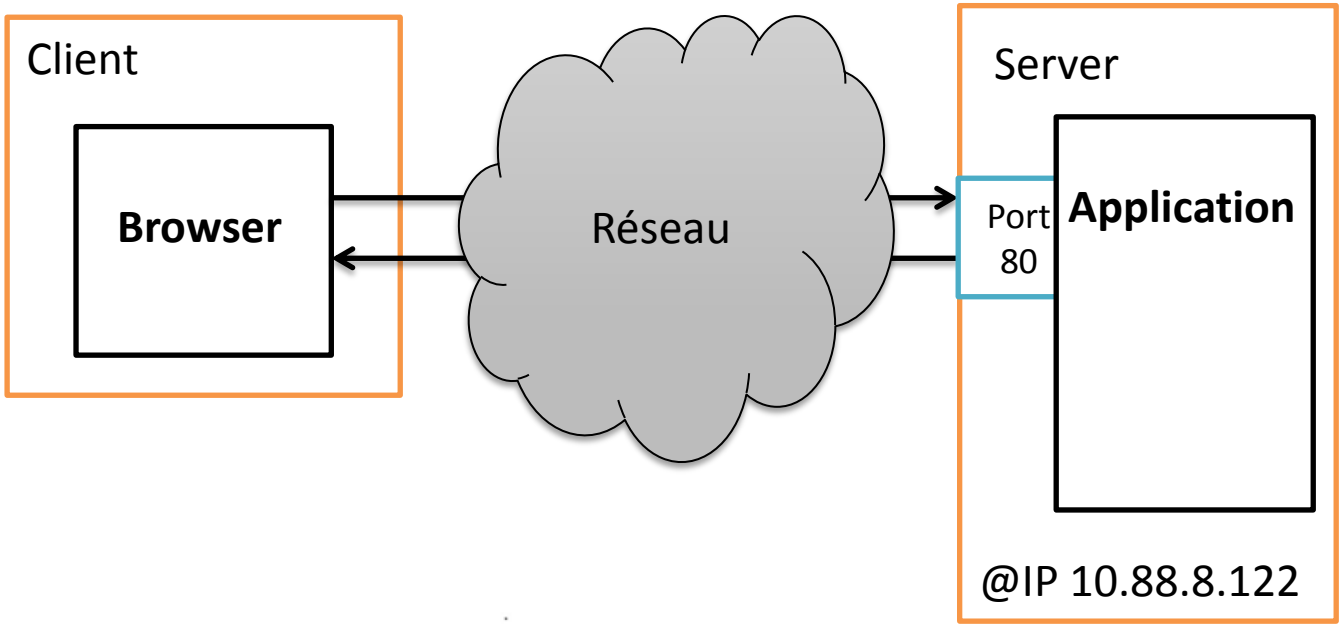
JAVA/JEE

Tomcat et les servlets

Rappel: le modèle client/serveur



Implémentation du web

Une application web s'appuie sur plusieurs couches du modèles ISO pour fonctionner

	ISO	Implémentation	Usage
7	Application	HTTP	Interagir avec une ressource
		DNS	Retrouver l'IP d'un nom d'hôte
4	Transport	TCP	Adresser une application
3	Réseau	IP	Adresser une machine

Pour accéder à une ressource

protocol://[hostname ou ip]:[port][/path]

Exemples:
https://epsi.com/quelquechose ou http://11.222.92.10:8040/qulequechose

Note: Historiquement les ressources web étaient stockées dans le répertoire www des serveurs

Le protocole HTTP

- **H**yper **T**ext **T**ransfer **P**rotocol
- RFC 2616 (HTTP 1.1, 1999)
- Version sécurisée HTTP**S** (RFC 2818, 2000)
- Port d'écoute standard: 80 ou 443 (SSL/TLS)
- HTTP2 en cours de spécification

Le protocole HTTP

- Une requête est composée des informations suivantes:
  - URL
  - Méthode
  - En-têtes (headers)
  - Contenu

- Certains headers sont enrichis automatiquement par les navigateurs/devices (ex. User-Agent)

Verbes HTTP

Le protocole HTTP spécifie plusieurs méthodes

Verbe	Usage
GET	Récupération
POST	Création
PUT	Mise à jour
DELETE	Suppression
OPTIONS	Options supportées (ex. encodage)
HEAD	Meta réponse (headers uniquement)

