

Automating your Github library releases to Maven Central

by Benny Bottema - Wednesday, February 20, 2019

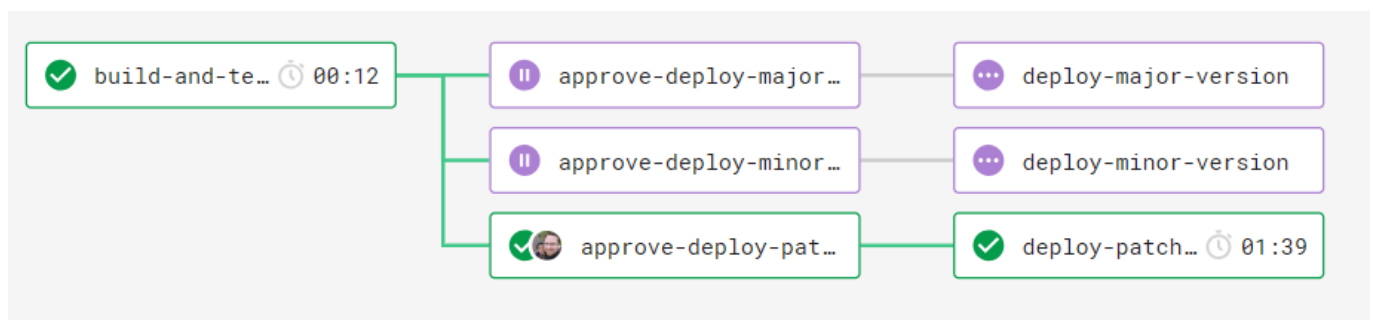
<http://www.bennybottema.com/2019/02/20/automating-your-github-library-releases-to-maven-central/>

After a long time performing manual releases on my own laptop, I decided to jump on the CI/CD bandwagon and automate everything.

I've been working professionally with Jenkins and Bamboo for many years, but I never took the time to properly set it up for my open source project, [Simple Java Mail](#). I finally decided I would combine learning a new CI/CD tool with setting up auto-releases to Maven Central.

Now Simple Java Mail is a multi-modular Maven project, so that makes things a little more complicated, so for this blog I've created a test project that you can fork and study:

- <https://github.com/bbottema/auto-deploy-test> (tag: "circleci-deploy-final-without-orb")



Contents

- [1 Plan of Attack](#)
 - [1.1 Introducing CircleCI](#)
- [2 Checkout the source code from Github.com](#)
- [3 Compile, test the project](#)
- [4 Manually select patch, minor or major version release](#)
- [5 Auto-update POM with semver based on manual selection](#)
- [6 Build the deployable artifacts \(jar, source jar, javadoc jar\)](#)
- [7 Sign the artifacts with GPG so OSS Sonatype will accept them](#)
 - [7.1 Introducing OSS Sonatype](#)
 - [7.2 From CircleCI to OSS Sonatype](#)
- [8 Commit the updated POM and tag the commit with the new version](#)
- [9 Push changes back to repo](#)
 - [9.1 Adding Github.com as a trusted host](#)
 - [9.2 Configuring GIT to use our SSH key and user](#)
 - [9.3 Finally, performing the push to repo](#)
- [10 The final deploy scripts](#)

Plan of Attack

Here's what we want to achieve!

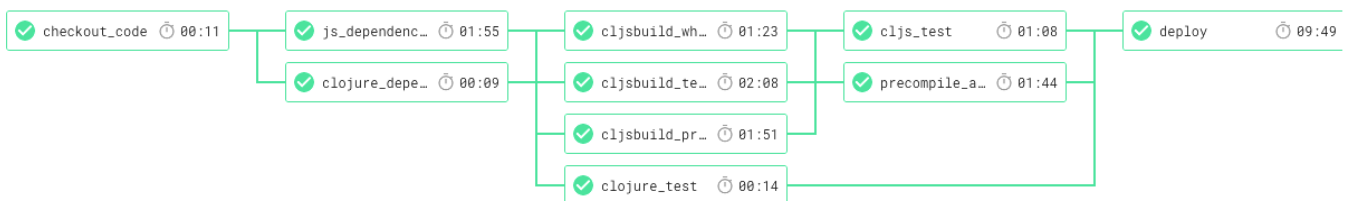
1. Checkout the source code from the Github repository
2. Compile, test the project
3. Manually choose if the release entails a patch, minor or major version
4. Automatically update the POM with the new release [semver](#) based on the previous choice
5. Build the deployable artifacts (jar, source jar, Javadoc jar)
6. Sign the artifacts with GPG so OSS Sonatype will accept them
7. Deploy the artifacts to staging area, automatically closing and releasing to Maven Central upon successful upload
8. Commit the updated POM and tag the commit with the new version
9. Push changes back to the Github repository

Introducing CircleCI

My weapon of choice is [CircleCI](#) because of its intuitive design to build scripts, standard integration with Github, its potential for speeding up complex builds and standard Docker integration.

