

Red Hat トレーニング

DevOps Culture & Practice (TL500)

技術演習手順書

2024/07/27

V1.0 Revision 0.91

Red Hat トレーニング**DevOps Culture & Practice (TL500)****0. 技術演習環境の確認 (10分)**

0-1. 技術環境の概要説明

0-1-1. 技術演習の進め方

0-1-2. 技術演習でエラーが発生したら

0-2. Red Hat Online Learning Portal

0-2-1. TL500 ダッシュボードへのアクセス

0-2-2. TL500 仮想マシンの生成と起動

0-2-3. Workstationへのログイン

0-2-4. ディスプレイ解像度の変更

0-3. OpenShiftへのアクセス

0-3-1. OpenShiftへのログイン

0-3-2. OpenShift Webコンソール

1. コンテナ化されたアプリケーション (20分)

1-1. PetBattle アプリケーションの動作確認

1-1-1. PetBattle アプリケーションへのアクセス

1-1-2. PetBattle アプリケーションのGUI画面

1-2. PetBattle アプリケーションを支えるサービス

1-2-1. PetBattle APIへのアクセス

1-2-3. Keycloakへのアクセス

2. Everything As Code (70分)

2-1. 開発環境の準備 (10分)

2-1-1. 開発環境の設定

2-1-2. Java環境設定

2-1-3. VSCodiumのインストール

2-2. Gitコマンドの基礎 (20分)

2-2-1. GitLab アプリケーション

2-2-2. Gitリポジトリの作成

2-2-3. Gitユーザーの設定

2-2-4. Gitコマンドの基本操作

2-3. PetBattle API アプリケーションのビルドから実行まで (20分)

2-3-1. PetBattle API ソースコードのダウンロード

2-3-2. PetBattle API ソースコードのビルド

2-3-3. PetBattle API コンテナーイメージ作成

2-3-4. PetBattle API MongoDB コンテナーの起動

2-3-5. PetBattle API OpenAPI UI の表示



2-4. PetBattle API Swagger UIの操作 (20分)

- 2-4-1. GET /cats: すべての猫の情報を返す
- 2-4-2. GET /cats/ids: すべての猫のIDを返す
- 2-4-3. GET /cats/{id}: 指定されたIDの猫の情報を返す
- 2-4-4. POST /cats: 猫データを生成または更新する
- 2-5. [オプション] PetBattleとPetBattle APIの間の連携

3. CI/CDパイプライン (40分)

- 3-1. PetBattle API パイプライン全体の構造を調べる (10分)
 - 3-1-1. PetBattle API パイプラインを構成するタスクを調べる。
 - 3-1-2. workspaces宣言
 - 3-1-3. params宣言
 - 3-1-4. git-cloneタスク
 - 3-1-5. mavenタスク
- 3-2. PetBattle API パイプラインを実行する (30分)
 - 3-2-1. PetBattle APIのソースコードを修正する
 - 3-2-2. PetBattle APIリポジトリへの変更のコミットとプッシュ
 - 3-2-3. PetBattle APIパイプラインの起動確認
 - 3-2-4. PetBattle APIアプリケーションの修正確認

4. テストの自動化 (40分)

- 4.1. PetBattle API パイプラインに静的テストを追加する (20分)
- 4-1. PetBattle API パイプラインにテストタスクを追加する(20分)

参考リスト

0. 技術演習環境の確認 (10分)

目的	<ul style="list-style-type: none">TL500の演習環境でPetBattleアプリケーションを起動し、機能概要を理解する
目標	<ul style="list-style-type: none">TL500の演習環境に慣れるWebコンソールからPetBattleアプリケーションを起動するPetBattleアプリケーションを操作するPetBattleアプリケーションを支えるインフラを確認する

0-1. 技術環境の概要説明

0-1-1. 技術演習の進め方

技術演習では、各チームの代表者1名(ドライバー)が演習を実施し、同チームの他のメンバーはそのサポートをします。以後、ドライバーが実施する手順になります。ドライバーは、毎日交代で実施しても構いませんが、操作のためのユーザー アカウントは同じものを使いまわします。

0-1-2. 技術演習でエラーが発生したら

技術演習の途中で、以下のような問題が発生したら技術演習担当のファシリテーターに質問してください。

- 仮想マシンの調子が悪い(文字入力ができない等)
- OpenShiftクラスターにネットワーク接続できない
- 演習手順で先に進めないようなエラーが発生した

0-2. Red Hat Online Learning Portal

技術演習環境で使うOpenShiftクラスターは、Red Hatのクラウド環境で動作し、Red Hat Online Learning Portal(以後、ROLポータル)からアクセスします。

