

Red Hat トレーニング

DevOps Culture & Practice (TL500)

技術演習事前準備

2024/07/27

V1.0 Revision 0.91

Red Hat トレーニング

DevOps Culture & Practice (TL500)

1. TL500 VLab リクエスト

2. インストラクターサーバー設定

- 2-1. インストラクター仮想マシン生成
- 2-2. インストラクターWorkstationのコンソールを開く
- 2-3. WorkstationのターミナルでTL500ネットワークのIPアドレス取得

3. 受講者サーバー設定

- 3-1. 受講者仮想マシン生成
- 3-2. 受講者Workstationのコンソールを開く
- 3-3. ターミナルでTL500ネットワークのIPアドレス設定
- 3-4. OpenShift管理コンソールを開く

4. CodeReady Workspaces 作成

- 4-1. CodeReady Workspacesを開く
- 4-2. CodeReady Workspacesのログイン

5. 技術演習環境作成

- 5-1. Workspaceに構築スクリプトをcloneします。
- 5-2. GitLabプロジェクトの作成
- 5-3. 環境変数の設定
- 5-4. Ubiquitous-journeyのインストール
- 5-5. Nexus, Keycloak, PetBattleのインストール
- 5-6. Tecktonのインストール
- 5-7. SonarQubeのインストール
- 5-8. Allureのインストール
- 5-9. 受講者環境の削除

参考文献

本文書は、GLSインストラクターがTL500用の演習環境を構築する手順を示します。

1. TL500 VLab リクエスト

LMS上でのオフリングの作成および受講者リストが作成されたら、GLSインストラクターがAPAC VLab SupportチームにTL500 VLabリクエストをメールで送信します。

VLabの構築が完了するとSupportチームより連絡が来ます。これ以降、rol.redhat.comでTL500ダッシュボードにアクセスできるようになります。

2. インストラクターサーバー設定

VLabのインストラクターサーバーにOpenShiftクラスターを構築します。この構築作業は[Create] ボタンを押すだけで自動で実施されるので何もする必要はありません。ただし、受講者のサーバーからOpenShiftクラスターにネットワーク接続をするためのネットワーク設定を各受講者サーバーで実施する必要があります(後述)。

2-1. インストラクター仮想マシン生成

TL500ダッシュボードのLabsページから、インストラクターの[Create] ボタンを押して、インストラクター用の仮想マシンを生成します。

2-2. インストラクターWorkstationのコンソールを開く

2-3. WorkstationのターミナルでTL500ネットワークのIPアドレス取得

utilityサーバーにsshでログインして、`get_tl500_network.sh` コマンドを実行します。このコマンドの結果として出力されたIPアドレスは受講者サーバーに設定します(後述)。

```
$ sudo lab@utility  
$ sudo ./get_tl500_network.sh  
10.82.0.109
```

3. 受講者サーバー設定

TL500では受講者を4チームに分けて演習を実施します。4チーム分の技術演習環境構築をおこなうためにVLabに登録された受講者の4つの環境を使います(どの受講者を選ぶかは自由)。各受講者の仮想環境でCodeReady Workspacesを開き、そこでそのチーム用の演習環境を構築します。演習で使うCodeReady Workspacesは受講者サーバーではなく、OpenShift内に構築されます。4チーム分の技術演習環境が構築できたら、作業に使用した受講者の仮想マシンはDeleteします。

3-1. 受講者仮想マシン生成

3-2. 受講者Workstationのコンソールを開く

3-3. ターミナルでTL500ネットワークのIPアドレス設定

[2-3](#) で取得したIPアドレスを、`set_tl500_network.sh`コマンドを使って受講者Workstationに設定する。これ以降、OpenShift Webコンソールにアクセスできるようになる。

```
$ sudo ./set_tl500_network.sh --ip=10.82.0.109
```

3-4. OpenShift管理コンソールを開く

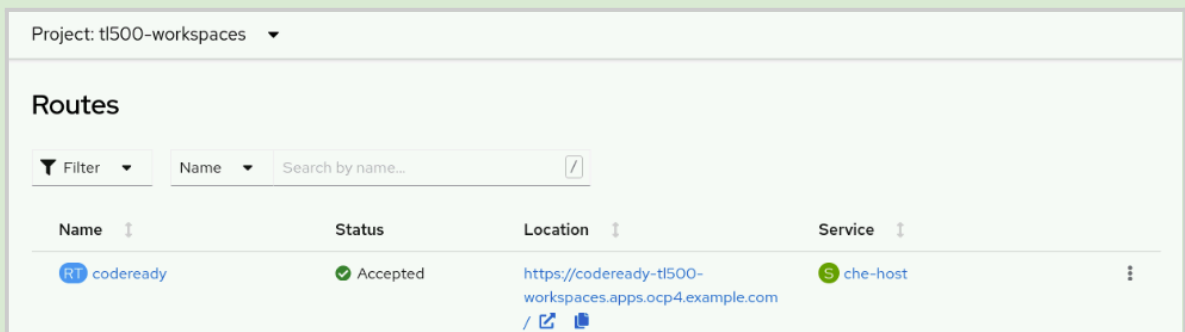





受講者のWebブラウザで以下のURLを開きます。ログインはlab0X/lab0X (例: lab01/lab01)

<https://console-openshift-console.apps.ocp4.example.com>

4. CodeReady Workspaces作成

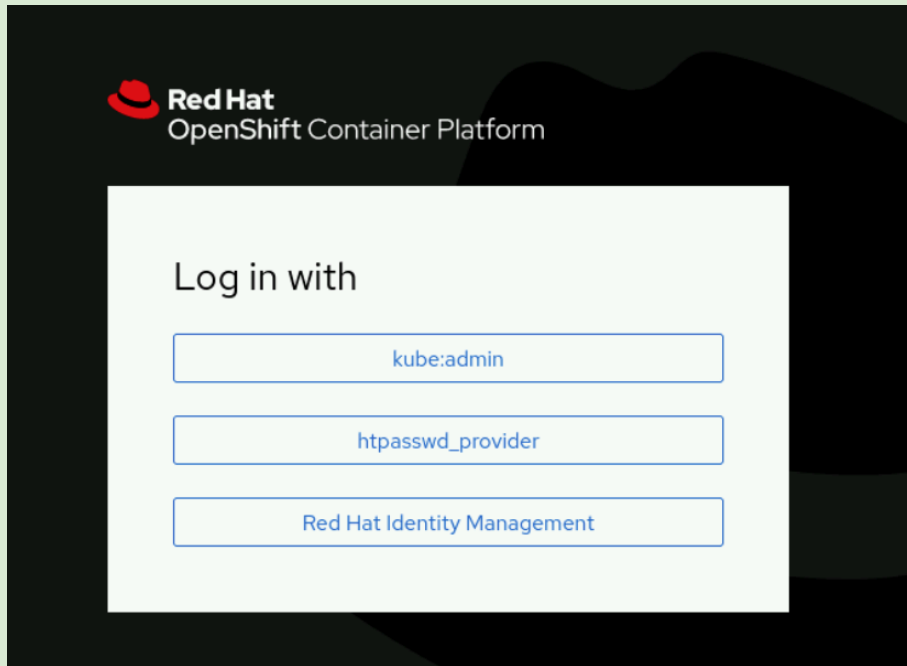
4-1. CodeReady Workspacesを開く

OpenShift Webコンソール上でtl500-workspacesプロジェクトを探してcodereadyルートを表示しLocationのURLを開きます。

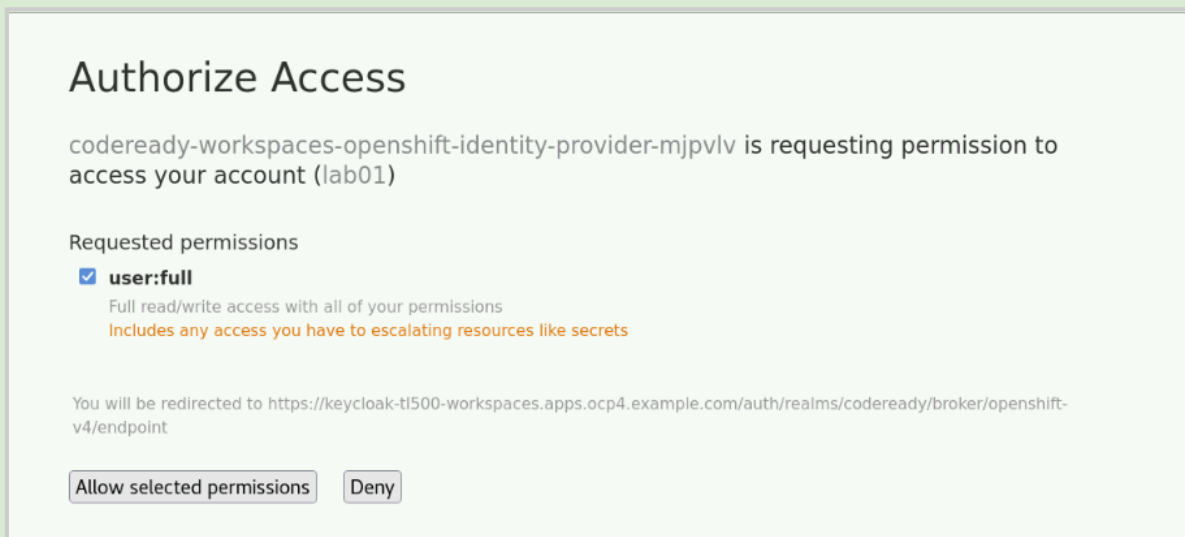
Project: tl500-workspaces ▼			
Routes			
▼ Filter ▼	Name ▼	Search by name... 	
Name ↑	Status	Location ↑	Service ↑
 codeready	✔ Accepted	https://codeready-tl500-workspaces.apps.ocp4.example.com  	 che-host 

