Clustering and Performance Testing on Amazon Web Services (AWS)

February 2020

Table of Contents

Executive Summary **4**
- Introduction 4
- Test Environment 4
- Summary of Results 6
  - Throughput 6
  - Application Performance Index (Apdex) 7
- Conclusion and Recommendations 8

Test Environment Setup **9**
- Test Environment 9
  - Application Server 9
  - Database Server 9
  - Web Server/Load Balancer 9
  - Load Test Client 10
  - Test App 10
  - Test Script 11
  - Test Methodology 11
- Setup the Joget Server Cluster 12
  - Launch EC2 Instance 12
  - Install Java 12
  - Install Joget 12
  - Install Nginx 12
  - Configure Load Balancer 12
  - Configure Shared Database 13
  - Configure Shared File Directory 14
  - Optimize Java 14
  - Optimize Tomcat 15
  - Tomcat Session Persistence 15
  - Optimize MySQL 16
- Add a New Joget Node 16
  - Launch New Joget Node 16
  - Configure New Joget Node 17
  - Add to Load Balancer 17
- Use the EC2 Elastic Load Balancer 17
- Setup Load Testing Clients 18
  - Create a folder to store Jmeter test file, results and reports 18
  - Download & Configure Jmeter 18
  - Run Jmeter 18

Performance Test Results **19**
100 users 1 node 19
250 users 1 node 20
500 users 1 node 21
750 users 1 node 22
1000 users 1 node 23
1000 users 2 node cluster 24
2000 users 2 node cluster 25
2000 users 3 node cluster 26

Appendix: Sample Test Output **27**
1000 users 1 node Jmeter output 27
2000 users 3 node cluster Jmeter output 27

1. Executive Summary

1.1) Introduction

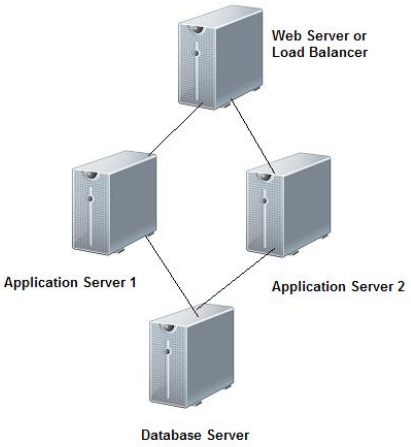
Joget DX is a next generation open source application platform for faster, simpler digital transformation (DX). Joget DX combines the best of business process automation, workflow management and low code application development in a simple, flexible and open platform.

This document is intended to describe and analyze the results of performance tests on a clustered deployment of Joget DX on Amazon Web Services (AWS).

1.2) Test Environment

The tests were conducted on Amazon Web Services (AWS), specifically using the Elastic Compute Cloud (EC2). AWS offered great flexibility in allowing servers and clients to be created and scaled up as required.

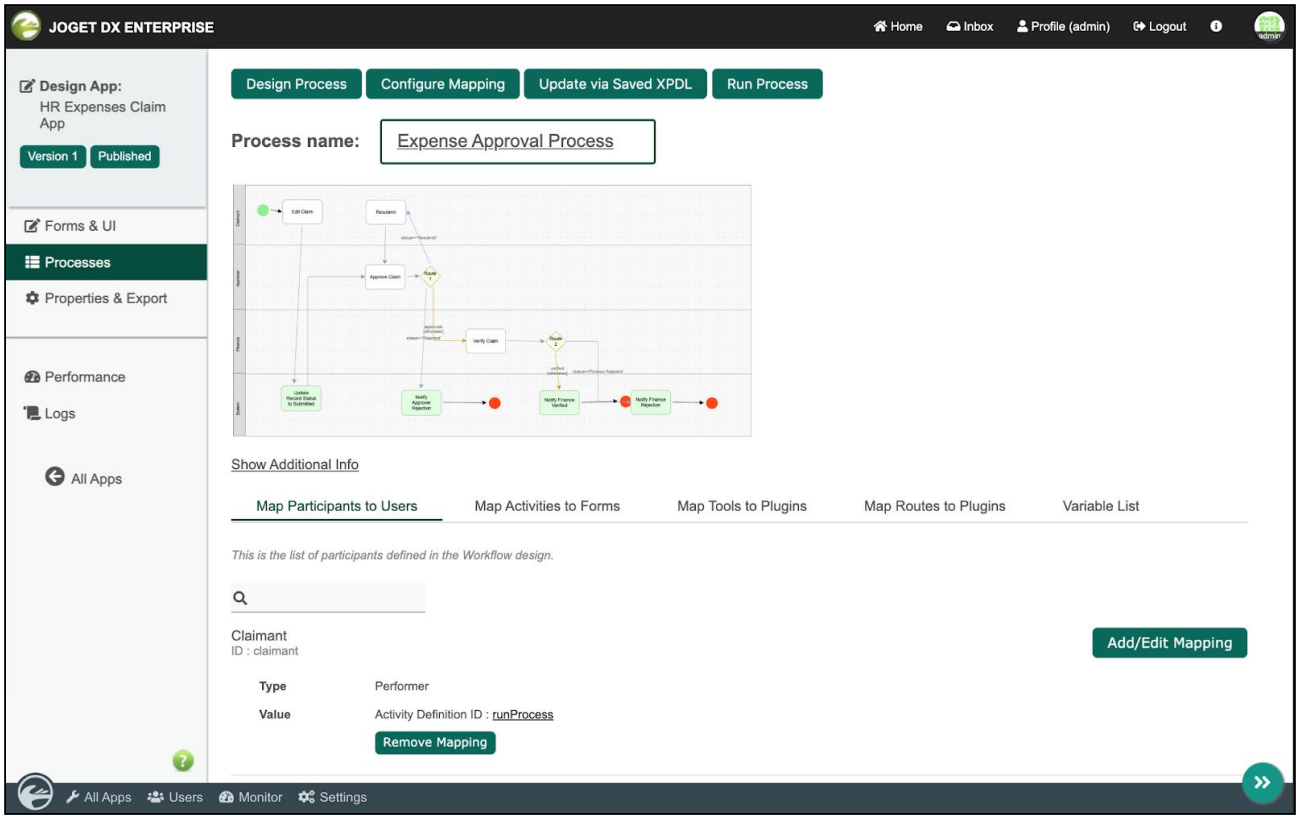
The architecture of the clustered deployment is similar to the following diagram:



The test was conducted using the following product versions:

Joget: Joget DX Large Enterprise Edition Snapshot build 5ea5994
OS: Ubuntu 18.04.4 LTS
Java: OpenJDK 11.0.6
Web Application Server: Apache Tomcat 8.5.41
Database: MySQL 5.7
Web Server/Load Balancer: Nginx Web Server 1.14
Load Testing Tool: Apache JMeter 5.2.1

To establish the baseline performance, a HR Expenses Claim test app was used.



