

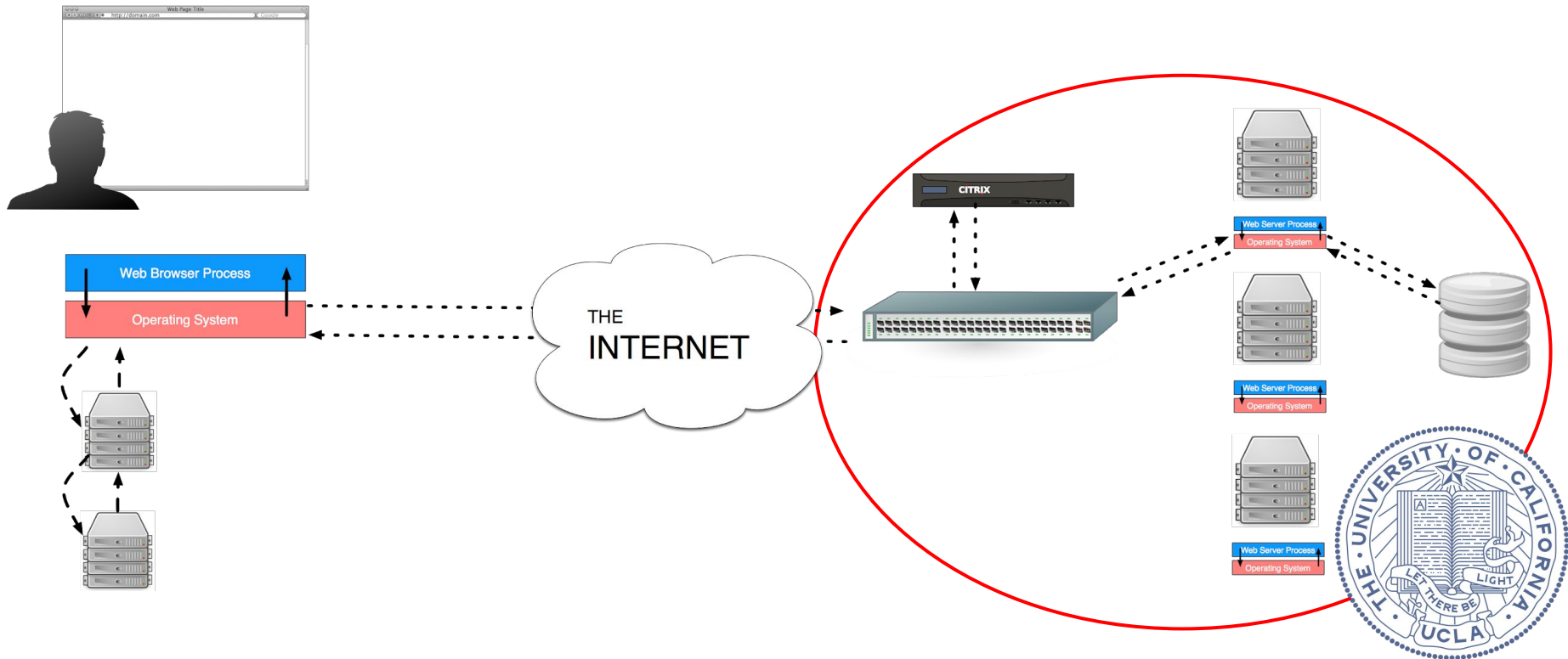
# CS 188

## Scalable Internet Services

Andrew Mutz  
October 24, 2019



# For Today



# Motivation

After today you will know how to evaluate the scalability of a deployed application using Tsung.

Today will be interactive, so if you've brought your laptops, please get them out.



# Motivation

Let's say we are considering a significant change to our web application.

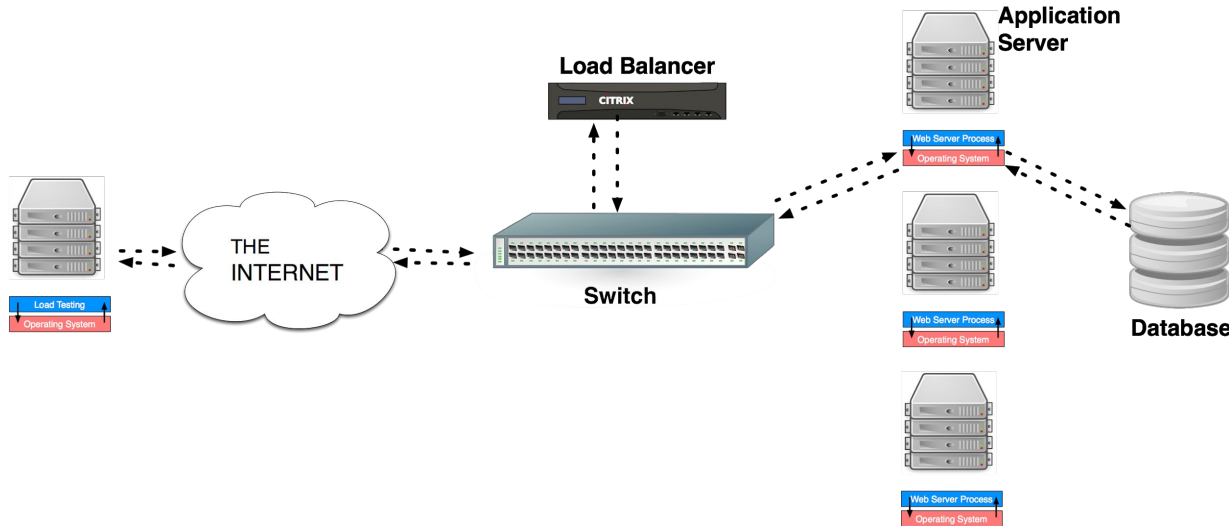
We think it may improve scalability, but our intuition might be wrong.

