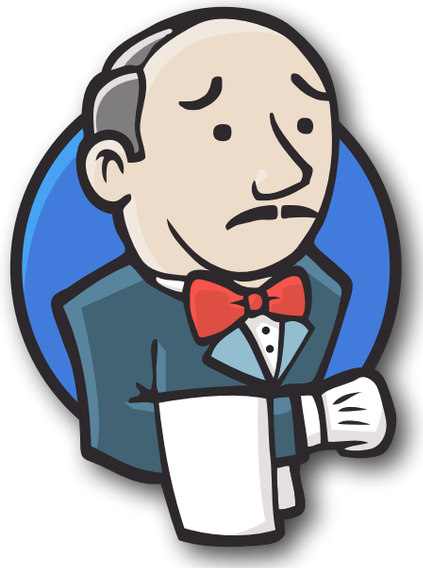
Jenkinsではじめる継続的インテグレーション

JaSST'16 Niigata

・・・

JaSST'14
Tohokuと
同じパターンだ

「予稿でネタバレしたくない」
って原稿できてない
言い訳なんじゃ？

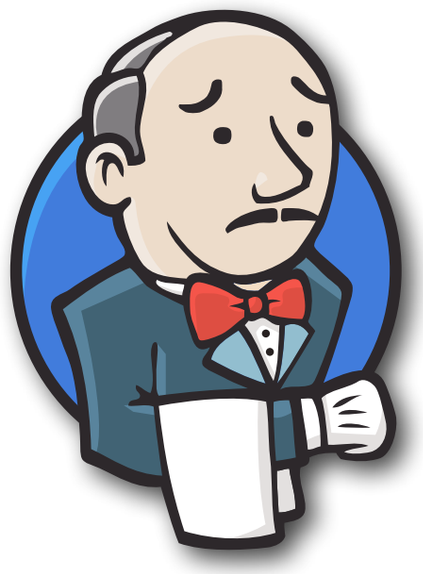


JaSST'16 Niigataでは
はじめてJenkinsを使う人の
ためのチュートリアルを
話すよ

これを聞けば
誰もが
スーパーJenkinsマンに
なれるぞ！

…だといいね

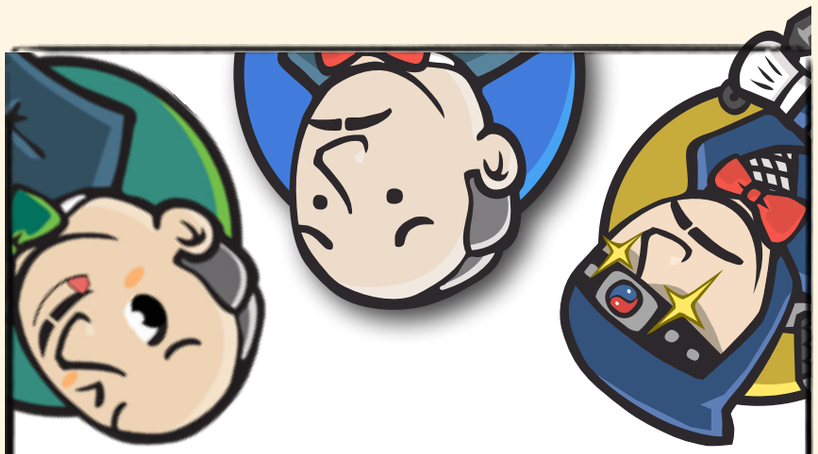
不安になること
言わないでよ



Jenkinsの入門話って
往々にして
ビルドツールの話に
なるよね

開き直ってる

ビルドツールなくして
Jenkinsは語れない！

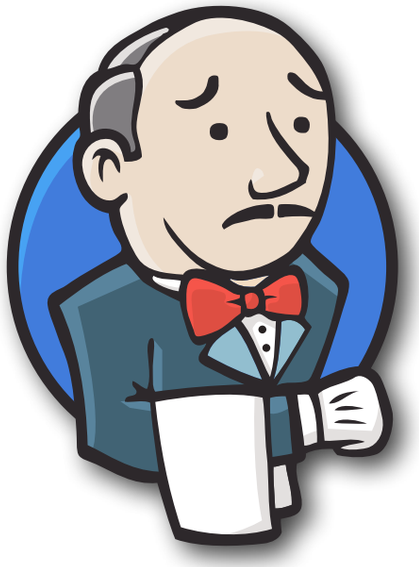


あれ、だれ？

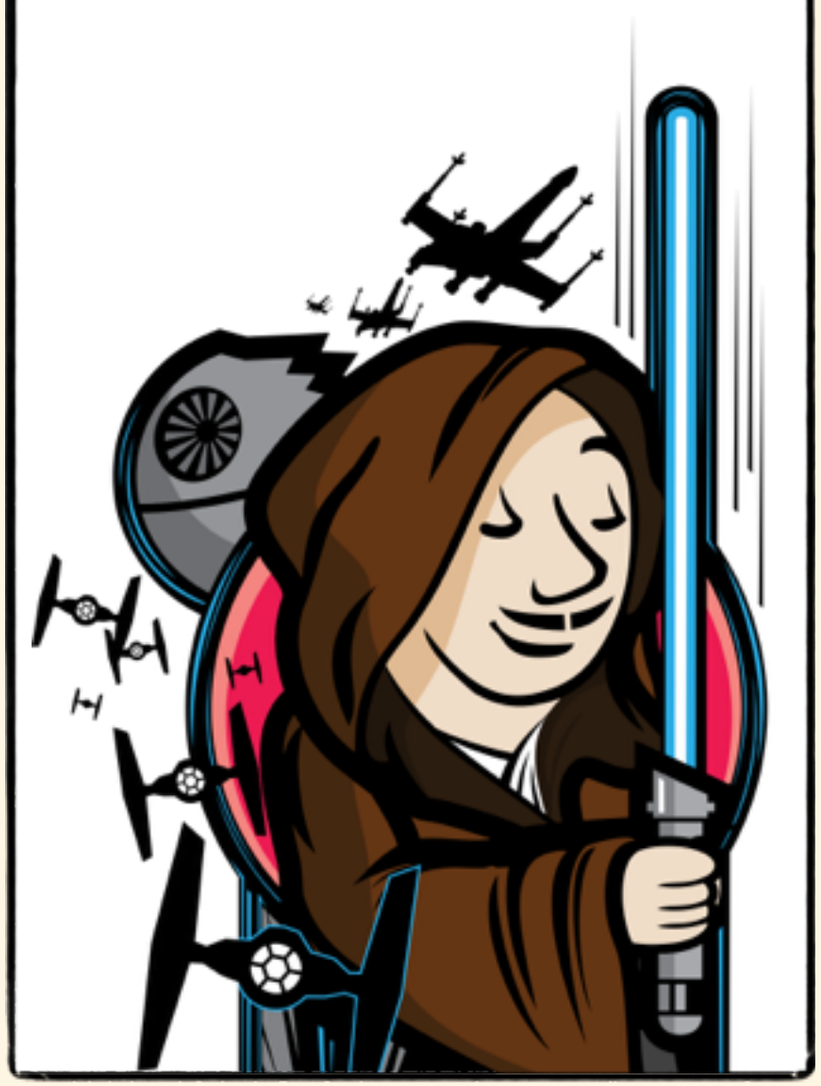
話者の都合上
Javaの話をしまーす

ところで、
最近はやりの六つ子
Jenkinsでもできるんじゃ？

ひとり多かった…



そんなことより、
ビルドを信じるのです



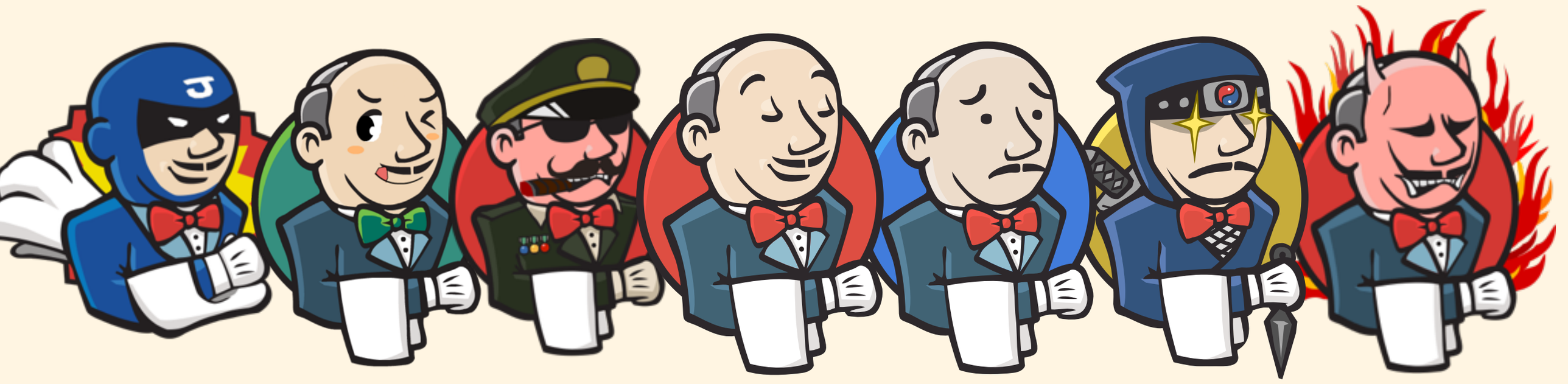
乞うご期待！

Jenkinsではじめる継続的インテグレーション

JaSST'16 Niigata

自己紹介

- 今井勝信
  @masanobuimai
- 日本ユニシス（株）勤務



アジェンダ

- Jenkinsとは
- 継続的インテグレーションとは
- プロジェクトの自動化（ビルドの自動化）
- Jenkinsで行う継続的インテグレーション
- まとめ

