Marketplace for self-publishers

(study project)

- github.com/elegantsignal/marketplace
- marketplace.elegantsignal.com

> contact with me if you want to see the project alive

Own Goals

- use all types of relationships in database
- make complex database queries
- file upload and serving
- hibernate search
- YAML/JSON serialization and deserialization
- JSON API
- something new in infrastructure (caddy proxy server)

Infrastructure

- caddy + tomcat + spring + postgresql
- docker + docker-compose
- ansible pipline

Docker images are builded on the server.
I have two reasons for that — I have slow internet and long build process by CI-server.
Don't do that on real project.

> SSH agent forwarding — grate practice!

Dockerfile

- don't use build plugins — use multistage build.
- "yes", I know, I should use embedded Tomcat. Next time.

```
FROM maven:3.6.3-jdk-8 AS builder
WORKDIR /usr/src/app

COPY pom.xml .
COPY dao/pom.xml dao/pom.xml
COPY service/pom.xml service/pom.xml
COPY web/pom.xml web/pom.xml
COPY web/src/main/webapp/WEB-INF/web.xml web/src/main/webapp/WEB-INF/web.xml
RUN  mvn package && mvn clean

COPY ./ .
RUN  mvn package -DskipTests



FROM tomcat:9-jre8-alpine
WORKDIR ${CATALINA_HOME}

RUN  rm -rf webapps/*
COPY config/tomcat/server.xml config/tomcat/context.xml conf/
COPY --from=builder /usr/src/app/web/target/${APP_NAME}.war webapps/ROOT.war

RUN  addgroup www-data && \
     adduser -D -H -u 1000 -s /bin/bash www-data -G www-data && \
     chown -R www-data:www-data webapps temp

USER www-data
CMD  ["catalina.sh", "run"]
```

> `.war` size **47Mb**; docker image size **200Mb**; builder image > **800Mb**.

Futures

user's actions

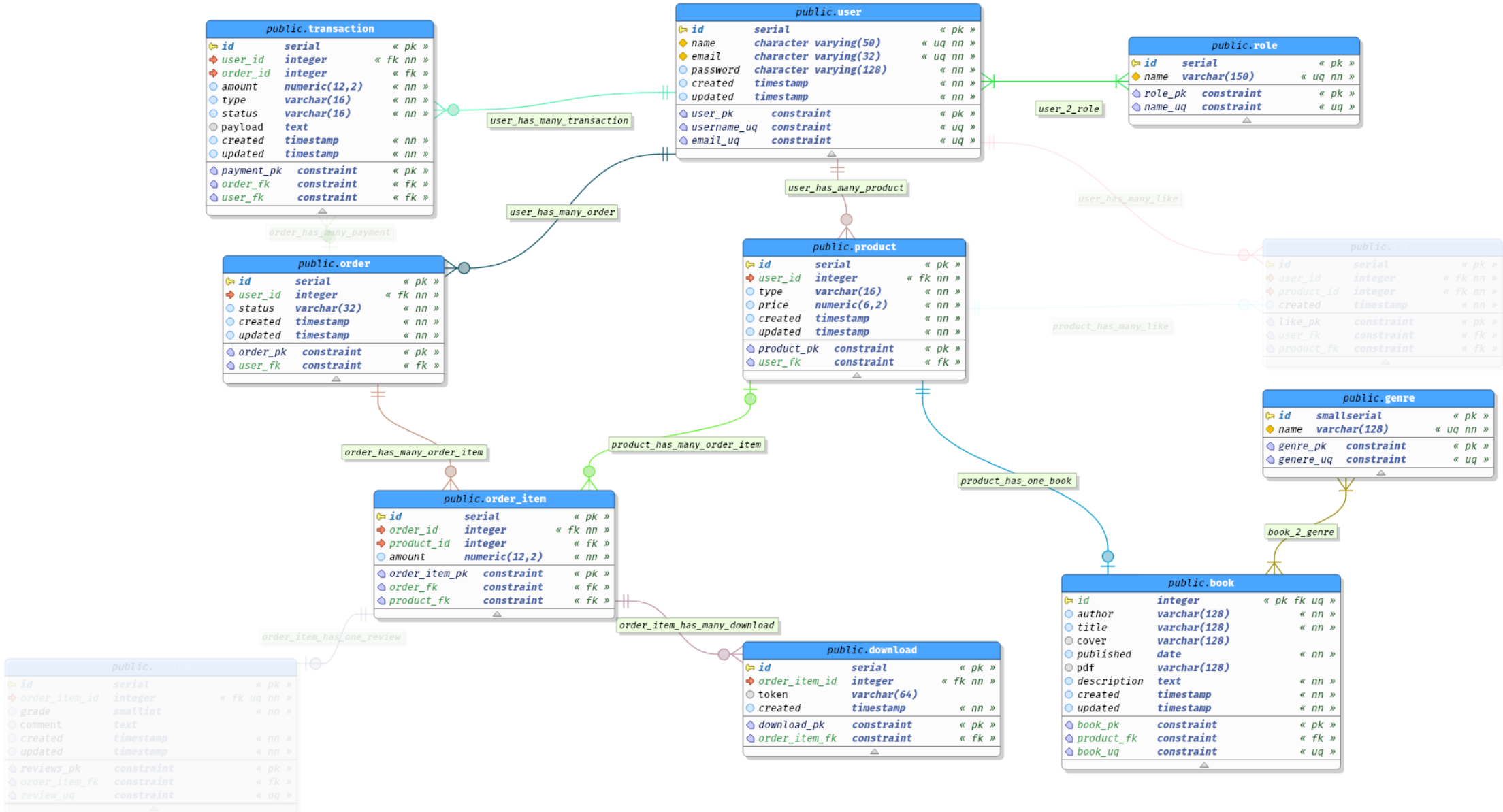
- create goods (books) and make purchases
- download bought books via access link
- withdraw profit

