Deployment Best Practices

Bob Vawter
Google Inc.
BobV@google.com

Ryan Dewsbury
Ryan@rdews.com

http://FooApp.com

- What happens?
  - DNS lookup
  - Send HTTP request
  - Get the contents.
- Not quite that simple

DNS request

- Localhost queries local DNS server
  - Usually ISP or corporate proxy
- Recursive requests
  - Resolve SOA for TLD or 2LD with root-servers.net
  - Query your DNS servers for A record
- Opportunity for global load balancing

Send HTTP request

- Can’t just send HTTP request
  - Need to open a TCP connection
  - Three-way handshake
  - SYN, SYN+ACK, ACK
  - Then you can send data
- Unless you’re using SSL
- Connections are expensive

Get the contents

- GET / HTTP/1.1
  Host: FooApp.com
- HTTP/1.1 302 Moved Temporarily
  Location: http://fooapp.com/fooapp

Get the contents

- GET /fooapp HTTP/1.1
  Host: FooApp.com
- HTTP/1.1 302 Moved Temporarily
  Location:
  http://fooapp.com/fooapp/Foo.html

Rinse, repeat, ugh…

- This repeats for every distinct hostname
  - images.foo.com, static.foo.com
- Minimize number of connections
  - TCP keepalive and HTTP pipelining
  - Decreases overall time spent setting up
- Minimize number of HTTP requests
  - Decreases connections, “hiccups”

Just getting started

- This was just to get the HTML, JS, etc.
- My app is running, now what?
  - Additional runtime resources
  - RPC requests
  - Typical limit of two outstanding requests

Practical measures

- Why is my app slow?
  - This AJAX thing was supposed to be fast
  - It's about managing expectations
  - Increased freedom to control delay timing
  - Still just sweeping dirt under the carpet
- Get some hard data

A metaphor

http://flickr.com/photos/amagill/368648462/

Practical measures

- What measurements are useful?
  - Request latency is too coarse
  - AJAX webapps are a sandwich
  - Backend, Logic, Transport (, Presentation)
- Where to measure?
  - Internal vs. external
  - Beware the observer effect

Internal measurements

- Measurements taken by application
  - Built into servlet filters, data access layer
  - Prefer decile measurements over min/max
  - Also useful to tie into monitoring system
  - Don’t go overboard: timeseries explosion

External measurements

- Typically made without code mods
  - Things like tcpdump, ethereal
  - HTTP proxies are incredibly useful here
  - Feed measurements back into monitoring
- Charles demo

Wrapup - Part 1

- None of this is GWT-specific
- Think about the entire stack
  - May be looking at a second-order effect
- Turn lightweight profiling into monitoring
- Set concrete goals
  - App startup in 300ms
  - All RPC responses in sub-150.

Topics - Part 2

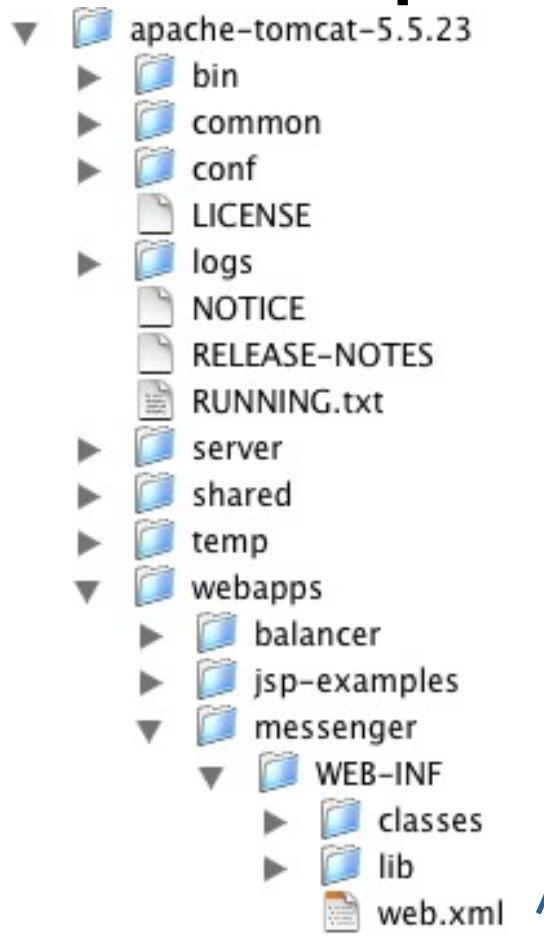
- Basic Deployment
- Deploying RPC Servlets
- Automation
- Port Issues
- Compression
- Caching
- Bundling

