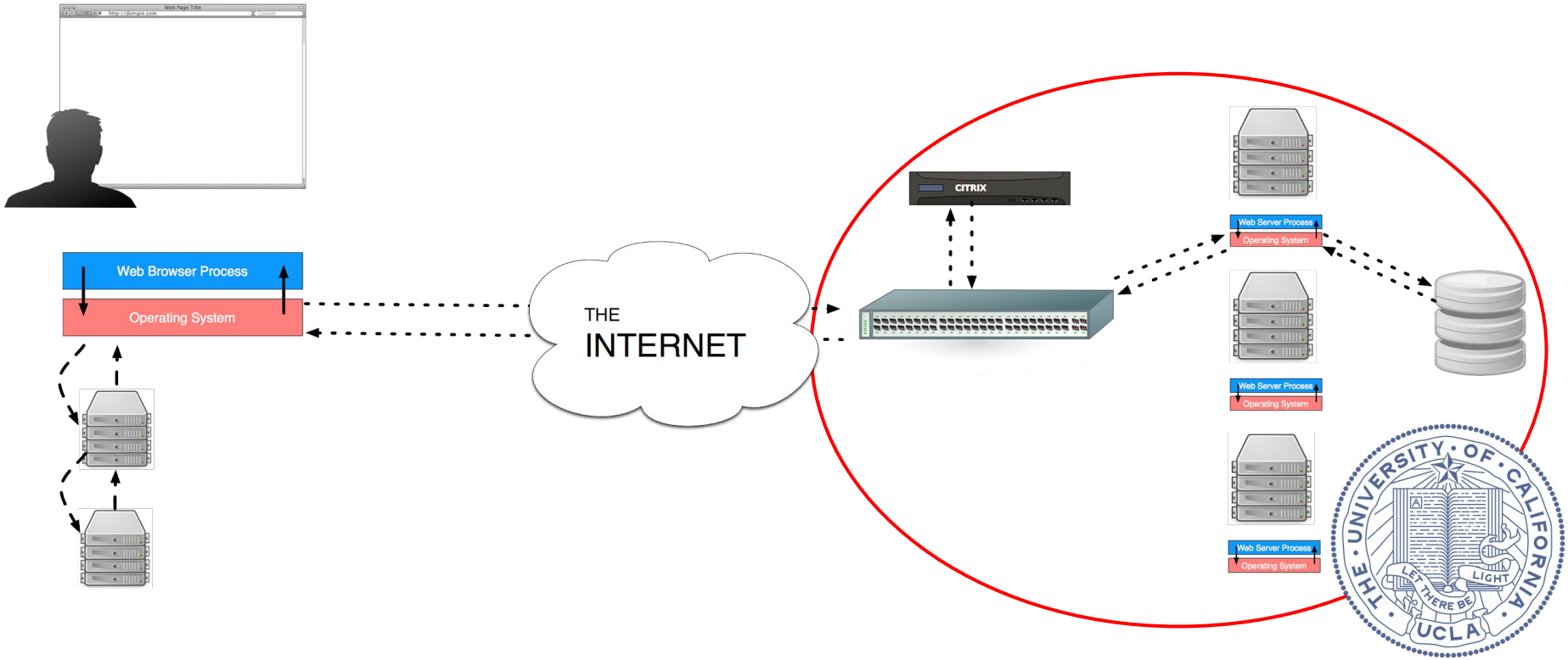
CS 188

Scalable Internet Services

Andrew Mutz

October 24, 2019

For Today


Web Page Title
http://domain.com
Google
Web Browser Process
Operating System
THE
INTERNET
CITRIX
Web Server Process
Operating System
Web Server Process
Operating System
Web Server Process
Operating System
THE UNIVERSITY OF CALIFORNIA
LET THERE BE LIGHT
UCLA

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

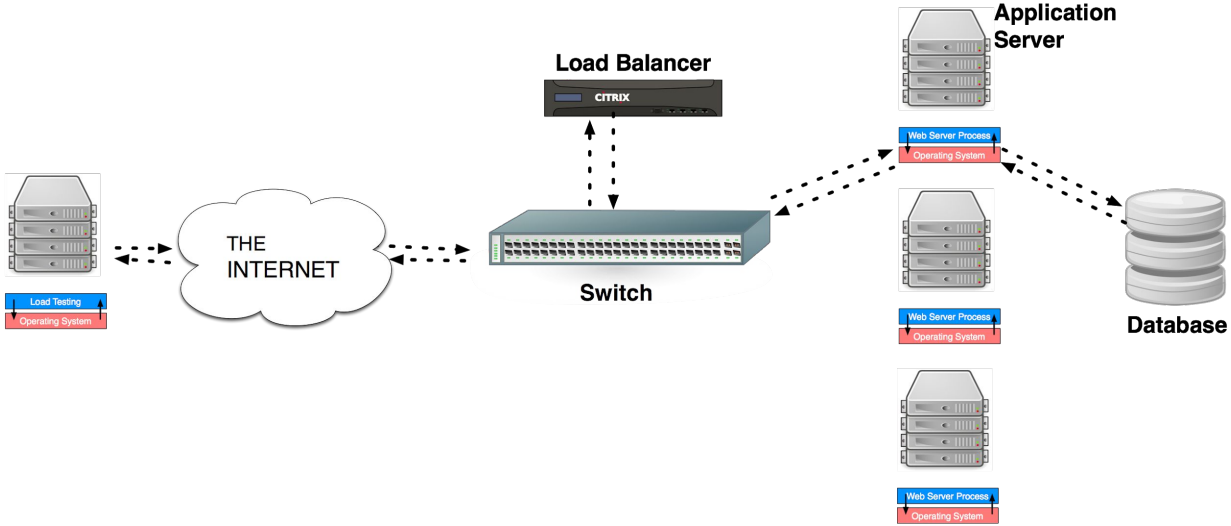
Let's say we are considering a significant change to our web application.