おことわり

あまり高度は話は
しませんよ。

Jenkinsとは

オープンソースのCIサーバ
CIサーバの代名詞的存在
http://jenkins-ci.org
@jenkinsci

日本Jenkinsユーザ会

http://build-shokunin.org

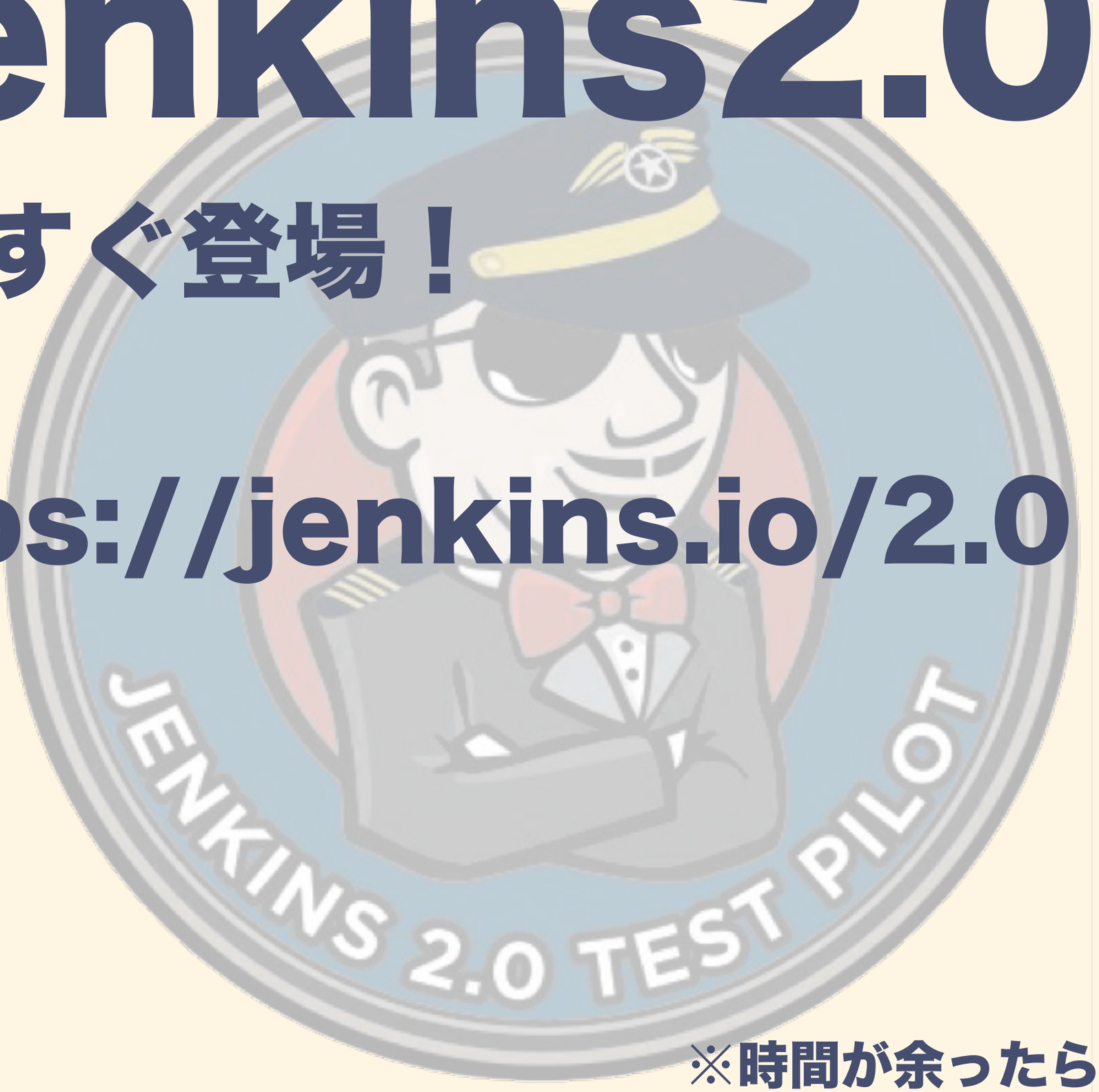
@jenkinsja
#jenkinsja

超高機能

cron

Jenkins2.0

もうすぐ登場！



https://jenkins.io/2.0

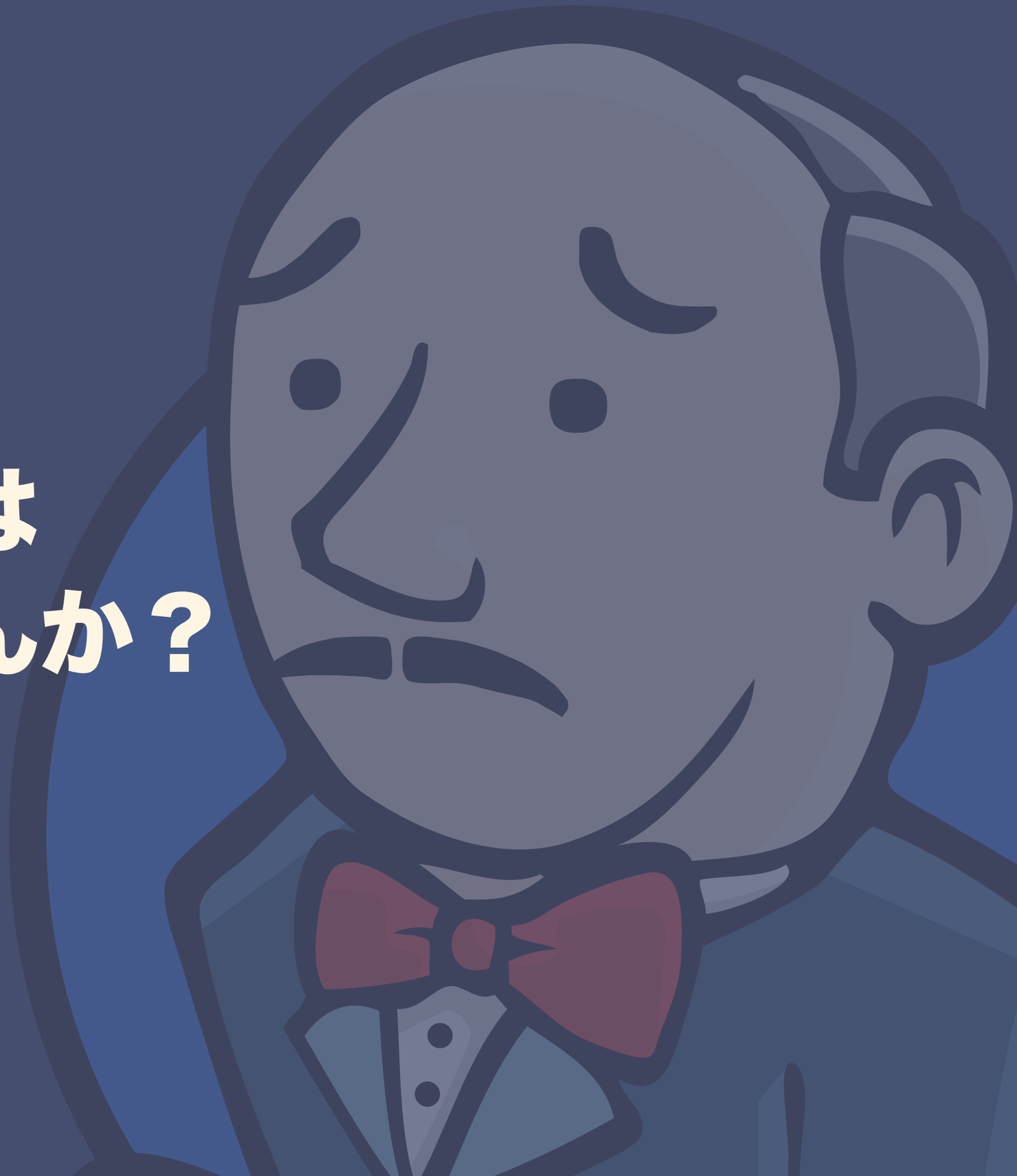
※時間が余ったらデモをします

継続的
インテグレーション
Continuous Integration
とは

『テストも含めて
「すべてが自動化された」
「再現可能な」ビルドを、
「日に何度も」行うこと』

Continuous Integration - Martin Fowler より

心当たりは
ありませんか？



- ビルドが複雑でミスを招きやすい／属人化する
- ビルドできる環境が限定される
- 「まとめてビルド」によるビルド地獄

ならばよろしい。

常にビルドだ

現代ソフトウェア開発の三本柱

1.バージョン管理
2.テスティング
3.自動化

テスト書いてないとかお前それ
@t_wadaの前でも
同じこと言えんの?

