JBS 中級コース(案1)／Day3／2026-05-28

Day2の実装を Docker化してCI/CDで自動回転させる

マルチステージビルドの理解、docker-compose による複数サービス起動、GitHub Actions 全ジョブ緑、trivy セキュリティスキャン、@claude メンションでの AI 自動 PR まで、3時間で本番運用に近い CI/CD を実機で動かします。

13:00〜16:00(オンライン)　Docker + GitHub Actions　Vercel デプロイは発展課題

本日のアジェンダ

13:00〜13:10　Session 01 Day2復習と本日のゴール
動くアプリと CI の現状を確認、Day3 の到達点を握る。[10分]

13:10〜13:30　Session 02 Dockerfile の設計と動作確認
マルチステージ・非rootユーザ・健全な layer 順を理解し、実機で build。[20分]

13:30〜13:55　Session 03 docker-compose で複数サービス起動
app + DB の構成、healthcheck、ホットリロード設定の動作確認。[25分]

13:55〜14:15　Session 04 CI 全ジョブを緑のまま docker-build 追加
GitHub Actions の lint/typecheck/test/docker-build を全て緑に。[20分]

14:25〜14:50 **Session 05 trivy でセキュリティスキャン**

イメージ脆弱性検出を CI に組み込み、検出された脆弱性に対応。[25分]

14:50〜15:15 **Session 06 @claude メンションで AI 自動 PR**

Issue に @claude を書いて Claude Code が PR を起こす流れを体験。[25分]

15:15〜15:50 **Session 07 Vercel デプロイ(発展課題)とツール戦略**

自分の Vercel アカウントへ自動デプロイ、業務適用への落とし込み。[35分]

15:50〜16:00 **3日間のまとめと業務棚卸し**

持ち帰る3アクションと1ヶ月後のフォロー予告。[10分]

SESSION 01 ／ 13:00〜13:10

Day2復習と本日のゴール

Day2で完成したアプリと CI の現状を確認し、Day3で達成する状態を明確にします。

Day2までの到達点

- [x] FastAPI + SQLAlchemy 2.0で全エンドポイント実装済み
- [x] pytest 全件 PASS、カバレッジ80%以上
- [x] ruff / mypy 違反ゼロ
- [x] CI(4ジョブ)が緑、main にマージ済み

Day3の3つのゴール

- [x] マルチステージ Dockerfile を理解し、ローカルで build → run が動く
- [x] GitHub Actions の docker-build ジョブを含む全ジョブを緑に保てる
- [x] @claude メンションで Claude Code が自動 PR を起こす流れを1回完走する

本日の進め方

配布物の Dockerfile / docker-compose.yml を起点に拡張

Day1開始時点で雛形が配布されています。これを「読む」「動かす」「改善する」の3段階で進めます。ゼロから書く時間はありません。

Docker Desktop の起動確認

研修開始前に Docker Desktop が起動していることを確認してください。`docker --version` でバージョンが返ればOK。返らなければ Docker Desktop の起動か、WSL2の更新が必要です。

SESSION 02 ／ 13:10〜13:30

Dockerfile の設計と動作確認

20分で配布物の Dockerfile を読み解き、build → run までを実機で動かします。

HANDS-ON 配布 Dockerfile の構造理解と build

20分

1 配布 Dockerfile を読む

リポジトリの `Dockerfile` を開きます。builder ステージで依存導入、runtime ステージで非rootユーザ起動の二段構成です。読み解くポイントは下記4つ。

- `FROM python:3.11-slim AS builder` でビルド専用ステージ
- `pip install --prefix=/install` で site-packages を独立配置
- `FROM python:3.11-slim AS runtime` で実行ステージを切り分け
- `USER app` で UID 1000の非rootユーザに切り替え

2 Claude Code に Dockerfile レビューを依頼

カスタムプロンプトを呼び出して、配布 Dockerfile に対する改善余地を出させます。

PROMPT

```
/dockerfile-review

Dockerfile を上記観点でレビューしてください。
ファイルパス: Dockerfile
特に layer キャッシュの効き方と、runtime ステージのサイズ削減観点を厳しく見てください。
```