Frankly speaking, I was so glad I finally got everything working perfectly that I first wanted to write everything down before attempting the same setup with Azure Devops and Gitlab.

CircleCI (2.1) works with something called "[workflows](#)", which is basically a pipeline of several build jobs, which if defined smartly, can run parallel. Moreover, one job type is a "manual approval" job, which can be used to force a specific path in a workflow. I use this technique to manually select an automated patch, minor or major release.



Checkout the source code from Github.com

CircleCI seamlessly integrates with public Github repo's, so it can import Simple Java Mail automatically. CircleCI manages its own SSH key registration with the repository (with your confirmation) for read access and can [checkout](#) the code during the build.

Compile, test the project

To compile and test we need a docker image with Maven and specifically for Simple Java Mail: JDK 8. [circleci/openjdk:8u171-jdk](#) will do the trick nicely (complete list [here](#)).

Let's define our initial flow with our selected container, run tests and collect our artifacts:

```
version: 2.1

executors:
  maven-executor:
    docker:
      - image: circleci/openjdk:8u171-jdk

jobs:

  build-and-test:
    executor: maven-executor

    steps:
      - checkout
      - restore_cache:
          key: auto-deploy-test-{{ checksum ".circleci/config.yml" }}
      - run:
          command: mvn verify -DexcludeLiveServerTests=true -Dmaven.javadoc.skip=true
      - persist_to_workspace:
          root: .
          paths:
            - .
      - run:
          shell: /bin/bash -eo pipefail -O globstar
          command: |
            mkdir -p artifacts/junit
            cp **/target/*.jar artifacts/
            cp -a **/target/surefire-reports/. artifacts/junit
      - store_artifacts:
          path: artifacts
      - store_test_results:
          path: artifacts/junit

workflows:
  workflow:
    jobs:
      - build-and-test
```

Since we have a separate build job for producing the deployable artifacts (because we don't know the release version yet), we can skip some things here to speed up this job, such as producing javadoc.

Manually select patch, minor or major version release

In Jenkins or Bamboo I would configure target environments to pick up the “shared artifacts” and trigger the right version bump manually, but CircleCI works a bit differently with its “workflow” approach.

Instead of deployment pipelines, CircleCI has a special type of build job that will pause for manual confirmation. The subsequent build jobs will wait until it is approved. This way you can implement multiple deployment pipelines within one workflow. The way I'm using it though, I haven't seen that on the web yet.

