- **What is the difference between an Application Server and a Web Server?**

S.NO	WEB SERVER	APPLICATION SERVER
1.	Web server encompasses web container only.	While application server encompasses Web container as well as EJB container.
2.	Web server is useful or fitted for static content.	Whereas application server is fitted for dynamic content.
3.	Web server consumes or utilizes less resources.	While application server utilize more resources.
4.	Web servers arrange the run environment for web applications.	While application servers arrange the run environment for enterprises applications.
5.	In web servers, multithreading is not supported.	While in application server, multithreading is supported.
6.	Web server's capacity is lower than application server.	While application server's capacity is higher than web server.
7.	In web server, HTML and HTTP protocols are used.	While in this, GUI as well as HTTP and RPC/RMI protocols are used.

- **What is Catalina?**

  Catalina is Tomcat's servlet container. Catalina implements Sun Microsystems' specifications for servlet and JavaServer Pages (JSP). In Tomcat, a Realm element represents a "database" of usernames, passwords, and roles (similar to Unix groups) assigned to those users. Different implementations of Realm allow Catalina to be integrated into environments where such authentication information is already being created and maintained, and then use that information to implement Container Managed Security as described in the Servlet Specification

- **Describe tomcat directory structure.**

1. **bin**: for Tomcat's binaries and startup scripts.

2. **conf**: global configuration applicable to all the webapps. The default installation provides:
    i. catalina.policy for specifying security policy.
    ii. Two Properties Files: catalina.properties and logging.properties,
    iii. Four Configuration XML Files: server.xml (Tomcat main configuration file), web.xml (global web application deployment descriptors), context.xml (global Tomcat-specific configuration options) and tomcat-users.xml (a database of user, password and role for authentication and access control).
3. The conf also contain a sub-directory for each engine, e.g., Catalina, which in turn contains a sub-sub-directory for each of its hosts, e.g., localhost. You can place the host-specific context information (similar tocontext.xml, but named as webapp.xml for each webapp under the host).
4. **lib**: Keeps the JAR-file that are available to all webapps. The default installation include servlet-api.jar (Servlet), jasper.jar (JSP) and jasper-el.jar (EL). External JARs can be put here such as MySQL JDBC driver (mysql-connector-java-5.1.{xx}-bin.jar) and JSTL (jstl.jar and standard.jar).
5. **logs**: contains the engine logfile Catalina.{yyyy-mm-dd}.log, host logfile localhost.{yyyy-mm-dd}.log, and other application logfiles such as manger and host-manager. The access log (created by theAccessLogValve) is also kept here.
6. **webapps**: the default appBase – web applications base directory of the host localhost.
7. **work**: contains the translated servlet source files and classes of JSP/JSF. Organized in hierarchy of engine name (Catalina), host name (localhost), webapp name, followed by the Java classes package structure.
8. temp: temporary files.

- **Connect any sample.war to MySQL running on localhost.**

  Make changes in the context.xml file and create a resource for your mysql credentials.

```
<Context>

    <!-- Default set of monitored resources. If one of these changes, the    -->
    <!-- web application will be reloaded.                                   -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <WatchedResource>WEB-INF/tomcat-web.xml</WatchedResource>
    <WatchedResource>${catalina.base}/conf/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -->
    <!--
    <Manager pathname="" />
    -->

  <Resource name="jdbc/TestDB" auth="Container" type="javax.sql.DataSource"
              maxActive="100" maxIdle="30" maxWait="10000"
              username="rishabh" password="Rishabh@12" driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/tomcat"/>


</Context>
"context.xml" 38L, 1693C
```

Create the resource reference in web.xml file present in the WEB-INF directory

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <display-name>Hello, World Application</display-name>
    <description>
        This is a simple web application with a source code organization
        based on the recommendations of the Application Developer's Guide.
    </description>

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>mypackage.Hello</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
<resource-ref>
    <description>Connection Pool</description>
    <res-ref-name>jdbc/TestDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>

</web-app>
~
```

Created a file hello.jsp to display results

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<sql:query var="rs" dataSource="jdbc/TestDB">
select id, foo, bar from testdata;
</sql:query><html>
<head>
<title>Sample Application JSP Page</title>
</head>
<body bgcolor=white>
<h1>Sample Application JSP Page</h1>
<c:forEach var="row" items="${rs.rows}">
        ${row.id}     ${row.foo}     ${row.bar}<br/>
</c:forEach>

</body>
</html>
~
~
```

