GitLab
(CI/CDパイプライン編)

矢野　哲朗

2019年11月24日

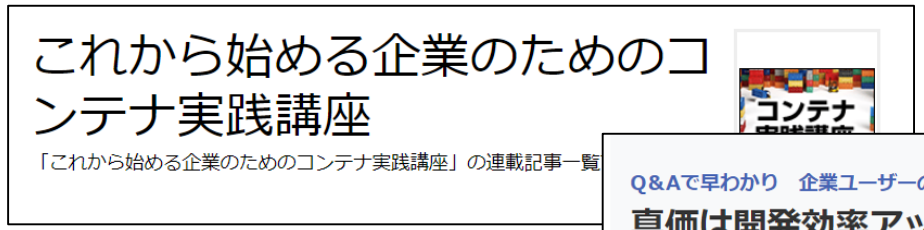
自己紹介

矢野　哲朗

tetsurow.yano

□経歴　：システム運用 10年・ネットワーク 6年・SI 8年
　近頃はNextcloud/Rancherを担当

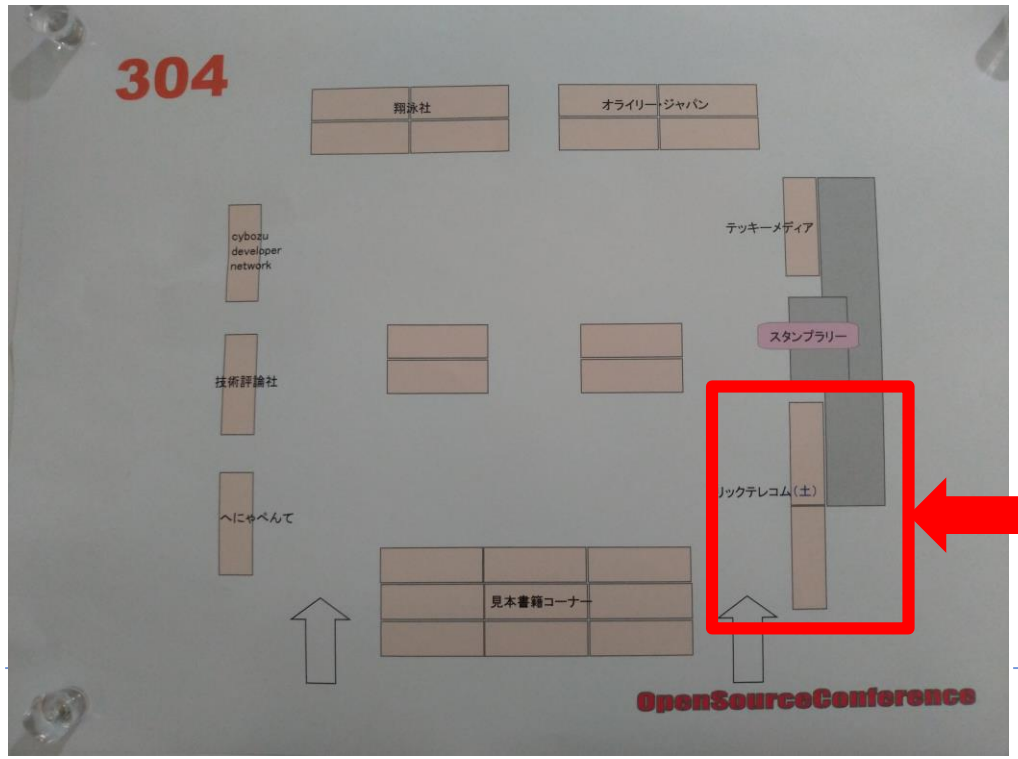
□その他　：全く上達しないRubyist
　一番最初のPCは、OKI if-800 でした…。



＜宣伝＞GitLab CIリファレンス翻訳本！

- GitLab CIのドキュメントが
  なかったので翻訳しました

本日、304教室で販売中



【電子書籍版】日本語版GitLab CIリファレンス
https://techiemedia.booth.pm/items/1579669

GitLabの概要説明

GitLabとは？

GitLabは、Gitリポジトリをホスティングするオープンソースソフトウェアです

- Gitリポジトリの管理以外にも、Issue（課題）の管理、コードレビュー、CI と CD（継続的デリバリ）を1つのツールとして統合的に提供しています
- GitHub と同様にソースコードを見たり、Issue（課題）管理を行ったり、Wiki を書いたり、おおよそ開発に必要な機能が含まれています。これらの機能をウェブブラウザーで利用することができます

GitLab.comとGitLab CE/EEの違い

- GitLab.comについて
  - GitLab.com はGithubのSaaSと同じWebサービス
    無料版の制限がGitHubは、プライベートリポジトリが共同編集者が3人までなのに対して、GitLab.comは無制限（機能制限あり）
  - GitLab.comは、GitHubと同じように有料版があります。
- GitLab CE/EE（= Community Edition/Enterprise Edition）について
  - GitLab CE/EEは、GitHub Enterpriseと同様に、自前のインフラにインストールして動かします。GitHubと違う所は、GitHub Enterpriseは有料のみですが、無料版のCommunity Editionがあります。

GitLabの機能について（無料版で使えるもののみ）

- Gitリポジトリ管理
- Issueトラッカー/ボード
- GitLabパイプライン
- タイムトラッキング
- サイクルアナリシス
- 変更点をレビューするReviewApps
- GitLab ページ
- Git LFS2.0対応
- Wiki
- GitLab Container Registry
- GitLab ウェブエディター
- 組み込みMattermost（オンプレ用）
- Slack連携（ChatOps）
- Webhook
- Kubernetes連携
- AutoDevOps
- Serverless
- Snippet機能

お勧めのGitLab書籍

GitLab実践ガイド

GitLab実践ガイド - インプレスブックス
https://book.impress.co.jp/books/1116101163

GitLab実践ガイド

インフラCI実践ガイド - 翔泳社の本
https://www.shoeisha.co.jp/book/detail/9784798155128

GitLabパイプライン

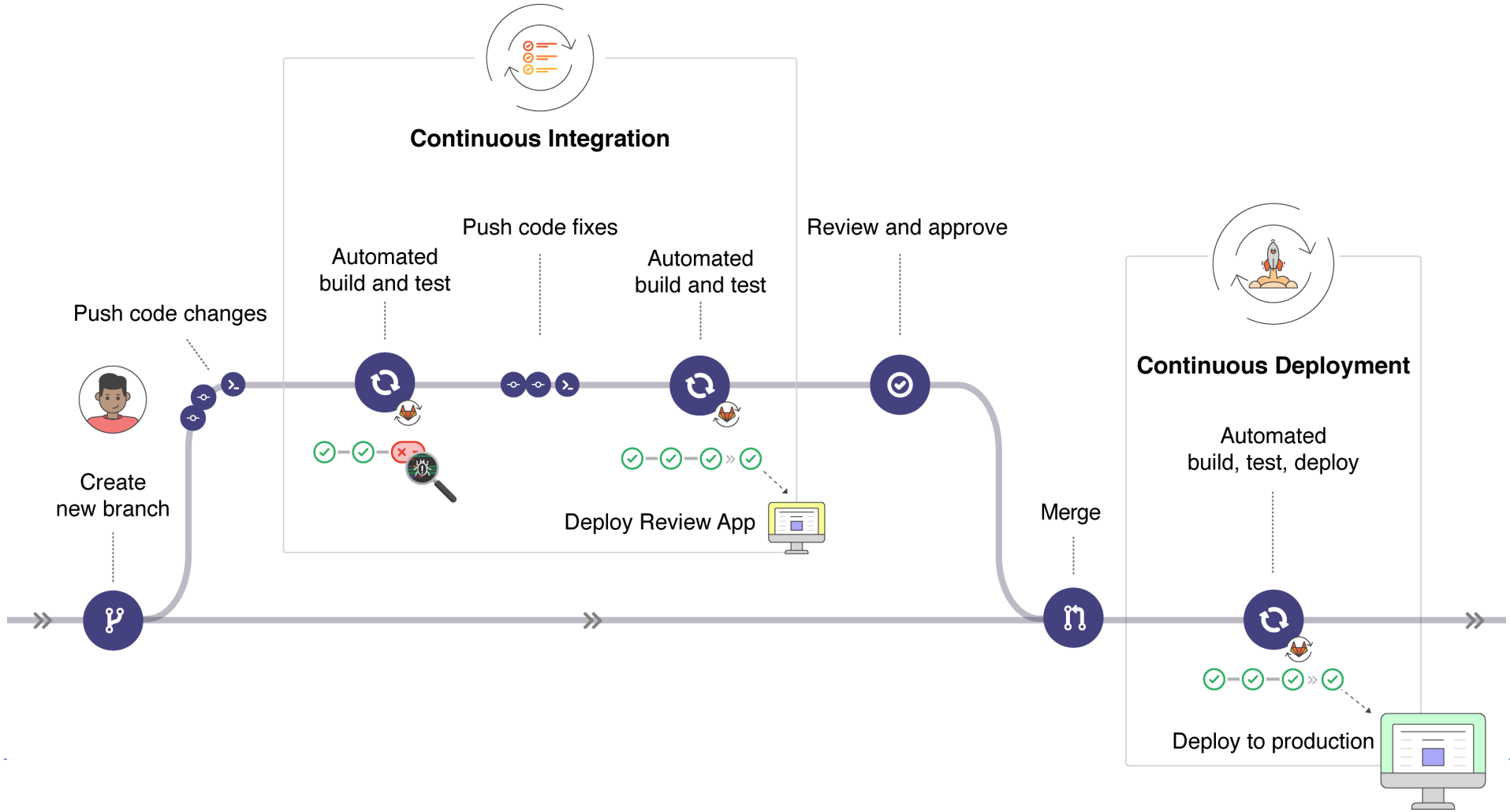
GitLabパイプラインについて

GitLabパイプラインはどういったものなのか？

- Jenkinsのようなジョブ実行機能
- CI (Continuous Integration) とCD (Continuous Deployment or Delivery) を実現する
- GitLab、GitLab Container Registryとの統合
- GitLabへのgit pushイベントで実行される
- .gitlab-ci.yamlファイルに手続き的に記述
- 実行はGitLabサーバー外にあるRunnerが担当

