OpenLDAP Load Testing with Apache JMeter

LDAPCon 2026

October 7, 2026
Tübingen

Intro

- Shawn McKinney
- Nadezhda Ivanova

Agenda

- A load testing Overview
- A slamd primer
- An Apache Jmeter primer
- Show some load testing examples

Using

- OpenLDAP: server
- Apache Maven: build and execution env
- Apache JMeter: test framework
- Apache LDAP API + Fortress: framework

Load Testing Overview

What is it?

Load Testing Overview

- Sizing and Scalability
  - Service, Database, Directory, Image
- Performance improvements

Types of Load Tests

- Smoke
- Soak
- Stress
- Break

Common Objectives

- Accurate sizing
- Evaluate efficiency
- Hunt for bottlenecks
  - test new features
  - catch regressions
- Bug Reproduction -> Race conditions

Common Requirements

- Automated
- Scales per hardware specs
- Generates load with many machines (simultaneously)
- Reporting and logging
- Dashboard for real-time evaluation
- Monitorable
- Configurable runtime
  - # procs, threads
  - # iterations
  - # requests per minute (throttle)
- Flexible/extensible tests

When to do it

- After your server crashes
- Before the next big feature
- With Every major release
- ... minor release
- ... build

Why do it

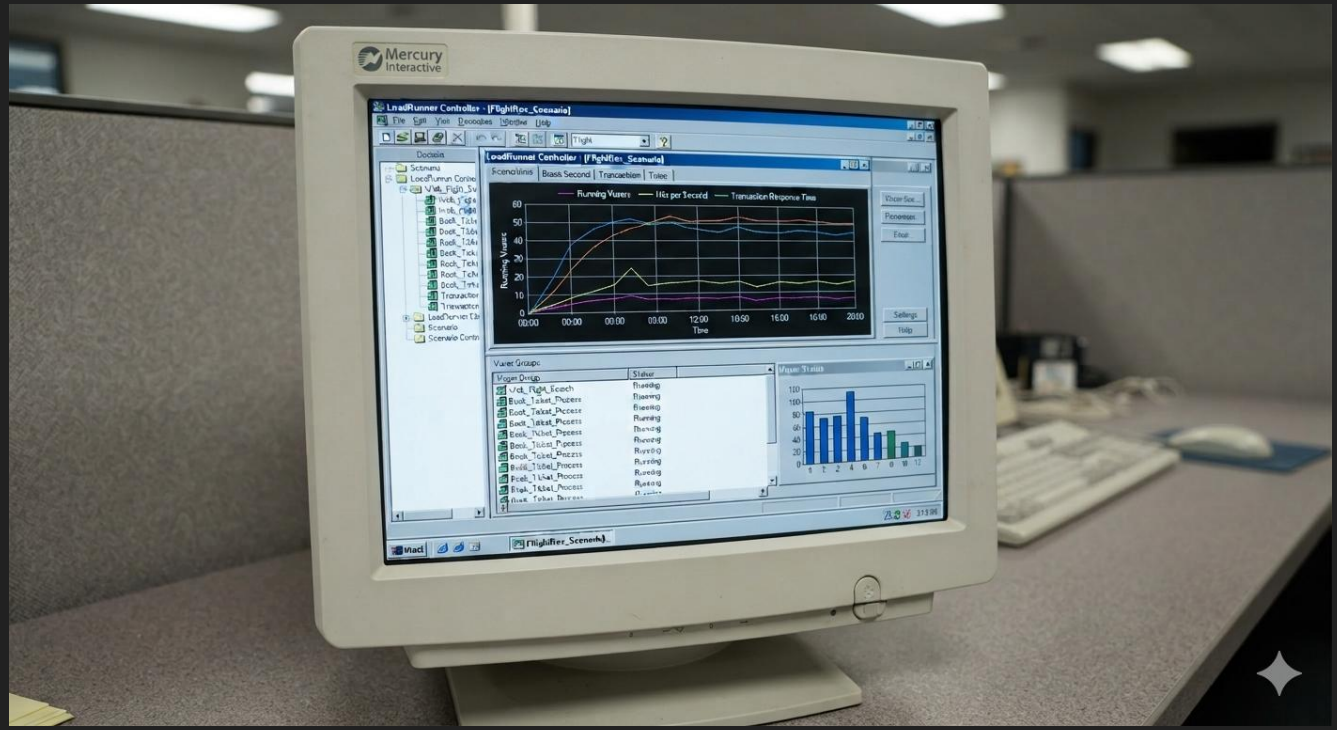
- Recreate real world scenarios
- gain competitive advantage
- Because it's fun to break things

Load testing tools

- Commodity-based market
- Homegrown, open source and commercial
- https://testguild.com/load-testing-tools/
- ...

Example 0: Pre–World Wide Web

Example 0.5: Mercury Interactive Suite



Example 1: Pre-Apache JMeter

```
# WEBLOGIC 8.X TEST
# CAN'T USE SUN'S 1.4 JDK, CLASHES WITH WL JAAS.
# IF USE 1.3, ensure jaas.jar not in /ext directory.
echo starting WLS 8 BENCHMARK...
set CFG_HOME=$DEV_HOME/dev/config/openldap
export
CLASSPATH=$CFG_HOME:/var/bea/weblogic81/server/lib/weblogic.jar:$DEV_HOME/pub/infra.jar:$DEV_HOME/pub/core.
jar:$DEV_HOME/lib/xerces.jar:$DEV_HOME/lib/ldapjdk.jar:$DEV_HOME/lib/log4j-1.2.7.jar:$DEV_HOME/lib/ejb20_ba
sic_statelessSession.jar
export JAVA_HOME=/usr/java/j2sdk1.4.2_03
export=INFRA_CFG=$DEV_HOME/test/com/infra/junit.xml
#HOST="remote-test-server"
export HOST="localhost"
```