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, httperf

Some tools have rich feature sets

- Funkload

Tsung is a good combination of the two

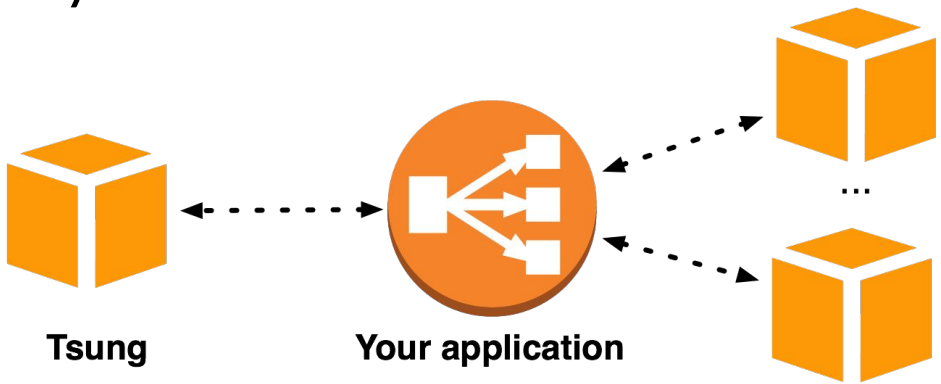
Tsung

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

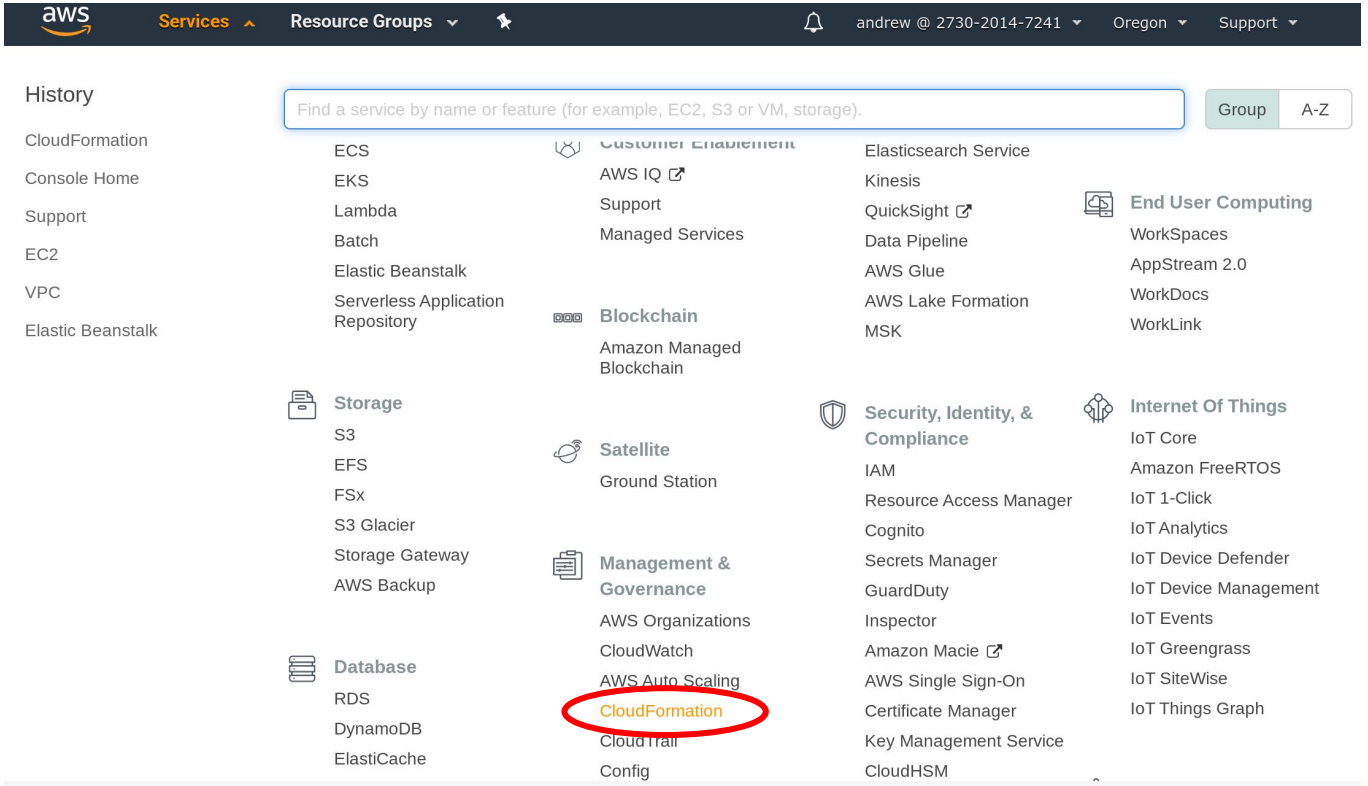
