

Jenkins – Continuous Deployment

Jenkins Continuous Deployment

- This session we will see how to deploy a web app to Tomcat server after the build
- Pre-requisites
 - Apache Tomcat 8.5 is installed
 - Configure Tomcat 8.5 to listen to port **8081** or any other port to avoid port conflict with Jenkins
 - **Note that this is done only in training sessions**
 - **In production, Jenkins and Tomcat are deployed on separate machines**

Tomcat Configuration – change port

- Under Tomcat Installation Directory, Navigate to '**Conf**' directory and open '**Server.xml**' file
- Better to use an editor like **notepad++**
- Edit the file and change the port number from 8080 to 8081
- Then restart Tomcat service
- Open a browser and type <http://localhost:8081> you should see Tomcat initial page

```
Java AJP  Connector: /docs/config/ajp.html
APR (HTTP/AJP) Connector: /docs/apr.html
Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
-->
<Connector port="8081" protocol="HTTP/1.1"
          connectionTimeout="20000"
          redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
<!--
<Connector executor="tomcatThreadPool"
          port="8080" protocol="HTTP/1.1"
          connectionTimeout="20000"
          redirectPort="8443" />
```

Tomcat – User Configuration

- We need to add a Tomcat User who's credentials will be used by Jenkins to deploy the build
- Navigate to Tomcat Installation Directory → Conf
- Edit *tomcat-users.xml*
- Make an entry for user with role as *manager-script*

```
<user username="deployer" password="deployer" roles="manager-script" />
```



```
</tomcat-users>
```

Jenkins – add Deployment Plugin

- Go to Manage Jenkins → Manage Plugins
- Select the “Available” tab, locate the “[Deploy to container](#)” plugin and install it.

The screenshot shows the Jenkins Manage Plugins interface. The 'Available' tab is selected. A search filter 'container' is applied. The 'Deploy to container Plugin' is highlighted with a red box. The 'Install without restart' button is also highlighted with a red box.

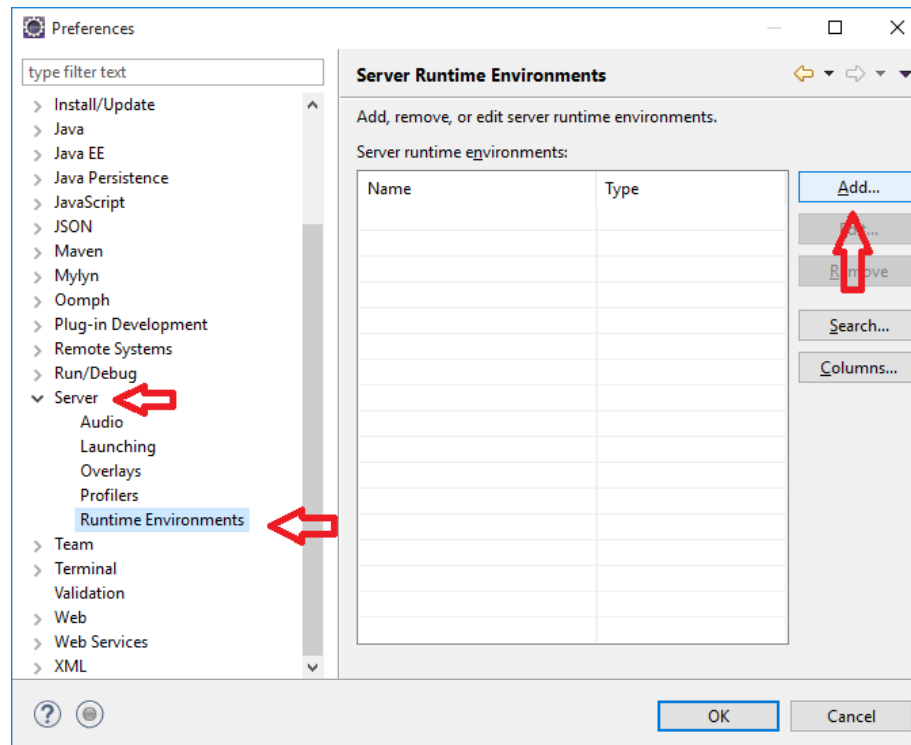
Install	Name	Version
<input type="checkbox"/>	Amazon EC2 Container Service plugin Jenkins plugin to run dynamic slaves in a Amazon ECS/Docker environment	1.11
<input type="checkbox"/>	Amazon EC2 Container Service plugin with autoscaling capabilities Jenkins plugin to run dynamic slaves in a Amazon ECS/Docker environment	1.0
<input checked="" type="checkbox"/>	Deploy to container Plugin	1.10
<input type="checkbox"/>	Deploy to websphere container Plugin	1.0
<input type="checkbox"/>	Anchore Container Image Scanner Plugin Integrates Jenkins with the Anchore Image Scanner	1.0.9
<input type="checkbox"/>	Google Container Registry Auth Plugin This plugin exposes a credential for use with the Docker Build Step plugin for authenticating with Google Container Registry as a service account.	0.3
<input type="checkbox"/>	Aliyun-Container-Service-Deploy Enable Jenkins to deploy applications to Aliyun Container Service.	0.1.1
<input type="checkbox"/>	Amazon ECR plugin Integrate Jenkins with Amazon EC2 Container Registry (ECR)	1.4

Buttons: [Install without restart](#), [Download now and install after restart](#), [Check now](#)