Using a think time of 10 seconds with random deviation of 3 seconds, the test script used covers the following app usage:

1. View Login Page
2. Submit Login Form
3. View Expenses Claim Form
4. Get CSRF Token
5. Submit Expenses Claim Form
6. Get CSRF Token
7. Submit Expenses Claim Form to Approver
8. Logout

Tests were carried out for the following:

1. 100 concurrent users on 1 node (c5.xlarge)
2. 250 concurrent users on 1 node (c5.xlarge)
3. 500 concurrent users on 1 node (c5.xlarge)
4. 750 concurrent users on 1 node (c5.xlarge)
5. 1000 concurrent users on 1 node (c5.xlarge)
6. 1000 concurrent users on 2 nodes (c5.xlarge)
7. 2000 concurrent users on 2 nodes (c5x.large)
8. 2000 concurrent users on 3 nodes (c5x.large)

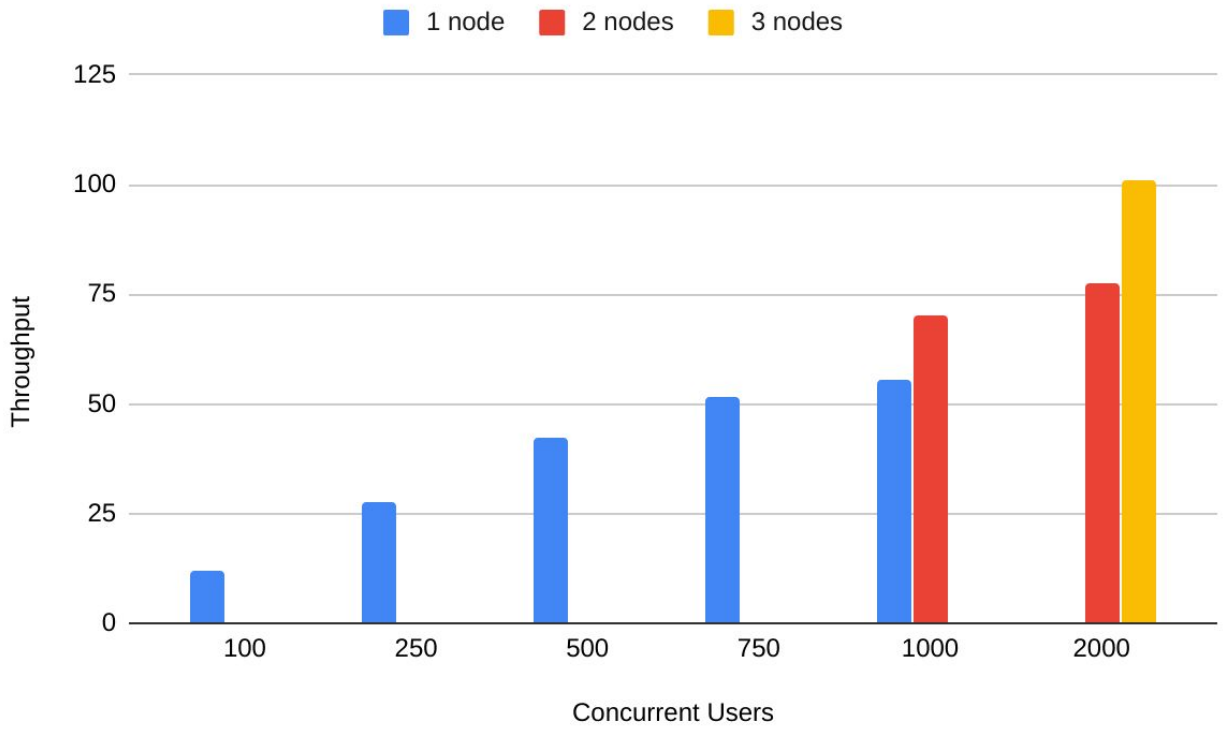
For each test, the JMeter summary results were collected. Once all the results were collected, the throughput (requests per second) and average response times were compared and analyzed.

1.3) Summary of Results

Throughput

The results are summarized in the table and graph below:

Throughput (Request/Second)			
Concurrent Users	1 node	2 nodes	3 nodes
100	11.96		
250	27.54		
500	42.08		
750	51.68		
1000	55.44	70.25	
2000		77.69	101.04



In terms of throughput, a single c5.xlarge node (2 vCPU, 8GB RAM) maxed out at about 51 requests per second at 750 concurrent users. When the concurrent users increased to 1000, the throughput remained about the same.

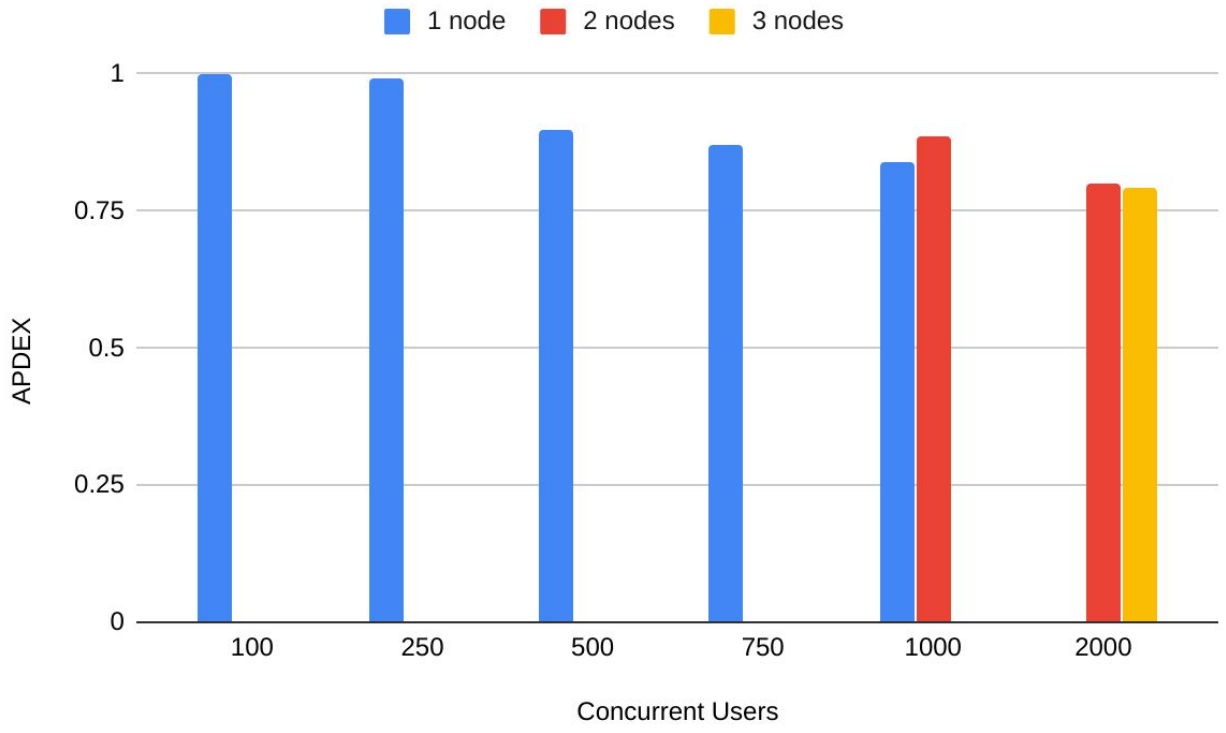
When scaling out (adding nodes to the cluster), the throughput seems to improve linearly as can be seen by the throughput graph i.e. 70 requests per second for a 2 node cluster at 1000 concurrent users, and 101 requests per second for a 3 node cluster to handle 2000 concurrent users.