Tsung

- 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

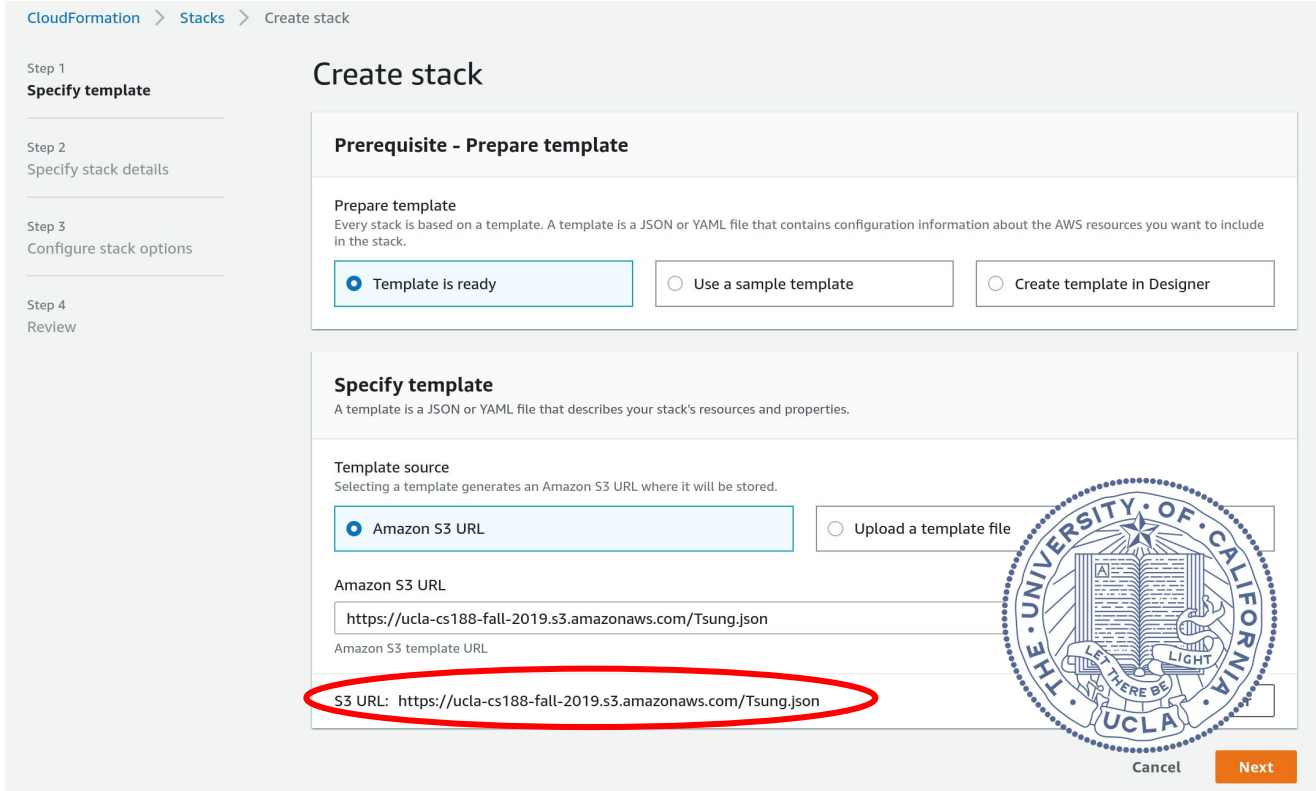


Tsung

Create a new stack.

For the S3 URL use:

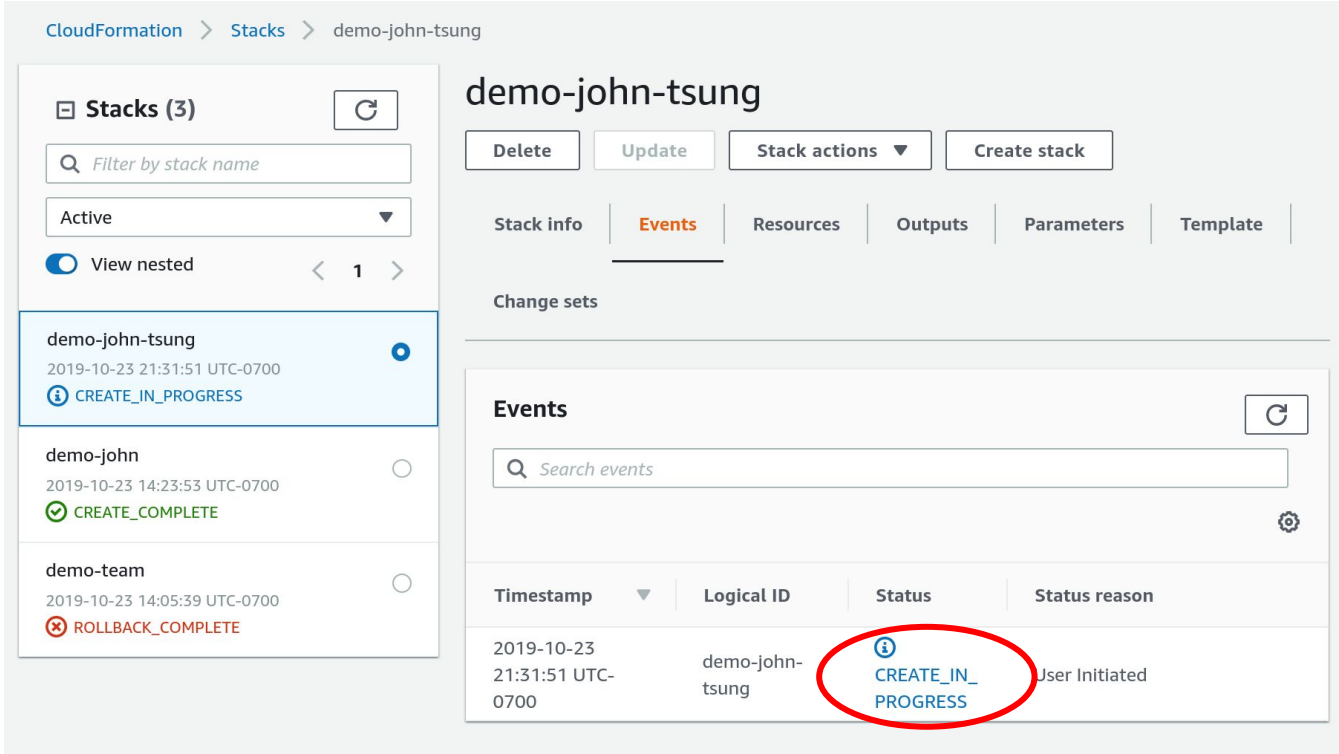
`https://ucla-cs188-fall-2019.s3.amazonaws.com/Tsung.json`