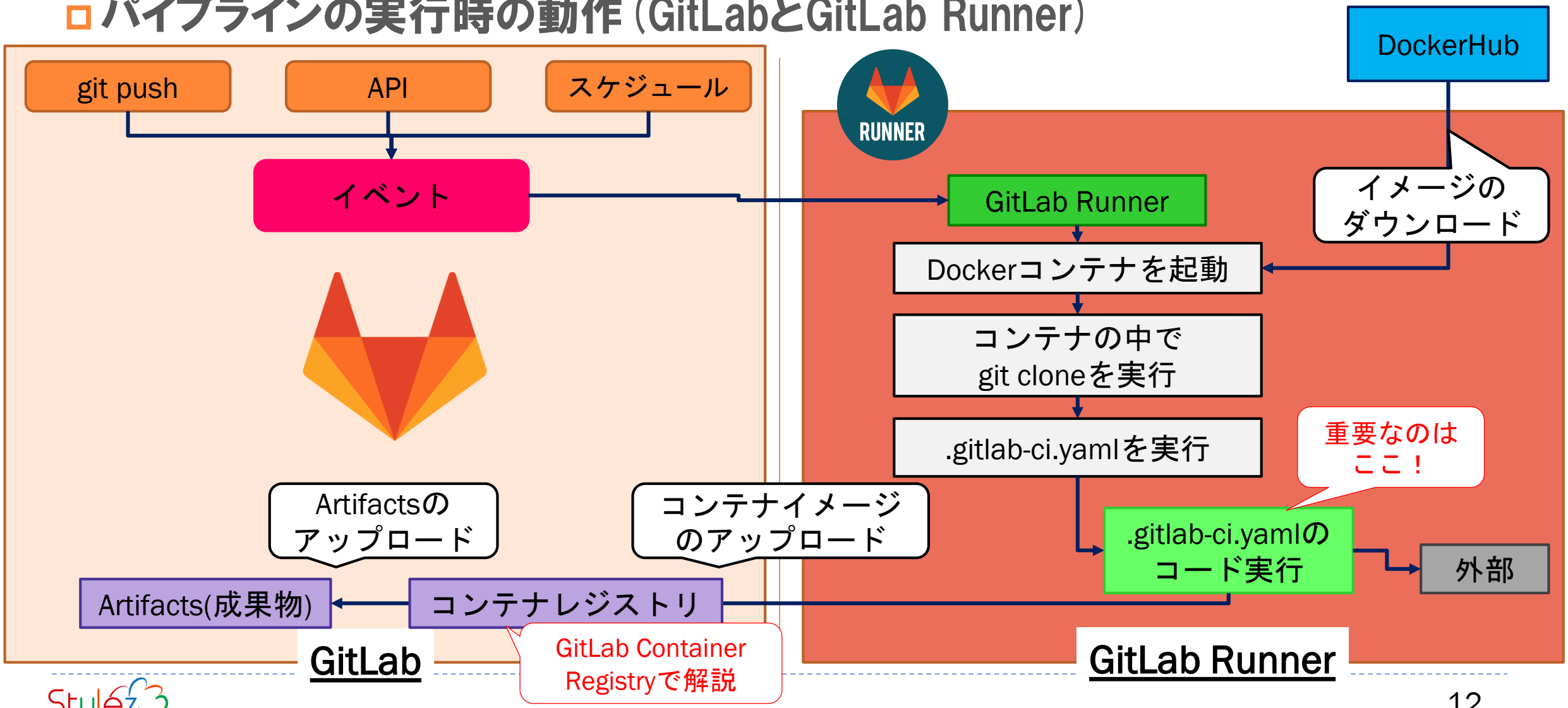
GitLabパイプラインについて

GitLabパイプラインが実現するCI/CD

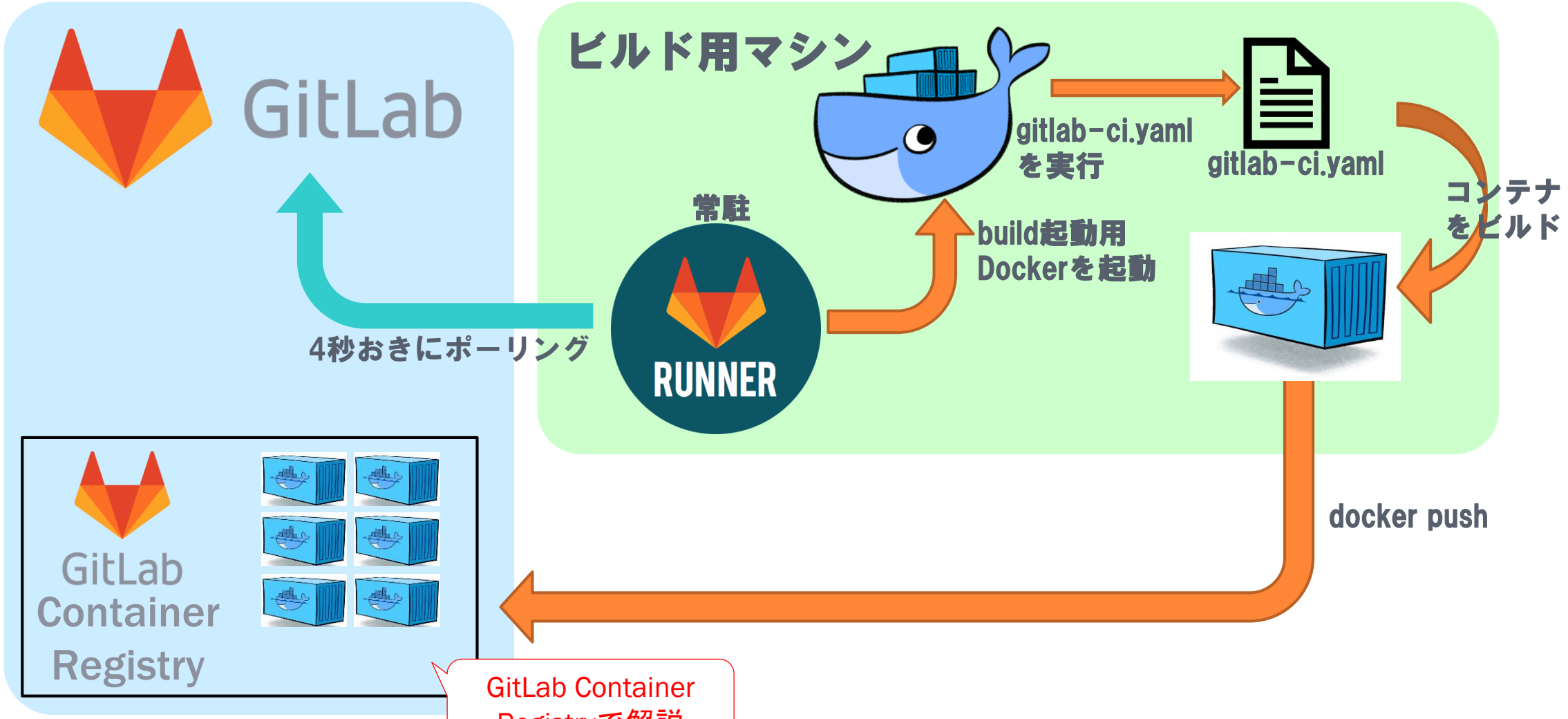


GitLabパイプライン

パイプラインの実行時の動作（GitLabとGitLab Runner）



GitLab CIのRunnerがコンテナを動かす仕組み



GitLab Container Registryで解説

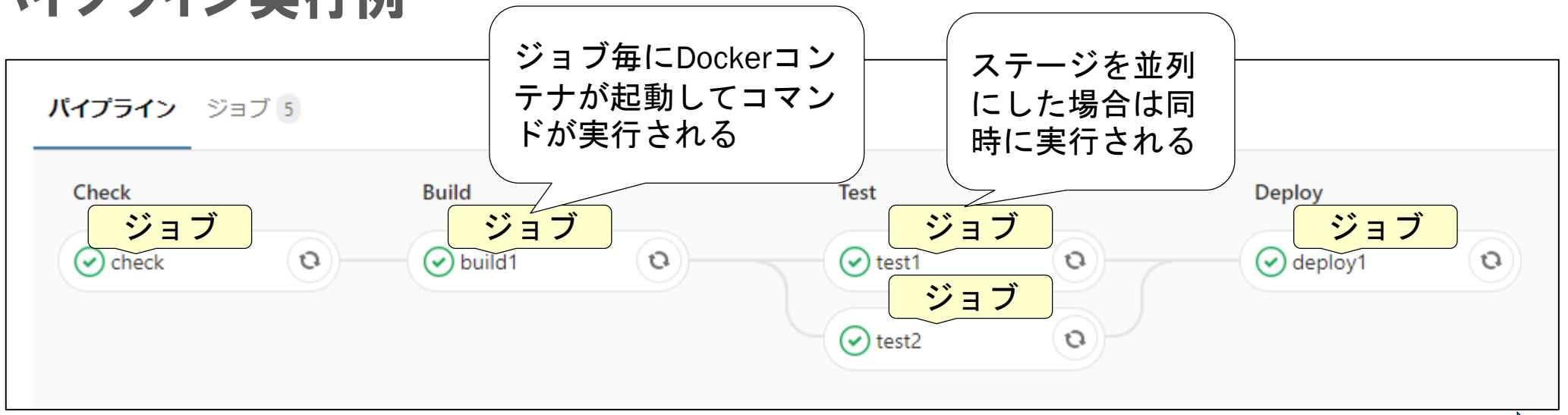
.gitlab-ci.ymlの概要説明

.gitlab-ci.yaml 概要

- YAMLで記述する（サンプルを後述）
- パイプラインの構造と順序を定義する
- GitLab Runnerで実行される
- プロセスの終了状況により次のジョブが実行されるかどうかを決定する
- ファイルの設置場所
  gitリポジトリの直下に置くのがデフォルト（他の場所にも設置可）
  /.gitlab-ci.yaml
- GitLab内の各種環境変数を利用できる

ジョブ実行環境

パイプライン実行例



ジョブ間で共有したいファイルはartifactsで指定

artifacts.pathsで指定されると指定されたpaths配下は別のジョブでも利用できる

```
artifacts:
  paths:
    - binaries/
    - .config:
    - <キャッシュしたいファイルパスを指定>
```

.gitlab-ci.yamlのサンプル解説

```
image: java:8

stages:
  - build
  - run

variables:
  MAVEN_CLI_OPTS: "-s .m2/settings.xml --batch-mode"

build:
  stage: build
  script: mvn $MAVEN_CLI_OPTS compile

run:
  stage: run
  script:
    - mvn $MAVEN_CLI_OPTS package
    - mvn $MAVEN_CLI_OPTS exec:java -Dexec.mainClass="com.example.app.App"
```