Update information obtained: 23 hr ago

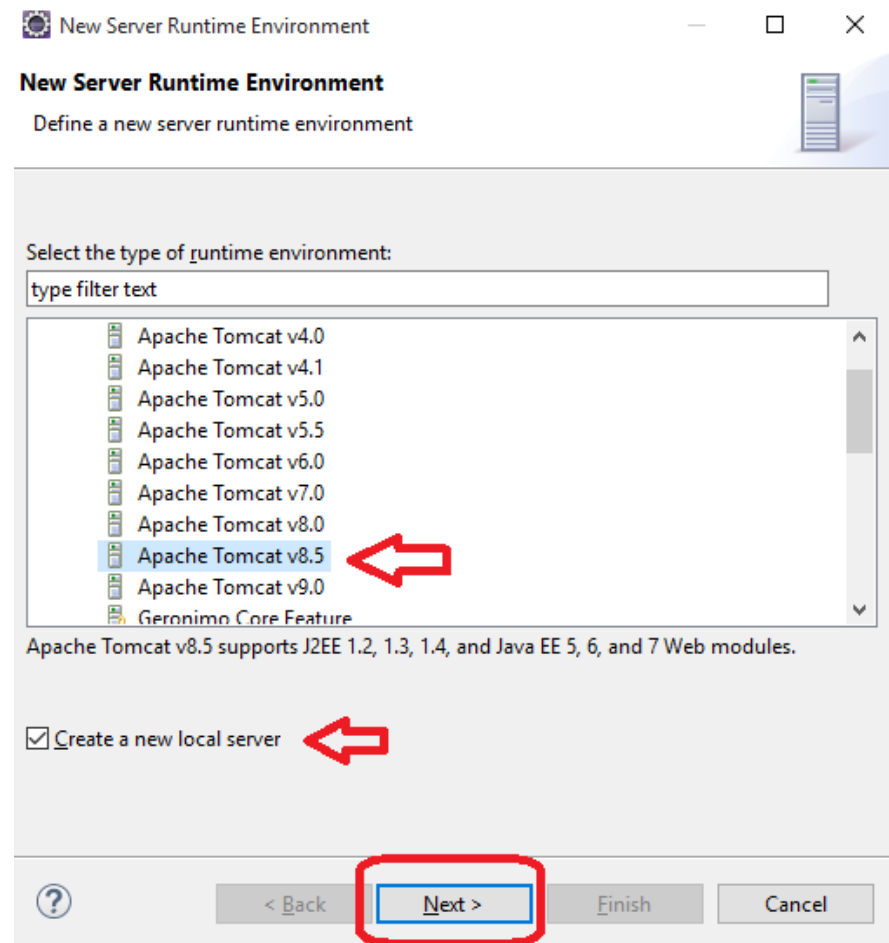
Create a Web App in Eclipse

- Firstly, even before creating a web app in Eclipse, let's create a local Tomcat Server configuration
- In Eclipse, go to *Window* → *Preferences*
- Expand *Server* → Click *on Runtime Environment*
- Click on *Add* button



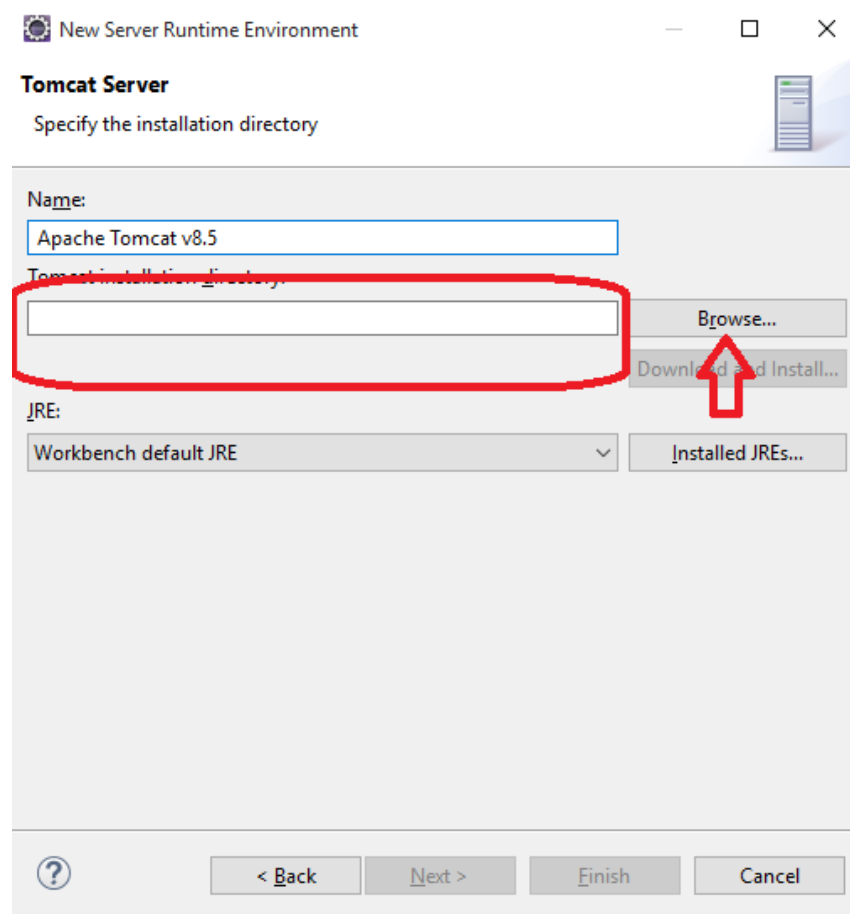
Create a Web App in Eclipse contd

- Select **Apache Tomcat 8.5**
- Check on '**Create Local Server**'
- Click **Next**