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

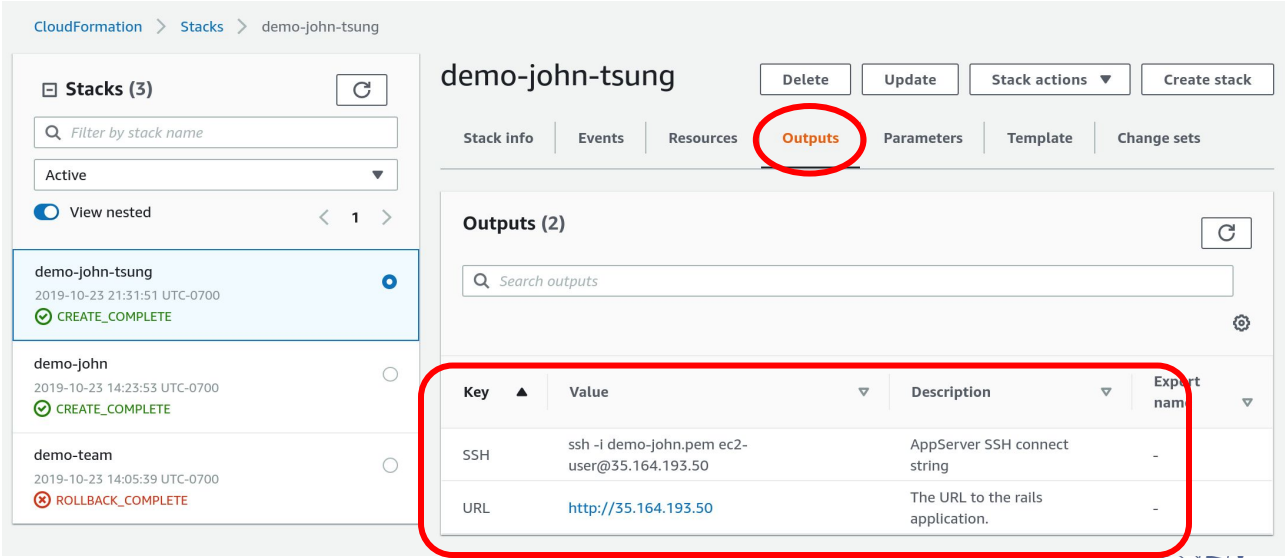


Tsung

When its up, click on outputs.

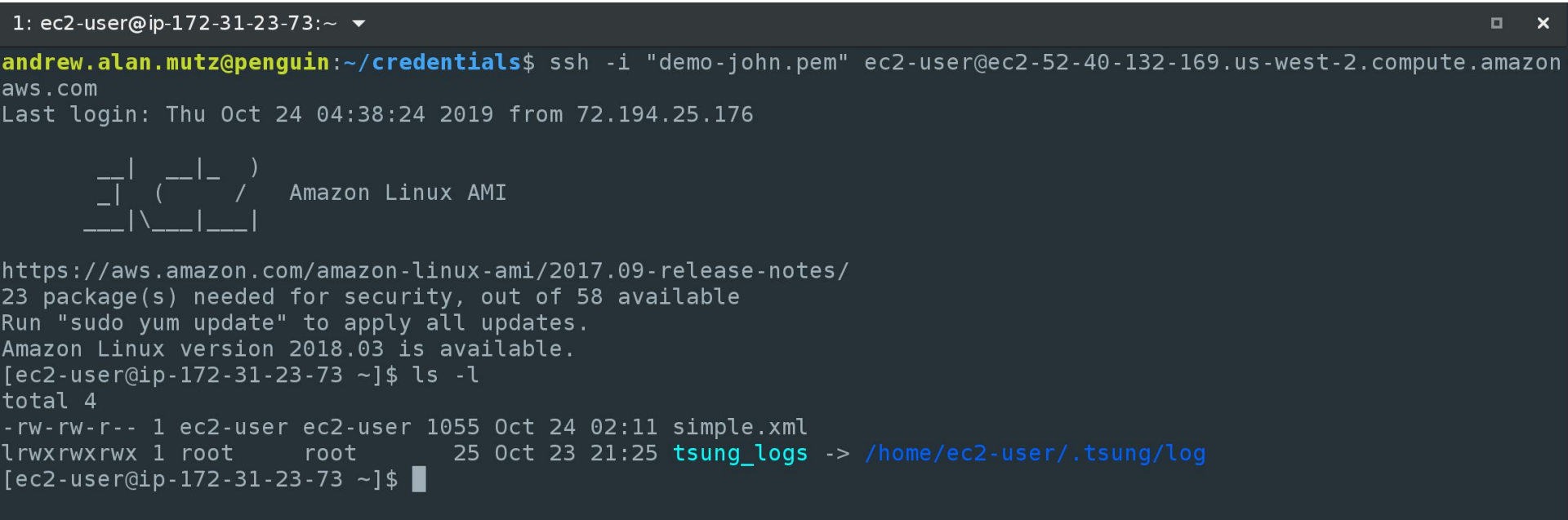
The SSH info is how you will connect to the Tsung instance