このスクリプトを実行する
コンテナイメージを指定

直列に実行する順番を指定
これがない場合は並列に実行

変数指定
GitLabの環境変数も利用できる

ジョブ名

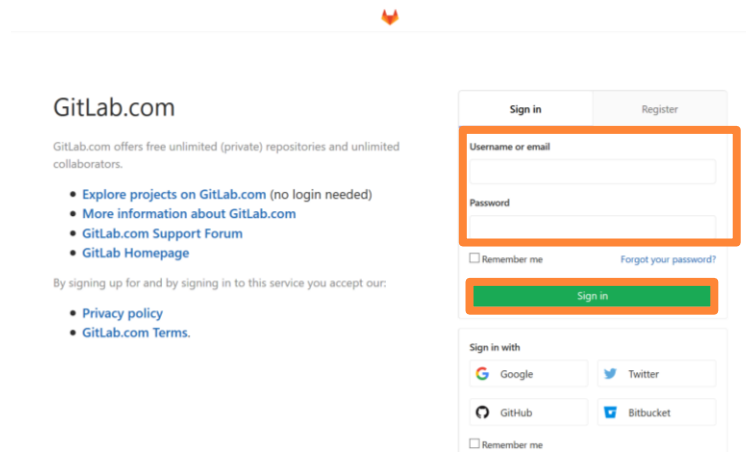
ステージ名
このジョブがどのステージに属しているかを記載

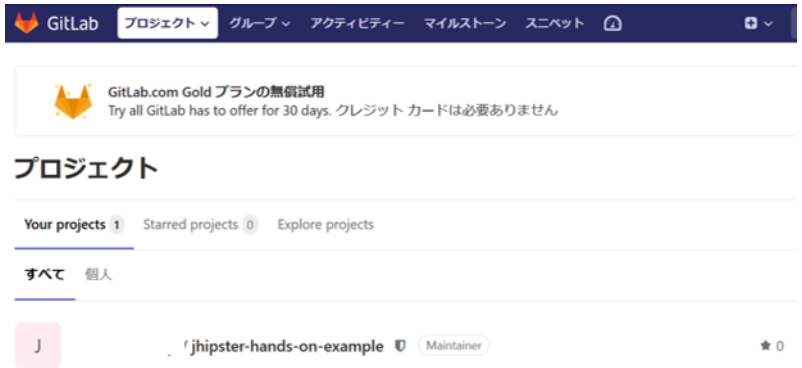
このジョブで実行するコマンドを記載

【Demo】リポジトリを作成

GitLab.com にサインイン

- GitLab.com（https://gitlab.com/users/sign_in）にアクセスしサインイン
- プロジェクトの画面が表示される





Gitlab.com にアクセスし、「Username or email」と「Password」を入力しサインインします。

プロジェクトの画面が表示されます。

.gitlab-ci.yamlファイルの場所

- ファイルの設置場所
  gitリポジトリの直下に置くのがデフォルト（他の場所にも設置可）

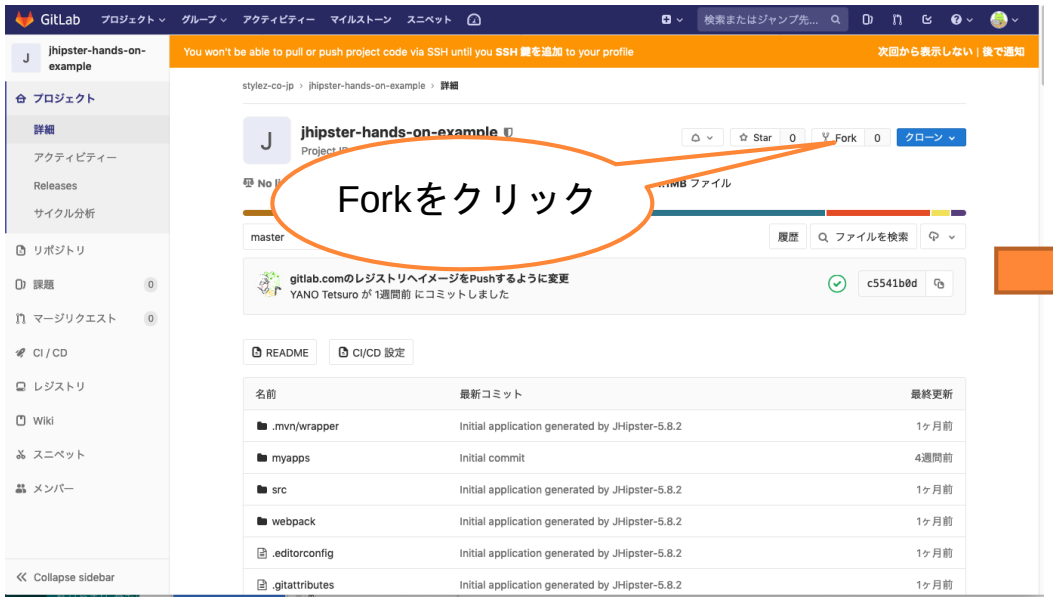
名前	最新コミット
.mvn/wrapper	Initial application generated by JHipster-5.8.2
myapps	Initial commit
src	Initial application generated by JHipster-5.8.2
webpack	Initial application generated by JHipster-5.8.2
.editorconfig	Initial application generated by JHipster-5.8.2
.gitattributes	Initial application generated by JHipster-5.8.2
.gitignore	Initial application generated by JHipster-5.8.2
.gitlab-ci.yml	Update .gitlab-ci.yml

ソースコードのクローン（コピー）

- 演習用リポジトリにアクセス
- Forkをクリック
- 作成したアカウントを選択

演習用ソースコードはこちら

https://gitlab.com/stylez-co-jp/jhipster-hands-on-example

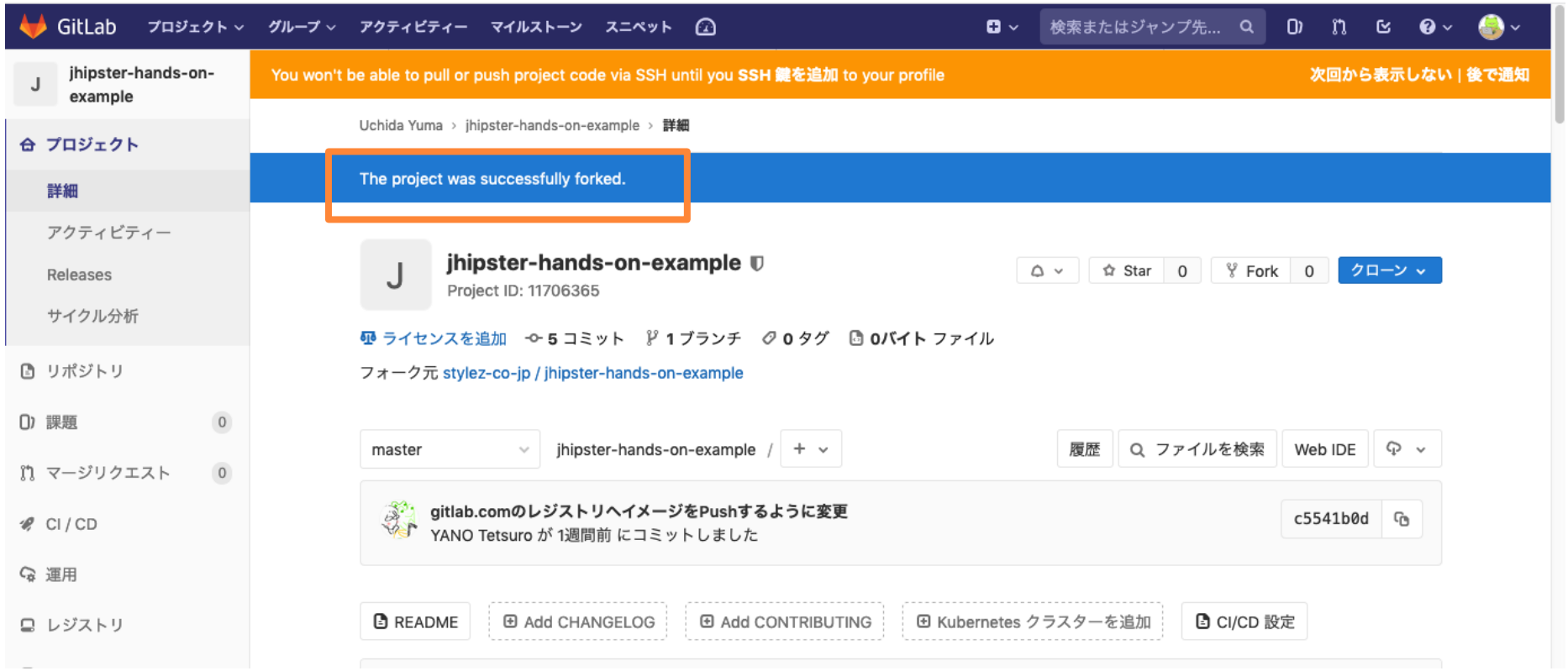


先ほど作成したアカウントをクリック

ソースコードのクローンが終わるまで待ちます

ソースコードクローンの確認

- クローンが完了したメッセージを確認

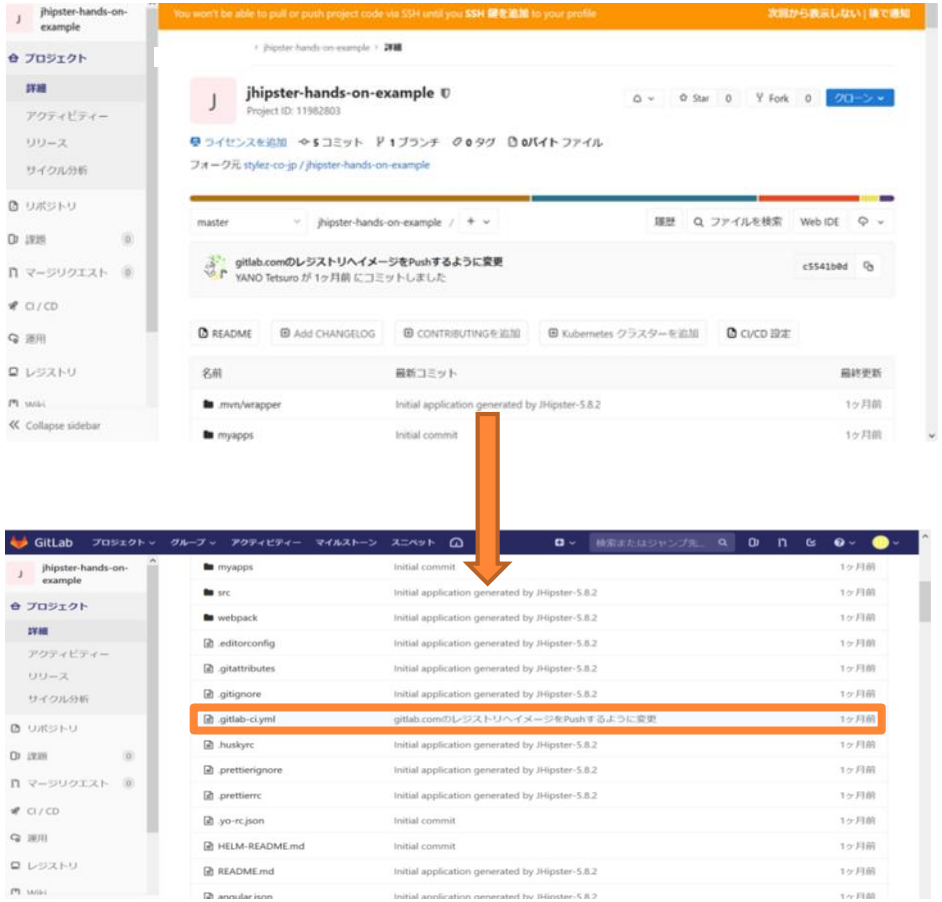


The project was successfully forked. と表示されればクローン完了です

【Demo】パイプラインの起動

.gitlab-ci.yml を開く

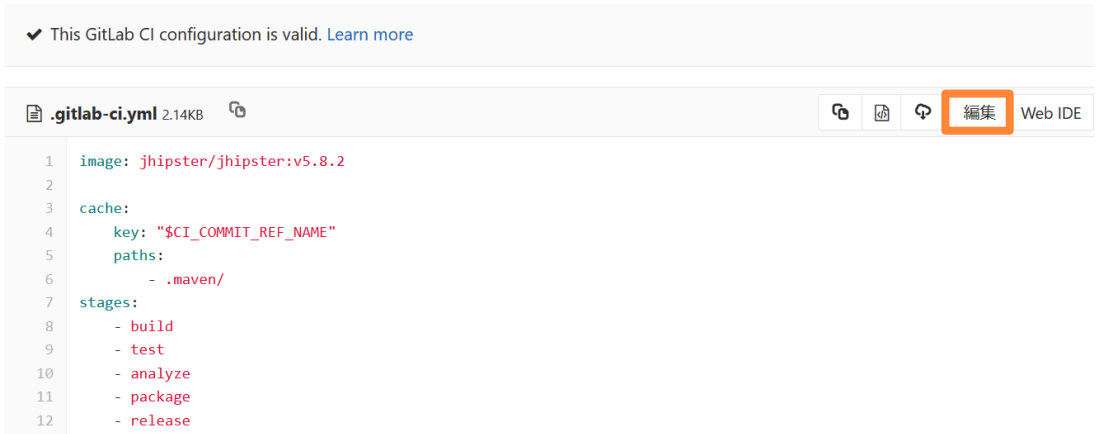
- クローンした「jhipster-hands-on-example」をクリック
- 「.gitlab-ci.yml」をクリック