Application Performance Index (Apdex)

Apdex is an open standard for measuring performance of software applications The results are summarized in the table and graph below:

Apdex Score

Concurrent Users	1 node	2 nodes	3 nodes
100	1		
250	0.993		
500	0.899		
750	0.872		
1000	0.841	0.888	
2000		0.799	0.794



1.4) Conclusion and Recommendations

From the results it can be seen that for a basic baseline app, a single modestly spec-ed c5.xlarge server (2 vCPU, 8GB RAM) can handle 500 concurrent users with acceptable response times. The tests also show that scaling out horizontally (adding nodes to a cluster), supports an almost linear increase in concurrent users. Performing vertical scaling can also improve performance, as demonstrated in the use of the increasingly powerful EC2 instance types.

With emphasis on performance optimization at the core platform, Joget DX incurs low overhead when running apps. If there are any specific bottlenecks, it would usually be at the application or plugin level. At the application level, there are various guidelines and best practices that are available in the Performance Optimization and Scalability Tips article in the Joget DX 7 Knowledge Base. Joget DX provides many performance related features such as Application Performance Monitoring and Alerts, Performance Analyzer, and Userview Caching.

For large deployments that support large numbers of concurrent users, it is important that the environment is tuned and optimized e.g. Java VM tuning, app server tuning, database optimization, as per the Deployment Best Practices article.

It is important to note that as Joget is a platform and not directly an end-user app, the scalability and performance would depend on potentially many factors:

1. Total number of users
2. Maximum expected concurrent users
3. Number of apps running on the platform
4. Complexity of each of the apps
5. Amount of data generated in each app
6. Network infrastructure

The recommended deployment architecture would very much depend on the environment and usage. Perhaps some things to be considered:

1. How many total and concurrent users are there? Will this grow in future?
2. In the current environment, is the current infrastructure sufficient for the load? Would it be possible to increase the server resources?
3. If the needs outgrow one server node, it might be time to consider implementing clustering and/or load balancing.
4. Another possible approach could be to partition the apps. Are there specific apps that incur the highest load? Maybe it might be appropriate to separate apps into different servers.
5. Deploy Joget on cloud native platforms like Red Hat OpenShift to take advantage of autoscaling.

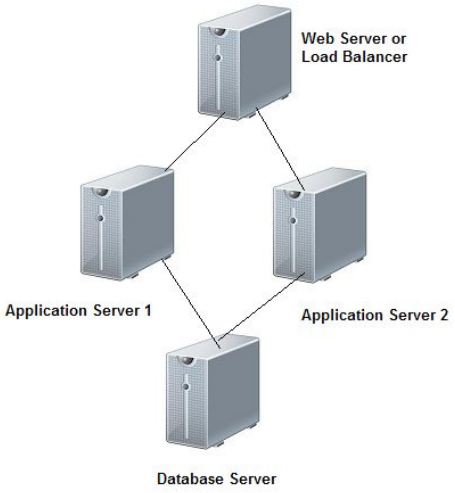
In summary, this report demonstrates the baseline performance of the Joget DX platform for a basic app and shows how horizontal and vertical scaling can be used to support larger deployments. Although these results can serve as a base guideline, it is recommended that performance testing and optimisations are performed based on each deployment's unique requirements, environments and usage patterns.

2. Test Environment Setup

2.1) Test Environment

The tests were conducted on Amazon Web Services (AWS), specifically using the Elastic Compute Cloud (EC2). AWS offered great flexibility in allowing servers and clients to be created and scaled up as required.

The architecture of the clustered deployment is similar to the following diagram:



Application Server

Joget: Joget DX Large Enterprise Edition Snapshot build 5ea5994
OS: Ubuntu 18.04.4 LTS
Java: OpenJDK 11.0.6
Web Application Server: Apache Tomcat 8.5.41
EC2 Instance: c5.xlarge

- 2 vCPU (virtual CPUs)
- 8GB RAM
- Java VM Options: -XX:MaxPermSize=256M -Xms4096M -Xmx4096M

Database Server

Database: MySQL 5.7
EC2 Instance: c5.xlarge

- 4 vCPU
- 8GB RAM
- Moderate network performance
- 1000 PIOPS

Web Server/Load Balancer

OS: Ubuntu 18.04.4 LTS
Web Server/Load Balancer: Nginx Web Server 1.14
EC2 Instance: c5.large:

- 2 vCPU

- 4GB RAM

Load Test Client

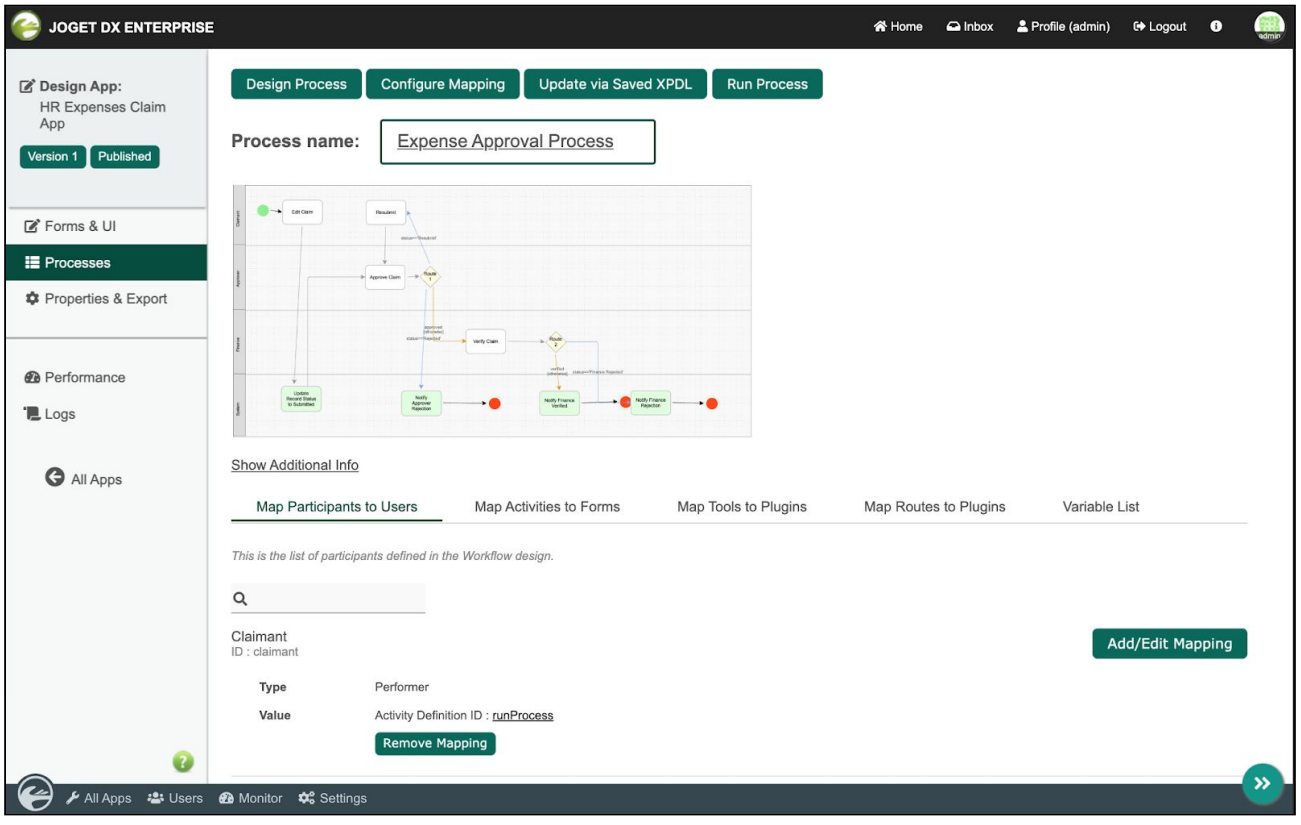
Load Testing Tool: Apache JMeter 5.2.1
OS: Ubuntu 18.04.4 LTS
EC2 Instance: t2.medium:

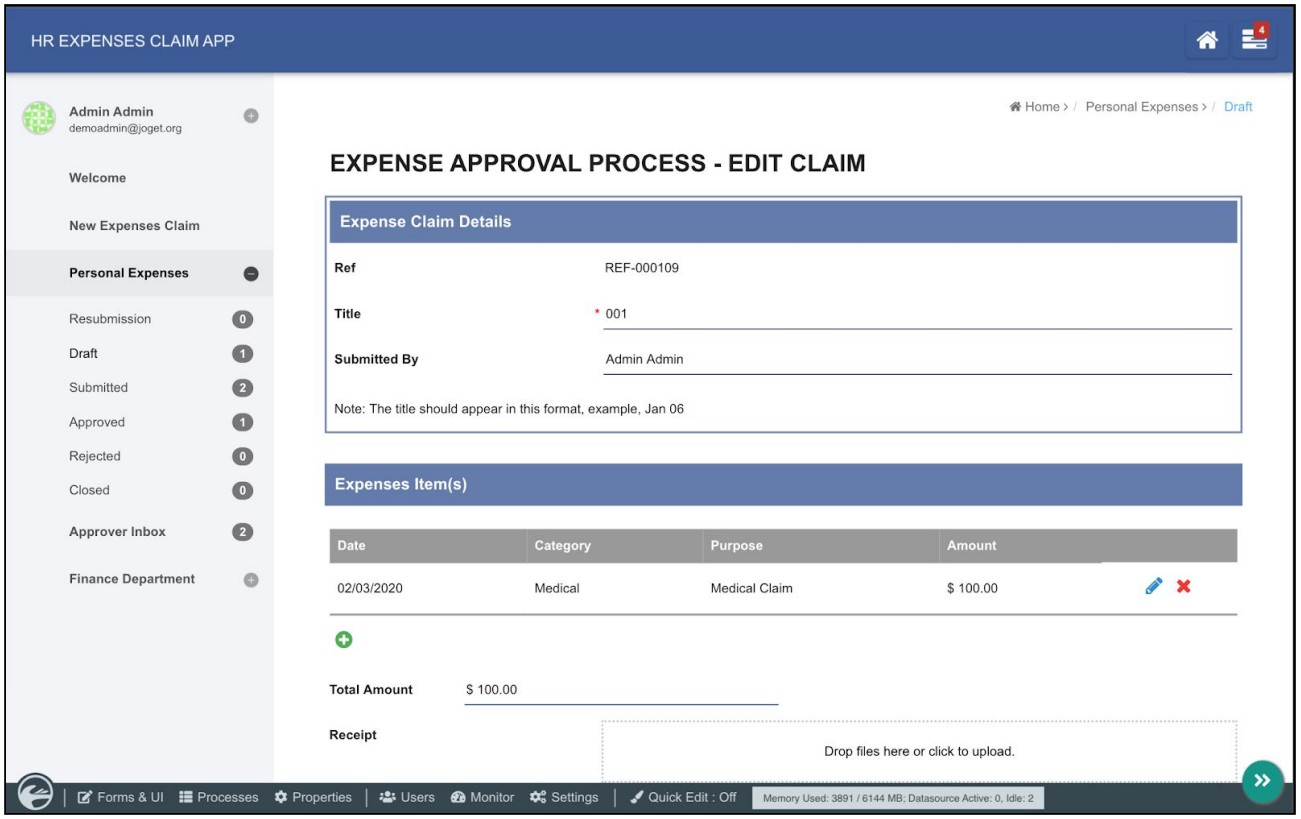
- 2 vCPU
- 4GB RAM

Test App

To establish the baseline performance, a HR Expenses Claim test app was used consisting of:

1. 1 process with 4 activities and 4 tools
2. 8 forms
3. 8 datalists
4. 1 userview containing menu pages to run the process and display the datalist and inbox





Test Script

The test script used covers the following app usage:

1. View Login Page
2. Submit Login Form
3. View Expenses Claim Form
4. Get CSRF Token
5. Submit Expenses Claim Form
6. Get CSRF Token
7. Submit Expenses Claim Form to Approver
8. Logout

A think time of 10 seconds was used, with random deviation of 3 seconds.

Test Methodology

The load tests were executed by using the latest Apache Jmeter, which provides an automated way of launching, running and collecting JMeter results.

