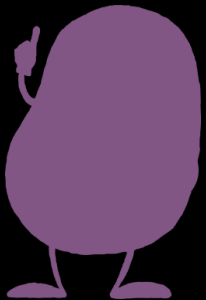


AWSとGitHubを使ってみよう勉強会

～ 第1回 KickOff ～

株式会社 豆蔵
ビジネスソリューション事業部



AWSとGitHubを使ってみよう勉強の概要

- 目的

- クラウドやAWSとか良く話にでてくるけど、それってオンプレと何が違うか？どんな感じで使うか？そして何がそんなに嬉しいのかを体感する

- 目標

- AWSやGitHubがこれでバッチリ！というところは目指さず、AWSやGitHubとかいってもそんなに難しくないのね！結構便利ね！機会があったら今度は自分でなにか試してみようかな！とクラウドやAWSへの心理的ハードルが下がるところ辺りを目指す

今回の勉強会の特徴

- 一方的に説明する講義スタイルではなく自習を中心とした反転学習形式
 - 次回までに各自が実施してくるお題やポイントをまずは解説
 - 持ち帰り各自でお題を実施。分からないことはネットで自分で調べる。それでもわからない場合は質問するか次回の勉強会で聞いてみる
 - 課題を出した次回は課題の解説とと皆さんからの質疑やディスカッションを行う
 - 勉強会は上記を繰り返し行う。なお、**お題を実施できなくても参加は可**。ただし解説は自分で手を動かしていることを前提にするので、その点は理解の上で参加のこと
- 勉強会の内容は極々簡単なRESTアプリを作成→コンテナ化→AWS EC2(オンプレ)へ→ECS Fargate(サーバーレス)へと**同じアプリをステップアップしながら別の実行環境で動かしていく**
 - GitHubは開発の中で使ってみて**Gitやクラウド上のCI/CD**がどのようなものかを実際に**体験**してみる

勉強会実施内容

- **1回目：キックオフ**

- キックオフと次回のお題の説明
- 次回までのお題：Helidonを使った簡単なRESTアプリ(Mavenでビルド&テストしてJavaコマンドで実行できるまで) & 開発にGitとGitHub Codespacesを使ってみる

- **2回目：GitHub CodespacesとHelidonの利用**

- 前回の課題の解説：GitHub CodespacesとHelidonについて
- 次回までのお題：コミットしたHelidonのサンプルアプリをGitHub Actionsでビルド～テスト～デプロイしてみる

- **3回目：GitHub Actionsを使ったCI環境の構築**

- 前回のお題の解説：GitHub ActionsとGitHub Packagesについて
- 次回までのお題：GitHub Actionsでビルド～テストしたものをDockerビルドしてGitHubのコンテナレジストリにデプロイ

- **4回目：GitHub Actionsを使ったCD環境の構築**

- 前回のお題の解説：コンテナイメージのビルドとGitHub Packages Container Registryについて
- 次回までのお題：AWS EC2にDockerをインストールした環境を構築し、GitHubにデプロイしたイメージを動作させる

- **5回目：AWS EC2環境の構築**

- 前回のお題の解説：AWSのユーザと権限 & AWSのネットワークの概要
- 次回までのお題：ECS Fargateを使ってGitHubにデプロイしたイメージを動作させる

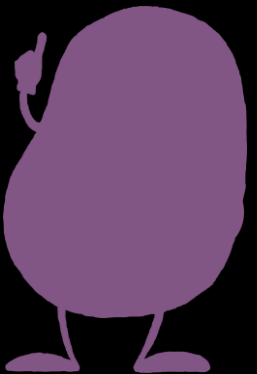
- **最終回：AWS ECS Fargate環境の構築**

- 前回のお題の解説：環境変数を使ったアプリの動作の変更とECS Fargateとは

勉強会参加に必要なもの

- GitHubおよびAWSの個人アカウント（今回は個人アカウントを使っていただきます。持っていないければ作成してもらいます）
 - GitHubは個人利用のため、すべて無料で使えます
 - AWSのアカウント登録にはクレジットカードが必要です。使わない場合はサービスを停止することで月1000円未満程度の利用を想定しています
 - なお、AWSはアカウントが乗っ取られた場合や誤って高額なサービスを利用した場合、高額な料金が請求される可能性がありますので、その点は注意ください。勉強会でそうならないための必要事項は説明します。それを守っていただければ心配することはないです
- 開発環境にはブラウザで利用可能なGitHub Codespacesを利用します
 - 個人PCにインストールしていただくものではありません

次回までの課題の説明



次回までの課題

- テーマ
 - 次回以降に使うサンプルアプリの作成
 - GitHubのCodespacesとgit repositoryに慣れる
- お題
 - Helidonを使った簡単なRESTアプリ(Mavenでビルド&テストしてJavaコマンドで実行できるまで)
 - 作ったアプリを自分のGitHubアカウントのリポジトリにコミットする
- ゴール
 - 予め準備されたテストケースがパスすること
 - GitHubにプロジェクトがコミットされていること