4-2. CodeReady Workspacesのログイン

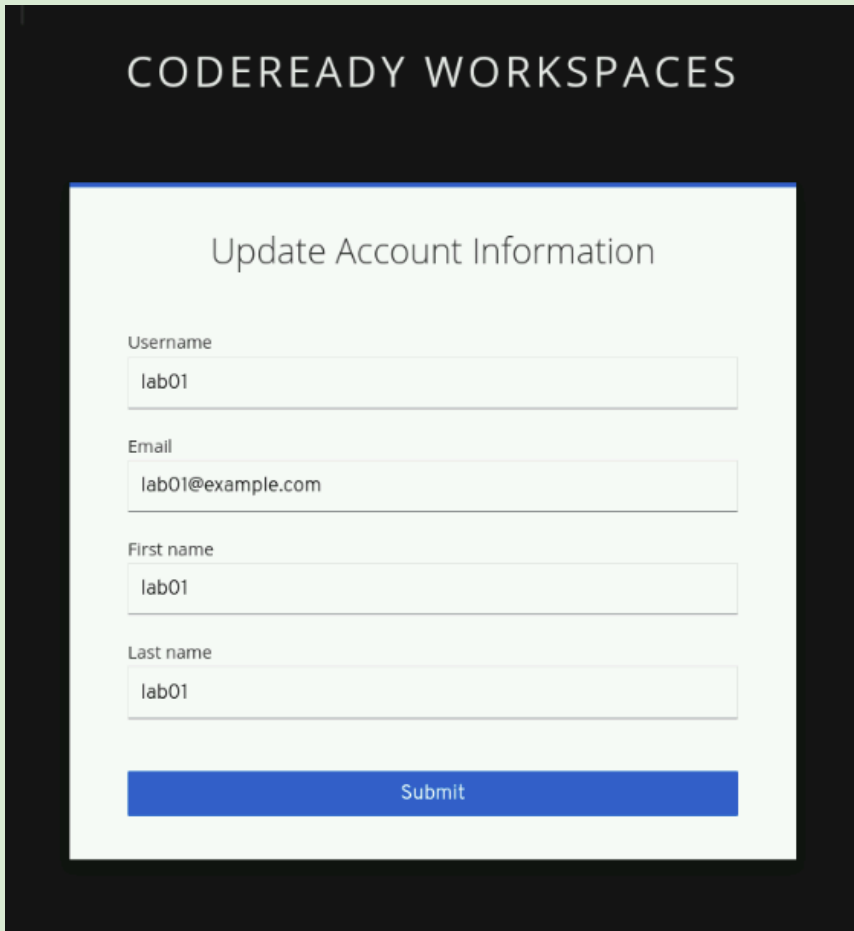
1. CodeReady Workspaceのログイン画面で、[Red Hat Identity Management] をクリックします。OpenShift Webコンソールのログイン画面が開きますので、OpenShiftと同じように、UsernameとPasswordを入力してください。



1. Authorized Access の画面では、[Allow selected permissions] のボタンを押します。



2. CodeReady Workspacesのユーザープロフィールを設定します。lab01の部分は、チームに割り当てられたユーザー名に置き換えてください。



CODEREADY WORKSPACES

Update Account Information

Username
lab01

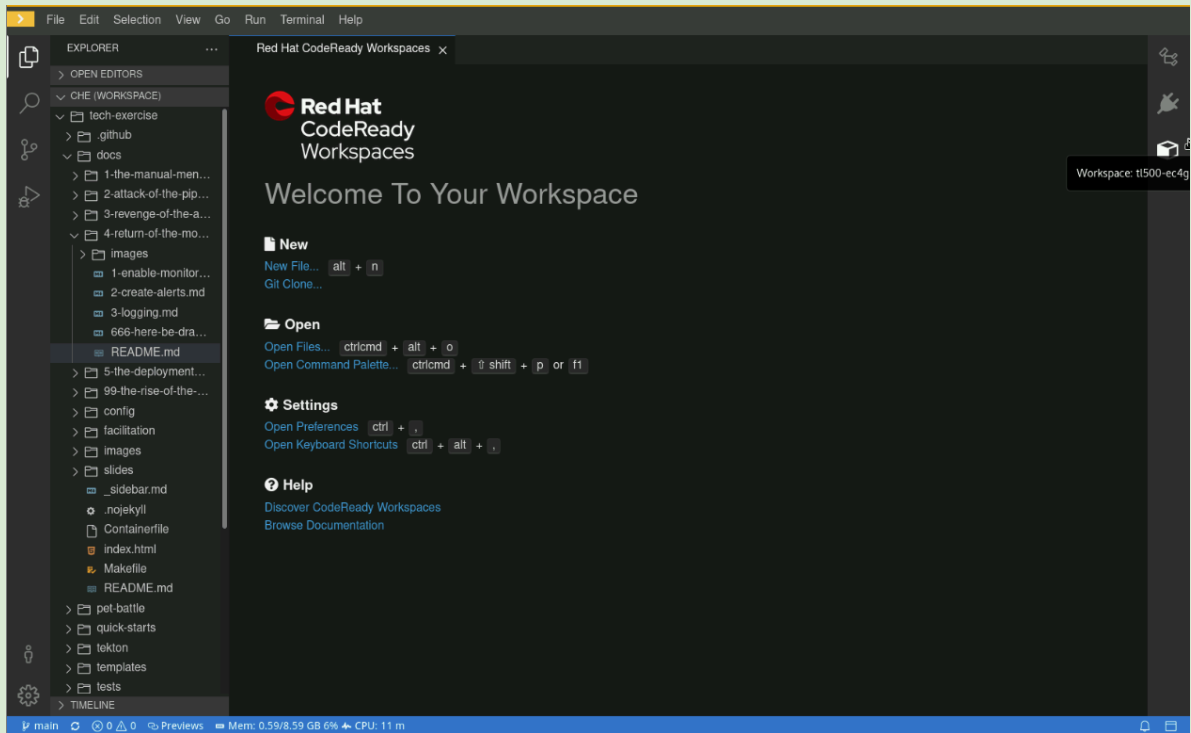
Email
lab01@example.com

First name
lab01

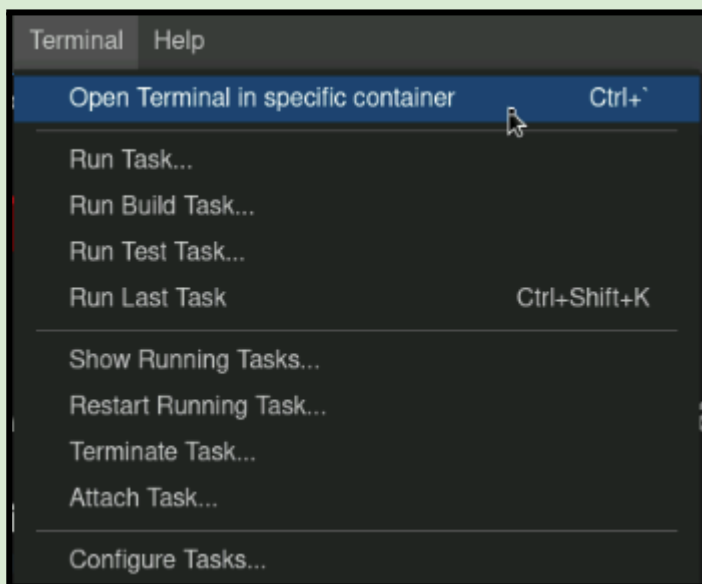
Last name
lab01

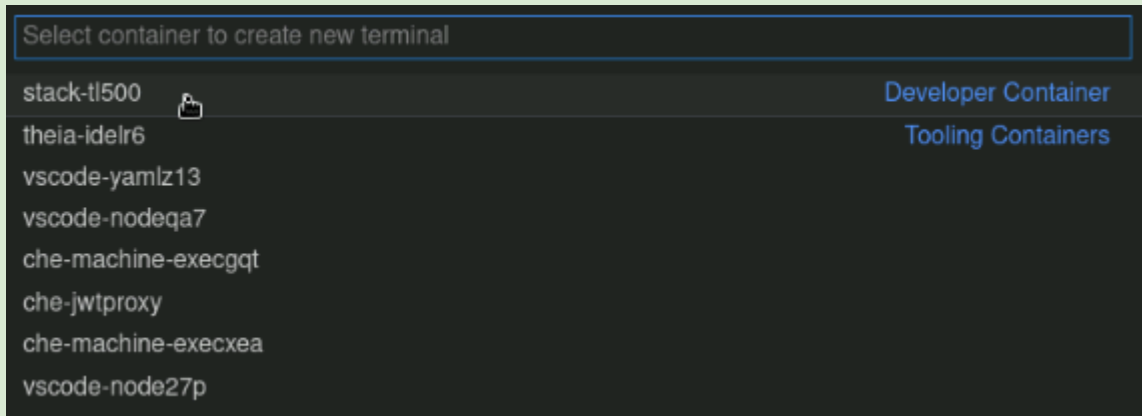
Submit

3. Workspaceの構築をします。CodeReady WorkspacesのDevFileとして以下を指定してください。
<http://utility.lab.example.com:8080/tl500/tl500-devfile.yaml>
4. Workspaceの構築が成功するとVSCodeの画面が表示されます。画面右下に"Do you trust the authors of the files in this workspace?" という質問が表示されるので、[YES, I trusted the authors] のボタンを選択します。
5. [Red Hat CodeReady Workspaces] の表示が見えれば、これでアプリケーションの設定やソースコードを変更できる環境の準備は完了です。これはVS Code for the Webというアプリケーションの開発環境です。