STAGING

```
# BenchmarkTest(String testClass, int threads, int time,
boolean prime, int sleep, int cycle, boolean serialize)
$JAVA_HOME/bin/java $VM_ARG -DEJB_HOST=$HOST -Xms256m
-Xmx256m com.infra.LoadTest com.infra.EJBWorker 10 60000
false 30000 1 false
```

HOME ROLLED

Roll Your Own Tests

```
package  …;

public abstract class MyLoadTest extends Thread
{
    public abstract void setCallback (MyLoadTest t);

    public abstract void setThreadId (int id);

    …
}
```

More stuff to do

```
MyLoadTest(  )

   int threads, loops, time, cycle, sleep;
   boolean prime, serialize, static;
   Benchmark test[] = startThreads();
   do, synchronize, catch, report, log
   …
```

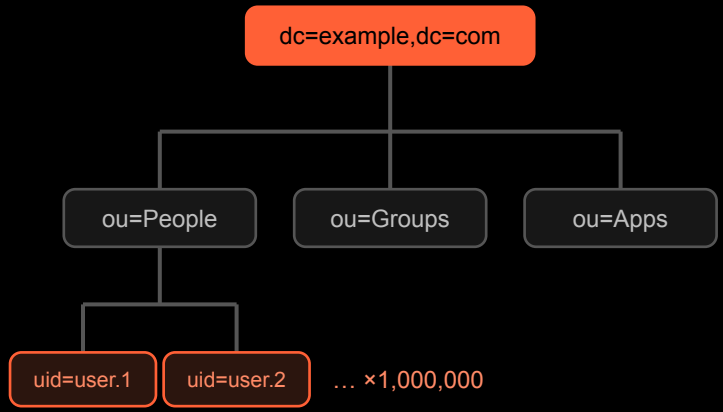
Example 2: SLAMD

Mature Java-based framework to loadtest LDAP and other systems using various network protocols including LDAP.

- Written by Neil Wilson
- [github repo](https://github.com/dirmgr/slamd)
- [Downloads](https://www.oracle.com/security/identity-management/technologies/slamd-downloads/)

SLAMD –

a short introduction



symas®

2002

Created at Sun Microsystems. Steve Shoaff proposes the idea and name; Neil A. Wilson writes most of the code.

2000s

Open-sourced under the Sun Public License. Grows to HTTP, SMTP, IMAP, POP3 and SQL jobs; 2.x ships.

~2010

Active maintenance stops. Oracle keeps SLAMD 2.1.0 available after the Sun acquisition.

Today

Revived at github.com/dirmgr/slamd: new Tomcat, UnboundID LDAP SDK inside, Java 8+.

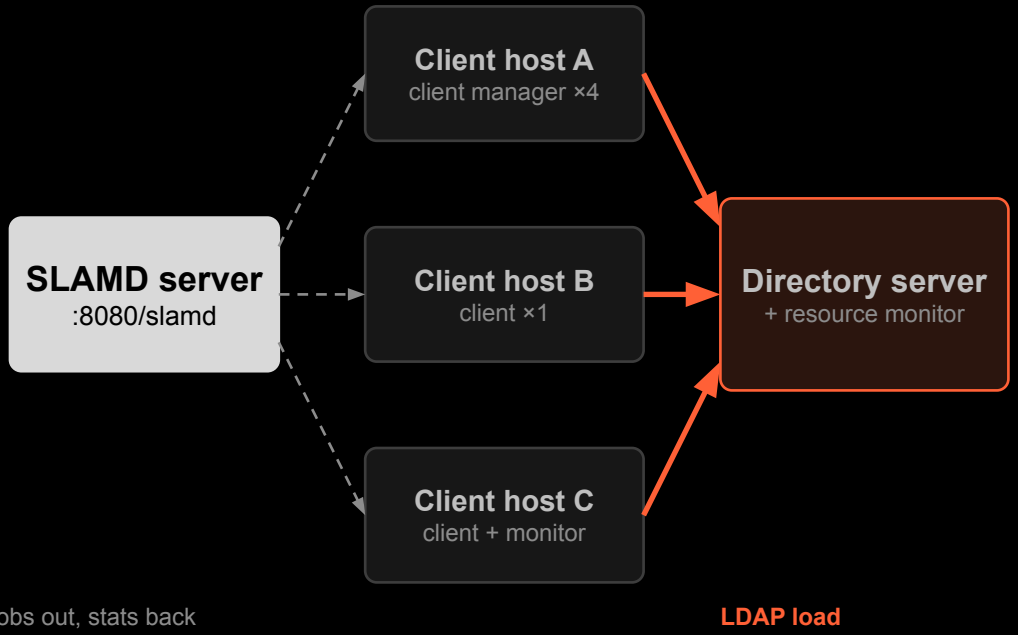
The revived build isn't backward compatible with old installs, and there's no formal release yet. Treat it as a fresh start.

What SLAMD can do

- **Distributed load**
  One server coordinates many client machines, started together.
- **Other protocols**
  Basic HTTP, SMTP, IMAP, POP3 and SQL jobs; custom jobs via a Java API.
- **Resource monitoring**
  CPU, disk, network and memory on clients and the server under test.
- **Scheduling**
  Start/stop times, dependencies, job groups, email on completion.
- **LDAP workloads**
  Search, modify, bind, add/delete, compare, modify DN, mixed loads.
- **Statistics and graphs**
  Per-interval stats, graphs, job comparison, tab-delimited export.
- **Optimizing jobs**
  Rerun a workload with more threads to find peak throughput, optionally under constraints.
- **Tools**
  LDIF generation, LDAP traffic decoding, TCP capture and replay.

Four moving parts

Server	Web app in Tomcat. Schedules jobs, collects stats, draws graphs.
Client	Runs job threads and generates the actual load.
Client manager	Creates and destroys client instances on demand from the web UI.
Resource monitor	Records CPU, disk, network and memory on any host during a job.



1. **Get the source**
   git clone https://github.com/dirmgr/slamd.git

2. **Build**
   Run build.sh (or build.bat). You get server, client and monitor-client zips in build/package.