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



Tsung

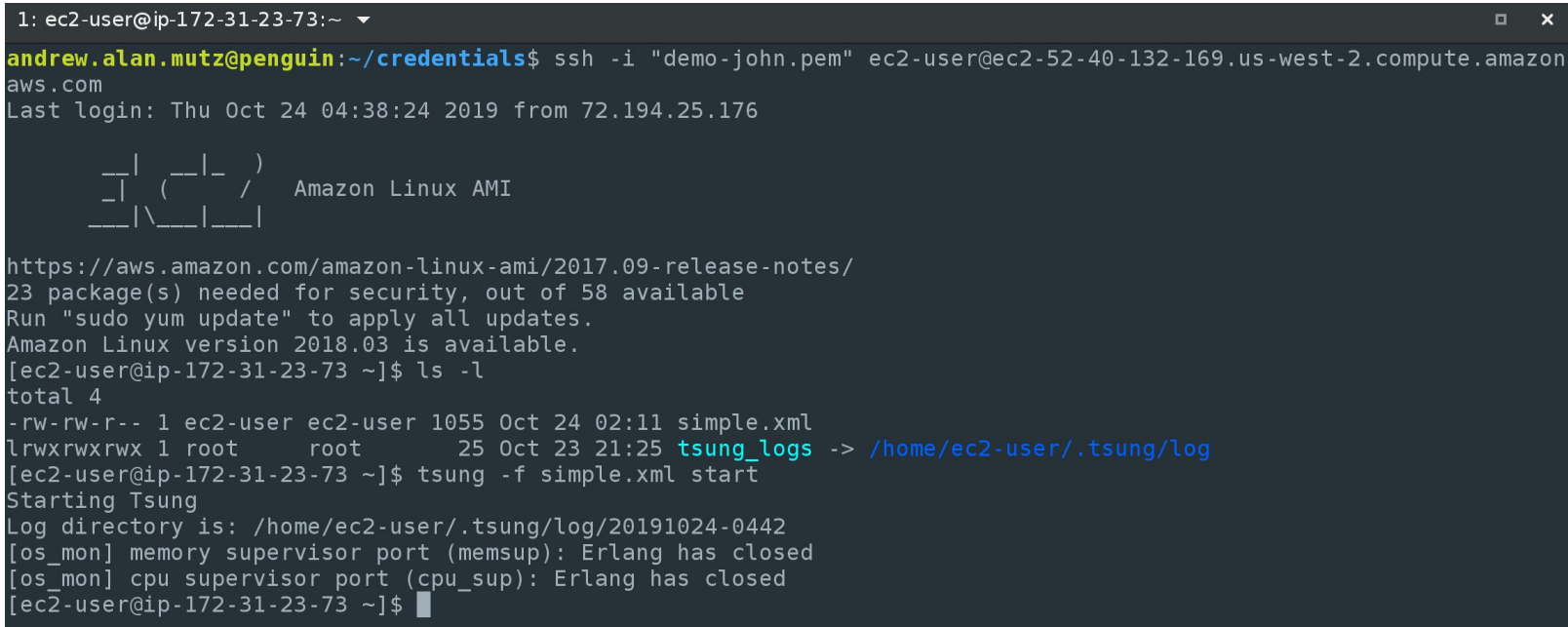
Let's SSH into our tsung instance. simple.xml is an example load testing script that comes pre-installed.