Basic Deployment

- Compile using your [appname]-compile script
- Deploy like it’s 1995!
- Just copy files from www to your server

```
4DE7BAE337B07B28CF9DCD99D57AD5DC.cache.html
5509B5CB134E0664F6C74270AD842ACD.cache.html
8B2896C30E249EA0BEAC62173FA5DAC7.cache.html
587B337E80FD4ED614B3655DA54B3D49.cache.html
94BB299E376E78522752931FD55AF0CC.cache.html
com.gwtapps.desktop.Desktop.nocache.js
Desktop.html
clear.cache.gif
64ED9A8F271A3E06D7D20B1CB77A337E.cache.png
tab-rest.gif
tab-selected.gif
tab.gif
titleback.gif
```

Deploying RPC Servlets




```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <servlet>
    <servlet-name>messenger</servlet-name>
    <servlet-class>com.gwtapps.messenger.
server.MessengerServiceImpl</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>messenger</servlet-name>
    <url-pattern>/messenger</url-pattern>
  </servlet-mapping>
</web-app>
```

Automating Deployment

- Use Ant to deploy
- Compile with Ant

```
<property name="app.base" value="/Users/ryan/Workspace/myapp"/>
<property name="gwt.compile" value="${app.base}/myapp-compile"/>
<target name="gwt-compile">
   <exec executable="${gwt.compile}" dir="${app.base}"/>
</target>
```

- Upload with Ant

```
<target name="gwt-upload" depends="gwt-compile">
     <ftp server= "myapp.com"
          remotedir="apache-tomcat-6.0.14/webapps/myapp"
          userid="tomcat" password="tomcat">
              <fileset dir="${app.base}/www/com.myapp.MyApp"/>
     </ftp>
</target>
```

Automating RPC Deployment

- Build a Web ARchive (WAR) file
- Compile your server code

```
<target name="compile">
     <javac destdir="${app.base}/bin" classpathref="classpath"
            excludes="**/*Test*.java" debug="on">
              <src path="${app.base}/src"/>
     </javac>
</target>
```

- Build the WAR file with Ant

```
<target name="buildwar" depends="gwt-compile,compile">
     <war destfile="${app.base}/MyApp.war"
          webxml="${app.base}/web.xml">
              <classes dir="${app.base}/bin"/>
              <lib file="/gwt-mac-1.4.61/gwt-servlet.jar"/>
              <fileset dir="${app.base}/www/com.myapp.MyApp‟/>
     </war>
</target>
```

Deploying a WAR

- Just copy the WAR to the webapps directory
- Or, use Tomcat Ant tasks to deploy
  - Copy the catalina-ant.jar file to the Ant lib directory
  - Declare Tomcat deploy tasks and a target

```
<taskdef name="deploy" classname="org.apache.catalina.ant.DeployTask"/>
<taskdef name="undeploy"
   classname="org.apache.catalina.ant.UndeployTask"/>
<target name="deploy" depends="buildwar">
   <deploy url="http://myapp.com/manager"
        username= "tomcat"
        password= "tomcat"
        path="myapp"
        update="true"
        war="file:${app.base}/MyApp.war"
     />
</target>
```

