CloudとSaaS型CI/CDツールを利用した開発ガイドライン

https://github.com/check-c-search/gh_pages_acr_result

2020-04-15

目次

1. はじめに

このガイドラインでは、クラウドネイティブな開発を実現する上で要となるCloud環境とCI/CD（継続的インテグレーション/デリバリー）ツールを組み合わせた開発の指針を提示します。

1.1. 背景

CI/CDツール

- CI/CDツールについてはJenkinsが最も有名であり、多くの運用事例がありますが、実運用に足る構成を実現するためには、チューニングに多くの知識が必要となるため、作業の属人化が発生しやすく、また構築難易度・維持コスト共に高くなるという課題があります。
- 一方で、近年ではTraviceCI、CircleCIと言ったSaaS型のCI/CDツールの運用事例も増えています。また、主要なSaaS型ツールはチューニングが設定ファイル1つであることやサービス利用型の特性から、Jenkins運用で懸念される課題を改善することが期待されます。
- そのため、このガイドラインではCI/CDをSaaS型のツールを利用して実施します。

Cloud環境

- 近年のPublicクラウド市場は今だ20%以上の年率で成長していますが、徐々に従来のAWS一強状態からの変化が生まれており、MicrosoftAzureやAlibabaCloudといった脱AWSの選択肢をとる企業も増えてきているため、今後はマルチクラウド対応が求められます。
- そのため、このガイドラインではAWS以外のCloud環境に対する知見を得る意味も込めてCloud環境として「AlibabaCloud」を利用します。

1.2. 全体像

このガイドラインで取り扱う環境の全体像は下図の通りです。

[adopt] | network.png

Cloud環境・CI/CDツール以外の構成要素として、CI/CDの結果を確認する手段として「Slack」を、リソースのバージョン管理ツールとして「GitHub」を利用します。

2. クラウド環境の設定指針

本手順で用いる Asciidoc 文書変換用スクリプトはビルドツールである Gradle
を活用しており、実行するためには Java 実行環境が必要です。

Java 実行環境は、文書変換スクリプトを動作させる過程で唯一 OS
環境に手動で導入する必要があるプロダクトです。それ以外のものは Gradle
によりプロジェクトとして独立した形で自動的に導入されます。

お使いのコンピューターのコマンドライン環境（macOS/Linux ではターミナル、Windows では cmd.exe か powershell.exe）で `java -version` コマンドを入力し、Java 8
以上のバージョンが表示されるようであれば既に準備は整っています。

macOS/Linux の場合

```
$ java -version
openjdk version "1.8.0_192"
OpenJDK Runtime Environment (Zulu 8.33.0.1-macosx) (build 1.8.0_192-b01)
OpenJDK 64-Bit Server VM (Zulu 8.33.0.1-macosx) (build 25.192-b01, mixed mode)
```

Windows の場合

```
C:¥> java -version
openjdk version "1.8.0_192"
OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_192-b12)
OpenJDK 64-Bit Server VM (AdoptOpenJDK)(build 25.192-b12, mixed mode)
```

現在 Java 9 以降の環境ではビルド時にワーニングが出力されるため本手順では Java 8
を使って解説しています。筆者の Java 11
環境で変換の動作は正しいことが確認できていますので適宜読み替えて Java
を導入してください。この問題は将来解消されるでしょう。

2.1. Java 実行環境の導入（macOS / Linux の場合）

もし macOS / Linux 環境に Java 実行環境がなければ `SDKMAN`
を利用することで、ターミナルから簡単に導入することができます。

> SDKMAN! is a tool for managing parallel versions of multiple Software Development Kits on most Unix based systems.
>
> https://sdkman.io/
>
> — SDKMAN

手順. SDKMAN を用いた Java の導入

```
$ curl -s "https://get.sdkman.io" | bash ❶
$ source "$HOME/.sdkman/bin/sdkman-init.sh" ❷
$ sdk list java ❸
================================================================================
Available Java Versions
================================================================================
     12.ea.20-open
     11.0.1-zulu
     11.0.1-open
     10.0.2-zulu
     10.0.2-open
     9.0.7-zulu
     9.0.4-open
     8.0.192-zulu ❹
     8.0.191-oracle
     7.0.181-zulu
     1.0.0-rc9-graal
     1.0.0-rc8-graal
     1.0.0-rc7-graal
$ sdk install java 8.0.192-zulu ❹
```