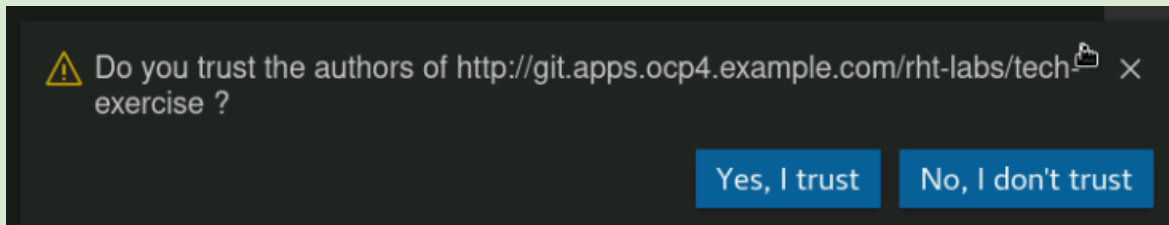


6. 次に左上のメニューから [Terminal] - [Open Terminal in specific container] を選択してターミナルを開きます。コンテナの種類は `stack-tl500` を選択してください。





7. 画面右下にDo you trust the authors ...というメッセージが現れたら [Yes, I trust] を選択します。



5. 技術演習環境作成

4チーム用の環境を構築するため、4つの受講生の仮想マシンを使います（一つの環境で OpenShiftログイン・ログアウトを繰り返しながら実施することも可能ですが、事故が起こりやすいので、この手順書では4つの仮想マシンを使います）。

以下の手順はひとつのチーム用の設定です。チーム名、ユーザー名、パスワードを変更しながら、以下の手順を4回繰り返します。

team1用

チーム名	team1
ユーザー名	lab01
パスワード	lab01

team2用

チーム名	team2
------	-------

ユーザー名	lab02
パスワード	lab02

team3用

チーム名	team3
ユーザー名	lab03
パスワード	lab03

team4用

チーム名	team4
ユーザー名	lab04
パスワード	lab04

5-1. Workspaceに構築スクリプトをcloneします。

```
cd /projects
git clone https://github.com/fminamot/tl500-setup.git
cd tl500-setup/scripts
chmod u+x *.sh
```

以下、scripts/README.md書かれた手順に従って構築します。

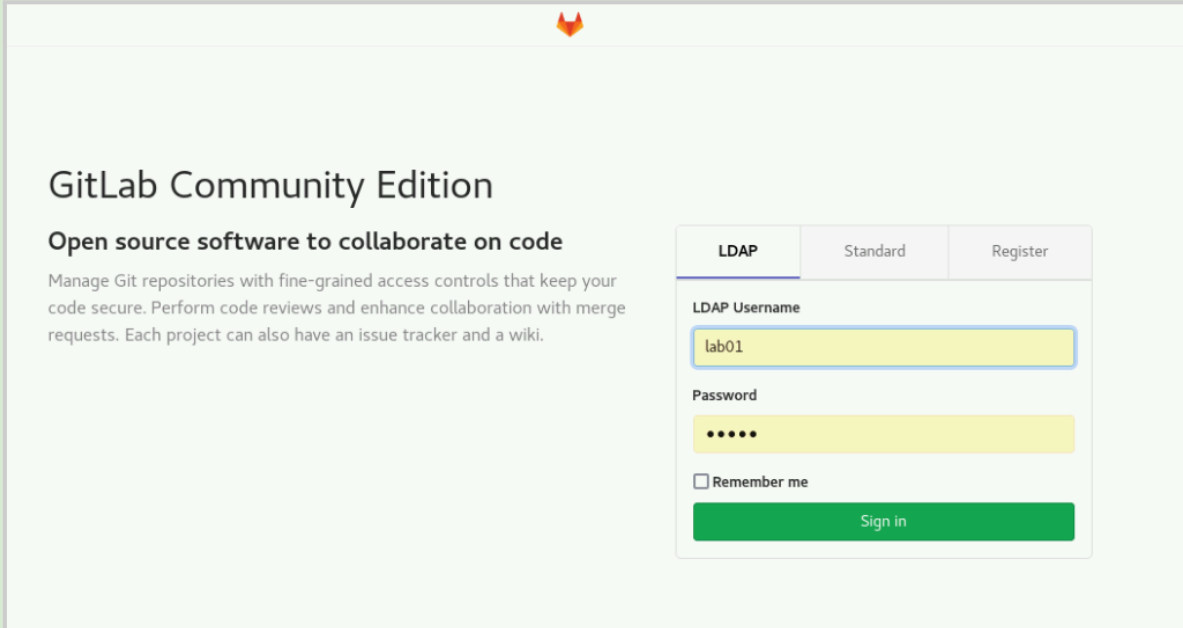
5-2. GitLabプロジェクトの作成

1. basicスクリプトを編集します。チーム名、ユーザー名、パスワードを必ず編集してください。
2. GitLabサーバーのURLを取得します。GitLabはOpenShift環境で一つだけで共通なので、各環境でURLは同じものになります。

```
source basic
```