Tests were carried out for the following:

1. 100 concurrent users on 1 node (c5.xlarge)
2. 250 concurrent users on 1 node (c5.xlarge)
3. 500 concurrent users on 1 node (c5.xlarge)
4. 750 concurrent users on 1 node (c5.xlarge)
5. 1000 concurrent users on 1 node (c5.xlarge)

6. 2000 concurrent users on 2 nodes (c5.xlarge)
7. 2000 concurrent users on 3 nodes (c5.xlarge)

For each test, the JMeter summary results were collected. Once all the results were collected, the throughput (requests per second) and average response times were compared and analyzed.

2.2) Setup the Joget Server Cluster

The following are brief descriptions of the steps used to setup the server instances:

Launch EC2 Instance

Login to the AWS Management Console and launch the appropriate EC2 instance running on Ubuntu 18.04.4. Once started, SSH into the server.

Install Java

```
sudo apt-get install openjdk-11-jdk
```

Install Joget

Download Linux tar.gz bundle
Extract into /opt/joget
Run setup.sh and configure to the database

Install Nginx

For the load balancer, install Nginx web server

```
sudo apt-get install nginx
```

Configure Load Balancer

For the load balancer, another section in /etc/nginx/nginx.conf has been added

```
upstream joget {
      least_conn;
      server 172.31.31.172:8080 weight=1;
      server 172.31.30.203:8080 weight=1;
}
```

Increase the maximum number of open files by adding

```
fs.file-max=100000
```

into /etc/sysctl.conf

Increase the limit on the maximum number of open files for worker processes in Nginx by adding

```
worker_rlimit_nofile 30000;
```

into /etc/nginx/nginx.conf

Create a new file in /etc/nginx/sites-available, named joget

```
sudo vim /etc/nginx/sites-available/joget
```

Add the contents

```
server {
  listen 80;
  server_name 172.31.17.170;
  underscores_in_headers on;
  client_body_buffer_size 10K;
  client_header_buffer_size 1k;
  client_max_body_size 8m;
  large_client_header_buffers 2 1k;
  location / {
      proxy_pass http://joget;
      proxy_redirect  off;
      proxy_pass_header X-CSRF-TOKEN;
      proxy_set_header   Host 172.31.17.170;
      proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
      proxy_set_header   X-NginX-Proxy true;
      proxy_read_timeout 3000;
      proxy_buffers           32 4m;
      proxy_busy_buffers_size     25m;
      proxy_buffer_size 512k;
      proxy_ignore_headers "Cache-Control" "Expires";
      proxy_max_temp_file_size 0;
      client_max_body_size        1024m;
      client_body_buffer_size     4m;
      proxy_connect_timeout 3000;
      proxy_headers_hash_max_size 512;
      proxy_send_timeout 3000;
      proxy_intercept_errors off;
  }
}
```

Enable the new site and reload Nginx

```
sudo ln -s /etc/nginx/site-available/joget /etc/nginx/site-enabled/joget
sudo nginx -s reload
```

Configure Shared Database

To install a local MySQL (instead of RDS)

sudo apt-get install mysql-server

Configure database permissions

```
mysql -u root
```

Run the following MySQL commands to grant permissions to user **joget** and password **joget**

```
grant all privileges on jwdb.* to 'joget'@'%' identified by 'joget';
flush privileges;
quit
```

Configure MySQL to listen to database connections from remote hosts. Edit the my.cnf file with your favourite editor

```
sudo vim /etc/mysql/my.cnf
```

Comment away the bind-address directive by adding a # in front of the line

```
#bind-address = 127.0.0.1
```

Restart MySQL

```
sudo systemctl restart mysql
```

Test remote connections. In the application server, test a remote connection to the database server **database_host**

```
mysql -h database_host -u joget -p
```

Configure Shared File Directory

Install NFS (for sharing file system)

```
sudo apt-get install portmap nfs-kernel-server nfs-common
```

Detailed instructions can be found at
http://theredblacktree.wordpress.com/2013/05/23/how-to-setup-a-amazon-aws-ec2-nfs-share/

Create new directory /opt/joget/shared/wflow to mount the shared directory and set the directory permissions

```
sudo mkdir -p /opt/joget/shared/wflow
sudo chmod 777 /opt/joget/shared/wflow
```

Mount the shared directory.

```
sudo mount -t nfs joget-server:/export/wflow /opt/joget/shared/wflow
```

Test read-write permissions to confirm that the directory sharing works.

```
echo test123 > /opt/joget/shared/wflow/test.txt
```

Optimize Java

Set appropriate Java heap settings e.g.

```
export JAVA_OPTS="-XX:MaxPermSize=256m -Xms4096M -Xmx4096M
-Dwflow.home=/opt/joget/wflow/ "
```

Optimize Tomcat

Edit server.xml and add connectors, especially maxThreads

```
<Connector URIEncoding="UTF-8" port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000" maxThreads="2000"
           redirectPort="8443" />
 <Connector port="9090" protocol="HTTP/1.1"
           connectionTimeout="20000" maxThreads="2000"
           scheme="https"
           proxyPort="443"
           redirectPort="443" />
```

Configure Linux ulimit Configuration:

```
ulimit -n 4096
```

Tomcat Session Persistence

To simulate an actual environment, in the event the load balancer does not support sticky sessions, we can implement Persistent Manager in Tomcat, which has the capability to swap active (but idle) sessions out to a persistent storage mechanism, as well as to save all sessions across a normal restart of Tomcat.

We need to set `org.apache.catalina.session.StandardSession.ACTIVITY_CHECK=true` in /opt/joget/apache-tomcat-8.5.41/conf/catalina.properties to ensure the persistent manager works correctly.

In this testing we use a JDBC Based Store to save sessions in individual rows of a preconfigured table in a database that is accessed via a JDBC driver. Create a database named tomcat and table with the following SQL queries:

```
create database tomcat;
grant all privileges on tomcat.* to 'tomcat'@'%' identified by 'tomcat';
use tomcat;
create table tomcat_sessions (
  session_id     varchar(100) not null primary key,
  valid_session  char(1) not null,
  max_inactive   int not null,
  last_access    bigint not null,
  app_name       varchar(255),
  session_data   mediumblob,
  KEY kapp_name(app_name)
);
```

In order for the JDBC Based Store to successfully connect to the database, we need to place the JAR file containing MySQL JDBC driver into /opt/joget/apache-tomcat-8.5.41/lib directory.