Port Issues

- Ports are not friendly: http//myapp.com:8080
- Serving client from Apache and server from Tomcat doesn't work…Same-Origin policy applies to ports too.
- Install mod_jk for Apache (httpd.conf)

```
LoadModule jk_module libexec/mod_jk.so
AddModule mod_jk.c
JkWorkersFile
  /usr/tomcat/conf/workers.properties
JkLogFile /usr/local/apache/logs/mod_jk.log
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
JkRequestLogFormat "%w %V %T"
```

- Forward servlet URL to Tomcat (httpd.conf)

```
JkMount /MyServlet ajp13
```

Optimization: Compression

- Save bandwidth by turning on compression
- GWT apps can shrink approx. 70%
- Content negotiation is automatic
- Turn compression on in Apache (httpd.conf or .htaccess)

```
SetOutputFilterByType DEFLATE text/html text/css
  application/x-javascript
```

- Turn compression on in Tomcat (server.xml)

```
<Connector  compression="on" compressableMimeType=
  "text/css,text/html,text/xml,text/plain,
  application/x-javascript" …
```

Optimization: Caching

- Web distribution has rich cache control on every resource – not optimal for Ajax apps
- Ajax apps can optimize cache freshness for the entire app (all resources)
- The browser just checks the freshness of one file (com.myapp.MyApp.nocache.js)
- HTTP calls for resource freshness are dramatically reduced – startup time increases

Optimization: Caching

- Use names unique to the app's version
- Change the names for new versions
  - automatic for GWT code, manual for other resources
- Add a far future expires header to resources in Apache (httpd.conf or .htaccess)

```
<Files *.cache.*>
ExpiresDefault "now plus 1 year"
</Files>
```

Optimization: Caching

- Add a far futures header to Tomcat using a filter
- Filter declaration (web.xml)

```
<filter>
   <filter-name>CacheFilter</filter-name>
   <filter-class>com.rdews. filters.CacheFilter</filter-class>
</filter>
<filter-mapping>
   <filter-name>CacheFilter</filter-name>
   <url-pattern>/*url-pattern>
</filter-mapping>
```

Optimization: Caching

- Filter class (CacheFilter.java)

```
public class CacheFilter implements Filter {
    private FilterConfig filterConfig;

    public void doFilter( ServletRequest request, ServletResponse response,
                FilterChain filterChain) throws IOException, ServletException {

      HttpServletRequest httpRequest = (HttpServletRequest)request;

      String requestURI = httpRequest.getRequestURI();
      if( !requestURI.contains(".nocache.") ){
                long today = new Date().getTime();
                HttpServletResponse httpResponse = (HttpServletResponse)response;
                httpResponse.setDateHeader("Expires", today+31536000000L);
      }
      filterChain.doFilter(request, response);

    }

    public void init(FilterConfig filterConfig) throws ServletException {
      this.filterConfig = filterConfig;
    }

    public void destroy() {
      this.filterConfig = null;
    }
```

Optimization: Bundling

- Application code is automatically bundled together by GWT
- Still, many HTTP requests are needed to get other resources when a browser's cache is not primed
- Bundling resources together reduces HTTP trips and improves startup time

Optimization: Bundling

- Use ImageBundle
- GWT combines many images into a single image and “extracts” them at run time
- Declare ImageBundle interface

```
public interface Images extends ImageBundle {

  /** @gwt.resource membersm.png */
  AbstractImagePrototype member();

  /** @gwt.resource away.png */
  AbstractImagePrototype away();

  /** @gwt.resource starsm.gif */
  AbstractImagePrototype star();
```

Optimization: Bundling

- Extract images

```
Images images = (Images) GWT.create(Images.class);
mainPanel.add( images.turn().createImage() );
```

- One extra file to deploy

```
64ED9A8F271A3E06D7D20B1CB77A337E.cache.png
```

- The Gpokr image bundle:

  - 73 images
  - 100kb
  - 1 HTTP request

Optimization: Bundling

- What about other resources like CSS and CSS images?
- Can’t use images from ImageBundle in CSS
- GWT Incubator has a ImmutableResourceBundle to bundle any text or data resource together
- Also, the incubator has StyleInjector to replace CSS image placeholders with inline images