futures

- product's files shouldn't be public available
- user's balance calculated on the fly
- type of uploaded files should be checked
- generate dummy data from `seed.yml` file

More about finance side

- product price may be change without negative consequence — `order_item` should store price.
- "order" should be builded from two entities — `product` + `book` (certain implementation).
- $balance = \sum(order\_item.amount) - \sum(transaction.amount)$


public.transaction
id serial « pk »
user_id integer « fk nn »
order_id integer « fk »
amount numeric(12,2) « nn »
type varchar(16) « nn »
status varchar(16) « nn »
payload text
created timestamp « nn »
updated timestamp « nn »
payment_pk constraint « pk »
order_fk constraint « fk »
user_fk constraint « fk »
public.user
id serial « pk »
name character varying(50) « uq nn »
email character varying(32) « uq nn »
password character varying(128) « nn »
created timestamp « nn »
updated timestamp « nn »
user_pk constraint « pk »
username_uq constraint « uq »
email_uq constraint « uq »
public.role
id serial « pk »
name varchar(150) « uq nn »
role_pk constraint « pk »
name_uq constraint « uq »
user_has_many_transaction
user_2_role
user_has_many_order
user_has_many_product
user_has_many_like
order_has_many_payment
public.order
id serial « pk »
user_id integer « fk nn »
status varchar(32) « nn »
created timestamp « nn »
updated timestamp « nn »
order_pk constraint « pk »
user_fk constraint « fk »
public.product
id serial « pk »
user_id integer « fk nn »
type varchar(16) « nn »
price numeric(6,2) « nn »
created timestamp « nn »
updated timestamp « nn »
product_pk constraint « pk »
user_fk constraint « fk »
public.
id serial « pk »
user_id integer « fk nn »
product_id integer « fk nn »
created timestamp « nn »
like_pk constraint « pk »
user_fk constraint « fk »
product_fk constraint « fk »
product_has_many_like
public.genre
id smallserial « pk »
name varchar(128) « uq nn »
genre_pk constraint « pk »
genere_uq constraint « uq »
order_has_many_order_item
product_has_many_order_item
product_has_one_book
book_2_genre
public.order_item
id serial « pk »
order_id integer « fk nn »
product_id integer « fk »
amount numeric(12,2) « nn »
order_item_pk constraint « pk »
order_fk constraint « fk »
product_fk constraint « fk »
public.book
id integer « pk fk uq »
author varchar(128) « nn »
title varchar(128) « nn »
cover varchar(128)
published date « nn »
pdf varchar(128)
description text « nn »
created timestamp « nn »
updated timestamp « nn »
book_pk constraint « pk »
product_fk constraint « fk »
book_uq constraint « uq »
order_item_has_many_download
order_item_has_one_review
public.download
id serial « pk »
order_item_id integer « fk nn »
token varchar(64)
created timestamp « nn »
download_pk constraint « pk »
order_item_fk constraint « fk »
public.
id serial « pk »
order_item_id integer « fk uq nn »
grade smallint « nn »
comment text
created timestamp « nn »
updated timestamp « nn »
reviews_pk constraint « pk »
order_item_fk constraint « fk »
review_uq constraint « uq »