課題の実施手順

Step1. GitHubアカウントを作成する（持っていない場合）

Step2. リポジトリの作成&ひな形プロジェクトのimport

Step3. Codespacesを開始する

Step4. コードを修正してテストをパスさせる

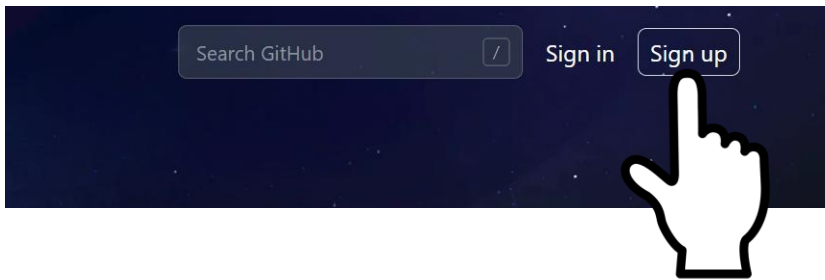
Step5. 修正したコードをコミットする

画面キャプチャやコマンド操作等はGitHubのssi-mz-studygroupユーザで行った例となります。キャプチャやコマンド等の該当部分は自分のユーザIDに読み替えてください

- ssi:simple_server_infra
- mz:mamezou

Step1. GitHubアカウントを作成する（持っていない場合） 1/4

- 個人のGitHubアカウントを作成する



<https://github.com>から開始

A screenshot of the GitHub account creation form. The form is titled 'Welcome to GitHub! Let's begin the adventure'. It contains several input fields: 'Enter your email*' (with a checkmark and 'ssimzstudy@'), 'Create a password*' (with a checkmark and '.....'), 'Enter a username*' (with a checkmark and 'ssi-mz-studygroup'), and 'Would you like to receive product updates and announcements via email?' (with a checkmark and 'n'). Below these is a section 'Verify your account' with a grid of six images. A Japanese text overlay on the right side of the form reads '必要なアカウント情報を入力' (Enter required account information).

Welcome to GitHub!
Let's begin the adventure

Enter your email*
✓ ssimzstudy@

Create a password*
✓

Enter a username*
✓ ssi-mz-studygroup

Would you like to receive product updates and announcements via email?
Type "y" for yes or "n" for no
✓ n

Verify your account

同一の物体が2つ表示されているマスを1つ選んでください。

必要なアカウント情報を入力

A screenshot of the GitHub verification code entry screen. It says 'You're almost done! We sent a launch code to ssimzstudy@'. Below this is a prompt 'Enter code*' followed by a row of eight empty boxes for entering the code.

You're almost done!
We sent a launch code to ssimzstudy@

→ Enter code*

登録したアドレスにメールがくるのでそこに載っている認証コードを入力する

Step1. GitHubアカウントを作成する（持っていない場合） 2/4

単なるアンケート

How many team members will be working with you?

This will help us guide you to the tools that are best suited for your projects.

Just me 2 - 5 5 - 10

10 - 20 20 - 50 50+

Are you a student or teacher?

Student Teacher

Continue

単なるアンケート

What specific features are you interested in using?

Select all that apply so we can point you to the right GitHub plan.

- ☐ Collaborative coding
Codespaces, Pull requests, Notifications, Code review, Code review assignments, Code owners, Draft pull requests, Protected branches, and more.
- ☒ Automation and CI/CD
Actions, Packages, APIs, GitHub Pages, GitHub Marketplace, Webhooks, Hosted runners, Self-hosted runners, Secrets management, and more.
- ☐ Security
Private repos, 2FA, Required reviews, Required status checks, Code scanning, Secret scanning, Dependency graph, Dependabot alerts, and more.
- ☐ Client Apps
GitHub Mobile, GitHub CLI, and GitHub Desktop.
- ☐ Project Management
Projects, Labels, Milestones, Issues, Unified Contribution Graph, Org activity graph, Org dependency insights, Repo insights, Wikis, and GitHub Insights.
- ☐ Team Administration
Organizations, Invitations, Team sync, Custom roles, Domain verification, Audit Log API, Repo creation restriction, and Notification restriction.
- ☐ Community
GitHub Marketplace, GitHub Sponsors, GitHub Skills, and Electron.

Continue

Where teams collaborate and ship.

Unlock advanced features with GitHub Team or continue with a free plan for the basics.

Free

- ① Unlimited public/private repositories
- ① 2,000 CI/CD minutes/month
Free for public repositories
- ① 500MB of Packages storage
Free for public repositories
- ① 120 core-hours of Codespaces compute
- ① 15GB of Codespaces storage
- ① Community support

Continue for free

Team Recommended for you

← Everything included in Free, plus...

