Deploying JBoss in a Large Scale Production Environment

Rick MacConnell
Sameer Nanda

AutoTrader.com

Background

- We run our web applications out of two co-locations.
- We're a very unix-centric environment and, as a result, use unix tools like rsync and scp for doing deployments.
- We don't run our web-servers in clusters. If a web server dies, we take it out of service until it can be repaired or replaced.
- Development has no access to the qa or production environments.
- We do production deployments every two weeks. These deployments consist of building a tarball, deploying it to qa for testing and ultimately copying it to the target server, unpacking and restarting JBoss.

Motivating Factors

- We need a fool-proof way of configuring web servers and their associated applications.
- We wanted a method that had a minimum number of dependencies on the target hardware. Our current method requires only the presence of java and apache binaries to run.
- We needed the configuration to be explicit. We use a three tier architecture and need to be able to adjust the load across the tiers with a fairly fine degree of control.
- Once the distribution has left development, it needs to be able to accommodate the qa and production environments with little to no human intervention.
- We wanted a mechanism that fitted our unix-centric environment.

Customizing the JBoss Distro

- Our customization process begins when we want to perform a JBoss upgrade.
- After we download the latest appropriate stable JBoss release, we determine what customization is required to fit our needs.
- One of the first steps in building our customized solution is to unpack the JBoss distribution and remove configuration modes, services, and documents we do not need to distribute to our servers.
- We also have some common java libraries that are used across all our applications, so we place these in the JBoss server lib directory.
- Next we patch many of the JBoss files for various reasons, for example, we may patch run.sh to pass certain flags to the JVM or patch the web-console web.xml to turn on authentication.
- All of this is done via Ant which makes it relatively painless however; upgrading from one JBoss release to the next can be challenging.

JBoss Modifications

- We remove services we aren't currently using

```
<echo message="removing unused configuration files" />
    <delete file="${final.dir}/jboss/server/atc/deploy/user-service.xml"/>
    <delete file="${final.dir}/jboss/server/atc/deploy/scheduler-service.xml"/>
    <delete file="${final.dir}/jboss/server/atc/deploy/schedule-manager-service.xml"/>
    <delete file="${final.dir}/jboss/server/atc/deploy/hsqldb-ds.xml"/>
    <delete file="${final.dir}/jboss/server/atc/deploy/mail-service.xml"/>
    <delete file="${final.dir}/jboss/server/atc/deploy/cache-invalidation-service.xml"/>
    <delete dir="${final.dir}/jboss/server/atc/deploy/snmp-adaptor.sar"/>
    <delete dir="${final.dir}/jboss/server/atc/deploy/jms"/>
```

- We also patch some of the existing configuration files (note the .base file name extension below)

```
<echo message="patching jboss-minimal.xml" />
    <copy tofile="${final.dir}/atcbase/jboss/server/atc/conf/jboss-minimal.xml.base"
      file="${final.dir}/jboss/server/atc/conf/jboss-minimal.xml"/>
    <patch patchfile="${atcbase}/jboss/server/atc/conf/jboss-minimal.xml.patch"
      originalfile="${final.dir}/atcbase/jboss/server/atc/conf/jboss-minimal.xml.base"
    />
```

JBoss Modifications

- Finally, we introduce configurations that are specific to our needs (again, note the .base file name extension)

```
<echo message="copying oracle datasource configuration" />
   <copy tofile="${final.dir}/atcbase/jboss/server/atc/deploy/oracle-ds.xml.base"
     file="${atcbase}/jboss/server/atc/deploy/oracle-ds.xml.base"/>
```

- When we patch the config files, we add a .base extension to them. As part of the startup phase, we personalize these config files via sed using the portconfig files that we'll be discussing next.
- The .base extension helps us by explicitly identifying the configurations that we've modified. We can then see exactly how things are set up in the event of a configuration issue.

ATC Add Ons

- The next step in customizing the JBoss distribution is to build and include our specific additional features.
- We use the interceptor features of Tomcat to include valves that we find useful.
  - AccessLog4JValve – This allows us to send the tomcat access logs through log4j to control log granularity and rolling intervals. This is not explicitly necessary if you're also using Apache to log requests as we do but it can be useful.
  - AllStopValve – In cases where we've got an internal failure, we use this valve to turn on a stop sign page that intercepts all requests and redirects them.
  - CountingValve – This is useful for keeping track of metrics such as the number of sessions that are in use or have been created by Tomcat over a specific period of time.