3. **Start the server**
   Run start-slamd-server.sh and open http://host:8080/slamd.

4. **Create the database**
   One click on first launch.

5. **Start clients**
   On each load machine, after pointing them at the server.

```
# slamd-client.conf
SLAMD_ADDRESS=slamd.example.com
SLAMD_LISTEN_PORT=3000
RESTRICTED_MODE=off
AUTO_CREATE_CLIENTS=4
MAX_CLIENTS=8

# one client
./start-client.sh
# or a client manager
./start-client-manager.sh
```

Clients on many machines, in sync

One load generator often saturates before a good directory server does. SLAMD spreads the work so the client side never becomes the bottleneck.

Total load
Number of Clients × Threads per Client

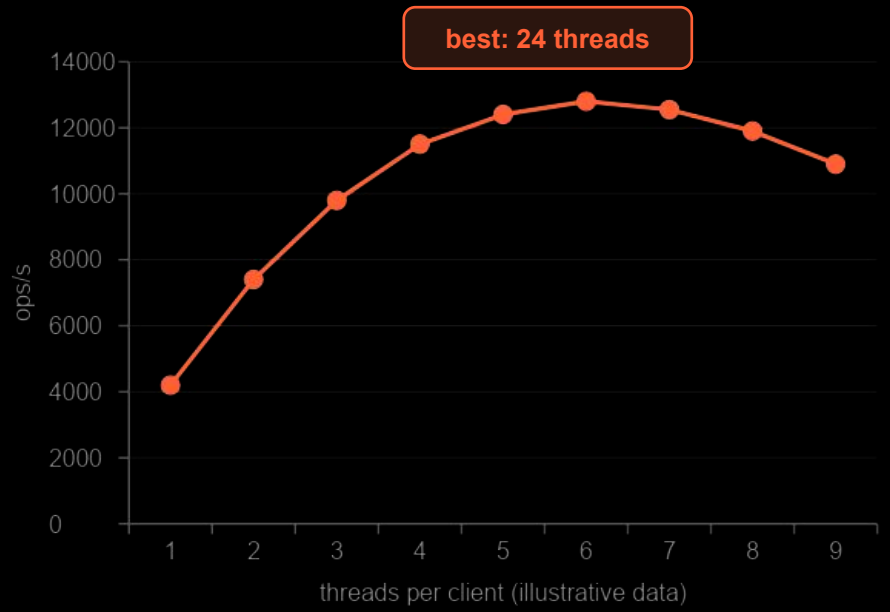
SLAMD prefers clients on different hosts and starts them as close to simultaneously as it can.

Use Specific Clients	Pin a job to named hosts (advanced options).
Restricted mode	A client only runs jobs that explicitly ask for it.
Wait for clients	Hold the job until enough clients connect, instead of failing.
Monitor clients	Record client-side CPU to prove load generators weren't saturated.
Client manager	Add or remove client instances from the Server Status page.

Optimize for peak performance

Min / max threads	Range to explore per client
Thread increment	Step between iterations
Non-improving limit	Stop after N iterations with no gain
Re-run best	Confirm the winner with a longer run

Algorithms can optimize a single statistic, or do it under a constraint: CPU utilization, a secondary statistic like response time, or replication latency.



3 non-improving → stop

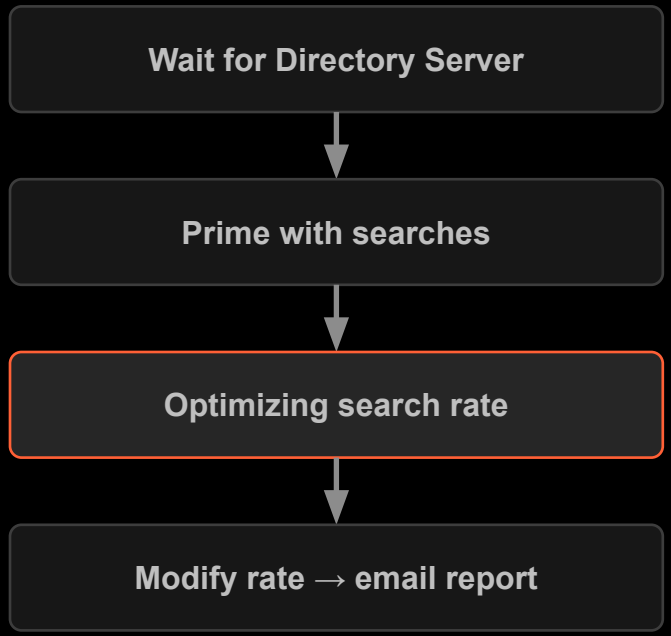
Chain jobs into a test plan

Job dependencies
A job waits for the listed jobs (optimizing jobs included) to finish, even once its start time arrives.

Interdependent copies
Schedule N copies that run one after another.

Job groups
Bundle jobs and optimizing jobs with shared parameters and schedule the whole plan at once.

Notify on completion
Email a results summary with a link back to the job.