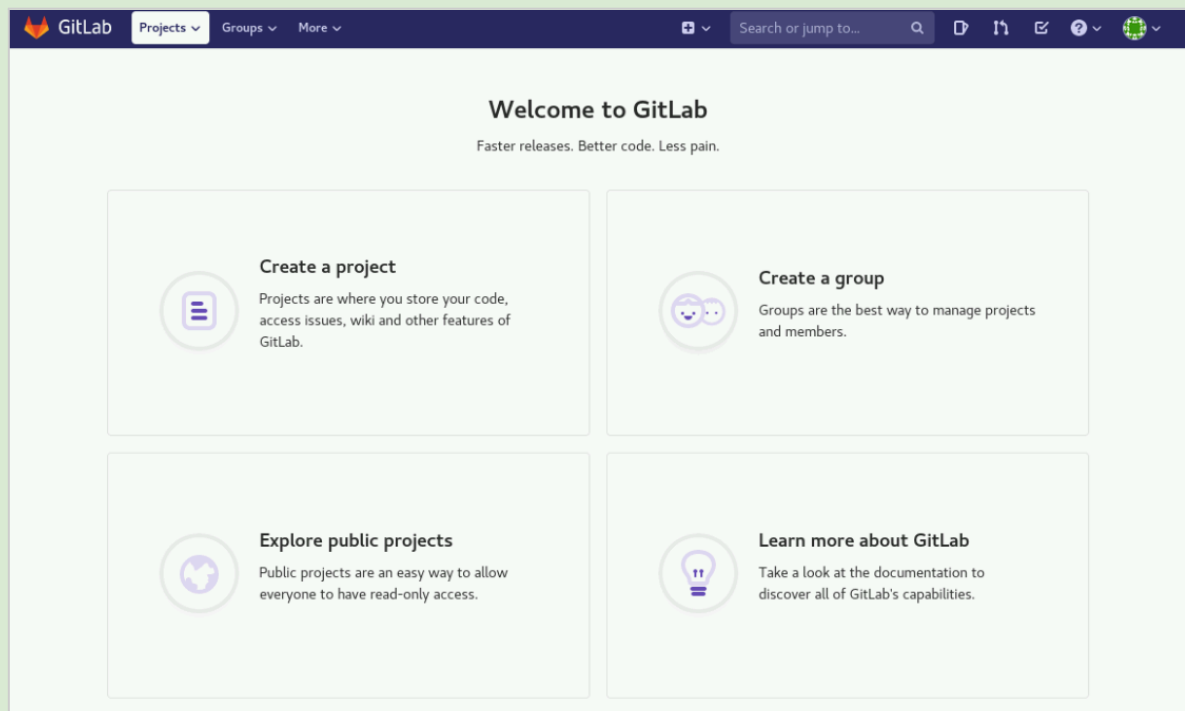
```
echo "https://${GIT_SERVER}"
```

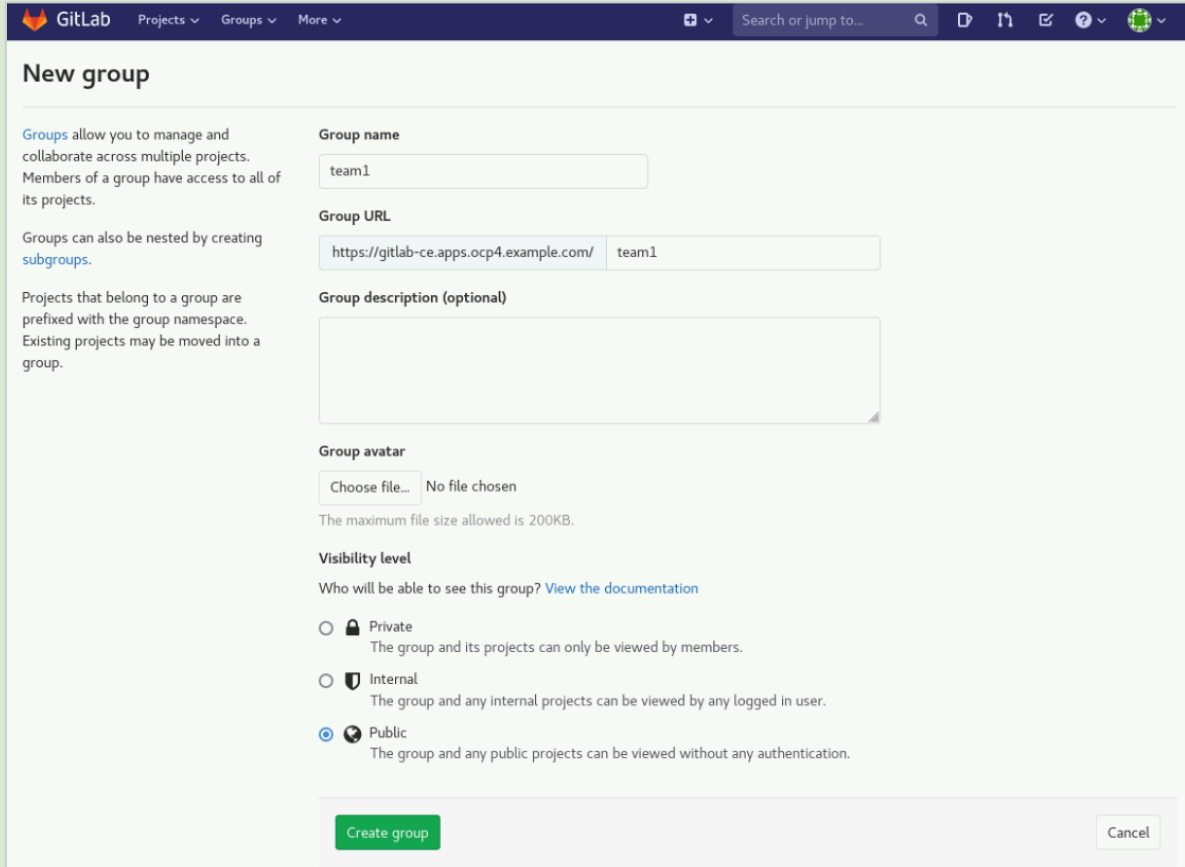
3. GitLabにログインします。ログインユーザー名はOpenShiftのログインアカウントと同じでlab0X/lab0Xになります。



The screenshot shows the GitLab Community Edition login page. The header includes the GitLab logo and the text "GitLab Community Edition". Below this, it says "Open source software to collaborate on code" and provides a brief description of GitLab's features. On the right side, there are three tabs: "LDAP", "Standard", and "Register". The "LDAP" tab is selected. Below the tabs, there are input fields for "LDAP Username" (containing "lab01") and "Password" (masked with dots). There is also a checkbox for "Remember me" and a green "Sign in" button.

4. [Create a group] を選択して、チーム名と同じ名前のpublicグループを作成します。





New group

Groups allow you to manage and collaborate across multiple projects. Members of a group have access to all of its projects.

Groups can also be nested by creating [subgroups](#).

Projects that belong to a group are prefixed with the group namespace. Existing projects may be moved into a group.

Group name

team1

Group URL

https://gitlab-ce.apps.ocp4.example.com/ team1

Group description (optional)

Group avatar

Choose file... No file chosen

The maximum file size allowed is 200KB.

Visibility level

Who will be able to see this group? [View the documentation](#)

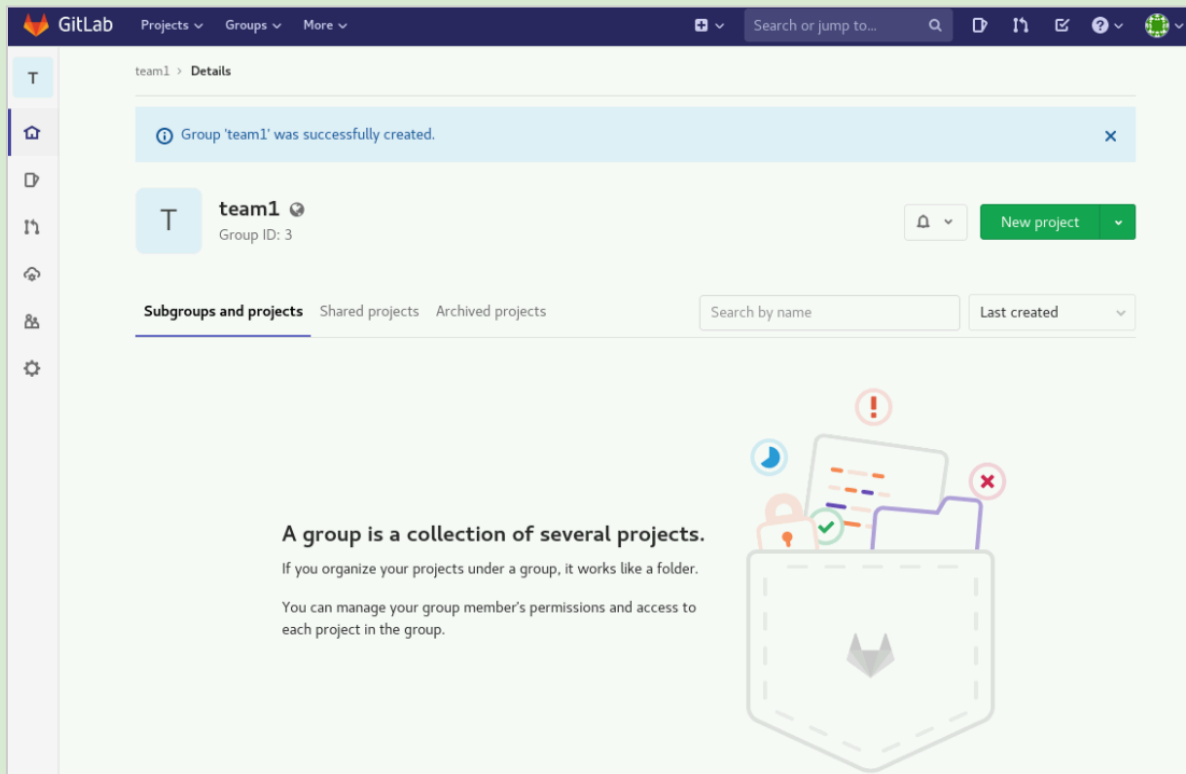
☐ Private
The group and its projects can only be viewed by members.

☐ Internal
The group and any internal projects can be viewed by any logged in user.

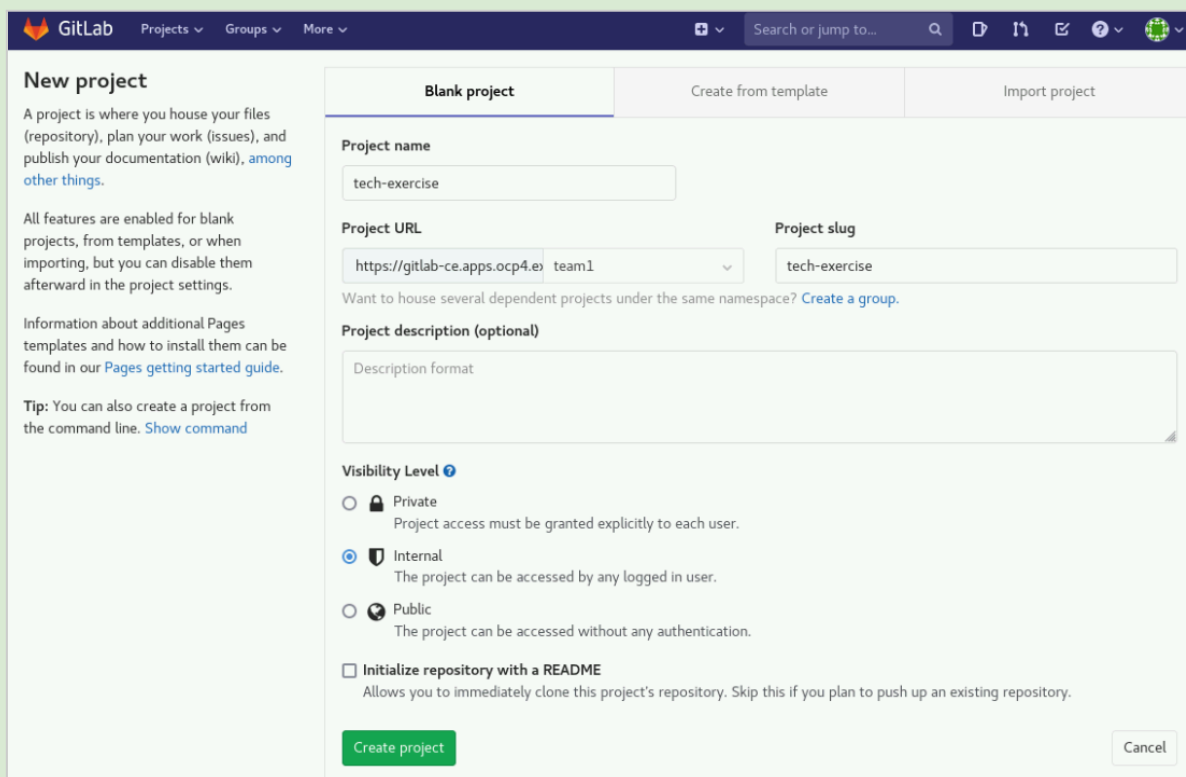
☒ Public
The group and any public projects can be viewed without any authentication.