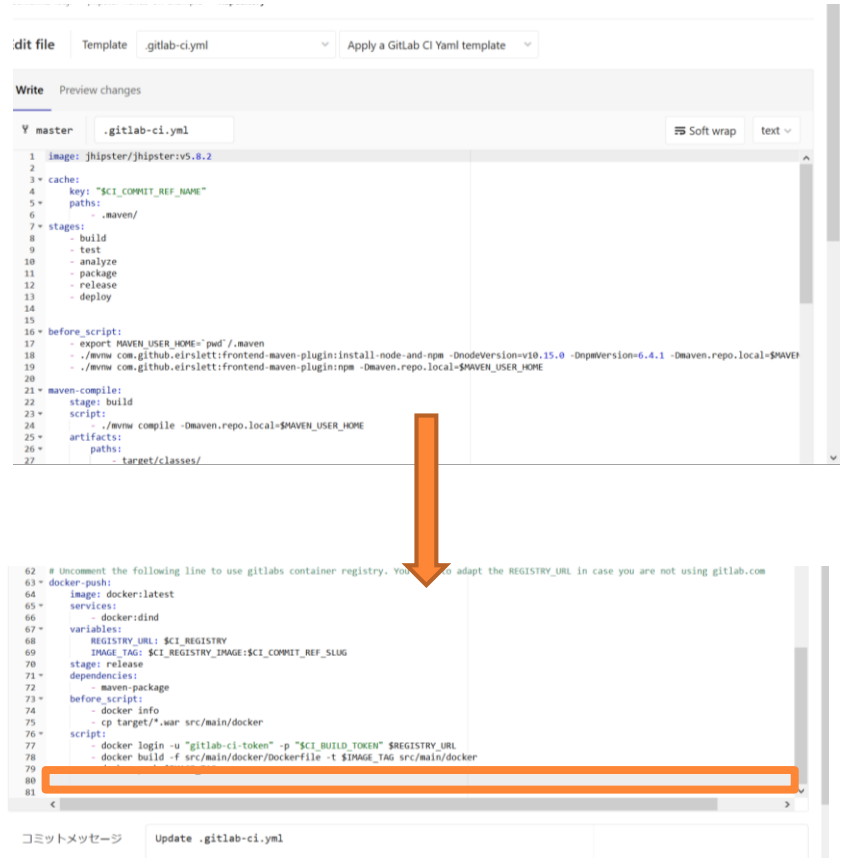


下にスクロールさせて「.gitlab-ci.yml」をクリックします。

.gitlab-ci.yml を編集する

- 「.gitlab-ci.yml」の「編集」をクリック
- 最後の行にひとつ改行を入れます



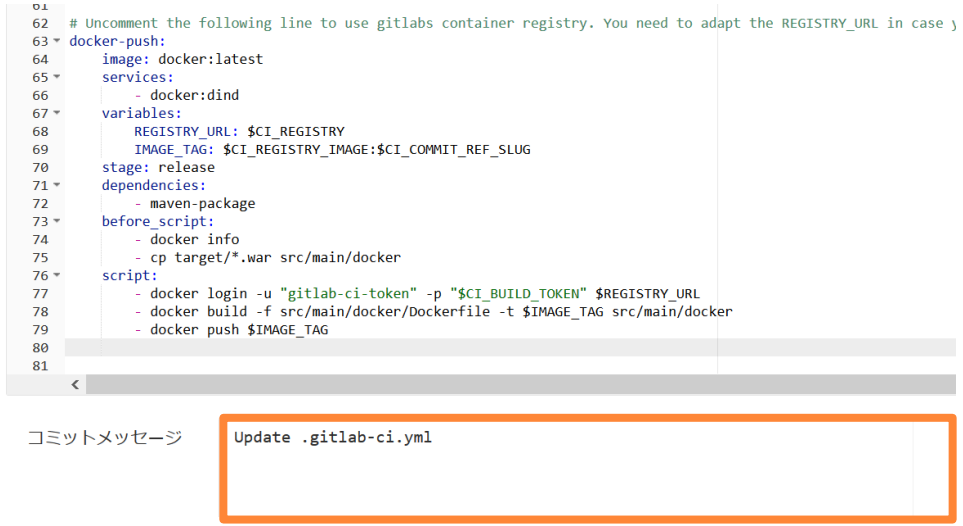


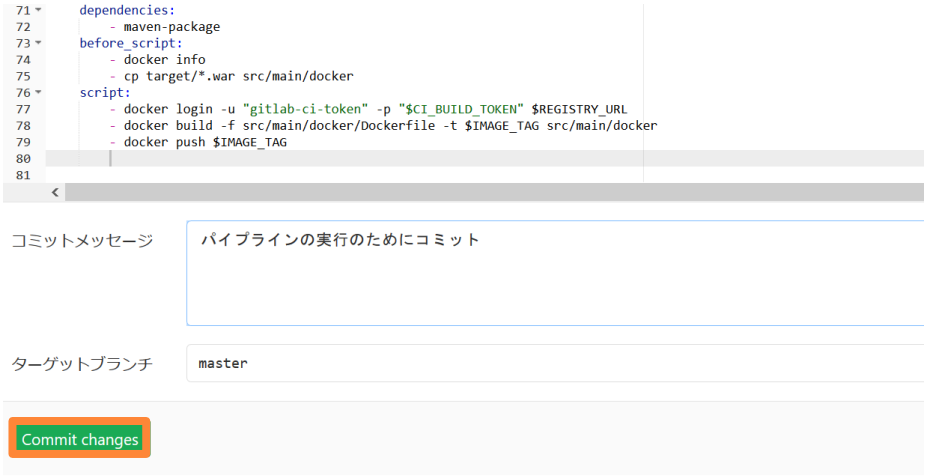
これは今回 JHipster をビルドするパイプラインの記述です。このパイプラインを動かすためには Git レポジトリへのコミットが必要になります。.gitlab-ci.yml の右の「編集」をクリックします。

編集画面の一番下まで画面をスクロールさせます。最後の行で「Enter」を押し、ひとつ改行を入れます。

コミットメッセージを追加する

- 画面下のコミットメッセージにメッセージを入力
- 左下の「Commit changes」をクリック





入力メッセージは任意のもので構いませんが、
今回は「パイプラインの実行のためにコミット」と入力します。

「Commit changes」をクリックします。

コミットの確認

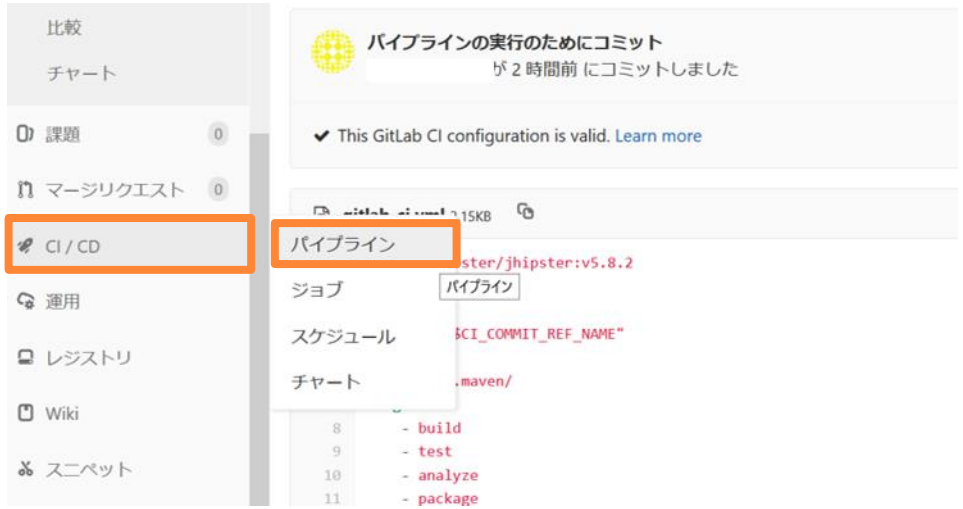
- 「Your changes have been successfully committed.」を確認
- 改行が入っていることを確認

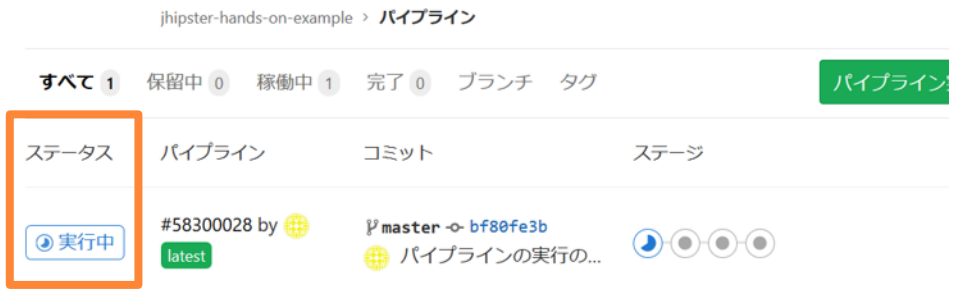
画面が戻り「 Your changes have been successfully committed. 」メッセージが表示されます。