The demo setup

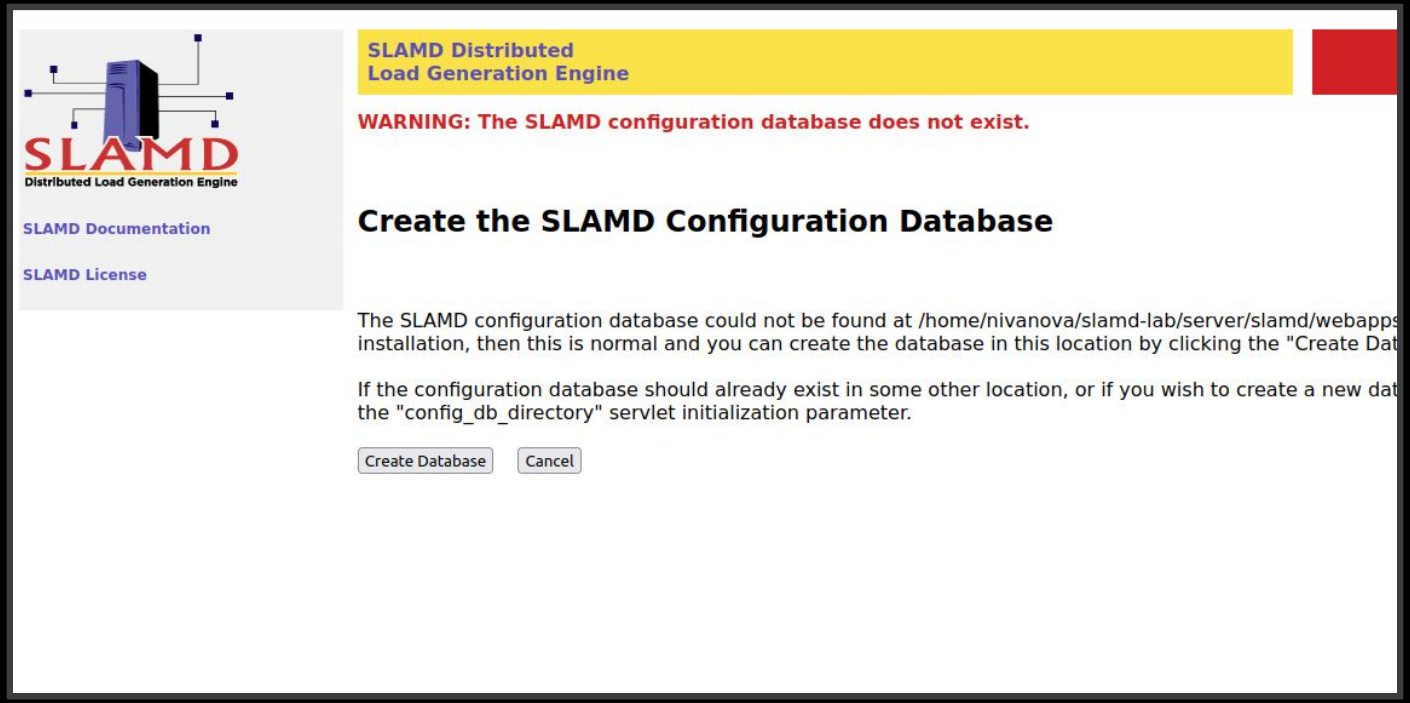
Machine	Role	Lab address	vCPUs	RAM
Host	SLAMD server (web UI :8080, clients :3000–3003)	`192.168.100.1`	–	–
ldap	OpenLDAP: mdb for dc=example,dc=com + back-asyncmeta for o=meta	`192.168.100.10`	2	4 GB
client1	SLAMD client	`192.168.100.11`	2	4 GB
client2	SLAMD client	`192.168.100.12`	2	4 GB
ldap-east	asyncmeta target: ou=east,o=meta (users 1–5000)	`192.168.100.20`	1	2 GB
ldap-west	asyncmeta target: ou=west,o=meta (users 5001–10000)	`192.168.100.21`	1	2 GB

All VMs: Ubuntu 26.04 LTS on QEMU/KVM, connected by a private bridge (br-slamd, 192.168.100.0/24).

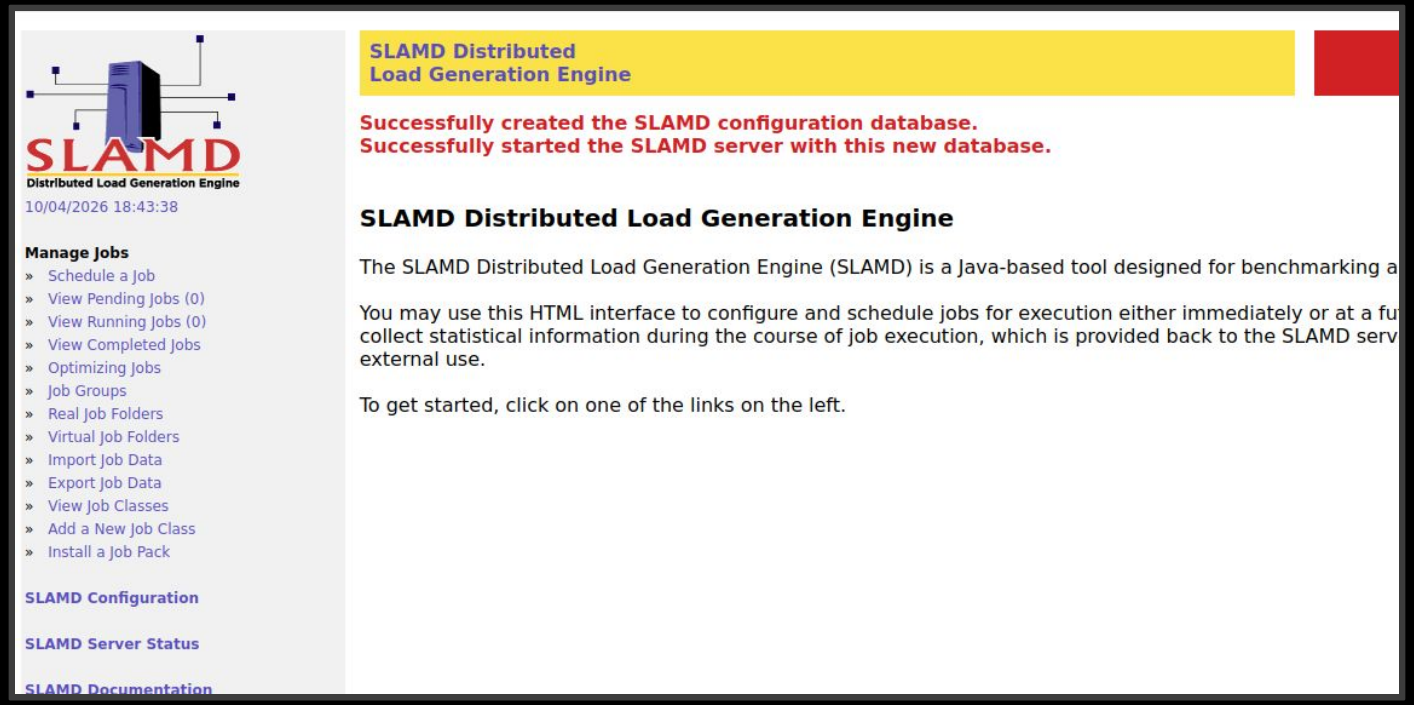
Setting up the SLAMD server

```
# Clone and build SLAMD (must end with BUILD SUCCESSFUL)
cd ~/slamd-lab
git clone https://github.com/dirmgr/slamd.git
cd slamd
./build.sh
ls build/package

# Install the server outside the build tree and start it
mkdir -p ~/slamd-lab/server
unzip -q ~/slamd-lab/slamd/build/package/slamd-2*.zip -d ~/slamd-lab/server
mkdir -p ~/slamd-lab/server/slamd/temp
~/slamd-lab/server/slamd/start-slamd-server.sh

# Now open http://localhost:8080/slamd in a browser and click "Create Database"

# Check the server listens on 3000-3003 and 8080
ss -ltn | grep -E ':(3000|3001|3002|3003|8080) '
```