Last but not least, add the following content into /opt/joget/apache-tomcat-8.5.41/conf/context.xml

```
<Valve className="org.apache.catalina.valves.PersistentValve"/>

<Manager className="org.apache.catalina.session.PersistentManager"
    maxIdleBackup="0"
    maxIdleSwap="0"
```

```
        minIdleSwap="1"
        processExpiresFrequency="1"
        saveOnRestart='true'
        >
    <Store className="org.apache.catalina.session.JDBCStore"

connectionURL="jdbc:mysql://172.31.27.184/tomcat?user=tomcat&amp;password=tomcat"
        driverName="com.mysql.jdbc.Driver"
        sessionAppCol="app_name"
        sessionDataCol="session_data"
        sessionIdCol="session_id"
        sessionLastAccessedCol="last_access"
        sessionMaxInactiveCol="max_inactive"
        sessionTable="tomcat_sessions"
        sessionValidCol="valid_session"
    />
    </Manager>
```

Remark: `172.31.27.184` is the IP address of MySQL Database server.

Optimize MySQL

Create /etc/mysql/mysql.cnf containing the following and restart MySQL

```
[mysqld]
query_cache_limit=1M
query_cache_size=32M
max_allowed_packet=16M
lower_case_table_names=1
collation_server=utf8_unicode_ci
character_set_server=utf8
key_buffer_size=16M
read_buffer_size=16M
read_rnd_buffer_size=16M
max_connections=10000
```

2.3) Add a New Joget Node

When adding a new node to the server cluster, the following steps are taken (in this sample the new node hostname will be joget-server3):

Launch New Joget Node

Launch new instance of AMI

Choose appropriate security groups (default and nfs)

Configure New Joget Node

SSH into node

Edit /etc/hosts to add node hostname, and modify joget-server IP if necessary e.g.

```
127.0.0.1 joget-server3
172.31.30.203 joget-server
```

Edit /etc/hostname to modify node hostname e.g.

```
joget-server3
```

Modify hostname e.g.

```
sudo hostname joget-server3
```

Remount NFS share (if joget-server shared directory IP was modified)

Configure Tomcat for clustering by editing server.xml. Add **jvmRoute="node03"** to the **Engine** tag.

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="node03">
```

Restart Tomcat.

Add to Load Balancer

In the load balancer, edit /etc/nginx/nginx.conf to add the BalancerMember node e.g.

```
upstream joget {
      least_conn;
      server 172.31.31.172:8080 weight=1;
      server 172.31.30.203:8080 weight=1;
      server 172.31.27.120:8080 weight=1;
}
```

then reload/restart Nginx.

2.4) Use the EC2 Elastic Load Balancer

It is possible to use the EC2 Elastic Load Balancer (ELB) instead of Nginx web server. To do so:

In the AWS Management Console,

Create a New Load Balancer

For Health Check, use /jw/web/json/workflow/currentUsername

If sticky sessions are required, it is configurable under Description, Port Configuration, Edit Stickiness and select Enable Application Generated Cookie Stickiness with JSESSIONID.

2.5) Setup Load Testing Clients

Create a folder to store Jmeter test file, results and reports

mkdir -p ~/load_tests/reports

Download & Configure Jmeter

Download Jmeter from from https://jmeter.apache.org/

Extract the installer and edit user.properties file

```
vi apache-jmeter-5.2.1/bin/user.properties
```

change the value of APDEX satisfied and tolerated threshold.

```
# Change this parameter if you want to override the APDEX satisfaction threshold.
jmeter.reportgenerator.apdex_satisfied_threshold=5000

# Change this parameter if you want to override the APDEX tolerance threshold.
jmeter.reportgenerator.apdex_tolerated_threshold=10000
```

Run Jmeter

copy the jmeter test file (perf_v7_test_expenses_app-aws.jmx) in ~/load_tests/ and run jmeter-ec2