ATC MBeans

- After adding in the Tomcat changes, we build and deploy a set of MBeans that we use to control various parts of the infrastructure.
  - AppStatus – We use this MBean to control whether the application is currently available or not. In conjunction with the AllStopValve, we can temporarily redirect requests to a maintenance message if necessary.
  - HTMLAdaptor – We use this mbean to control the authentication for accessing the configuration web apps.
  - SessionTracker – We use this mbean to control how we track session statistics.
  - Log4J – This allows us to control the Log4J configuration through the interface rather than having to adjust the log4j.xml file.
- We use a couple of other MBeans to control which data sources the application is currently using. This allows us to move applications across databases during maintenance windows.

ATC Application Configuration

- We use a similar mechanism to externalize configuration elements specific to our applications.
- As an example, we want to serve images from the correct environment based on where the application is currently running (development, qa or production).
- We create a properties file that we sed-ify at startup based on the portconfig file.
- At runtime, we load up a ResourceBundle using the properties file to get at the correct configuration elements.

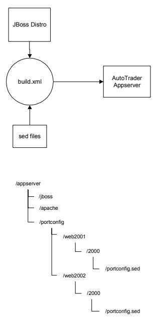
This line is in the properties file:

image_file_server_home.external_image_server_context_name=@@image_file_server_home.external_context_name@@

At startup, we replace the @@ marker token with the appropriate value and then load the properties file into a ResourceBundle.

Building the Application Server

- We build a sed file specific to each web server.
- The contents of the sed files are generated via Ant.
- We distribute the sed files for all of the web servers with each application distribution.
- The cost for generating these sed files is very low because it's been automated with Ant.
- We end up with a distribution that includes the configurations for all of the web servers and only the specific files for the desired web server are ever used.



Ant Target

During the build, we call the portconfig target for each web server in production, qa and development.

The target includes replacement values (see ${pc.user}) below that make the generated file specific to the server and application. The echo task in ant is used to generate the file.

```
 <target name="build_portconfig_consumersite">
    <echo message="Building port config file for ${pc.server} port 9200" />
    <mkdir dir="${final.dir}/portconfig/${pc.server}/9200"/>
    <echo file="${final.dir}/portconfig/${pc.server}/9200/${pc.filename}.apache">s|@@APACHE_PORT_NUMBER@@|9200|g
################################################################################
#              ISP  Consumer Site Specific Settings                    #
################################################################################
s|@@HYPERSONIC_PORT@@|9201|g
s|@@JBOSS_MGMT_PORT@@|9202|g
s|@@JNP_PORT@@|9203|g
s|@@MOD_JSERV_AJP13_PORT@@|9204|g
s|@@MOD_JSERV_PORT@@|9205|g
s|@@SSL_CONN_HNDLR@@|9206|g
s|@@WEBSERVER_SERVICE@@|9207|g
s|@@TOMCAT_WEBSERVER_PORT@@|9208|g
#
s|@@APACHE_HOME@@|/usr/local/packages/apache-1.3.29|g
s|@@APACHE_PORT_DIR@@|/data/port9200/appserver/apache|g
s|@@APACHE_LOG_DIR@@|/data/port9200/logs|g
s|@@USER_ID@@|${pc.user}|g
   </echo>
</target>
```

We currently have approximately 200 servers that we generate sed files for. With a little abstraction in your ant targets, you can very easily make a change in one place that applies to a large number of hosts.

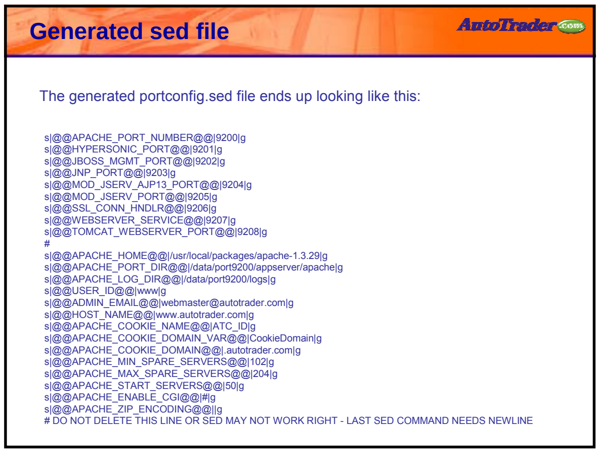
Personalizing the sed files

For each server, we call the appropriate build_portconfig target (shown on the previous slide) using replacement values (see pc.server below).