First start: create the database



Database created: the SLAMD console



Starting the SLAMD clients

1. Copy the client, or build SLAMD, on each client machine
2. Set `SLAMD_ADDRESS` in slamd-client.conf to the server address
3. Run `./start-client.sh`

A simple example – an LDAP search-rate job

Choose **Schedule a Job → LDAP → Basic Search Rate**, then fill in the target and workload.

- Other LDAP jobs cover modify, bind, search-and-bind, add-and-delete, compare, modify DN and mixed loads.
- Value patterns spread requests across your whole data set instead of one cached entry.
- Click Test Job Parameters first so the server can confirm it can connect, bind and find the base DN.

Description	Demo search rate
Duration	1 minute, 15 seconds
Number of Clients	2
Threads per Client	8
Directory Server Address	192.168.100.10
Directory Server Port	389
Security Method	None
Bind DN	cn=admin,dc=example,dc=com
Bind Password	secret
Search Base DN Pattern	ou=People,dc=example,dc=com
Search Scope	wholeSubtree
Search Filter Pattern	(uid=user.[1-10000])
Attributes to Return	cn, mail
Warm-Up Duration	10 seconds
Cool-Down Duration	5 seconds

Asynchronous search through back-asyncmeta

Choose **Schedule a Job → LDAP → Asynchronous Search Rate**, then fill in the target and workload.

- back-asyncmeta sends each search to both targets in parallel and merges the answers.
- One thread per client: the load comes from many requests in flight on each connection.
- Bind anonymously; the admin DN belongs to the other database.
- Test Job Parameters reports a missing base entry. That is expected: o=meta is a virtual root.

Description	Async search via back-asyncmeta
Duration	1 minute, 15 seconds
Number of Clients	2
Server Addresses and Ports	192.168.100.10:389
Security Method	None
Bind DN / Password	(empty, anonymous)
Search Base	o=meta
Search Scope	Whole Subtree
Attributes to Return	cn, mail
Search Filter 1	(uid=user.[1-5000])
Search Filter 2	(uid=user.[5001-10000])
Filter 1 Percentage	50
Connections per Client	2
Max Outstanding Requests	50
Warm Up Time	10
Cool Down Time	5

Apache Jmeter Overview

The **Apache JMeter™** application is open source software, a 100% pure Java application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

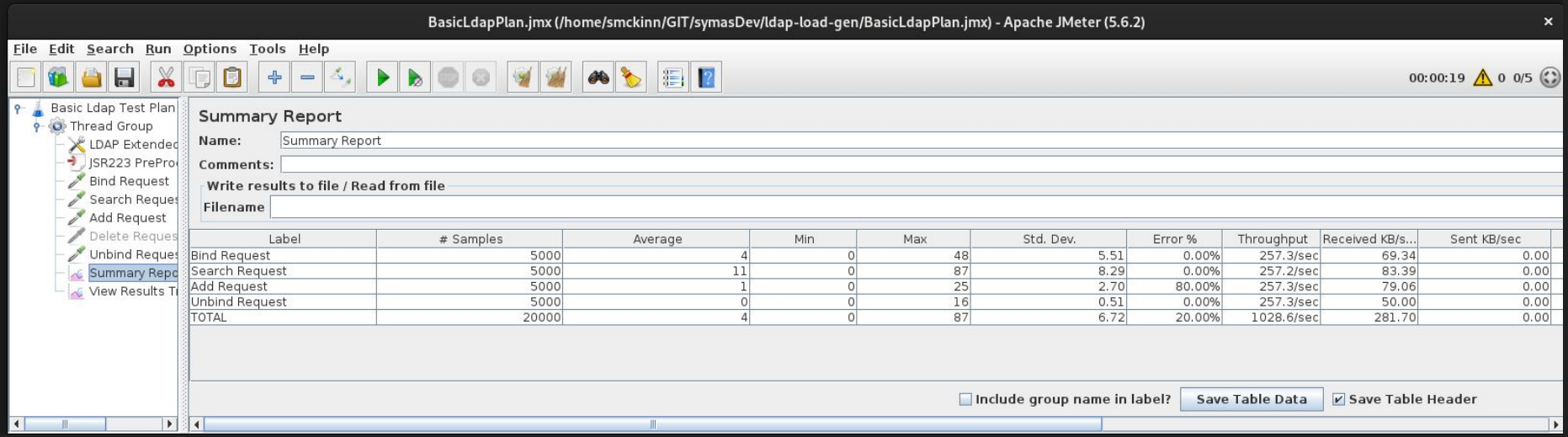
- [Project](https://jmeter.apache.org/)
- [Book](https://leanpub.com/master-jmeter-from-load-test-to-devops/)

More on Apache Jmeter

- Started as internal project to test Apache JServ (a precursor to Apache Tomcat)
- Became an Apache TLP in 2011
- Tests may be run from a GUI, command line, or other processes.
- Works out-of-box with common workloads.
- Provides a framework for creating specialized use cases.
- Scales to large loads with a distributed mode.
- Reports to CSV, XML, HTML.
- Works with various monitoring tools.
- Test dev simplified with scripting languages like Apache Groovy.