ＴＤＤＢＣ(テスト駆動開発Boot Camp)より

継続的インテグレーションの
基本は

自動化

なんでも
自動化しよう

継続的インテグレーションの導入の7つのステップ

1.ビルドサーバがない

2.ナイトリービルド

3.ナイトリービルドと基本的な自動テスト

4.メトリクスの導入

5.テストのさらなる真剣な取り組み

6.自動受け入れテストと、さらなる自動化デプロイメント

7.継続的デプロイメント

『Jenkins』(オライリー)より

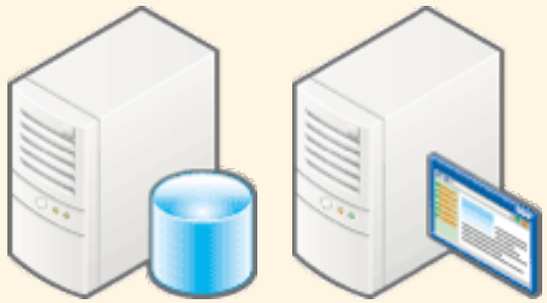
みなさんは、
どのあたりに居ますか？

ステップ0

「ビルドが自動化されていない」

ありふれた開発環境

実行環境



・デプロイ
・テスト

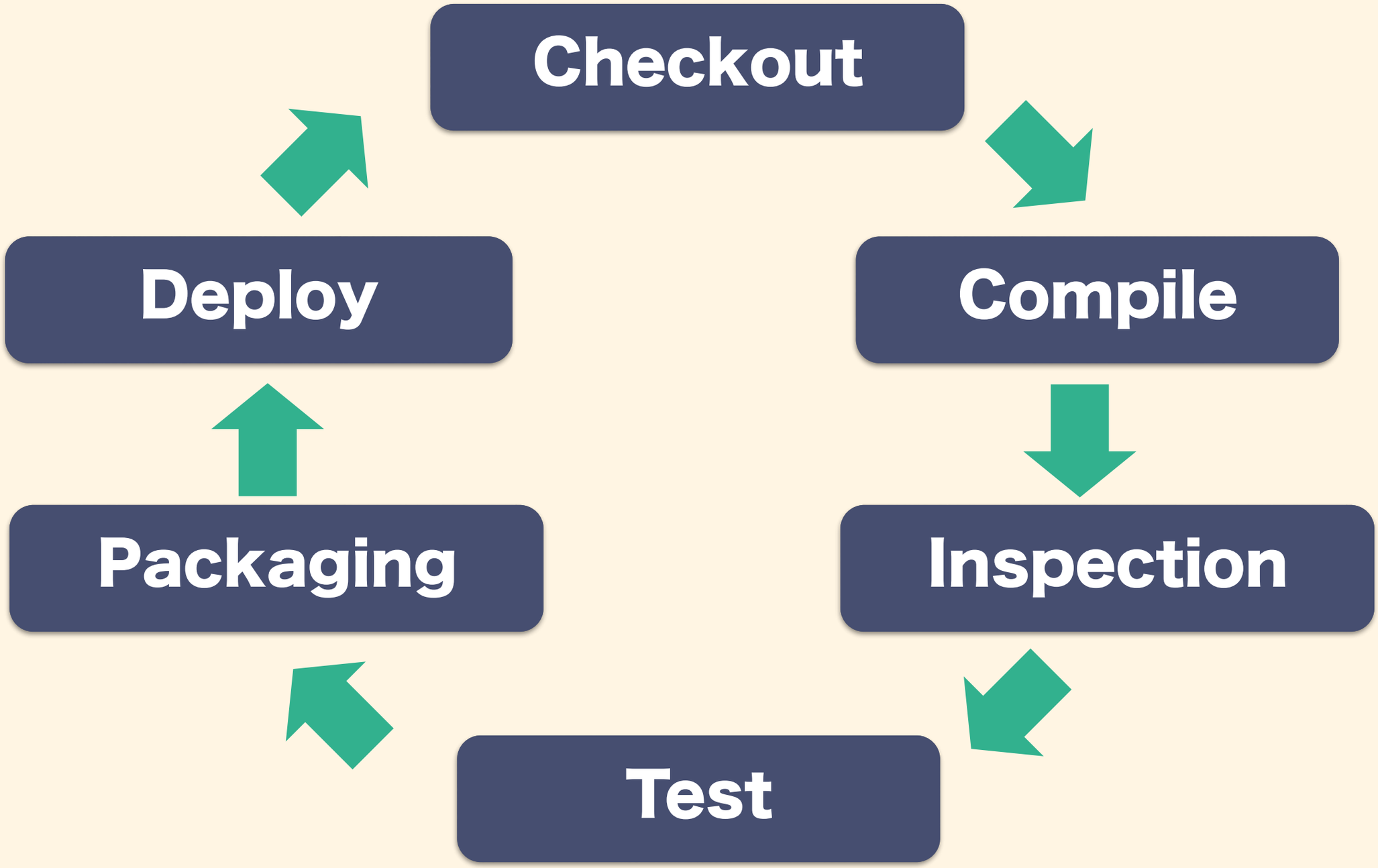
リポジトリサーバ

・チェックアウト
・コミット

・プログラミング
・テスト

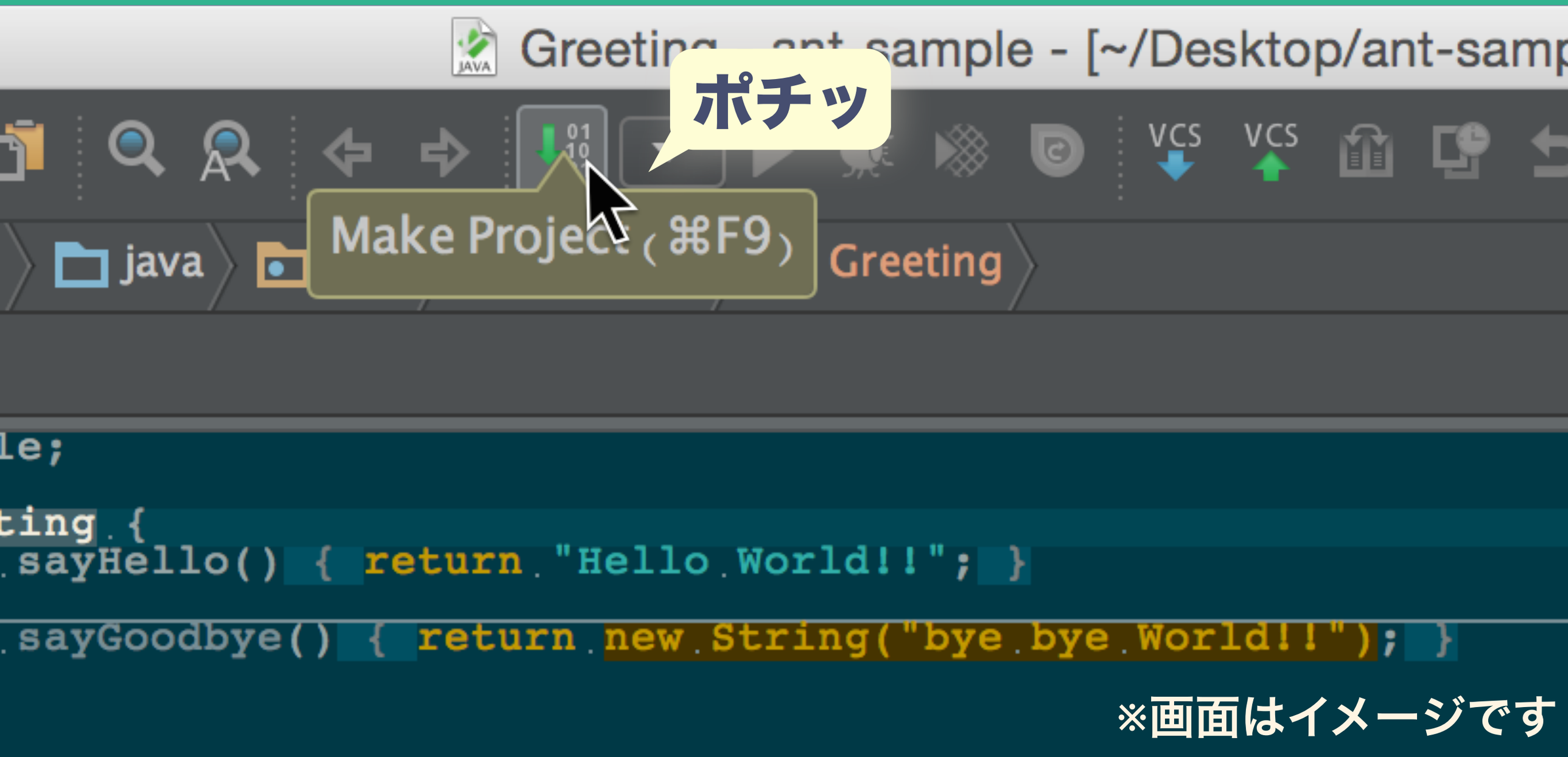
開発用PC

日々の作業



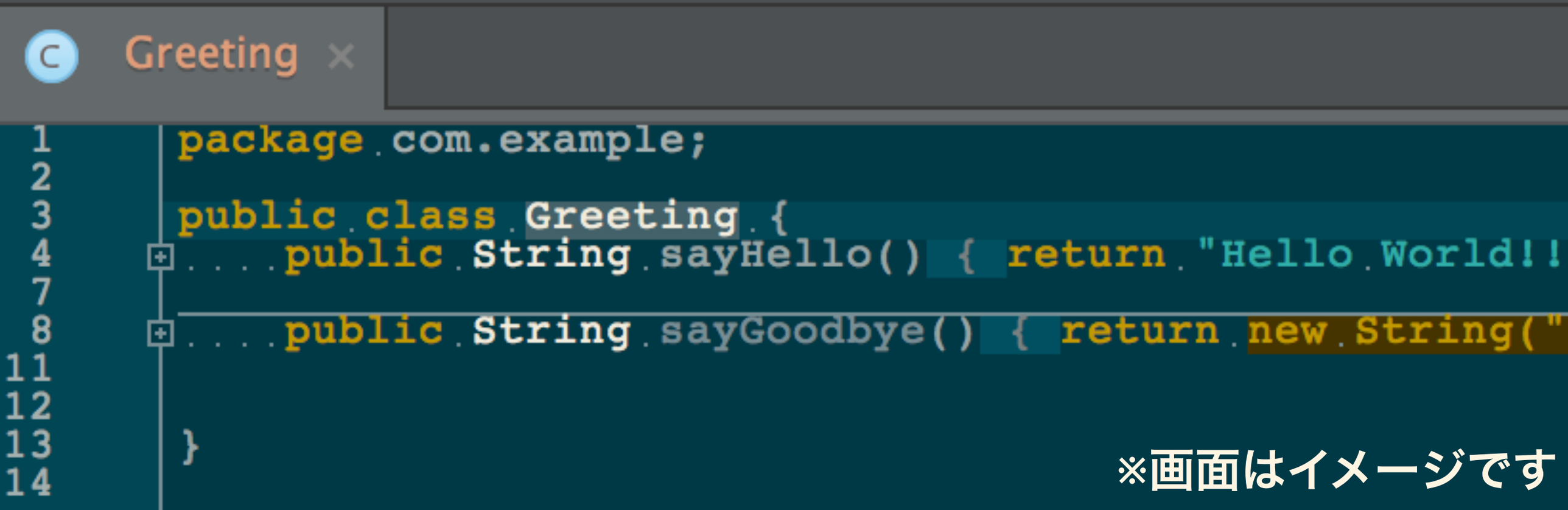
こんなこと
やっていませんか？

ビルドしてみてください



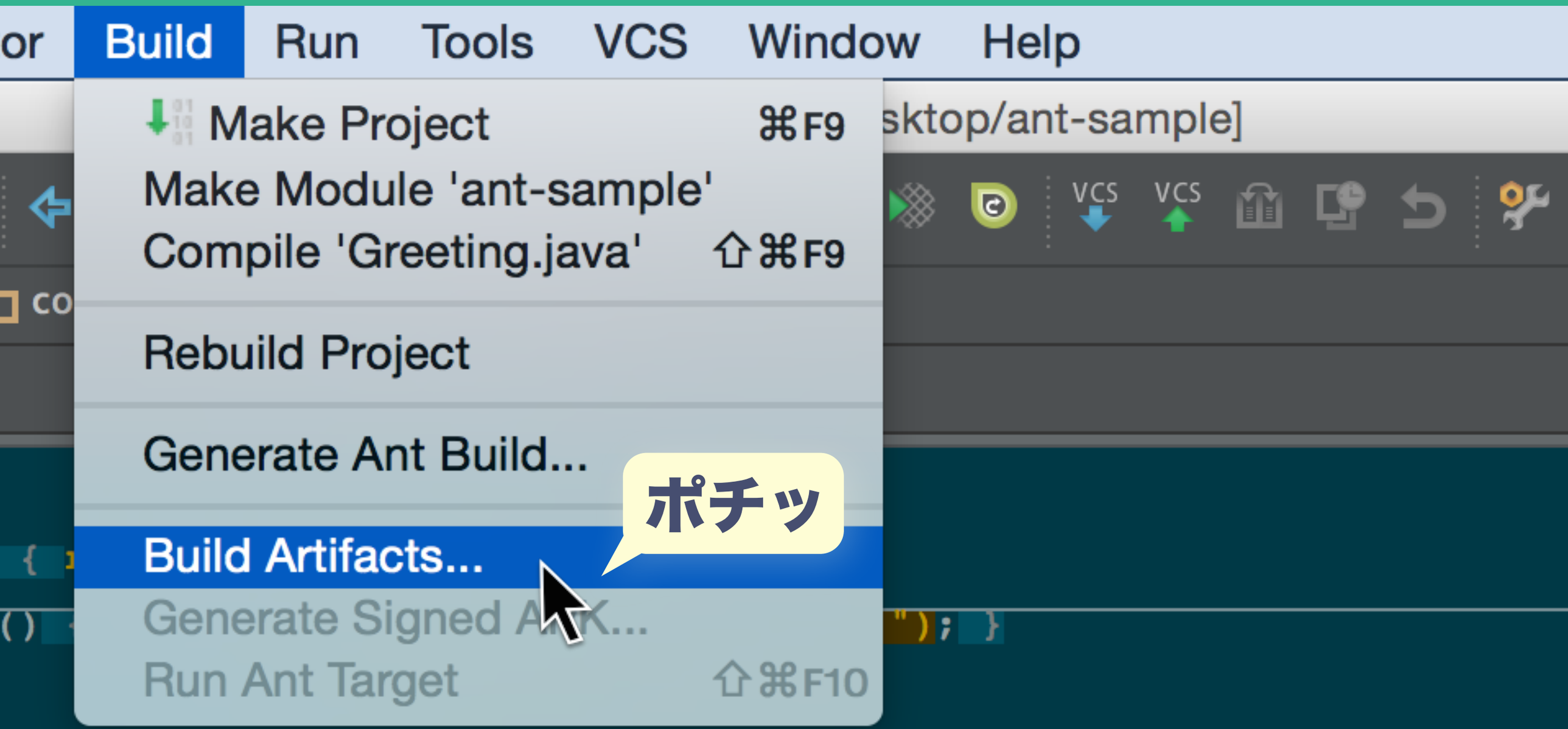
※画面はイメージです

テストしてみてください



※画面はイメージです

リリースしてみてください



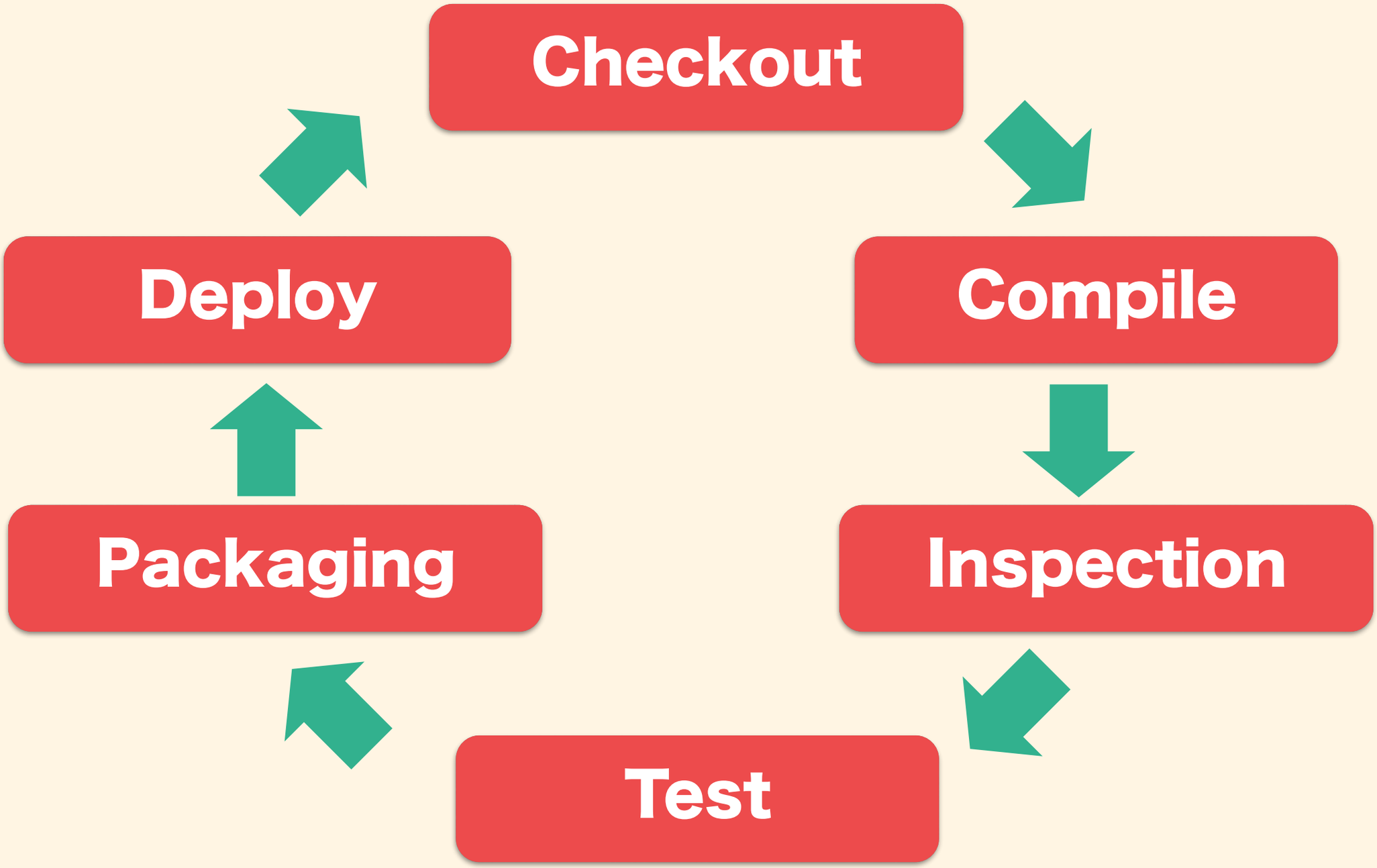
※画面はイメージです

それ自動化
できますか？

そこで
Jenkinsです！

とは
なりません…

日々の作業はIDE頼り



機械(Jenkins)に
任す前に、
ビルドツールで
ビルド手順をスクリプト化
します

Jenkinsは
IDEとは仲良くないが、
ビルドツールとは仲が良い

ビルドツールとは？

コンパイルやパッケージング
などの定型的な作業を自動化
するソフトウェア

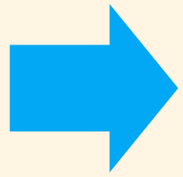
『Gradle徹底入門』より

ビルドツールでできること

- コンパイル
- テスト
- コード検査
- パッケージング
- ドキュメント作成
- などなど

ビルドツールの進化

	手続き的	規約中心
script	make	gradle
XML	ant	maven



『Gradle徹底入門』より

それでもボクはantを使う

- Mavenの型にはまるプロジェクトはそうない
- スタンドアロンでビルドできる
- XMLはイヤだけど、IDEのサポートがある

ビルドしてみてください

```
<target name="clean" description="クリーンアップ">
  <delete dir="${dest.dir}" />
</target>

<target name="compile" depends="prepare">
  <my.javac src="${src.java.dir}" destdir="${dest.java.dir}" />
</target>

<target name="build-webapps" description="WARファイルの作成"
depends="compile">
  <war destfile="${dest.dir}/${ant.project.name}.war"
     update="true" duplicate="preserve"
     webxml="${src.web.dir}/WEB-INF/web.xml">
    <classes dir="${dest.java.dir}" />
    <lib file="${lib.compile.dir}/*.jar" />
    <fileset dir="${src.web.dir}" excludes="**/web.xml" />
  </war>
</target>
```

ビルドしてみてください

```
$ ant build-webapps
```

テストしてみてください

```
<target name="compile-test" depends="compile">
  <my.javac src="${src.test.dir}" destdir=“${dest.test.dir}">
    <options>
      <classpath refid="test.classpath" />
    </options>
  </my.javac>
</target>
```

```
<target name="test" depends="compile-test" description="テストの実行">
  <delete dir="${result.test.dir}" />
  <mkdir dir="${result.test.dir}" />
  <junit fork="on" printsummary="on" maxmemory="256m">
    <classpath location="${dest.test.dir}" />
    <classpath refid="test.classpath" />
    <formatter type="xml" />
    <batchtest todir="${result.test.dir}">
      <fileset dir="${dest.test.dir}" includes="**/*Test.class" />
    </batchtest>
  </junit>
</target>
```

テストしてみてください

```
$ ant test
```

リリースしてみてください

```
<target name=“release"
        depends="clean, test, build-webapps, javadoc"
        description="リリースモジュールの作成">

  <zip destfile="${dest.dir}/${ant.project.name}.zip">
    <fileset file="${dest.dir}/${ant.project.name}.war" />
    <zipfileset dir="${dest.javadoc.dir}" prefix="javadoc" />
  </zip>
</target>
```

リリースしてみてください

```
$ ant release
```

antは便利だけれど、
すべての手順を記述しなければ
ならない面倒さをもつ

```
<?xml version="1.0" encoding="utf-8" ?>
<project name="sample-apps" default="build-webapps" basedir=".">
  <import file="build-common.xml" />
  <target name="clean" description="クリーンアップ">
    <delete dir="${dest.dir}" />
  </target>
  <target
    <mkdir dir="$
    <mkdir dir="$
    <mkdir d
    <mkdir d
  </target>
  <target na
    <my.j
  </targe
  <target name="build-webapps" description="WARファイルの作成" depends="compile-
main">
    <war destfile="${dest.dir}/${ant.project.name}.war"
         update="true" duplicate="
         webxml="${src.web.dir}/W
      <cl
      <li
      <fileset dir="${s
    </war>
  </target>
  <target name
    <my.javac
      <options>
        <classpath refid="test.classpath" />
      </options>
    </my.javac>
  </target>
  <target name="test" depends="compile-test" description="テストの実行とカバレッシ
ポートの作成">
    <!-- EclEmma(JaCoCo)の定義 -->
    <property name="jacoco.home" value="${tool.dir}/jacoco-0.5.6" />
    <taskdef resource="org/jacoco/ant/antlib.xml">
      <classpath path="${jacoco.home}/lib/jacocoant.jar" />
    </taskdef>
    <!-- ここまで -->
    <delete dir="${result.test.dir}" />
    <mkdir dir="${result.test.dir}" />
    <coverage>
      <junit fork="on" printsummary="on" maxmemory="256m">
        <classpath location="${dest.test.dir}" />
        <classpath refid="test.classpath" />

        <formatter type="xml" />
        <batchtest todir="${result.test.dir}">
          <fileset dir="${dest.test.dir}" includes="**/*Test.class" />
        </batchtest>
      </junit>
    </coverage>

    <report>
      <executiondata>
        <file file="jacoco.exec" />
```

```
      </executiondata>
      <structure name="${ant.project.name}">
        <classfiles>
          <fileset dir="${dest.java.dir}" />
        </classfiles>
        <sourcefiles encoding="UTF-8">
          <fileset dir="${src.java.dir}" />
        </sourcefiles>
      </structure>
      <xml destfile="${result.dir}/coverage_jacoco.xml" />
      <html destdir="${dest.dir}/report/coverage" />
    </report>
    <!-- JaCoCoのレポートをEMMA形式に変換する -->
    <xslt style="${jacoco.home}/jacoco_to_emma.xslt"
          in="${result.dir}/coverage_jacoco.xml"
          out="${result.dir}/coverage.xml"
          classpath="${jacoco.home}/saxon9he.jar">
      <xmlcatalog>
        <dtd publicid="-//JACOCO//DTD Report 1.0//EN" location="${jacoco.home}/
coverage/report.dtd" />
      </xmlcatalog>
    </xslt>
  </target>
  <target name="javadoc" depends="prepare" description="Javadocの作成">
    <javadoc sourcepath="${src.java.dir}" destdir="${dest.javadoc.dir}"
             packagenames="com.example.*"
             classpathref="base.classpath"
             encoding="utf-8" />
  </target>
  <target name="release" depends="clean, test, build-webapps, javadoc"
          description="リリースモジュールの作成">
    <zip destfile="${dest.dir}/${ant.project.name}.zip">
      <fileset file="${dest.dir}/${ant.project.name}.war" />
      <zipfileset dir="${dest.javadoc.dir}" prefix="javadoc" />
    </zip>
  </target>


  <target name="checkstyle" depends="prepare">
    <property name="checkstyle.home" value="${tool.dir}/checkstyle-5.1" />
    <taskdef resource="checkstyletask.properties"
             classpath="${checkstyle.home}/checkstyle-all-5.1.jar" />
    <checkstyle config="${checkstyle.home}/sun_checks.xml"
failOnViolation="no">
      <fileset dir="${src.java.dir}" includes="**/*.java" />
      <formatter type="xml" toFile="${result.dir}/checkstyle-result.xml" />
    </checkstyle>
  </target>
```