Here's what the update CircleCI config looks like:

```
version: 2.1

executors:
  maven-executor:
    docker:
      - image: circleci/openjdk:8u171-jdk

jobs:

  build-and-test:
    executor: maven-executor
    (..)

  deploy-patch-version:
    executor: maven-executor
    steps:
      - run:
          command: # deploy patch version

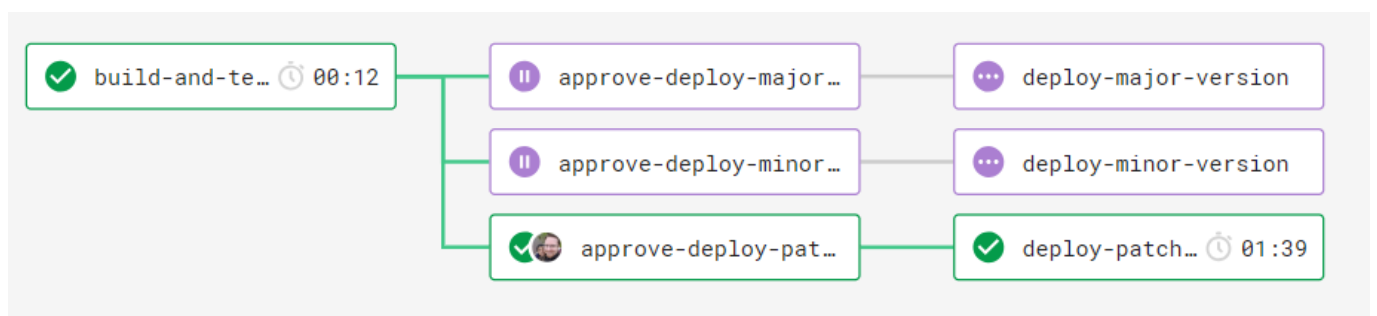
  deploy-minor-version:
    executor: maven-executor
    steps:
      - run:
          command: # deploy patch version

  deploy-major-version:
    executor: maven-executor
    steps:
      - run:
```

```
command: # deploy patch version

workflows:
  workflow:
    jobs:
      - build-and-test
      - approve-deploy-patch-version:
          type: approval
          requires:
            - build-and-test
      - approve-deploy-minor-version:
          type: approval
          requires:
            - build-and-test
      - approve-deploy-major-version:
          type: approval
          requires:
            - build-and-test
      - deploy-patch-version:
          requires:
            - approve-deploy-patch-version
      - deploy-minor-version:
          requires:
            - approve-deploy-minor-version
      - deploy-major-version:
          requires:
            - approve-deploy-major-version
```

The result looks like this in CircleCI:



Auto-update POM with semver based on manual selection

Ok, so now that we know based on the workflow execution path what version bump we want to perform, how can we actually do the version bump?

There is a little bit of an obscure Maven feature that was undocumented for a long time: `versions:set` combined with `build-helper:parse-version`. For example, to bump the minor version (ie. 2.3.4 becomes 2.4.4), you can do the following:

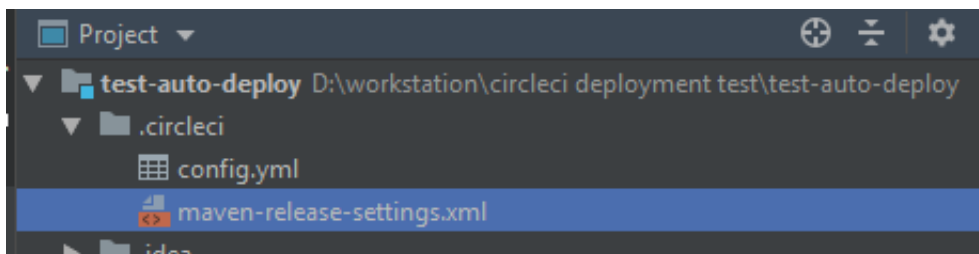
```
mvn build-helper:parse-version versions:set -DnewVersion=\${parsedVersion.majorVersion}.\${parsedVersion.nextMinorVersion}.\${parsedVersion.incrementalVersion} versions:commit
```

How it works

What happens is that `versions:set` performs the actual update to the POM and will look for a property `newVersion`. We use `build-helper:parse-version` to produce that variable using properties available only to the build-helper. We need to escape the `$`-signs, because otherwise Bash will try to resolve them before they reach Maven. Finally `versions:commit` just gets rid of the POM backups from before the version bump.

Build the deployable artifacts (jar, source jar, javadoc jar)

Build your artifacts as you normally would, but use a custom maven settings.xml for your build. We'll need it to configure GPG and OSS Sonatype login credentials in the next step.



We'll use it in our deploy in the next step like so:

```
mvn -s .circleci/maven-release-settings.xml clean deploy ...other options...
```

Since we have a separate build job for compiling and testing the code, we can skip things like testing, instrumentation, spotbugs/pmd etc. by providing the options `-DskipTests` and `-Dspotbugs.skip=true`.

Sign the artifacts with GPG so OSS Sonatype will accept them

Now it gets interesting, because you'll have to configure some keys and secrets as environmental string variables so you can refer to it from your build script.

Here's our checklist:

1. produce a GPG key pair with passphrase
2. distribute the public key to one of the public servers OSS Sonatype validates signed artifacts with
3. make the private available in CircleCI as environment variable
4. Include the passphrase as environment variable so you can use the private key for signing the deployable artifacts

Introducing OSS Sonatype

Sonatype is an artifact server that synchronizes to Maven Central if you release a non-SNAPSHOT deploy. It has some rules for artifacts it can accept such as source, javadoc and binary jars should all be present and signed with GPG.

