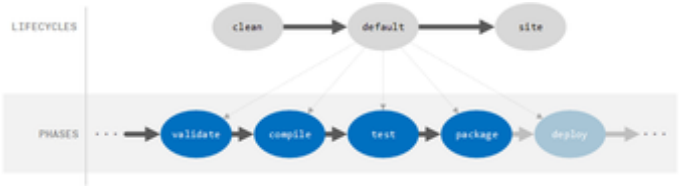
Build & Deployment

The purpose of this article is to guide users on building and deploying the Trip Management application.

Build & Deployment Instructions

Typical Build Process Overview:

1. **Validate configuration**: The configuration is checked by validating the `.platform` directory and scanning the repository for any app configurations to validate individually.
2. **Pull container images**: Any container images that have been built before and that don't have any changes are pulled to be reused.
3. **Install dependencies**: Additional global dependencies are installed in this step.
4. **Run build hook**: The `build` hook comprises one or more shell commands that are written to finish creating the Production environment's code base. It could be compiling Sass files, running a bundler, rearranging files on disk, or compiling. The committed build hook runs in the build container.
5. **Freeze app container**: The file system is frozen and produces a read-only container image, which is the final build artifact.



Steps to Build the application:

Before you build the application, please ensure that Maven is installed in your system. For more information on Maven installation, please access the page: Project Dependencies & Tools

1. Clean the target directory into which Maven builds the project.

```
mvn clean
```

2. Validate that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded.

```
mvn validate
```

3. Compile the source code of the project.

```
mvn compile
```

4. Execute tests against the compiled source code using JUnit and Mockito.

```
mvn test
```

5. Package the code in a distributable format.

```
mvn package
```

6. Install the package into a local repository.

```
mvn install
```

7. Copy the final package to the remote repository.

```
mvn deploy
```

Typical Deploy Process Overview:

1. **Hold requests**: Incoming requests are held to prevent service interruption
2. **Unmount current containers**: Any previous containers are disconnected from their file system mounts
3. **Mount file systems**: The file system is connected to the new containers. New branches have file systems cloned from their parent
4. **Expose services**: Networking connections are opened between any containers specified in your app and services configurations
5. **Run start commands**: The commands necessary to start your app are run
6. **Run deploy hook**: The `deploy` hook is any number of shell commands you can run to finish your deployment. This can include clearing caches, running database migrations, and setting configuration that requires relationship information
7. **Serve requests**: Incoming requests to your newly deployed application are allowed

Setup & Configure Heroku:

1. Navigate to https://www.heroku.com/ and create an account to set up a server
2. Generate an API key and store it in GitLab's CI/CD variables. `Note: The API Key is labelled as "HEROKU_API_KEY" in our project.`
3. Create a new Application. `Note: The application name for our project is tripmanagement.`

```
heroku apps:create [app_name]
```

To display the most recent CI runs for a pipeline, execute:

`heroku ci` --app [app_name]

Steps to Deploy the application:

1. Define a docker image for the latest version of ruby

```
image: ruby:latest
```

2. Download package information from all configured sources and check if ssh-agent is already installed, if not, Install SSH to communicate with the host SSH-agent

```
'command -v ssh-agent >/dev/null || ( apt-get update -y && apt-get
install openssh-client -y )'
```

3. Start the SSH agent

```
eval $(ssh-agent -s)
```

4. Add the private key stored in the variable -"DEPLOY_SSH_KEY" to the SSH registry

```
echo "$DEPLOY_SSH_KEY" | tr -d '\r' | ssh-add -
```

5. Download package information and do not display any output on the console except for errors

```
apt-get update -qy
```

6. Install the Ruby Dev Kit

```
apt-get install -y ruby-dev
```

7. Install dpl - a continuous deployment tool, to deploy artifacts on Heroku

```
gem install dpl
```

8. Deploy the application's source code on Heroku by specifying the app's name and the API Key.

```
dpl --provider=heroku --app=$HEROKU_APP_NAME --api-
key=$HEROKU_API_KEY
```

📋 Related articles

Articles relevant to Continuous Integration & Development or Usage Scenarios relevant to the application, can be accessed through the following pages:

- Usage Scenario
- Continuous Integration & Development (CI/CD)
- Build & Deployment