A Theory and Approach for Developing Cloud Applications

Danko Naydenov

Abstract: *In this paper, a methodology will be considered that can be used to create cloud applications. There are many possible approaches, but this one was chosen because it is easy to be used and accessed. System requirements overview is made. For selected approach it is considered some application programing interfaces for user maintenance and authentication, storing data and deploying the application. It is discussed how standard technologies can be apply to this service also an overview of some security requirements is made.*

Keywords: *Cloud computing, PaaS, Cloud application*

Introduction

With the development of information technology, users and developers turn to cloud services. The reasons for this are several, namely the low cost of initial investment, the high availability and the reliability of applications, the low cost of maintenance and others [6].

Goals and objectives

The goal that sets this paper is to demonstrate a methodology for developing cloud applications.

To achieve this goal, a task is defined to develop a sample application representing a chat room. This application has to be designed for cloud environment. It has to be deployed there and has to work in such an environment.

Analysis of the problem

On the market, there are many cloud vendors and various cloud services that can be grouped into three main models [6]:

Infrastructure as a Service (IaaS)
Platform as a Service (PaaS)
Software as a Service (SaaS)

The most appropriate service for the development of applications is PaaS. The reason for this is that the approach which is used to create a cloud application largely resembles the development of a standard WEB application.

The provider selected is Google and their offering PaaS - Google App Engine. This choice is based on following factors:

- Cloud computing is a relatively new technology, and offering such a service requires a significant investment, not a variety of providers are available, and participants are giants like Microsoft, Amazon, Google and others;
- Google prevailed in the election because they currently offer free access to develop, test and deploy cloud applications;
- The cloud service provided by Google offers the possibility to use several languages. This paper will focus on Java [3];

System requirements and installation of work environment

It is assumed that the developer has an installed and working version of Java [3]. Because the Google App Engine works using Java 7, the cloud service provider recommends using the same version to build the application.

To develop cloud application, Google offers App Engine Java Software Development Kit (SDK). This is an environment that emulates all aspects of the cloud, which however, works locally and allows the used to develop and test the application.
App Engine Java SDK is a standalone application that does not have a high degree of automation. Eclipse [2] is one of the most popular development environments for Java [3] applications, and that is why Google offers a Plugin for it. Developing application for Google App Engine is easy and similar to the development of a servlet [5] application. By using the Google Plugin, this task is even easier because it allows all actions to be performed directly from the development environment.

Installing the Google Plugin for Eclipse

At the time of development of this paper, the current version of the development environment is Eclipse Kepler (4.3.2). To install the Plugin, one has to perform the following actions:

1. Choose from „Help” menu „Install New Software...”
2. In newly open window in field „Work with” is entered:
   https://dl.google.com/eclipse/plugin/4.3

The next activity is pressing the button „Add...” right to the field. Confirm with „OK” for the new window. Field „Name” leave blank, as it will be filled later by page of Plugin.

3. After loading the table, press the icon to expand two headers next to "Google Plugin for Eclipse" and "SDKs". Place a check mark in the boxes with the following elements: "Google Plugin for Eclipse 4.3" and "Google App Engine Java SDK". The user has to ensure that a checkbox next to "Contact all update sites during install to find required software" is selected. Press the "Next>" and follow the instructions on the installation.
4. When the installation is complete, the working environment Eclipse will prompt to reboot. After restarting the environment, new additions will be active and available.

Creating a Java App Engine project

Google has chosen Java Servlet [5] technology to be one of its cloud platforms service. This determines the way the application will interact with the server.
To create a new project from "File" menu select "New / Other ...". From the opened dialog choose the section "Google" and then "Web Application Project". After pressing the "Next>" a dialog is opened to define a new project. As name of the project can be set "ChatRoom" and for the package name "com.cc.chatroom". Additional settings will be: deselect a "Use Google Web Toolkit" and select "Use Google App Engine". After clicking the "Finish" button the structure of the sample project is automatically created. Main content is divided in two directories:

src – contains application code files;

war – contains the compiled application in WAR format (WAR archive is not supported)