Example 3: Apache Jmeter Out-of-Box

- CRUD
- Works for basic use cases.
- Good for building test plans to run with its CLI



Creating Custom Jmeter Tests

1. Create new script: *myLoadTest.jmx*
   a. Set the parameters in *jmeterTestPlan*
   b. threads, loops, duration, throughput, …
2. Create new sampler: MyLoadTest.java
   a. Extend *AbstractJavaSamplerClient*
   b. Control results with *JavaSamplerContext*
3. Add the JMeter Maven plug-in to the pom file.

Creating Custom Jmeter Tasks

```
public class MyLoadTest extends AbstractJavaSamplerClient
{
  public void setupTest(JavaSamplerContext context)
  {
  }
  public void teardownTest(JavaSamplerContext context)
  {
  }
  public SampleResult runTest(JavaSamplerContext context)
  {
  }
}
```

Knobs: Threads, Loops

```
<ThreadGroup guiclass="ThreadGroupGui" testclass="ThreadGroup" testname="Test Ldap" enabled="true">

…

<stringProp name="LoopController.loops">100</stringProp>

<stringProp name="ThreadGroup.num_threads">10</stringProp>

<stringProp name="ThreadGroup.ramp_time">10</stringProp>
<boolProp name="ThreadGroup.scheduler">false</boolProp>
<stringProp name="ThreadGroup.duration"></stringProp>
<stringProp name="ThreadGroup.delay">1</stringProp>
</ThreadGroup>
```

10 THREADS X 100 LOOPS = 1000 TIMES

Knobs: Throughput

```
<ConstantThroughputTimer guiclass="TestBeanGUI"
testclass="ConstantThroughputTimer" testname="MyTest" enabled="true">
        <intProp name="calcMode">3</intProp>
        <doubleProp>

        <name>throughput</name>
        <value>600</value>

        </doubleProp>
</ConstantThroughputTimer>
```

<-- Per Minute

Example 4: Bash Loader

- https://git.symas.net/symas-public/bash-load-meter
- Basic sample
- Tests using Bash
  - System Tests: Basic commands
  - LDAPv3 operations
    - CRUD
    - User and Group
    - Standard data format

Example 5: Apache Fortress Load Tests

- https://github.com/apache/directory-fortress-core/blob/master/README-LOAD-TESTING.md
- Intermediate sample
- Tests using
  - LDAPv3 operations
    - CRUD
    - RBAC
    - Apache Fortress data format

Example 6: LDAP Load Generator

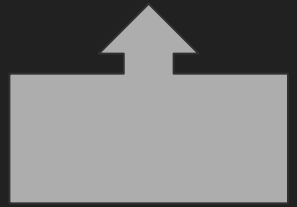
- https://git.symas.net/symas-public/ldap-load-gen
- Advanced Test cases
- Apache LDAP API & OpenLDAP
  - CRUD ops
  - Varied data formats
  - Advanced use cases
    - dups, mod conflict, read after update
    - password policies, controls, dynamic modules, sasl binds
    - autogroup, memberof, dynlist, lastbind, replication

Apache Maven Jmeter Plug-in Overview

- Maven plugin that enables JMeter integration into Java project.
- Simplifies deployment as it manages dependencies.
- The tests cases are built directly into the project.
- Becomes a simple matter to run as part of a release.

Advantage of Apache JMeter running as Plug-In

- Maven executes the load tests like a normal test
- Test code handles interaction with backend
- Tests use existing Dependencies, configs, artifacts. (don't need to stage)
- JMeter framework handles everything else
- Perfect pairing of concerns



Apache Maven Jmeter Plug-in More Info

- Sponsor
  - http://jmeter.lazerycode.com/
- Source
  - https://github.com/jmeter-maven-plugin/jmeter-maven-plugin
- Documentation
  - https://github.com/jmeter-maven-plugin/jmeter-maven-plugin/wiki/Basic-Configuration
  - https://github.com/jmeter-maven-plugin/jmeter-maven-plugin/wiki/Advanced-Configuration