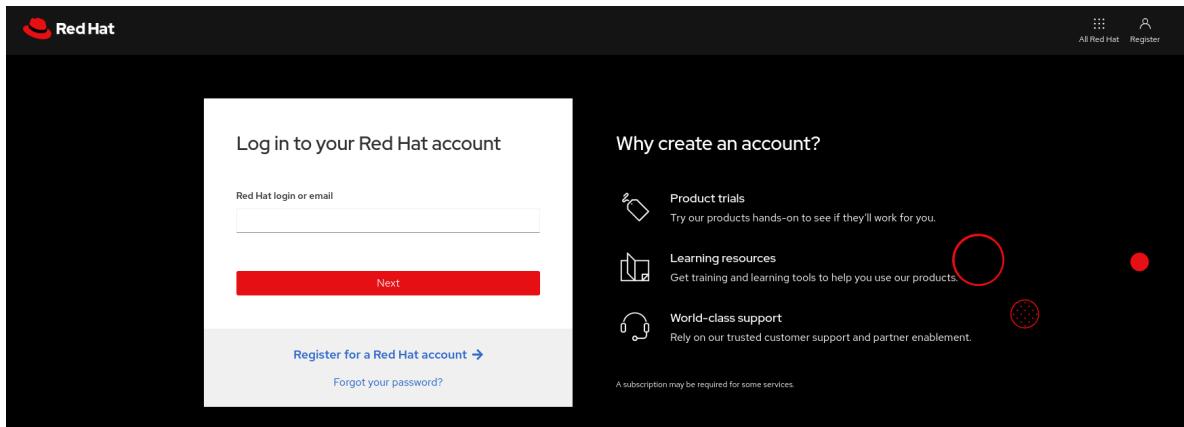
0-2-1. TL500 ダッシュボードへのアクセス

- ROLポータルにアクセスするには、Webブラウザーから以下のURLを開きます。

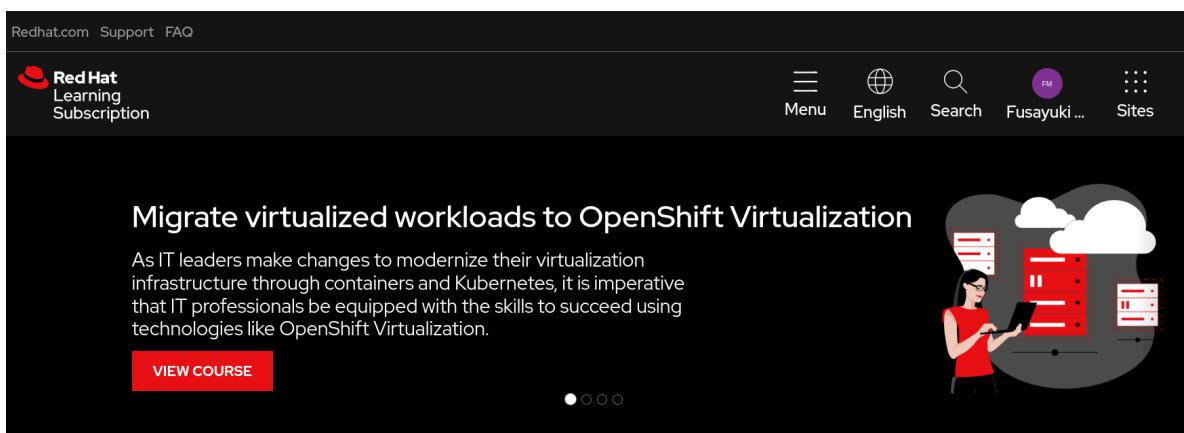
<https://rol.redhat.com/>



2. ログインが求められますので、ドライバーのRed Hatアカウントでログインしてください。
ROLに初めてアクセスする場合は、Terms & Conditionの確認が求められますので、先に進めてください。



3. ログインに成功するとROLポータルのホームページが開きます。



4. このページに [MY INSTRUCTOR-LED TRAINING CLASSES] TL500 という表示が見つかるはずです。JOINボタンを押すと、以下のようなTL500のダッシュボードが表示されます(実際の画面のボタンやタブの配置は下図とは異なります)。



Redhat.com Support FAQ

Red Hat Learning Subscription

Menu English Search Fusayuki ... Sites

Search

Class Dashboard - 20240805-NA-TL500 (62039024)