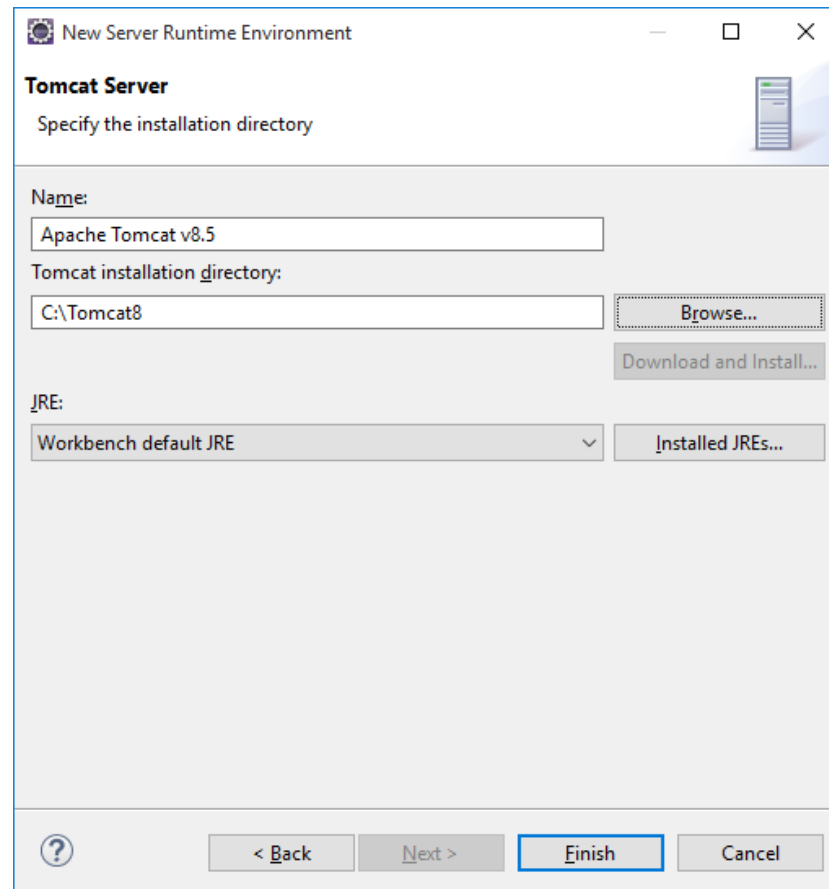
Create a Web App in Eclipse contd

- Use *Browse* button and navigate to *Tomcat Installation Directory*



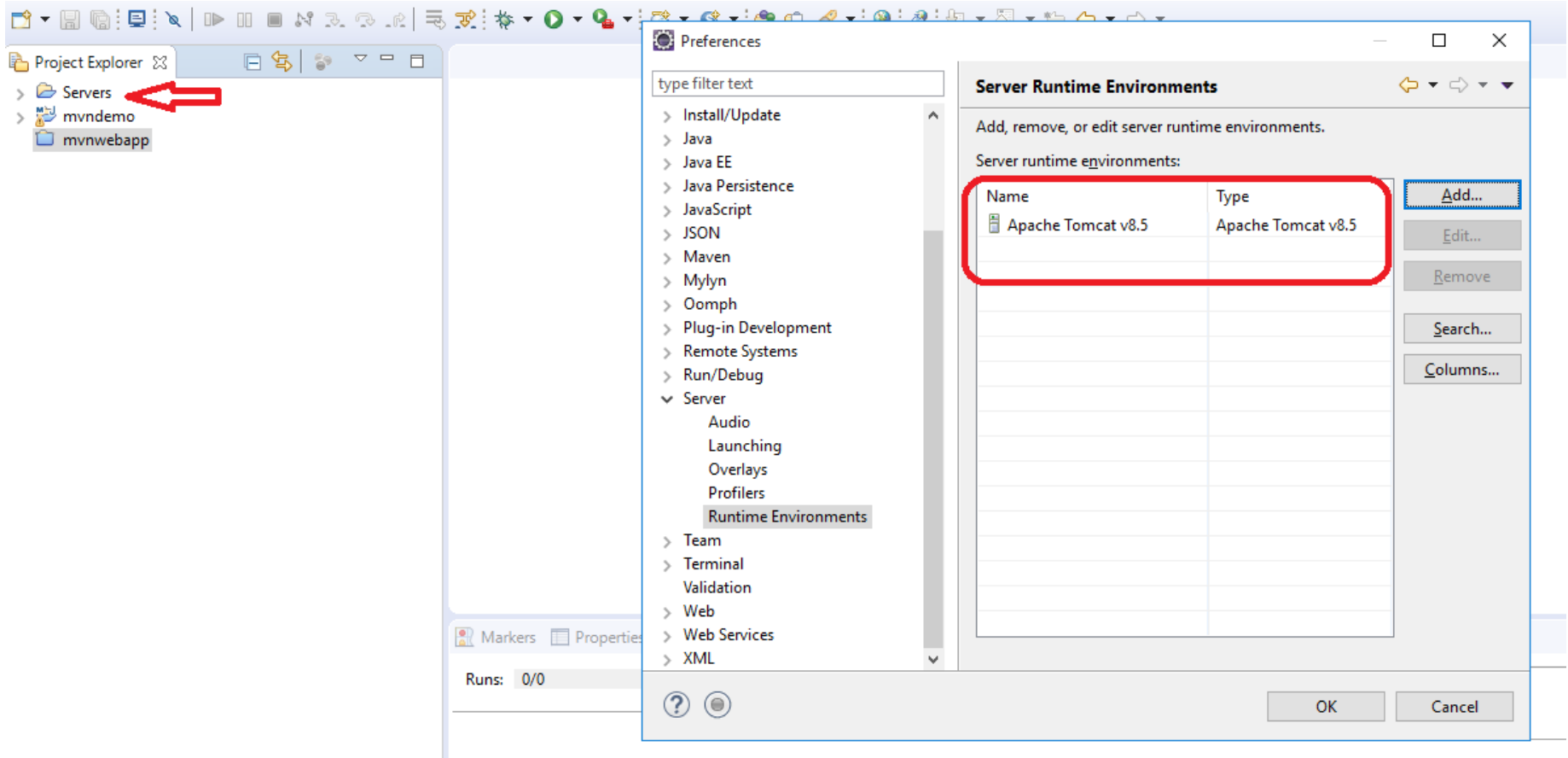
Create a Web App in Eclipse contd

- Click Finish after selecting the Tomcat Installation Directory



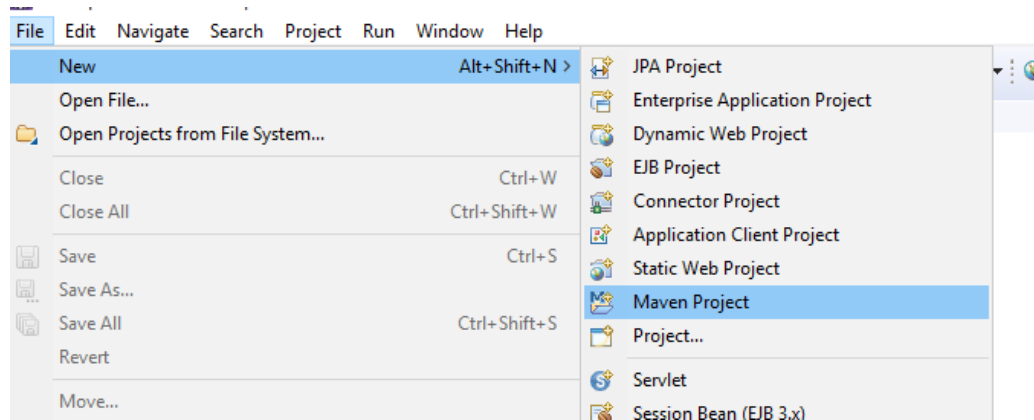
Create a Web App in Eclipse contd

- You can see that *Local Server Run Time Environment* is created from within Eclipse