Create group Cancel

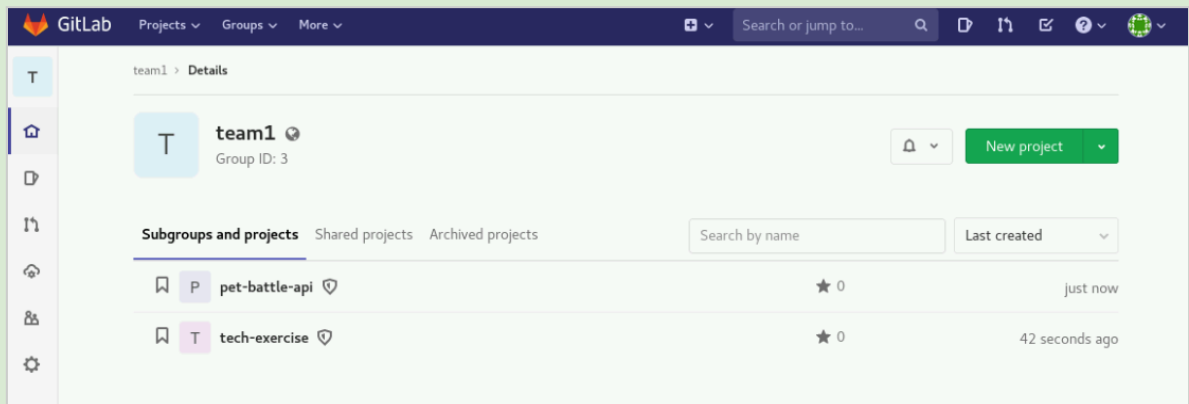
5. 今作成したグループのページにおいて、[New project] ボタンを押して、tech-exercise と pet-battle-api という2つのプロジェクトを作ります。



プロジェクト作成時に、Visibility LevelでInternalを指定します。



同様な手順で `pet-battle-api` についてもプロジェクトを作成します。2つのプロジェクトを作成後には、以下のように、`team1`グループの下に2つのプロジェクトが表示されます。



5-3. 環境変数の設定

以下の環境設定では `CodeReady Workspaces`のターミナルを使います。

1. `install-basic.sh` を実行して環境変数を設定します。

```
./install-basic.sh
```

2. このスクリプトによって環境変数は、`~/.zshrc` と `~/.bashrc` に設定されます。カレントシェルで変数の内容を確認できるように、`source ~/.zshrc` をしておきます。

```
source ~/.zshrc
echo $TEAM_NAME
```

5-4. Ubiquitous-journeyのインストール

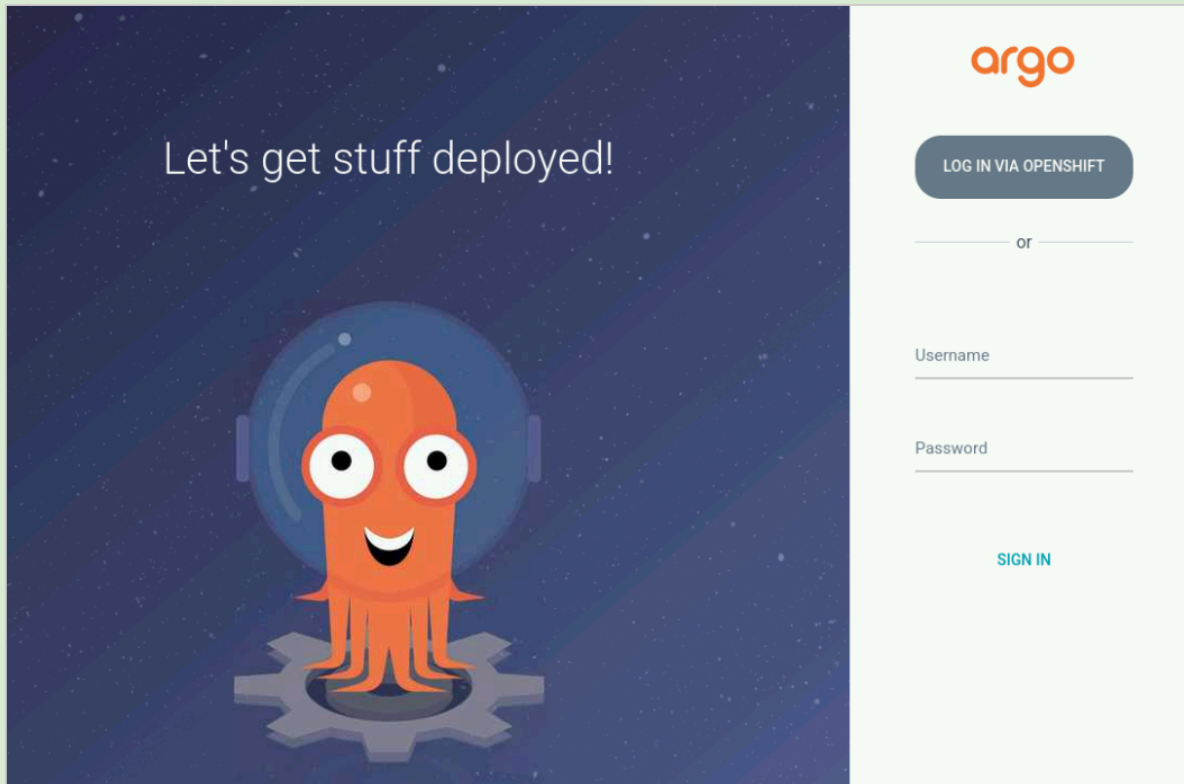
1. `./install-uj.sh` を実行してArgoCDとUbiquitous-journeyをインストールします。このスクリプトでArgoCDがインストールされます。

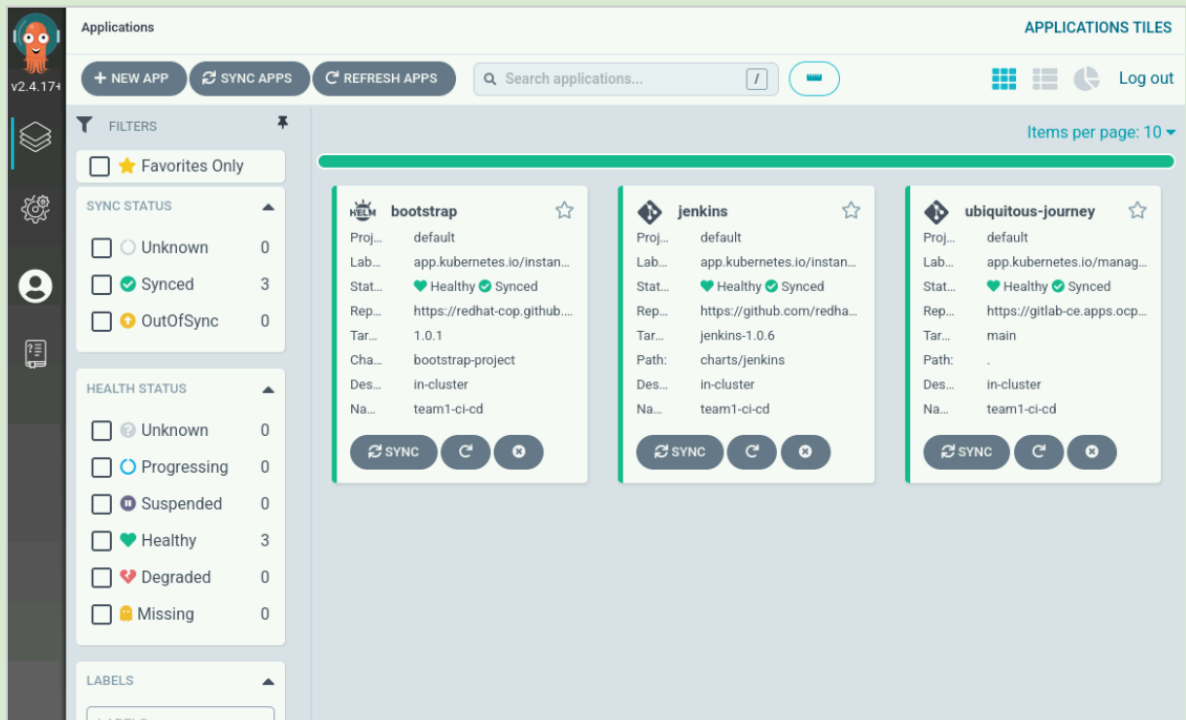
```
./install-uj.sh
```

スクリプト実行途中で `oc login` するところで `Use insecure connections? (y/n)` をきいてくるので `y` を入力します。

2. スクリプトが終了したら、以下のコマンドによって ArgoCD UIのURLが表示されるので、Argo CDにログインしてください (ユーザー名、パスワードはOCPと同じ)。

```
echo https://$(oc get route argocd-server --template='{{ .spec.host }}' -n  
${TEAM_NAME}-ci-cd)
```