MARK CLASS COMPLETE ENTER CLASSROOM COURSE SURVEY EMAIL SURVEY

Overview Enrollments 18 Labs Resources Actions

Description Experience the possibilities of DevOps through proven open culture and practices used by Red Hat to support customer innovation. Course description DevOps Culture and Practice Enablement (TL500) is a five-day, immersive class offering students an opportunity to experience and implement cultural shifts that are utilized in many successful DevOps adoption journeys. Many agile training offerings focus on a particular framework, delivery mechanism, or technology. Instead, DevOps Culture & Practice combines the best tools from many leading frameworks to blend continuous discovery and continuous delivery with cultural and technical practices into a unique, highly-engaging experience simulating real-world scenarios and applications. To achieve the learning objectives, participants should include multiple roles from an organization. Business product owners, architects,

0-2-2. TL500 仮想マシンの生成と起動

1. TL500ダッシュボードの [Labs] タブを開いてください。
2. [Labs]タブにある[CREATE] ボタンを押して、技術演習で使う仮想マシンを生成します。

SSH Private Key & Instructions

DOWNLOAD SSH KEY

Lab Controls

WATCH TUTORIAL

CREATE

3. 仮想マシンの状態が、以下のようにすべてActiveになるまでしばらく待ちます (2~3分程度です)。



SSH Private Key & Instructions

DOWNLOAD SSH KEY

Lab Controls

WATCH TUTORIAL

DELETE STOP i

	Active	ACTION ▾	OPEN CONSOLE
bastion	Active	ACTION ▾	OPEN CONSOLE
classroom	Active	ACTION ▾	OPEN CONSOLE
workstation	Active	ACTION ▾	OPEN CONSOLE

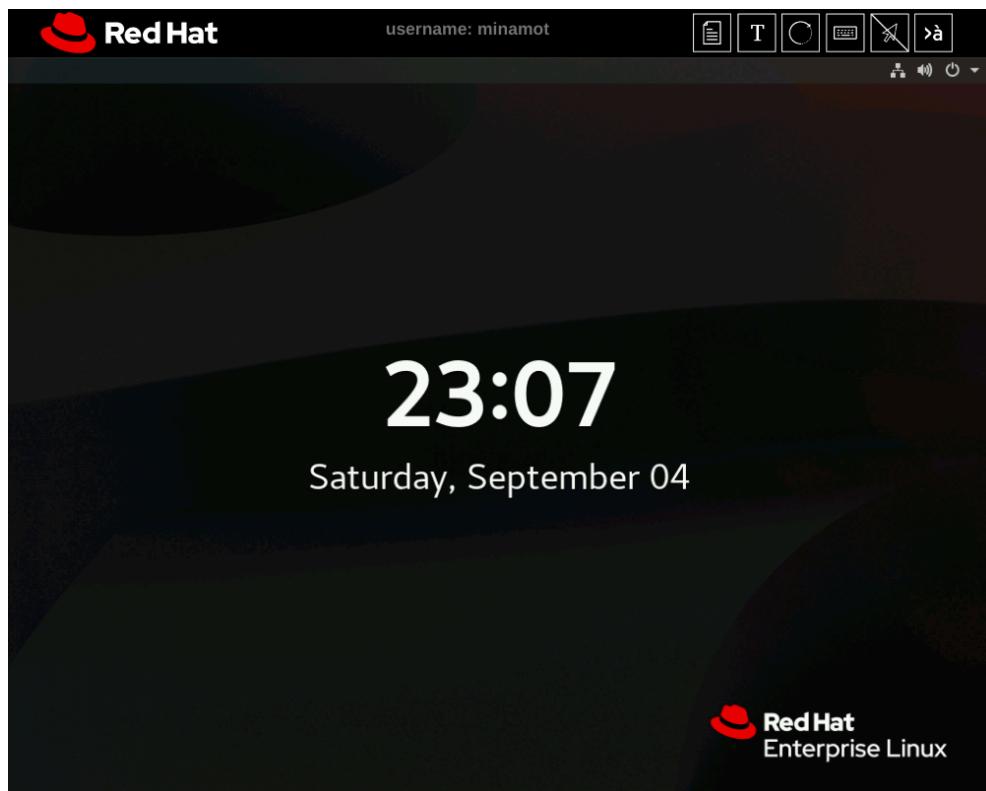
Auto-stop in 16 days.
Auto-destroy in 16 days.