This generates a distinct file for each server which is then packaged up with the application server distribution.

```
<?xml version="1.0"?>
<!DOCTYPE project [
   <!ENTITY consumersite_portconfig SYSTEM "file:./consumersite-portconfig.xml">
]>

<project name="build-portconfig" default="build_portconfig_files" basedir=".">

  &consumersite_portconfig;

  <target name="build_portconfig_consumersite_cp">
   <antcall target="build_portconfig_consumersite">
      <param name="pc.filename" value="portconfig.sed.atdbate1"/>
      <param name="pc.server" value="web2001"/>
      <param name="pc.user" value="www"/>
  </antcall>
  <antcall target="build_portconfig_consumersite">
      <param name="pc.filename" value="portconfig.sed.atdbate1"/>
      <param name="pc.server" value="web2002"/>
      <param name="pc.user" value="www"/>
  </antcall>
```

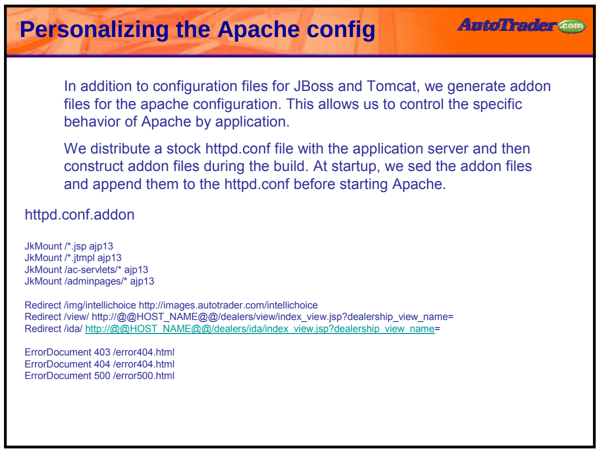
●
●
●



Generated sed file

The generated portconfig.sed file ends up looking like this:

```
s|@@APACHE_PORT_NUMBER@@|9200|g
s|@@HYPERSONIC_PORT@@|9201|g
s|@@JBOSS_MGMT_PORT@@|9202|g
s|@@JNP_PORT@@|9203|g
s|@@MOD_JSERV_AJP13_PORT@@|9204|g
s|@@MOD_JSERV_PORT@@|9205|g
s|@@SSL_CONN_HNDLR@@|9206|g
s|@@WEBSERVER_SERVICE@@|9207|g
s|@@TOMCAT_WEBSERVER_PORT@@|9208|g
#
s|@@APACHE_HOME@@|/usr/local/packages/apache-1.3.29|g
s|@@APACHE_PORT_DIR@@|/data/port9200/appserver/apache|g
s|@@APACHE_LOG_DIR@@|/data/port9200/logs|g
s|@@USER_ID@@|www|g
s|@@ADMIN_EMAIL@@|webmaster@autotrader.com|g
s|@@HOST_NAME@@|www.autotrader.com|g
s|@@APACHE_COOKIE_NAME@@|ATC_ID|g
s|@@APACHE_COOKIE_DOMAIN_VAR@@|CookieDomain|g
s|@@APACHE_COOKIE_DOMAIN@@|.autotrader.com|g
s|@@APACHE_MIN_SPARE_SERVERS@@|102|g
s|@@APACHE_MAX_SPARE_SERVERS@@|204|g
s|@@APACHE_START_SERVERS@@|50|g
s|@@APACHE_ENABLE_CGI@@|#|g
s|@@APACHE_ZIP_ENCODING@@||g
# DO NOT DELETE THIS LINE OR SED MAY NOT WORK RIGHT - LAST SED COMMAND NEEDS NEWLINE
```