画面を下にスクロールさせ、改行が入っていることを確認してください。

パイプラインのステータス確認

- 「CI/CD」から「パイプライン」をクリック
- ステータスが「実行中」であることを確認



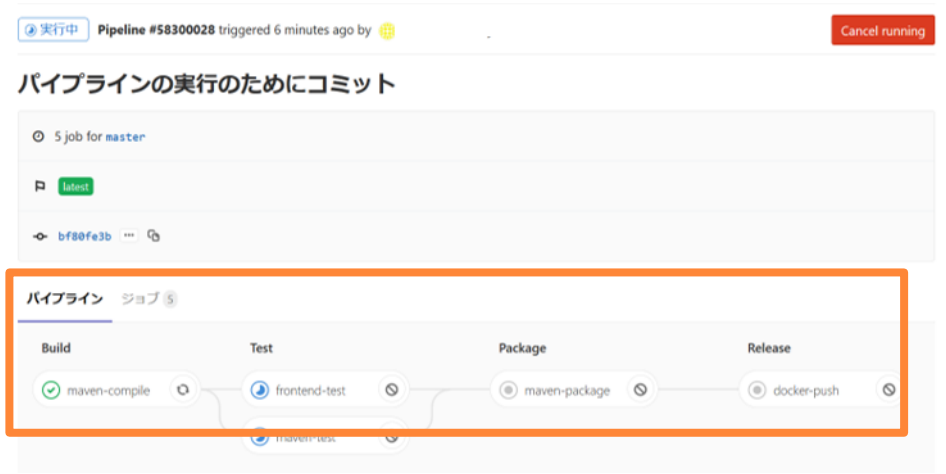


左のナビゲーションの「CI/CD」から「パイプライン」をクリックします。

パイプライン画面になりましたら、ステータスが「実行中」であることを確認してください。

パイプラインの繋がりとジョブの確認

- 「実行中」または「# 数値」をクリック
- パイプラインを確認

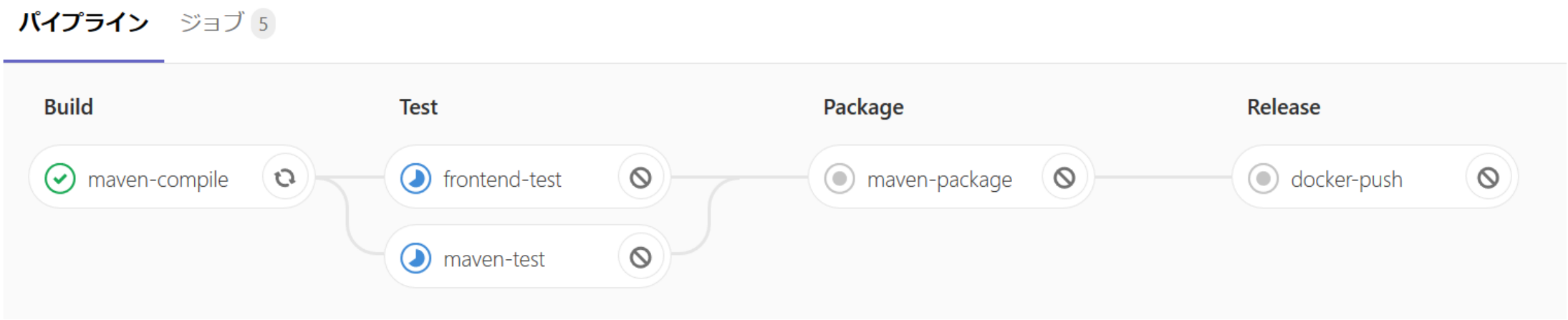


ステータスの「実行中」かパイプラインの「# 数値」をクリックします。

今回実行しているパイプラインの繋がりとジョブが表示されます。

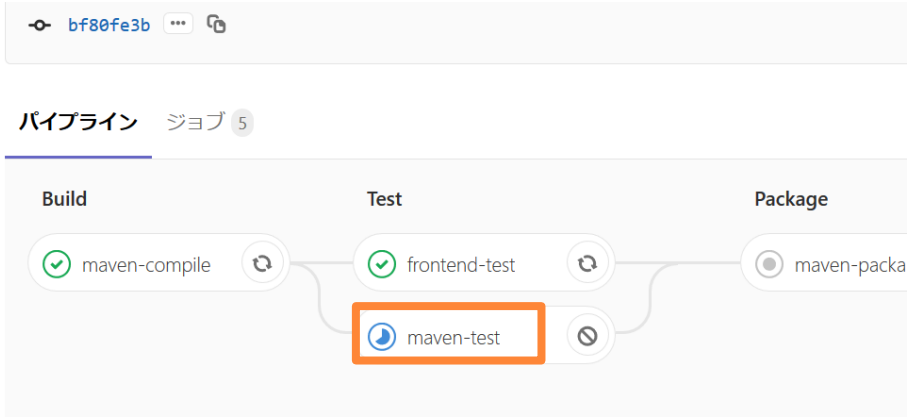
（参考）ステータスについて

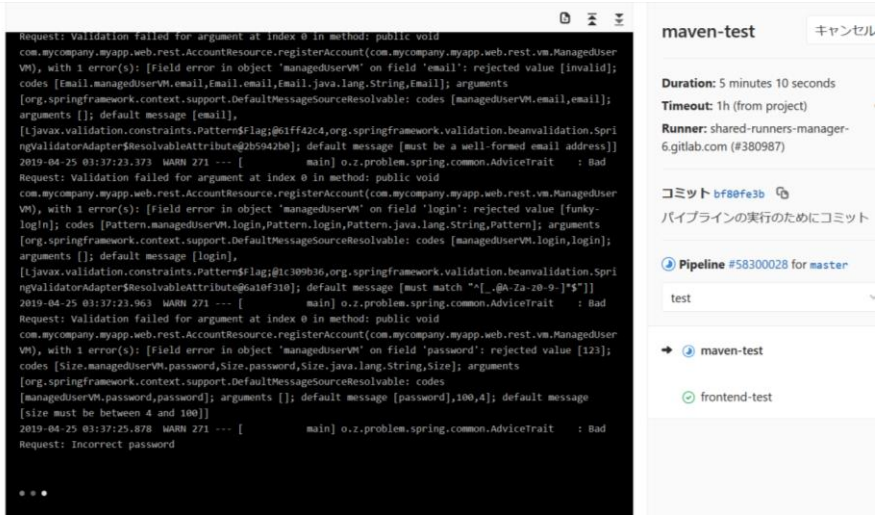
- 青いタイマーは「実行中」です
- 緑色のチェックマークは「完了」になります
- グレーの丸は「未実行」です



実行中の確認

- 「実行中」のものをクリック
- 内容を確認



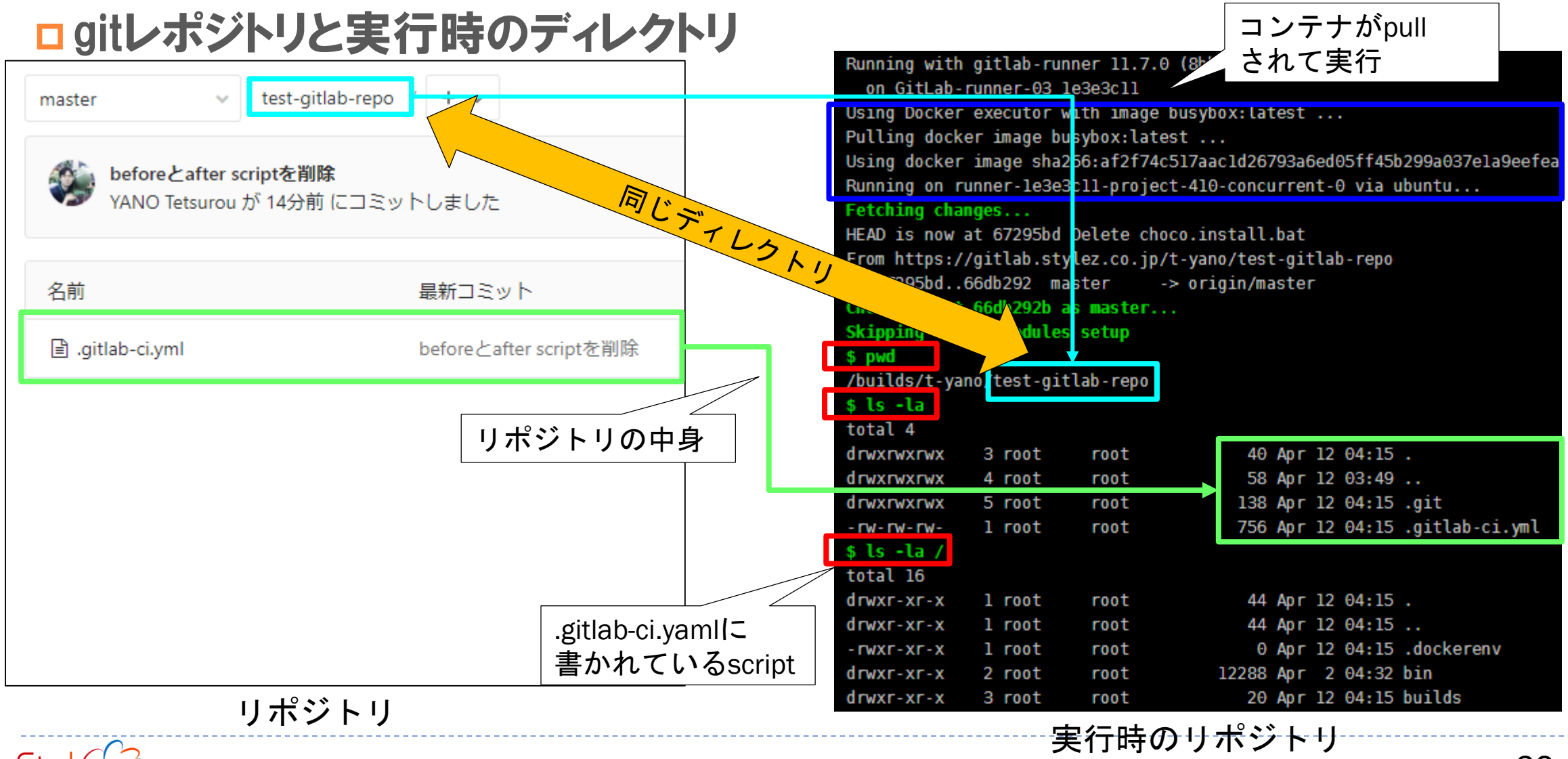


いずれか「実行中」のものをクリックします。

黒い画面にテキストメッセージが表示され、
実行中であることが確認できます。

1つのジョブが実行される様子

gitレポジトリと実行時のディレクトリ



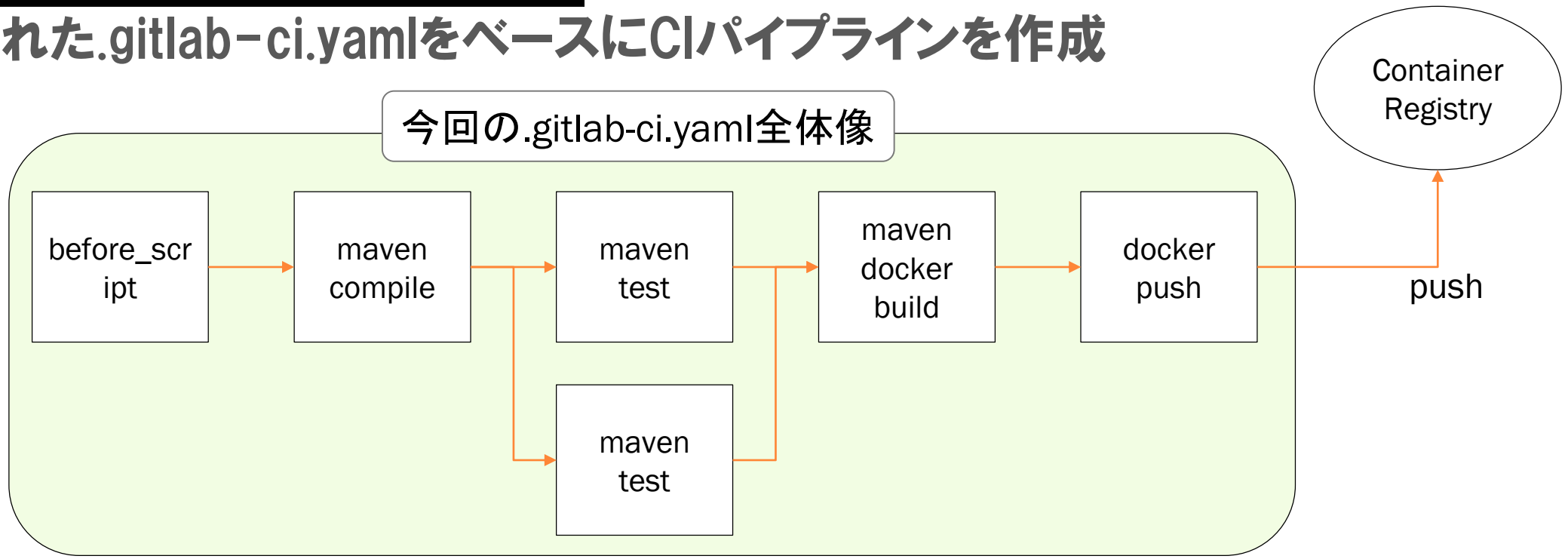
今回の.gitlab-ci.yamlの流れ

- JHipsterで.gitlab-ci.yamlを生成する
  jhipsterで生成するコマンド

```
jhipster ci-cd autoconfigure-gitlab
```

```
その他のCIツールのオプション
autoconfigure-travis    # Travis
autoconfigure-jenkins   # Jenkins
autoconfigure-azure     # Azure
```

- 生成された.gitlab-ci.yamlをベースにCIパイプラインを作成



実際のYAMLファイル https://gitlab.com/stylez-co-jp/jhipster-hands-on-example/blob/master/.gitlab-ci.yml

.gitlab-ci.yamlの制御構造 (only/except)

- onlyの基本
  - ここで指定のブランチ名、タグ名がgit pushされた時に実行する
- exceptの基本
  - ここで指定のブランチ名、タグ名がgit pushされた時には実行しない
- 共通ルール
  - 正規表現での指定も可能 (RE2なので注意)
  - branches、tags、api等のgit名、サービス名での実行指定も可能
- その他の指定の仕方
  - only:refs/except:refs ： 指定のgit refの時
  - only:variables/except:variables ： 指定の変数があった時
  - only:changes/except:changes ： 指定のファイルの変更時

サンプル

```
job:
  # use regexp
  only:
    - /^issue-.*$/
    - dev
  # use special keyword
  except:
    - master
```

ステージ間でのファイルのやり取り（artifactsとcache）

- artifacts
  - ステージ間のジョブとジョブでのファイルのやり取り
  - コンパイルした成果物は、artifactsを使う
  - artifactsを使うと指定部分が必ずGitLabにアップロードされる
  - パイプライン終了後もGitLabから参照したい場合も、artifactsを利用する
- cacheの使い道
  - プロジェクト（パイプライン）で一時的にインターネットからダウンロードされるファイルをジョブ間でキャッシュとして共有し高速化するために使う
  - キャッシュの存在は保証はされていない
  - 保証されない理由は、キャッシュはRunnerホスト内でのみ共有する仕組みで別のRancherホストで動いた場合には利用できないから
  - 保証されていないため、before_scriptと組み合わせて使うのを推奨