- ① Protect your branches
Ensure that collaborators on your repository cannot make irrevocable changes to branches.
- ① Multiple pull requests reviewers
Add more control with multiple pull requests reviewers for any changes to your code before it can be merged.
- ① Code owners
Define who owns the code and who is notified for reviews for pull requests.
- ① Draft pull requests
- ① Required reviewers
- ① Pages and Wikis
- ① Environment deployment branches and secrets
- ① 3,000 CI/CD minutes/month
Free for public repositories
- ① 2GB of Packages storage
Free for public repositories
- ① Web-based support

Level up to GitHub Team for \$4 per user/month



絶対にこっち(Free)を選択

Step1. GitHubアカウントを作成する（持っていない場合） 3/4

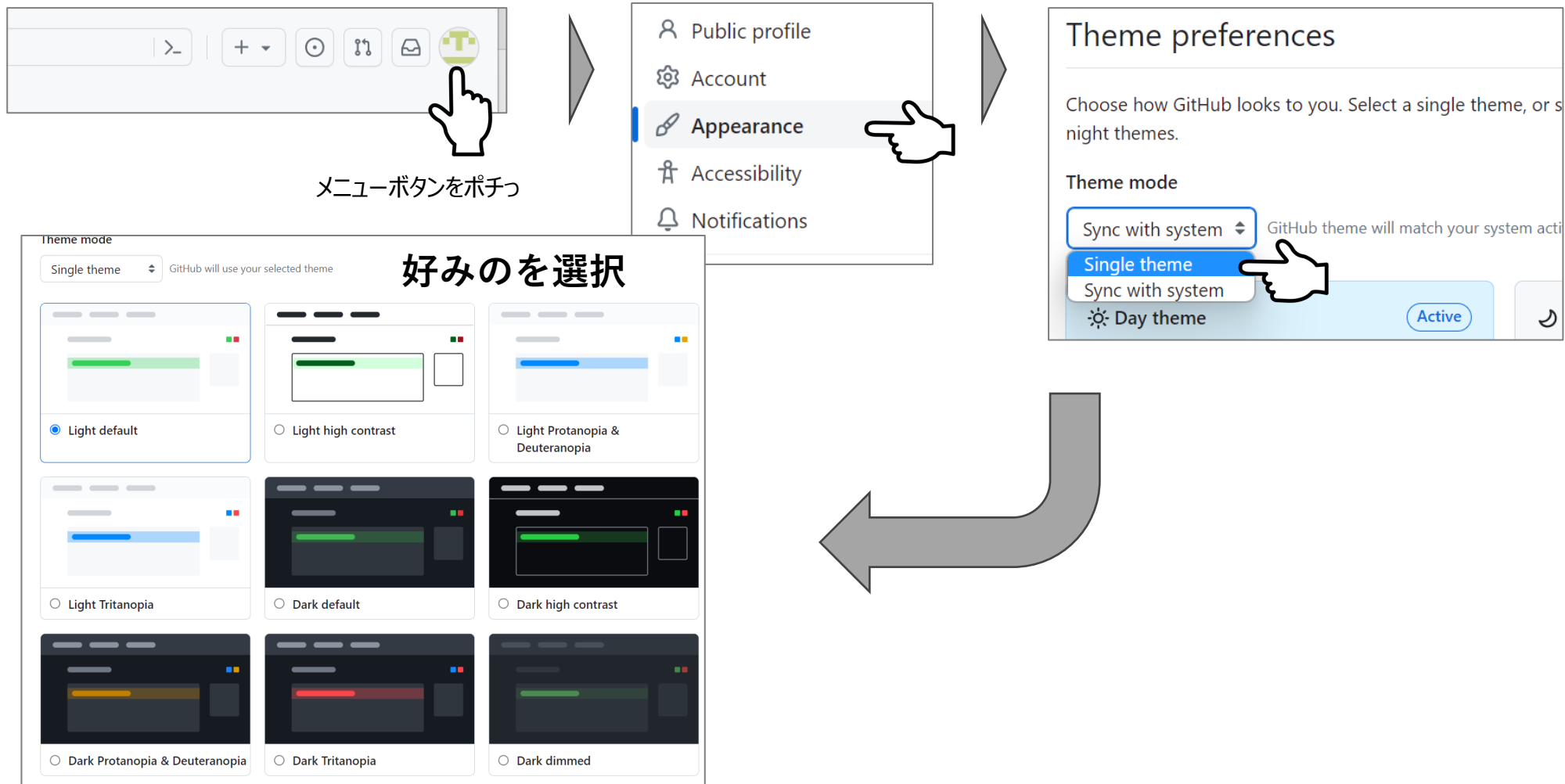


Step1. GitHubアカウントを作成する（持っていない場合） 4/4

- 個人のFreeプランでパブリックリポジトリを使えば、Codespacesを除きなにをいくら使ってもタダ
- 見たとおりクレジットカード登録はないので気が付いたらたくさん課金されていた！ということはないので安心
 - 反対にAWSはクレジットカード登録があるので要注意
- よって、利用量はCodespacesを除き全く気にする必要はない
- 気をつけるべきはセキュリティの1点のみ！
 - パブリックリポジトリを使うので、個人情報や機微情報は挙げないこと（例）電話番号やAWSのアクセスキーなど
 - 会社の資料やソースコードは絶対に挙げないこと