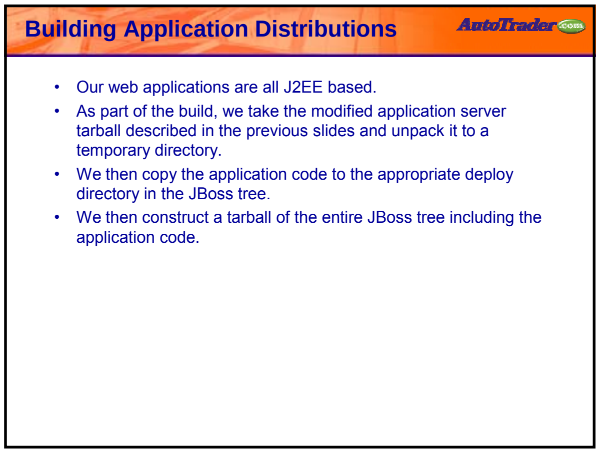
Personalizing the Apache config

In addition to configuration files for JBoss and Tomcat, we generate addon files for the apache configuration. This allows us to control the specific behavior of Apache by application.

We distribute a stock httpd.conf file with the application server and then construct addon files during the build. At startup, we sed the addon files and append them to the httpd.conf before starting Apache.

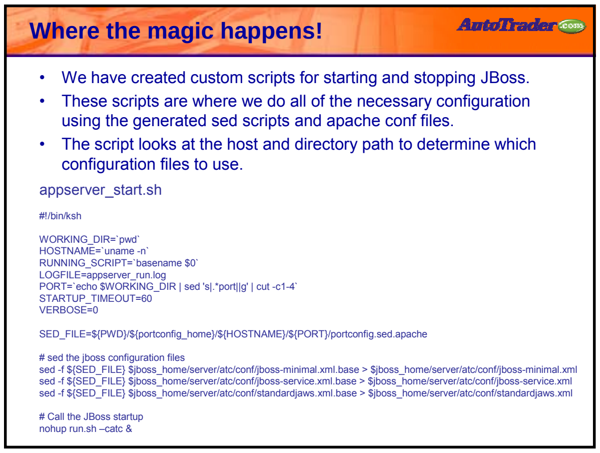
httpd.conf.addon

```
JkMount /*.jsp ajp13
JkMount /*.jtmpl ajp13
JkMount /ac-servlets/* ajp13
JkMount /adminpages/* ajp13

Redirect /img/intellichoice http://images.autotrader.com/intellichoice
Redirect /view/ http://@@HOST_NAME@@/dealers/view/index_view.jsp?dealership_view_name=
Redirect /ida/ http://@@HOST_NAME@@/dealers/ida/index_view.jsp?dealership_view_name=

ErrorDocument 403 /error404.html
ErrorDocument 404 /error404.html
ErrorDocument 500 /error500.html
```



Building Application Distributions

- Our web applications are all J2EE based.
- As part of the build, we take the modified application server tarball described in the previous slides and unpack it to a temporary directory.
- We then copy the application code to the appropriate deploy directory in the JBoss tree.
- We then construct a tarball of the entire JBoss tree including the application code.

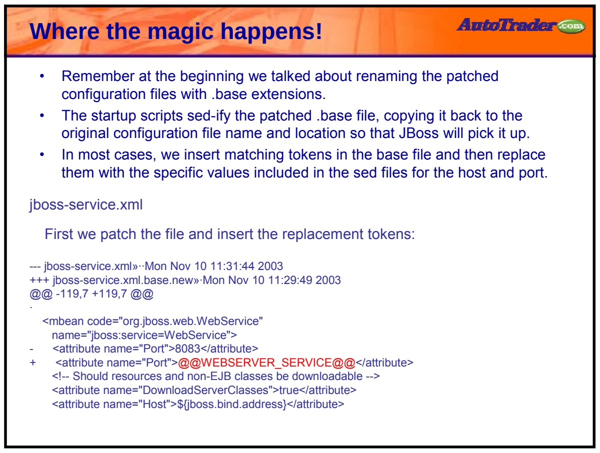


Where the magic happens!