How should we go about testing this?



# Motivation

Let's deploy the system, send actual requests to it, and watch how it responds.



# Motivation

What should we observe?



# Motivation

## What should we observe?

- Response times
- Error rates
- Are our synthetic users able to finish their tasks?



# Motivation

## Some observations:

- We want a mixture of reads and writes
  - Why is this important?
- Not all users have the same habits
  - Why is this important?
- We want to be able to respond to application output.
  - When is this important?





# Motivation

Some load testing tools have high performance

- apachebench, httpperf

Some tools have rich feature sets

- Funkload

**Tsung is a good combination of the two**



# Tsung

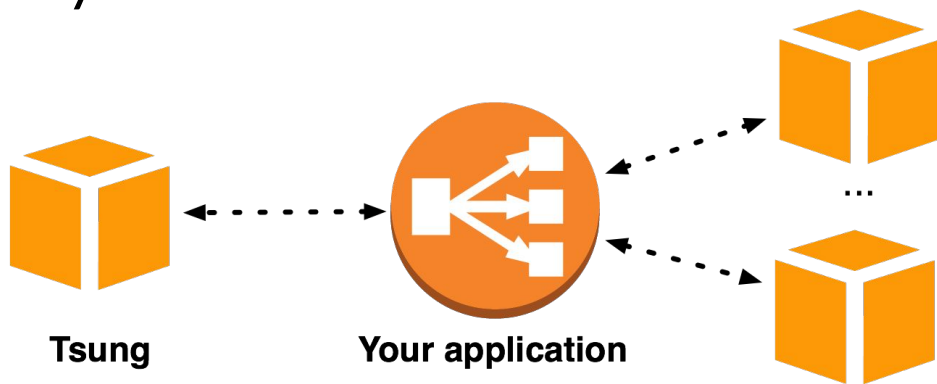
## Why Tsung?

- There are many load testing tools out there
- Most make you choose between flexibility and performance
- If you use a low-performance tool, you need to deploy a fleet of machines to do load testing
- Tsung is extremely configurable and delivers high performance



# Tsung

- We will do all load testing within AWS.
  - Saves money
  - More predictable
- Don't use T-series (t1 micro, etc) for measurement.
  - Why?



# Tsung

## We will use CloudFormation to deploy Tsung

The screenshot shows the AWS Management Console interface. At the top, the navigation bar includes the AWS logo, a 'Services' dropdown menu, 'Resource Groups', a search icon, a user profile 'andrew @ 2730-2014-7241', the region 'Oregon', and a 'Support' dropdown. Below the navigation bar is a search bar with the placeholder text 'Find a service by name or feature (for example, EC2, S3 or VM, storage)'. To the right of the search bar are two buttons: 'Group' and 'A-Z'. The main content area displays a grid of service categories and their respective services. The 'CloudFormation' service is highlighted with a red circle. The left sidebar contains a 'History' section with links to 'CloudFormation', 'Console Home', 'Support', 'EC2', 'VPC', and 'Elastic Beanstalk'.

**History**

- CloudFormation
- Console Home
- Support
- EC2
- VPC
- Elastic Beanstalk

**Services**

- Customer Enablement**
  - ECS
  - EKS
  - Lambda
  - Batch
  - Elastic Beanstalk
  - Serverless Application Repository
- Blockchain**
  - Amazon Managed Blockchain
- Storage**
  - S3
  - EFS
  - FSx
  - S3 Glacier
  - Storage Gateway
  - AWS Backup
- Database**
  - RDS
  - DynamoDB
  - ElastiCache
- Satellite**
  - Ground Station
- Management & Governance**
  - AWS Organizations
  - CloudWatch
  - AWS Auto Scaling
  - CloudFormation**
  - CloudTrail
  - Config
- Customer Enablement**
  - AWS IQ
  - Support
  - Managed Services
- Elasticsearch Service**
  - Kinesis
  - QuickSight
  - Data Pipeline
  - AWS Glue
  - AWS Lake Formation
  - MSK
- Security, Identity, & Compliance**
  - IAM
  - Resource Access Manager
  - Cognito
  - Secrets Manager
  - GuardDuty
  - Inspector
  - Amazon Macie
  - AWS Single Sign-On
  - Certificate Manager
  - Key Management Service
  - CloudHSM
- End User Computing**
  - WorkSpaces
  - AppStream 2.0
  - WorkDocs
  - WorkLink
- Internet Of Things**
  - IoT Core
  - Amazon FreeRTOS
  - IoT 1-Click
  - IoT Analytics
  - IoT Device Defender
  - IoT Device Management
  - IoT Events
  - IoT Greengrass
  - IoT SiteWise
  - IoT Things Graph

# Tsung

Create a new stack.