⇒ Historiquement GET et POST sont les actions les plus utilisées mais leur valeur sémantique a longtemps été ignoré

⇒ La philosophie **REST** tend à aller vers plus de conformité

Les serveurs web

- Le premier serveur web a été écrit en 1990 par **Tim Berners Lee**
- On distingue les serveur web HTTP qui servent du contenu statique ou servent de reverse proxy
  - ⇒ Peu d'intelligence
  - ⇒ Sert des fichiers HTML, CSS, images, documents…
  - ⇒ Principaux serveurs HTTP: Apache (hors mod_php), nginx, lighthttpd, IIS…
- Des serveurs web applicatifs qui servent du contenu dynamique
  - ⇒ Les requêtes sont interprétés et la réponse envoyée au client dépend des paramètres reçus
  - ⇒ Principaux serveurs applicatifs JEE: **Tomcat**, Jetty, node.JS, WEBrick…

Conteneur d'application

Les serveurs web applicatifs JEE sont également appelés **conteneur d'application**
⇒ Ils implémentent les standards EE et permettent d'exécuter des servlets
⇒ Ils offrent un contexte d'exécution aux applications

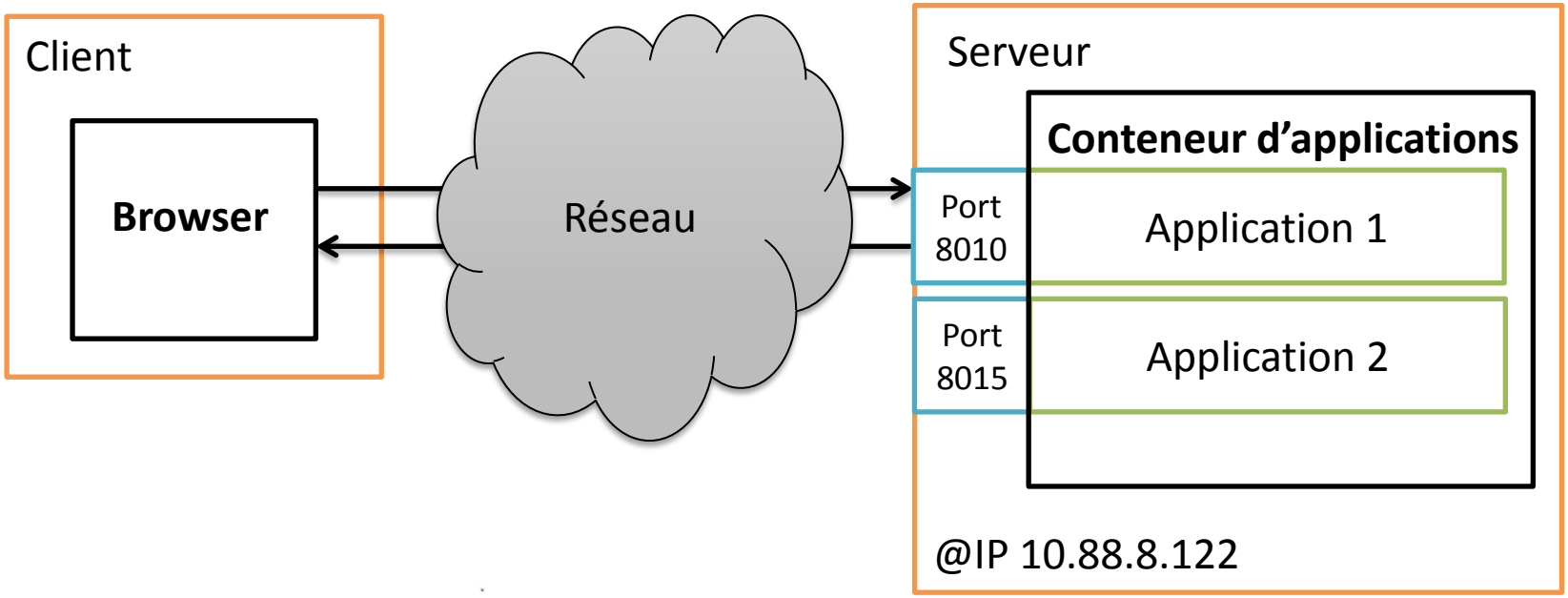
« Un conteneur d'application est une application Java qui écoute sur un ou plusieurs port et redirige les requêtes reçues vers des applications web qu'il héberge »