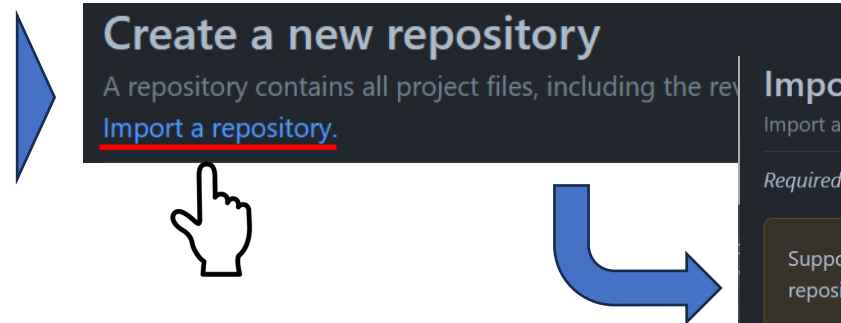
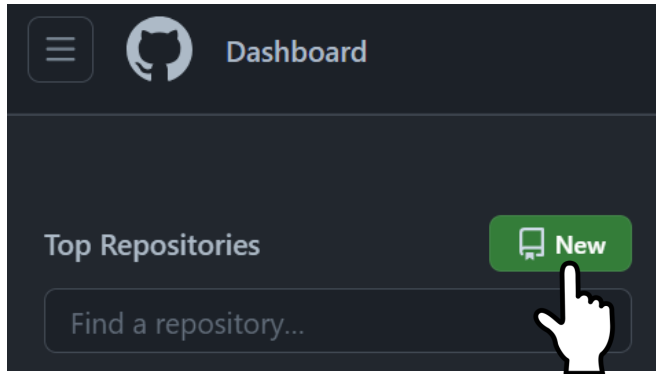
Step2. リポジトリの作成&ひな形プロジェクトのimport 1/3

- (本題とは関係ないが)ダークモードに変更する



Step2. リポジトリの作成&ひな形プロジェクトのimport 2/3

- welcomeページに移動(<https://github.com/>)



ひな形プロジェクトのURLを入力
<https://github.com/mamezou-tech/try-aws-github-learning>

A screenshot of the 'Import your project to GitHub' form. The title is 'Import your project to GitHub'. Below it, a subtitle says 'Import all the files, including revision history, from another version control system.' There's a note: 'Required fields are marked with an asterisk (*).'. A warning box states: 'Support for importing Mercurial, Subversion and Team Foundation Version Control (TFVC) repositories will end on October 17, 2023. For more details, see the [changelog](#).' The form has three main sections: 1. 'Your old repository's clone URL *' with a text input field containing 'https://github.com/mamezou-tech/try-aws-github-learning'. 2. 'Your new repository details' with two sub-sections: 'Owner *' with a dropdown menu showing 'ssi-mz-studygroup' and 'Repository name *' with a text input field containing 'my-sample-app'. Below the repository name field is a green checkmark and the text 'my-sample-app is available.' 3. 'Visibility' with two radio buttons: 'Public' (selected) and 'Private'. Below the 'Public' radio button is a red box. At the bottom, there's a note: 'You are creating a public repository in your personal account.' and two buttons: 'Cancel' and 'Begin import'. A hand cursor is pointing at the 'Begin import' button. The text '入力完了後' is written in the bottom right corner.

リポジトリ名（任意）を入力

publicを選択

この例ではssi-mz-studygroupユーザのmy-sample-appリポジトリにひな形プロジェクトをインポートしています。

入力完了後

Step2. リポジトリの作成&ひな形プロジェクトのimport 3/3

- import完了

<> Code

Issues

Pull requests

Projects

Wiki

Security

Importing complete!になったら
ココから移動

mamezou-tech/try-aws-github-learning

✓ Importing complete! Your new repository ssi-mz-studygroup ready.