❶ SDKMAN を導入します。

❷ SDKMAN を環境に設定します。

❸ 導入できる Java のバージョンを一覧します。

❹ 8.0 系の最新バージョンを指定して Java を導入します。

また、Gradle は `JAVA_HOME` 環境変数に実行環境の Java
のパスが設定されていることを期待していますので、`.bash_profile` で次のように `JAVA_HOME` を設定します。

手順. JAVA_HOME の設定

```
$ vi ~/.bash_profile ❶
export JAVA_HOME=~/.sdkman/candidates/java/current ❷
$ source ~/.bash_profile ❸
```

❶ vi エディタで `.bash_profile` を開きます。

❷ 本ラインをファイルの最下部に追加し vi を保存終了します。

❸ 設定を適用します。

これで準備完了です。

SDKMAN について

SDKMAN は主に Java
エコシステムの開発環境をコマンドラインから簡単に導入・設定するためにつくたれた管理ソフトウェアです。

たとえば簡単に各種 Java のバージョンを導入し切り替えることができます。

手順. SDKMAN による Java のバージョン切り替え

```
$ sdk install java 11.0.1-open ❶
$ sdk default java 11.0.1-open ❷
$ sdk default java 8.0.192-zulu ❸
```

❶ Java 11 を導入

❷ Java 11 をデフォルトに設定

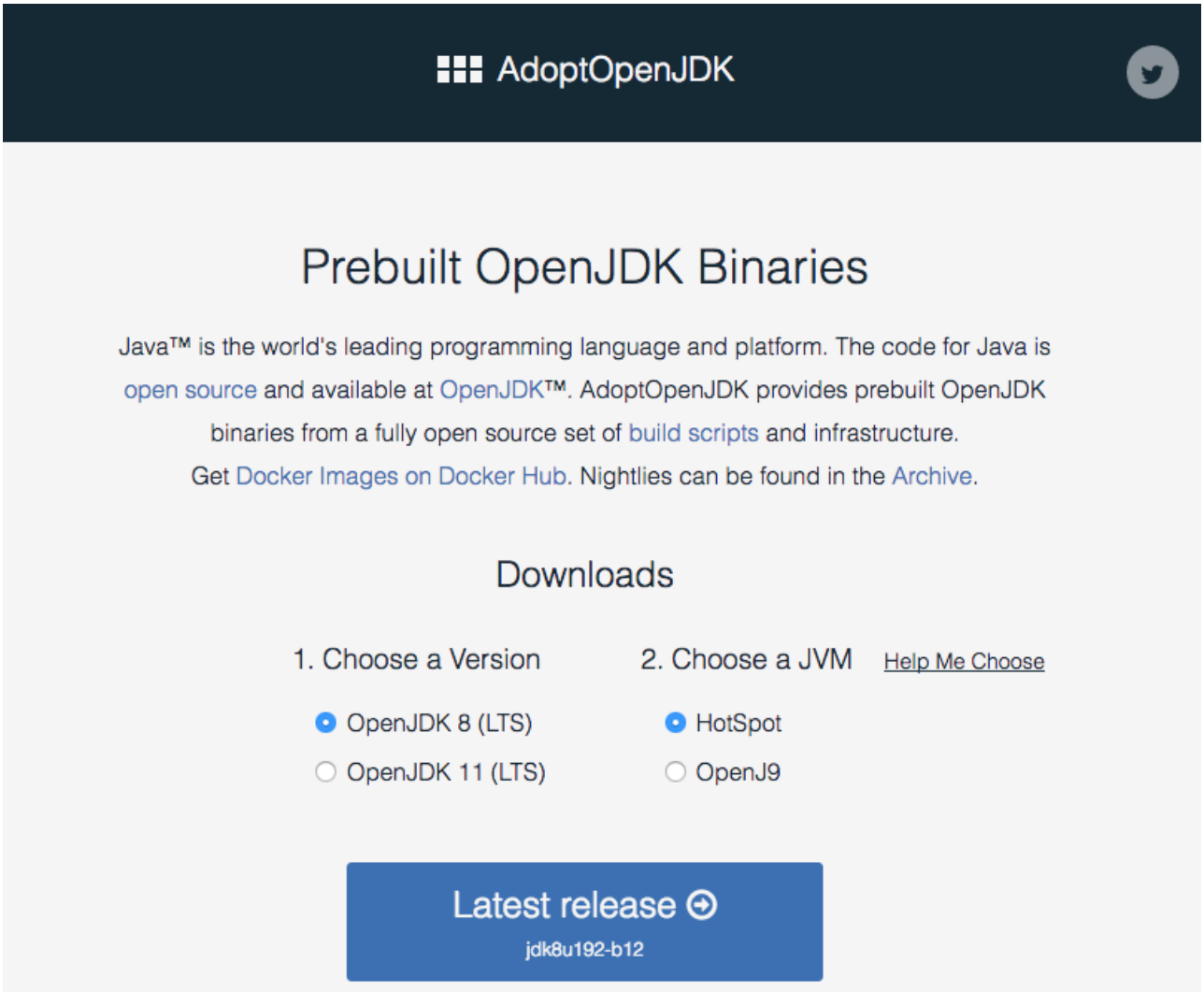
❸ Java 8 をデフォルトに設定

2.2. Java 実行環境の導入（Windows の場合）

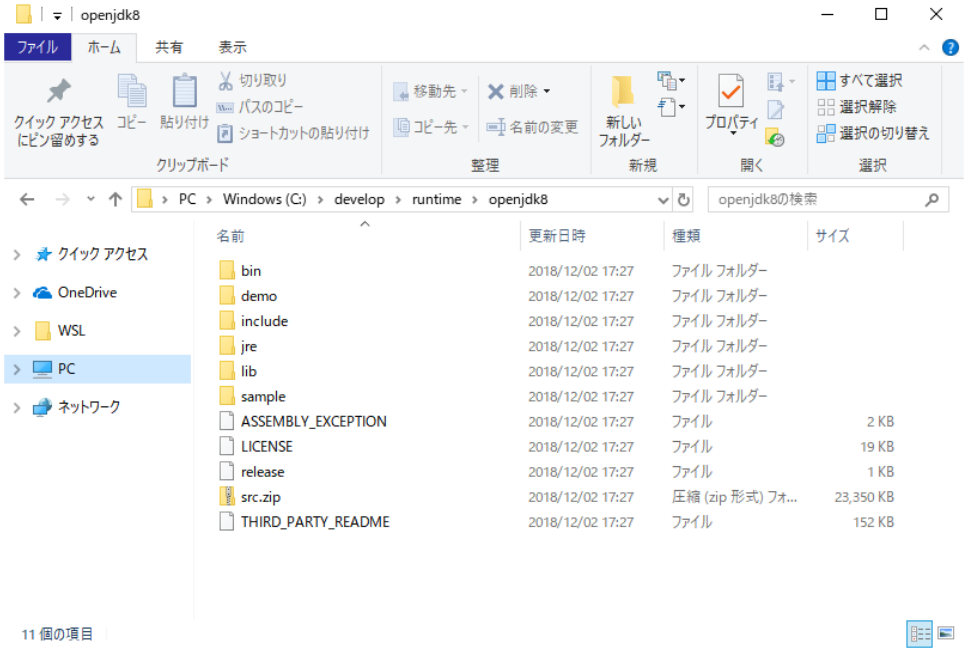
もし Windows 環境に Java 実行環境がなければ AdoptOpenJDK プロジェクトが提供する OpenJDK のバイナリを導入すると良いでしょう。

> Java™ is the world's leading programming language and platform. The code for Java is open source and available at OpenJDK™. AdoptOpenJDK provides prebuilt OpenJDK binaries from a fully open source set of build scripts and infrastructure.
>
> https://adoptopenjdk.net
>
> — AdoptOpenJDK

