
Task: Deploying a Java Application with JFrog Artifactory, Tomcat, and Nginx Reverse Proxy

Objective:

The task is to set up a **multi-server environment** consisting of:

- **Build Server**: Where the Java application will be built and the artifacts will be pushed to **JFrog Artifactory**.
- **JFrog Artifactory**: A repository server to store and manage the Java application artifacts (such as `.war` files).
- **Deploy Server**: Where **Tomcat** will run the Java application, and **Nginx** will be configured as a reverse proxy to serve the application.

Prerequisites:

- **Java Web Application**: Use a sample Java web application repository (e.g., Java Web Calculator).
- **Three Servers**:
    - **Build Server**: This server will be responsible for building the application, packaging it, and pushing the artifact to **JFrog Artifactory**.
    - **JFrog Artifactory Server**: This server will store the application artifacts (like `.war` files).

- **Deploy Server**: This server will run **Tomcat** to serve the Java web application, and **Nginx** will be configured as a reverse proxy.

---

1. Java Build Server Setup

Task Overview:

The **Build Server** will clone the Java application repository, build it using **Maven**, and then push the resulting artifact (typically `.war` file) to **JFrog Artifactory**.

Steps:

1. **Clone the Application Repository**:
    - Clone the provided Java application repository to the Build Server.

git clone https://github.com/mrtechreddy/Java-Web-Calculator-App.git
cd Java-Web-Calculator-App

2.
3. **Install Java and Maven**:
    - Ensure **Java** (JDK) and **Maven** are installed on the Build Server.

sudo apt install openjdk-17-jdk
sudo apt install maven

4.
5. **Build the Application**:
    - Use Maven to clean and package the Java application into a `.war` file.

mvn clean package

6.
    - The `.war` file will be located in the `target/` directory of the project.

7. **Configure JFrog Artifactory**:
    - Set up **JFrog Artifactory** for storing Java artifacts.
    - Create an **npm repository** or **Maven repository** in Artifactory to store the `.war` file.

8. **Publish the Artifact to JFrog Artifactory**:
    - Configure the `settings.xml` file to include your **Artifactory credentials** (username and password) and repository details.

Example configuration (`~/.m2/settings.xml`):

```
 <servers>
   <server>
     <id>artifactory</id>
     <username>your-artifactory-username</username>
     <password>your-artifactory-password</password>
   </server>
</servers>
```

9.
    - Deploy the `.war` artifact to **JFrog Artifactory**:

mvn deploy

10. This will upload the artifact to Artifactory.

Deliverables:

- Application successfully cloned and built.

- `.war` artifact pushed to JFrog Artifactory.

---

2. JFrog Artifactory Repository Server Setup

Task Overview:

The **JFrog Artifactory Server** stores the built `.war` artifacts and makes them available for deployment.

Steps:

1. **Install JFrog Artifactory**:
    - Follow the JFrog Artifactory installation guide to install Artifactory on the repository server.
2. **Create a Repository in Artifactory**:
    - Create a **Maven repository** within **Artifactory** to store `.war` files.
    - Log into the **Artifactory UI** and create a **local Maven repository**.
3. **Configure Permissions**:
    - Create a user in Artifactory with proper permissions to push and pull artifacts.
4. **Verify Artifact Storage**:
    - After the **Build Server** pushes the artifact, log into Artifactory and verify that the `.war` file is stored in the repository.

Deliverables:

- JFrog Artifactory installed and configured.

- Artifact uploaded and stored in the Artifactory repository.

---

3. Deploy Server Setup (Tomcat with Nginx as Reverse Proxy)

Task Overview:

The **Deploy Server** is where the application will be deployed using **Tomcat**, and **Nginx** will be used as a reverse proxy to manage traffic to the Java application.

Steps:

1. **Install Java and Tomcat**:
    - Install **Java** (JDK) and **Tomcat** on the **Deploy Server**.

sudo apt install openjdk-17-jdk
sudo apt install tomcat9

2.
3. **Install Nginx**:
    - Install **Nginx** to act as a reverse proxy for Tomcat.

sudo apt install nginx

4.
5. **Configure Nginx**:
    - Create an Nginx configuration file to forward HTTP requests to Tomcat.

Example Nginx Configuration (`/etc/nginx/sites-available/javaapp`):

```
server {
  listen 80;
```

```
    server_name your_domain_or_ip;

    location / {
        proxy_pass http://localhost:8080;  # Tomcat runs on port 8080
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

6.
    - Enable the Nginx configuration:

```
sudo ln -s /etc/nginx/sites-available/javaapp /etc/nginx/sites-enabled/
sudo systemctl restart nginx
```

7.
8. **Fetch the Artifact from JFrog Artifactory**:
    - On the **Deploy Server**, use Maven to fetch the `.war` file from **JFrog Artifactory**.

Example Maven command to fetch the `.war` artifact:

```
 mvn dependency:get -Dartifact=com.web:java-web-calculator:war:0.0.1
-DrepoUrl=https://your-artifactory-domain/artifactory/maven-repository/
```

9.
10. **Deploy the Application**:
    - Copy the `.war` file into the `webapps/` directory of the **Tomcat** server.

```
cp java-web-calculator-0.0.1.war /var/lib/tomcat9/webapps/
```

11.

12. **Restart Tomcat**:
    - Restart **Tomcat** to deploy the `.war` file.

sudo systemctl restart tomcat9

13.
14. **Verify the Deployment**:
    - Ensure that the Java application is accessible via **Nginx** at `http://<deploy-server-ip>`.
    - Nginx should forward requests to Tomcat, where the Java application is running.

Deliverables:

- **Tomcat** successfully configured and running the Java application.
- **Nginx** successfully configured as a reverse proxy.
- Java application accessible in the browser.

---

4. End-to-End Testing

Task Overview:

Test the entire setup from building the application to deploying it on the Deploy Server.

Steps:

1. **Trigger the Build**:
    - On the **Build Server**, trigger the build and deployment process to push the `.war` file to **JFrog Artifactory**.

2. **Verify Artifact in Artifactory**:
    - Log into **Artifactory** and verify that the `.war` artifact has been successfully uploaded.
3. **Deploy the Artifact**:
    - On the **Deploy Server**, fetch the artifact from **Artifactory** and deploy it to **Tomcat**.
4. **Test the Application**:
    - Open a browser and access the Java application at `http://<deploy-server-ip>`. The application should be served through **Nginx** and running in **Tomcat**.

Deliverables:

- The Java application is successfully deployed and accessible.
- Nginx successfully routes traffic to the application running in Tomcat.

Optional Enhancements:

1. Automating with CI/CD:

- Set up a **CI/CD pipeline** using Jenkins or GitLab CI to automatically build, test, and deploy the Java application.

2. Dockerizing the Java Application:

- Package the Java application into a **Docker container** and deploy it with Docker instead of directly using Tomcat.

3. SSL Configuration with Nginx:

- Set up **SSL** using **Let's Encrypt** on **Nginx** to serve the application over HTTPS.