これと同じファイル/ディレクトリが
importされていること



ssi-mz-studygroup / my-sample-app

入力したプロジェクト名

<> Code

Issues

Pull requests

Projects

Wiki

Security

Insights

Settings



my-sample-app

Public

Pin

Unwatch 1

Fork 0

Star 0

main

Go to file

Add file

<> Code

About

No description, website, or topics provided.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published
Create a new release



ssi-mz-studygroup add extentions

1 hour ago 4

.devcontainer

add extentions

1 hour ago

src

できたよー

1 hour ago

.gitignore

initial commit

2 hours ago

README.md

Initial commit

2 hours ago

java-formatter.xml

initial commit

2 hours ago

pom.xml

initial commit

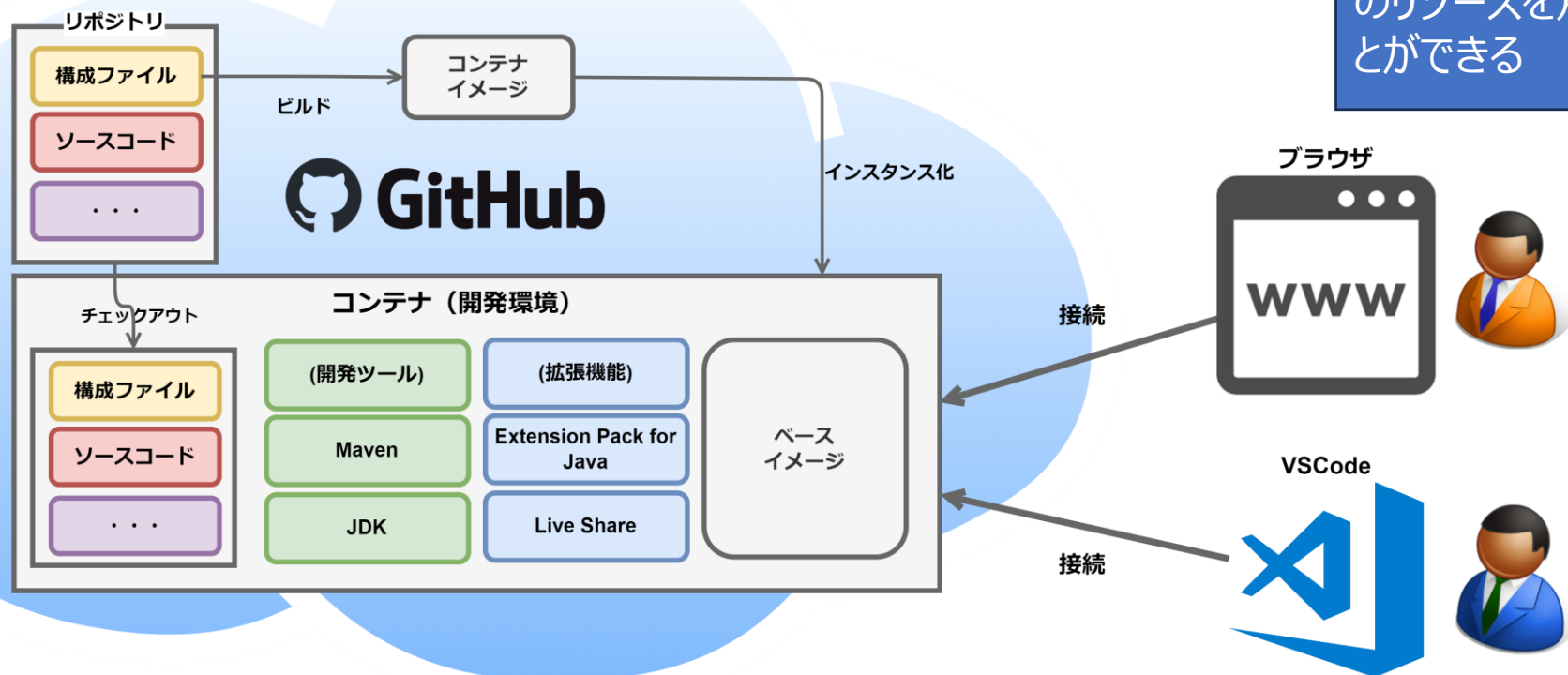
2 hours ago



Step3. Codespacesを開始する 1/3

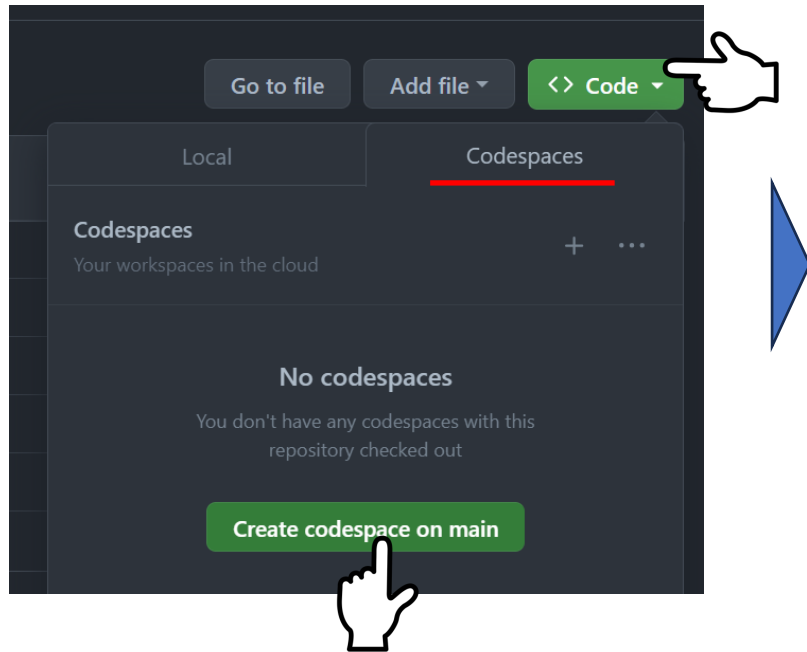
- GitHub Codespacesとは

個人アカウントでも2コアCPU/4GBメモリ
のリソースを月60時間まで無料で使う
ことができる

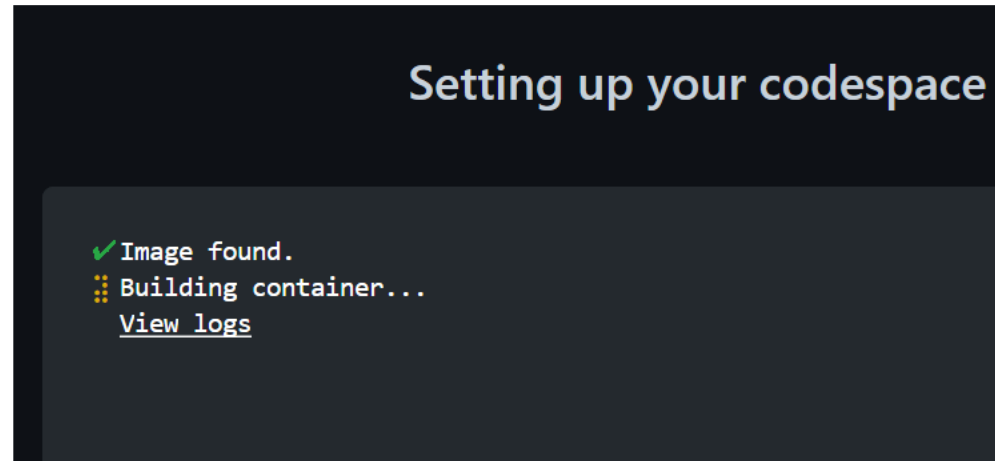


- 構成ファイルをもとにビルドされたコンテナイメージから生成されたコンテナインスタンスを個別の開発環境としてユーザーに提供する機能
