0 errors

Graphs Results Processes Script

Accumulated test statistics

| Test | Description | Successful Tests | Errors | Mean Time | Mean Time Standard Deviation | TPS | Peak TPS | Mean Response Length | Response Bytes Per Second | Response Errors | Mean time to resolve host | Mean time to establish connection | Mean time to first byte |
|--------------|-------------|------------------|----------|------------|------------------------------|-------------|------------|----------------------|---------------------------|-----------------|---------------------------|-----------------------------------|-------------------------|
| Test 1 | Requ... | 600 | 0 | 195 | 75.0 | 6.52 | 124 | 233 | 1520 | 0 | 10.6 | 92.9 | 194 |
| Total | | 600 | 0 | 195 | 75.0 | 6.52 | 124 | 233 | 1520 | 0 | 10.6 | 92.9 | 194 |

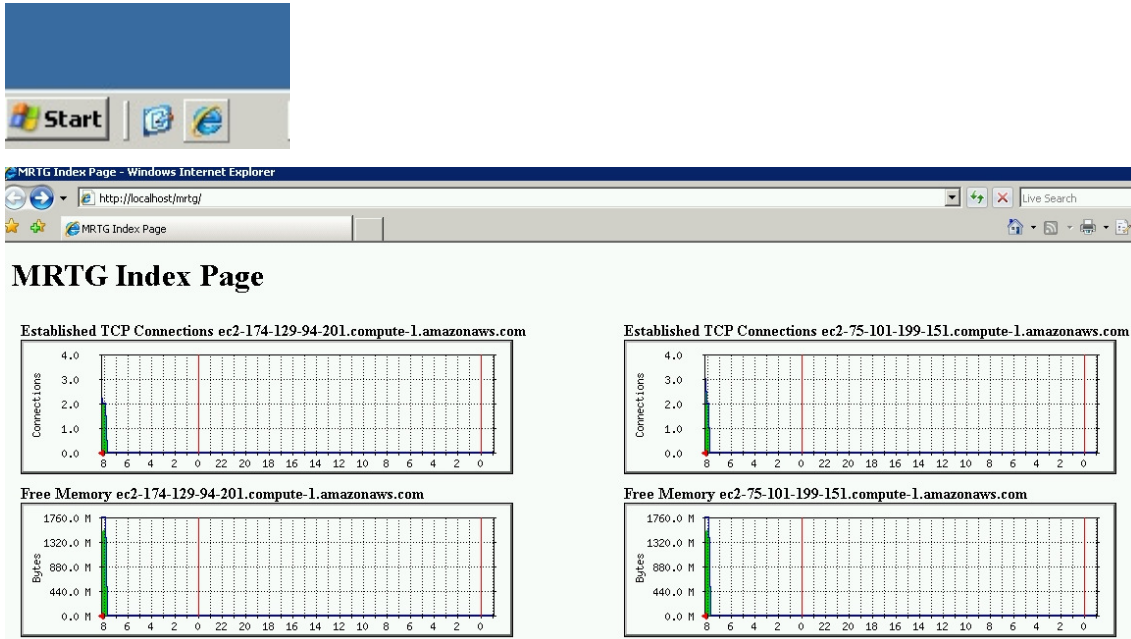
Latest sample

| Test | Description | Successful | Errors | Mean | Mean | TPS | Mean | Response | Response | Mean | Mean | Mean |
|------|-------------|------------|--------|------|------|-----|------|----------|----------|------|------|------|
| | | | | | | | | | | | | |





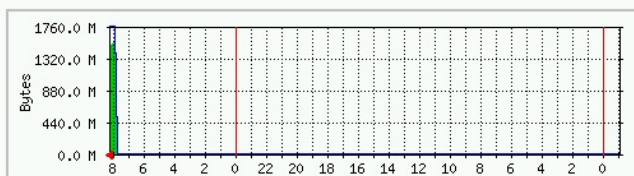
Monitor the health of your agents while they inject load into the SuT. Simply startup IE and study the MRTG charts:



Free Memory ec2-75-101-199-151.compute-1.amazonaws.com

The statistics were last updated Wednesday, 29 April 2009 at 8:23, at which time 'domU-12-31-38-01-D0-D2' had been up for 0:36:53.