For the S3 URL use:

`https://ucla-cs188-fall-2019.s3.amazonaws.com/Tsung.json`

CloudFormation > Stacks > Create stack

Step 1  
Specify template

Step 2  
Specify stack details

Step 3  
Configure stack options

Step 4  
Review

## Create stack

### Prerequisite - Prepare template

#### Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

☒ Template is ready

☐ Use a sample template

☐ Create template in Designer

### Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

#### Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

☒ Amazon S3 URL

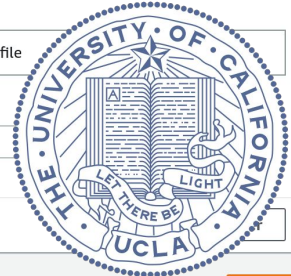
☐ Upload a template file

Amazon S3 URL

`https://ucla-cs188-fall-2019.s3.amazonaws.com/Tsung.json`

Amazon S3 template URL

S3 URL: `https://ucla-cs188-fall-2019.s3.amazonaws.com/Tsung.json`



Cancel

Next

# Tsung

For step 3 and 4, no need to modify anything, just click next and Create Stack.

It will take a few minutes for the stack to come up

The screenshot displays the AWS CloudFormation console for a stack named 'demo-john-tsung'. On the left, a list of stacks shows 'demo-john-tsung' as 'CREATE\_IN\_PROGRESS', 'demo-john' as 'CREATE\_COMPLETE', and 'demo-team' as 'ROLLBACK\_COMPLETE'. The main panel shows the 'demo-john-tsung' stack details with tabs for 'Stack info', 'Events', 'Resources', 'Outputs', 'Parameters', and 'Template'. The 'Events' tab is selected, showing a table of events. The first event, with a timestamp of '2019-10-23 21:31:51 UTC-0700' and logical ID 'demo-john-tsung', has a status of 'CREATE\_IN\_PROGRESS' (circled in red) and a status reason of 'User Initiated'.

CloudFormation > Stacks > demo-john-tsung

**demo-john-tsung**

Stack actions: Delete, Update, Stack actions, Create stack

Stack info | **Events** | Resources | Outputs | Parameters | Template

Change sets

**Events**

Search events

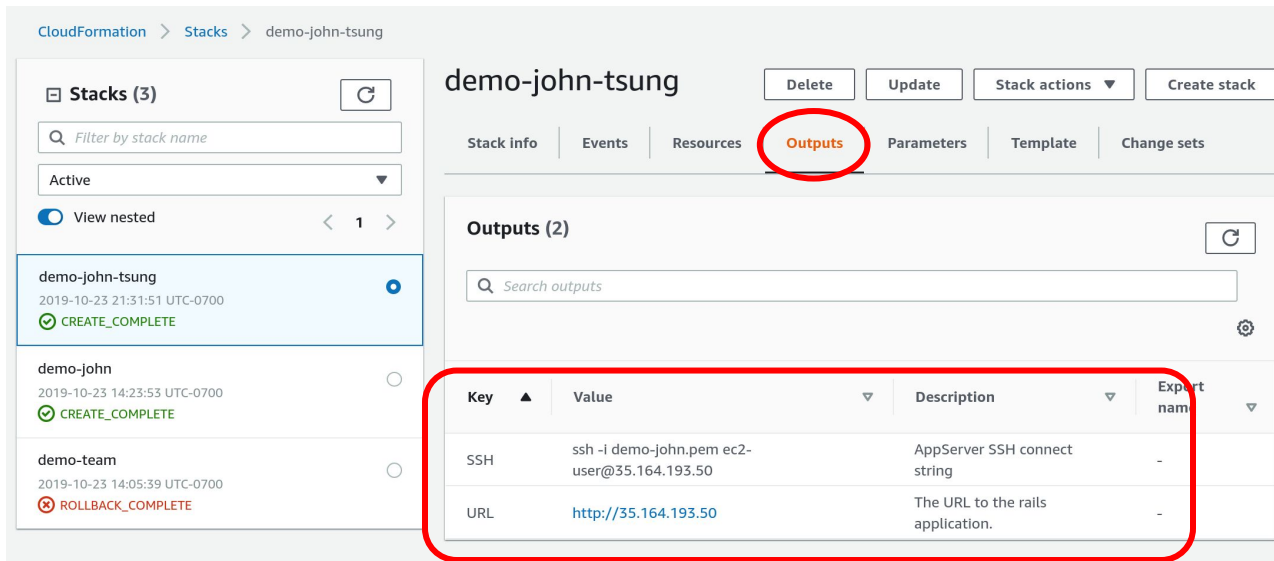
Timestamp	Logical ID	Status	Status reason
2019-10-23 21:31:51 UTC-0700	demo-john-tsung	<b>CREATE_IN_PROGRESS</b>	User Initiated

# Tsung

When its up, click on outputs.

The SSH info is how you will connect to the Tsung instance

The URL is how you will monitor load tests when they are in progress



CloudFormation > Stacks > demo-john-tsung

demo-john-tsung

Stack Info | Events | Resources | **Outputs** | Parameters | Template | Change sets

Outputs (2)

Key	Value	Description	Export name
SSH	ssh -i demo-john.pem ec2-user@35.164.193.50	AppServer SSH connect string	-
URL	<a href="http://35.164.193.50">http://35.164.193.50</a>	The URL to the rails application.	-