/!\ Deux applications web hébergées dans le même conteneur partagent donc la même JVM

Principaux conteneurs :

- **Tomcat**
- Jetty
- Jboss
- GlassFish

Conteneur d'applications

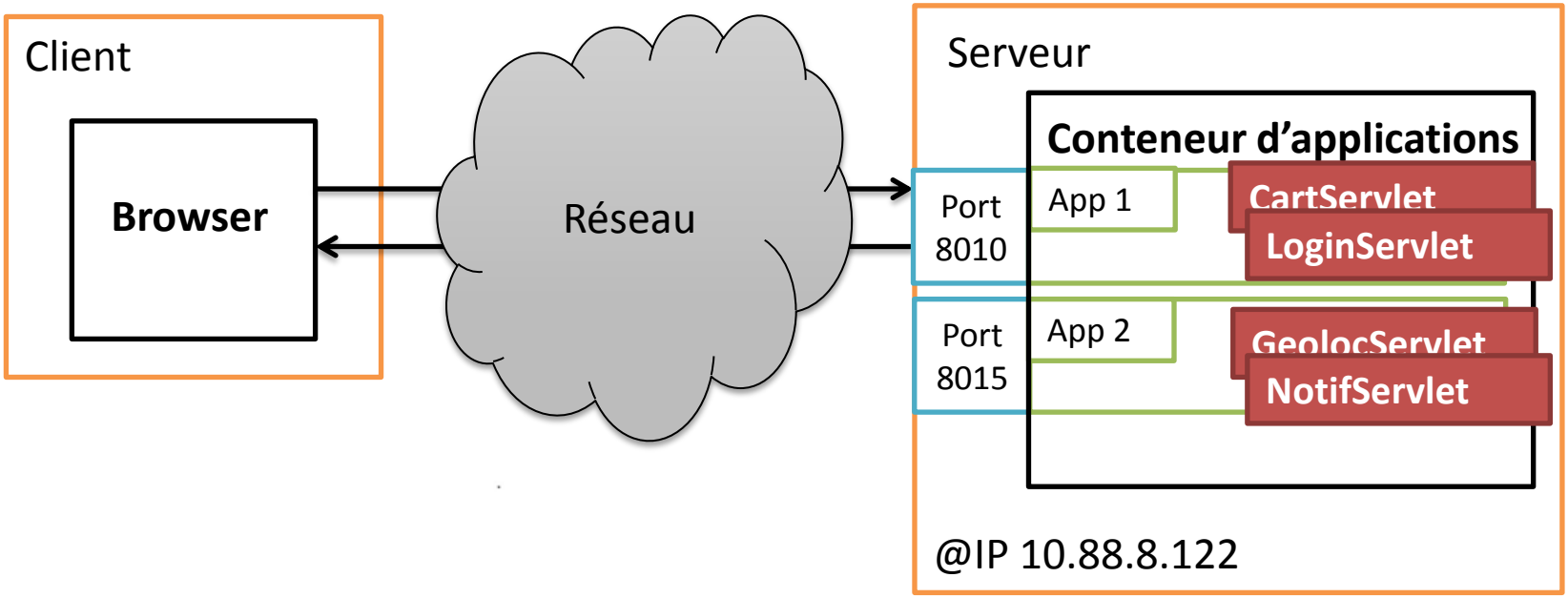


Tomcat

- Tomcat est le conteneur d'application JEE le plus utilisé
- Tomcat supporte les Servlet, les JSP, le langage EL, les WebSocket…
- Il est constitué des 3 principaux éléments:
  - **Catalina**: le conteneur d'application à proprement parlé
  - **Coyote**: le serveur web HTTP pour servir des ressources statiques
  - **Jasper**: moteur de compilation des JSP