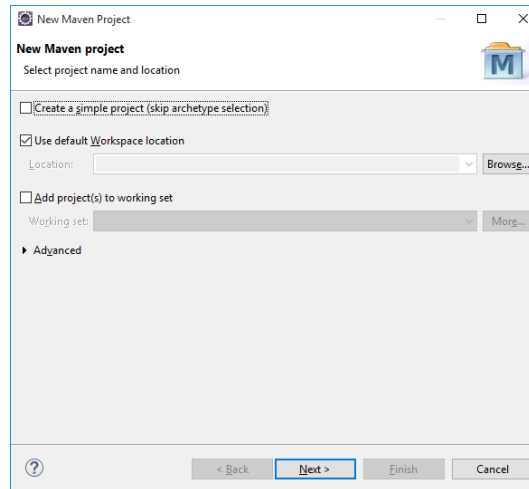


Create a Web App in Eclipse

- Go to File → New → Maven Project

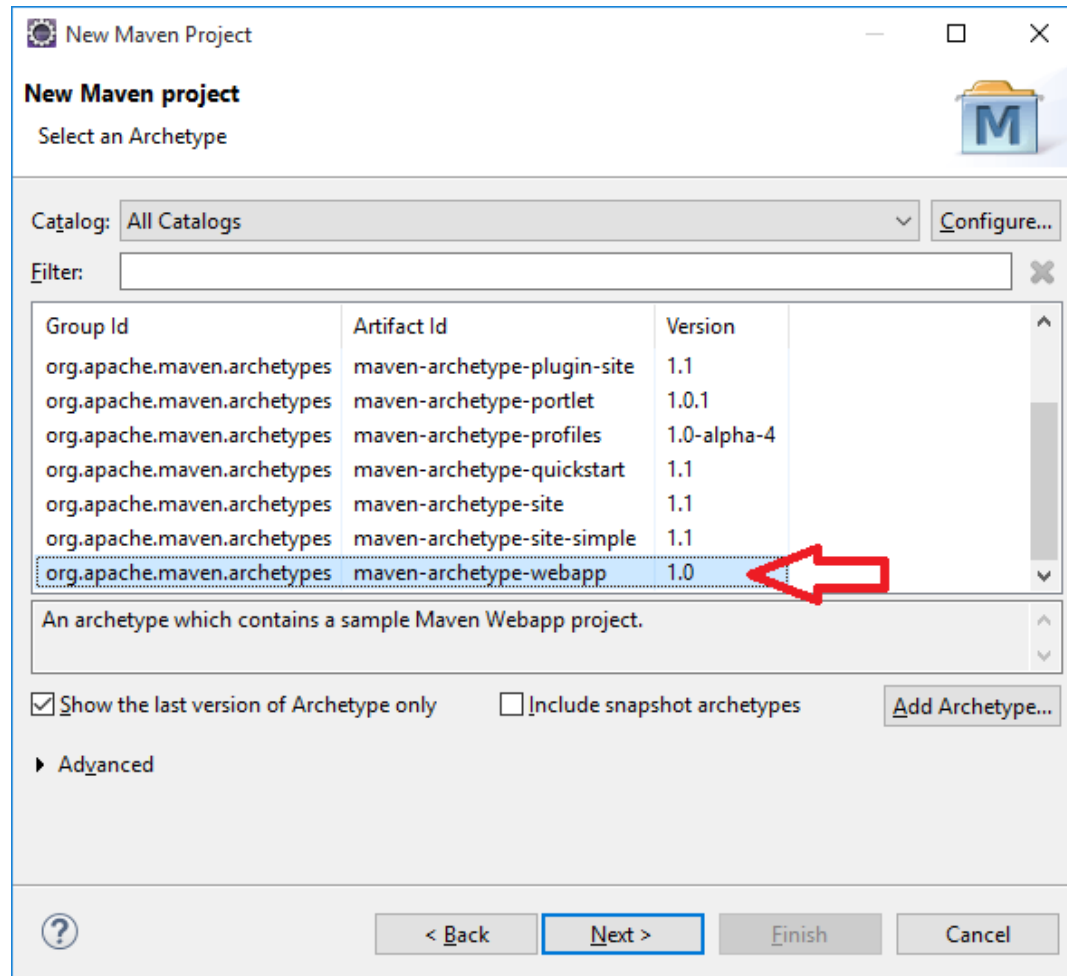


- Use default work space location and click **Next**



Create a Web App in Eclipse contd..

- Scroll down and select **maven-archetype-webapp**
- And click next



Create a Web App in Eclipse contd..

- Provide a Group Id and Artifact Id
- And Finish

New Maven Project
Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

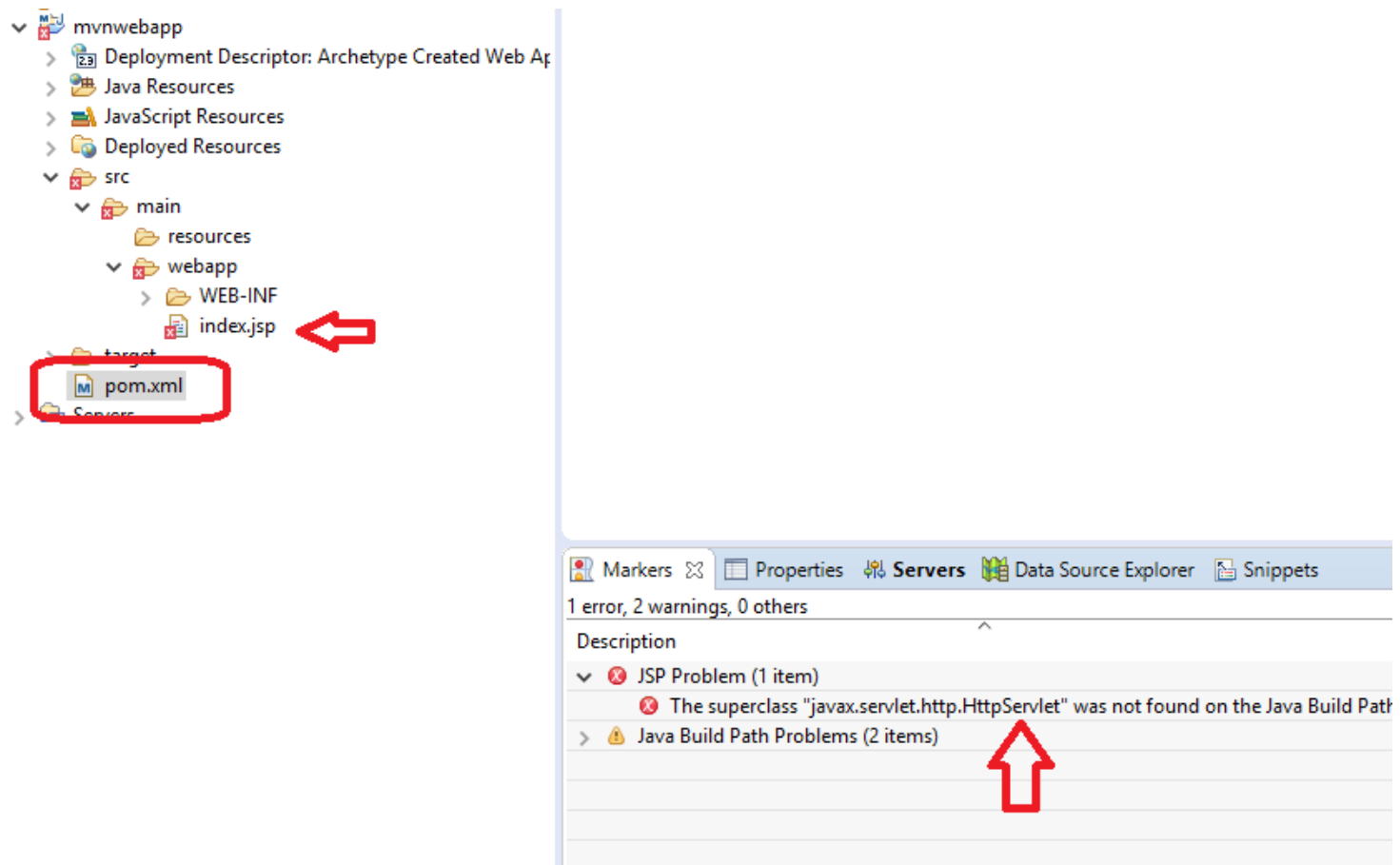
Properties available from archetype:

Name	Value

Advanced

Create a Web App in Eclipse contd..

- If you get a compilation error, saying **HttpServlet** is not on classpath
- Open **pom.xml**



Create a Web App in Eclipse contd..

- Add dependency in [pom.xml](#) for Servlet API and Save

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

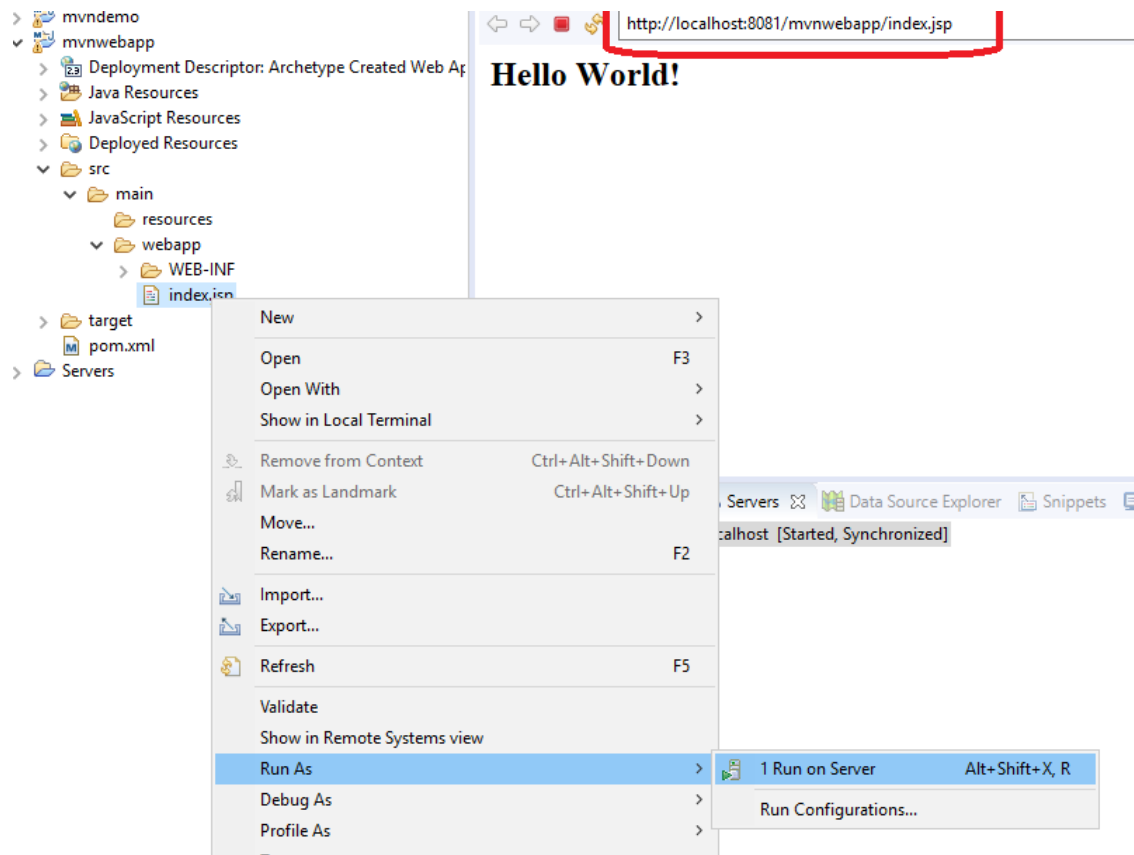
<version>3.0.1</version>

</dependency>

```
10 <dependencies>
11   <dependency>
12     <groupId>junit</groupId>
13     <artifactId>junit</artifactId>
14     <version>3.8.1</version>
15     <scope>test</scope>
16   </dependency>
17
18   <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
19   <dependency>
20     <groupId>javax.servlet</groupId>
21     <artifactId>javax.servlet-api</artifactId>
22     <version>3.0.1</version>
23   </dependency>
24
25 </dependencies>
```

Test Locally

- Go to Services console and make sure the external tomcat is not running.
- You can see the [index.jsp](#) displayed in Eclipse internal browser
- So the web app is working fine!



Push the code to Git Repository

- On Bitbucket lets create a new Repository and call it *mvnwebapp*

Create a new repository

[Import repository](#)

Owner

AnandR72

Repository name *

mvnwebapp

Access level

☒ This is a private repository

Repository type

☒ Git
☐ Mercurial

> Advanced settings

Create repository

Cancel

Push the code to Git Repository contd..