Tsung

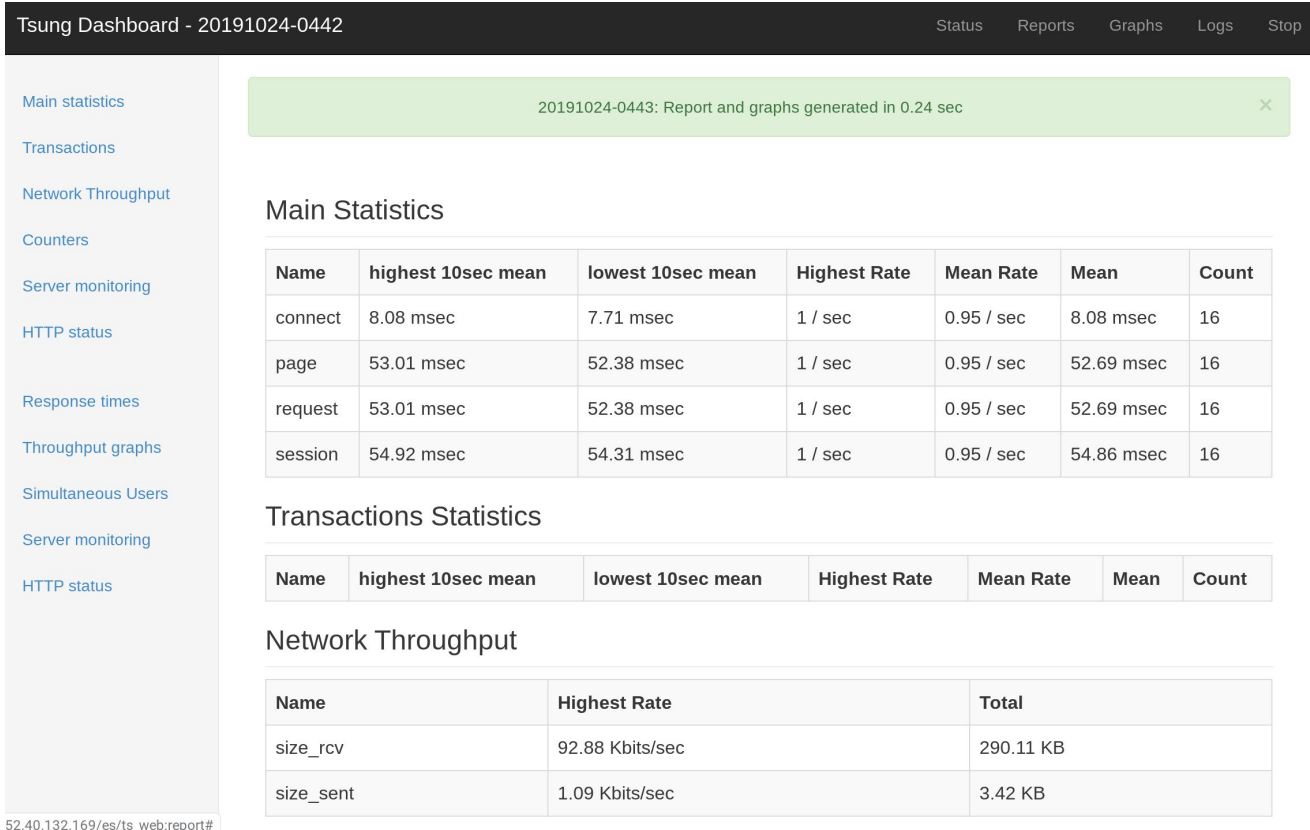
Let's kick the tires by testing how www.google.com responds to light load:

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



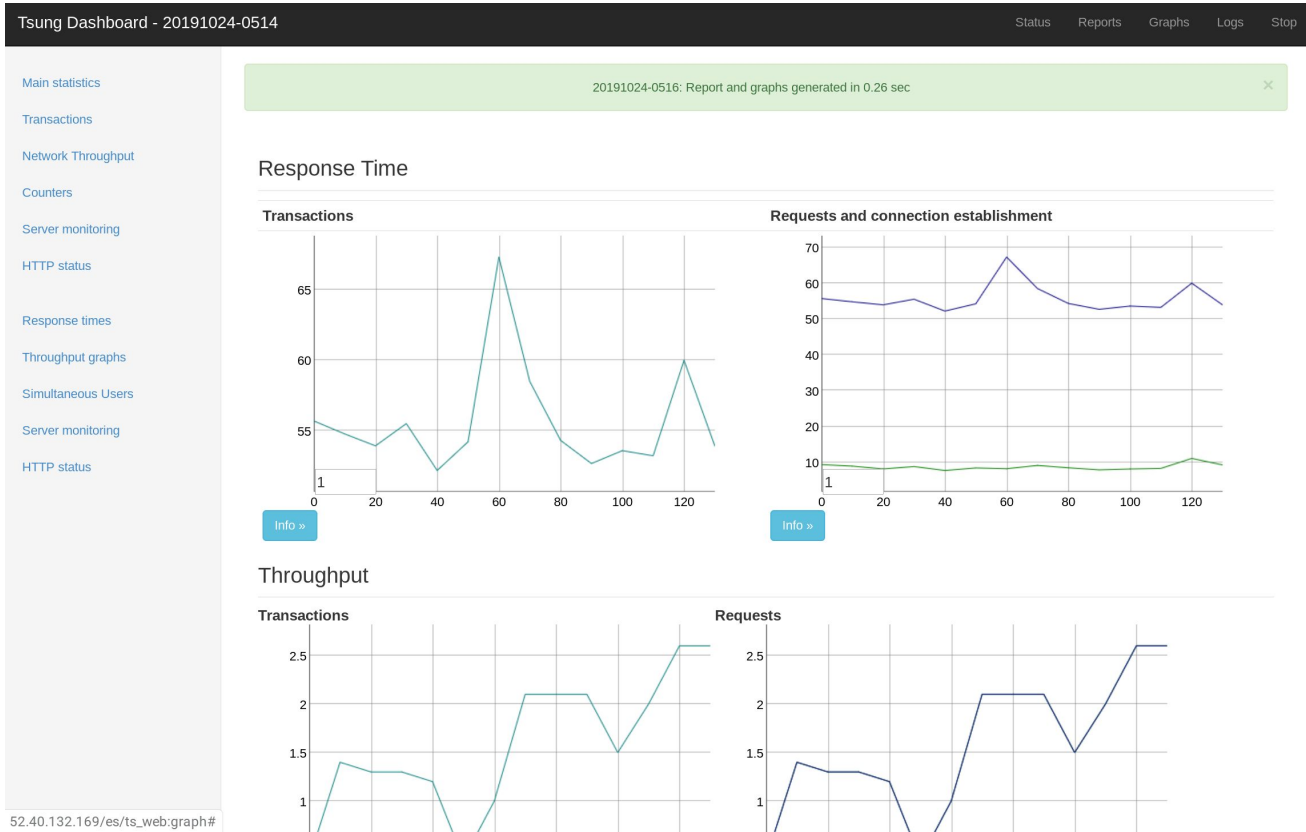
Tsung

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

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

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

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

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

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="9999999">
        <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%%&amp;su
bmission%5Burl%5D=http%3A%2F%2Fwww.article.com%2F%%_random_submission_n
ame%%&amp;submission%5Bcommunity_id%5D=%%_created_community_id%%&amp;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