```
cd apache-jmeter-5.2.1

bin/jmeter.sh -n -t ~/load_tests/perf_v7_test_expenses_app-aws.jmx -l
~/tests/result.csv -e -o ~/load_tests/reports/
```

3. Performance Test Results

3.1) 100 users 1 node

Application Server: 1 c5.xlarge node

Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 100 users in 1 t3.medium instance

Concurrent Users: 100 users

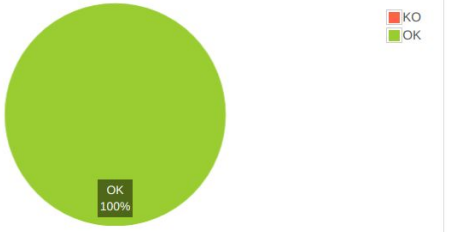
Ramp-up Time: 10s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	**5 sec**	**10 sec**	**Total**
1.000	5 sec	10 sec	View Login Page-2
1.000	5 sec	10 sec	View Login Page-1
1.000	5 sec	10 sec	View Login Page-0
1.000	5 sec	10 sec	View Login Page
1.000	5 sec	10 sec	Submit Expenses Claim Form to Approver-1
1.000	5 sec	10 sec	Logout-2

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1963	0	0.00%	155.24	0	2527	451.60	895.00	1919.00	11.96	219.17	4.97

3.2) 250 users 1 node

Application Server: 1 c5.xlarge node

Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 250 users in 1 t3.medium instance

Concurrent Users: 250 users

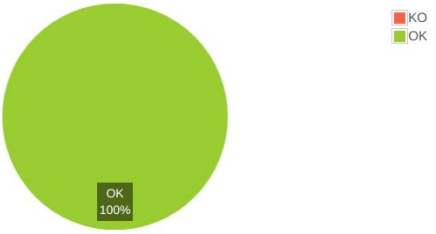
Ramp-up Time: 25s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.993	**5 sec**	**10 sec**	**Total**
1.000	5 sec	10 sec	View Login Page-2
1.000	5 sec	10 sec	View Login Page-1
1.000	5 sec	10 sec	View Login Page-0

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	5001	0	0.00%	380.36	0	9426	863.00	2567.80	5713.76	27.54	494.44	11.42

3.3) 500 users 1 node

Application Server: 1 c5.xlarge node

Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 500 users in 1 t3.medium instance

Concurrent Users: 500 users

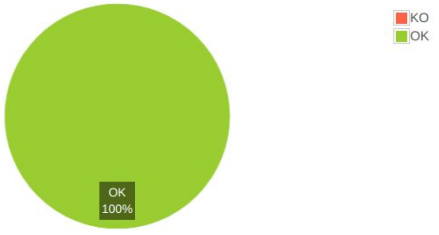
Ramp-up Time: 50s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.899	**5 sec**	**10 sec**	**Total**
1.000	5 sec	10 sec	View Login Page-1
1.000	5 sec	10 sec	View Login Page-0
1.000	5 sec	10 sec	Logout-11
1.000	5 sec	10 sec	Logout-10

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	9826	0	0.00%	2724.50	0	46612	7694.60	24125.55	32623.92	42.08	767.52	17.49

3.4) 750 users 1 node

Application Server: 1 c5.xlarge node

Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 750 users in 1 t3.medium instance

Concurrent Users: 750 users

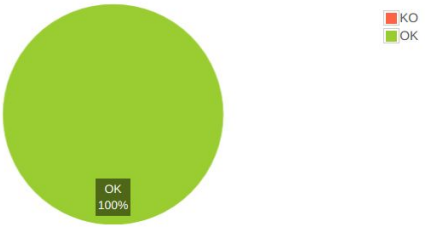
Ramp-up Time: 75s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.872	**5 sec**	**10 sec**	**Total**
1.000	5 sec	10 sec	Logout-13
1.000	5 sec	10 sec	View Login Page-2
1.000	5 sec	10 sec	Logout-12
1.000	5 sec	10 sec	View Login Page-1

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	14699	0	0.00%	4507.43	0	79798	18784.00	38024.00	51562.00	51.68	945.93	21.50

3.5) 1000 users 1 node

Application Server: 1 c5.xlarge node

Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 1000 users in 1 t3.medium instance

Concurrent Users: 1000 users

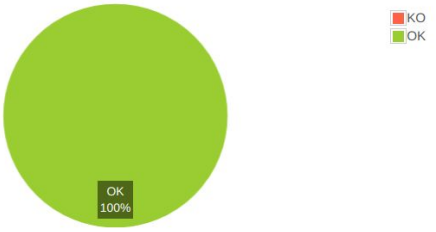
Ramp-up Time: 100s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.841	5 sec	10 sec	Total
1.000	5 sec	10 sec	View Login Page-1
1.000	5 sec	10 sec	View Login Page-0
1.000	5 sec	10 sec	Logout-11
1.000	5 sec	10 sec	Logout-10

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	19510	0	0.00%	7069.60	0	90129	35570.80	53156.60	68422.67	55.44	1018.93	23.07

3.6) 1000 users 2 node cluster

Load Balancer: Nginx web server c5.large

Application Server: 2 c5.xlarge node

Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 1000 users in t3.medium instance

Concurrent Users: 1000 users

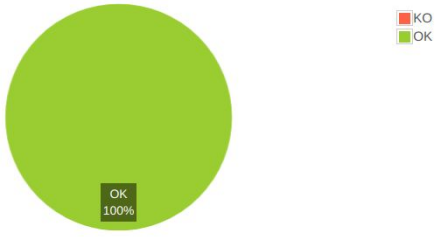
Ramp-up Time: 100s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.888	**5 sec**	**10 sec**	**Total**
1.000	5 sec	10 sec	View Login Page-2
1.000	5 sec	10 sec	View Login Page-1
1.000	5 sec	10 sec	View Login Page-0
1.000	5 sec	10 sec	Logout-11

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	19603	0	0.00%	2317.51	2	39758	9691.00	17733.80	23816.76	70.25	1285.26	29.65

3.7) 2000 users 2 node cluster

Load Balancer: Nginx web server c5.large

Application Server: 2 c5.xlarge node

Database: Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 2000 users in t3.medium instance

Concurrent Users: 2000 users

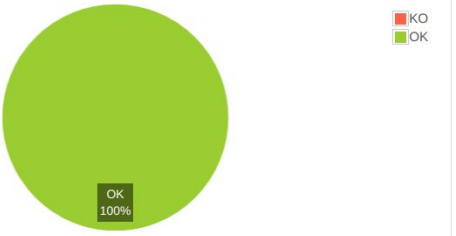
Ramp-up Time: 100s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.799	5 sec	10 sec	Total
1.000	5 sec	10 sec	Logout-13
1.000	5 sec	10 sec	View Login Page-2
1.000	5 sec	10 sec	Logout-12
1.000	5 sec	10 sec	View Login Page-1

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	38938	0	0.00%	11770.43	3	147863	71214.10	79113.85	90354.94	77.69	1432.06	32.84

3.8) 2000 users 3 node cluster

Application Server: 3 c5.xlarge node

Database: Database Server: 1 c5.xlarge node with 1000 PIOPS

Client: 2000 users in 1 t3.medium instances

Concurrent Users: 2000 users

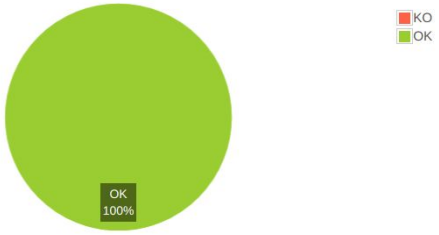
Ramp-up Time: 100s ramp-up

Think Time: 10s random delay 3s deviation

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.778	**5 sec**	**10 sec**	**Total**
1.000	5 sec	10 sec	View Login Page-2
1.000	5 sec	10 sec	View Login Page-1
1.000	5 sec	10 sec	View Login Page-0
1.000	5 sec	10 sec	Logout-11

Requests Summary



Statistics

Requests	Executions			Response Times (ms)						Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	39014	0	0.00%	9879.65	2	131461	63909.70	72540.85	84951.76	101.04	1857.94	42.70

Appendix: Sample Test Output

1000 users 1 node Jmeter output

```
summary +    163 in 00:00:21 =    7.7/s Avg:    33 Min:    24 Max:   166 Err:     0 (0.00%) Active: 213 Started: 213 Finished: 0
summary +    573 in 00:00:30 =   19.2/s Avg:    54 Min:    20 Max:   214 Err:     0 (0.00%) Active: 512 Started: 512 Finished: 0
summary =    736 in 00:00:51 =   14.4/s Avg:    50 Min:    20 Max:   214 Err:     0 (0.00%)
summary +   1011 in 00:00:30 =   33.7/s Avg:    52 Min:     4 Max:   194 Err:     0 (0.00%) Active: 812 Started: 812 Finished: 0
summary =   1747 in 00:01:21 =   21.5/s Avg:    51 Min:     4 Max:   214 Err:     0 (0.00%)
summary +   1337 in 00:00:30 =   44.5/s Avg:   148 Min:     4 Max:  1127 Err:     0 (0.00%) Active: 1000 Started: 1000 Finished: 0
summary =   3084 in 00:01:51 =   27.7/s Avg:    93 Min:     4 Max:  1127 Err:     0 (0.00%)
summary +   1306 in 00:00:30 =   43.5/s Avg:   263 Min:     4 Max:  4389 Err:     0 (0.00%) Active: 1000 Started: 1000 Finished: 0
summary =   4390 in 00:02:21 =   31.1/s Avg:   144 Min:     4 Max:  4389 Err:     0 (0.00%)
summary +   1042 in 00:00:30 =   34.7/s Avg:  3934 Min:     4 Max: 20620 Err:     0 (0.00%) Active: 1000 Started: 1000 Finished: 0
summary =   5432 in 00:02:51 =   31.7/s Avg:   871 Min:     4 Max: 20620 Err:     0 (0.00%)
summary +    840 in 00:00:30 =   28.0/s Avg:  9891 Min:     4 Max: 30783 Err:     0 (0.00%) Active: 847 Started: 1000 Finished: 153
summary =   6272 in 00:03:21 =   31.2/s Avg:  2079 Min:     4 Max: 30783 Err:     0 (0.00%)
summary +    827 in 00:00:30 =   27.6/s Avg: 10937 Min:     4 Max: 39758 Err:     0 (0.00%) Active: 642 Started: 1000 Finished: 358
summary =   7099 in 00:03:51 =   30.7/s Avg:  3111 Min:     4 Max: 39758 Err:     0 (0.00%)
summary +    724 in 00:00:30 =   24.2/s Avg:  1083 Min:     4 Max: 11287 Err:     0 (0.00%) Active: 414 Started: 1000 Finished: 586
summary =   7823 in 00:04:21 =   30.0/s Avg:  2923 Min:     4 Max: 39758 Err:     0 (0.00%)
summary +    177 in 00:00:40 =    4.4/s Avg:    36 Min:    12 Max:   684 Err:     0 (0.00%) Active: 0 Started: 1000 Finished: 1000
summary =   8000 in 00:05:01 =   26.6/s Avg:  2859 Min:     4 Max: 39758 Err:     0 (0.00%)
Tidying up ...    @ Tue Feb 11 03:09:09 UTC 2020 (1581390549841)
... end of run
```

2000 users 3 node cluster Jmeter output