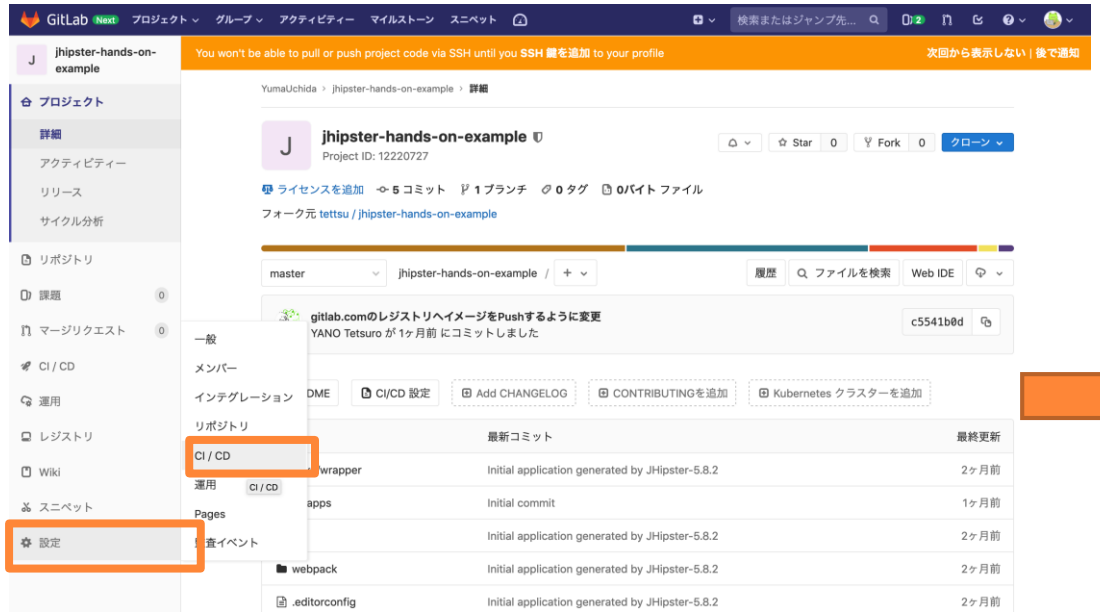
GitLab.comでのPipeline制限

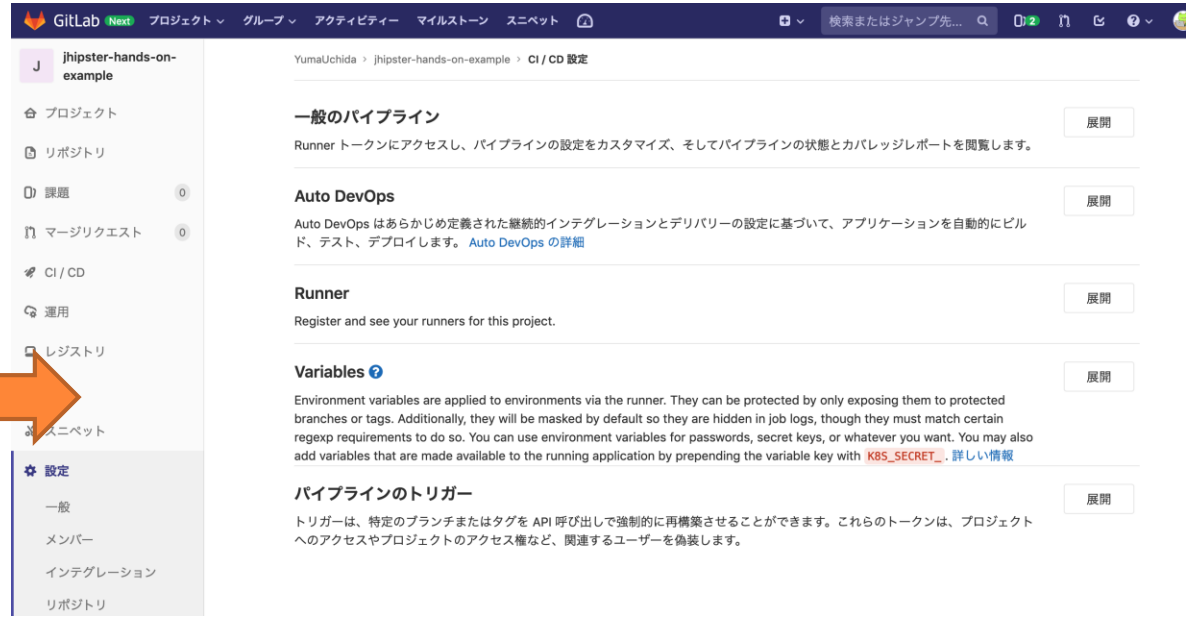
- 無料版で利用できる時間
  - 使える時間は、2000分まで
- 利用時間の確認方法

GitLabRunnerとGitLabPipeline、GitLabレジストリの解説

Gitlab.com上でのCI/CD設定概要

- gitlab-ci.ymlファイルとCI/CD画面以外の重要項目を解説します
- 3つの注意するべき点を説明します





左サイドバー → 設定 → CI/CDをクリック

CI/CDの動作を設定する画面

各種設定項目

一般のパイプライン

「一般のパイプライン」はパイプラインに関する一般的な設定を行います

Auto DevOPS

GitLab側で必要と考えているDevOpsの作業を自動的に実行してくれる機能

Runner

パイプラインの実行に必要なRunnerプロセスに関する設定を行う場所

このRunnerは、.gitlab-ci.yamlで書かれたジョブを実行するプロセスになります

Variables

Runnerでジョブを実行するときに利用する環境変数を設定する場所

パイプラインのトリガー

パイプライン実行の契機は、基本的にGitLab.comへの操作ですが、外部からAPIで呼び出すことによりパイプラインを実行することもできる設定

一般のパイプライン

Git strategy for pipelines

ソースコードを取得する際のコマンドを選択できる設定

・git clone: 毎回全てのソースコードをダウンロード

・git fetch: 既存コードとの差分だけを取得

タイムアウト

ジョブ全体のタイムアウト時間。設定値以上の時間がかかる場合はfailとなる

カスタムCI configパス

リポジトリ内にある.gitlab-ci.yamlの場所を指定します

デフォルト以外にgitlab-ci.ymlが存在する場合にパスを記入

ファイル名も変更可能

一般のパイプライン - 2

公開パイプライン

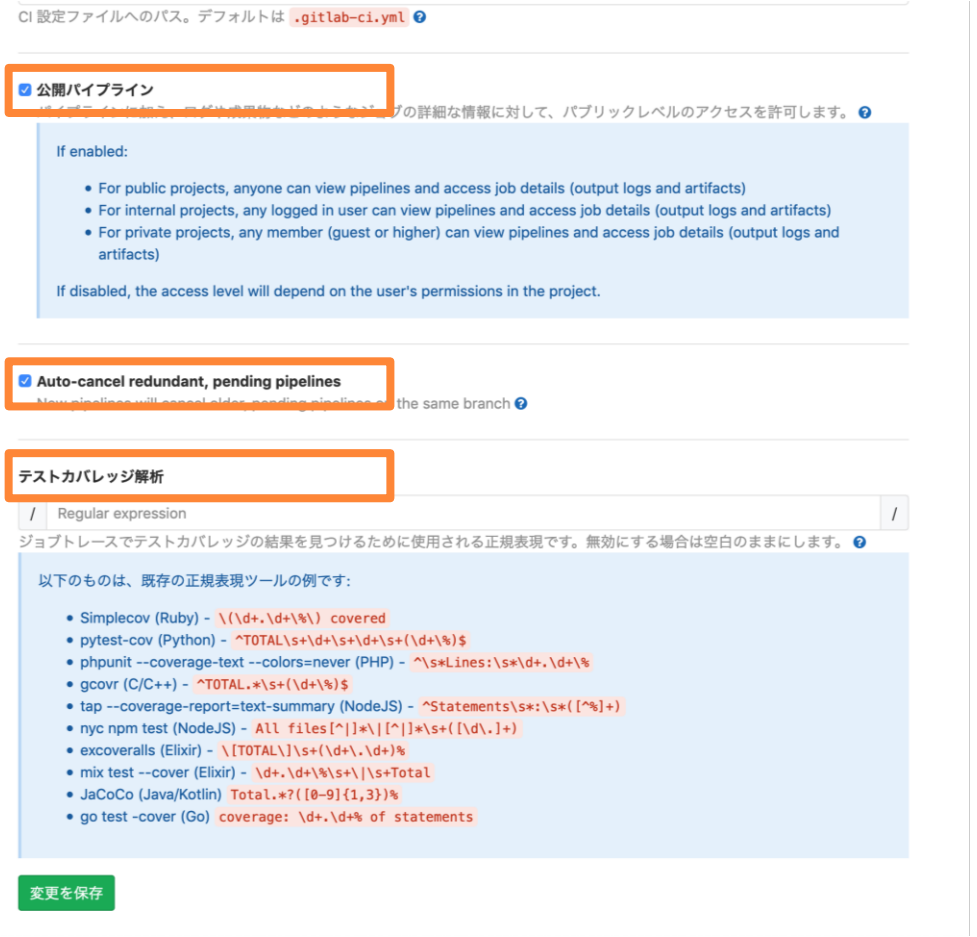
ジョブで生成したファイルなどをダウンロードできるようにするか
否かを設定できる機能(対象ファイルはコンパイル後のファイルなど)

Auto-cancel redundant, pending pipelines

HEADブランチ以外でパイプラインが重複した場合に古いジョブを
自動的に停止。複数ジョブが起動した場合に適用される

テストカバレッジ解析

カバレッジログを取得してテストのカバレッジの割合を取得する機能



一般のパイプライン - 3

Pipline status

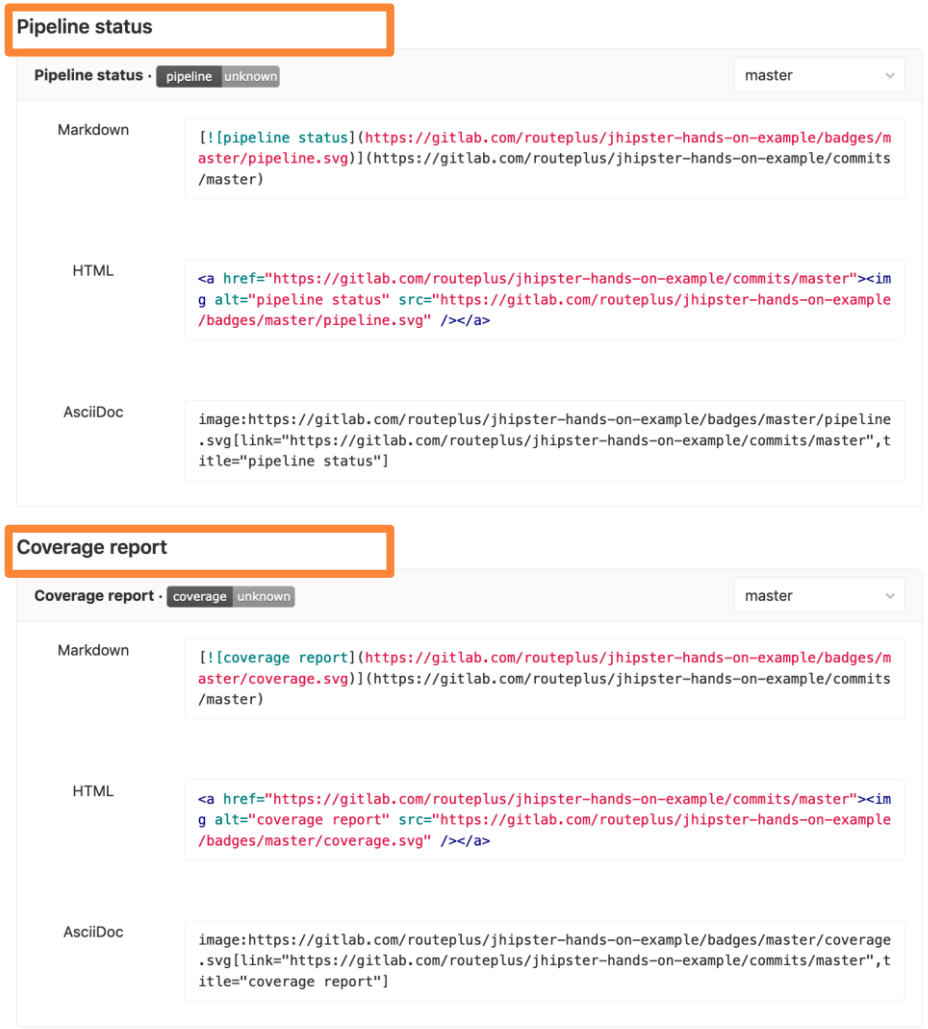
パイプラインの状況を表示する機能

成功すれば「passed」、どこかで失敗していたら「fail」で表示

Coverage report

カバレッジの状態を表示する機能

カバレッジ解析の結果を画像として表示する



Auto DevOps

Auto DevOps

Auto DevOpsはGitLabが規定するDevOpsのやり方に沿り自動的にパイプラインを実行してくれる機能

Auto DevOpsにより、ビルドからコードのテスト、チェック、アプリのレビューまで自動的に進行する事が可能

Runner

Runnerの概要

オンプレで動かす場合は、別途自分でRunnerを準備する必要がありますが、GitLab.comでは他のユーザーと共有する共有Runnerが自動的に割りあてらるため、特別な設定が必要なくすぐに利用可能