# Tsung

Let's SSH into our tsung instance. simple.xml is an example load testing script that comes pre-installed.

```
1: ec2-user@ip-172-31-23-73:~ ▾
andrew.alan.mutz@penguin:~/credentials$ ssh -i "demo-john.pem" ec2-user@ec2-52-40-132-169.us-west-2.compute.amazonaws.com
Last login: Thu Oct 24 04:38:24 2019 from 72.194.25.176

__|  __|_ )
_| (___/   Amazon Linux AMI
___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
23 package(s) needed for security, out of 58 available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2018.03 is available.
[ec2-user@ip-172-31-23-73 ~]$ ls -l
total 4
-rw-rw-r-- 1 ec2-user ec2-user 1055 Oct 24 02:11 simple.xml
lrwxrwxrwx 1 root      root      25 Oct 23 21:25 tsung_logs -> /home/ec2-user/.tsung/log
[ec2-user@ip-172-31-23-73 ~]$ █
```



# Tsung

Let's kick the tires by testing how [www.google.com](http://www.google.com) responds to light load:

```
tsung -f test.xml start
```

```
1: ec2-user@ip-172-31-23-73:~ ▾
andrew.alan.mutz@penguin:~/credentials$ ssh -i "demo-john.pem" ec2-user@ec2-52-40-132-169.us-west-2.compute.amazonaws.com
Last login: Thu Oct 24 04:38:24 2019 from 72.194.25.176

  _|_  _|_  )
 _|_ ( _|_ /  Amazon Linux AMI
__|_\___|___|

https://aws.amazon.com/amazon-linux-ami/2017.09-release-notes/
23 package(s) needed for security, out of 58 available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2018.03 is available.
[ec2-user@ip-172-31-23-73 ~]$ ls -l
total 4
-rw-rw-r-- 1 ec2-user ec2-user 1055 Oct 24 02:11 simple.xml
lrwxrwxrwx 1 root     root       25 Oct 23 21:25 tsung_logs -> /home/ec2-user/.tsung/log
[ec2-user@ip-172-31-23-73 ~]$ tsung -f simple.xml start
Starting Tsung
Log directory is: /home/ec2-user/.tsung/log/20191024-0442
[os_mon] memory supervisor port (memsup): Erlang has closed
[os_mon] cpu supervisor port (cpu_sup): Erlang has closed
[ec2-user@ip-172-31-23-73 ~]$
```

# Tsung

While the report is running, you can go to the web interface (the link from the CF output) to monitor how it is going

Tsung Dashboard - 20191024-0442

Status Reports Graphs Logs Stop

Main statistics

Transactions

Network Throughput

Counters

Server monitoring

HTTP status

Response times

Throughput graphs

Simultaneous Users

Server monitoring

HTTP status

20191024-0443: Report and graphs generated in 0.24 sec

## Main Statistics

Name	highest 10sec mean	lowest 10sec mean	Highest Rate	Mean Rate	Mean	Count
connect	8.08 msec	7.71 msec	1 / sec	0.95 / sec	8.08 msec	16
page	53.01 msec	52.38 msec	1 / sec	0.95 / sec	52.69 msec	16
request	53.01 msec	52.38 msec	1 / sec	0.95 / sec	52.69 msec	16
session	54.92 msec	54.31 msec	1 / sec	0.95 / sec	54.86 msec	16

## Transactions Statistics

Name	highest 10sec mean	lowest 10sec mean	Highest Rate	Mean Rate	Mean	Count
------	--------------------	-------------------	--------------	-----------	------	-------

## Network Throughput

Name	Highest Rate	Total
size_rcv	92.88 Kbits/sec	290.11 KB
size_sent	1.09 Kbits/sec	3.42 KB

# Tsung

While the report is running, you can go to the web interface (the link from the CF output) to monitor how it is going

Tsung Dashboard - 20191024-0514

Status Reports Graphs Logs Stop

Main statistics

Transactions

Network Throughput

Counters

Server monitoring

HTTP status

Response times

Throughput graphs

Simultaneous Users

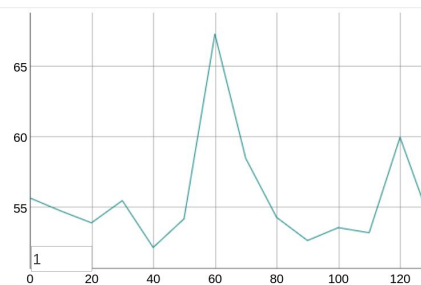
Server monitoring

HTTP status

20191024-0516: Report and graphs generated in 0.26 sec

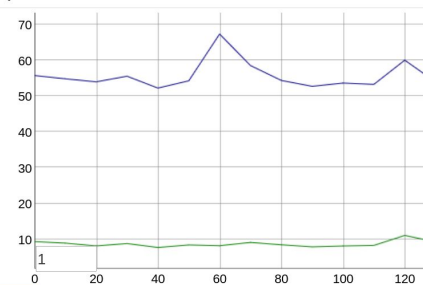
## Response Time

Transactions



Info »

Requests and connection establishment



Info »

## Throughput

Transactions



Requests



52.40.132.169/es/ts\_web.graph#

# Tsung

After you run a test you care about, make sure you copy the results to your workstation

```
scp -r -i "TEAM_NAME.pem"  
ec2-user@ec2-52-40-132-169.us-west-2.compute.am  
azonaws.com:tsung_logs ./tsung_logs
```



# Tsung

Your load testing scripts will be specific to your app and will be complex. It is best to keep them in your project repo.