To continue, please first [register your OSS project with OSS Sonatype](#) if you haven't yet and then complete the steps outlined in [Sonatype's guide to GPG keys](#), including uploading it to one of the public key servers.

From CircleCI to OSS Sonatype

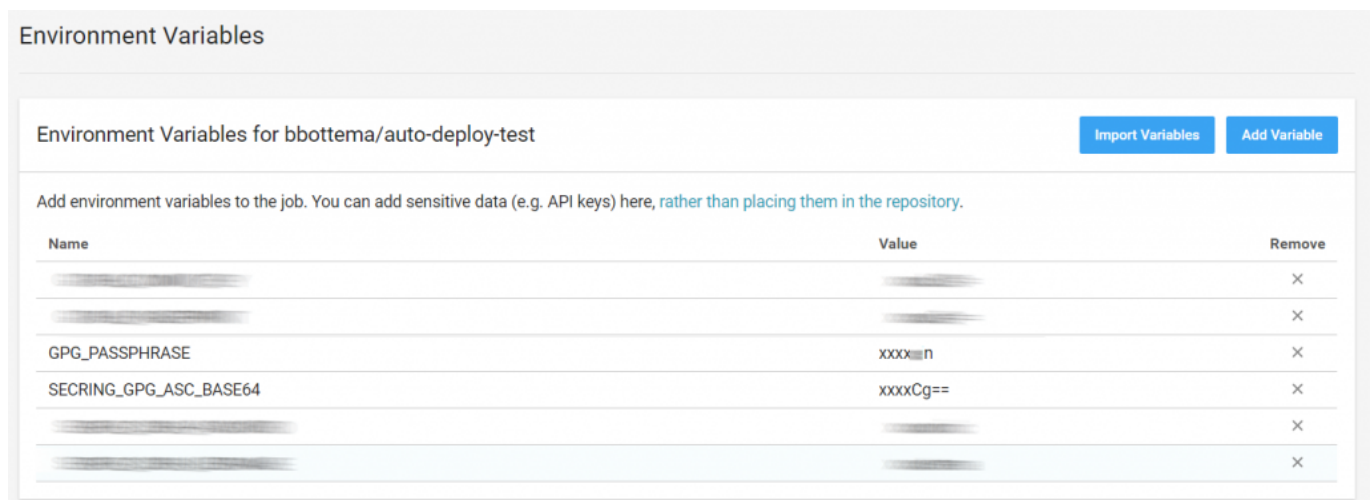
Now that we have an OSS Sonatype project and distributed a public GPG key, we can start signing and releasing artifacts to Maven Central.

Adding the private GPG key to CircleCI

Take your private key in ASCII, which should be something like `secring.gpg.asc`. If you only have a `.gpg` file, you need to [convert it to ASCII](#) first. This is dangerous, so throw it away after you're done adding it to CircleCI:

```
gpg --no-default-keyring --armor --secret-keyring ./secring.gpg --keyring ./pubring.gpg --export-secret-key your@email.com > secring.gpg.asc
```

To get your ASCII key on a single line, you can [use sed](#) in linux with some black magic regex, or much simpler: paste it in an [base64 converter](#) and convert it to a base64 string. Import this string as environment variable and also add you GPG passphrase:



Environment Variables

Environment Variables for bbottema/auto-deploy-test Import Variables Add Variable

Add environment variables to the job. You can add sensitive data (e.g. API keys) here, rather than placing them in the repository.

Name	Value	Remove
[REDACTED]	[REDACTED]	X
[REDACTED]	[REDACTED]	X
GPG_PASSPHRASE	xxxxn	X
SECRING_GPG_ASC_BASE64	xxxxCg==	X
[REDACTED]	[REDACTED]	X
[REDACTED]	[REDACTED]	X

you can use these in your CircleCI build script

Configure Maven to connect to OSS Sonatype

We'll define a Maven profile for GPG signing that is deactivated by default, so that we don't have to deal with that when testing things locally on our own laptops. What's more, OSS Sonatype requires you to define a couple of things before it accepts your artifacts, such as a developer tag:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>ossrh</id>
      <username>${env.SERVER_OSSRH_USERNAME}</username>
      <password>${env.SERVER_OSSRH_PASSWORD}</password>
    </server>
  </servers>