Seed data with `YAML`

```yaml
- email: alice@example.com
  name: alice
  password: "u2PX"
  roles:
    - consumer
    - supplier

- email: bob@example.com
  name: bob
  password: "u2PX"
  roles:
    - consumer
```

```java
protected <T> void createUser(final Map<String, T> userData) {
  final IUser user = userService.createEntity()
    .setName((String) userData.get("name"))
    .setEmail((String) userData.get("email"))
    .setPassword((String) userData.get("password"));

  final Set<IRole> roleSet = new HashSet<>();
  final List<String> userRoles = (List<String>) userData.get("roles");
  userRoles.forEach(roleName -> roleSet.add(getOrCreateRole(roleName)));
  user.setRole(roleSet);
  userService.save(user);
}
```

File upload

1. Save file to `/tmp`
2. Identify file type with Apache Tika
3. Rename file based on parent entity rules
4. Update entity

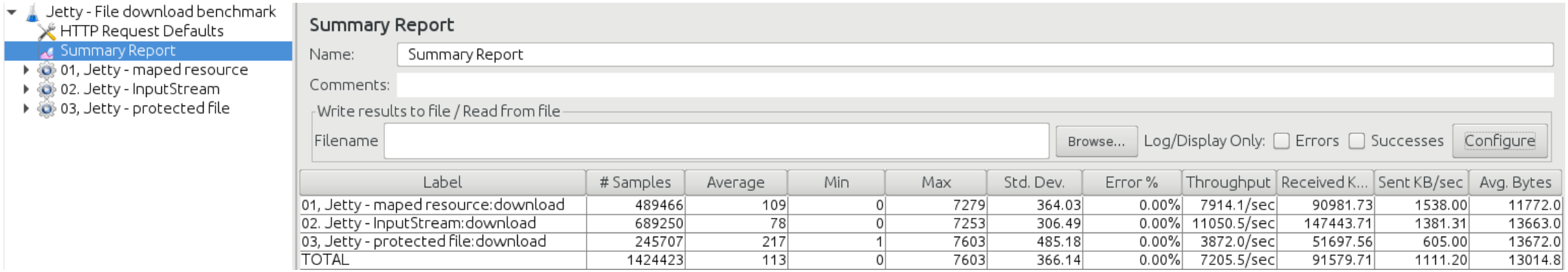
Benchmark of file serving

Three test for **Jetty** and **Tomcat** (jmeter + visalvm)