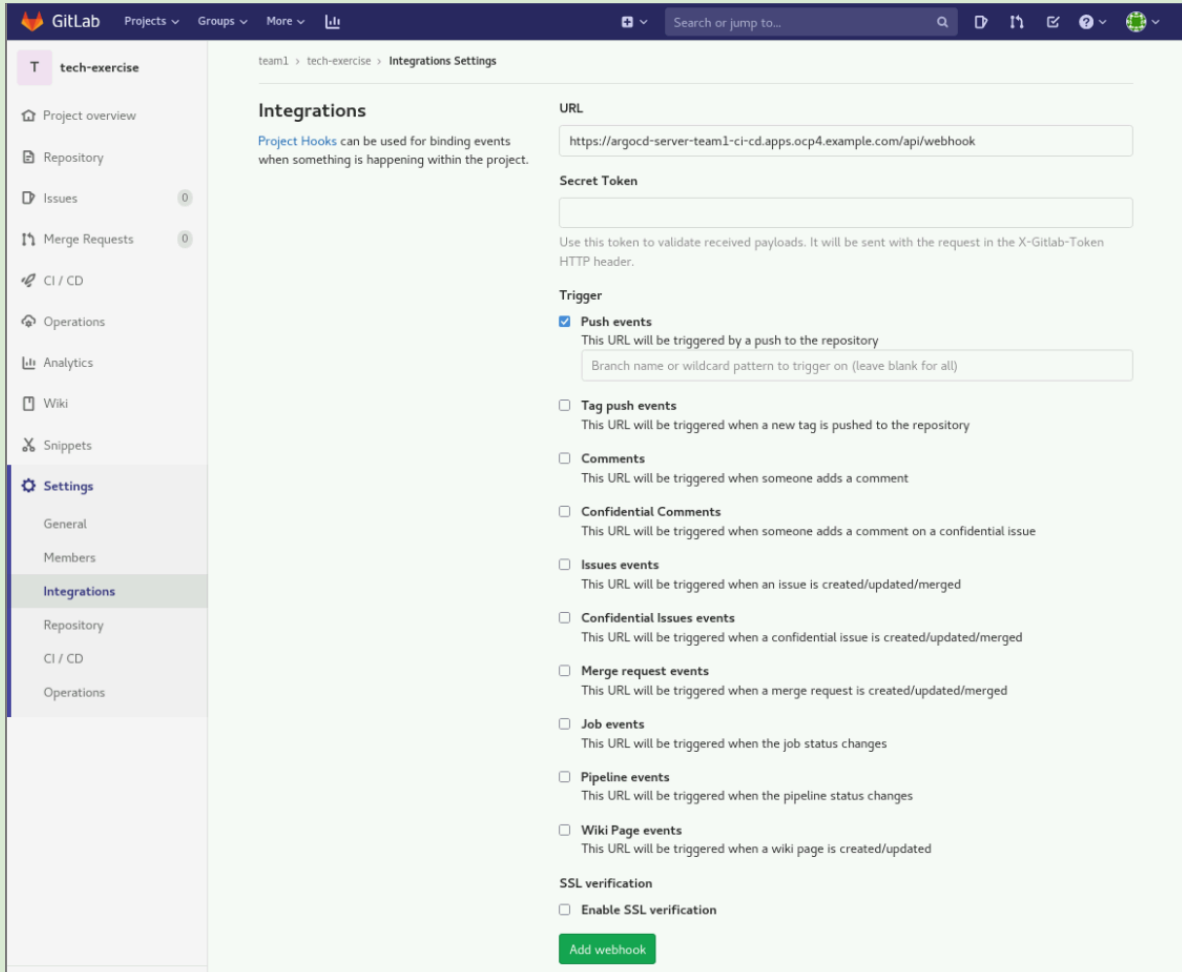




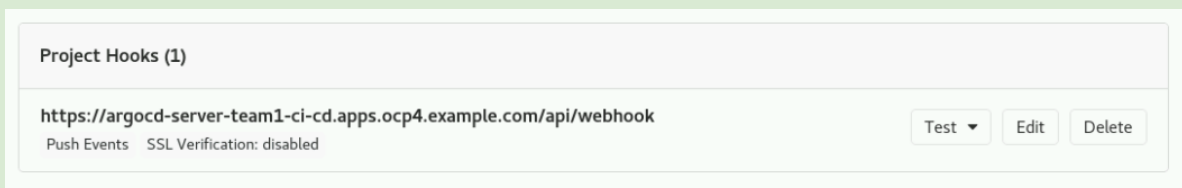
3. tech-exercise プロジェクトにWebHookを追加します。これによって当該プロジェクトへのコミットがArgoCDに反映されます。WebHookのURLは以下のコマンドで表示されます。

```
echo https://$(oc get route argocd-server --template='{{ .spec.host
}}'/api/webhook -n ${TEAM_NAME}-ci-cd)
```

tech-exercise プロジェクトにおいて [Settings] > Integration]を選択し、上記の WebHook URLをペーストします。**SSL verification**のチェックは外しておいてください。[Add Webhook] ボタンを押すと WebHook が登録されます。



登録済み WebHookの [Test] > [Push Events] を押すと、WebHookイベントが ArgoCDに飛びます。これはテストをするときに便利です。

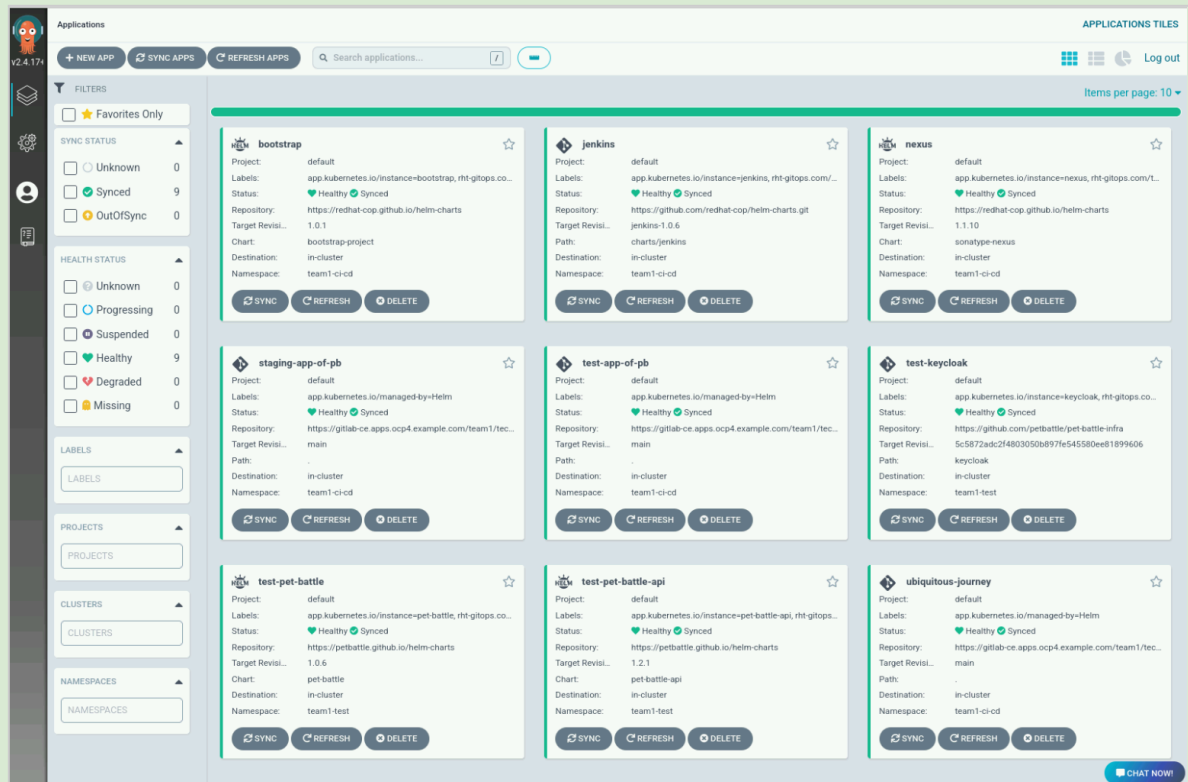


5-5. Nexus, Keycloak, PetBattleのインストール

1. `./install-uj2.sh` スクリプトを実行します。このスクリプトによって、Nexus, Keycloak, PetBattleがArgoCD経由でインストールされます。