[Create]によって、bastion、classroom、workstationという3台の仮想マシンが作成され、起動状態になります。起動した3台の仮想マシンは、OpenShiftクラスターにアクセスして演習を実施するための作業環境を提供します。実際に作業を実施する場所は Workstationという名前の仮想マシンです。

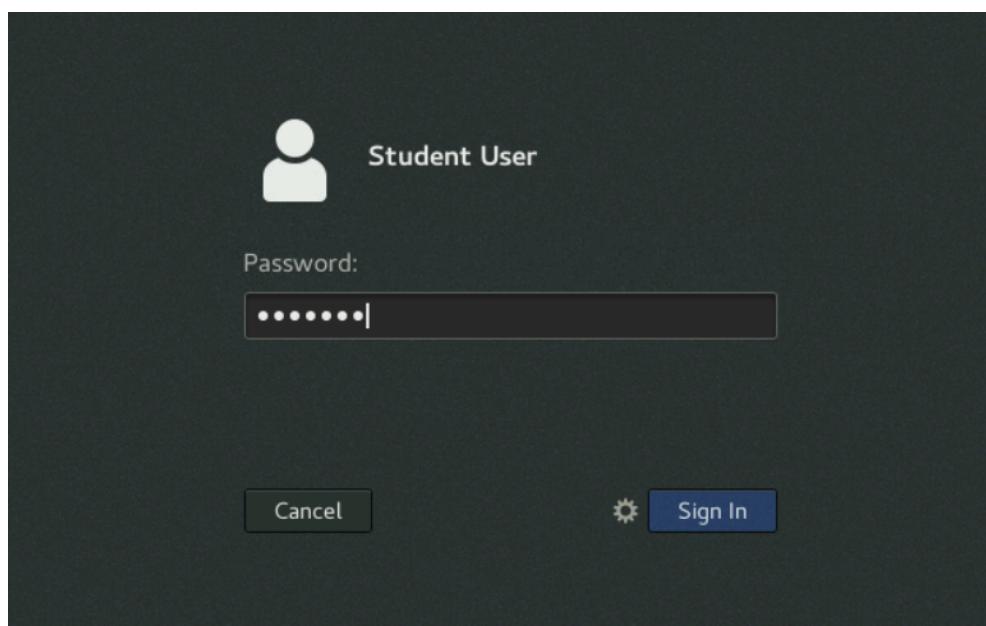
4. workstationと表示された行の[OPEN CONSOLE] ボタンを押すと、Webブラウザーの新しいタブにWorkstation仮想マシンの画面が開きます。

0-2-3. Workstationへのログイン

1. Workstationには、Red Hat Enterprise Linux (RHEL) がインストールされています。最初にスクリーンセーバーの画面が見えますので、Enterキーを教えて解除してください。



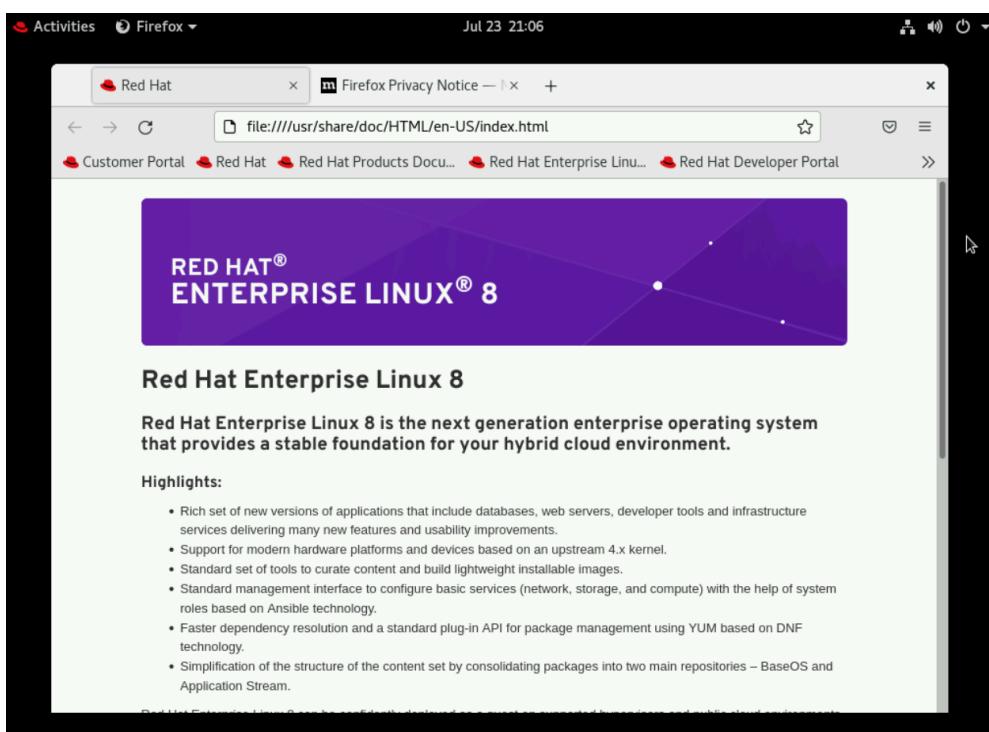
2. 次にログイン画面が現れます。[Student User] を選択して、パスワードとしてstudentを入力して [Sign in] ボタンを押してください。



