Making Molehills out of a Mountain

Competition!


Loyalty
Activity
CIT
CIT
CIT Test
ST
ST
ST
LIVE
UAT
UAT

If you deserve a prize for understanding it...

If you think its complicated now...

...blah blah blah building functionality twice blah blah blah monolithic application blah blah blah load balancing proxy **blah blah blah** clustered session management blah blah blah smaller applications blah blah blah...


Mountain


Molehills

- Use multiple teams to build smaller applications using reusable components which perform a subset of the overall functionality
- Have a properly versioned set of libraries and components with USEFUL documentation so we know exactly what was added in a particular release and any problems there may be with backward compatibility
-
- Oh yes... Maven would really help with this... :-)

- Spend less time integrating streams
- Maintain different release schedules for different components and allow small changes to be released more swiftly
- Reduce our regression testing effort
- Allow us to more easily introduce new technologies into our stack (latest version of JSF, Spring Framework etc...)
- Canary releasing and live environment testing of new components
- Allows us to better load balance applications based on actual or expected load

What's stopping us doing this right now?

The Session Problem

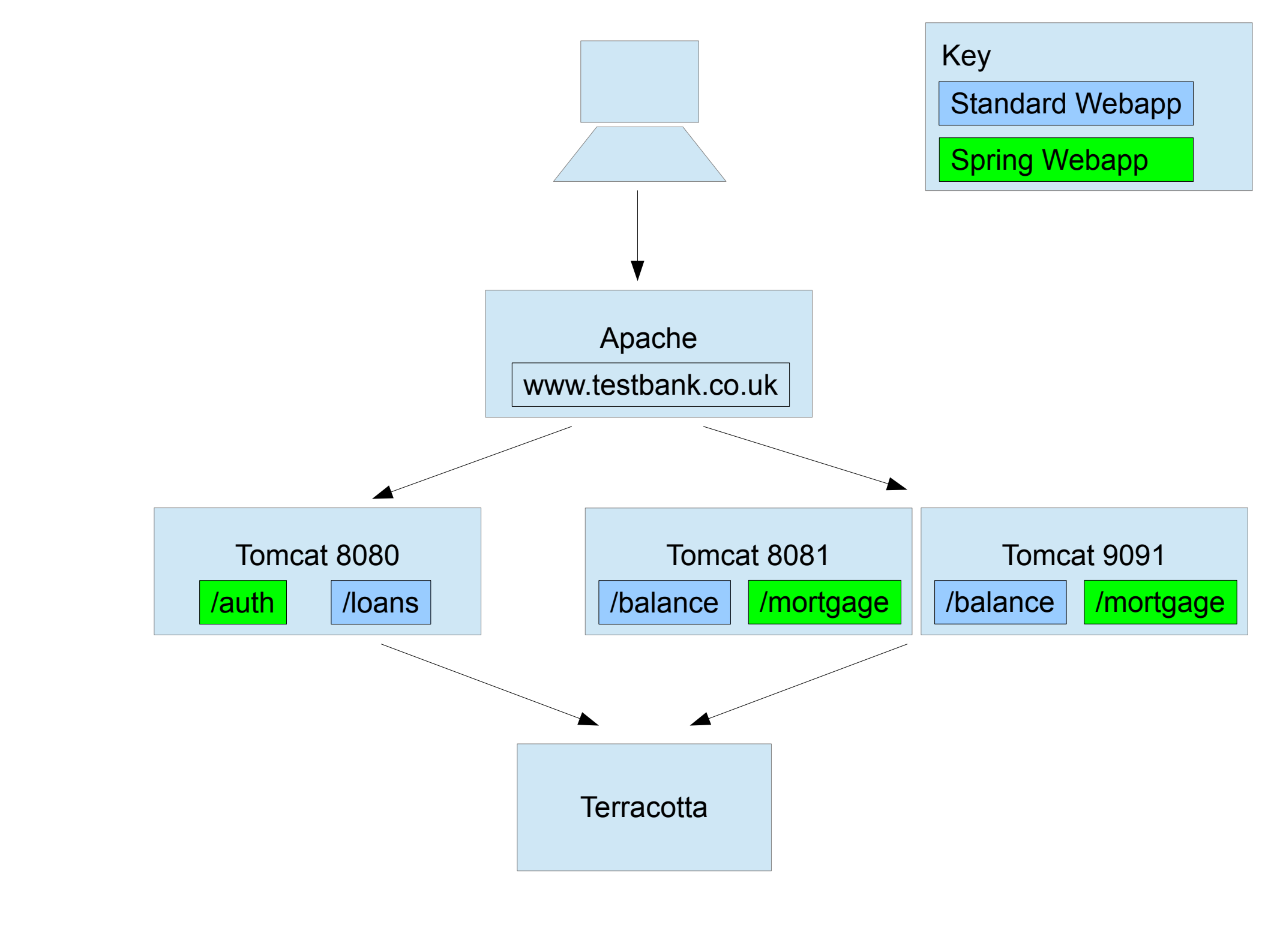
- Most complex applications need to store some form of state (either using sessions, databases or something else)
- Right now we have the same application deployed to every machine in the cluster, using standard jee sessions and a load balancer ensuring sticky sessions to bind a customer to a particular machine.
- To build smaller apps we would need to enable users to log in once and be authenticated across multiple applications deployed to multiple machines