カスタムプロンプトは `.github/prompts/dockerfile-review.prompt.md` に定義されています。

3 ローカルで build → run

```
TERMINAL
# build
docker build -t jbs-helpdesk:dev .

# サイズ確認
docker images jbs-helpdesk:dev

# run (バックグラウンド)
docker run -d -p 8000:8000 --name helpdesk-local jbs-helpdesk:dev

# 動作確認
sleep 2
docker logs helpdesk-local
docker exec helpdesk-local python -c "import urllib.request; print(ur
llib.request.urlopen('http://localhost:8000/health').read())"

# 停止と削除
docker stop helpdesk-local && docker rm helpdesk-local
```

build 後のイメージサイズが300MB 以下に収まっているかを確認してください。slim ベースなので200MB 前後が目安です。

---

4 Claude Code に改善反映を依頼

Session 02で出た指摘の中から「致命的」と「中」だけを Dockerfile に反映させます。

```
PROMPT
レビュー指摘のうち下記を反映してください。
1. HEALTHCHECK の interval を30s から10s に短縮 (起動直後の検出を早める)
2. apt-get install を最小限にして runtime に残らないよう builder のみに移す
3. .dockerignore に tests/ docs/ api/ vercel.json を含める (既に対応済なら確認のみ)

反映後、再度 docker build して image サイズの差分を報告してください。
```

よくある詰まり

- `Cannot connect to the Docker daemon` → Docker Desktop が起動していない
- build が遅い → 初回は2〜3分、2回目以降はキャッシュで30秒程度に短縮される
- Mac M1/M2ユーザは `--platform linux/amd64` を build に付けると CI と同じアーキで検証可能

SESSION 03 ／ 13:30〜13:55

docker-compose で複数サービス起動

25分で app + postgres の2サービス構成を動かし、healthcheck と依存順序まで含めて理解します。

HANDS-ON

postgres 有効化と複数サービス起動

25分

1 docker-compose.yml の postgres セクション有効化

配布物では postgres サービスがコメントアウトされています。Claude Code に有効化を依頼します。

PROMPT

```
docker-compose.yml の postgres サービスのコメントアウトを外し、下記を追加してください。

1. postgres コンテナに healthcheck (pg_isready で内部監視)
2. app の depends_on で postgres の healthcheck 完了を待つ条件 (condition: service_healthy)
3. app の環境変数 DATABASE_URL を postgres 接続文字列に切り替え可能にする
4. ボリュームを永続化 (postgres_data)
5. ローカル開発時のみホットリロードできるよう volumes で app/ をマウント

実機で動かすため Docker イメージの port (5432) を localhost に publish するのは省略 (app 経由でしかアクセスしない)。
```

2 app 側を postgres 対応に

aiosqlite から asyncpg に切り替えるため、requirements.txt と app/config.py を更新します。

PROMPT

```
requirements.txt に asyncpg==0.30.0を追加し、
app/config.py の database_url を環境変数 DATABASE_URL 優先にしてください。
DATABASE_URL の例: postgresql+asyncpg://helpdesk:helpdesk@postgres:5432/helpdesk

scripts/seed.py が postgres でも動くよう、CREATE EXTENSION や明示的なテーブル作成順序を整備してください。
```

3 compose で起動

```
TERMINAL">DOCKER COMPOSE UP -D --BUILD
DOCKER COMPOSE PS
```

app と postgres の両方が healthy になることを確認してください。

4 動作確認

```
TERMINAL">DOCKER COMPOSE EXEC APP PYTHON SCRIPTS/SEED.PY
DOCKER COMPOSE EXEC POSTGRES PSQL -U HELPDESK -D HELPDESK -C "SELECT COUNT(*) FROM TICK
```

シードが入って health チェックが通れば、ローカルで本番想定の構成が動いています。

5 停止と片付け

```
TERMINAL">DOCKER COMPOSE DOWN -V
```

`-v` で永続ボリュームも削除します。SQL の状態をリセットしたいときも使います。

SESSION 04 ／ 13:55〜14:15

CI 全ジョブを緑のまま docker-build 追加

20分で GitHub Actions の docker-build ジョブを動作させ、4ジョブ(lint / typecheck / test / docker-build)全て緑の PR を作ります。

HANDS-ON **CI を実機で回す** 20分

1 配布 ci.yml の docker-build ジョブを確認

`.github/workflows/ci.yml` を開いて、docker-build ジョブの定義を読みます。`docker/setup-buildx-action` と `docker/build-push-action` でビルドキャッシュを GHA Cache に持たせています。

2 意図的に CI を落とす

動作を理解するために、わざと CI を落としてみます。`app/main.py` の末尾に `print("debug")` を追加して push してください。ruff の lint ジョブが落ちます。