Clone your repo to the load testing instance so you can get at your load testing scripts:

```
git clone git@github.com:scalableinternetservices/YOUR_TEAM.git
```



# Tsung - test.xml

Let's go over the basic xml file:

```
<?xml version="1.0"?><tsung loglevel="notice" version="1.0">
  <clients>
    <client host="localhost" use_controller_vm="true" maxusers="15000"/>
  </clients>
  <servers>
    <server host="www.google.com" port="80" type="tcp"/>
  </servers>
  <load>
    <arrivalphase phase="1" duration="10" unit="second">
      <users arrivalrate="1" unit="second"/>
    </arrivalphase>
  </load>
```



# Tsung - test.xml

## XML Boilerplate

```
<?xml version="1.0"?><tsung loglevel="notice" version="1.0">
  <clients>
    <client host="localhost" use_controller_vm="true" maxusers="15000"/>
  </clients>
  <servers>
    <server host="www.google.com" port="80" type="tcp"/>
  </servers>
  <load>
    <arrivalphase phase="1" duration="10" unit="second">
      <users arrivalrate="1" unit="second"/>
    </arrivalphase>
  </load>
```



# Tsung - test.xml

Client-side configuration. Maxusers is the maximum number of simulated users.

```
<?xml version="1.0"?><tsung loglevel="notice" version="1.0">
  <clients>
    <client host="localhost" use_controller_vm="true" maxusers="15000"/>
  </clients>
  <servers>
    <server host="www.google.com" port="80" type="tcp"/>
  </servers>
  <load>
    <arrivalphase phase="1" duration="10" unit="second">
      <users arrivalrate="1" unit="second"/>
    </arrivalphase>
  </load>
```





# Tsung - test.xml

Server configuration: where we are directing load.

```
<?xml version="1.0"?><tsung loglevel="notice" version="1.0">
  <clients>
    <client host="localhost" use_controller_vm="true" maxusers="15000"/>
  </clients>
  <servers>
    <server host="www.google.com" port="80" type="tcp"/>
  </servers>
  <load>
    <arrivalphase phase="1" duration="10" unit="second">
      <users arrivalrate="1" unit="second"/>
    </arrivalphase>
  </load>
```



# Tsung - test.xml

## Defining phases of user arrival rates

```
<?xml version="1.0"?><tsung loglevel="notice" version="1.0">
  <clients>
    <client host="localhost" use_controller_vm="true" maxusers="15000"/>
  </clients>
  <servers>
    <server host="www.google.com" port="80" type="tcp"/>
  </servers>
  <load>
    <arrivalphase phase="1" duration="10" unit="second">
      <users arrivalrate="1" unit="second"/>
    </arrivalphase>
  </load>
```



# Tsung - test.xml

A section to set options (timeouts, useragents)

```
<options>
  <option name="glocal_ack_timeout" value="2000"/>
  <option type="ts_http" name="user_agent">
    <user_agent probability="100">Mozilla/5.0 (Windows; U; Windows NT 5.2; fr-FR; rv:1.7.8) Gecko/20050511
    Firefox/1.0.4</user_agent>
  </option>
</options>

<sessions>
  <session name="http-example" probability="100" type="ts_http">
    <request>
      <http url="/" version="1.1" method="GET"/>
    </request>
  </session>
</sessions>
</tsung>
```



# Tsung - test.xml

We define the actual series of requests that a user will perform.

```
<options>
  <option name="glocal_ack_timeout" value="2000"/>
  <option type="ts_http" name="user_agent">
    <user_agent probability="100">Mozilla/5.0 (Windows; U; Windows NT 5.2; fr-FR; rv:1.7.8) Gecko/20050511
    Firefox/1.0.4</user_agent>
  </option>
</options>
<sessions>
  <session name="http-example" probability="100" type="ts_http">
    <request>
      <http url="/" version="1.1" method="GET"/>
    </request>
  </session>
</sessions>
</tsung>
```



# Tsung - test.xml

Lets change our tests to point to our server

```
<?xml version="1.0"?><tsung loglevel="notice" version="1.0">
  <clients>
    <client host="localhost" use_controller_vm="true" maxusers="15000"/>
  </clients>
  <servers>
    <server host="ec2-52-24-120-129.us-west-2.compute.amazonaws.com" port="80"
type="tcp"/>
  </servers>
  <load>
    <arrivalphase phase="1" duration="10" unit="second">
      <users arrivalrate="1" unit="second"/>
    </arrivalphase>
  </load>
```



# Tsung - test.xml

```
<load>
  <arrivalphase phase="1" duration="30" unit="second">
    <users arrivalrate="1" unit="second"></users>
  </arrivalphase>
  <arrivalphase phase="2" duration="30" unit="second">
    <users arrivalrate="2" unit="second"></users>
  </arrivalphase>
  <arrivalphase phase="3" duration="30" unit="second">
    <users arrivalrate="4" unit="second"></users>
  </arrivalphase>
  <arrivalphase phase="4" duration="30" unit="second">
    <users arrivalrate="8" unit="second"></users>
  </arrivalphase>
</load>
```

For examples of more complex load testing, see the demo app's `load_tests/critical.xml`

Increasing the rate of user creation over time.



# Tsung - test.xml

```
<load>
  <arrivalphase phase="1" duration="30" unit="second">
    <users arrivalrate="1" unit="second"></users>
  </arrivalphase>
  <arrivalphase phase="2" duration="30" unit="second">
    <users arrivalrate="2" unit="second"></users>
  </arrivalphase>
  <arrivalphase phase="3" duration="30" unit="second">
    <users arrivalrate="4" unit="second"></users>
  </arrivalphase>
  <arrivalphase phase="4" duration="30" unit="second">
    <users arrivalrate="8" unit="second"></users>
  </arrivalphase>
</load>
```

For examples of more complex load testing, see the demo app's `load_tests/critical.xml`

Increasing the rate of user creation over time.