- We have created custom scripts for starting and stopping JBoss.
- These scripts are where we do all of the necessary configuration using the generated sed scripts and apache conf files.
- The script looks at the host and directory path to determine which configuration files to use.

appserver_start.sh

```
#!/bin/ksh

WORKING_DIR=`pwd`
HOSTNAME=`uname -n`
RUNNING_SCRIPT=`basename $0`
LOGFILE=appserver_run.log
PORT=`echo $WORKING_DIR | sed 's|.*port||g' | cut -c1-4`
STARTUP_TIMEOUT=60
VERBOSE=0

SED_FILE=${PWD}/${portconfig_home}/${HOSTNAME}/${PORT}/portconfig.sed.apache

# sed the jboss configuration files
sed -f ${SED_FILE} $jboss_home/server/atc/conf/jboss-minimal.xml.base > $jboss_home/server/atc/conf/jboss-minimal.xml
sed -f ${SED_FILE} $jboss_home/server/atc/conf/jboss-service.xml.base > $jboss_home/server/atc/conf/jboss-service.xml
sed -f ${SED_FILE} $jboss_home/server/atc/conf/standardjaws.xml.base > $jboss_home/server/atc/conf/standardjaws.xml

# Call the JBoss startup
nohup run.sh –catc &
```



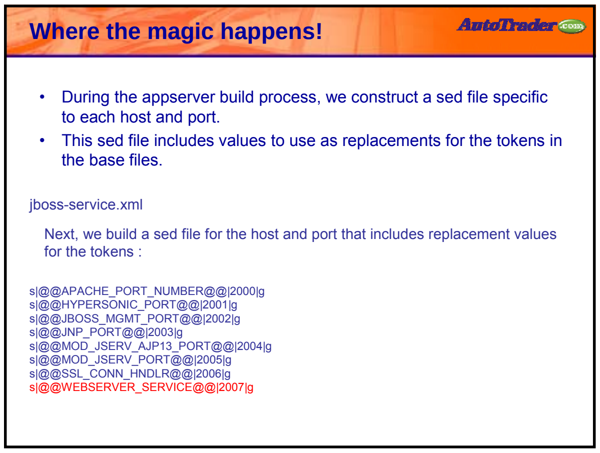
Where the magic happens!

- Remember at the beginning we talked about renaming the patched configuration files with .base extensions.
- The startup scripts sed-ify the patched .base file, copying it back to the original configuration file name and location so that JBoss will pick it up.
- In most cases, we insert matching tokens in the base file and then replace them with the specific values included in the sed files for the host and port.

jboss-service.xml

First we patch the file and insert the replacement tokens:

```
--- jboss-service.xml»··Mon Nov 10 11:31:44 2003
+++ jboss-service.xml.base.new»·Mon Nov 10 11:29:49 2003
@@ -119,7 +119,7 @@
·
   <mbean code="org.jboss.web.WebService"
      name="jboss:service=WebService">
-     <attribute name="Port">8083</attribute>
+      <attribute name="Port">@@WEBSERVER_SERVICE@@</attribute>
      <!-- Should resources and non-EJB classes be downloadable -->
      <attribute name="DownloadServerClasses">true</attribute>
      <attribute name="Host">${jboss.bind.address}</attribute>
```



Where the magic happens!

- During the appserver build process, we construct a sed file specific to each host and port.
- This sed file includes values to use as replacements for the tokens in the base files.

jboss-service.xml

Next, we build a sed file for the host and port that includes replacement values for the tokens :

```
s|@@APACHE_PORT_NUMBER@@|2000|g
s|@@HYPERSONIC_PORT@@|2001|g
s|@@JBOSS_MGMT_PORT@@|2002|g
s|@@JNP_PORT@@|2003|g
s|@@MOD_JSERV_AJP13_PORT@@|2004|g
s|@@MOD_JSERV_PORT@@|2005|g
s|@@SSL_CONN_HNDLR@@|2006|g
s|@@WEBSERVER_SERVICE@@|2007|g
```

Where the magic happens!

- At startup, we use sed to replace the tokens in the .base file with the appropriate values from the portconfig.sed files.
- This results in a configuration file that has been customized on the fly to suit the host, port and application

jboss-service.xml

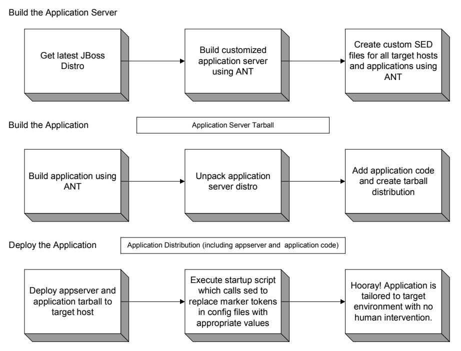
Finally, during the startup phase, we sed the file :

```
sed -f ${SED_FILE} $atcbase_jboss_home/server/atc/conf/jboss-service.xml.base >
$jboss_home/server/atc/conf/jboss-service.xml
```