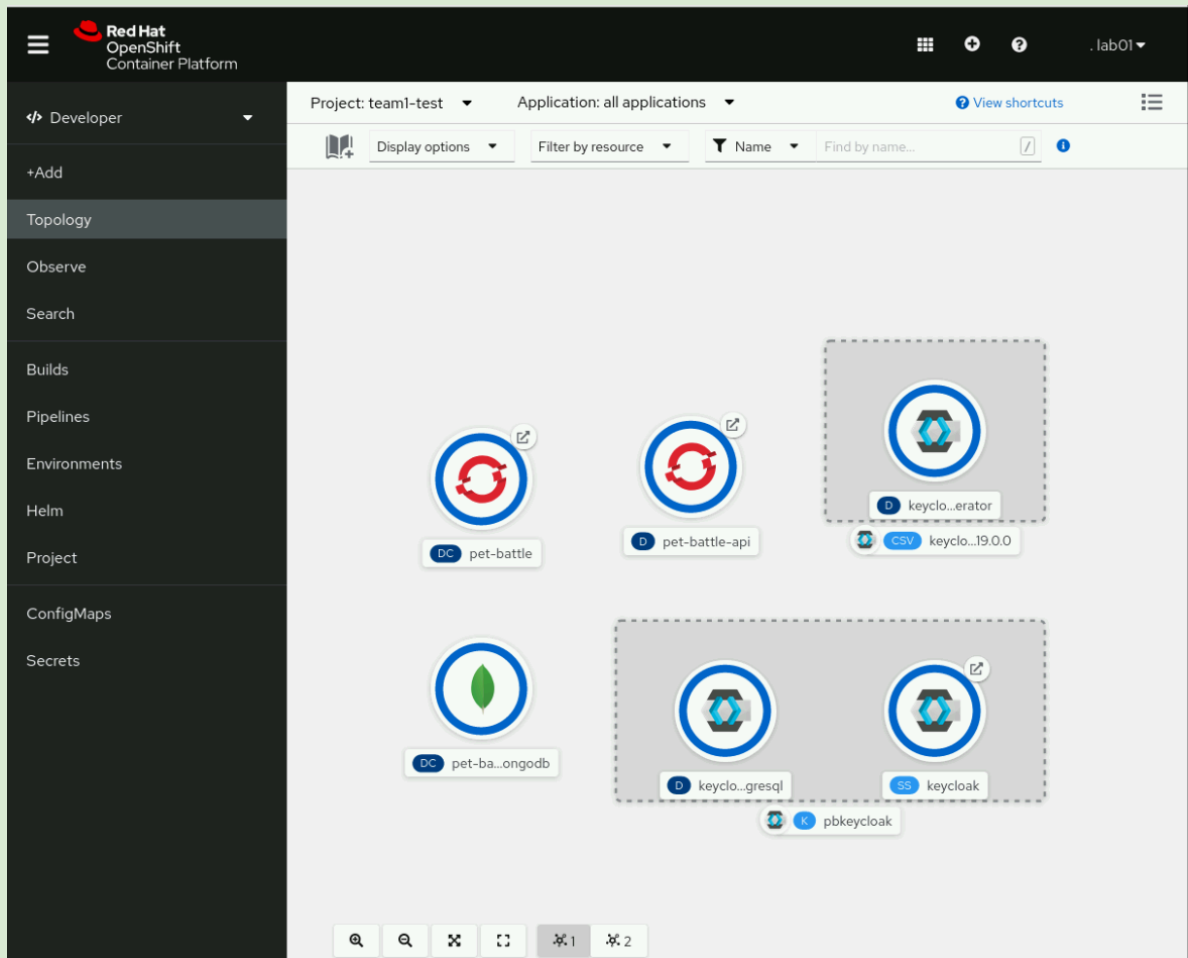
```
./install-uj2.sh
```


2. tech-exerciseプロジェクトに登録済みの WebHookの [Test] > [Push Events] を押して、ArgoCDの更新にトリガーをかけます (TODO: これを実施するShell Scriptを作成)。
3. ArgoCDのUI画面を開きます。ArgoCD上ですべてがHealy, Synchedの状態になるまで待ちます。Nexusのデプロイに3~5分くらいの時間がかかります。

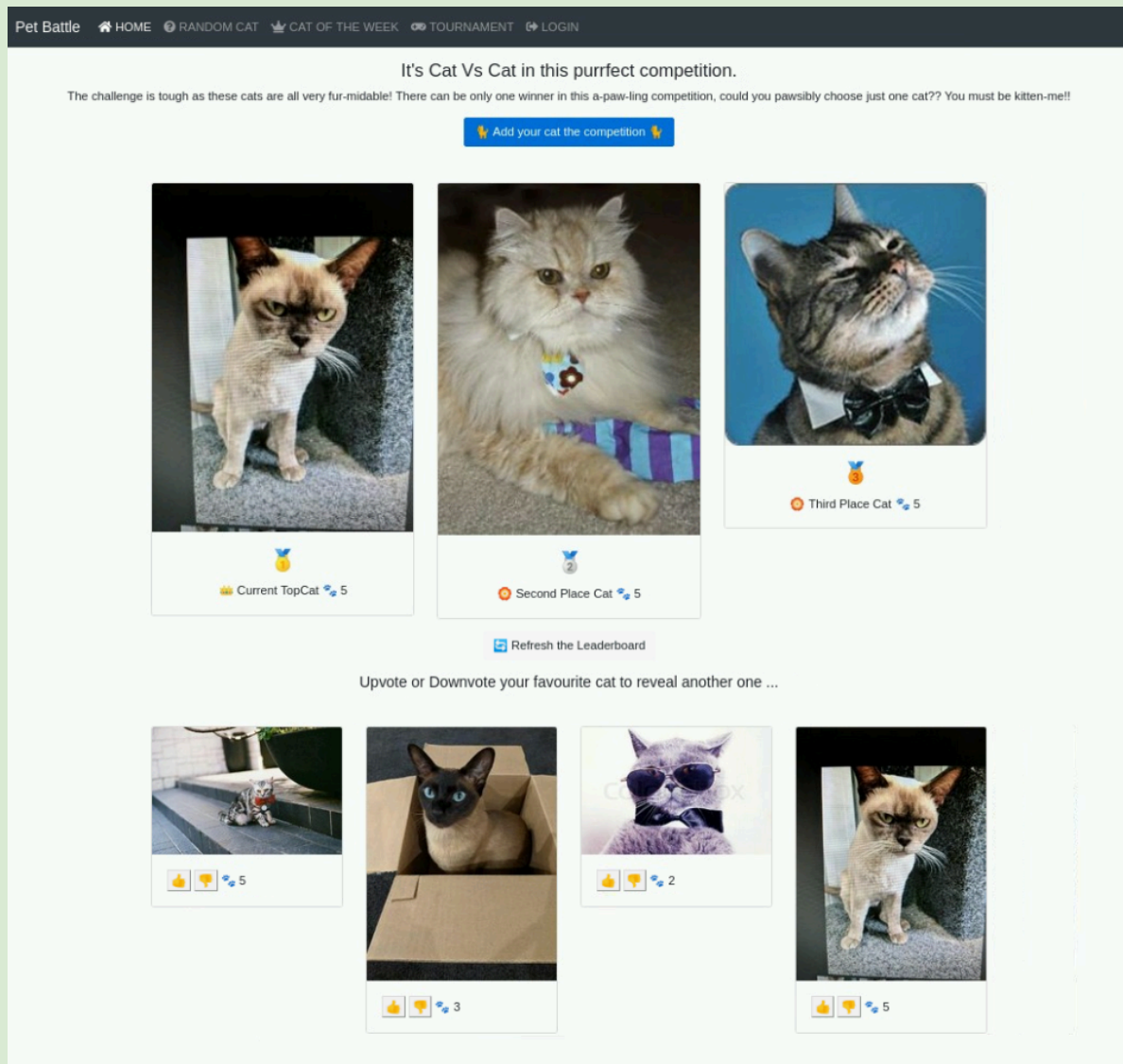


5-6. PetBattleの動作確認

1. OpenShift WebコンソールでDeveloper Perspectiveを開きます。
左側のメニューから [Topology] を選択し、右側上部のプロジェクト選択メニューから、<チーム名>-testプロジェクトを選択します。以下の例では、チーム名がteam1の場合で、プロジェクト名はteam1-testプロジェクトになります。