Introducing Apache Shiro

Java Security Framework providing,

- Authentication
- Authorization
- Cryptography
- Web Integration
- and Session Management

The Demo


Key
Standard Webapp
Spring Webapp
Apache
www.testbank.co.uk
Tomcat 8080
/auth
/loans
Tomcat 8081
/balance
/mortgage
Tomcat 9091
/balance
/mortgage
Terracotta

The httpd.conf file

```
<Proxy balancer://mortgage>
BalancerMember http://localhost:8081/mortgage
BalancerMember http://localhost:9091/mortgage
</Proxy>

<Proxy balancer://balance>
BalancerMember http://localhost:8081/balance
BalancerMember http://localhost:9091/balance
</Proxy>

ProxyPass /auth http://localhost:8080/auth
ProxyPassReverse /auth http://localhost:8080/auth

ProxyPass /loans http://localhost:8080/loans
ProxyPassReverse /loans http://localhost:8080/loans

ProxyPass /mortgage balancer://mortgage
ProxyPassReverse /mortgage balancer://mortgage

ProxyPass /balance balancer://balance
ProxyPassReverse /balance balancer://balance
```

The shiro.ini file

```
[main]
# Cache for single sign on
ssoCacheManager = org.apache.shiro.cache.ehcache.EhCacheManager
ssoCacheManager.cacheManagerConfigFile = classpath:ehcache.xml
securityManager.cacheManager = $ssoCacheManager
# native for single sign on
securityManager.sessionMode = native

# DAO for single sign on
sessionDAO = org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO
securityManager.sessionManager.sessionDAO = $sessionDAO
# cookie for single sign on
cookie = org.apache.shiro.web.servlet.SimpleCookie
cookie.name = SSOcookie
cookie.path = /
securityManager.sessionManager.sessionIdCookie = $cookie

[ urls]
# The /login.jsp is not restricted to authenticated users (otherwise no one could log in!), but
# the 'authc' filter must still be specified for it so it can process that url's
# login submissions. It is 'smart' enough to allow those requests through as specified by the
# shiro.loginUrl above.
/login.jsp = authc
/** = authc
```

Apache Shiro Spring Config

```
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <property name="cacheManager" ref="cacheManager"/>
        <property name="sessionMode" value="native"/>
        <property name="realm" ref="iniRealm"/>
        <property name="sessionManager" ref="sessionManager"/>
    </bean>
    <bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
        <property name="sessionDAO" ref="sessionDao" />
        <property name="sessionIdCookie" ref="ssoCookie" />
    </bean>
    <bean id="sessionDao" class="org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO"/>
    <bean id="ssoCookie" class="org.apache.shiro.web.servlet.SimpleCookie">
        <property name="name" value="SSOcookie"/>
        <property name="path" value="/"/>
    </bean>
    <bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">
        <property name="cacheManagerConfigFile" value="classpath:ehcache.xml"/>
    </bean>
    <bean id="ehCacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
    <property name="shared" value="true"/>
</bean>
```