正直つらい

おまけ

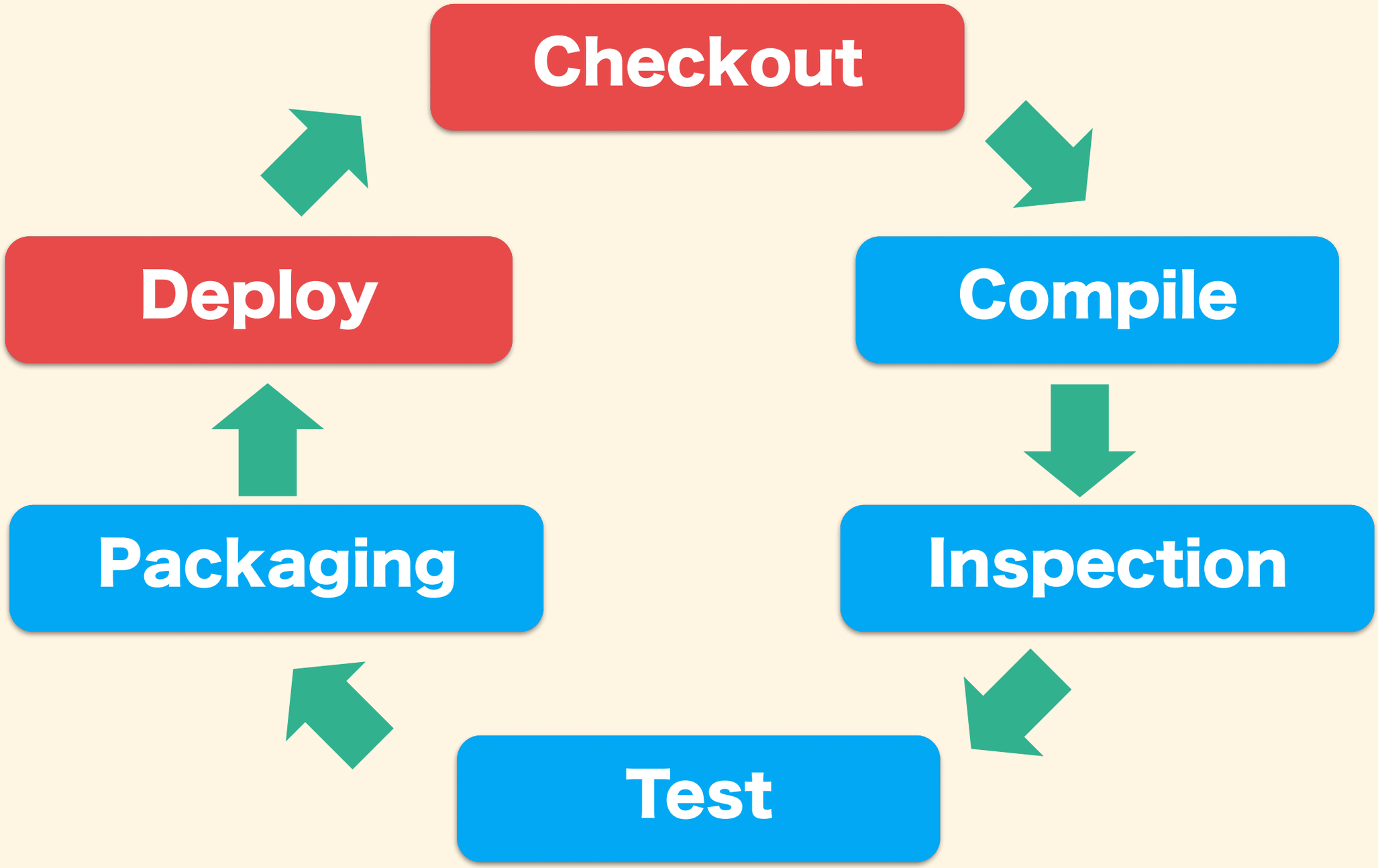
Gradleでやってみた

```
repositories {
    jcenter()
}

apply {
  plugin 'java'
  plugin 'war'
  plugin 'jetty'
}
```

拍子抜けするほど簡単にできる

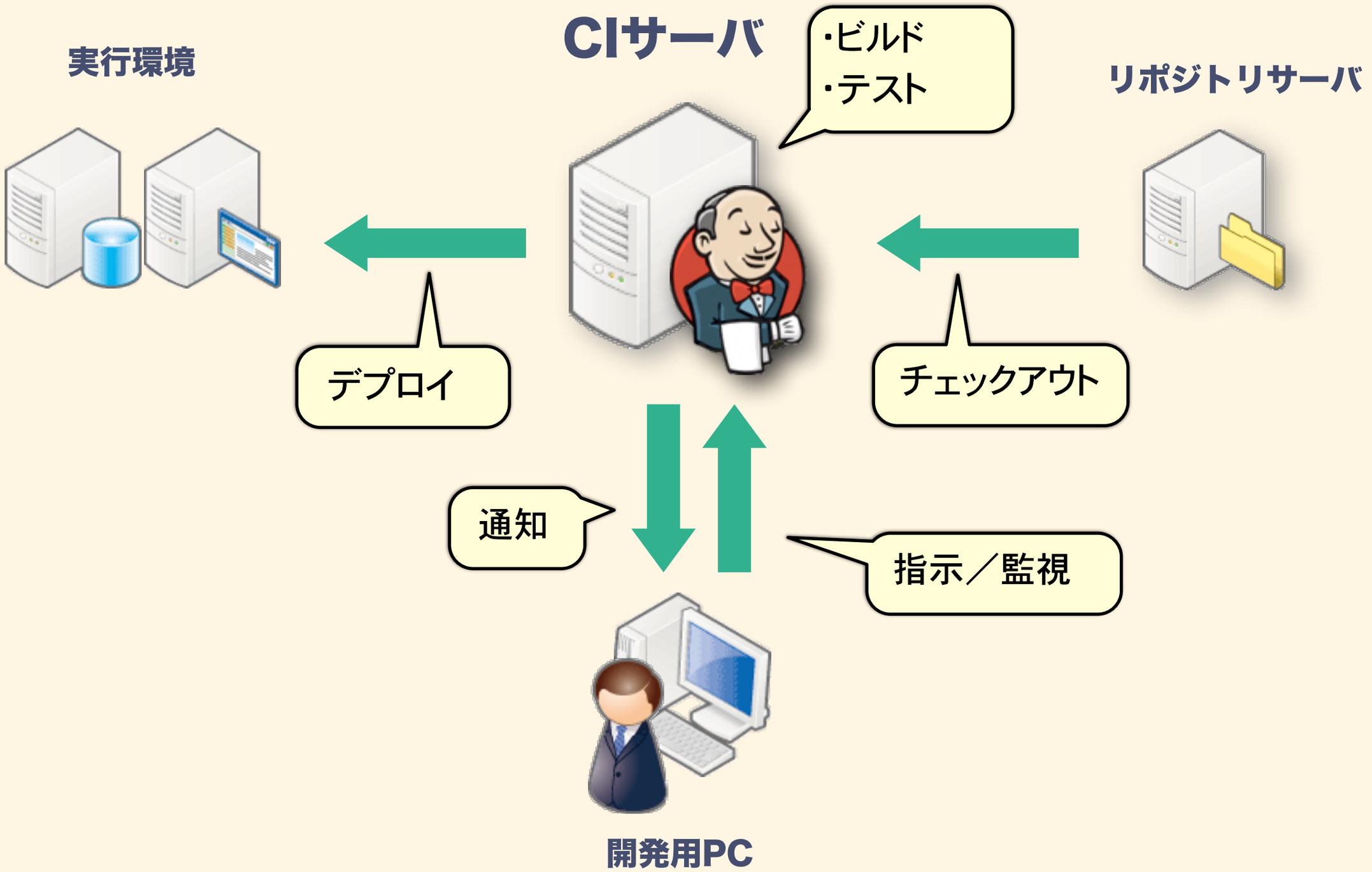
```
// jarファイルの名前
archivesBaseName = "sample-apps"
// javacのオプションに encoding=UTF-8 を追加する
[compileJava, compileTestJava]*.options*.encoding = 'UTF-8'

dependencies {
  testCompile 'junit:junit:4.12'
}
```

日々の作業をビルドツールに任せる



ここでようやく
Jenkinsの出番です！

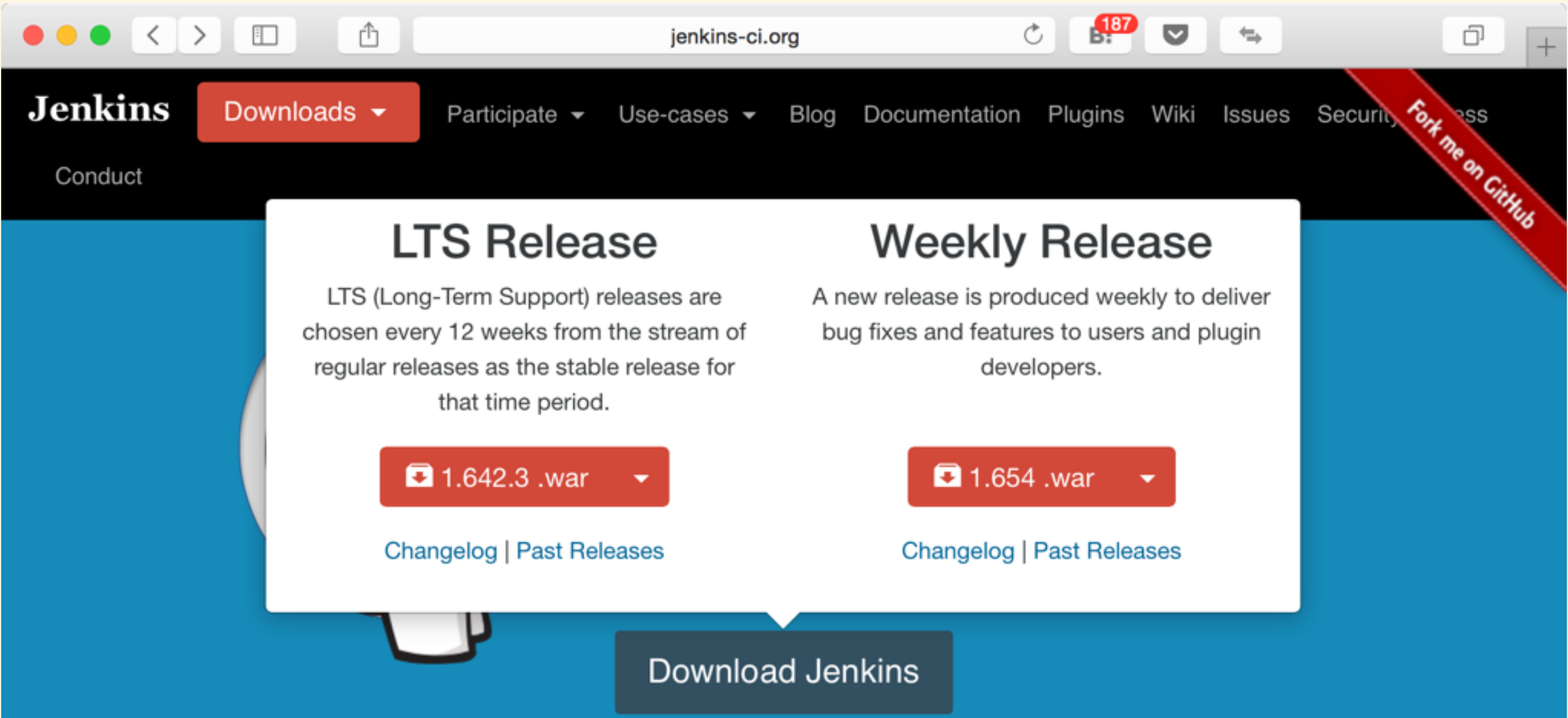
Jenkinsはじめの一歩



まずはインストール

jenkins-ci.org
187
Jenkins
Downloads
Participate
Use-cases
Blog
Documentation
Plugins
Wiki
Issues
Conduct
Fork me on GitHub
Jenkins
Build great things at any scale
The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.
Download Jenkins
Get 1.642.3 LTS .war or the latest 1.654 weekly release