利用可能な共有 Runner

オレンジ枠に割り当てられた共有Runnerが表示される

オンプレミス上のGitlabの場合は管理者からの設定が必要なため、共有Runnerの利用が必要な場合は事前に問い合わせる

＊ gitlab.com上であっても自前のRunnerを設定することが可能

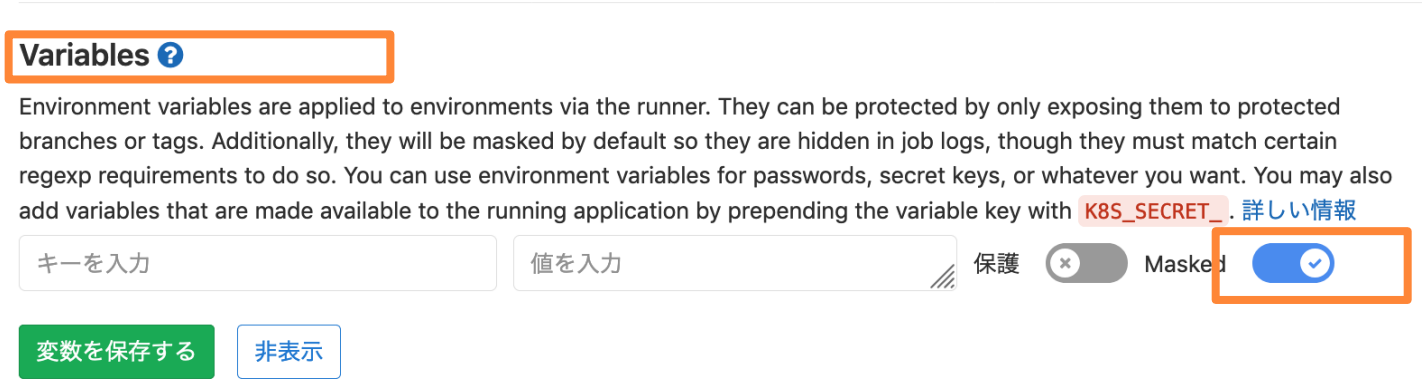
Variables

Variablesの内容

Runnerでジョブを実行するときに利用する環境変数を設定する

ジョブ実行時に値を表示しない「Masked」がデフォルト（オレンジ枠の部分をクリックで切り替え可能）

保護をonにすると、保護ブランチ、タグ上でしか利用されない環境変数になる



GitLab CI/CD実行時の環境変数

- 自動的にCI_xxxxの環境変数がセットされる

$ env
CI_BUILD_ID=12866
CI_SERVER_VERSION_PATCH=7
CI_SERVER_REVISION=93e3215
CI_COMMIT_SHORT_SHA=66db292b
GITLAB_USER_LOGIN=t-yano
CI_BUILD_REF=66db292be13a6738dfa472aeaf25aac0bcb25bc5
CI=true
CI_PROJECT_NAME=test-gitlab-repo
CI_RUNNER_REVISION=8bb608ff
HOSTNAME=runner-1e3e3c11-project-410-concurrent-0
CI_JOB_STAGE=check
CI_COMMIT_DESCRIPTION=
CI_SERVER_VERSION=11.9.7
SHLVL=2
HOME=/root
OLDPWD=/
CI_JOB_ID=12866
CI_COMMIT_REF_NAME=master
CI_PIPELINE_SOURCE=push
CI_RUNNER_VERSION=11.7.0
CI_SERVER_VERSION_MAJOR=11
CI_BUILD_TOKEN=xxxxxxxxxxxxxxxxxxxx
CI_REGISTRY_PASSWORD=xxxxxxxxxxxxxxxxxxxx
GITLAB_FEATURES=
CI_PROJECT_ID=410
CI_REGISTRY_IMAGE=registry.stylez.co.jp/t-yano/test-gitlab-repo
GITLAB_CI=true
CI_COMMIT_SHA=66db292be13a6738dfa472aeaf25aac0bcb25bc5
CI_REGISTRY_USER=gitlab-ci-token
CI_PROJECT_DIR=/builds/t-yano/test-gitlab-repo
CI_PROJECT_PATH=t-yano/test-gitlab-repo
CI_PROJECT_NAMESPACE=t-yano
CI_JOB_TOKEN=xxxxxxxxxxxxxxxxxxxx
CI_SERVER_NAME=GitLab
CI_NODE_TOTAL=1
CI_PIPELINE_URL=https://gitlab.stylez.co.jp/t-yano/test-gitlab-repo/pipelines/4829
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
CI_SERVER_VERSION_MINOR=9
CI_RUNNER_DESCRIPTION=gitlab-runner-03
CI_BUILD_REF_SLUG=master
CI_PROJECT_VISIBILITY=private
CI_COMMIT_TITLE=beforeとafter scriptを削除
GITLAB_USER_EMAIL=tetsuro.yano@stylez.co.jp
CI_SERVER=yes
CI_BUILD_BEFORE_SHA=67295bd87b1ce237dd0f698b434816f36a16f4ce
CI_PAGES_URL=https://t-yano.pages.stylez.co.jp/test-gitlab-repo
FF_K8S_USE_ENTRYPOINT_OVER_COMMAND=true
CI_PAGES_DOMAIN=pages.stylez.co.jp
CI_REPOSITORY_URL=https://gitlab-ci-token:xxxxxxxxxxxxxxxxxxxx@gitlab.stylez.co.jp/t-yano/test-gitlab-repo.git
CI_RUNNER_ID=6
CI_REGISTRY=registry.stylez.co.jp
CI_API_V4_URL=https://gitlab.stylez.co.jp/api/v4
GITLAB_USER_NAME=YANO Tetsurou
CI_PIPELINE_IID=12
CI_JOB_URL=https://gitlab.stylez.co.jp/t-yano/test-gitlab-repo/-/jobs/12866
CI_BUILD_NAME=check
CI_RUNNER_EXECUTABLE_ARCH=linux/amd64
CI_COMMIT_REF_SLUG=master
CI_DISPOSABLE_ENVIRONMENT=true
PWD=/builds/t-yano/test-gitlab-repo
CI_RUNNER_TAGS=
CI_SERVER_TLS_CA_FILE=/builds/t-yano/test-gitlab-repo.tmp/CI_SERVER_TLS_CA_FILE
CI_PIPELINE_ID=4829
CI_COMMIT_BEFORE_SHA=67295bd87b1ce237dd0f698b434816f36a16f4ce
CI_BUILD_REF_NAME=master
CI_PROJECT_PATH_SLUG=t-yano-test-gitlab-repo
CI_PROJECT_URL=https://gitlab.stylez.co.jp/t-yano/test-gitlab-repo
CI_CONFIG_PATH=.gitlab-ci.yml
CI_BUILD_STAGE=check
CI_COMMIT_MESSAGE=beforeとafter scriptを削除
GITLAB_USER_ID=4
CI_JOB_NAME=check

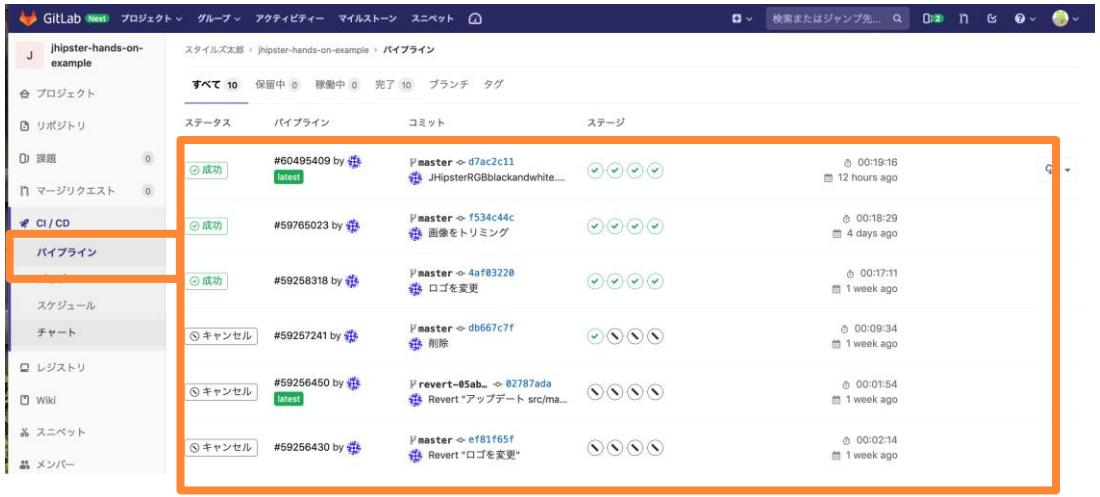
GitLab CI/CD Variables を翻訳しました。 - Qiita
https://qiita.com/yuuta/items/4c5085b4cd6221bb71c5

GitLab CI/CDの画面解説

ダッシュボード

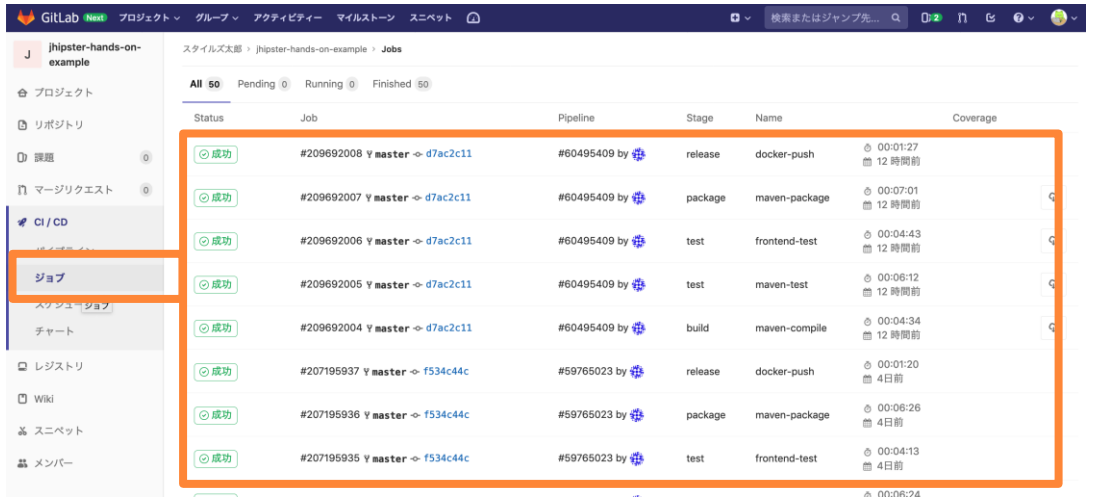
メインカラムにジョブの一覧が表示される（ダッシュボード）

各ジョブの実行状態が表示されている



ジョブ詳細

パイプライン中のジョブを1つずつ詳細に表示している



スケジュール画面

スケジュールとは

イベントやトリガー契機でパイプラインを実行するのではなく、時間でパイプラインを実行する。

実行の時間指定方法は、cron構文で行い、タイムゾーンとブランチの指定、環境変数も設定が可能

スケジュールが設定されていた場合表示される

スケジュール設定

新規スケジュールをクリック →
・パイプラインの説明
・タイムゾーン
・ 実行パターン（CRON形式での指定）
・ ターゲットブランチ
・ 変数
などを指定し、スケジュール実行が可能