  <profiles>
    <profile>
      <id>gpg</id>
      <properties>
        <gpg.executable>gpg</gpg.executable>
        <gpg.passphrase>${env.GPG_PASSPHRASE}</gpg.passphrase>
      </properties>
    </profile>
  </profiles>
  <activeProfiles>
    <activeProfile>gpg</activeProfile>
  </activeProfiles>
</settings>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.simplejavamail</groupId>
  <artifactId>test-auto-deploy</artifactId>
  <version>4.3.9</version>

  <name>test-auto-deploy</name>
  <description>Testing out auto-deployments with CircleCI</description>
  <url>http://http://www.simplejavamail.org/</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <licenses>
    <license>
      <name>The Apache Software License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0.txt</url>
      <distribution>repo</distribution>
    </license>
  </licenses>

  <scm>
    <connection>scm:git:git://github.com/bbottema/auto-deploy-test.git</connection>
    <developerConnection>scm:git:git@github.com:bbottema/auto-deploy-test.git</developerConnection>
    <url>https://github.com/bbottema/auto-deploy-test</url>
  </scm>

  <developers>
    <developer>
      <id>benny</id>
      <name>Benny Bottema</name>
      <email>benny@bennybottema.com</email>
      <url>http://www.bennybottema.com</url>
      <roles>
```

```
    <role>developer</role>
    <role>packager</role>
  </roles>
</developer>
</developers>

<distributionManagement>
  <snapshotRepository>
    <id>ossrh</id>
    <url>https://oss.sonatype.org/content/repositories/snapshots</url>
  </snapshotRepository>
  <repository>
    <id>ossrh</id>
    <url>https://oss.sonatype.org/service/local/staging/deploy/maven2/<
/ur
l>
  </repository>
</distributionManagement>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-release-plugin</artifactId>
      <version>2.5.2</version>
    </plugin>
    <plugin>
      <groupId>org.sonatype.plugins</groupId>
      <artifactId>nexus-staging-maven-plugin</artifactId>
      <version>1.6.3</version>
      <extensions>>true</extensions>
      <configuration>
        <serverId>ossrh</serverId>
        <nexusUrl>https://oss.sonatype.org/</nexusUrl>
        <autoReleaseAfterClose>>true</autoReleaseAfterClose>
      </configuration>
    </plugin>
  </plugins>
</build>

<profiles>
  <profile>
    <id>release-sign-artifacts</id>
    <activation>
      <property>
        <name>performRelease</name>
        <value>>true</value>
      </property>
    </activation>
  </profile>
</profiles>
```

```
</activation>
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-gpg-plugin</artifactId>
      <version>1.5</version>
      <executions>
        <execution>
          <id>sign-artifacts</id>
          <phase>verify</phase>
          <goals>
            <goal>sign</goal>
          </goals>
          <!-- fixes tty error under linux -->
          <configuration>
            <gpgArguments>
              <arg>--pinentry-mode</arg>
              <arg>loopback</arg>
            </gpgArguments>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
</profile>
</profiles>
</project>
```

For the maven-release-settings.xml to work you need to add your OSS Sonatype credentials to CircleCI as well:

Environment Variables

Environment Variables for bbottema/auto-deploy-test Import Variables Add Variable

Add environment variables to the job. You can add sensitive data (e.g. API keys) here, rather than placing them in the repository.

Name	Value	Remove
████████████████████	████████████████████	×
████████████████████	████████████████████	×
████████████████████	████████████████████	×
████████████████████	████████████████████	×
SERVER_OSSRH_PASSWORD	xxxx███C	×
SERVER_OSSRH_USERNAME	xxxx███a	×

Now that we configured our deployment plugins to sign artifacts and connect to OSS Sonatype, deploy the signed artifacts to staging area, automatically closing and releasing to Maven Central upon successful upload (or else you still need to manually [login into OSS Sonatype](#) to release it):

First define a command we can call from our deploy job that will configure GPG by importing our base64 ASCII key into the GPG tool already included in the Docker image:

```
commands:
  configure-gpg:
    steps:
      - run:
          name: Configure GPG private key for signing project artifacts in OSS Sonatype
          command: |
            echo $SECRING_GPG_ASC_BASE64 | base64 --decode | gpg --batch --no-tty --import --yes
```

Then implement the deploy jobs for the three semver deploy paths:

```
version: 2.1

jobs:
  (...)