- ユーザーはブラウザもしくはローカルのVSCodeから生成されたコンテナインスタンスに接続して作業を行う
- ユーザーは目の前にあるブラウザやVSCodeを操作するが、リポジトリからチェックアウトしたファイルやJavaのビルドや実行などはすべてGitHub側のコンテナ内に存在し行われます

Step3. Codespacesを開始する 2/3



開始！



数分待つと

Step3. Codespacesを開始する 3/3

画面が立ち上がって諸々拡張機能が有効になるまで少し時間が掛かる
JAVA PROJECTタブ⇒MAVENタブの順番でタブがでてくるので2つ出るまでしばらく待つ→出たら完了

他にも色々案内がでてくるがすべて何もせず閉じるOK

ボタンをクリックしてJavaプロジェクトとして認識させる

このリポジトリ 用のおすすめ拡張機能 'Extension Pack for Java' 拡張機能 提供元: Microsoft をインストールしますか?
インストール 推奨事項の表示

18

Step4. コードを修正してテストをパスさせる 1/2



The screenshot shows an IDE interface with a file explorer on the left and a code editor on the right.

File Explorer (Left):

- MY-SAMPLE-APP [CODESPACES]
 - .devcontainer
 - src
 - main
 - java / sample
 - HelloApplication.java
 - HelloApplicationMain.java
 - HelloResource.java** (selected with a hand icon)
 - resources
 - test
 - target
 - .gitignore
 - java-formatter.xml
 - pom.xml
 - README.md

Code Editor (Right):