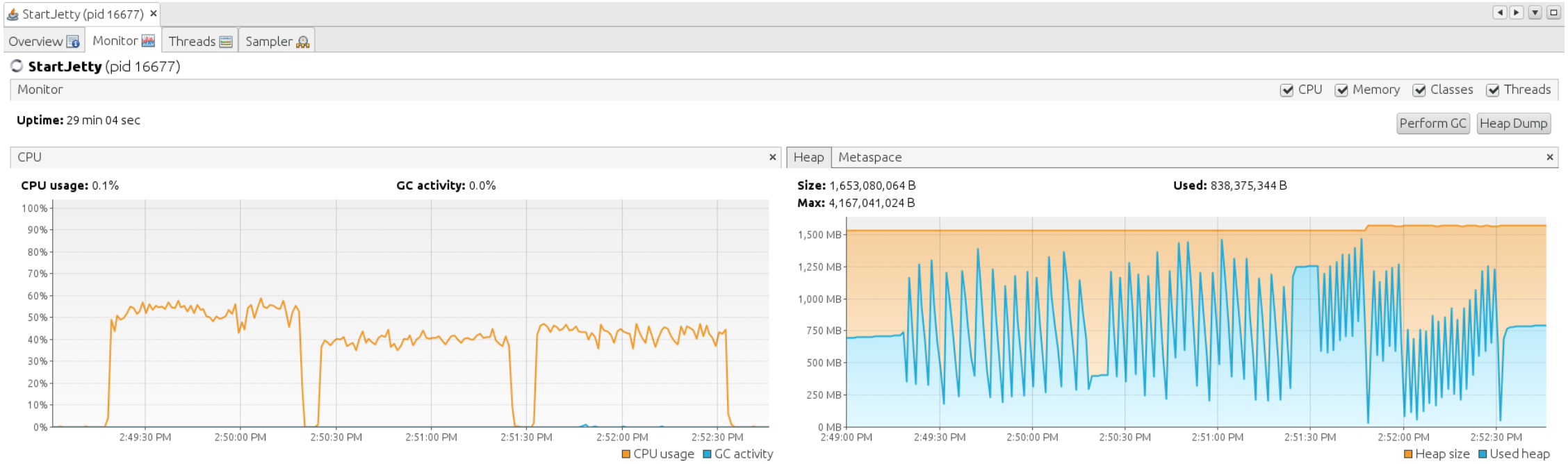
1. serve by servlet
2. serve by app
3. serve private file by access link

> Spoiler: **Tomcat** do the job better then **Jetty**, we have **Nginx** for similar tasks.

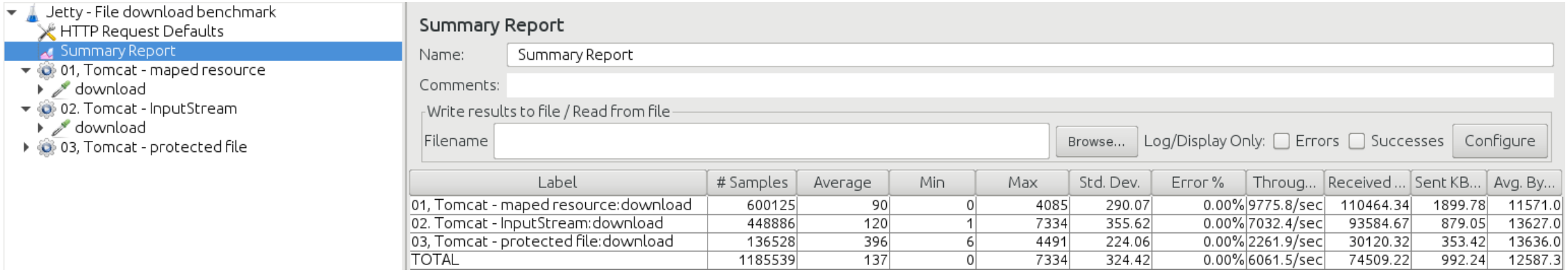
Jetty — download benchmark (public vs static files)



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
01, Jetty - maped resource:download	489466	109	0	7279	364.03	0.00%	7914.1/sec	90981.73	1538.00	11772.0
02. Jetty - InputStream:download	689250	78	0	7253	306.49	0.00%	11050.5/sec	147443.71	1381.31	13663.0
03, Jetty - protected file:download	245707	217	1	7603	485.18	0.00%	3872.0/sec	51697.56	605.00	13672.0
TOTAL	1424423	113	0	7603	366.14	0.00%	7205.5/sec	91579.71	1111.20	13014.8



Tomcat — download benchmark (public vs static files)



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throug...	Received ...	Sent KB...	Avg. By...
01, Tomcat - maped resource:download	600125	90	0	4085	290.07	0.00%	9775.8/sec	110464.34	1899.78	11571.0
02. Tomcat - InputStream:download	448886	120	1	7334	355.62	0.00%	7032.4/sec	93584.67	879.05	13627.0
03, Tomcat - protected file:download	136528	396	6	4491	224.06	0.00%	2261.9/sec	30120.32	353.42	13636.0
TOTAL	1185539	137	0	7334	324.42	0.00%	6061.5/sec	74509.22	992.24	12587.3