3. ログインが成功するとRed Hat Enterprise Linux (RHEL) のデスクトップ画面が現れます。画面左上の [Activities] ボタンを押すと、画面左側からよく使うアプリケーションのアイコンのバーがスライドして現れます。

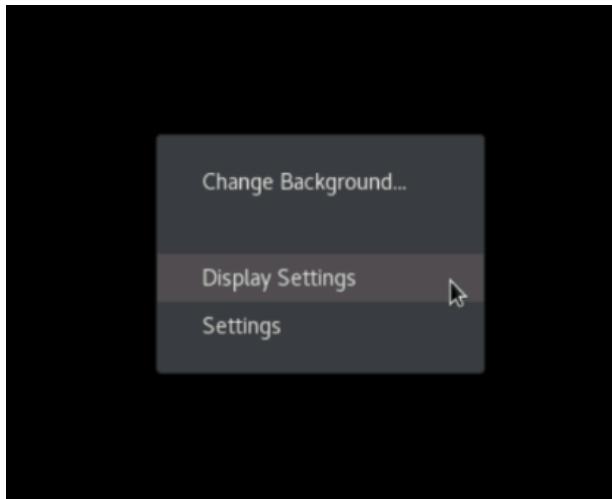


4. アイコンの一番上のFireFoxをクリックして、Webブラウザーを起動してください。このWebブラウザーからOpenShiftクラスターにアクセスします。

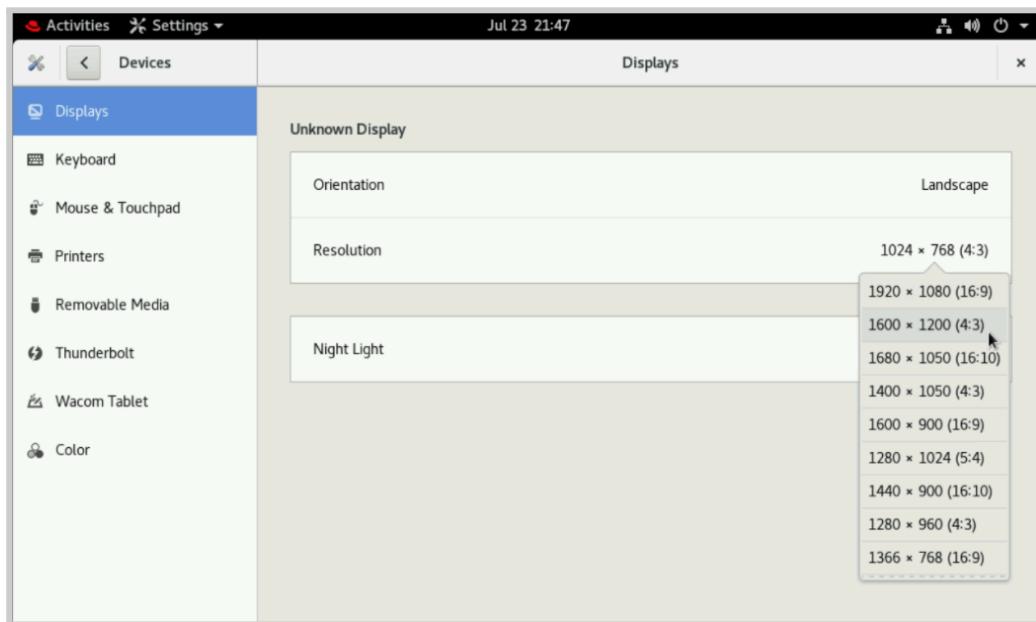


0-2-4. ディスプレイ解像度の変更

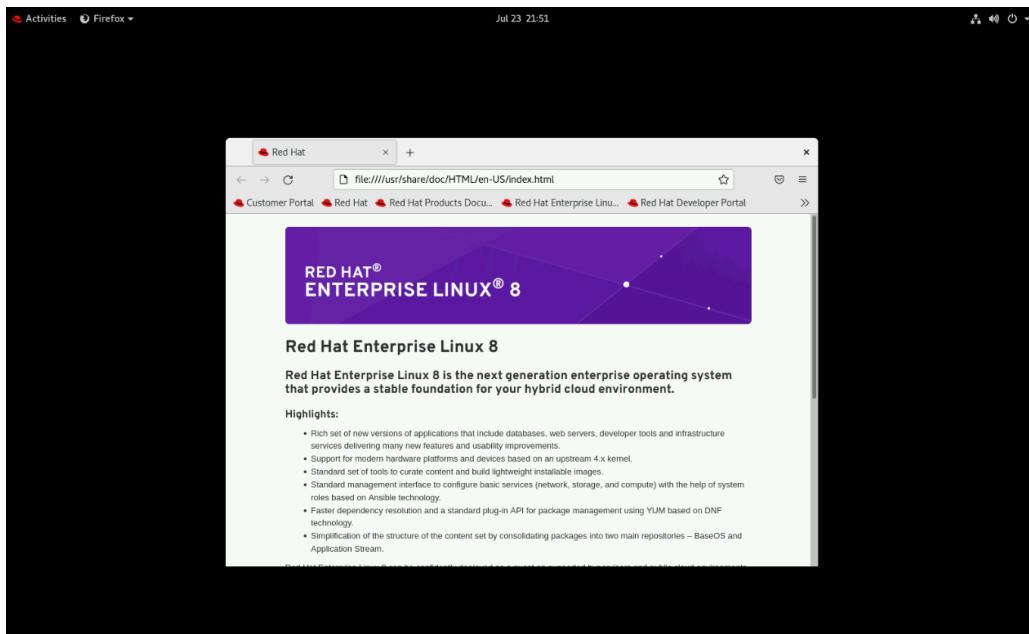