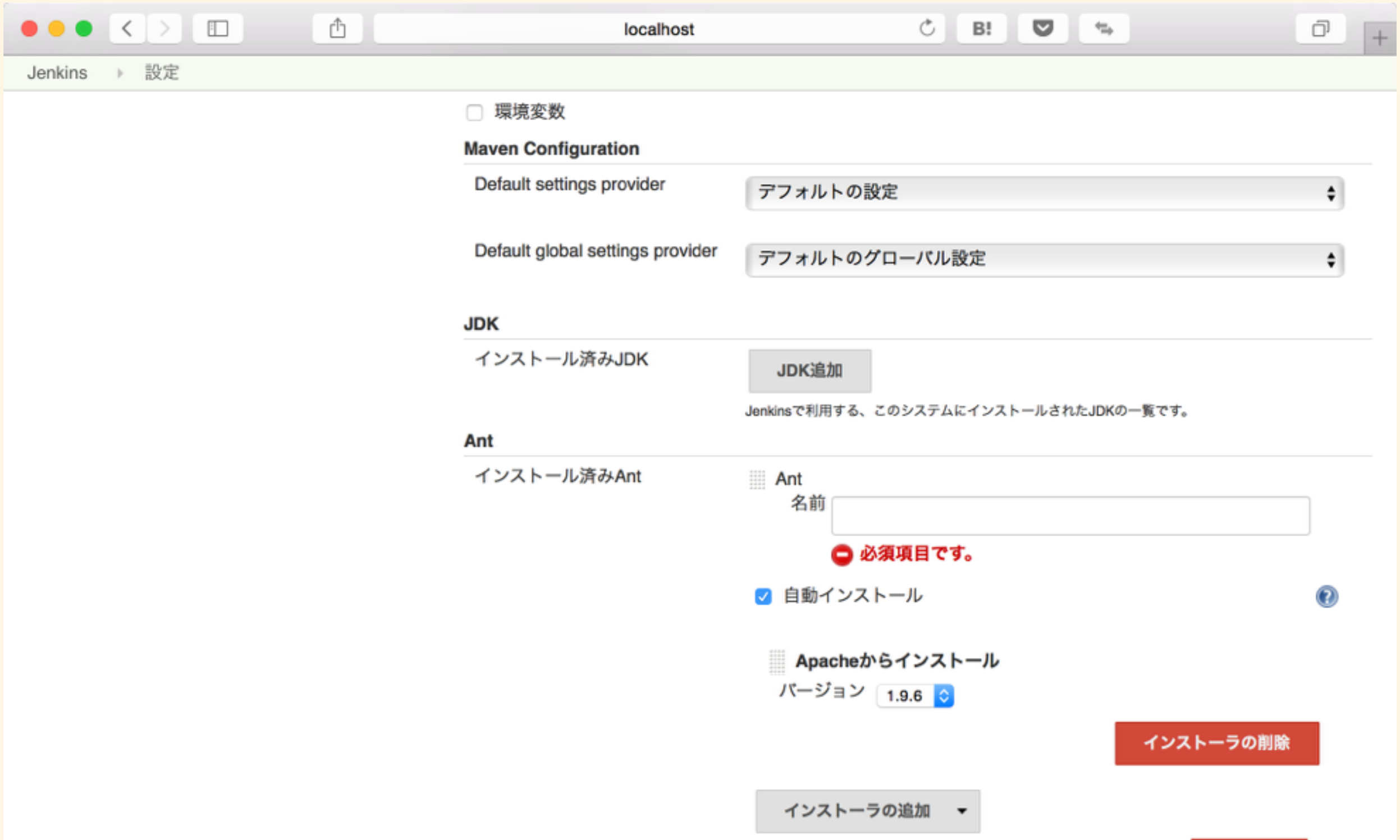
まずはインストール


jenkins-ci.org
187
Jenkins
Downloads
Participate
Use-cases
Blog
Documentation
Plugins
Wiki
Issues
Conduct
Fork me on GitHub
LTS Release
LTS (Long-Term Support) releases are chosen every 12 weeks from the stream of regular releases as the stable release for that time period.
1.642.3 .war
Changelog | Past Releases
Weekly Release
A new release is produced weekly to deliver bug fixes and features to users and plugin developers.
1.654 .war
Changelog | Past Releases
Download Jenkins
Get 1.642.3 LTS .war or the latest 1.654 weekly release

まずはインストール

```
$ java -jar jenkins.war
```

つづいて初期設定


localhost
Jenkins
設定
環境変数
Maven Configuration
Default settings provider
デフォルトの設定
Default global settings provider
デフォルトのグローバル設定
JDK
インストール済みJDK
JDK追加
Jenkinsで利用する、このシステムにインストールされたJDKの一覧です。
Ant
インストール済みAnt
Ant
名前
必須項目です。
自動インストール
Apacheからインストール
バージョン
1.9.6
インストーラの削除
インストーラの追加

覚えておくべき
Jenkins用語

- プロジェクト(ジョブ)
- ワークスペース
- ビルド

プロジェクト

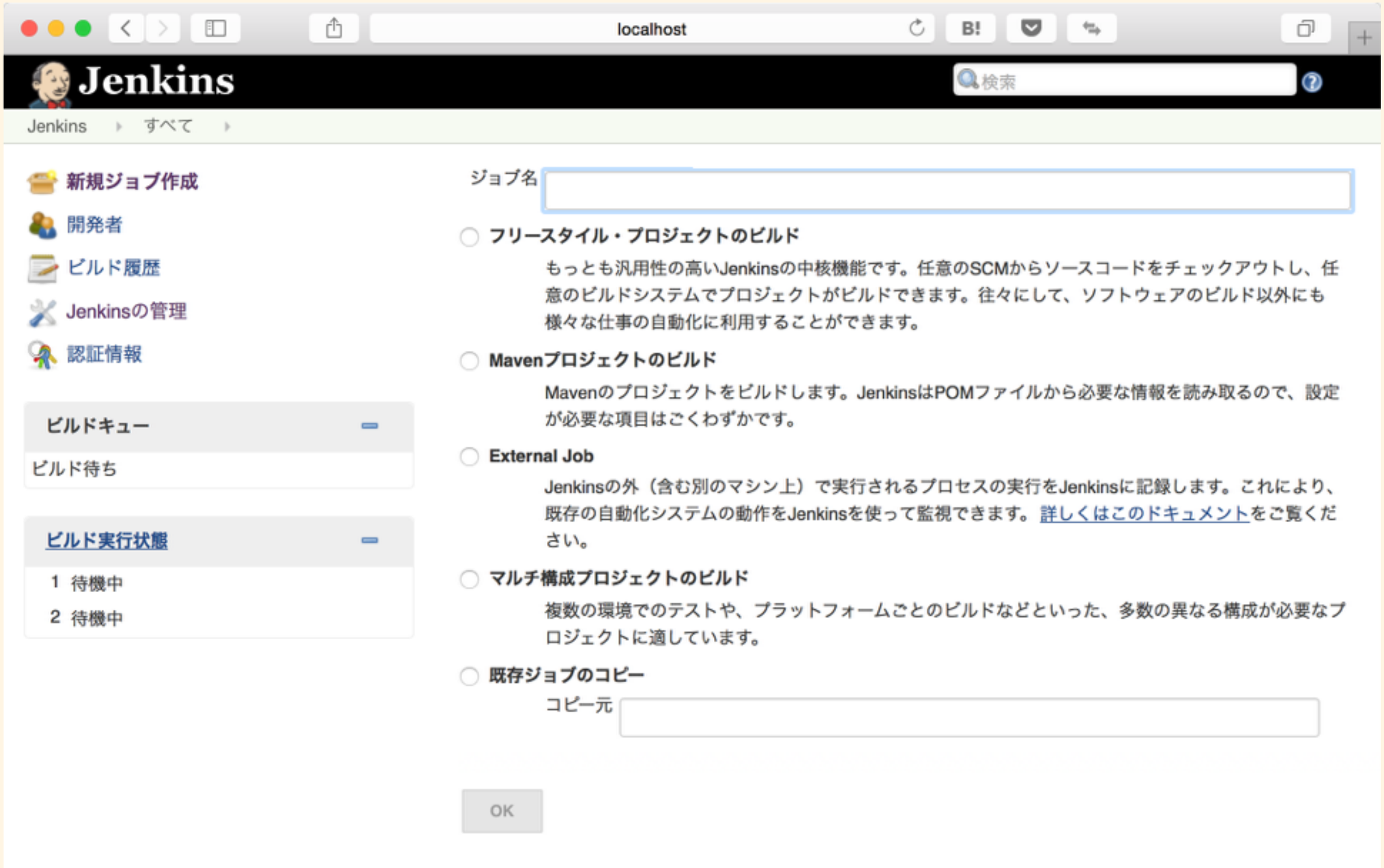
ワークスペース

ビルド#1

ビルド#2

⋮

ジョブの登録


localhost
Jenkins
検索
Jenkins
すべて
新規ジョブ作成
開発者
ビルド履歴
Jenkinsの管理
認証情報
ビルドキュー
ビルド待ち
ビルド実行状態
1 待機中
2 待機中
ジョブ名
フリースタイル・プロジェクトのビルド
もっとも汎用性の高いJenkinsの中核機能です。任意のSCMからソースコードをチェックアウトし、任意のビルドシステムでプロジェクトがビルドできます。往々にして、ソフトウェアのビルド以外にも様々な仕事の自動化に利用することができます。
Mavenプロジェクトのビルド
Mavenのプロジェクトをビルドします。JenkinsはPOMファイルから必要な情報を読み取るので、設定が必要な項目はごくわずかです。
External Job
Jenkinsの外（含む別のマシン上）で実行されるプロセスの実行をJenkinsに記録します。これにより、既存の自動化システムの動作をJenkinsを使って監視できます。詳しくはこのドキュメントをご覧ください。
マルチ構成プロジェクトのビルド
複数の環境でのテストや、プラットフォームごとのビルドなどといった、多数の異なる構成が必要なプロジェクトに適しています。
既存ジョブのコピー
コピー元
OK

ジョブの設定

localhost
Jenkins
ant-sample
設定
Subversion
ビルド・トリガ
SCMをポーリング
他プロジェクトの後にビルド
定期的に実行
ビルド
Antの呼び出し
使用するAnt
ant1.9.6
ターゲット
clean test build-webapps
高度な設定...
削除
ビルド手順の追加
ビルド後の処理
JUnitテスト結果の集計
テスト結果XML
dest/result/test/*.xml
Antのファイルセット includes属性の書式に従ってビルドから生成されるXMLレポートファイル
保存
適用

ビルドの実行

localhost
Jenkins
検索
Jenkins ▸ すべて ▸ ant-sample ▸
自動リロードをon
ダッシュボードへ戻る
状態
変更履歴
ワークスペース
ビルド実行
プロジェクトの削除
設定
プロジェクト ant-sample
説明を記入
プロジェクトの無効化
ワークスペース
変更履歴
最新のテスト結果 (1個の失敗)
ビルド履歴
推移
find
#5 2016/04/02 21:13
#4 2016/04/02 21:12
#3 2016/04/02 21:11
#2 2016/04/02 21:08
#1 2016/04/02 21:07
RSS 全ビルド
RSS 失敗ビルド
永続リンク
最新のビルド (#5), 12 秒 前
最新の安定ビルド (#2), 5 分 30 秒 前
最新の成功ビルド (#5), 12 秒 前
最新の失敗ビルド (#4), 50 秒 前
最新の不安定ビルド (#5), 12 秒 前

Jenkins
すごい便利！！

もっと
いろいろ自動化したい

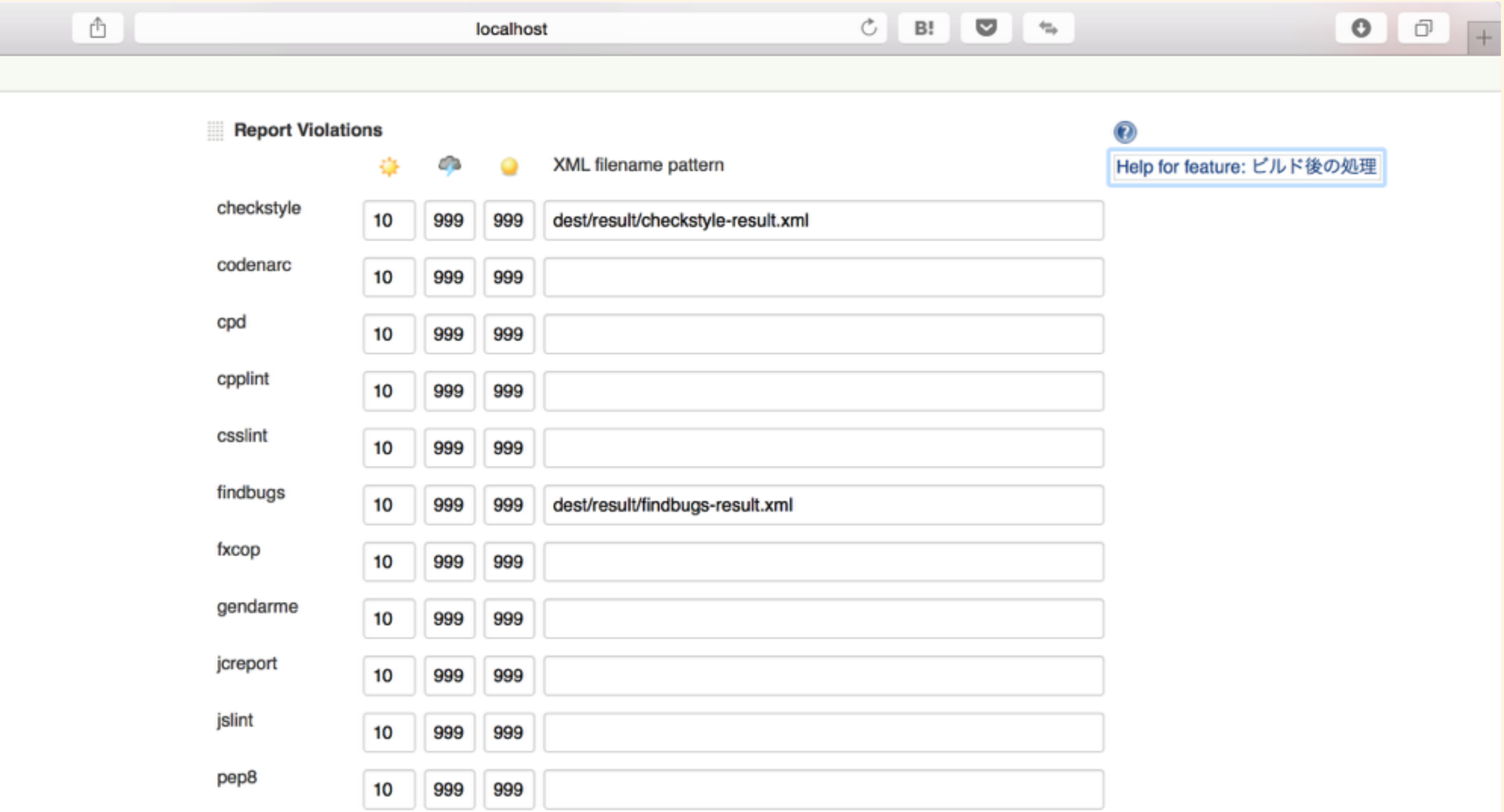
コード検査したい！

```
  <target name="checkstyle" depends="prepare">
    <property name="checkstyle.home" value="${tool.dir}/checkstyle-5.1" />
    <taskdef resource="checkstyletask.properties"
             classpath="${checkstyle.home}/checkstyle-all-5.1.jar" />
    <checkstyle config="${checkstyle.home}/sun_checks.xml"
failOnViolation="no">
      <fileset dir="${src.java.dir}" includes="**/*.java" />
      <formatter type="xml" toFile="${result.dir}/checkstyle-result.xml" />
    </checkstyle>
  </target> <target name="findbugs" depends="compile-main">
    <property name="findbugs.home" value="${tool.dir}/findbugs-1.3.9" />
    <taskdef name="findbugs"
             classname="edu.umd.cs.findbugs.anttask.FindBugsTask"
             classpath="${findbugs.home}/lib/findbugs-ant.jar" />

    <findbugs home="${findbugs.home}"
              output="xml"
              outputFile="${result.dir}/findbugs-result.xml">
      <sourcePath path="${src.java.dir}" />
      <class location="${dest.java.dir}" />
    </findbugs>
  </target>

  <target name="inspection" depends="checkstyle, findbugs" description="イン
スペクション">

  </target>
```