'Daily' Graph (1 Minute Average)



| | Max | Average | Current |
|-------|-----------|-----------|-----------|
| Free | 1572.1 MB | 1436.4 MB | 1469.3 MB |
| Total | 1747.8 MB | 1645.4 MB | 1747.8 MB |



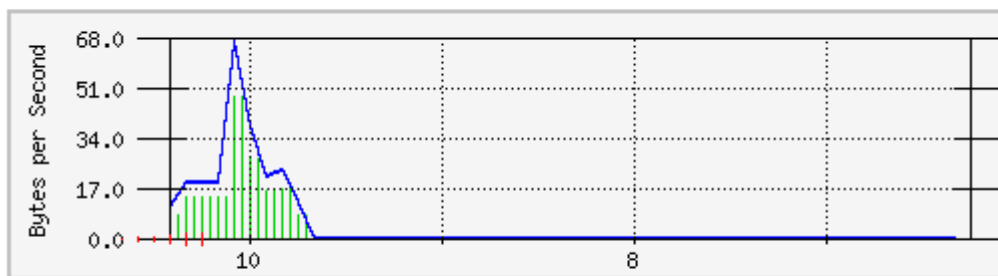
You can customize the MRTG configuration in case you feel the need to do so. Go to `c:\software\mrtg-2.16.2\bin\mrtg.cfg` and add your modifications. In order to add your SuT to the monitoring add your machine names to the list of agents (which you find in line 2 of `mrtg.cfg`) and execute “`cfg-make.bat listofagenthosts listofyourhosts`” in `c:\software\mrtg-2.16.2\bin\mrtg.cfg`. This will build a configuration containing your SuT hosts. Snmp must be running on your machines with `rocommunity public`. The UDP port 161 must be reachable from the internet which your security team will most probably prohibit.

In case you need more detailed monitoring you can copy the lines

```
$maxx=50;
$yscale=8;
```

in `c:\software\mrtg-2.16.2\bin\mrtg` after line 960. This is a workaround to zoom the monitoring interval to approx 4h:

'Daily' Graph (1 Minute Average)

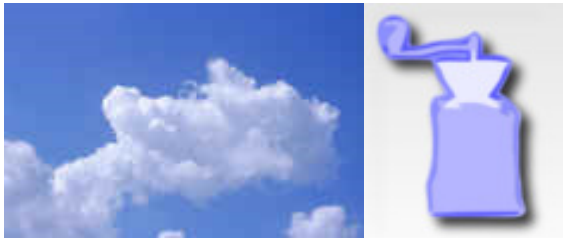


| | Max | Average | Current |
|-----|-----------------|-----------------|-----------------|
| In | 48.0 B/s (0.0%) | 18.0 B/s (0.0%) | 8.0 B/s (0.0%) |
| Out | 66.0 B/s (0.0%) | 25.0 B/s (0.0%) | 11.0 B/s (0.0%) |

The changed config will become active after the next polling interval.

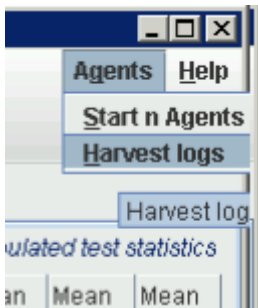
In case you need even more detailed monitoring you can pick any one agent



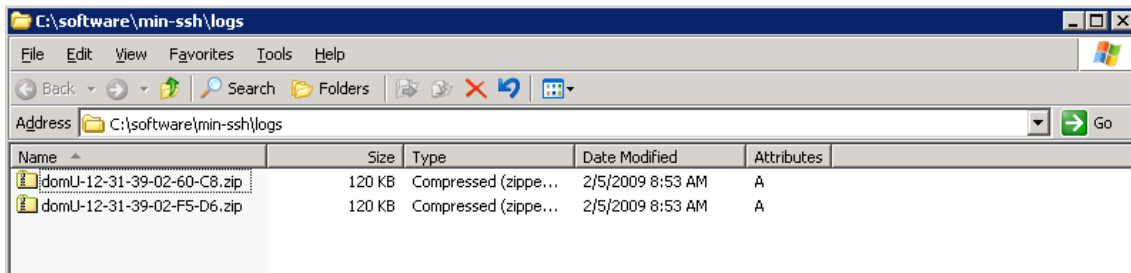


(assuming load being distributed more or less evenly), connect to the instance with something like “ssh -i itbjk.ppk root@yourinstance”. You’ll find itbjk.ppk in c:\software\min-ssh on the console machine. You can then use your favourite OS tool to monitor the agent.