```
summary +     37 in 00:00:07 =    5.0/s Avg:   171 Min:    26 Max:   743 Err:     0 (0.00%) Active: 147 Started: 147 Finished: 0
summary +    829 in 00:00:30 =   27.6/s Avg:    91 Min:    20 Max:  3402 Err:     0 (0.00%) Active: 748 Started: 748 Finished: 0
summary =    866 in 00:00:37 =   23.2/s Avg:    95 Min:    20 Max:  3402 Err:     0 (0.00%)
summary +   1573 in 00:00:30 =   52.5/s Avg:    74 Min:    18 Max:  3170 Err:     0 (0.00%) Active: 1347 Started: 1347 Finished: 0
summary =   2439 in 00:01:07 =   36.2/s Avg:    81 Min:    18 Max:  3402 Err:     0 (0.00%)
summary +   2299 in 00:00:30 =   76.6/s Avg:   135 Min:     3 Max:  3556 Err:     0 (0.00%) Active: 1947 Started: 1947 Finished: 0
summary =   4738 in 00:01:37 =   48.7/s Avg:   107 Min:     3 Max:  3556 Err:     0 (0.00%)
summary +   2111 in 00:00:30 =   70.3/s Avg:  2823 Min:     3 Max: 23811 Err:     0 (0.00%) Active: 2000 Started: 2000 Finished: 0
summary =   6849 in 00:02:07 =   53.8/s Avg:   944 Min:     3 Max: 23811 Err:     0 (0.00%)
summary +   1617 in 00:00:30 =   53.8/s Avg:  6847 Min:     3 Max: 44940 Err:     0 (0.00%) Active: 2000 Started: 2000 Finished: 0
summary =   8466 in 00:02:37 =   53.8/s Avg:  2072 Min:     3 Max: 44940 Err:     0 (0.00%)
summary +   1241 in 00:00:30 =   41.2/s Avg: 16730 Min:     3 Max: 66539 Err:     0 (0.00%) Active: 2000 Started: 2000 Finished: 0
summary =   9707 in 00:03:08 =   51.8/s Avg:  3946 Min:     3 Max: 66539 Err:     0 (0.00%)
summary +    899 in 00:00:30 =   30.1/s Avg: 24349 Min:     4 Max: 83574 Err:     0 (0.00%) Active: 1980 Started: 2000 Finished: 20
summary =  10606 in 00:03:37 =   48.8/s Avg:  5675 Min:     3 Max: 83574 Err:     0 (0.00%)
summary +    905 in 00:00:30 =   30.0/s Avg: 31698 Min:     4 Max: 82242 Err:     0 (0.00%) Active: 1876 Started: 2000 Finished: 124
summary =  11511 in 00:04:08 =   46.5/s Avg:  7721 Min:     3 Max: 83574 Err:     0 (0.00%)
summary +    831 in 00:00:30 =   27.9/s Avg: 38202 Min:     4 Max: 105529 Err:     0 (0.00%) Active: 1722 Started: 2000 Finished: 278
summary =  12342 in 00:04:37 =   44.5/s Avg:  9773 Min:     3 Max: 105529 Err:     0 (0.00%)
summary +    849 in 00:00:30 =   28.3/s Avg: 44639 Min:     4 Max: 141380 Err:     0 (0.00%) Active: 1592 Started: 2000 Finished: 408
summary =  13191 in 00:05:07 =   42.9/s Avg: 12017 Min:     3 Max: 141380 Err:     0 (0.00%)
summary +    773 in 00:00:30 =   25.7/s Avg: 47773 Min:     4 Max: 126539 Err:     0 (0.00%) Active: 1361 Started: 2000 Finished: 639
summary =  13964 in 00:05:37 =   41.4/s Avg: 13997 Min:     3 Max: 141380 Err:     0 (0.00%)
summary +    741 in 00:00:30 =   24.8/s Avg: 45306 Min:    15 Max: 141447 Err:     0 (0.00%) Active: 992 Started: 2000 Finished: 1008
summary =  14705 in 00:06:07 =   40.0/s Avg: 15574 Min:     3 Max: 141447 Err:     0 (0.00%)
summary +    605 in 00:00:31 =   19.5/s Avg: 38311 Min:    27 Max: 170312 Err:     0 (0.00%) Active: 613 Started: 2000 Finished: 1387
summary =  15310 in 00:06:38 =   38.4/s Avg: 16473 Min:     3 Max: 170312 Err:     0 (0.00%)
summary +    550 in 00:00:29 =   18.9/s Avg: 41292 Min:    11 Max: 132847 Err:     0 (0.00%) Active: 287 Started: 2000 Finished: 1713
summary =  15860 in 00:07:08 =   37.1/s Avg: 17334 Min:     3 Max: 170312 Err:     0 (0.00%)
summary +    140 in 00:00:23 =    6.0/s Avg:    40 Min:    11 Max:  3079 Err:     0 (0.00%) Active: 0 Started: 2000 Finished: 2000
summary =  16000 in 00:07:31 =   35.5/s Avg: 17182 Min:     3 Max: 170312 Err:     0 (0.00%)
Tidying up ...    @ Wed Feb 12 06:52:53 UTC 2020 (1581490373470)
... end of run
```