The newly created application can be launched from the menu "Run" by selecting "Debug As> Web Application", and it can be accessed via a standard browser at:

http://localhost:8888/

Authentication of users

By creating a standard project, a servlet is also created with the name of the project, which is registered in the "web.xml" file. In this example it is in "ChatRoomServlet.java". This is a

class that inherits "javax.servlet.http.HttpServlet" class. Initially, the effects of this class are to handle a GET request and display as text message "Hello, world".
This servlet will be modified so that a POST request is forwarding to be processed by function processing GET request.
Authenticating the users is a major task for any application and can be very complex. When creating applications for Google App Engine, it is possible to use the authentication provided by Google. This means that instead of writing a code in an application that is being developed to take care of the maintenance of users, this task is given to Google. For this purpose, if a user wants to work with the application, it must be registered first and have a valid Google account. Code that makes authentication in this case is rather short:

```
public class ChatRoomServlet extends HttpServlet {
  public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws IOException {
    doGet(req, resp);
  }
  public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws IOException {
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();
    if (user != null) {
      resp.getWriter().println(
        "<p>Welcome " + user.getNickname()
        + ". To sign out click " + "<a href="
        + userService.createLogoutURL(req.getRequestURI())
        + ">here</a>.</p>");
    } else {
      resp.getWriter().println(
        "<p>Welcome to ChatRoom. You can sign in from <a href="
        + userService.createLoginURL(req.getRequestURI())
        + ">here</a></p>");
    }
  }
}
```

UserService is interface that gives access to the main tasks of identifying the user. `getCurrentUser()` returns an object for current user. If this object is null, this means that there is no currently registered user. With `userService.createLoginURL( req.getRequestURI())` browser is redirected to Google page for user authentication. When using a local server for development and testing, it is possible cases when Internet is not available and in the cases the browser is redirected instead of Google login form to a standard form where the authentication is always possible only with e-mail address. The parameter that has been passed to the function `createLoginURL`, is an address which will be used to redirect the browser after successful authentication. For logout it is used other function `createLogoutURL`, and the parameter again is return address.

The logic of demonstrating code is following:

- if there is currently authenticated user it is displayed the message "Welcome" followed by the name of the user. Then an unsubscribe link is present;
- if there is no currently authenticated user it is displayed "Welcome to ChatRoom", following by the link to authenticate the user;

Dynamic content

Using servlets allows generation of dynamic content in pages, but unfortunately when there is the need to generate more complex pages, the task can become quite difficult. To solve this problem, multiple template systems for Java are developed. These are systems that combine static HTML content and dynamically changing fragments. Thus, complex layouts on a page can be generated separately and independently, even using assistive environments and tools, and specific content can be added at the designated places. In this paper, we will look at JavaServer Pages (JSPs) [1], as one of the most popular systems, at the same time supported by Google App Engine.

By default, all files with the extension .jsp from the war directory and all subdirectories except WEB-INF are mapped to the corresponding URLs.

To use JSPs, Eclipse must be configured to use the JDK, not the JRE. This can be done from the menu: Window > Preferences > Java > Installed JREs.

Similar result to that obtained from already discussed servlet can be obtained from a JSP template. This is a file that can be placed in the war directory and that can be given the name index.jsp. Although the file is named index.jsp, this does not mean that it will be loaded by default. To become this true a modification is required for the configuration file web.xml located in the war / WEB-INF directory. It is needed to update the contents of the tag <welcome-file>:

```
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

The main fragment of code that performs authentication of users using JSP templates is as follows:

```
...
<body>
  <%
  UserService userService = UserServiceFactory.getUserService();
  User user = userService.getCurrentUser();
  if (user != null) {
    %>
    <p>Welcome <%=escapeHtml3(user.getNickname()) %>. To sign out click <a
href="