- Open git bash terminal
- `$ cd workspace/mvnwebapp/`
- `$ git init`
- `$ touch .gitignore`
- `$ vim .gitignore`

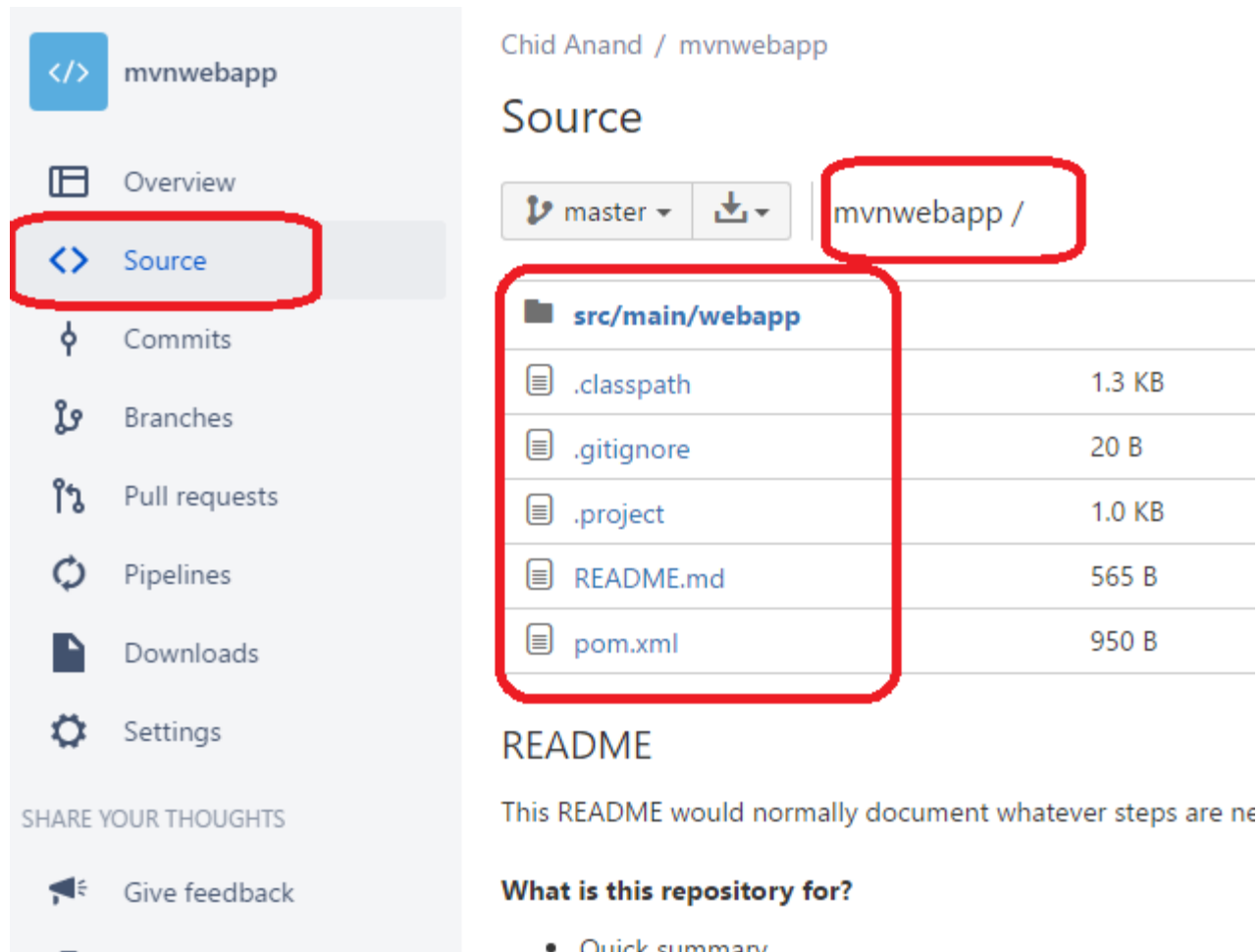
Add the following lines to `.gitignore` file

`.settings/`
`target/`

- `$ git status`
- `$ git add .`
- `$ git remote add origin`
`https://AnandR72@bitbucket.org/AnandR72/mvnwebapp.git`
- `$ git pull origin master`

Push the code to Git Repository contd..

- \$ git commit -m'initial'
- \$ git push origin master
- You can check the repo on bitbucket to see the code



The screenshot shows the Bitbucket web interface for a repository named 'mvnwebapp' owned by 'Chid Anand'. The left sidebar contains navigation links: Overview, Source (highlighted with a red box), Commits, Branches, Pull requests, Pipelines, Downloads, and Settings. The main content area is titled 'Source' and shows the 'master' branch selected. A red box highlights the breadcrumb 'mvnwebapp /' and another red box highlights the file list under 'src/main/webapp'. The file list includes:

File Name	Size
src/main/webapp/.classpath	1.3 KB
src/main/webapp/.gitignore	20 B
src/main/webapp/.project	1.0 KB
src/main/webapp/README.md	565 B
src/main/webapp/pom.xml	950 B

Below the file list, there is a 'README' section with the text: 'This README would normally document whatever steps are ne'. At the bottom, there is a section titled 'What is this repository for?' with a bullet point: '• Quick summary'.

Jenkins – create a build job

Enter an item name

» Required field



Freestyle project



This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Maven project

Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.



External Job

This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.