# Tsung - test.xml

```
<sessions>  
  <session name="http-example" probability="100" type="ts_http">  
  
    <!-- start out at the dashboard. -->  
    <request>  
      <http url="/" version='1.1' method='GET'></http>  
    </request>
```

For examples of more complex load testing, see the demo app's `load_tests/critical.xml`

The session defines the virtual user that you will be simulating.

This one starts at “/”





# Tsung - test.xml

```
<!-- wait for up to 2 seconds, user is looking at posts -->  
<thinktime value="2" random="true"></thinktime>
```

For examples of more complex load testing, see the demo app's `load_tests/critical.xml`

Insert realistic random wait times in your simulations.



# Tsung - test.xml

```
<!-- create a random number to make a unique community name -->
<setdynvars sourcetype="random_number" start="1000"
end="999999">
  <var name="random_community_name" />
</setdynvars>

<!-- post to /communities to create a community.
remember the url of the created community so we can delete
it later -->
<request subst="true">
  <http
    url='/communities'
    version='1.1'
    method='POST'
    contents='community%5Bname%5D=community_name_%_random_community_name%_
&amp;commit=Create+Community'>
  </http>
</request>
```

For examples of more complex load testing, see the demo app's `load_tests/critical.xml`

Deal with uniqueness constraints by defining dynamic variables.

Insert dynamic variables by using `%%` syntax.



# Tsung - test.xml

```
<request subst="true">
  <dyn_variable name="created_submission_url" re="[Ll]ocation:
(http://.*)\r"/>
  <dyn_variable name="created_submission_id" re="[Ll]ocation:
http://.*/submissions/(.*)\r"/>
  <http
    url='/submissions'
    version='1.1'
    method='POST'

contents='submission%5Btitle%5D=link_%_random_submission_name%&su
bmission%5Burl%5D=http%3A%2F%2Fwww.article.com%2F%_random_submission_n
ame%&submission%5Bcommunity_id%5D=%_%created_community_id%&co
mmit=Create+Submission'>

  </http>
</request>
```

For examples of more complex load testing, see the demo app's `load_tests/critical.xml`

At times, you will need the output of one request to feed into the next.

Use dynamic variables for this



# Tsung - test.xml

```
<!-- Uncomment the following to debug print your dynamic variables
-->

<!--
<setdynvars sourcetype="eval" code="fun( {Pid, DynVars} ) ->
  io:format([126, $p, 126, $n, 126, $n], [DynVars]),
  ok end.">
  <var name="dump" />
</setdynvars>
-->
```

For examples of more complex load testing, see the demo app's `load_tests/critical.xml`

If you have difficulty with your dynamic variables, use this code to debug



# Tsung - test.xml

Name

submissions

51

application-13e64a74b7ab4dfdd9d96f8038945dd.css

application-f221e1ec5ab975eeadb83b9e995b0d3.js

bootstrap.min.css

bootstrap-theme.min.css

bootstrap.min.js

nr-632.min.js

33e249c12f7a=5272963&pl=1430718915465&v=632.2b17625to=JQ5dQUUMWwIVR...

favicon.ico

Headers

Preview

Response

Cookies

Timing

General

Remote Address: 52.24.120.128:80

Request URL: http://ec2-52-24-120-129.us-west-2.compute.amazonaws.com/submissions

Request Method: POST

Status Code: 302 Found

Response Headers

view source

Cache-Control: no-cache

Connection: keep-alive

Content-Length: 137

Content-Type: text/html; charset=utf-8

Date: Mon, 04 May 2015 05:55:15 GMT

Location: http://ec2-52-24-120-129.us-west-2.compute.amazonaws.com/submissions/51

Server: WEBrick/1.3.1 (Ruby/2.1.5/2014-11-13)

Set-Cookie: \_demo\_session=QULyenRocLVdC8lzb1hNMwRwXZNVd3008GdXZ5bEVnZGjhb0hjhZ4VepITy9hNG0wb0M2K24yUp0dn1WeEp0HFKd5LCTFm5nLLWGFPTVYVYjZ2aFLfC182NDJJMEJzanltYUJ3TDcyd090RwX0d0WraIordnZn2bEVu5Igx51R1WVJH0HwCv1JWHL1Mw4yU2NwHm5Dh0Xxwdu5Wk53WwFTbPpNXDEwFRWVpLd3p1LzVSUBw1JG1nQ3RJHd9Qk3wac92WmLeL72ZzR3Y1cxdlFwcGndwhLRE9XvAV2HCTEBzeFmFmBvOmPmVZVXbmZuZUxuZzdRdphWkxkZwFjWV12y9CSUxaZwC9P58tzc2PaHoyV3c4d3FCamul2dhwZadz09--3ec242e7eb210b1c8177fb2da1b421138de34146; path=/; HttpOnly

Set-Cookie: request\_method=POST; path=/

X-Content-Type-Options: nosniff

X-Frame-Options: SAMEORIGIN

X-Request-Id: 65906fbc-b0f7-4587-892c-1e8e7e7c814

X-Runtime: 0.829838

X-Xss-Protection: 1; mode=block

Request Headers

view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Accept-Encoding: gzip, deflate