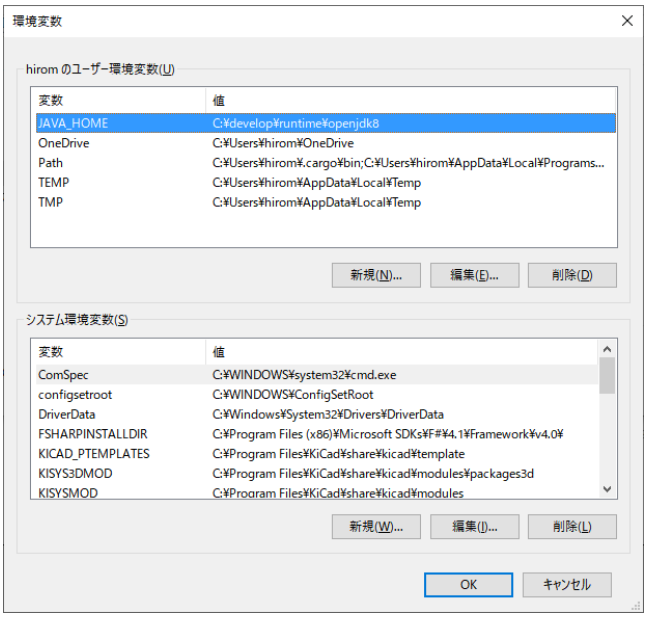
`https://adoptopenjdk.net` サイトにブラウザでアクセスし、OpenJDK 8 (LTS) - HotSpot を選択した後、zip ファイルをダウンロードしてください。



zip ファイルを任意の場所に展開します。ここでは `C:\develop\runtime\openjdk8` に展開したとします。



Gradle は `JAVA_HOME` 環境変数に実行環境の Java のパスが設定されていることを期待していますので、**エクスプローラー** › **PC（右クリック）** › **プロパティー** › **詳細設定** › **環境設定** からユーザー環境変数に `JAVA_HOME` を追加し、先ほど .zip を展開したパス（`C:\develop\runtime\openjdk8` ）を設定します。



Gradle は JAVA_HOME 環境変数を元に Java の実行環境を探すため、`java` コマンドを使うための PATH 環境変数は設定しなくてもかまいません。

これで準備完了です。

3. CI/CDツールの設定指針

3.1. サンプル文書の変換を試す

環境の準備ができましたので Asciidoc 文書を HTML/PDF に変換してみます。

変換に使うスクリプトは github のリポジトリに公開されており、リポジトリには HTML/PDF
変換に使うファイル一式と、文書サンプルとして "この文書" の Asciidoc
ファイルが置かれています。まずはサンプル文書が正しく変換できるかを試してみましょう。

macOS / Linux の場合は次のようにします。

手順. PDF 変換ビルドスクリプトの取得と実行

```
$ curl -L -O https://github.com/h1romas4/asciidoctor-gradle-template/archive/master.zip ❶
$ unzip master.zip ❷
$ cd asciidoctor-gradle-template-master ❸
$ ./gradlew docs ❹
BUILD SUCCESSFUL in 19s ❺
2 actionable tasks: 2 executed
```