Multi-configuration project

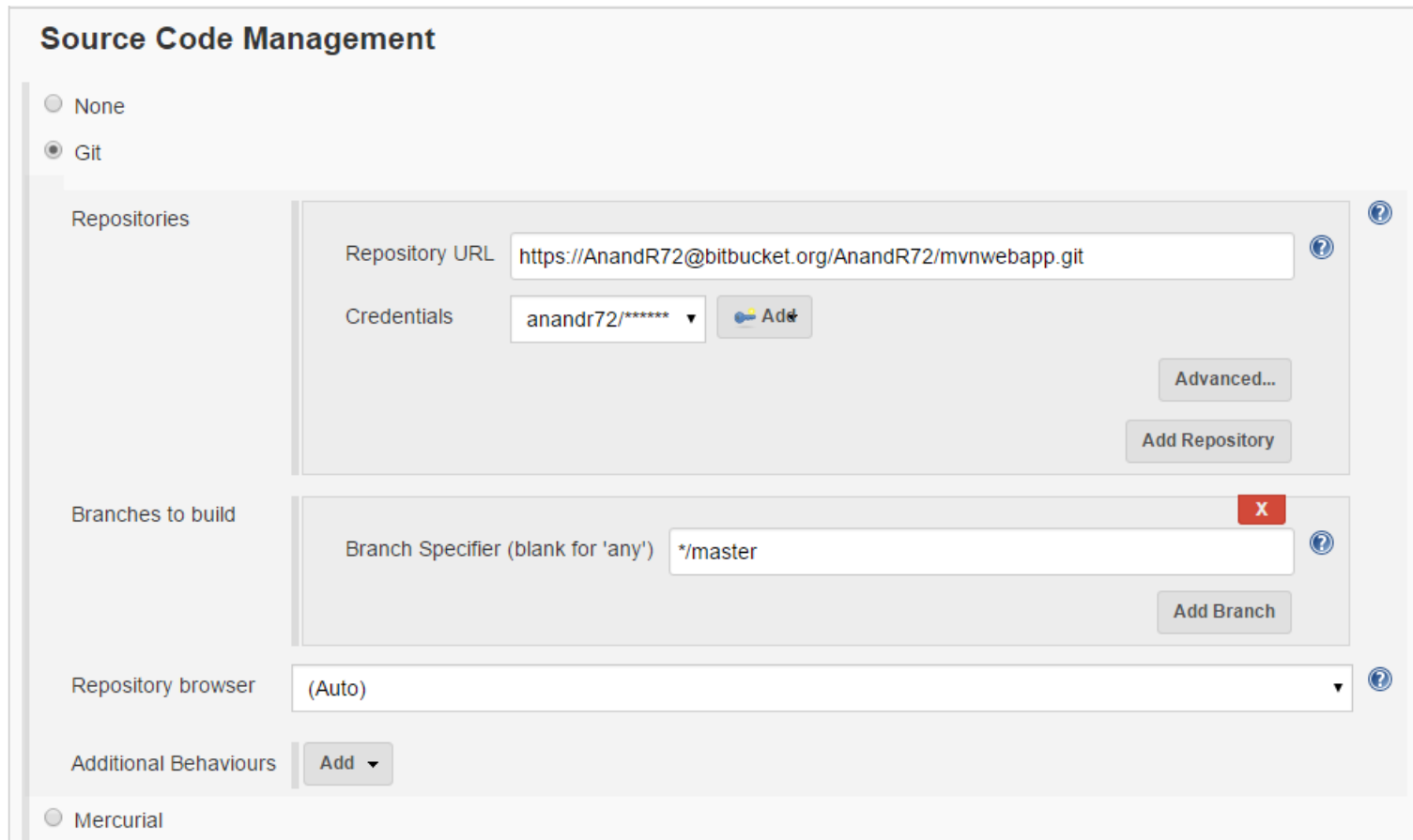
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Jenkins – create a build job contd..



The image shows the 'Source Code Management' configuration page in Jenkins. It features a sidebar with radio buttons for 'None', 'Git' (selected), and 'Mercurial'. The main area is divided into sections: 'Repositories' with fields for 'Repository URL' (https://AnandR72@bitbucket.org/AnandR72/mvnwebapp.git) and 'Credentials' (anandr72/*****), and 'Branches to build' with a 'Branch Specifier' field containing */master. There are also buttons for 'Advanced...', 'Add Repository', 'Add Branch', and 'Repository browser' set to '(Auto)'. The bottom has an 'Additional Behaviours' section with an 'Add' button.

Source Code Management

☐ None
☒ Git
☐ Mercurial

Repositories

Repository URL

Credentials

Branches to build

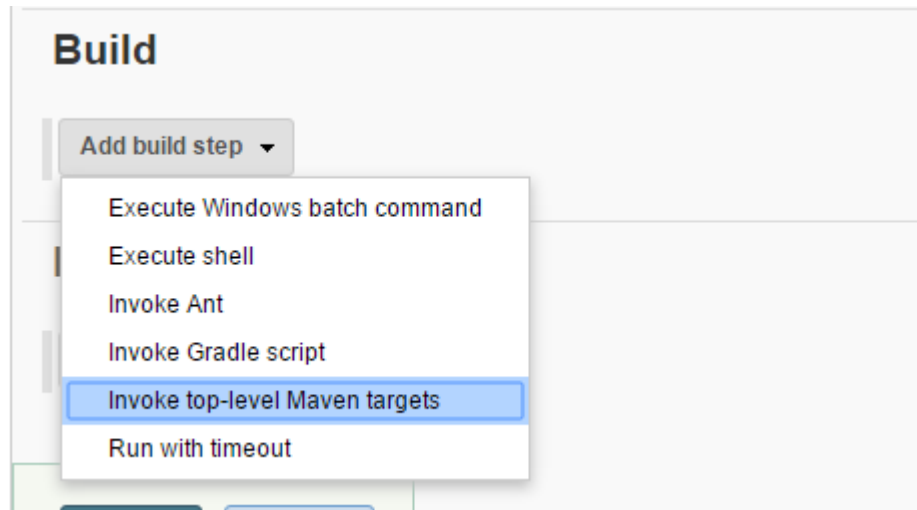
Branch Specifier (blank for 'any')

Repository browser

Additional Behaviours

Jenkins – create a build job contd..

- Add Build Step, and run the build to check



Jenkins – view output

- You can see that build was successful

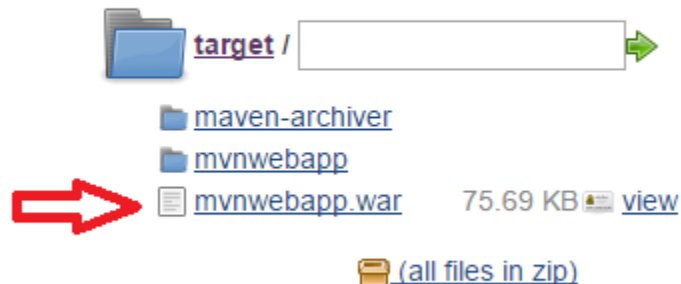
```
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ mvnwebapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ mvnwebapp ---
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory C:\Users\HP\.jenkins\workspace\mvnwebapp\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ mvnwebapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ mvnwebapp ---
[INFO] No tests to run.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.576 s
[INFO] Finished at: 2017-05-10T12:22:33+05:30
[INFO] Final Memory: 2M/110M
[INFO] -----
Finished: SUCCESS
```

Go back to Configure job

- Change the Build Goal from *test* to *package*
- Run the build again and see the difference
- As you can see, *.war* has been generated

