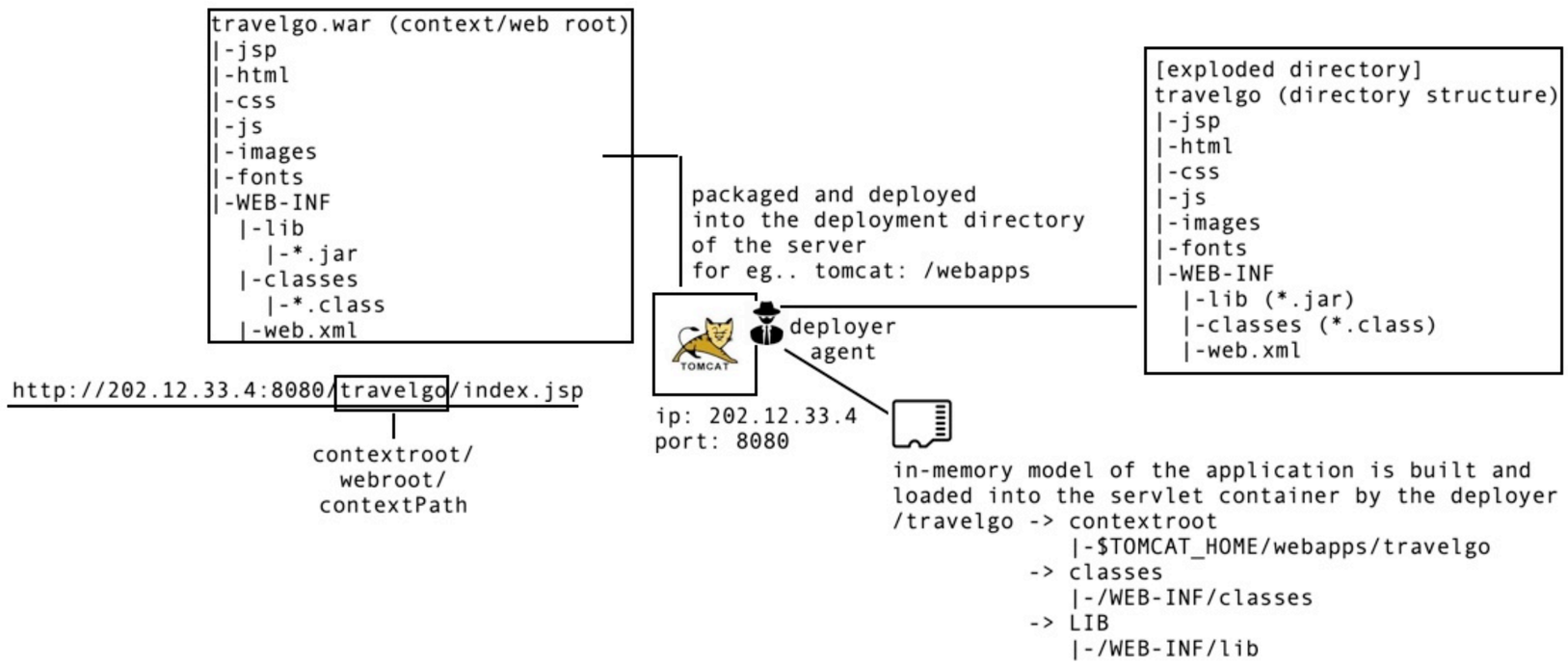

In a standalone servlet container environment, when we package our application as an .war and copy it into the deployment directory of the servlet container, and start it. The servlet containers kickoff the an agent called deployer to deploy or register that application with the servlet container engine.

The deployer agent will goes to the deployment directory of the servlet container. picks one war after another and validates it. if it is valid, then the deployer unpacks/extracts the contents of the war into directory structure under the deployment directory of the container itself. This extracted directory structure of the application is called “exploded directory structure” then it takes the root directory of the exploded application and registers it as an in-memory application model to the servlet container.

From the servlet containers perspective, all that it requires in serving an application is handful of information about the resources of your application like
APPLICATION (CONTEXT_ROOT)
1. where are the static resources are available: WEB_ROOT
2. what is the directory under which the dynamic components of this application exists: CLASSES
3. where are the libraries/jars of this application: LIB
if it has the physical directory locations of these respective resources of the application, the servlet container upon receiving the request can quickly locate these resources and serve the request. It doesnt care how your application is being structured or packaged, it just need these information to serve your application.

How do we pass these info about our application to the container?
That is where the containers provided deployer agents aspart of them, these agent softwares takes our application as an standard (war/ear) extracts and registers them with the container.

While working with embedded servlet container environment, we dont package our application as an “.war” file, because there is no physical location in which we setup the servlet container to deploy. The container itself is being packaged as a library with in our application and is executed as a program through the application itself, so there is no meaning of packaging our application as a War.

since there is not physical existence and no war, there is no deployment agent exists in registering the directories of our application as an in-memory application model into the servlet container. All that we need to do to have our application/its resources to be served by the servlet container is to have the application registerd as in-memory application model inside the container.
So, this job needs to be done programmaticaly by us through our application logic.

```
travelgo
|-src
  |-main
    |-java
      |-Main(..) {}
    |-resources
    |-webapp
      |-jsp
      |-html
      |-css
      |-js
      |-WEB-INF
        |-web.xml
|-target
  |-classes
    |-*.class | (resources)
```

Now through the code, we need to create an StandardContext object, which represents an application that is being deployed into the container.
1. into the StandardContext object we need to add WebRoot directory of the application

```
Tomcat tomcat = new Tomcat(8080);
File webRoot = new File(“src/main/webapp”);

StandardContext stdContext = (StandardContext) tomcat.addWebApp(“/travelgo”, webRoot);
```

The StandardContext object is representing the WebResourceRoot of our application, so create the WebResourceRoot from the StandardContext object as below

2.
```
WebResourceRoot webResourceRoot = new StandardRoot(stdContext); // root directory of our application
```

into the WebResourceRoot we need to add the classes directory of our application where the class files of our application are under target/classes.

```
webResourceRoot.addPreResources(new DirResourceSet(webResourceRoot, “/WEB-INF/classes”, new File(“/target/classes”).getAbsolutePath(), “/“);

stdContext.setResources(webResourceRoot);
tomcat.start();
tomcat.getServer().await();
```