<%=userService.createLogoutURL(request.getRequestURI())%>">here</a>.</p>
    <%
  } else {
    %>
    <p>Welcome to ChatRoom. You can sign in from <a href="
    <%=userService.createLoginURL(request.getRequestURI())%>">here</a>.</p>
    <%
  }
  %>
</body>
...
```

An important thing that needs to be addressed is the output of the text on the screen of the browser. In this case, it is the user's name. If it works as demonstrated in servlet `user.getNickname()`, this can lead to security problems. The reason for this is that if a malicious user as its name defines some HTML code, it will be put in the output, and will be interpreted by the browser. To avoid security problems of this nature, it is recommended that all strings are escaped, which will be displayed to the screen and the application does not have control of their content. There is a standard library Apache - Commons Lang [7], which has a similar function. To use this feature in application, one has to place the library file in war\WEB-INF\lib directory and include the library that contains the function with following line:

```
%@page import=" org.apache.commons.lang3.StringEscapeUtils. escapeHtml3"%
```

The function is named `escapeHtml3`, and by using it, all strings are converted so that they are displayed as text in the browser.
As a result, it is possible to display any text to the browser, even one that contains some HTML, and it will be transformed into the escape sequence:

```
<%=escapeHtml3(user.getNickname()) %>
```

Another important part of the web.xml file is the association of servlet to a physical address. If we define a new servlet in the application this does not automatically make it visible and accessible to the client browser. In order to make this servlet accessible via some address and to receive GET or POST requests, this servlet needs to be specified in the configuration file web.xml. Fowling example will demonstrate how this can be done for servlet that will receive messages sent to the server by users. Servlet will be defined in the file SendMessageServlet.java.

```
<servlet>
  <servlet-name>send</servlet-name>
  <servlet-class>com.cc.chatroom.SendMessageServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>send</servlet-name>
  <url-pattern>/send</url-pattern>
</servlet-mapping>
```

The first section `<servlet>` describes the name and which class will process requests.

The second section `<servlet-mapping>` specifies the address where the servlet will be available.

So, define and describe servlet will be activated from a HTML form which will be placed in the index.jsp:

```
<form action="/send" method="post" id=frm>
<pre id="output" style=
"height: 300px; width: 600px; overflow: auto; border: 1px solid black;">
</pre>
<input id="input" name="input" type="text" maxlength=300
  style="width: 600px"/><br/>
<input type="submit" value="Send"/>
<script type="text/javascript">
  document.forms.frm.input.focus();
</script>
</form>
```

The method that has been used to send a request to servlet is POST and action="/send" set the address where the servlet is available.

Data handling

The cloud infrastructure is a quite different architecture from normal PCs and even servers. If an application communicates with one node from the cloud infrastructure for some time, then the next requests can be redirected to another node. Therefore, it is difficult to determine exactly where to save the data so that they are available for next call. In addition, there are actions for backup the data in case of failure so that they can be restored in such cases.

To ignore all the details related to the storage and maintenance of data, Google App Engine offers a service called Datastore. This is a set of functions enabling storing and retrieving the data using simple API.

In the Datastore, a recorded unit of information is called entity. Each entity consists of a key and a value. The key must be unique, and it is used to identify the entity. The value can be any object.

When the entity has been saved in the Datastore, this leads to number of actions related to the sharing of data between data centers, backup, and many other activities related to cloud architecture, all of which is hidden from the developer.

To save a message which had been sent to chat room, it is necessary to modify the servlet SendMessageServlet like this:

```
String content = req.getParameter("input");
Date date = new Date();
Entity message = new Entity("Message");
message.setProperty("user", user);
message.setProperty("timestamp", date);
message.setProperty("text", content);
DatastoreService datastore =DatastoreServiceFactory.getDatastoreService();
datastore.put(message);
resp.sendRedirect("/index.jsp");
```

- the first row retrieve entered text by the user;
- the second line creates an object representing the current system time which is used as a characteristic of the message;
- the third line creates an item that will be recorded in the Datastore. This element is of type "Message", and don't contain a key value. After saving the item a unique key value will be assigned;
- each element can have any number of properties. In describe case these three properties are: user, timestamp of message and the message itself (lines 4 to 6);
- actual recording is on row number eight: `datastore.put(message)`;
- the final task is to redirect the browser to the default template;

Retrieving of data is related to the execution of the following sequence:

- creating an object which define the search query;
- create a list of items corresponding to the request;

In the code that looks like this:

```
DatastoreService datastore = DatastoreServiceFactory
    .getDatastoreService();
Query query = new Query("Message").addSort("timestamp",
    Query.SortDirection.ASCENDING);
```

