# Contributing to Jib

We'd love to accept your patches and contributions to this project. There are

just a few small guidelines you need to follow.

## Contributor License Agreement

Contributions to this project must be accompanied by a Contributor License

Agreement. You (or your employer) retain the copyright to your contribution;

this simply gives us permission to use and redistribute your contributions as

part of the project. Head over to <https://cla.developers.google.com/> to see

your current agreements on file or to sign a new one.

You generally only need to submit a CLA once, so if you've already submitted one

(even if it was for a different project), you probably don't need to do it

again.

## Building Jib

Jib comes as 3 public components:

- `jib-core`: a library for building containers

- `jib-maven-plugin`: a Maven plugin that uses `jib-core` and `jib-plugins-common`

- `jib-gradle-plugin`: a Gradle plugin that uses `jib-core` and `jib-plugins-common`

And 1 internal component:

- `jib-plugins-common`: a library with helpers for maven/gradle plugins

The project is configured as a single gradle build. Run `./gradlew build` to build the

whole project. Run `./gradlew install` to install all public components into the

local maven repository.

## Code Reviews

1. Set your git user.email property to the address used for step 1. E.g.

```

git config --global user.email "janedoe@google.com"

```

If you're a Googler or other corporate contributor,

use your corporate email address here, not your personal address.

2. Fork the repository into your own Github account.

3. We follow our own [Java style guide](STYLE\_GUIDE.md) that extends the [Google Java Style Guide](https://google.github.io/styleguide/javaguide.html).

3. Please include unit tests (and integration tests if applicable) for all new code.

4. Make sure all existing tests pass (but see the note below about integration tests).

\* run `./gradlew clean goJF build integrationTest`

5. Associate the change with an existing issue or file a [new issue](../../issues).

6. Create a pull request!

### Integration Tests

\*\*Note\*\* that in order to run integration tests, you will need to set one of the

following environment variables:

- If you are using a GCP project then set `JIB\_INTEGRATION\_TESTING\_PROJECT` to the GCP project to use for testing;

the registry tested will be `gcr.io/<JIB\_INTEGRATION\_TESTING\_PROJECT>`.

- Configure authentication to Container Registry by following these [steps](https://cloud.google.com/container-registry/docs/advanced-authentication).

- Enable the Google Container Registry API [here](https://console.cloud.google.com/apis/library/containerregistry.googleapis.com).

- If you're not using a GCP project then set `JIB\_INTEGRATION\_TESTING\_LOCATION` to a specific registry for testing. (For example, you can run `docker run -d -p 9990:5000 registry:2` to set up a local registry and set the variable to `localhost:9990`.)

You will also need Docker installed with the daemon running. Note that the

integration tests will create local registries on ports 5000 and 6000.

# Development Tips

## Configuring Eclipse

Although jib is a mix of Gradle and Maven projects, we build everything using one

unifed gradle build. There is special code to include some projects directly as

source, but importing your project should be pretty straight forward.

1. Ensure you have installed the Gradle tooling for Eclipse, called

\_Buildship\_ (available from [the Eclipse

Marketplace](https://marketplace.eclipse.org/content/buildship-gradle-integration)).

1. \*\*Import the Gradle project:\*\* Buildship does [not yet support

Eclipse Smart Import](https://github.com/eclipse/buildship/issues/356).

Use \_File &rarr; Import &rarr; Gradle &rarr; Existing Gradle Project\_

and import `jib`.

Note that you will likely need to re-apply these changes whenever

you refresh or update these projects.

## Debugging the Jib Maven Plugin (`jib-maven-plugin`)

### Build and use a local snapshot

To use a local build of the `jib-maven-plugin`:

1. Build and install `jib-maven-plugin` into your local `~/.m2/repository`

with `./gradlew jib-maven-plugin:install`;

1. Modify your test project's `pom.xml` to reference the `-SNAPSHOT`

version of the `com.google.cloud.tools.jib` plugin.

If developing from within Eclipse with M2Eclipse (the Maven tooling for Eclipse):

1. Modify your test project's `pom.xml` to reference the `-SNAPSHOT`

version of the `com.google.cloud.tools.jib` plugin.

1. Create and launch a \_Maven Build\_ launch configuration for the

test project, and ensure the \_Resolve Workspace artifacts\_ is checked.

### Attaching a debugger

Run `mvnDebug jib:build` and attach to port 8000.

If developing with Eclipse and M2Eclipse (the Maven tooling for Eclipse), just launch the \_Maven Build\_ with \_Debug\_.

## Debugging the Jib Gradle Plugin (`jib-gradle-plugin`)

### Build and use a local snapshot

To use a local build of the `jib-gradle-plugin`:

1. Build and install `jib-gradle-plugin` into your local `~/.m2/repository`

with `./gradlew jib-gradle-plugin:install`

1. Add a `pluginManagement` block to your test project's `settings.gradle` to enable reading plugins from the local maven repository. It must be the first block in the file before any `include` directives.

```groovy

pluginManagement {

repositories {

mavenLocal()

gradlePluginPortal()

}

}

```

1. Modify your test project's `build.gradle` to use the snapshot version

```groovy

plugins {

// id 'com.google.cloud.tools.jib' version '2.4.0'

id 'com.google.cloud.tools.jib' version '2.4.1-SNAPSHOT'

}

```

### Attaching a debugger

Attach a debugger to a Gradle instance by running Gradle as follows:

```shell

./gradlew jib \

--no-daemon \

-Dorg.gradle.jvmargs='-agentlib:jdwp:transport=dt\_socket,server=y,address=5005,suspend=y'

```