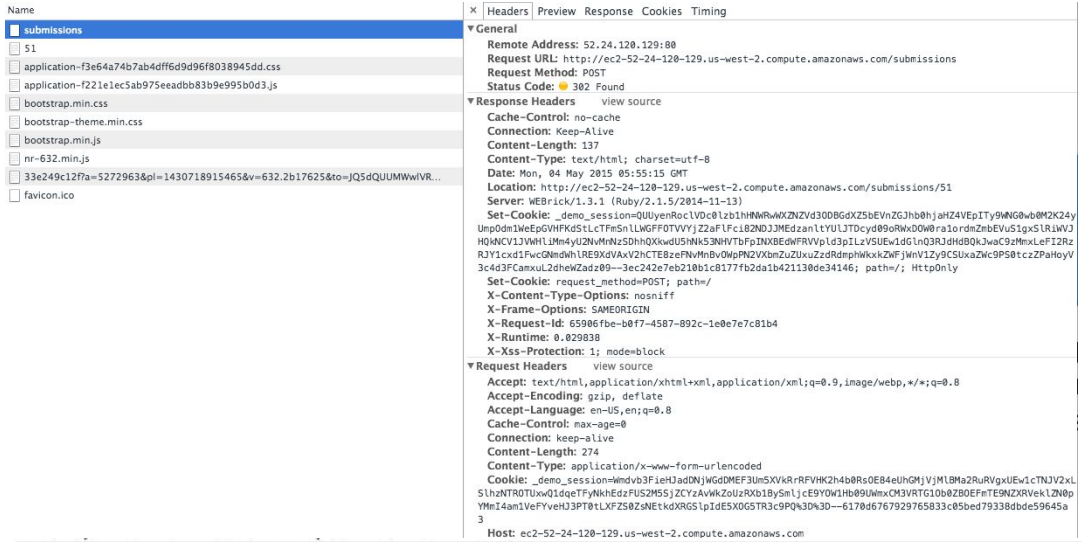
Tsung - test.xml

```
  <!-- 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



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.

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.

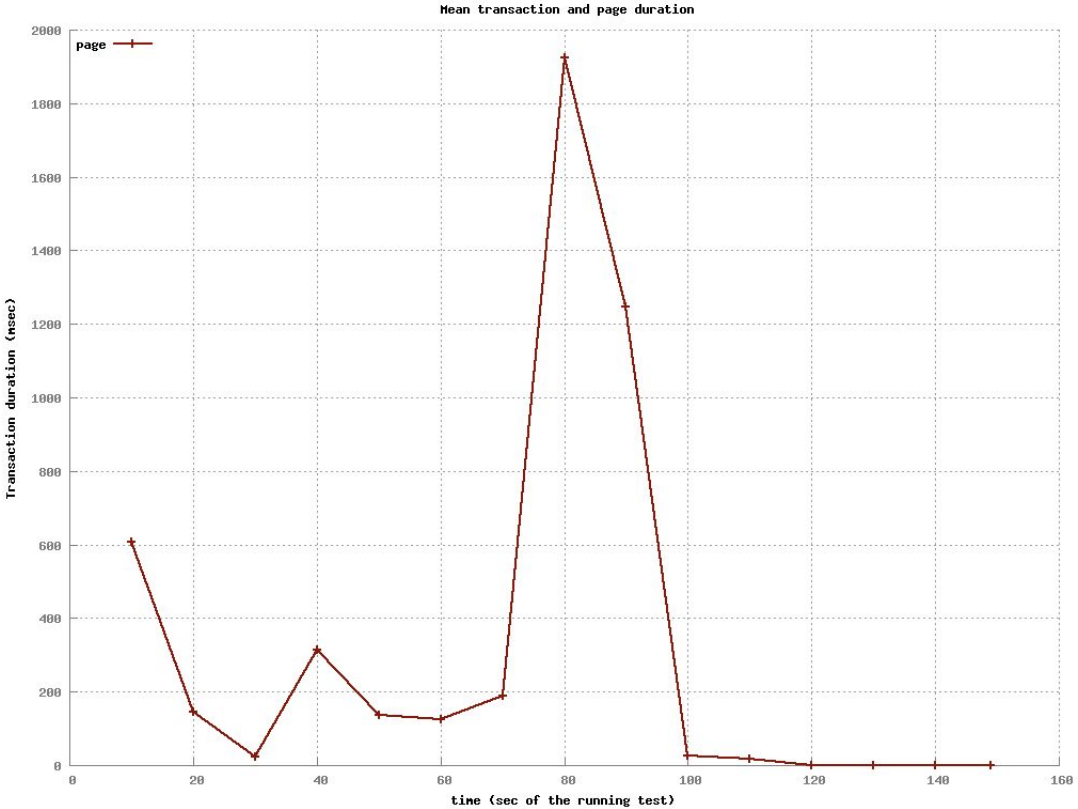
Tsung - Output

HTTP return code