- 初期状態ではWorkstationの画面が狭いので、RHELのディスプレイ設定で解像度を高めに設定し直します。バックグラウンドで右ボタンメニューを表示して、[Display Settings] を選択します。



- [Display] > [Resolution] で 1600 x 1200 を選択し、[Apply] ボタンを押します。



- 解像度変更後には、この設定を維持するかどうかがきかれるので、[Keep Changes] を選択します。



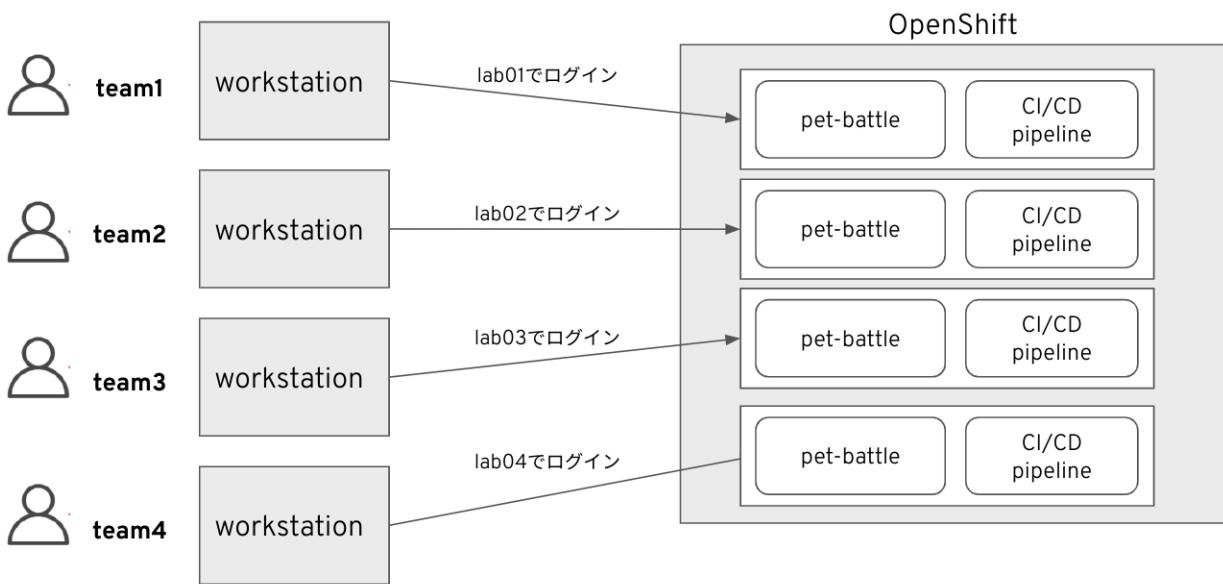
0-3. OpenShiftへのアクセス

技術演習はROLポータル上で実施します。日によってドライバーになる人が代わったとしても、同じRed Hatアカウントを使ってROLポータルにアクセスし、トレーニング期間中、同じWorkstationを使い続けてください。