The ehcache.xml file

```
<ehcache name="shiro">
    <terracottaConfig url="localhost:9510"/>
    <diskStore path="java.io.tmpdir/shiro-ehcache"/>
    <defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="false"
        diskPersistent="false"
        diskExpiryThreadIntervalSeconds="120">
        <terracotta/>
   </defaultCache>
   <cache name="shiro-activeSessionCache"
       maxElementsInMemory="10000"
       eternal="true"
       timeToLiveSeconds="0"
       timeToIdleSeconds="0"
       diskPersistent="false"
       overflowToDisk="false"
       diskExpiryThreadIntervalSeconds="600">
       <terracotta/>
   </cache>
</ehcache>
```

Using the Apache Shiro Session

```
import org.apache.shiro.SecurityUtils;

SecurityUtils.getSubject().login(token);
...
SecurityUtils.getSubject().getSession().setAttribute(BALANCE, AccountService.getBalance(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(LOAN, AccountService.getLoanInfo(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(MORTGAGE, AccountService.getMortgageInfo(accountHolder));
...
SecurityUtils.getSubject().getSession().getAttribute(BALANCE);
SecurityUtils.getSubject().getSession().getAttribute(LOAN);
SecurityUtils.getSubject().getSession().getAttribute(MORTGAGE);
...
```

Using the Apache Shiro Session

```
import org.apache.shiro.SecurityUtils;

SecurityUtils.getSubject().login(token);
...
SecurityUtils.getSubject().getSession().setAttribute(BALANCE, AccountService.getBalance(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(LOAN, AccountService.getLoanInfo(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(MORTGAGE, AccountService.getMortgageInfo(accountHolder));
...
SecurityUtils.getSubject().getSession().getAttribute(BALANCE);
SecurityUtils.getSubject().getSession().getAttribute(LOAN);
SecurityUtils.getSubject().getSession().getAttribute(MORTGAGE);
...
```

Alternatives To Shiro

- Spring Security + CAS
- Application Server Realms

Alternatives to Terracotta

- Oracle Coherence
- IBM Extreme Scale

Further Reading

- Have a look for a talk by Stefan Tilkov called “Breaking the Monolith” - available on InfoQ
- Or some of the stuff available InfoQ around the ebay Architecture

Making Molehills out of a Mountain

I would like to just say that the first couple of minutes of this talk will be spent adding some context as to why I have been looking into this stuff.

I hope that none of you perceive this as a rant but more a view on what can be improved in our existing architecture and the way we build our software.

Competition!

Sooo... there nothing like a bit of audience participation to get a brown bag going.