2. PetBattleのRouteのURLを開き、アプリが動作することを確認します。



5-6. Tektonのインストール

1. `install-tekton.sh` を実行してTektonパイプラインをインストールします。

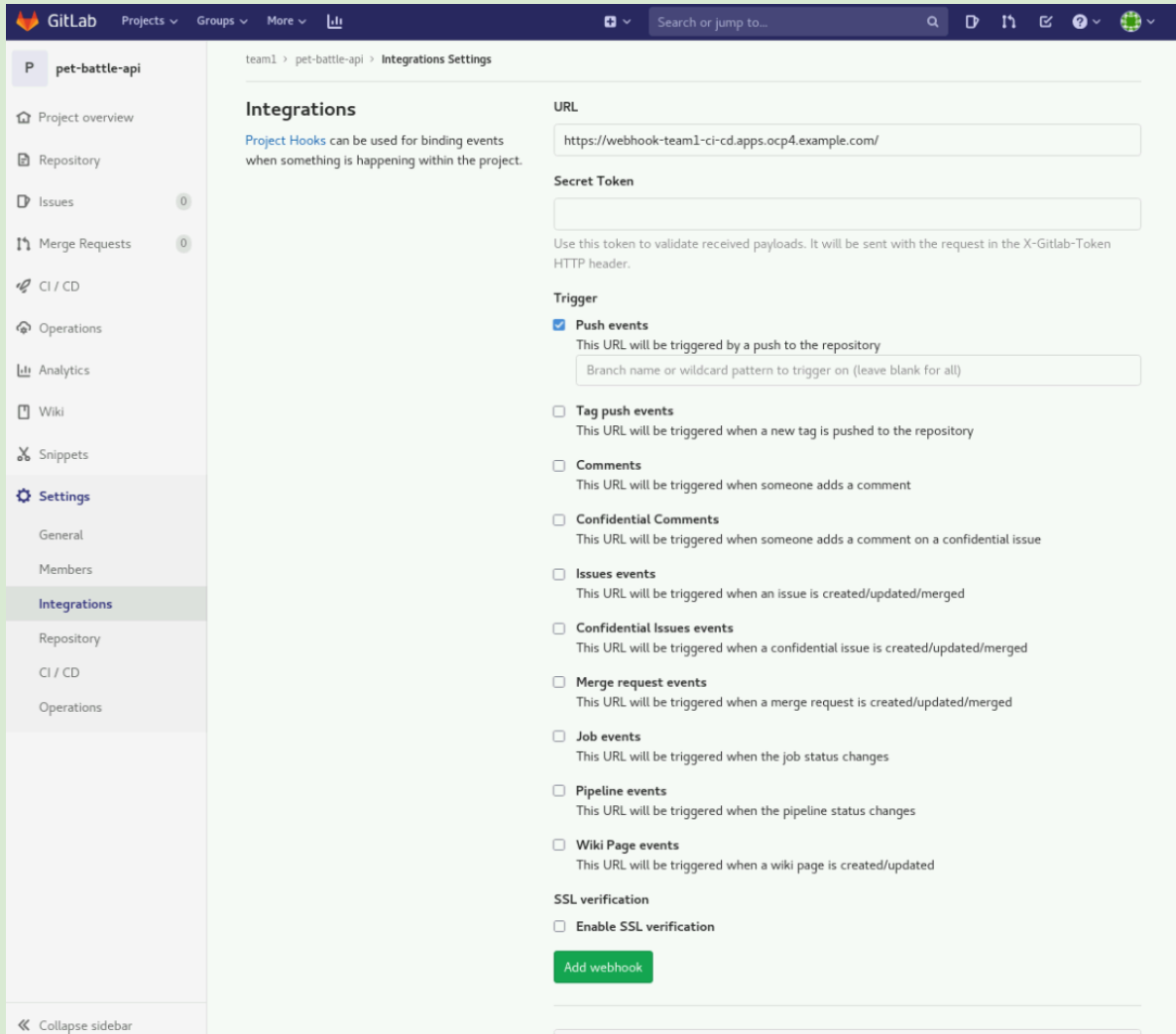
```
$ ./install-tekton.sh
```

2. `pet-battle-api` プロジェクトにパイプラインを起動するWebHookを設定します。

```
$echo https://$(oc -n ${TEAM_NAME}-ci-cd get route webhook --template='{{.spec.host}}')
```

`pet-battle-api` プロジェクトにおいて [Settings] > Integration]を選択し、上記の WebHook URLをペーストします。**SSL verification**のチェックは外しておいてください。

[Add Webhook] ボタンを押すと WebHook が登録されます。



team1 > pet-battle-api > Integrations Settings

Integrations

Project Hooks can be used for binding events when something is happening within the project.

URL

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

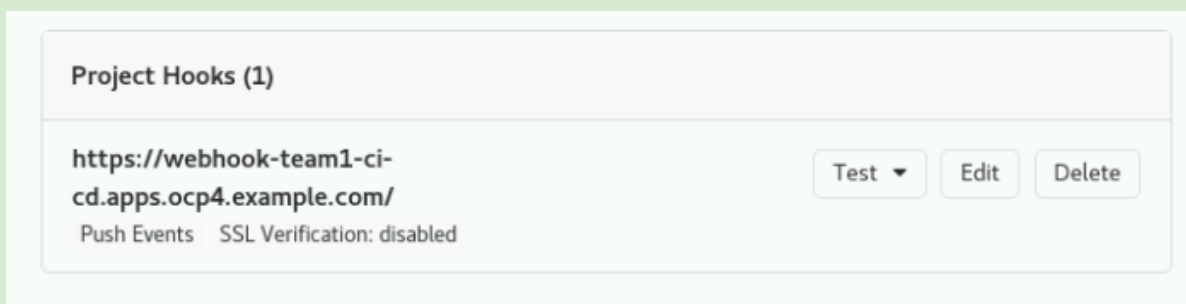
Trigger

- ☒ **Push events**
This URL will be triggered by a push to the repository
- ☐ **Tag push events**
This URL will be triggered when a new tag is pushed to the repository
- ☐ **Comments**
This URL will be triggered when someone adds a comment
- ☐ **Confidential Comments**
This URL will be triggered when someone adds a comment on a confidential issue
- ☐ **Issues events**
This URL will be triggered when an issue is created/updated/merged
- ☐ **Confidential Issues events**
This URL will be triggered when a confidential issue is created/updated/merged
- ☐ **Merge request events**
This URL will be triggered when a merge request is created/updated/merged
- ☐ **Job events**
This URL will be triggered when the job status changes
- ☐ **Pipeline events**
This URL will be triggered when the pipeline status changes
- ☐ **Wiki Page events**
This URL will be triggered when a wiki page is created/updated

SSL verification

- ☐ **Enable SSL verification**

Add webhook



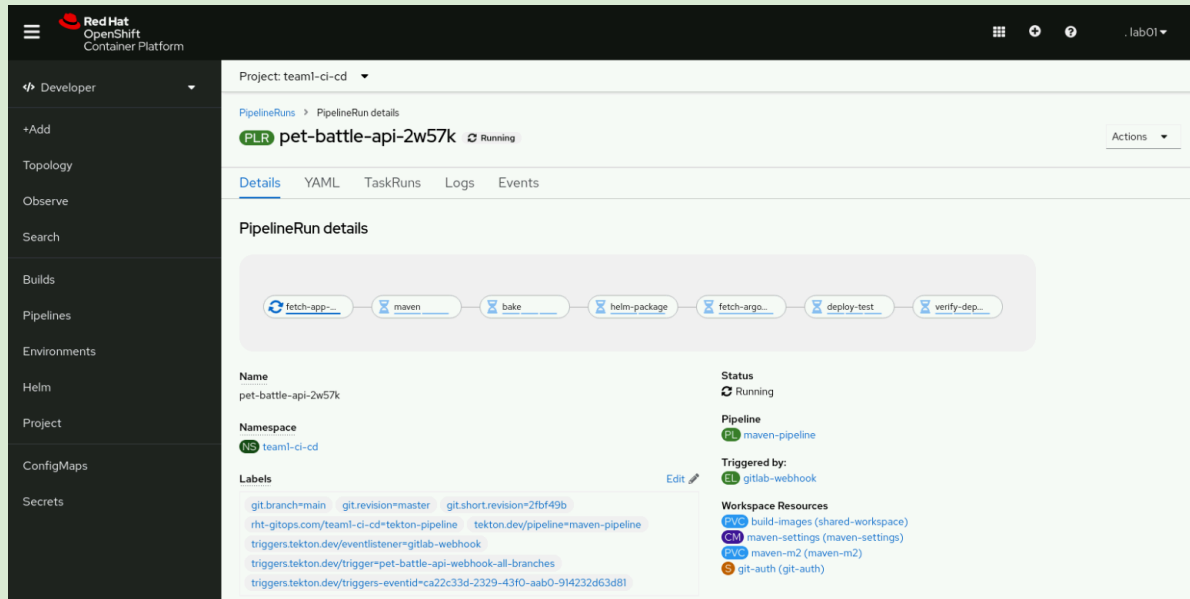
Project Hooks (1)

https://webhook-team1-ci-cd.apps.ocp4.example.com/
Push Events SSL Verification: disabled

Test **Edit** **Delete**

登録済み WebHook の [Test] > [Push Events] を押すと、WebHook イベントがパイプラインに飛び、パイプラインが実行されます。これはテストをするときに便利です。

3. Developer Perspective で <チーム名>-ci-cd プロジェクトの [Pipelines] を開きます。



4. pet-battle-api のWebHookを使ってパイプラインを実行します。

パイプラインのタスクが最後まで実行されて成功したことを確認してください。パイプラインの実行途中でエラーが出た場合は[6-1. パイプラインの失敗](#)を参照してください。

5-7. SonarQubeのインストール

1. `install-tekton.sh` を実行してSonarQubeをインストールします。

```
./install-sonarqube.sh
```

2. SonarQubeのUIは以下のURLからアクセスできます。SonarQubeのインストールには長い時間がかかるので、ArgoCDの状態をよく観察して、インストールが終了してから以下のコマンドを実行してください。

```
echo https://$(oc get route sonarqube --template='{{ .spec.host }}' -n  
${TEAM_NAME}-ci-cd)
```

5-8. Allureのインストール

1. `install-allure.sh` を実行してAllureをインストールします。

```
./install-allure.sh
```

2. AllureのUIは以下のURLからアクセスできます。

```
echo https://$(oc get route allure --template='{{ .spec.host }}' -n  
${TEAM_NAME}-ci-cd)/allure-docker-service/projects/pet-battle-api/reports  
/latest/index.html
```

5-9. 受講者環境の削除

1. パイプラインの動作が確認できたら受講者の仮想マシンを [Delete] します。構築したものはすべてOpenShiftクラスターに残るので削除しても問題ありません。

5-10. デプロイ状況の確認

1. パイプラインで使うサービスのデプロイ状況の確認

```
./watch-ci-cd.sh
```

2. コンソールの確認

```
./show-console.sh
```

6. トラブルシュート

6-1. パイプラインの失敗

1. パイプラインのYAMLの記述が間違っていると、パイプラインの実行中にエラーになります。そのような場合、以下を確認してください。
 - a. GitLabにYAMLの更新が反映されているか否かを確認します。YAML構文エラーの場合は、ArgoCDでエラーになるのでGitLabは更新されません。
 - b. ArgoCDにエラーが出ていないかを確認します。YAML構文エラーの場合、エラーメッセージが出力されます。
2. **ArgoCDの更新が遅いためにパイプラインの処理が途中で失敗することがあります** (**TL500 Instructor Guide** に記載あり)。その場合は、ファシリテーターのutilityサーバーから以下のコマンドを実行してErrorになったPodを削除し、その後、pet-battle-apiのWebHookを使ってパイプラインを再実行します。

```
[lab@utility ~]$ /home/lab/cluster_pod_cleanup.sh
```

参考文献

[1] TL500-Instructor-Guide-2023022210.pdf

[1] TL500 Technical Exercises