❶ リポジトリのファイルをダウンロードします。

❷ ダウンロードした .zip ファイルを展開します。

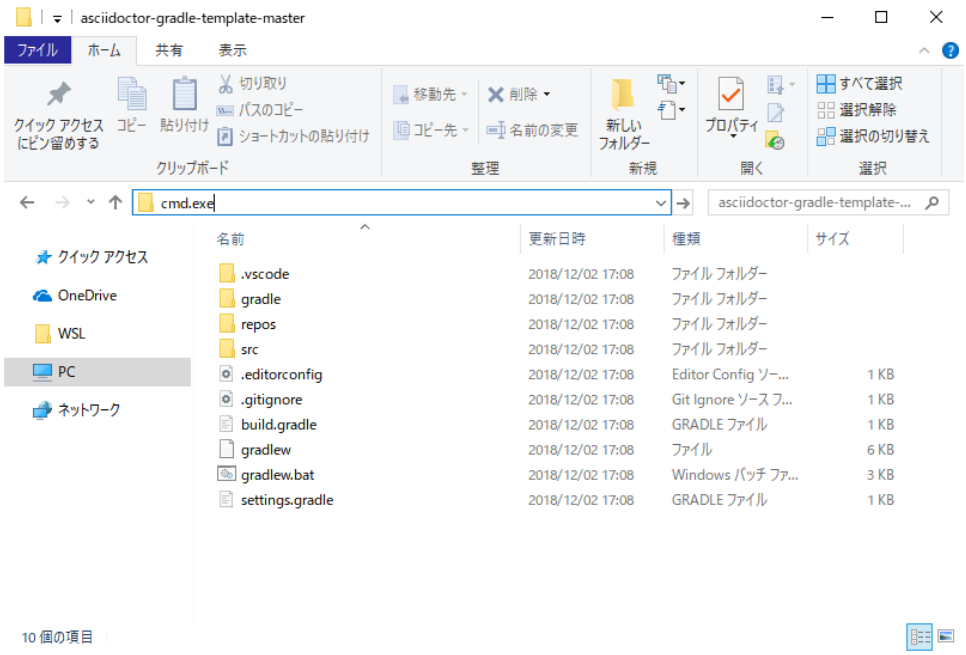
❸ カレントディレクトリを展開したフォルダの中に移します。

❹ Gradle
のビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回は数秒で完了します。

❺ `BUILD SUCCESSFUL` が出力されればビルド成功です。

Windows をお使いの場合は同等の操作をブラウザとエクスプローラーを使って行います。

1. ブラウザを使って `https://github.com/h1romas4/asciidoctor-gradle-template/archive/master.zip` にアクセスしリポジトリのファイルを取得します。
2. ダウンロードした .zip ファイルを右クリックし展開します。
3. 展開したフォルダ内をエクスプローラーで表示した上で、アドレスバーに `cmd.exe` と入力し、このフォルダをカレントディレクトリとしてコマンドプロンプトを起動します。



4. `.\gradlew.bat docs` と入力し Gradle ビルドを実行します。初回実行時はビルドに必要なファイルをダウンロードするため少し時間がかかります。次回は数秒で完了します。
5. `BUILD SUCCESSFUL` が出力されればビルド成功です。

プロキシーサーバーの設定

もしお使いのコンピューターがプロキシーサーバー経由のインターネットアクセスを行う場合は、次のコマンドを `./gradlew docs`
をする前に入力してください。インターネットを使ったライブラリの取得が正しく行われるようになります。ホスト名（`example.com`） と ポート番号（`8080`）部分はそれぞれの環境に合わせてください。

手順. プロキシーサーバー設定（Windows）

```
set JAVA_OPTS=-DproxyHost=example.com -DproxyPort=8080
```

手順. プロキシーサーバー設定（macOS / Linux）

```
export JAVA_OPTS=-DproxyHost=example.com -DproxyPort=8080
```

Asciidoc から変換された文書は次の場所に格納されます。`docs`
フォルダはビルド出力専用となっており、ビルド時にいったん全てのファイルが削除されますのでユーザーファイルは配置しないように注意してください。

```
docs/index.html
docs/index.pdf
```