Code	Highest Rate	Total number
200	0.4 / sec	6

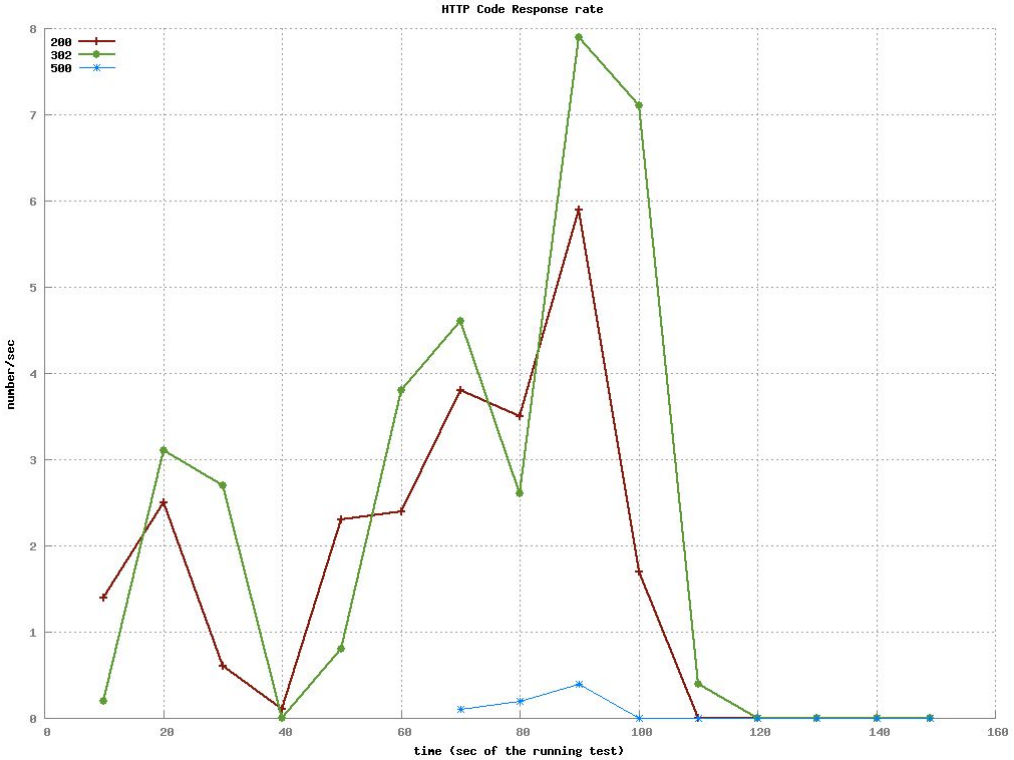
- Make sure you are getting back good status codes.
  - 200s and 300s are good
  - 400s and 500s are usually bad

Tsung - Output



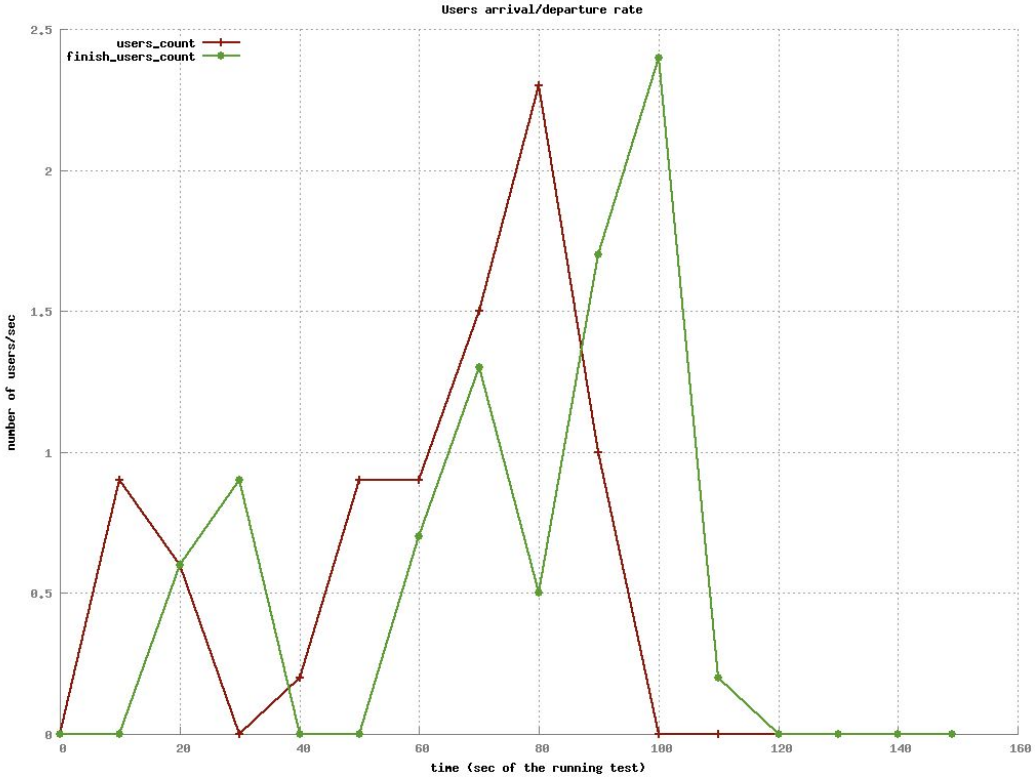
- Response times

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!