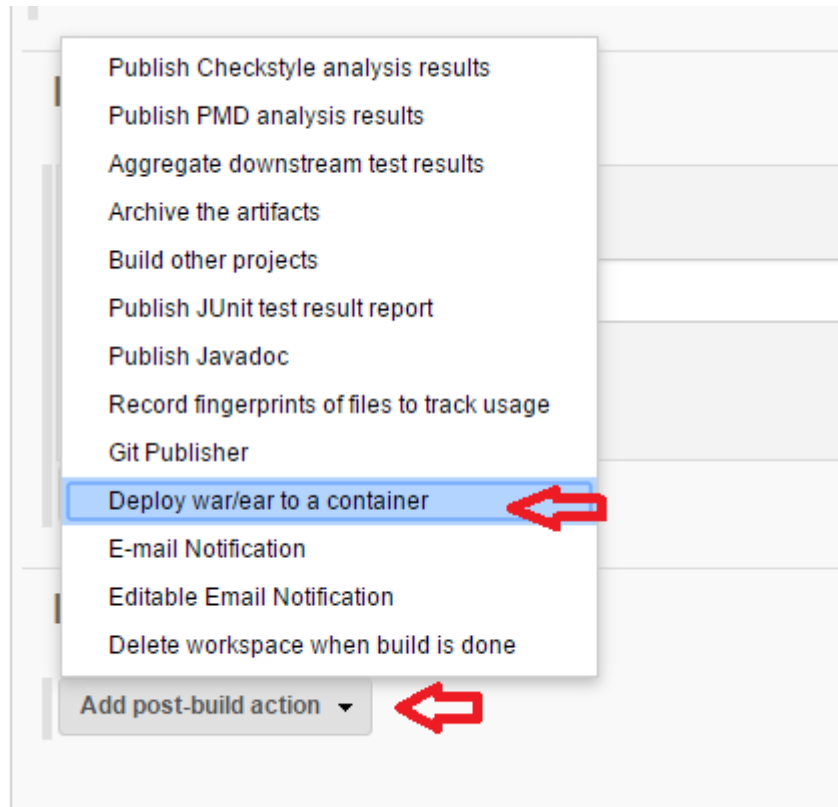


Workspace of mvnwebapp on master



Reconfigure the job for post build action

- Add post-build action – Deploy war/ear to a container



Reconfigure the job for post build action contd.

- Provide the path to war file
- Give a context path to be used for deployment (usually application name)
- Provide Tomcat user credentials for deployment (The entry we made in tomcat-users.xml)

Post-build Actions

Deploy war/ear to a container X

WAR/EAR files ?

Context path ?

Containers

Tomcat 7.x X

Manager user name

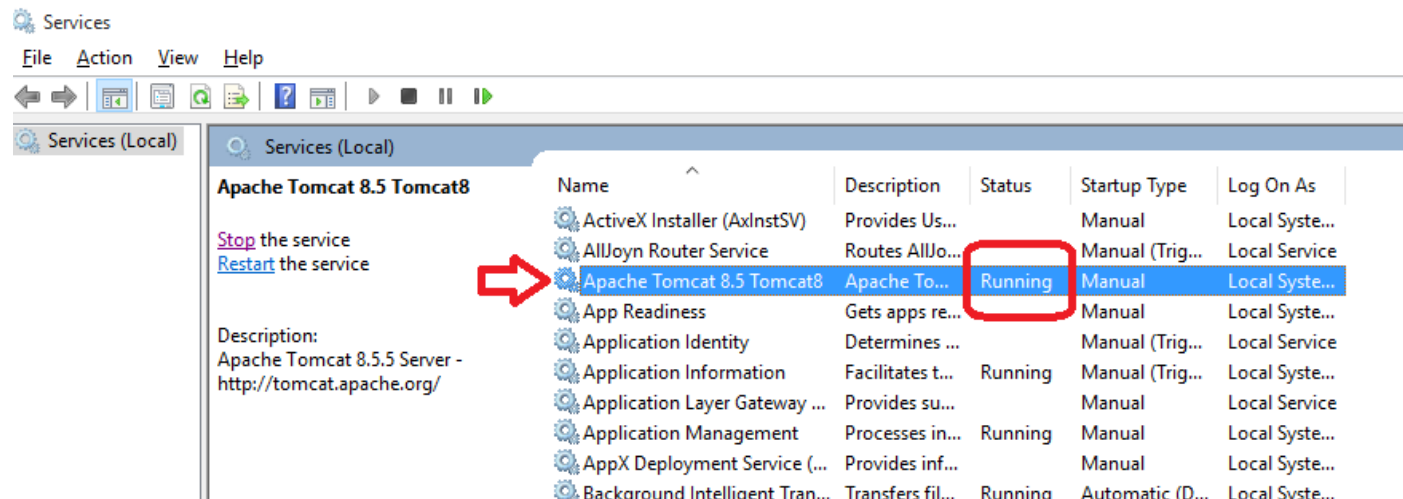
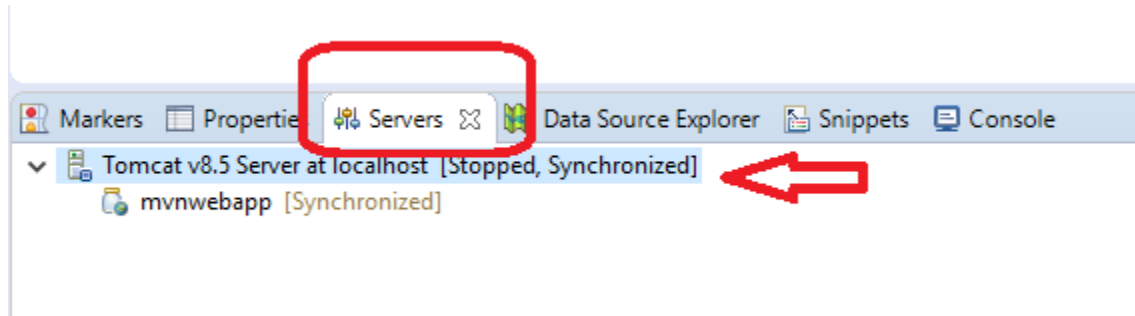
Manager password

Tomcat URL

Deploy on failure ☐

Reconfigure the job for post build action contd.

- Save and Run the build
- If you are running Eclipse's internal Tomcat, please stop the server
- Make sure External Tomcat is running – Go to Services management console and check the status



Check Build console

- You can see that build was successful and deployment was success

```
[INFO] Building war: C:\Users\HP\.jenkins\workspace\mvnwebapp\target\mvnwebapp.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.091 s
[INFO] Finished at: 2017-05-10T12:36:19+05:30
[INFO] Final Memory: 11M/110M
[INFO] -----
```



```
Deploying C:\Users\HP\.jenkins\workspace\mvnwebapp\target\mvnwebapp.war to container Tomcat 7.x Remote
[C:\Users\HP\.jenkins\workspace\mvnwebapp\target\mvnwebapp.war] is not deployed. Doing a fresh deployment.
Deploying [C:\Users\HP\.jenkins\workspace\mvnwebapp\target\mvnwebapp.war]
Finished: SUCCESS
```

- Open a web browser and type the url
<http://localhost:8081/mvnwebapp/>
- You should see 'Hello World' displayed