ビルドスクリプトに
コード検査を組み込む

```
$ ant inspection
```

Jenkinsに
その結果を取り込む



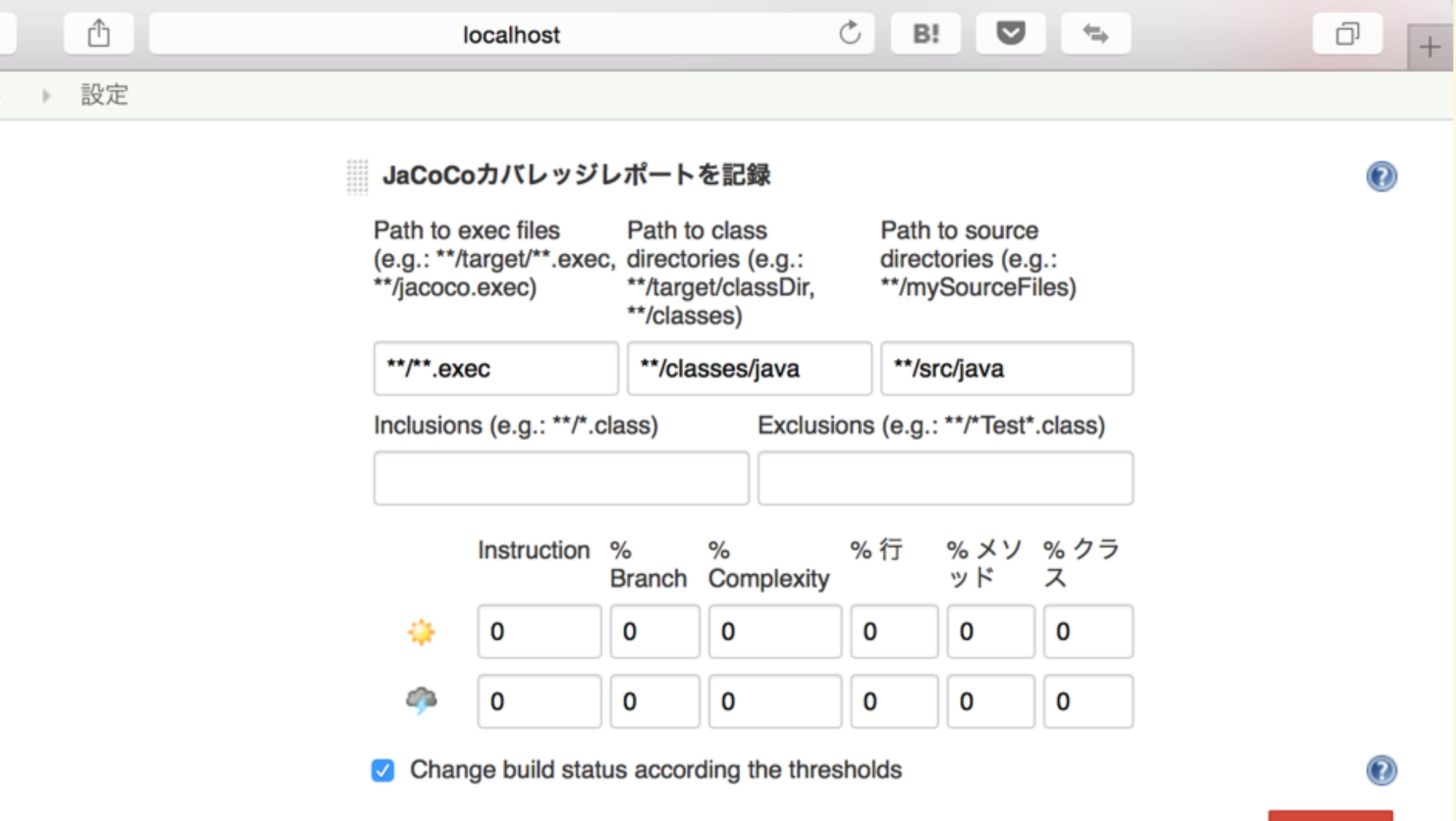
カバレッジを測定したい！

```
<target name="test" depends="compile-test" description="テストの実行とカバレッジレポートの作成">
  <!-- EclEmma(JaCoCo)の定義 -->
  <property name="jacoco.home" value="${tool.dir}/jacoco-0.5.6" />
  <taskdef resource="org/jacoco/ant/antlib.xml">
    <classpath path="${jacoco.home}/lib/jacocoant.jar" />
  </taskdef>
  <!-- ここまで -->

  <delete dir="${result.test.dir}" />
  <mkdir dir="${result.test.dir}" />
  <coverage>
    <junit fork="on" printsummary="on" maxmemory="256m">
      <classpath location="${dest.test.dir}" />
      <classpath refid="test.classpath" />

      <formatter type="xml" />
      <batchtest todir="${result.test.dir}">
        <fileset dir="${dest.test.dir}" includes="**/*Test.class" />
      </batchtest>
    </junit>
  </coverage>

  <report>
    <executiondata>
      <file file="jacoco.exec" />
    </executiondata>
    <structure name="${ant.project.name}">
      <classfiles>
        <fileset dir="${dest.java.dir}" />
      </classfiles>
      <sourcefiles encoding="UTF-8">
        <fileset dir="${src.java.dir}" />
      </sourcefiles>
    </structure>
    <xml destfile="${result.dir}/coverage_jacoco.xml" />
    <html destdir="${dest.dir}/report/coverage" />

  </report>
  <!-- JaCoCoのレポートをEMMA形式に変換する -->
  <xslt style="${jacoco.home}/jacoco_to_emma.xslt"
        in="${result.dir}/coverage_jacoco.xml"
        out="${result.dir}/coverage.xml"
        classpath="${jacoco.home}/saxon9he.jar">
    <xmlcatalog>
      <dtd publicid="-//JACOCO//DTD Report 1.0//EN" location="${jacoco.home}/coverage/report.dtd" />
    </xmlcatalog>
```

ビルドスクリプトに
カバレッジ測定を組み込む

```
$ ant test
```

Jenkinsに
その結果を取り込む



```
repositories {
    jcenter()
}

apply {
  plugin 'java'
  plugin 'war'
  plugin 'jetty'
  plugin 'checkstyle'
  plugin 'findbugs'
  plugin 'jacoco'
}

// jarファイルの名前
archivesBaseName = "sample-apps"
// javacのオプションに encoding=UTF-8 を追加する
[compileJava, compileTestJava]*.options*.encoding = 'UTF-8'

dependencies {
  testCompile 'junit:junit:4.12'
}
```