And the result is a jboss-service.xml file that has been customized.

```
<!-- ==================================================================== -->
<!-- Class Loading                                                        -->
<!-- ==================================================================== -->

<mbean code="org.jboss.web.WebService"
   name="jboss:service=WebService">
   <attribute name="Port">2007</attribute>
   <!-- Should resources and non-EJB classes be downloadable -->
   <attribute name="DownloadServerClasses">true</attribute>
   <attribute name="Host">${jboss.bind.address}</attribute>
   <attribute name="BindAddress">${jboss.bind.address}</attribute>
</mbean>
```

The Whole Process


Build the Application Server
Get latest JBoss Distro
Build customized application server using ANT
Create custom SED files for all target hosts and applications using ANT
Build the Application
Application Server Tarball
Build application using ANT
Unpack application server distro
Add application code and create tarball distribution
Deploy the Application
Application Distribution (including appserver and application code)
Deploy appserver and application tarball to target host
Execute startup script which calls sed to replace marker tokens in config files with appropriate values
Hooray! Application is tailored to target environment with no human intervention.

What does this do for us?

• After getting this (admittedly complex) build system set up, we have an appserver distribution that requires no intervention as it moves through from development to production.

• As we add servers to our infrastructure, we simply add the host specific fields to the ant build scripts and unpack the applicable tarball on the box.

• The configuration is evident. You can look at the config files and see how things are set up. This is beneficial in our environment where the operations team is responsible for all production support. They can easily track down configuration issues by looking at the portconfig files.

• The developers have explicit control over how the applications are configured and can be confident that the production servers are set up correctly.

• In cases where we need to test configuration changes in production (memory tuning, etc.) we can do that without having to get credentials for the production servers.

What can you do?

•Start small; when we first went down this road, our only goal was to be able to control the ports on which the various JBoss services listen at run time.

•Get your ant structure in place so that you can reliably and repeatably assemble an application server distro.

•Abstract the things that are common to your application environments.

•Wrap the stock startup script so that you can modify the appropriate config files before JBoss starts.

•If you're using Apache, look at the things that are specific to your config and move them into a separate file that you can append to the httpd.conf file at startup.

•Look at JMX and use it where appropriate. We still have progress to make in this area.

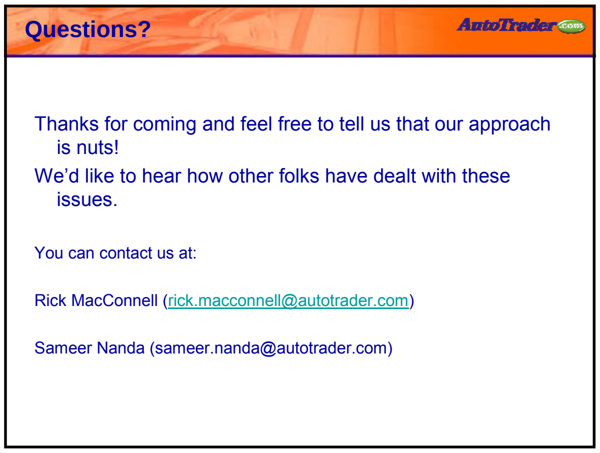
•Keep up with the JBoss upgrades. Waiting too long between them makes the whole process more painful.

Favorite Enterprise Features

- JMX – We use JMX and custom MBeans to control certain aspects of the run-time configuration (logging, database pools, etc.).
- JMS – We use JMS for asynchronous processes.
- Web-services Integration – We integrate with many heterogeneous internal and external applications via web-services.
- Custom Interceptors – We use custom EJB interceptors for monitoring, caching, and logging.
- Tomcat Integration – We heavily use Tomcat and many of its features.

Future Plans

- We are evaluating the use of jBPM for business process management.
- We are evaluating how we can use clustering.
- We are evaluating how we can use JBossCache.
- We plan to evaluate using Hibernate in the near future.
- We plan to evaluate using JBoss AOP in the near future.



Questions?

Thanks for coming and feel free to tell us that our approach is nuts!

We'd like to hear how other folks have dealt with these issues.

You can contact us at:

Rick MacConnell (rick.macconnell@autotrader.com)

Sameer Nanda (sameer.nanda@autotrader.com)