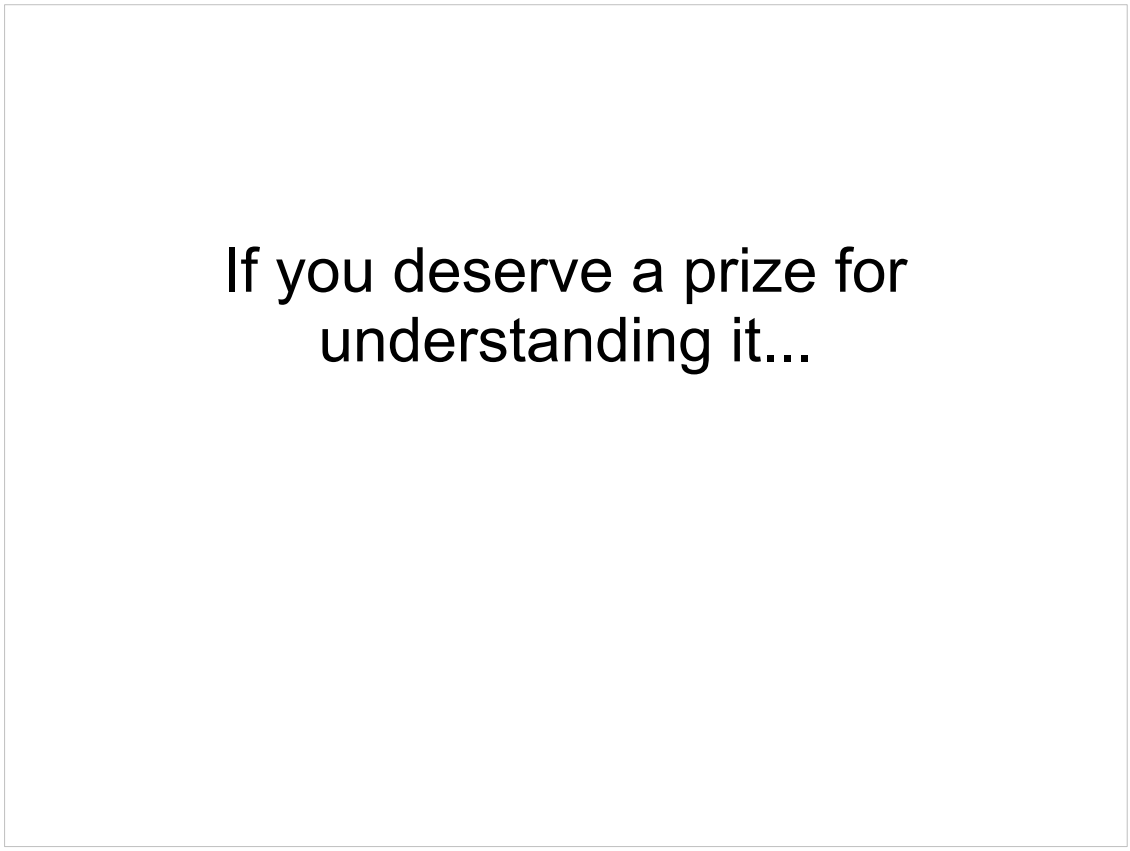
And there is nothing like the thought of a bribe or special mystery prize to motivate people.

I have here a rather gaudily wrapped prize that I went out to buy yesterday and then seemingly wrapped in the camp-est wrapping I could possibly find...

The prize is for whoever can explain what this diagram is and can talk us all through it...

Not to pick on somebody but Uros its your diagram – if you can't explain it then I'm not sure anyone else will be willing to give it a go...

Unfortunately there is no prize for the bottom question but I couldn't work it out myself...