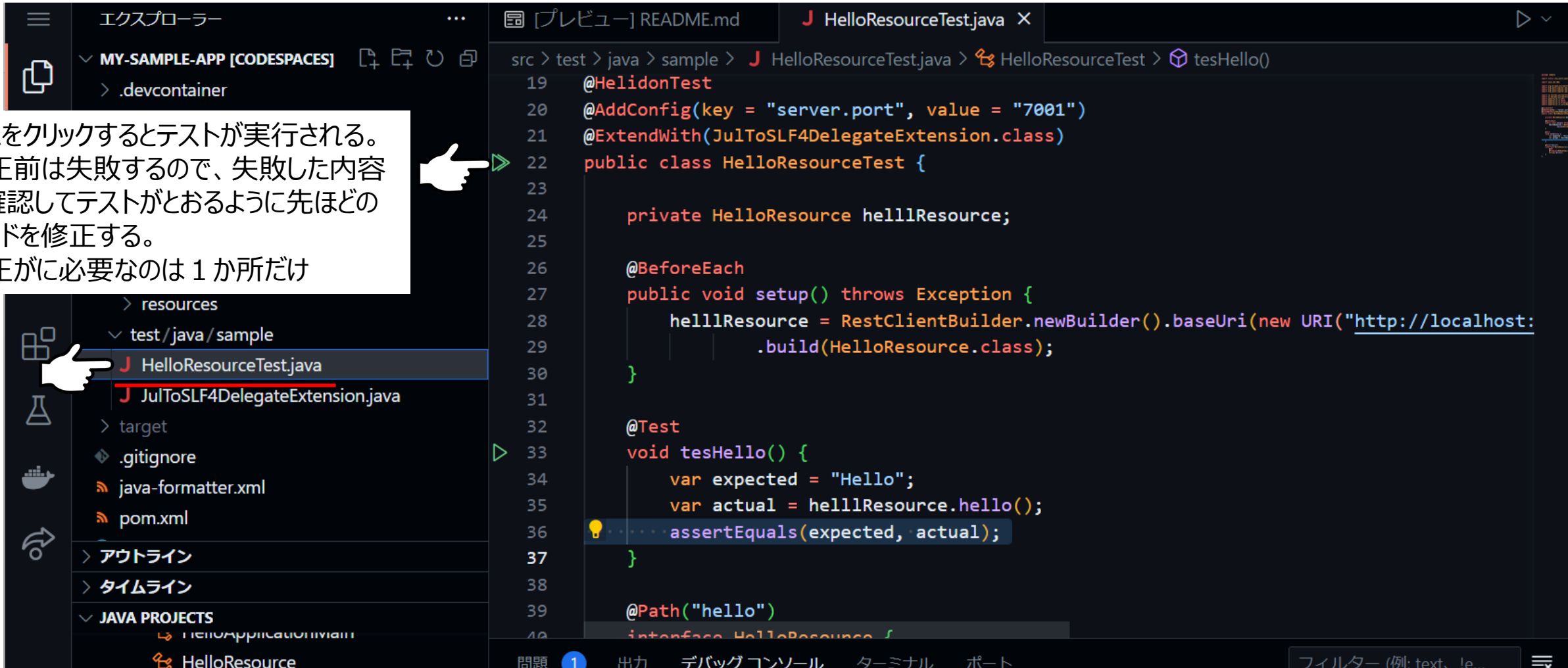
File: `src > main > java > sample > HelloResource.java > ...`

```
5 import jakarta.ws.rs.Path;
6 import jakarta.ws.rs.Produces;
7 import jakarta.ws.rs.core.MediaType;
8
9 @ApplicationScoped
10 @Path("hello")
11 public class HelloResource {
12     @GET
13     @Produces(MediaType.TEXT_PLAIN)
14     public String hello() {
15         return null; // TODO テストが通るように実装する
16     }
17 }
18
```

An orange arrow points to the `return null;` line, with the text `テストコードを見て修正` (Fix by looking at the test code) below it.

Step4. コードを修正してテストをパスさせる 2/2

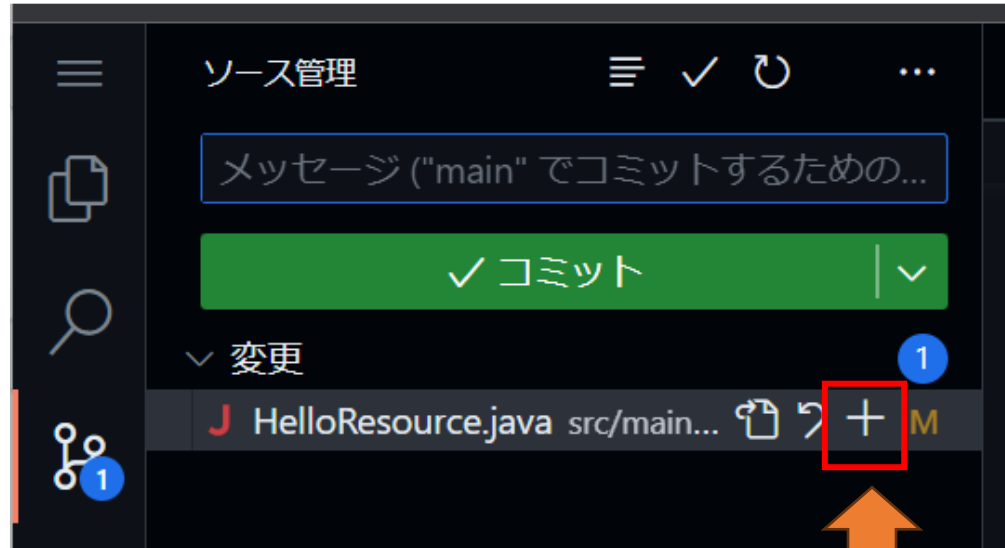
これをクリックするとテストが実行される。
修正前は失敗するので、失敗した内容
を確認してテストがとおるように先ほどの
コードを修正する。
修正がに必要なのは 1 か所だけ