3 Actions タブで失敗ログを確認

GitHub の Actions タブで、どのジョブが落ちたか確認します。失敗ジョブをクリックし、エラー出力をコピー。

4 Claude Code で修正

PROMPT

```
下記の CI エラーを直してください。
[エラーログをペースト]
```

Claude Code は `print` を `logging` に置換するか削除する提案を返します。反映して再 push。

5 **4ジョブ全緑を確認**

Actions タブで lint / typecheck / test / docker-build の4ジョブが全て緑になることを確認します。これが Day3の必須到達点です。

休憩(14:15〜14:25)

SESSION 05 ／ 14:25〜14:50

trivy でセキュリティスキャン

25分でビルド済みイメージに対する脆弱性スキャンを CI に組み込み、検出された脆弱性に対応します。

HANDS-ON trivy 統合

25分

1 ローカルで trivy 実行

```
TERMINAL">BREW INSTALL TRIVY  # 未インストールの方
DOCKER BUILD -T JBS-HELPDESK:SCAN .
```

初回は脆弱性 DB のダウンロードがあるので少し時間がかかります。HIGH / CRITICAL の脆弱性件数を確認してください。

2 CI に trivy ジョブを追加

PROMPT

```
.github/workflows/ci.yml の docker-build ジョブの直後に trivy-scan ジョブを追加してください。

- docker build した後にイメージを export
- aquasecurity/trivy-action を使用
- severity を CRITICAL,HIGH に限定
- exit-code を1にして CRITICAL があれば落とす
- HIGH は warn でレポートのみ (continue-on-error)

trivy ジョブは docker-build ジョブの完了に依存させます (needs: docker-build)。
```

3 検出された脆弱性に対応

HIGH / CRITICAL の検出があれば、Claude Code に対応策を聞きます。

PROMPT

```
trivy が下記の脆弱性を検出しました。対応策を提案してください。

[trivy出力]

base image を更新する、依存パッケージを上げる、運用回避策で済ますか、判断観点も含めて教えてください。
```

4 修正と CI 再実行

提案された修正を反映 → push → CI を緑にする、までやり切ります。trivy ジョブが緑になるか、HIGH のみで warn 表示の状態が許容ラインです。

SESSION 06 ／ 14:50〜15:15

@claude メンションで AI 自動 PR

25分で Issue → @claude メンション → Claude Code Action による自動 PR 生成を体験します。

HANDS-ON 非同期エージェントで PR を生成

25分

1 事前準備の確認

`.github/workflows/claude.yml` が配布されています。動かすには次の事前準備が必要です。

- リポジトリ admin が `claude /install-github-app` を実行
- Secrets に `ANTHROPIC_API_KEY` を登録
- workflow を main に push 済み

EMU 組織側で App インストールが社内承認待ちになっている場合は、講師実演を見る形でも構いません。

2 Issue を起票

GitHub の Issues タブから New Issue → feature_request テンプレを選び、下記の本文で起票します。

MARKDOWN

```
# [FEAT] チケット起票時に SLA 期限を自動計算してレスポンスに含める

## 背景
クライアント側でチケット起票時に SLA 期限を計算する処理が重複している。サーバー側で計算してレスポンスに含めたい。

## 期待する動作
POST /tickets のレスポンスに下記2フィールドを追加。

- response_deadline: 初回応答期限 (ISO 8601形式の UTC datetime)
- resolution_deadline: 解決期限 (同上)

## 受入基準
- [ ] レスポンスに上記2フィールドが含まれる
- [ ] SLAPolicy テーブルの値に従って計算される
- [ ] 営業時間外と土日は除外
- [ ] テスト追加 (正常系・境界値・営業時間外境界)

@claude このIssue の修正計画を立てて、ドラフトPRを作ってください。
```

3 Claude Code Action の起動を待つ

Issue 投稿後5〜10分で、Claude Code Action が次のいずれかの動作をします。

- 修正計画を Issue にコメント
- ブランチを切ってドラフト PR を作成
- 追加情報が必要な場合、Issue で質問

Actions タブで claude ジョブの実行ログを並行確認してください。

---

4 生成された PR をレビュー

ドラフト PR が上がったら、Files changed タブで差分を確認します。

- SLAPolicy の取得ロジックが正しいか
- 営業時間外と土日の除外計算が正確か
- レスポンススキーマに新フィールドが追加されているか
- テストが正常系・境界値・異常系を満たすか

気になる点は PR コメントに `@claude` を含めて返信すれば、Claude Code が追加修正をかけます。

---

5 CI が緑になったらマージ

4ジョブ全緑+trivy も緑になったら Squash and Merge。これで Issue 起票 → AI 実装 → CI 緑 → マージのフローが1周しました。