Accept-Language: en-US,en;q=0.8

Cache-Control: max-age=0

Connection: keep-alive

Content-Length: 274

Content-Type: application/x-www-form-urlencoded

Cookie: \_demo\_session=mdy03Flh3ad0NjWgdMEF3Im5YkR-RFVHKZ4b0Rg0E84eUHQjVjM1BHa2RuRVgxtIEv1cTNJv2xLS1hZNR0TU0wQ1dqETfYkKhEdzFUS2M5JZYzAvkZolzRXb18ySmIjce9Y0W1Hb09UmwCM3VRTG10b8ZB0FmTE9NZRXvklZNd0pYm14amVeFyVehQ3PT0LXPZ5Z0zNEtKdXRG5lpIdESXOG5TR3c9PQ3d03D--61704676792975833c85bed79338dbde59645a3

Host: ec2-52-24-120-129.us-west-2.compute.amazonaws.com

For examples of more complex load testing, see the demo app's [load\\_tests/critical.xml](#)

If you are unsure how to simulate your application, use the browser to get a firm idea of the exact HTTP requests.



Form Data view source view decoded

utf8: %E2%9C%93

authenticity\_token: yQ87%2F0xfj%2BkNMJ5Lp6r0hs60BeEDI1FY25HeJrf0jL9j7QdVuSRPCJ8Q0Ii8ZD8x%2Fe46eIS9cZkQ0GHFGzQQQ%3D%3D

submission%5Btitle%5D: this+is+a+test

submission%5Burl%5D: http%3A%2F%2Fwww.testing.com

submission%5Bcommunity\_id%5D: 1

commit: Create+Submission

# Tsung - Output

## Main Statistics

Name	highest 10sec mean	lowest 10sec mean	Highest Rate	Mean	Count
connect	0.42 sec	9.26 msec	0.5 / sec	0.35 sec	6
page	0.57 sec	66.26 msec	0.4 / sec	0.40 sec	6
request	0.57 sec	66.26 msec	0.4 / sec	0.40 sec	6
session	0.48 sec	68.15 msec	0.5 / sec	0.41 sec	6

- **request** Response time for each request.
- **page** Response time for each set of requests (a page is a group of request not separated by a thinktime).
- **connect** Duration of the connection establishment.
- **session** Duration of a user's session.



# Tsung - Output

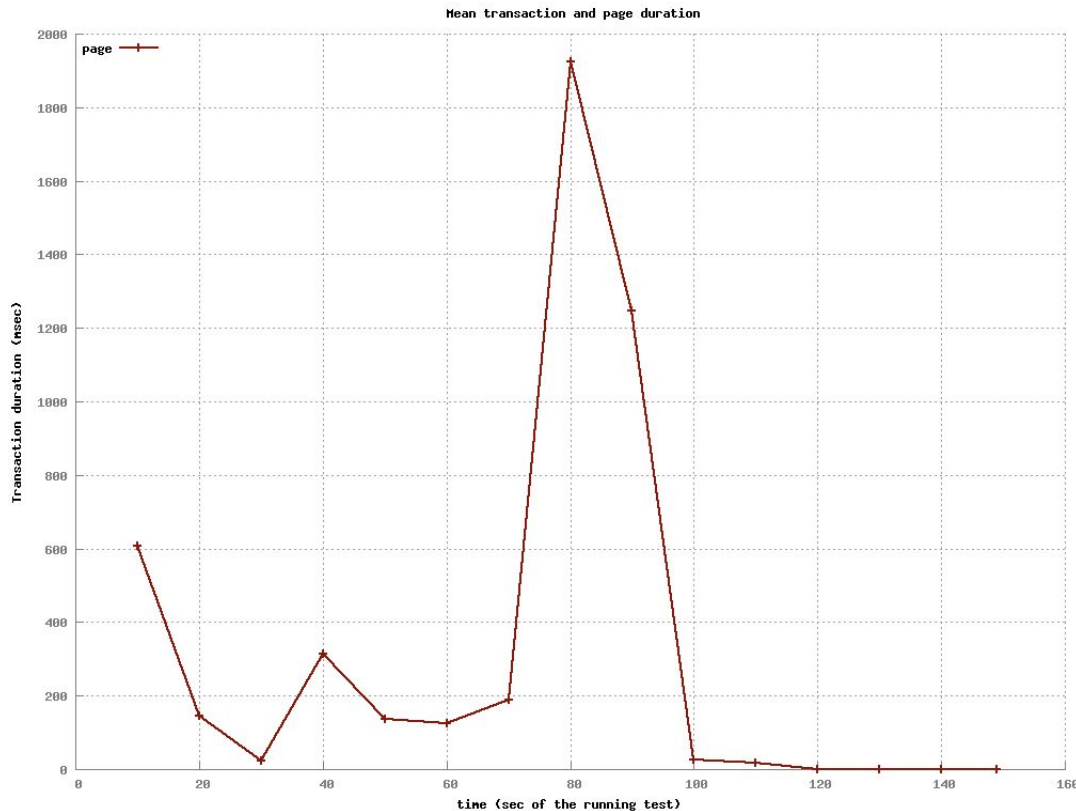
## HTTP return code

Code	Highest Rate	Total number
200	0.4 / sec	6

- Make sure you are getting back good status codes.
  - 200s and 300s are good
  - 400s and 500s are usually bad