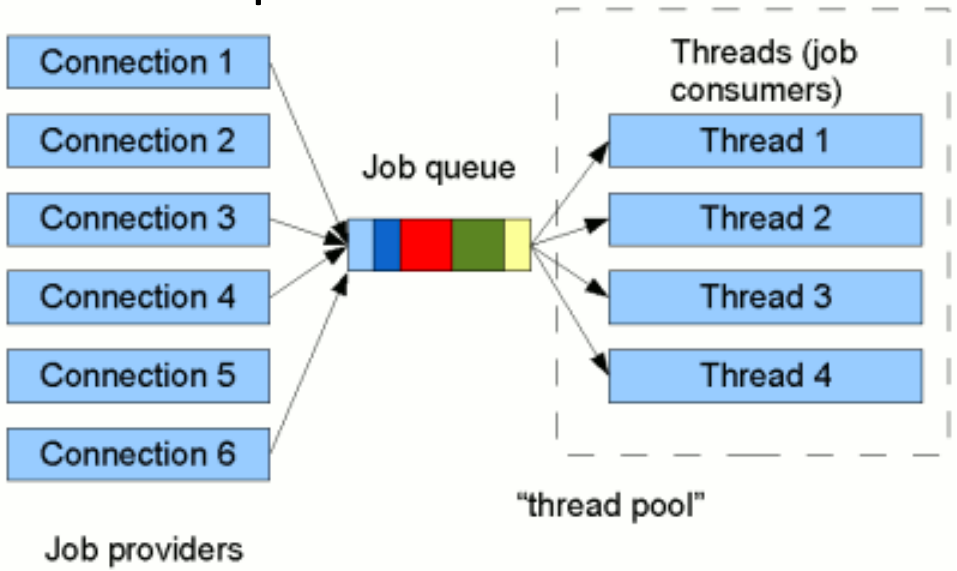
Tomcat

- Tomcat peut héberger plusieurs applications qui elles même peuvent contenir plusieurs Servlet



Parallèlisme

- Pour pouvoir traiter plusieurs requêtes en parallèle, Tomcat utilise un pool de thread



- Blocking I/O: un thread reste bloqué lorsqu'un un appel à une ressource externe est effectué (BDD, WS…)

Configuration d'une web app

- Le fichier web.xml
  - Permet de configurer les servlets du containeur
    - Définition des servlets à charger
    - Option `load on startup`: permet de forcer le chargement du Servlet au démarrage du conteneur
    - Définition des paramètres de la méthode init()
  - Permet de définir le mapping entre URL et Servlet
  - Permet de définir des paramètres de contexte accessible par tous les servlets

Autres fichiers de configuration

- Server.xml
  - Fichier de configuration de Tomcat, il définit notamment les paramètres d'écoute, le fonctionnement des loggers…
- Context.xml
  - Définit le contexte d'exécution d'une application au sein du conteneur
- Tomcat-users.xml
  - Définit les droits d'accès au conteneur
- Ces fichiers sont gérés automatiquement par Eclipse
- **Attention par défaut Eclipse ne configure pas votre projet à la racine du serveur mais sur /nom-du-projet**

Exemple

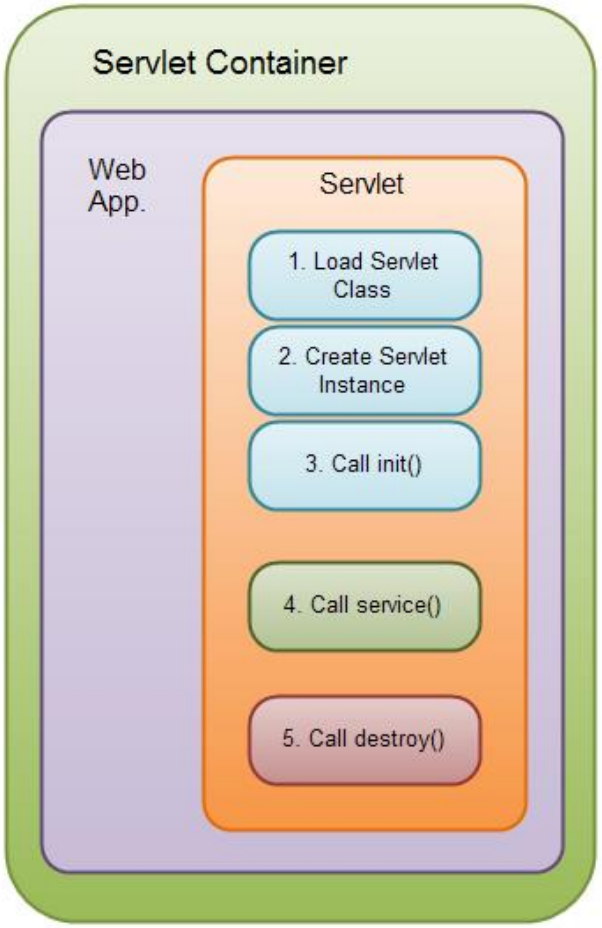
```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
    <servlet>
        <servlet-name>comingsoon</servlet-name>
        <servlet-class>mysite.server.ComingSoonServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
        <init-param>
            <param-name>lang</param-name>
            <param-value>FR</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>comingsoon</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>
    <error-page>
        <location>/error.html</location>
    </error-page>
</web-app>
```