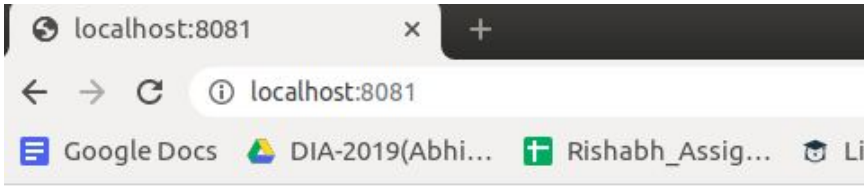
Entries in table

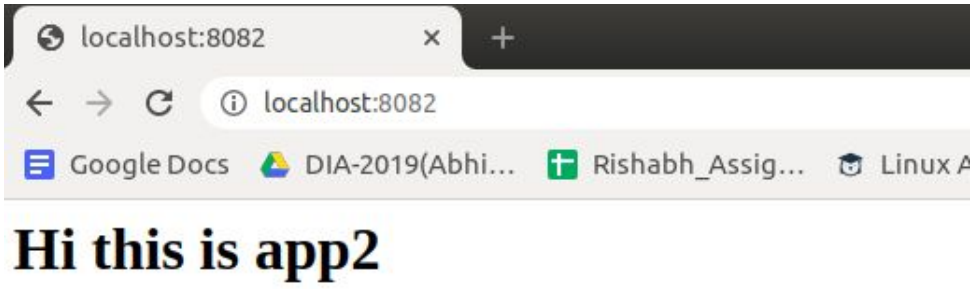
```
mysql> select * from testdata;
+----+----------+-------+
| id | foo      | bar   |
+----+----------+-------+
|  1 | hello    | 12345 |
|  2 | 2nd line |  1443 |
+----+----------+-------+
2 rows in set (0.00 sec)
```

- **Run multiple services on different ports with different connectors (AJP/HTTP) on same tomcat installation.**

Make changes in the server.xml

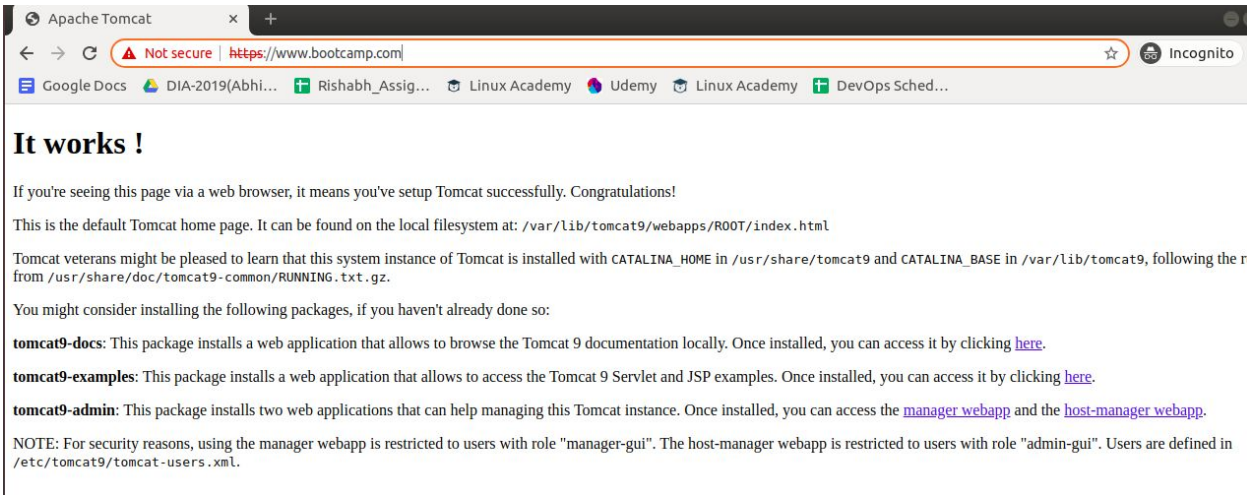
```
  <Service name="app1">
   <Connector port="8081" protocol="org.apache.coyote.http11.Http11NioProtocol"
              connectionTimeout="20000"
              redirectPort="8443" />
   <Engine name="Catalina" defaultHost="localhost">
      <Host name="localhost"  appBase="app1"
        unpackWARs="true" autoDeploy="true">
      </Host>
   </Engine>
</Service>
<Service name="app2">
   <Connector port="8082" protocol="org.apache.coyote.http11.Http11NioProtocol"
              connectionTimeout="20000"
              redirectPort="8443" />
   <Engine name="Catalina" defaultHost="localhost">
              <Host name="localhost"  appBase="app2"
         unpackWARs="true" autoDeploy="true">
      </Host>
   </Engine>
</Service>
</Server>
```





- **Use nginx as reverse proxy for tomcat application.**
    - **Setup self signed certificate on that nginx for bootcamp.com.**

```
server{
        listen 80;
        server_name www.bootcamp.com;
return 302 https://www.bootcamp.com;
}
server{
listen 443 ssl;
server_name www.bootcamp.com;
        ssl_certificate /etc/nginx/ssl/nginx.pem;
        ssl_certificate_key /etc/nginx/ssl/nginx.key;
location /{
                proxy_pass http://127.0.0.1:8080;
        }
}
~
```



    - **What is the difference between proxy_pass & proxy_pass reverse?**

      Proxy_pass is the most straight-forward type of proxy involves handing off a request to a single server that can communicate using http.

      The ProxyPassReverse is used to change the headers sent to Apache from a proxied app server, before Apache sends it to the browser. For example, if the app sits at Page on localhost:8080, and it tries to redirect the browser to, say, /new_path/, then it will respond with a redirect and location header of

http://localhost:8080/new_path/, and Apache will take this and send it off to the browser. The issue is that the browser will then try to send a request to Page on localhost:8080and receive an error.

What ProxyPassReverse can do is intercept those headers, and rewrite them to match the Apache proxy server .