  deploy-patch-version:
    executor: maven-executor
    steps:
      - deploy:
```

```
    versioncommand: mvn build-helper:parse-version versions:set
-DnewVersion=\${parsedVersion.majorVersion}.\${parsedVersion.minorVersion}.\${parsedVersion.nextIncrementalVersion} versions:commit
```

```
deploy-minor-version:
```

```
  executor: maven-executor
```

```
  steps:
```

```
    - deploy:
```

```
      versioncommand: mvn build-helper:parse-version versions:set
-DnewVersion=\${parsedVersion.majorVersion}.\${parsedVersion.nextMinorVersion}.\${parsedVersion.incrementalVersion} versions:commit
```

```
deploy-major-version:
```

```
  executor: maven-executor
```

```
  steps:
```

```
    - deploy:
```

```
      versioncommand: mvn build-helper:parse-version versions:set
-DnewVersion=\${parsedVersion.nextMajorVersion}.\${parsedVersion.minorVersion}.\${parsedVersion.incrementalVersion} versions:commit
```

```
commands:
```

```
  (..)
```

```
deploy:
```

```
  parameters:
```

```
    versioncommand:
```

```
      type: string
```

```
  steps:
```

```
    - attach_workspace:
```

```
      at: .
```

```
    - restore_cache:
```

```
      key: auto-deploy-
```

```
test-{{ checksum ".circleci/config.yml" }}
```

```
  - configure-gpg
```

```
  - run:
```

```
    name: Release new version to Maven Central
```

```
    command: |
```

```
      echo "Starting new release..."
```

```
      << parameters.versioncommand >>
```

```
      mvn -s .circleci/maven-release-settings.xml clean deploy
```

```
  -DdeployAtEnd=true -DperformRelease=true -DskipTests -Dspotbugs.skip=true
```

```
      echo "Successfully released new version"
```

```
  - save_cache:
```

```
    paths:
```

```
    - ~/.m2
    key: auto-deploy-
test-{{ checksum ".circleci/config.yml" }}
```

If everything was configured correctly, your script should now build, test, sign and deploy to Maven Central via OSS Sonatype.

Commit the updated POM and tag the commit with the new version

In order to provide a commit message with the new Maven version as well as tagging with that version, you need Maven to tell you that version first so you can store it in a variable. This is a little tricky, but can be done with a [Command substitution](#).

```
MVN_VERSION=$(mvn -q -Dexec.executable="echo" -Dexec.args='${project.version}' --non-recursive exec:exec)
git commit -am "released ${MVN_VERSION} [skip ci]"
git tag -a ${MVN_VERSION} -m "Release ${MVN_VERSION}"
```

Notice the text “[skip ci]”? That’s so CircleCI doesn’t trigger another build for this commit. It’s a convention which is also supported by other vendors (for example [TravisCI](#)).

Push changes back to repo

CircleCI setup a read-only SSH key for checking out the repo, but now you need to push something back. This means you need to provide your own SSH key pair that has write access. Moreover, you will need to explicitly acknowledge github.com as a trusted host by providing the server’s fingerprint.

Adding Github.com as a trusted host

Following [this SO](#), here is how you can obtain github.com’s fingerprint as base64 (1st command):

```
ssh-keyscan github.com >> githubKey
ssh-keygen -lf githubKey
```

Manually verify the fingerprint (2nd command) is the same as [the fingerprint Github published](#), and then add the entire content of the file we just created to CircleCI:

Environment Variables

Environment Variables for bbottema/auto-deploy-test Import Variables Add Variable

Add environment variables to the job. You can add sensitive data (e.g. API keys) here, rather than placing them in the repository.

Name	Value	Remove
[REDACTED]	[REDACTED]	×
GITHUB_FINGERPRINT	xxxxaQ==	×
[REDACTED]	[REDACTED]	×
[REDACTED]	[REDACTED]	×
[REDACTED]	[REDACTED]	×
[REDACTED]	[REDACTED]	×

Finally add this fingerprint to trusted hosted in your deploy script:

```
mkdir -p ~/.ssh  
echo $GITHUB_FINGERPRINT >> ~/.ssh/known_hosts
```

Configuring GIT to use our SSH key and user

Generate a new key pair (I did without password) and save it to `.\github_rsa.key` (the command will prompt you for it):