未有効化のとき

組織側で `claude.yml` が動かない場合は、ローカルの Claude Code でも同じ流れをシミュレートできます。`claude /init` でセッションを開始し、Issue 本文を貼って「このIssueを実装してPRを作って」と依頼してください。

SESSION 07 ／ 15:15〜15:50

Vercel デプロイ(発展課題)とツール戦略

35分で発展課題として Vercel 自動デプロイを試し、業務適用に向けたツール戦略を整理します。

FREE **Vercel 連携(発展課題)** 20分

1. **Vercel アカウント準備**

   https://vercel.com に GitHub ログインでアカウント作成。社内承認が必要な場合はスキップして、講師実演を見るだけでも構いません。

2. **プロジェクト連携**

   詳細は `docs/deploy_vercel.md` を参照。Vercel ダッシュボードから自分の Fork リポジトリを Import → デプロイ。

3. **動作確認**

   発行された URL (例: `https://<プロジェクト名>-<自分のアカウント>.vercel.app/api/health`) を curl で叩いて、 `{"status": "ok"}` が返ることを確認します。

4. **GitHub Actions 連携(さらに発展)**

   `.github/workflows/vercel.yml` が同梱されています。Secrets に VERCEL_TOKEN / ORG_ID / PROJECT_ID を登録すれば、main への push で自動デプロイが走ります。

本日扱ったツールの整理

ツール	役割	本研修での使い方
Claude Code	同期(Chat) / 非同期(GitHub Action) エージェント	Day1設計生成、Day2実装、Day3自動 PR
GitHub Copilot	補完中心の支援	初級研修で扱った領域。中級では Claude Code 主軸
Cursor / Continue	エディタ統合の対話的支援	本研修対象外、業務適用時の選択肢

ツール	役割	本研修での使い方
Docker / docker-compose	環境のコード化	Day3 Session 02-03
GitHub Actions	CI / CD パイプライン	Day3 Session 04-05
trivy	イメージ脆弱性スキャン	Day3 Session 05
Vercel	サーバレスホスティング	Day3 Session 07発展課題

業務に持ち帰る判断基準

- 1日で完結する作業 → 同期 Chat (Claude Code, Copilot Chat)
- 数時間〜半日待てる作業 → 非同期 Coding Agent (@claude メンション、Copilot Coding Agent)
- 定型的なコード生成 → 補完 (Copilot 補完)
- 新規プロジェクトの土台作り → Cursor / Claude Code の Edits タブ
- 本番ワークフロー組み込み → GitHub Actions + AI (claude.yml)

CLOSING ／ 15:50〜16:00

3日間のまとめと業務棚卸し

3日間で何ができるようになったか、業務にどう接続するかをまとめます。

3日間の到達点

- [x] 顧客ヒアリングメモから要件定義書・ER図・API仕様・シーケンス図を Claude Code で生成、自分の目で精査できる
- [x] 設計書から FastAPI + SQLAlchemy 2.0で実装、テストカバレッジ80%以上を確保できる
- [x] Docker マルチステージ build と docker-compose で複数サービス起動を実機で動かせる
- [x] GitHub Actions の5ジョブ(lint / typecheck / test / docker-build / trivy)全てを緑に保てる
- [x] @claude メンションで Issue → 自動 PR の流れを1周完走できる

明日から実行する3アクション

- [x] 自分の業務リポジトリに `.github/copilot-instructions.md` または `CLAUDE.md` を作り、エラー規約・命名規約・テスト方針を書く
- [x] 次に書く Issue から、受入基準を「入力・期待出力・HTTP ステータス・テストケース」レベルで具体化する
- [x] CI に最低限 lint + test を入れる。docker-build と trivy は段階的に追加

業務棚卸し(別途案内)

JBS 側で別途、業務工程の AI 適性スコア化アプリを案内します。日常業務のうち、どの工程が AI に任せやすいか、どこは人間が最終判断すべきかを可視化するブラウザアプリです。研修後の各自のワークフロー設計に役立ててください。

1ヶ月後の定着アンケート

2026/06/28頃に JBS 事務局からアンケートが届きます。「研修内容を業務で使えたか」「使えなかったとしたら何が足りなかったか」を率直に書いてください。次回研修の改善に活用します。

参考リンク

- Claude Code 公式ドキュメント
- claude-code-action GitHub リポジトリ
- Docker Multi-stage builds 公式
- trivy 公式
- Vercel 公式

制作 Givery／JBS 中級コース 案1 Day3配布用／2026-05-28