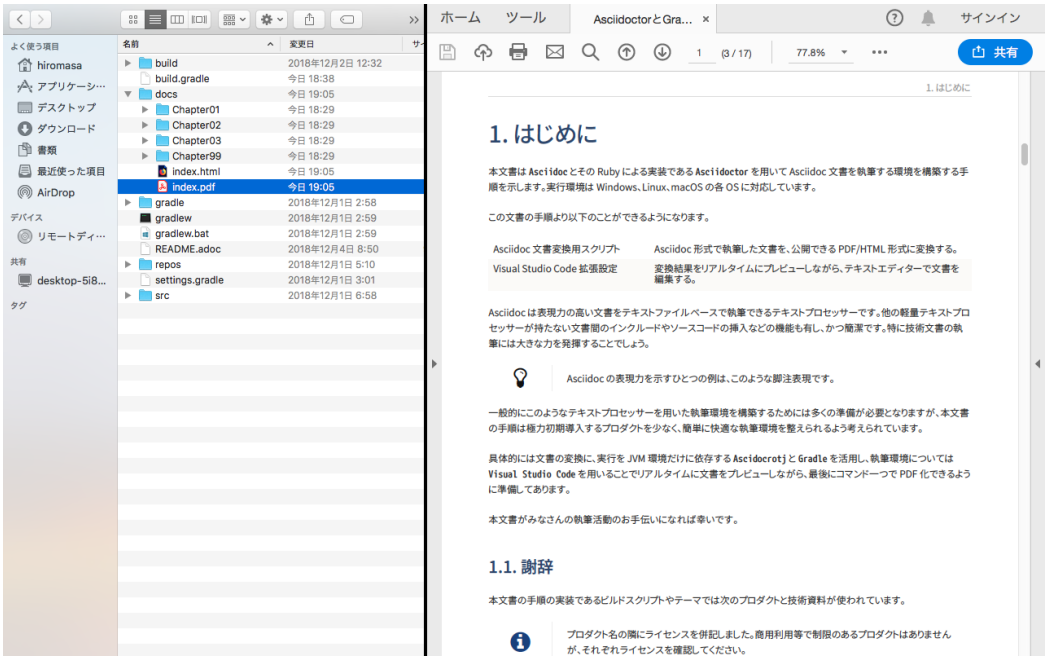


図 1. PDF 文書

図 2. HTML 文書

文書を github 上に公開する場合は、プロジェクトのファイルを全てコミットし、GitHub Pages に `docs`
フォルダを指定することで継続的に文書をパブリッシュすることができます。

3.2. テキストエディタで Asciidoc 文書を編集する

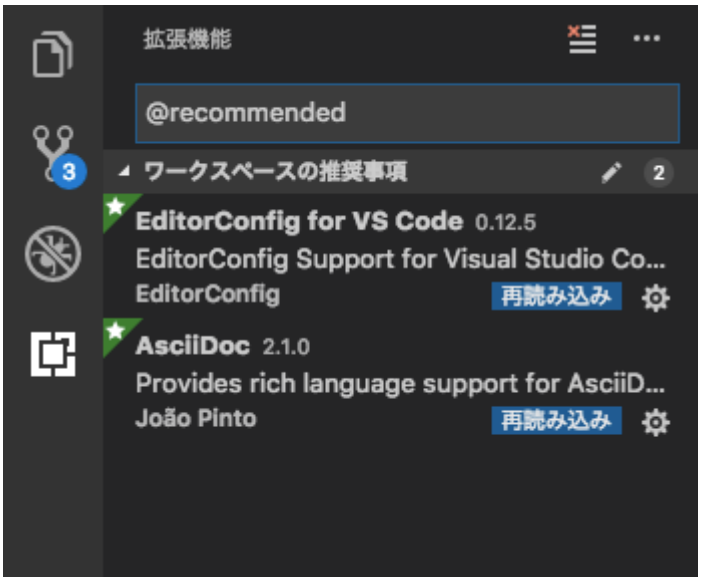
プロジェクトに配置された Asciidoc 文書は Visual Studio Code
を利用してリアルタイムに変換結果をプレビューしながら編集できるように準備されています。

プロジェクトフォルダ（`asciidoctor-gradle-template-master`）を Visual Studio Code
で開き、拡張機能の推奨事項から拡張を導入します。