Apache Jmeter Plugged Into Maven Pom

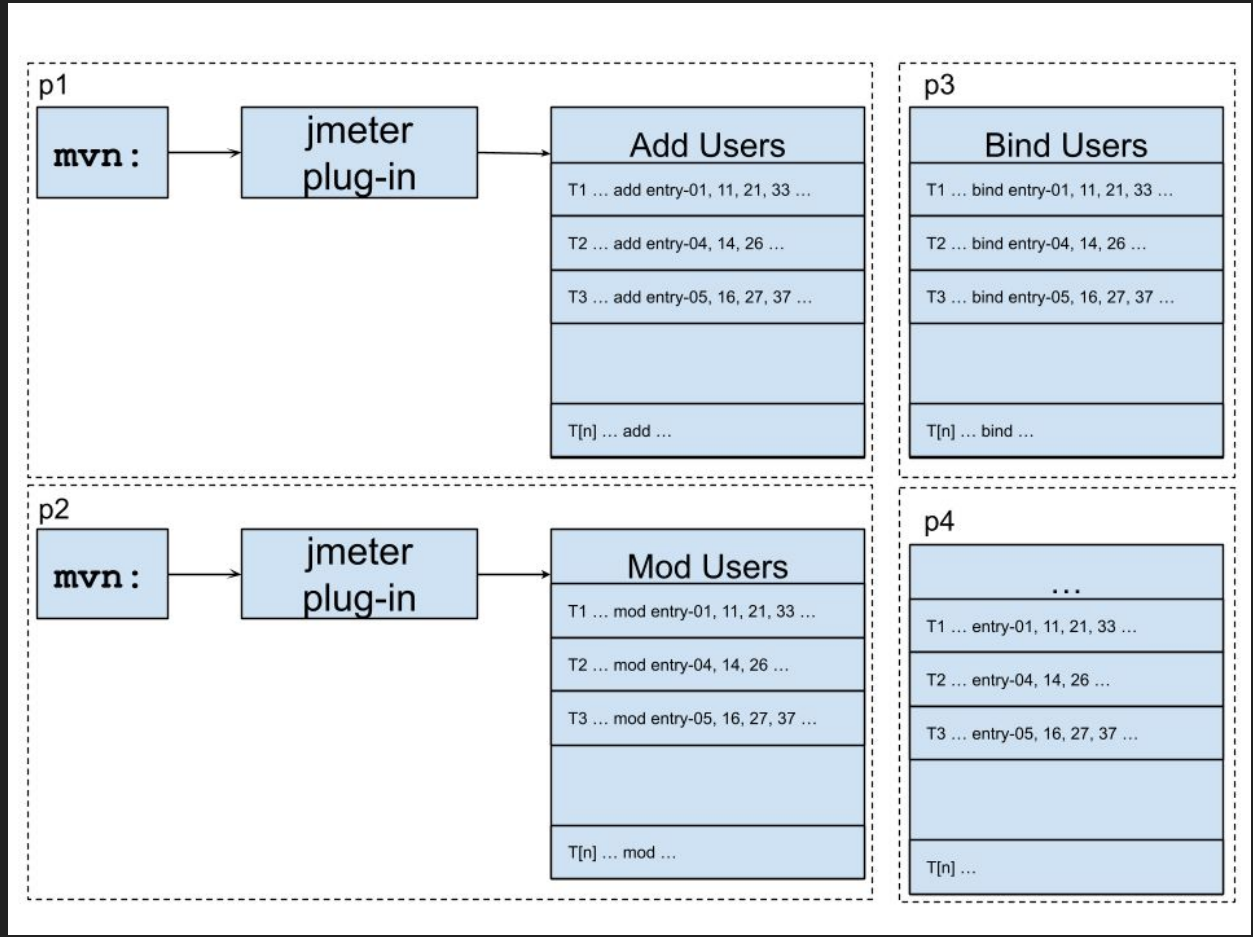
```
<profile>
  <id>load</id>
  <build>
    <plugins>
      <plugin>
        <groupId>com.lazerycode.jmeter</groupId>
        <artifactId>jmeter-maven-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <testFilesIncluded>
            <jMeterTestFile>${type}.jmx</jMeterTestFile>
          </testFilesIncluded>
          <propertiesSystem>
            <name>${name}</name>
                …
          </propertiesSystem>
        </configuration>
      <plugin>
```

<- Requires Maven 3.9

Apache JMeter via Maven plugin

```
mvn verify -Pload -Dtype=add
-Dqualifier=T3a
-Dfilter='(id=infra01-T3a*)'

mvn verify -Pload -Dtype=search
-Dbase='dc=infra,dc=com'
-Dfilter='(member=*)' -Dreverselookup=true
```

Running Jmeter Tests with maven



Test Design

- Generate data based on a passed in value + global counter
  - run add val=”FOO1”
- Total iterations = num_threads * num_loops
  - 10 threads * 1000 loops = 10K iterations
  - [FOO1-1, FOO1-2, FOO1-3 … FOO1-10000]
- Repeatable
  - run add val=”FOO1”
  - run add val=”FOO2”
- Reversible
  - run add val=”FOO1”
  - run del val=”FOO1”

Test Design

```
#!/bin/bash
for (( i=0; i<"$REPEAT"; i++ ))
do
  for (( j=0; j < $numops; j++))
  do
    for (( k=$START; k<="$END"; k++ ))
    do
      mvn verify -Pload -Dtype=${op} -Dqualifier=R${k}H${HOSTNUM} $args
    done
  done
done
```

Test Design

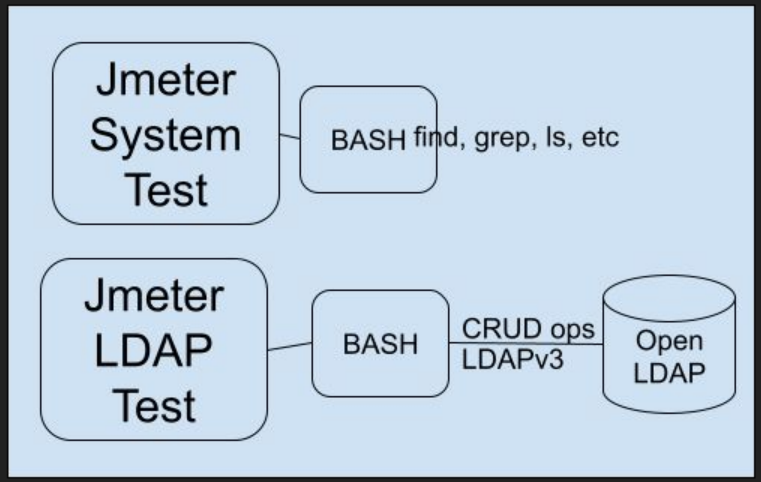
```
# Soak test 7 days:
load.sh 1 1 5 20
"addGroup:addUser:assignUser:checkU
ser:delUser:delGroup"
```