0-3-1. OpenShiftへのログイン

ファシリテーターからドライバーに対してOpenShiftにログインするための以下の情報が与えられます。

チーム名 (teamX)	演習環境内で使うチーム名です。演習環境ではteam1、team2、team3のように固定のチーム名が設定されています。
ユーザー名 (userX)	OpenShiftやGitLabにログインするときに使用するユーザー名です。各チームに対して、user1、user2、user3のような固定のユーザー名がひとつ割り当てられます。
パスワード	OpenShiftやGitLabにログインするときに使用するユーザー名に対応するパスワードです。



team1のグループは、以下のユーザー名とアカウントを使います。

チーム名	team1
ユーザー名	lab01
パスワード	lab01

team2のグループは、以下のユーザー名とアカウントを使います。

チーム名	team2
ユーザー名	lab02
パスワード	lab02

team3のグループは、以下のユーザー名とアカウントを使います。

チーム名	team3
ユーザー名	lab03
パスワード	lab03

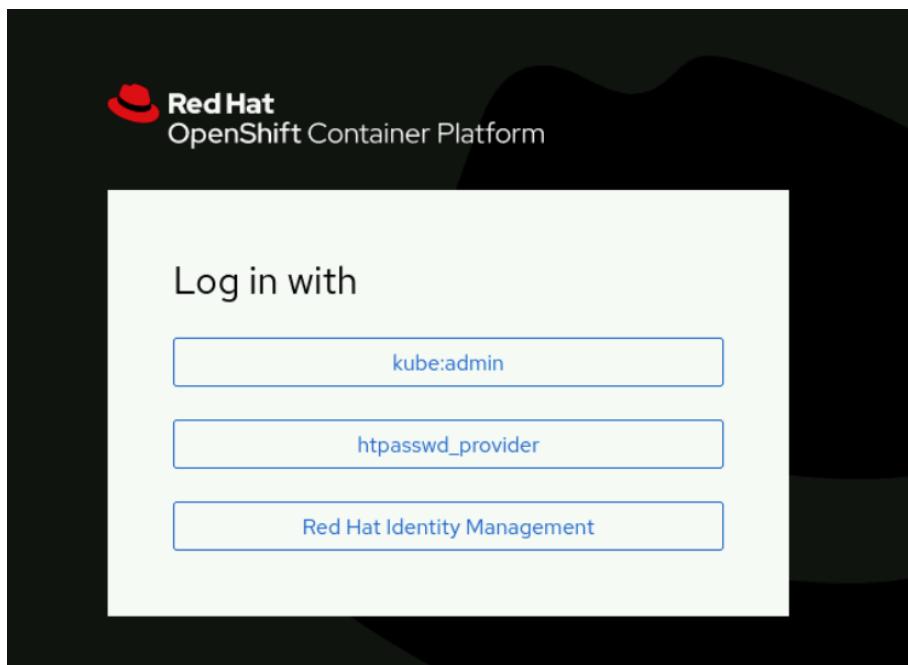
team4のグループは、以下のユーザー名とアカウントを使います。

チーム名	team4
ユーザー名	lab04
パスワード	lab04

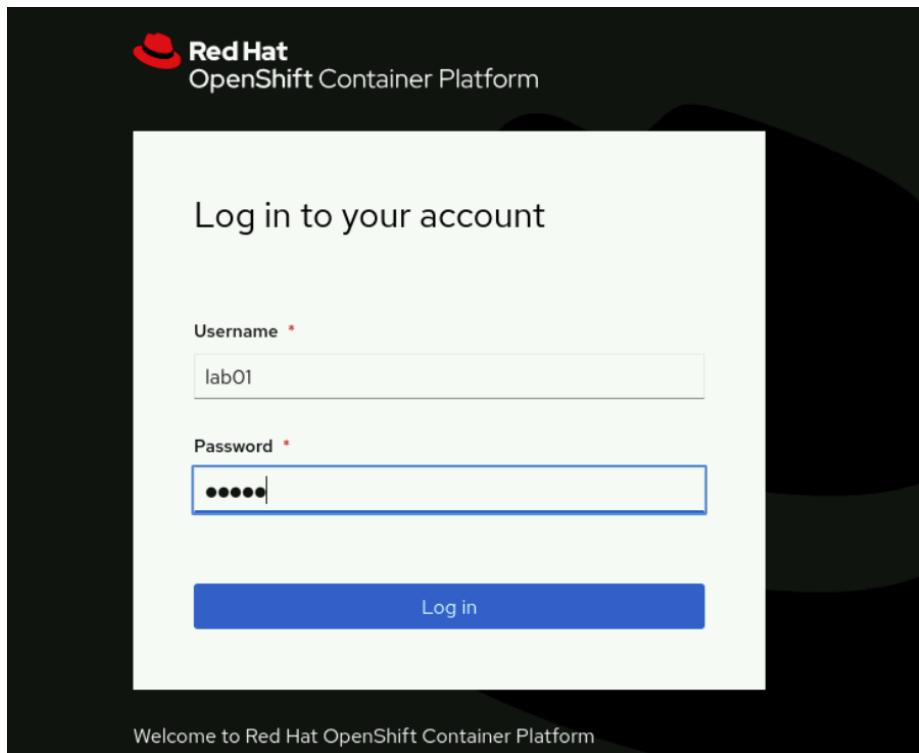
0-3-2. OpenShift Webコンソール

OpenShift Webコンソールを使うと、OpenShift上で動作するアプリケーションの管理や監視をすることができます。

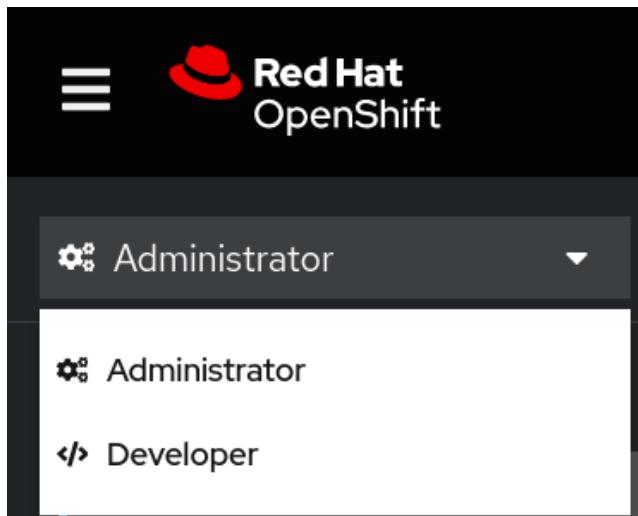
1. Webブラウザーで <https://console-openshift-console.apps.ocp4.example.com> を開くと、OpenShift Webコンソールのログイン画面が表示されます。Log in withの下の [Red Hat Identity Management] を選択します。