# Tsung - Output

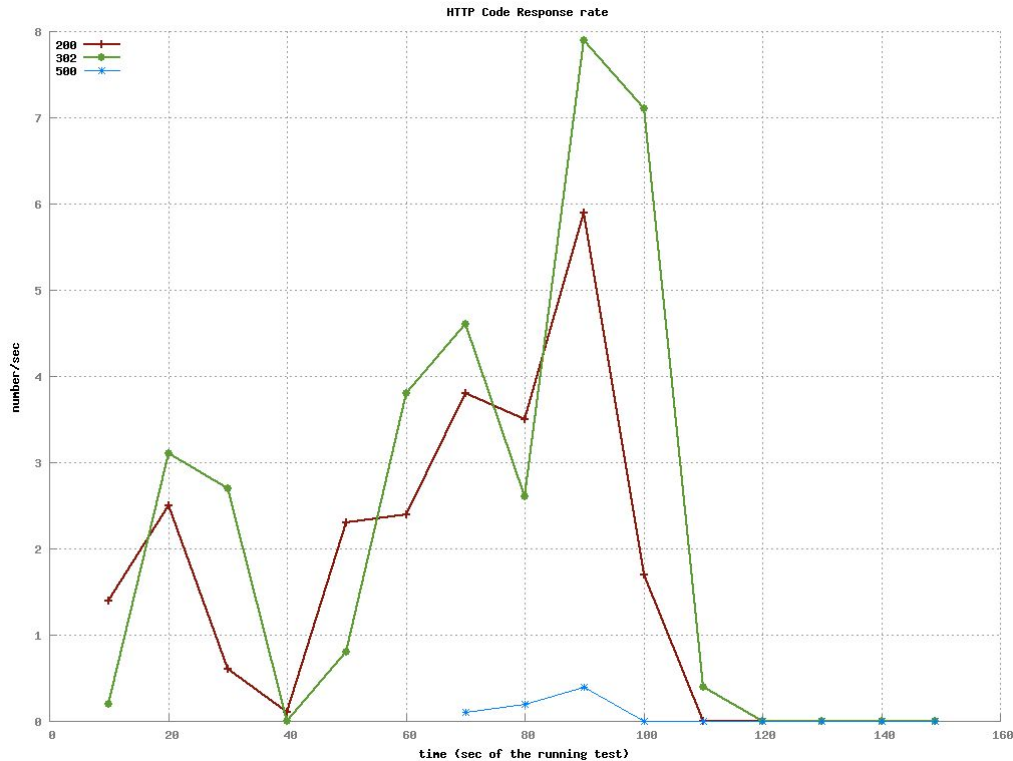


- Response times





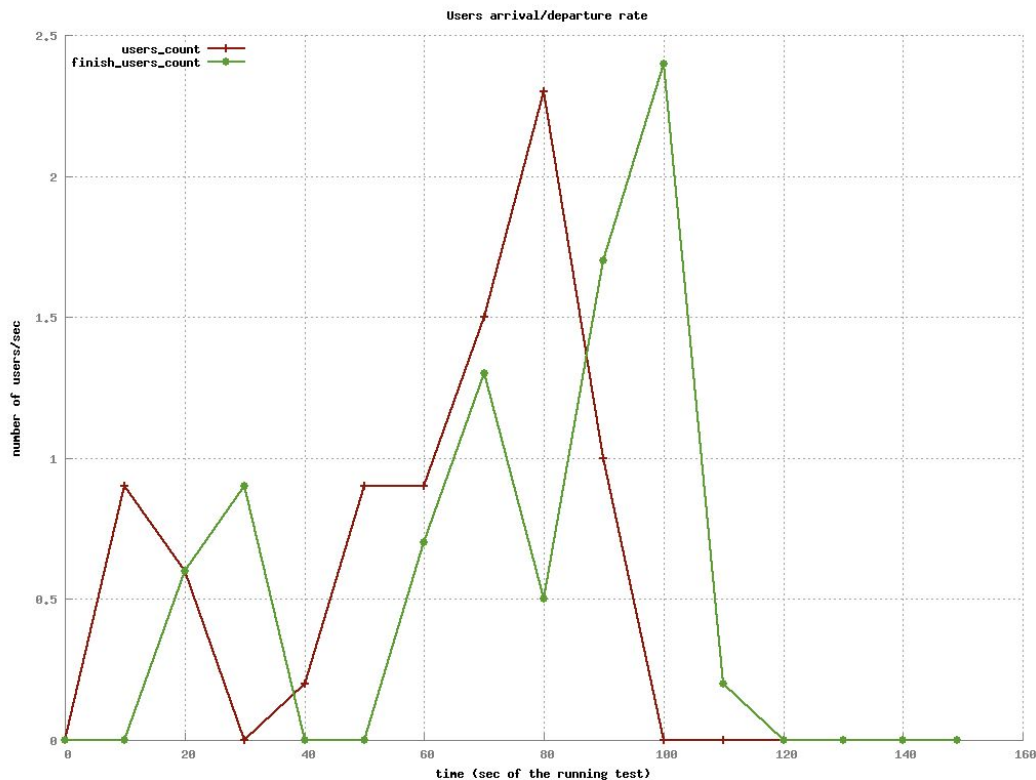
# Tsung - Output



- Error rates



# Tsung - Output



- Are our synthetic users able to finish their tasks?



# Tsung

Always remember that your Tsung instances will go down automatically (6 hours), so please scp off any important data or results immediately.



# For Next Time...

Attempt to create a simple load testing script for your current app

**Keep completing stories!**