Demos

- Basic
- Intermediate
- Advanced

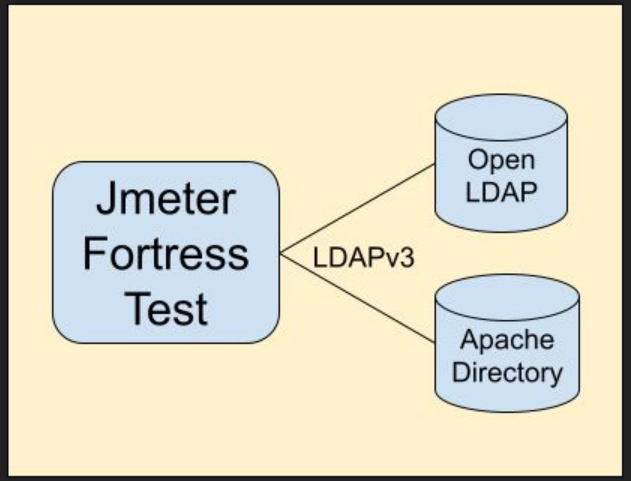
Basic

- Bash Load Meter
- https://git.symas.net/symas-public/bash-load-meter



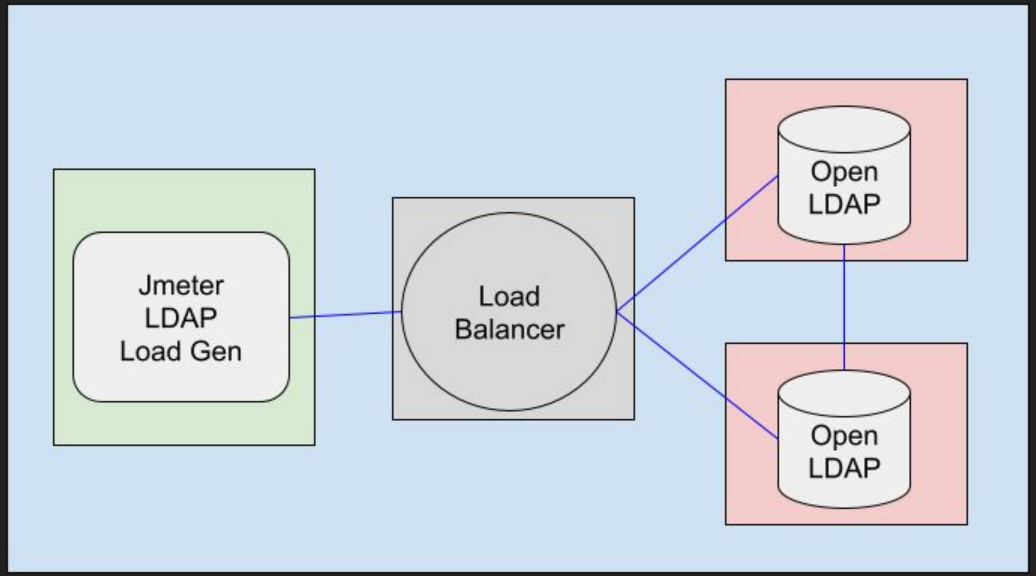
Intermediate

- Apache Fortress Load Tests
- https://github.com/apache/directory-fortress-core/blob/master/README-LOAD-TESTING.md



Advanced

- LDAP Load Generator
- https://gitlab.symas.net/symas-public/ldap-load-gen



Example 7: Loadgen (coming soon)

```
$ ./loadgen --duration 5m --numconns 64 --threads $(nproc) --inflight 15 'ldap://:2389/'
Connections established
Done.
Test duration was 5m with 2048 connections across 32 threads
Total traffic: sent/received 1.48GiB/613MiB, on average that's 5.07MiB/s sent 2.05MiB/s received
Processed 33.9M operations (+0 other pdus), that's 112k per second (112k pdu/s total)
First response arrived between 465us and 1.50s (averaging at: 271.68ms, sd: 214.62ms)
Histogram:
        0us-0us: 0, 1us-1us: 0, 2us-3us: 0, 4us-7us: 0, 8us-15us: 0,
        16us-31us: 0, 32us-63us: 0, 64us-127us: 0, 128us-255us: 0, 256us-511us: 5,
        512us-1.02ms: 81, 1.02ms-2.05ms: 721, 2.05ms-4.09ms: 3991, 4.10ms-8.19ms: 22.9k, 8.19ms-16.38ms: 144k,
        16.38ms-32.77ms: 965k, 32.77ms-65.54ms: 3.22M, 65.54ms-131.07ms: 6.06M, 131.07ms-262.14ms: 9.53M, 262.14ms-524.29ms: 9.70M,
        524.29ms-1000.00ms: 3.88M, 1s-1s: 365k, 2s-3s: 0, 4s-7s: 0, 8s-15s: 0,
        16s-31s: 0, 32s-63s: 0, 64s-127s: 0, 128s-255s: 0, 256s-300s: 0,

Time to finish an operation was between 465us and 1.50s (averaging at: 271.68ms, sd: 214.62ms)
Histogram:
        0us-0us: 0, 1us-1us: 0, 2us-3us: 0, 4us-7us: 0, 8us-15us: 0,
        16us-31us: 0, 32us-63us: 0, 64us-127us: 0, 128us-255us: 0, 256us-511us: 5,
        512us-1.02ms: 81, 1.02ms-2.05ms: 721, 2.05ms-4.09ms: 3991, 4.10ms-8.19ms: 22.9k, 8.19ms-16.38ms: 144k,
        16.38ms-32.77ms: 965k, 32.77ms-65.54ms: 3.22M, 65.54ms-131.07ms: 6.06M, 131.07ms-262.14ms: 9.53M, 262.14ms-524.29ms: 9.70M,
        524.29ms-1000.00ms: 3.88M, 1s-1s: 365k, 2s-3s: 0, 4s-7s: 0, 8s-15s: 0,
        16s-31s: 0, 32s-63s: 0, 64s-127s: 0, 128s-255s: 0, 256s-300s: 0,
Overhead was 341.62% (2m 2s user, 15m 1s sys)
```

Parting thoughts

- Apache JMeter to practice load testing
- Maven JMeter plugin's to integrate with load testing

Questions