拡張機能の推奨は `.vscode/extention.json`
で設定されています。文書に応じて設定し、執筆メンバーの環境を揃えることができます
。

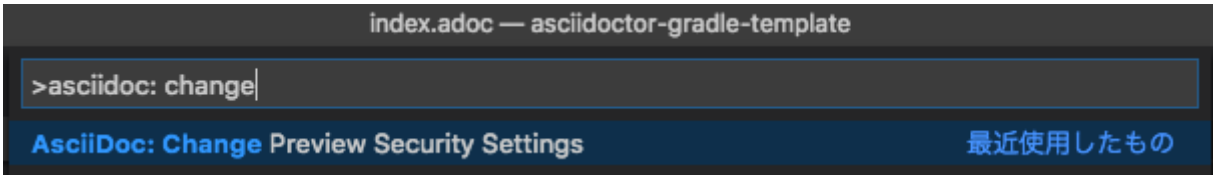
1. **[すべてインストール]** ボタンをクリックします。

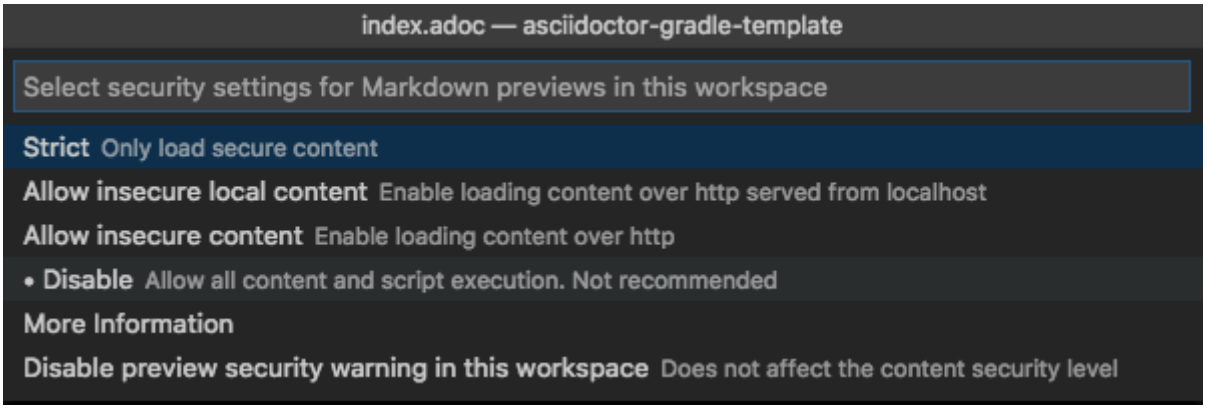
2. **[再読み込み]** ボタンをクリックします。



3. Asciidoc 文書（`src/docs/asciidoc/index.adoc`など）を Visual Studio Code のエクスプローラーから選択して開き、文書を開いたエディタ部右上に配置された **[Open Preview to the Side]**アイコンをクリックすると、画面右側に Asciidoc 文書の変換がリアルタイムに確認できるプレビューが表示されます。

なお、もしリアルタイムプレビューでソースコードのハイライトが動作しない場合は、Asciidoc
文書を開いた状態で [F1] を押し、**Change Preview Security Setting** › **Disable** を選択してください。





本設定は、プレビュー画面で動作する Webview
がインターネット上の外部リソースを評価できるようにセキュリティー権限を下げる設定です。

プロジェクト（ファイルが配置されたフォルダ）毎の設定となりますが、この設定をした後はこのプロジェクトで第三者の作成した未知の Asciidoc
文書を開かないようにしてください。外部に配置された悪意の JavaScript
により、セキュリティーを侵害される可能性があります。理解の上、設定してください。

拡張がソースコードのハイライトを行うため、CDN
に配置されたリソースを参照しているために発生している問題で、将来ローカルのリソースを参照するように修正され解決されると思われます。

3.3. 文書のファイル構成

文書は次のようなファイルで構成します。サンプル文書から執筆したい文書に合わせてカスタマイズしていくとよいでしょう。

文書を構成するファイル

```
src/docs/asciidoc/index.adoc ❶
src/docs/asciidoc/attribute.adoc ❷
src/docs/asciidoc/@style/*.css ❸
src/docs/asciidoc/@style/pdf-theme.yml ❹
src/docs/asciidoc/@font/* ❺
src/docs/asciidoc/Chapter{number}/index.adoc ❻
```

❶ 文書を作成する起点となる Asciidoc 文書です。大きな文書の場合はここから他の Asciidoc 文書を `include` して構成していきます。

❷ 文書設定をするためのファイルです。各文書から `incliude` します。

❸ HTML 出力とプレビュー用のスタイルシートです。文書に合わせて修正することができます。

❹ PDF
文書に変換する際に使われるスタイル定義です。文書に合わせて修正することができます。Asciidoctor PDF Theming Guide からドキュメントが参照できます。