ちなみに Gradleの場合

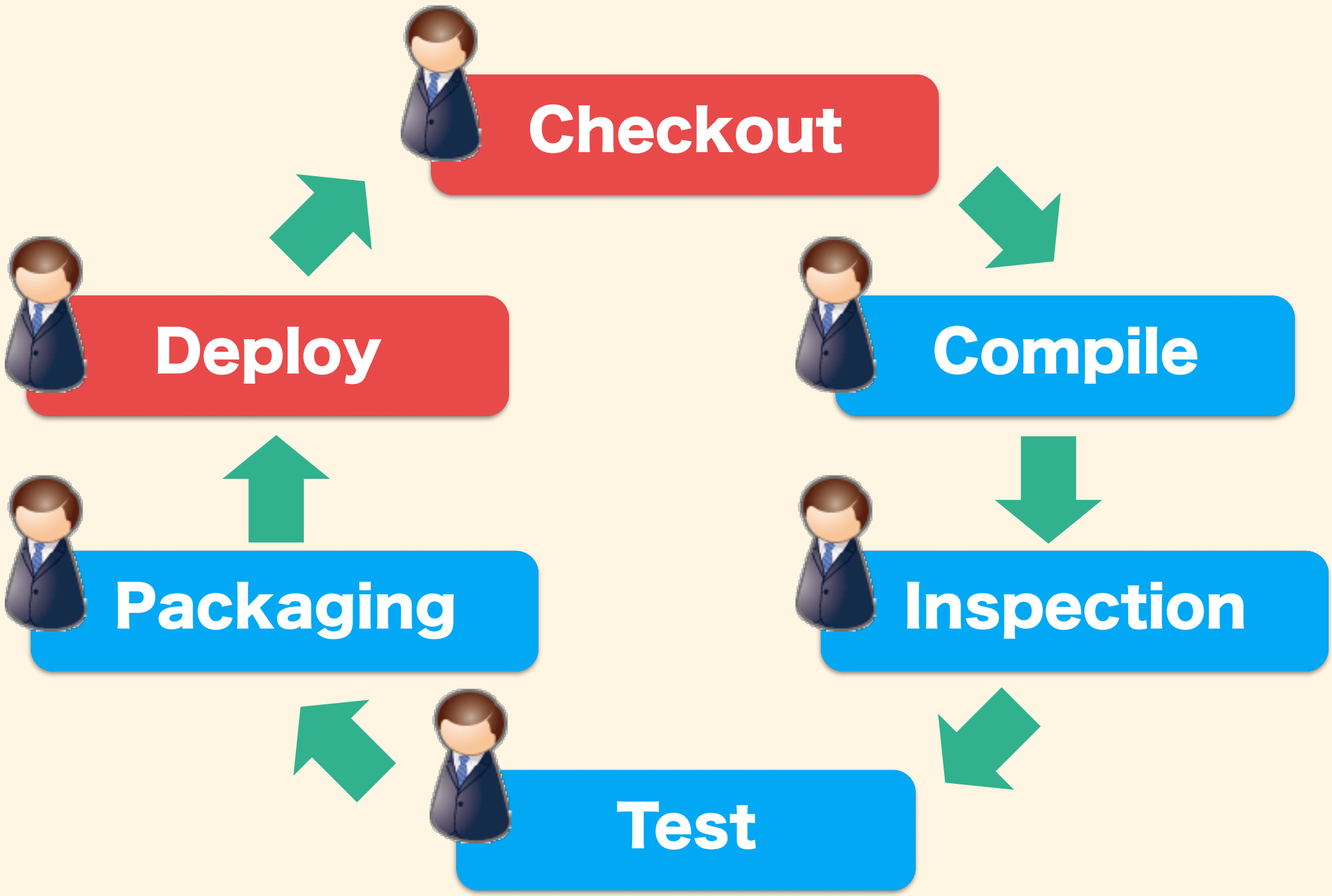
もしかして…

Jenkinsって
ビルドに関しては
何にもしてないのでは？

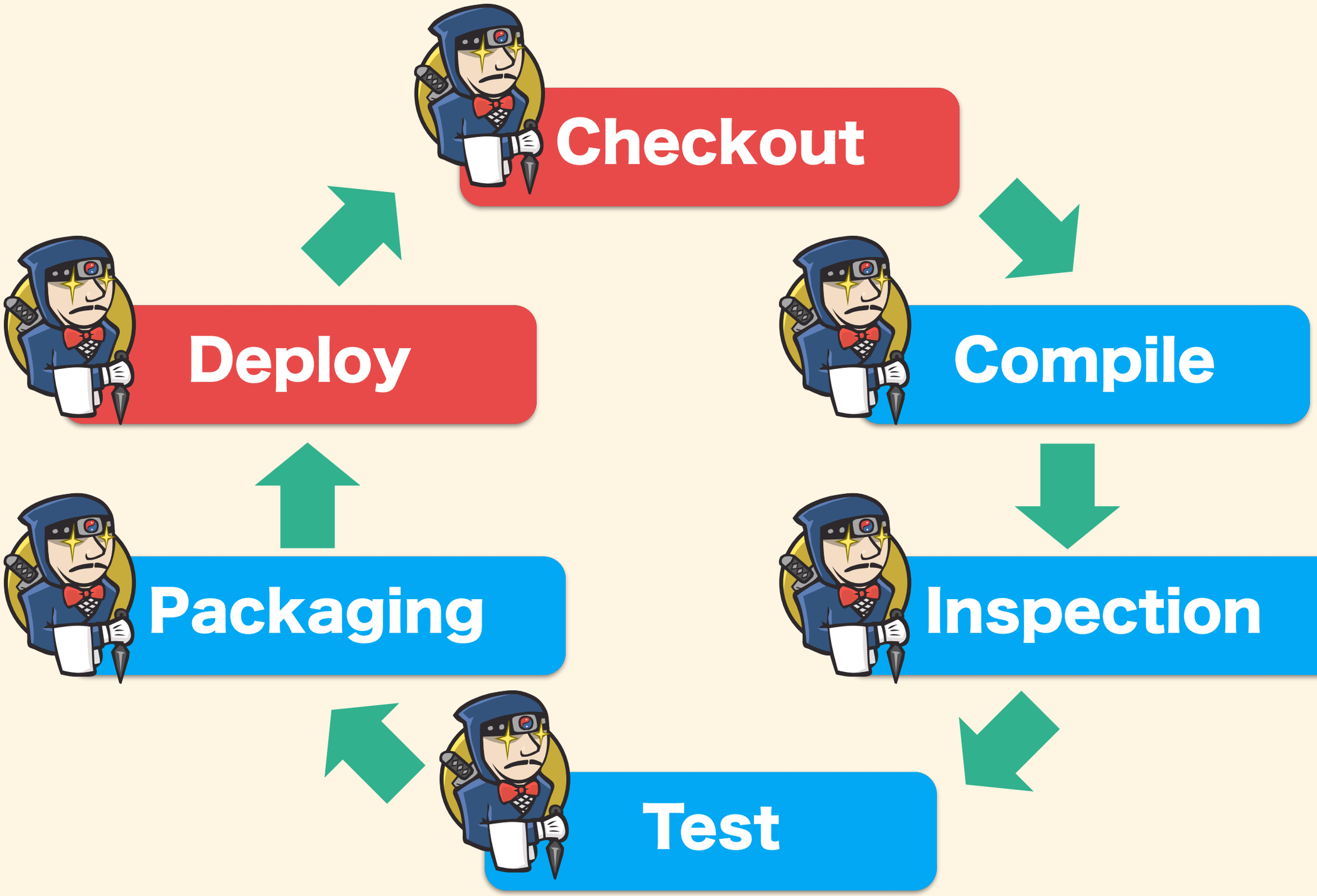
当たりです

ビルドは
ビルドツールの
仕事です

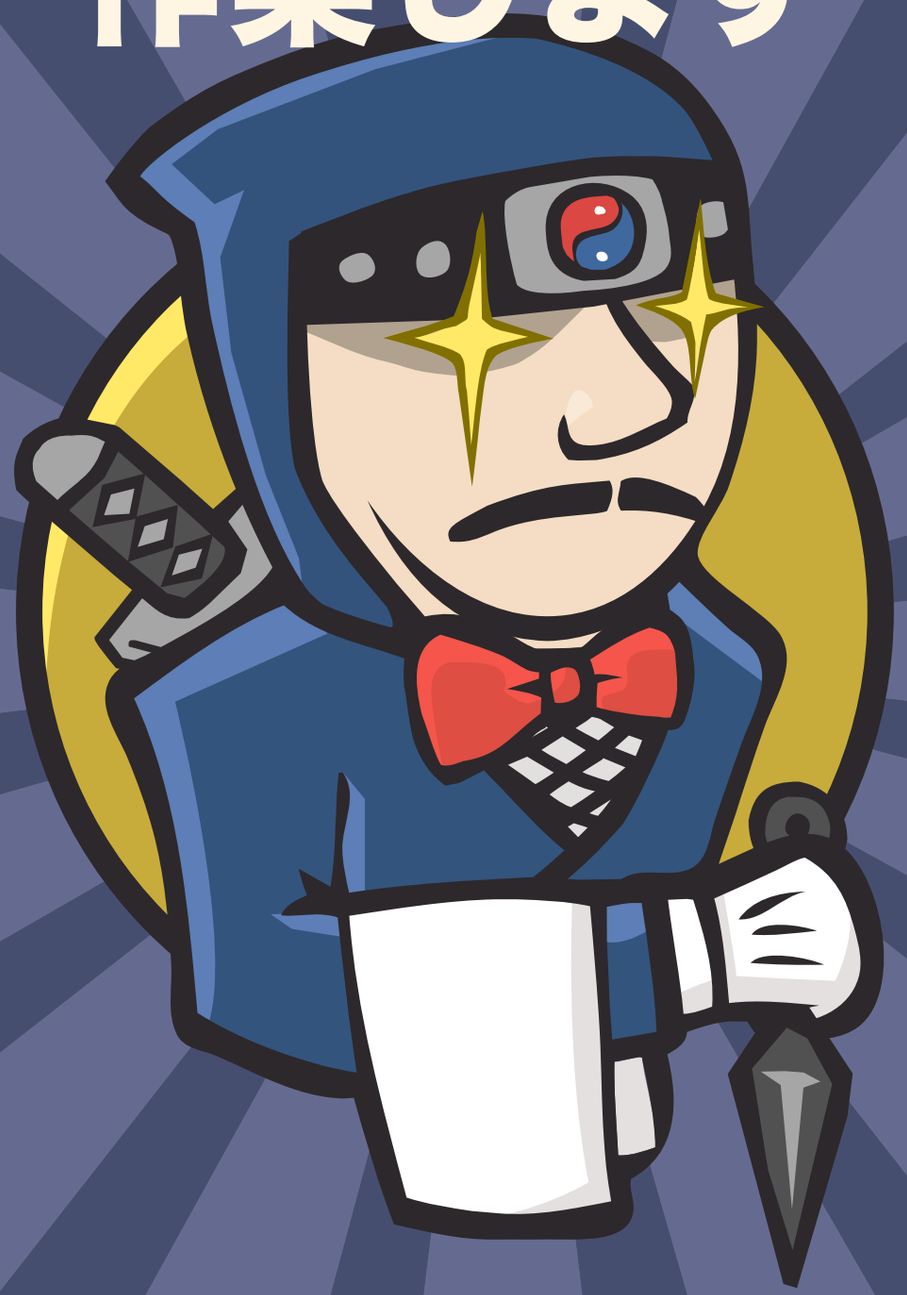
人が与えていたキッカケを…



Jenkinsが与えます



忙しいあなたに代わって
Jenkinsが昼夜問わず
作業します



もう一度
繰り返します

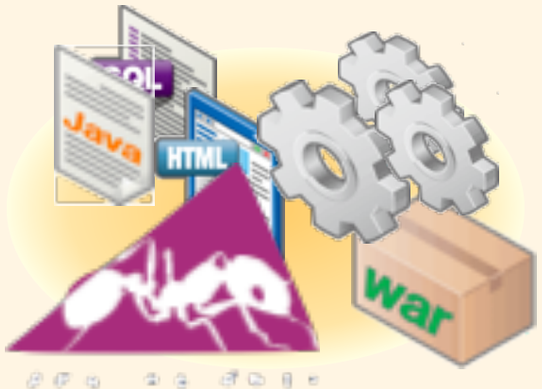
超高機能

cron

かみ砕いて紹介すると
こんな感じ

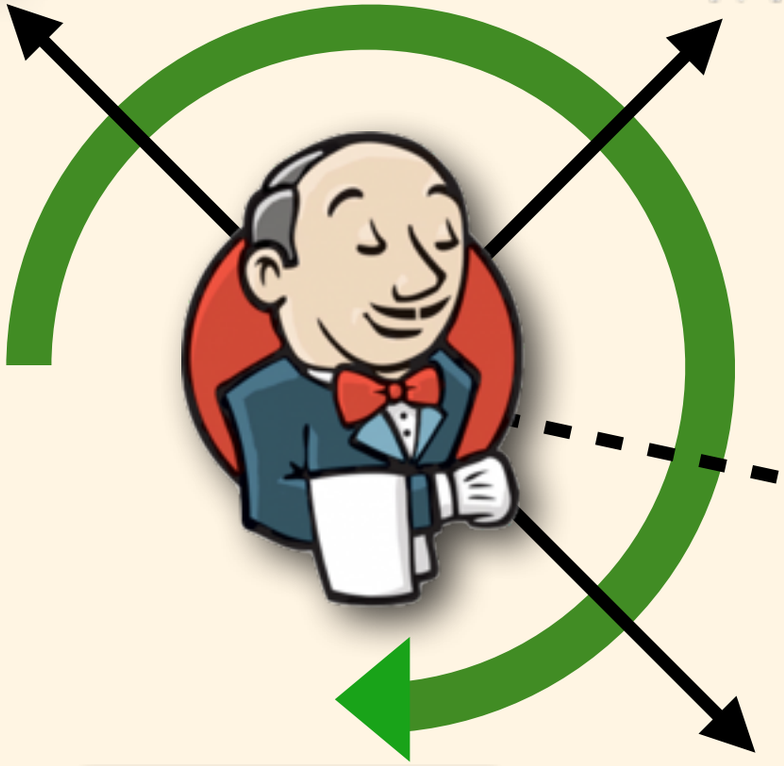
チェックアウト

ビルド実行



ビルドはビルドツールが行う

☆イベント発生



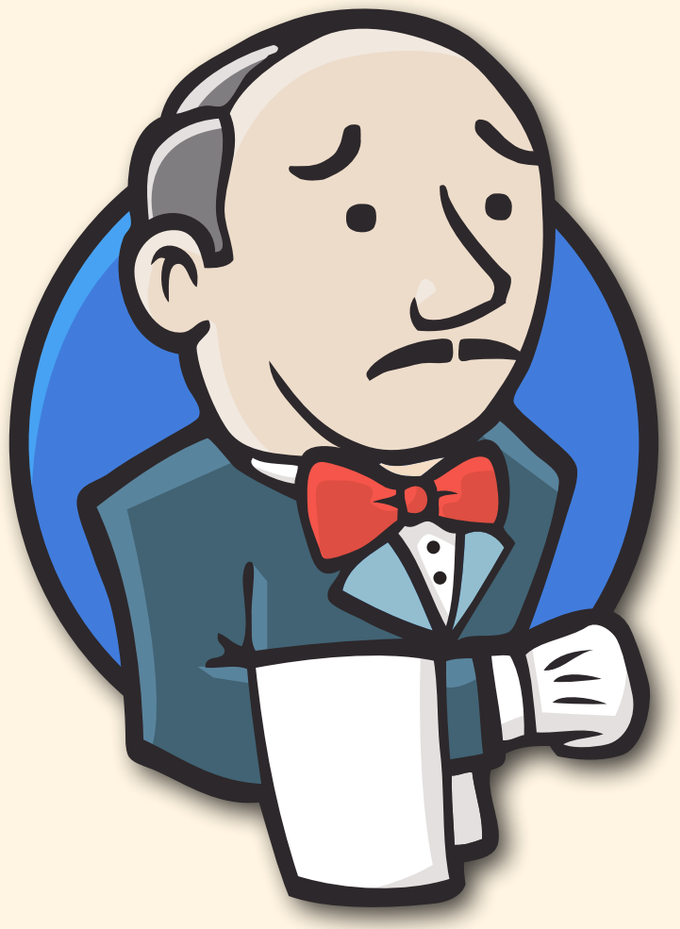
ビルドの後処理

他のジョブを起動する事も可能

レポートや通知を行う

1.イベントを起こす

- 定刻になったとき
- リポジトリが更新されたとき
- 他のジョブが成功したとき
- などなど



2.作業場所を準備する

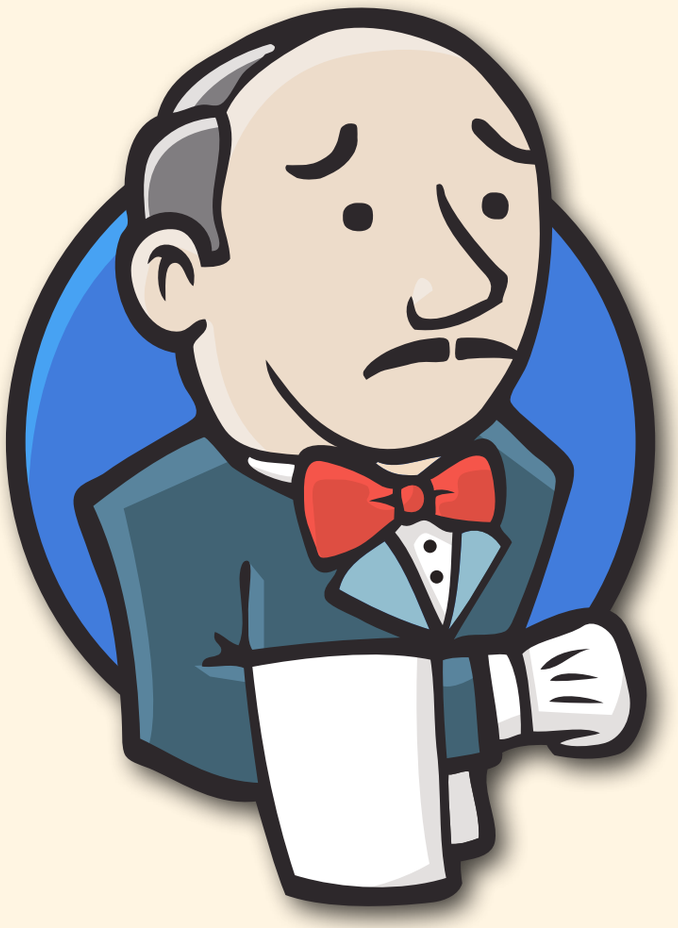
- リポジトリからチェックアウト
- 指定したパスからコピー
- などなど

3.ビルドを実行する

- ビルドツールを呼び出す
- シェルやバッチを起動する
- などなど

4.後始末をする

- ビルド結果からレポートを作る
- メールやSNSに通知する
- 他のジョブを実行する
- などなど



1.イベントを起こす
2.作業場所を準備する
3.ビルドを実行する
4.後始末をする

この4つのステップを
どう使うも、あなたの自由

たとえば…

- Jenkinsそのもののバックアップを行う
- URLを監視して、メールで通知する
- JenkinsのUIから、手動でジョブを実行
- などなど

ビルド以外に
Jenkinsを利用したって
良い！

極端な例ですが

localhost

ploy-sample ▸ 設定

Subversion

ビルド・トリガ

- [ ] SCMをポーリング
- [ ] 他プロジェクトの後にビルド
- [ ] 定期的に実行

ビルド

シェルの実行

シェルスクリプト

```
if [ -f sample-apps.war ];then
  rm sample-apps.war
fi
wget http://localhost:8080/job/ant-sample/lastSuccessfulBuild/artifact/dest/sample-apps.war
cp sample-apps.war /Users/masanobuimai/public/
```