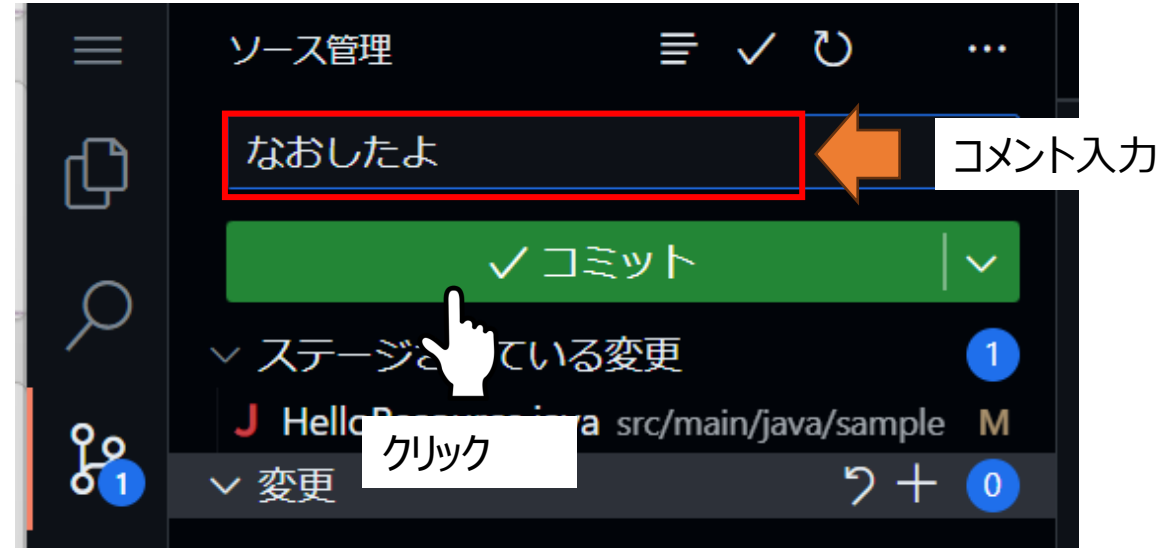
The screenshot shows an IDE with a dark theme. On the left, the 'EXPLORER' view shows a project structure with a file named 'HelloResourceTest.java' highlighted. A white hand icon points to this file. In the center, the 'PREVIEW' view shows the code of 'HelloResourceTest.java'. The code includes annotations like '@HelidonTest', '@AddConfig', and '@ExtendWith', and a test method 'tesHello()' that uses 'RestClientBuilder' and 'assertEquals'. A white hand icon points to the 'Run' button (a green play icon) in the top right of the code editor. At the bottom, the 'TERMINAL' view shows the output of the test execution, which includes a failure message.

```
src > test > java > sample > J HelloResourceTest.java > HelloResourceTest > tesHello()
19  @HelidonTest
20  @AddConfig(key = "server.port", value = "7001")
21  @ExtendWith(JulToSLF4DelegateExtension.class)
22  public class HelloResourceTest {
23
24      private HelloResource helllResource;
25
26      @BeforeEach
27      public void setup() throws Exception {
28          helllResource = RestClientBuilder.newBuilder().baseUri(new URI("http://localhost:
29              .build(HelloResource.class);
30      }
31
32      @Test
33      void tesHello() {
34          var expected = "Hello";
35          var actual = helllResource.hello();
36          assertEquals(expected, actual);
37      }
38
39      @Path("hello")
40      interface HelloResource {
```

Step5. 修正したコードをコミットする 1/1



コミット対象に追加する



<学習テーマ>

- リポジトリに反映されているか確認してみよう！
- コミットと変更の同期の違いを調べてみよう！
(ヒント: リモートとローカル)
- Git graphで変更を確認してみよう (インストール済み)

これでゴールです👏

- 誤って削除したりしても同じ手順でリポジトリは再作成できるので思い切っているいろいろやりましょう
- 人様のリポジトリを弄っても権限がないので、誤って削除はできません。なので思い切っているいろいろ弄ってみましょう
- GitHub/ Git / Codespaces(VSCode) はメジャーなツールなのでググればいろいろ出てきます。
- 分からないこと、疑問に思ったことはグーグル先生に聞いて調べてみましょう
- Codespacesを使い終わったら停止するようにしましょう。
GitHub CodespacesによるJavaのチーム開発環境の作り方 | 豆蔵デベロッパーサイト
⇒開発環境(コンテナ)のライフサイクル を参照

次回の予定

- 今回の課題の説明
 - 補足説明と質疑応答
- 次回までの課題の説明
 - テーマはGitHub Actionsを使ったCI/CD（ビルドからjarのPackageレジストリへのアップまで）