Harvest results on the controller.



The results will show up in the explorer on the controller.



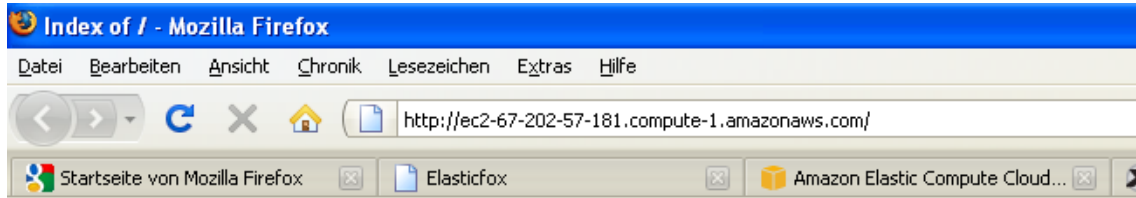
The result for each agent machine will be contained in a zip file named after the machine’s public Dns name.

Harvest results from the controller. Point your local browser at





http:// yourpublicdnsname.compute-1.amazonaws.com/



Index of /

- [domU-12-31-38-00-B4-33.zip](#)
- [domU-12-31-38-01-C4-04.zip](#)
- [domU-12-31-38-01-D1-81.zip](#)

Download your results for local postprocessing.



Enjoy! Destroy!



Best Practice

Parameter lists

Keep parameter lists as python objects and import them using the python module mechanism. Instead of keeping your user/login/whatever lists in files and bothering at runtime at the file location read the information into a python list, keep the list in a python file and import the python data.

Example:

In userdata.py:

```
usernames= ['Tic', 'Tac', 'Toe']
```

In testscript:

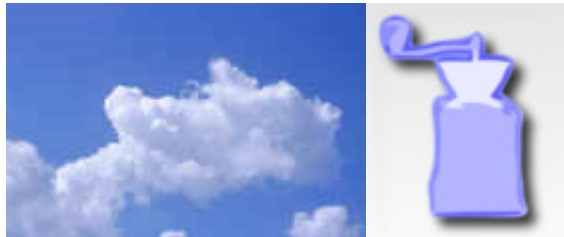
```
Import userdata  
firstUser=userdata.usernames[0]
```

Distribute userdata over agents

A common problem consists of a number of agents sharing a parameter list. In this situation each agent should process a distinct part of the parameter list.

Example:

Suppose three agents work on the userlist from the previous example.



In grinder.properties:

```
x.agents=3
```

In testscript:

Use code like the following

```
import userdata
idx= grinder.getAgentNumber()
allUserdata=userdata.usernames
props=grinder.getProperties()
chunk= len(UserData_T) / (props.getInt('x.agents',1))
idxa = chunk * idx
idxb = chunk * (idx +1)
print ('%s:%s' % (idxa, idxb))
userdata = userdata[idxa : idxb]
```

Log properties

Your properties file will most probably describe what you are doing. Log the contents of you properties file before actually doing your work. You can then easily see which configuration led to a given set of result files.

Use code like:

```
filename =
grinder.getFilenameFactory().createFilename(
    "grinder", "-%d.%s" % (time(), 'properties'))
prop = grinder.getProperties()
prop.store(FileOutputStream(filename), 'grinder.properties')
```



Support

In case you need further support, have questions, have feature requests, want to donate lots of money, want to submit bug reports, etc. contact j.kalsbach@jk-itberatung.de

In case you are interested in load test related consulting, consulting on test processes, test specification, scripting, outsourcing of parts of the test process or of the whole process please contact IT Beratung Jörg Kalsbach: j.kalsbach@jk-itberatung.de.

Conclusion

Grinder in the Cloud leverages the well known Grinder load test framework by putting it in the cloud. It offers an easy to use load test framework with virtually unlimited firepower at a competitive price.

Get loaded!