Servlet

- API standard définit par Java EE pour répondre à des requêtes
- Principalement pour gérer du HTTP mais pas que
- La classe HTTPServlet étends la classe Servlet et permet de servir facilement du contenu sur HTTP
- La classe GenericServlet est destinée aux autres protocoles

Cycle de vie d'un servlet

- Le cycle de vie d'un servlet est géré par le conteneur
- Les étapes 1,2,3 ne sont exécutées qu'une seule fois lorsque le servlet est chargé par le conteneur
- Par défaut un servlet n'est chargé que lors de la première requête



HTTP Servlet

- La classe HTTP servlet propose d'implémenter une méthode pour chaque verbe HTTP (GET, POST, PUT etc…)
- Chaque méthode prend reçoit deux paramètres:
- Un objet **HttpRequest** permettant de lire les paramètres d'entrée
  - **request.getParameter**
  - request.getHeader
  - request.getInputStream
  - request.getSession
- Un object **HttpResponse** permettant d'écrire une réponse
  - response.getWriter
  - response.setHeader
  - response.sendRedirect

Exemple

```
public class SimpleHttpServlet extends HttpServlet {

  protected void doGet( HttpServletRequest request,
HttpServletResponse response){

    ServletContext context = request.getSession().getServletContext();
    String param1          = request.getParameter("param1");
    String contentLength   = request.getHeader("Content-Length");

    response.setHeader("Content-Type", "text/html");
    response.getWriter().write(
        "<html><body>GET response</body></html>");
} }
```

Concurrence

- Un servlet n'est instancié qu'une seule fois par Tomcat et est partagé par l'ensemble des threads de l'application
- Afin de demeurer « thread safe » un Servlet doit suivre la règle suivante:
  - Ne pas accéder ou réassigner des variables de classe ou des variables statiques dans une classe héritant de Servlet

  $\Rightarrow$ Donc toujours utiliser des variables locales
- Les objets request et response sont « thread safe », ils sont instanciés à chaque requête

Mapping

- Le mapping est défini dans le fichier web.xml
- Il permet au serveur de déterminer quel servlet doit répondre en fonction de l'URL cible
- Plusieurs Servlet peuvent correspondre à une même URL, Tomcat choisi alors selon les critères suivants
  - Un chemin exact sera toujours préféré à un joker
    - Ex. /hello vs. /*
  - L'url la plus longue gagne sur l'url la plus courte
    - Ex. /app/user/cart vs. /app
  - Le type explicite est pris en compte priotairement

Mapping

- Le servlet par défaut est utilisé pour répondre aux requêtes non gérées par le contexte applicatif
- On peut le remplacer en faisant répondre un servlet sur le chemin reservé /
- Le servlet par défaut peut aussi être utilisé pour répondre sur d'autres chemin

Example

```
<!-- Sers toutes des requêtes sur /app/login -->
<servlet-mapping>
    <servlet-name>loginServlet</servlet-name>
    <url-pattern>/app/login</url-pattern>
</servlet-mapping>

<!- Tout ce qui se trouve sous /static est servi par le servlet par
défaut -->
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/static/*</url-pattern>
</servlet-mapping>

<!- Tout ce qui ne match pas les règles précédentes est traité ici -->
<servlet-mapping>
<servlet-name>Error Servlet</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>
```