```
Iterable<Entity> messages = datastore.prepare(query).asIterable();
```

Object `query` retrieves all elements of type "Message", using the characteristic "timestamp" to perform sorting.

`messages` is an object containing all the elements meeting the request.

Access to individual fields can be done as follows:

```
for (Entity message : messages) {
  ...
  ...escapeHtml3(message.getProperty("text").toString())...
  ...
}
```

Again, it should be kept in mind security, by simply using the function to convert special characters, if any, in the entered text.

Sending a message is done by submitting the page and passing the control to the servlet processing this request. Receiving information from the server side is more complicated. If this is done again with submit this can lead to loss of Writer current message. It is therefore necessary to use an asynchronous request to the server – Ajax [4]. In this case, the background will ask a server for current messages from the chat room. After receiving the response, all messages will be recorded in the appropriate controls. This whole concept is implemented using JavaScript as follows:

```
var req = null;
if (window.XMLHttpRequest) {
  req = new XMLHttpRequest();
} else if (window.ActiveXObject) {
  req = new ActiveXObject("Microsoft.XMLHTTP");
}
function processRequest() {
  if (req.readyState == 4 && req.status == 200) {
    var message = req.responseText;
    if (message != null && message != "") {
      var outputElement = document.getElementById("output");
      outputElement.innerHTML = message;
      outputElement.scrollTop = outputElement.scrollHeight;
    }
  }
}
function doReceive() {
  req.onreadystatechange = processRequest;
  req.open("POST", "getMessage", true);
  req.setRequestHeader("Content-type",            "application/x-www-form-
urlencoded");
  req.send();
}
```

`req` variable is an object serving to organize asynchronous communication. `doReceive` is a function that specifies which function will handle the response to the query, specifies the servlet which will handle the queries and what is the type of request, and makes the actual call. When a response is receipt, the function `processRequest` is activated to handle this particular answer. After checks for correctness of response and data the response is written as text in graphic control which contains all posted messages.

Deploying the application

The last step of creating a cloud application is to be deployed in cloud infrastructure so it can be available over the Internet. To do so, an application ID has to be created. This identifier can be setup from:

https://appengine.google.com/

To do this, you have to own a valid account for Google. After loading the portal for administrating the applications, the button "Create an Application" has to be pressed. The registered identifier is using to access the application from the Internet. If a free version of the domain is used and assume that the identifier is chatroom, then the application can be accesses from:

http://chatroom.appspot.com/

Deploying the finished application can be done from an Eclipse environment. For this purpose, it is necessary to edit the file appengine-web.xml and set the element `<application>` with registered identifier for this case chatroom. Actual deployment can be done from the toolbar by pressing the Google button and then the option "Deploy to App Engine ...". There are specifying issues for Google account and some options about loaded application into cloud infrastructure.

Conclusion

The process of creating and deploying a cloud Application for Google App Engine largely resembles the same process for standard Java WEB application, although there are some slight differences. The main difference is that cloud applications do not have a standard relational database. Cloud provides a technology for storing data that can be defined as an objective database. This, despite the initial stress in developers, is pretty powerful mechanism for storing data that requires some readjustment in thinking and structuring of an application.

References:

[1] Hans Bergsten; Java Server Pages (December 2000); Publisher: O'Reilly Media

[2] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick; Eclipse Modeling Framework: A Developer's Guide (August 2003); Publisher: Addison-Wesley Professional

[3] David Flanagan; Java in a Nutshell (March 2005); Publisher: O'Reilly Media

[4] Jesse James Garrett; Ajax: A New Approach to Web Applications (February 2005)

[5] Jason Hunter; Java Servlet Programming (November 1998); Publisher: O'Reilly Media

[6] Danko Naydenov, Modern paradigm and trend in Cloud Computing (Jun 2012); Informatics in the scientific knowledge 2012

[7] http://commons.apache.org/lang/

Danko Naydenov
Eurorisk Systems Ltd.
31, General Kiselov Str., 9002 Varna, Bulgaria
E-mail: sky at eurorisksystems dot com