ビルドから利用可能な環境変数の一覧

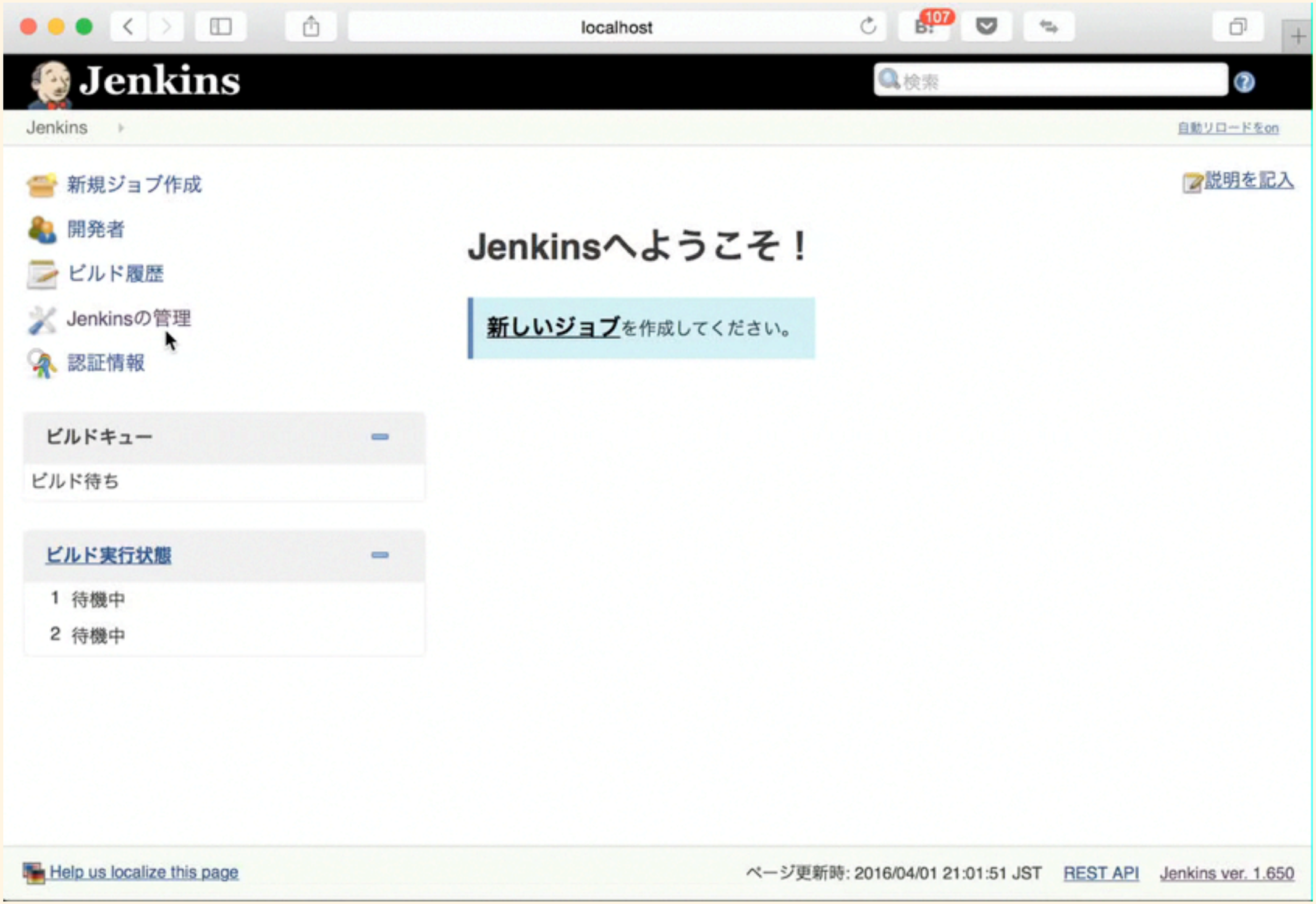
削除

ビルド手順の追加

ビルド後の処理

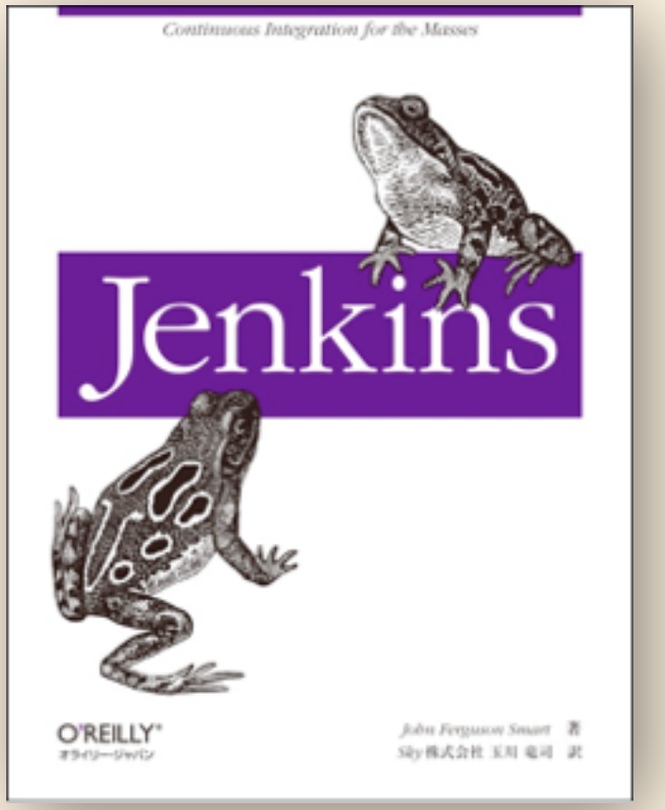
ビルド後の処理の追加

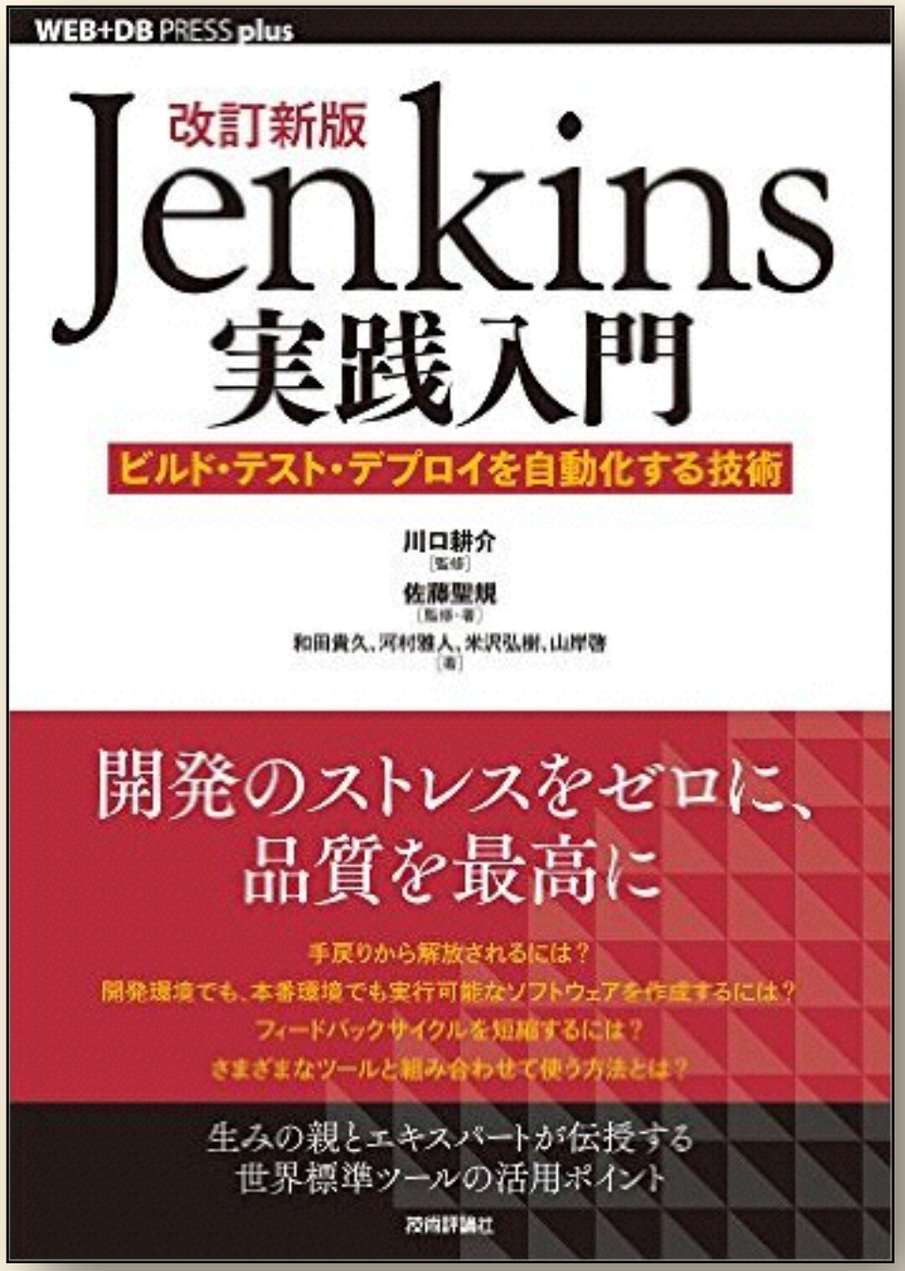
数多くのプラグインが
Jenkinsの可能性を広げます

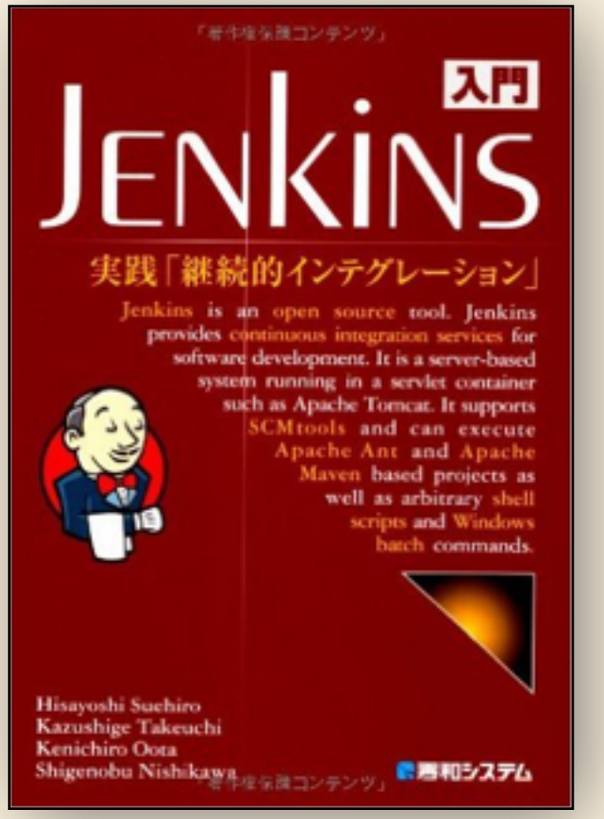


Jenkinsを使いこなす上で、ビルドツールやプラグインの知識は必須

【宣伝】書籍紹介


Continuous Integration for the Masses
Jenkins
O'REILLY
オライリージャパン
John Ferguson Smart 著
Sky株式会社 玉川 竜司 訳


WEB+DB PRESS plus
改訂新版
Jenkins
実践入門
ビルド・テスト・デプロイを自動化する技術
川口耕介
[監修]
佐藤聖規
[監修・著]
和田貴久、河村雅人、米沢弘樹、山岸啓
[著]
開発のストレスをゼロに、
品質を最高に
手戻りから解放されるには？
開発環境でも、本番環境でも実行可能なソフトウェアを作成するには？
フィードバックサイクルを短縮するには？
さまざまなツールと組み合わせて使う方法とは？
生みの親とエキスパートが伝授する
世界標準ツールの活用ポイント
技術評論社


「著作権保護コンテンツ」
入門
JENKINS
実践「継続的インテグレーション」
Jenkins is an open source tool. Jenkins provides continuous integration services for software development. It is a server-based system running in a servlet container such as Apache Tomcat. It supports SCMtools and can execute Apache Ant and Apache Maven based projects as well as arbitrary shell scripts and Windows batch commands.
Hisayoshi Suehiro
Kazushige Takeuchi
Kenichiro Oota
Shigenobu Nishikawa
「著作権保護コンテンツ」
秀和システム

自動化を阻害する大罪は

勤勉

Kohsuke Kawaguchi
@kohsukekawa

フォロー中

@masanobuimai 勤勉こそが自動化の敵です。怠け者万歳

返信 リツイート お気に入り

怠け者になって
Jenkinsに働いてもらおう

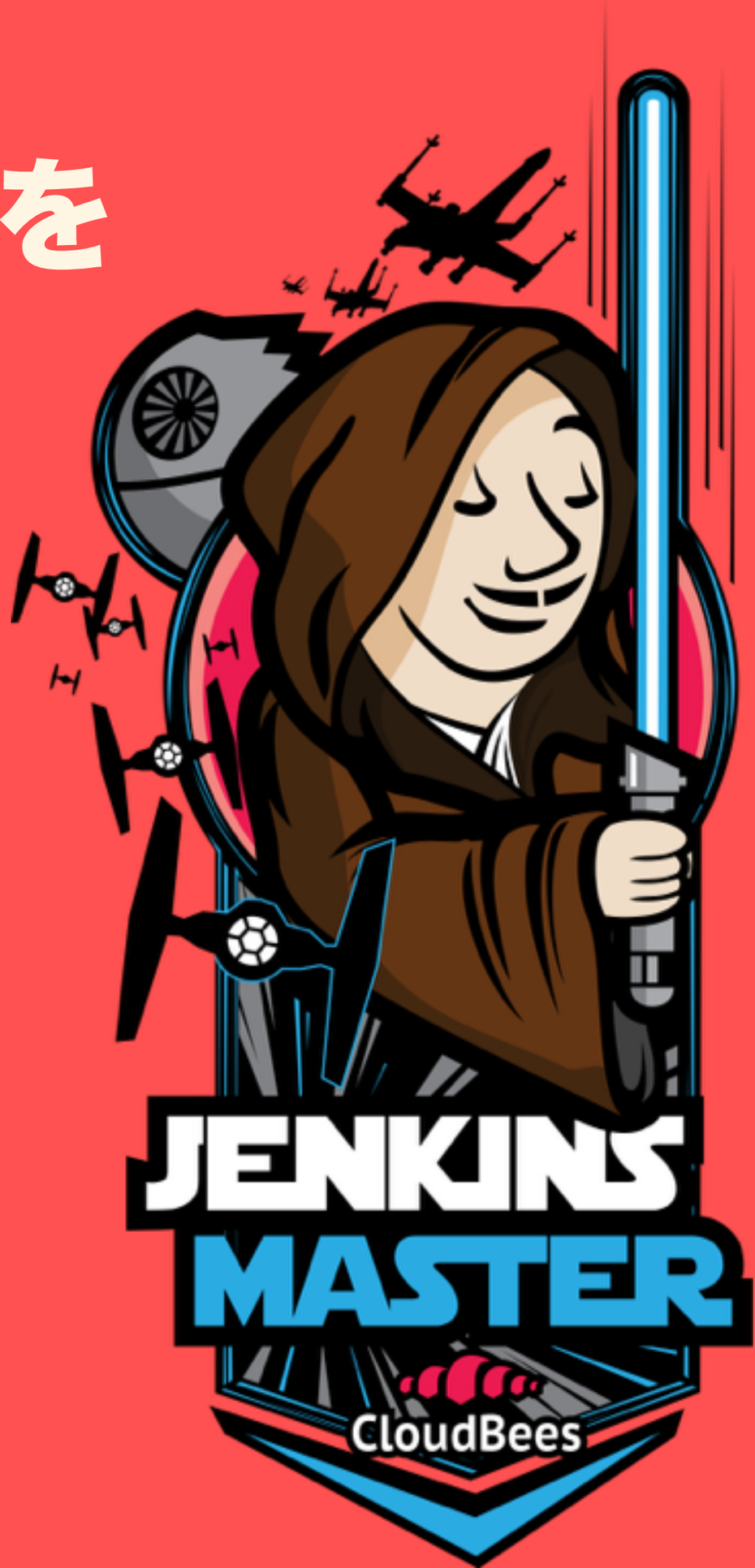
ただし

自動化も
やり過ぎには
注意！！

一番大切なのは

バランス

Jenkinsマスターを目指しましょう



ご静聴
ありがとう
ございました

Azusa Colors Keynote Theme
http://sanographix.github.io/azusa-colors/

Jenkinsではじめる継続的インテグレーション

JaSST'16 Niigata

番外編

より良い
プロジェクトとは

MavenやGradleが使えるならそれに従う

それが
叶わない場合

大事な
こと

3つ

目的に応じて
ディレクトリが
わかれている

必要なものは
すべて用意されている

IDEとの親和性が高い

プロジェクト構造、
つまりディレクトリ構成を
考えるのは大事なことです。

たかがディレクトリ構造と
侮ることなかれ。