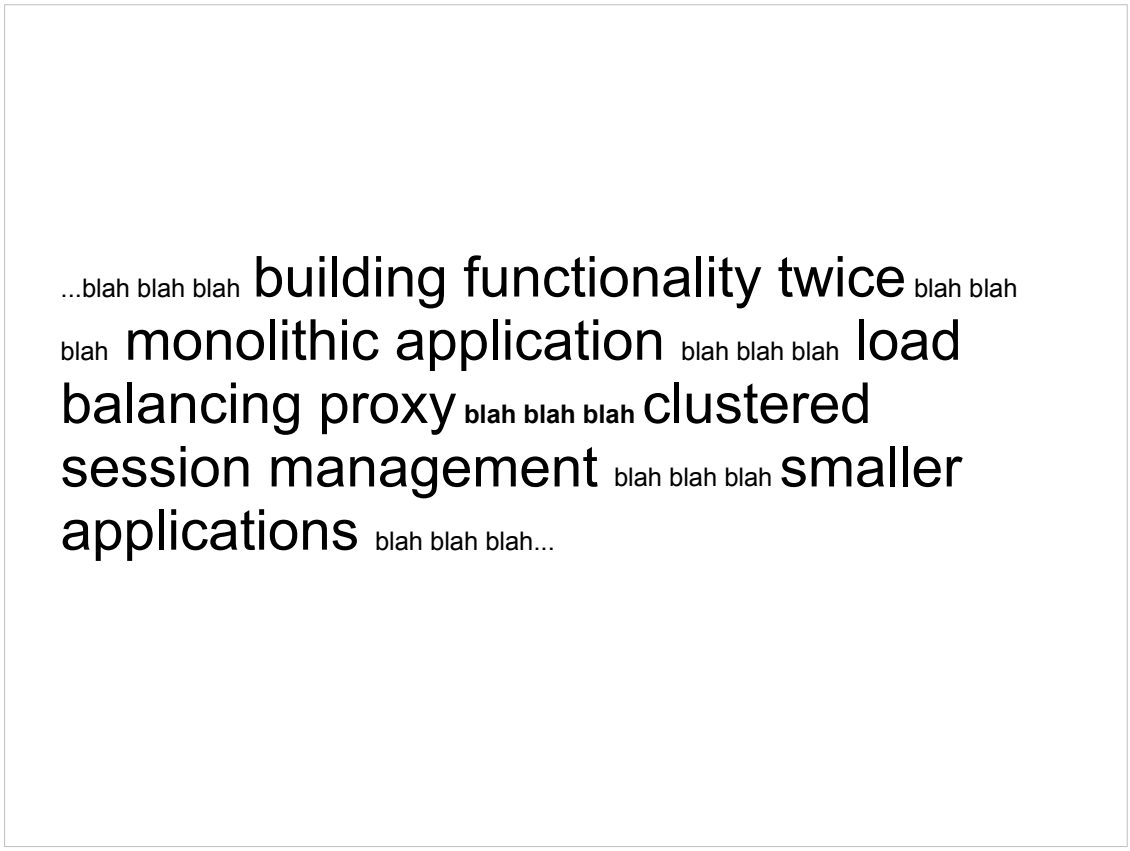


Then surely it is too complicated.

I have looked at this diagram many times and still can't fully wrap my head around how it all manages to actually work.

If you think its complicated now...

- It is only gonna get worse...
  - New initiatives are starting all the time you only have to glance at the gant chart that adorn the walls next door to realise that the integration task is only going to grow and get more involved...



So it was to this end I sent an email to Uros a couple of weeks back.... something along the lines of...

The bottom line was that we need to stop building this....

And start building these...

- Use multiple teams to build smaller applications using reusable components which perform a subset of the overall functionality
- Have a properly versioned set of libraries and components with USEFUL documentation so we know exactly what was added in a particular release and any problems there may be with backward compatibility
-
- Oh yes... Maven would really help with this... :-)

Is to stop building a monolithic application and break our functionality down into smaller more manageable chunks...

- Spend less time integrating streams
- Maintain different release schedules for different components and allow small changes to be released more swiftly
- Reduce our regression testing effort
- Allow us to more easily introduce new technologies into our stack (latest version of JSF, Spring Framework etc...)
- Canary releasing and live environment testing of new components
- Allows us to better load balance applications based on actual or expected load

What's stopping us doing this right now?