```
ssh-keygen -t rsa -b 4096 -C "b.bottema@projectnibble.org"
```

Now copy paste the content of the public key (`github_rsa.key.pub`) to Github in your repo and make sure to check "Allow write access":

bbottema / auto-deploy-test

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Options
Collaborators
Branches
Webhooks
Notifications
Integrations & services
Deploy keys
Moderation
Interaction limits

Deploy keys / Add new

Title
CircleCI deploy key

Key
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCAQC3qdqXiZUFbpYoa8OseRowpLj/aS4NhRDml3q9ZXcDla7aNNRD9Uhf
ibLibXbxjSU/QFR2VROkIA4ZQYP9VaLcSxtOLOD1CC5ANVhd6IOPSB1daaa7tq4kHdM5xyyoJg/JMPMzT+7ixidtG1
uY1reljSOYWwJ42kNGgdjR4441dbd9jL28pNT11nXH3OJ66SsajWR1qluDPsPeFWNasPjkkYVcUCnMi82HwAkIE/PH
NMBXfGop0YCqI43ooTbePOu+5wFBCVLejxiDZWNWJQ/7By1I6a47Xm1O7K4ofA2n26dgBFmEI0PCKNmnDe3nYKsS
scvtjp3OIeO3Jne8hNpq75PrCyYhEEoPZ0ox3Og/18em1Vlz mugQrEcqzQHfBI7UECA7ScrDwRmOiSdShiWFvXuU5z
zLKsEDVsSLZwo4KXIeh8k7w2P8hqmGeDzhikH8dZTsPyZL78RxsHnwh7aFIdX2xx3ji2HCgaF03brimjHJpxXY5P9GNU
KZLAI4HBnW3o05fuwYBAC165eA4FFcAke+VH2sv9L9eGio4KYPVXJL66zGtdHhYxfrahQdmAbvEc7k4M4cv6iE6cOf
SOHi7rszTE9pkOEHSB/XoyjsB1IMMVQOVmXjPmVvzoXapuXThForLN3jKCQncss7U/mBpRTZ4xC6uy/Q==
b.bottema@projectnibble.org

Allow write access
Can this key be used to push to this repository? Deploy keys always have pull access.

Add key

Take the private key and again convert it to base64 and add it to CircleCI environment variables for your project:

Environment Variables

Environment Variables for bbottema/auto-deploy-test

Import Variables Add Variable

Add environment variables to the job. You can add sensitive data (e.g. API keys) here, rather than placing them in the repository.

Name	Value	Remove
GITHUB_COMMIT_KEY	xxxxLS0=	×
████████████████████	████████████████████	×
████████████████████	████████████████████	×
████████████████████	████████████████████	×
████████████████████	████████████████████	×
████████████████████	████████████████████	×

Now you can refer to it from your CircleCI deploy script. Let's take the fingerprint script and club it together with the SSH key config in a new command to keep things tidy:

commands :
(...)


```
configure-git:
  steps:
    - run:
        name: Configure GIT with host fingerprint, user info and SSH
        key for pushing
        command: |
            mkdir -p ~/.ssh
            echo "Adding github.com as known host..."
            echo $GITHUB_FINGERPRINT >> ~/.ssh/known_hosts
            echo "Setting private SSH key for pushing new version to r
            epo..."
            echo $GITHUB_COMMIT_KEY | base64 --decode >> ~/.ssh/id_rsa
            chmod 400 ~/.ssh/id_rsa # prevents "UNPROTECTED PRIVATE KE
            Y FILE" error
            git config user.name "bbottema"
            git config user.email "b.bottema@projectnibble.org"
```

Finally, performing the push to repo

With the fingerprint and SSH key in place, we can finally perform the last step in our CI/CD script: push the change and tag back to the repo.

To perform GIT commands with an SSH key, you need to write the commands a little differently:

```
ssh-agent sh -c 'ssh-
add ~/.ssh/id_rsa; git push git@github.com:bbottema/auto-deploy-test'
ssh-agent sh -c 'ssh-add ~/.ssh/id_rsa; git push origin --tags'
```

The final deploy scripts

- [.circleci/maven-release-settings.xml](#)
- [.circleci/config.yml](#)
- [pom.xml](#)

To make this work you need github.com's fingerprint as environment variable as well as OSS Sonatype login credentials, GPG signing key and passphrase, and GIT read/write SSH key.