❺ PDF 文書に埋め込みされるフォントファイルです。`pdf-theme.yml` から参照されています。TrueType フォント `.ttf` が指定できます。

❻ `src/docs/asciidoc/index.adoc` から参照される子文書です。後述の `build.gradle` による画像パス解決のためフォルダ名は `Chapter{number}` とします。

変換スクリプトとなる `build.gradle` の `attribute`
設定でこれらのファイルパスや変換に必要な属性を設定しています。

build.gradle

```
asciidoctor {
    asciidoctorj {
        attributes 'stylesdir': '@style',
            'stylesheet': 'asciidoctor.css'
    }
}

asciidoctorPdf {
    asciidoctorj {
        attributes 'pdf-stylesdir': "@style",
            'pdf-style': 'pdf'
    }
}
```

`src/docs/asciidoc/attribute.adoc`
ではエディタのリアルタイムプレビュー用の属性定義を行い、文書変換スクリプトでその値を上書きするように構成すると良いでしょう。

Asciidoctor で利用可能な属性は次から参照できます。

> Attributes are one of the features that sets Asciidoctor apart from other lightweight markup languages. Attributes can activate features (behaviors, styles, integrations, etc) or hold replacement (i.e., variable) content.
>
> https://asciidoctor.org/docs/user-manual/#attributes
>
> — asciidoctor.org

また、文書に挿入する画像ファイルは `images`
というフォルダ名内に配置され、その文書からの相対パスでリンクされることを期待しています。設定は次の部分で変更可能です。

src/docs/asciidoc/attribute.adoc

```
ifndef::imagesdir[:imagesdir: ./images]
```

build.gradle

```
copy {
    from 'src/docs/asciidoc/'
    include 'Chapter*/images/*' // 子文書のフォルダ名
    into 'docs'
}
```

3.3.1. 文書中の相互参照

文書内の参照リンクは次のように指定できます。

```
[[project-structure]] ❶
=== 文書のファイル構成

[[introduction]] ❷
== はじめに

<<project-structure,章の冒頭>> ❶
<<Chapter01/index.adoc#introduction,はじめに>> ❷
```

❶ 同一 `.adoc` 内での内部参照

❷ `.adoc` をまたいだドキュメント間参照

リンクの例

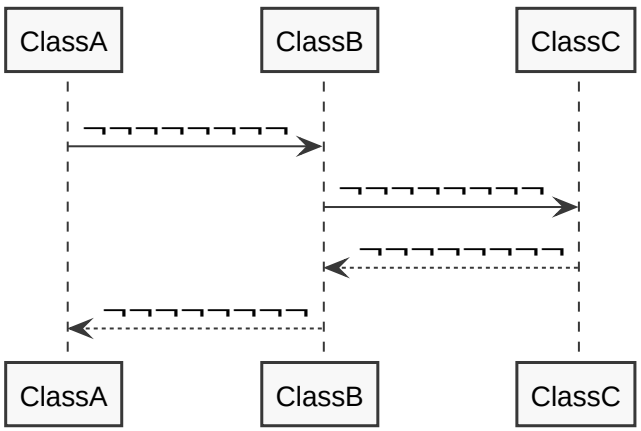
- 章の冒頭 を参照してください･･･
- はじめに で述べたように･･･

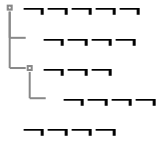
3.4. asciidoctorj-diagram

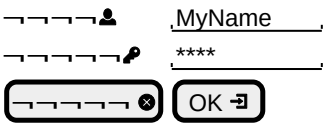
本変換ビルドスクリプトでは `asciidoctorj-diagram`
が有効になっており、いくつかのダイアグラム表記を使うことができます。

3.4.1. PlantUML

PlantUML 形式のダイアグラムを `.svg` ベクター画像で文書に埋め込みます。







That’s all. Happy coding!