The Session Problem

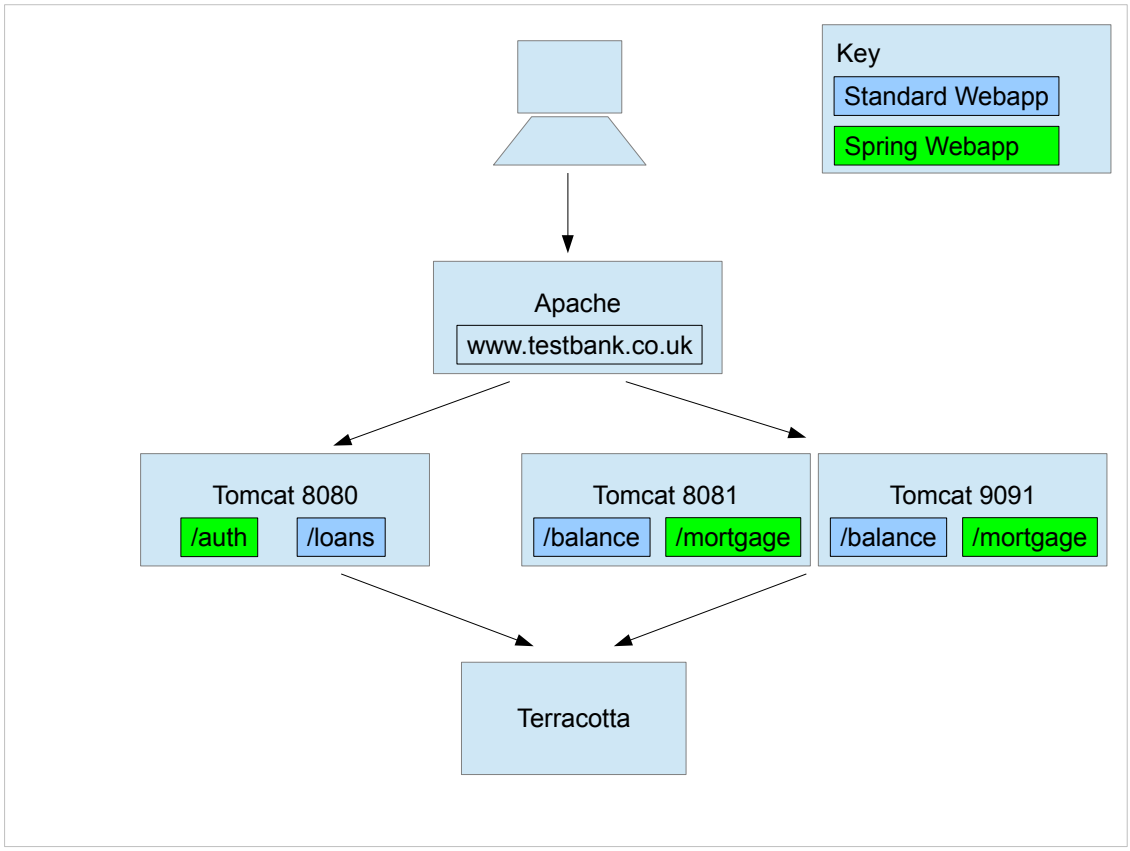
- Most complex applications need to store some form of state (either using sessions, databases or something else)
- Right now we have the same application deployed to every machine in the cluster, using standard jee sessions and a load balancer ensuring sticky sessions to bind a customer to a particular machine.
- To build smaller apps we would need to enable users to log in once and be authenticated across multiple applications deployed to multiple machines

Introducing Apache Shiro

Java Security Framework providing,

- Authentication
- Authorization
- Cryptography
- Web Integration
- and Session Management

The Demo


Key
Standard Webapp
Spring Webapp
Apache
www.testbank.co.uk
Tomcat 8080
/auth
/loans
Tomcat 8081
/balance
/mortgage
Tomcat 9091
/balance
/mortgage
Terracotta

The httpd.conf file

```
<Proxy balancer://mortgage>
BalancerMember http://localhost:8081/mortgage
BalancerMember http://localhost:9091/mortgage
</Proxy>

<Proxy balancer://balance>
BalancerMember http://localhost:8081/balance
BalancerMember http://localhost:9091/balance
</Proxy>

ProxyPass /auth http://localhost:8080/auth
ProxyPassReverse /auth http://localhost:8080/auth

ProxyPass /loans http://localhost:8080/loans
ProxyPassReverse /loans http://localhost:8080/loans

ProxyPass /mortgage balancer://mortgage
ProxyPassReverse /mortgage balancer://mortgage

ProxyPass /balance balancer://balance
ProxyPassReverse /balance balancer://balance
```