レジストリ

Container Registry とは

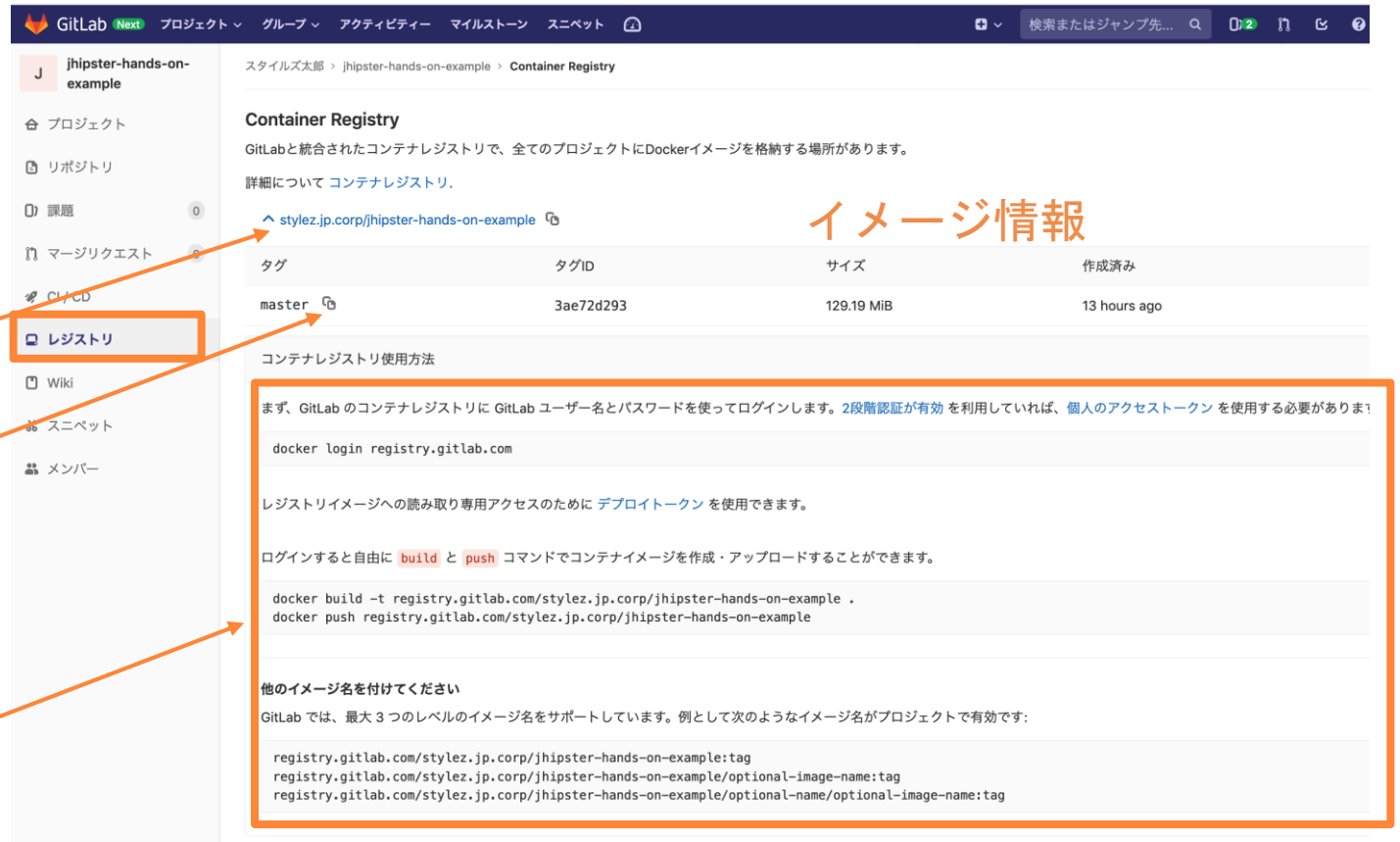
CI/CDで使用するDocker イメージを格納できる機能

現在保存されているコンテナイメージついての情報を確認できる

クリックで展開

リンクになっているためアイコンをクリックするとimageのURLをコピー可

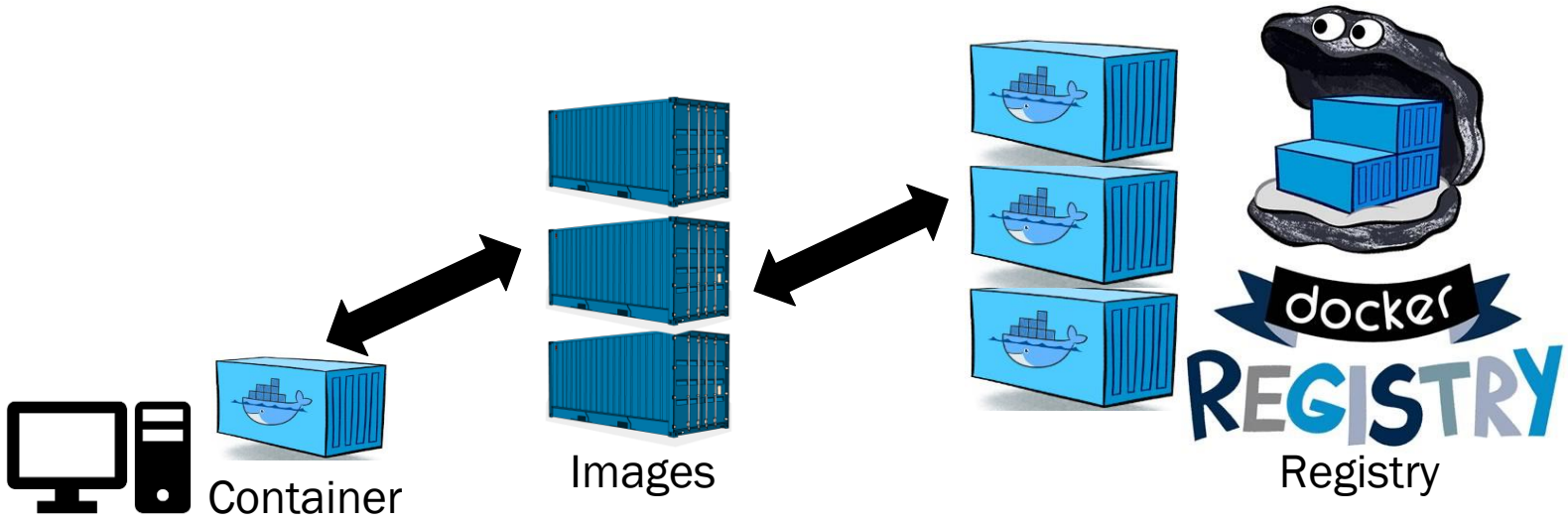
コンテナイメージ取得方法の解説



GitLab Container Registry

Docker Container Registry

- Docker Container Registryとは？
  - Dockerイメージを保存する場所（docker pushして送信する先）
  - Dockerイメージをプライベートに管理する（有料ではDocker Hubでも可能）
  - Dockerイメージを配布管理する



GitLab Container Registry

- GitLab Container Registryとは？
  - Docker Container Registryと同じもので、GitLabに統合されている
  - dockerコマンドで利用できる（docker loginが必要）
  - GitLabのアカウントで認証する
  - GitLabパイプラインの中から使うと認証は楽に統合できる
  - プロジェクト単位でPublic、Private設定が可能
  - その他の機能はDocker Container Registryと同じ
  - GitLab Container Registryにpushする場合は、docker imageにContainerレジストリのタグをつけること

レジストリ

Container Registry とは

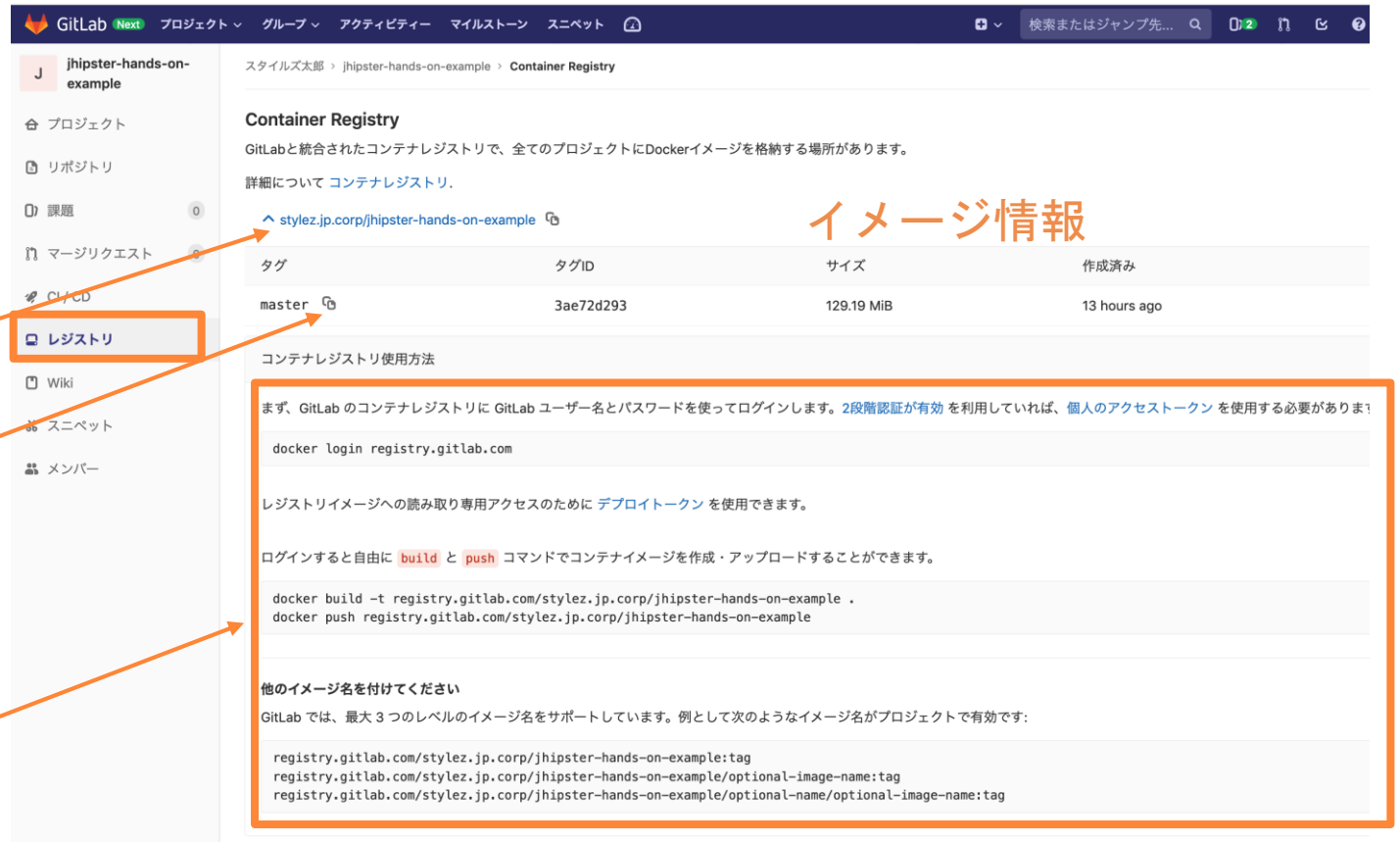
CI/CDで使用するDocker イメージを格納できる機能

現在保存されているコンテナイメージついての情報を確認できる

クリックで展開

リンクになっているためアイコンをクリックするとimageのURLをコピー可

コンテナイメージ取得方法の解説



GitLab CI/CDパイプライン設定リファレンスについて

- より詳細なGitLab CI/CDパイプラインの.gitlab-ci.yamlの書き方については、以下を参照してください。
  https://gitlab.com/stylez-co-jp/gitlab-ce/tree/dev-v12.0.3-ja.1/doc-ja/ci/yaml

まとめ

- GitLab CI/CDパイプラインは、Jenkinsのような自動ビルドをするための機能がある
- 設定方法は、.gitlab-ci.ymlファイルをGitリポジトリに設置することで自動的に実行される
- .gitlab-ci.ymlには構文があり、複数のステージとジョブを定義する
- ジョブは、GitLab Runnerというコンテナー内のシェルで実行される
- ビルド成果物は、artifactsとして保存され、次のジョブに利用することもできる
- 環境変数を指定することによりジョブの挙動を変更することができる