The shiro.ini file

```
[main]
# Cache for single sign on
ssoCacheManager = org.apache.shiro.cache.ehcache.EhCacheManager
ssoCacheManager.cacheManagerConfigFile = classpath:ehcache.xml
securityManager.cacheManager = $ssoCacheManager
# native for single sign on
securityManager.sessionMode = native

# DAO for single sign on
sessionDAO = org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO
securityManager.sessionManager.sessionDAO = $sessionDAO
# cookie for single sign on
cookie = org.apache.shiro.web.servlet.SimpleCookie
cookie.name = SSOcookie
cookie.path = /
securityManager.sessionManager.sessionIdCookie = $cookie

[urls]
# The /login.jsp is not restricted to authenticated users (otherwise no one could log in!), but
# the 'authc' filter must still be specified for it so it can process that url's
# login submissions. It is 'smart' enough to allow those requests through as specified by the
# shiro.loginUrl above.
/login.jsp = authc
/** = authc
```

Apache Shiro Spring Config

```
<bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <property name="cacheManager" ref="cacheManager"/>
        <property name="sessionMode" value="native"/>
        <property name="realm" ref="iniRealm"/>
        <property name="sessionManager" ref="sessionManager"/>
    </bean>
    <bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
        <property name="sessionDAO" ref="sessionDao" />
        <property name="sessionIdCookie" ref="ssoCookie" />
    </bean>
    <bean id="sessionDao" class="org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO"/>
    <bean id="ssoCookie" class="org.apache.shiro.web.servlet.SimpleCookie">
        <property name="name" value="SSOcookie"/>
        <property name="path" value="/"/>
    </bean>
    <bean id="cacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">
        <property name="cacheManagerConfigFile" value="classpath:ehcache.xml"/>
    </bean>
    <bean id="ehCacheManager" class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
    <property name="shared" value="true"/>
</bean>
```

The ehcache.xml file

```
<ehcache name="shiro">
    <terracottaConfig url="localhost:9510"/>
    <diskStore path="java.io.tmpdir/shiro-ehcache"/>
    <defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="false"
        diskPersistent="false"
        diskExpiryThreadIntervalSeconds="120">
        <terracotta/>
   </defaultCache>
   <cache name="shiro-activeSessionCache"
       maxElementsInMemory="10000"
       eternal="true"
       timeToLiveSeconds="0"
       timeToIdleSeconds="0"
       diskPersistent="false"
       overflowToDisk="false"
       diskExpiryThreadIntervalSeconds="600">
       <terracotta/>
   </cache>
</ehcache>
```

Using the Apache Shiro Session

```
import org.apache.shiro.SecurityUtils;

SecurityUtils.getSubject().login(token);
...
SecurityUtils.getSubject().getSession().setAttribute(BALANCE, AccountService.getBalance(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(LOAN, AccountService.getLoanInfo(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(MORTGAGE, AccountService.getMortgageInfo(accountHolder));
...
SecurityUtils.getSubject().getSession().getAttribute(BALANCE);
SecurityUtils.getSubject().getSession().getAttribute(LOAN);
SecurityUtils.getSubject().getSession().getAttribute(MORTGAGE);
...
```

Using the Apache Shiro Session

```
import org.apache.shiro.SecurityUtils;

SecurityUtils.getSubject().login(token);
...
SecurityUtils.getSubject().getSession().setAttribute(BALANCE, AccountService.getBalance(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(LOAN, AccountService.getLoanInfo(accountHolder));
SecurityUtils.getSubject().getSession().setAttribute(MORTGAGE, AccountService.getMortgageInfo(accountHolder));
...
SecurityUtils.getSubject().getSession().getAttribute(BALANCE);
SecurityUtils.getSubject().getSession().getAttribute(LOAN);
SecurityUtils.getSubject().getSession().getAttribute(MORTGAGE);
...
```

Alternatives To Shiro

- Spring Security + CAS
- Application Server Realms

Alternatives to Terracotta

- Oracle Coherence
- IBM Extreme Scale

Further Reading

- Have a look for a talk by Stefan Tilkov called “Breaking the Monolith” - available on InfoQ
- Or some of the stuff available InfoQ around the ebay Architecture