2. [Username] と [Password] を入力します。それぞれの値はファシリテーターからチームに割り当てられたものを使います。



3. 左上のRed Hat OpenShiftロゴの下のメニューで、AdministratorまたはDeveloperに切り替えることができます。管理者向け表示である、[Administrator]を選択してください。



4. 左側のナビゲーションメニューから [Home] - [Project]を開きます。右側にログインユーザーの権限で表示可能なプロジェクトのリストが表示されます。



Name	Display name	Status	Requester	Created
PR stackrox	No display name	Active	No requester	Nov 3, 2023, 12:32 PM
PR tl500-gitlab	No display name	Active	No requester	Nov 3, 2023, 12:32 PM
PR tl500-shared	No display name	Active	No requester	Nov 3, 2023, 12:32 PM
PR tl500-workspaces	No display name	Active	No requester	Nov 3, 2023, 12:32 PM

1. コンテナー化されたアプリケーション (20分)

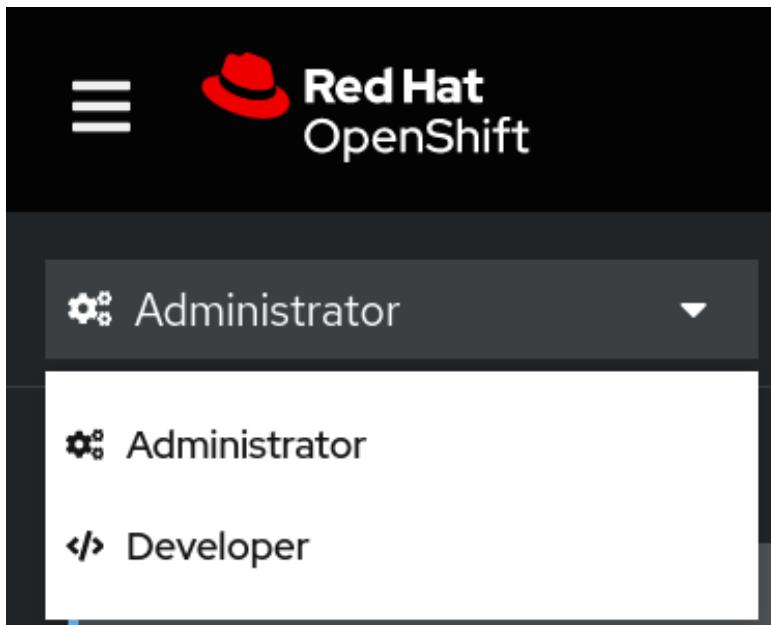
目的	<ul style="list-style-type: none">PetBattleアプリケーションの概要を理解する
目標	<ul style="list-style-type: none">OpenShift 上にインストールされたPetBattleを確認するPetBattleのUIを操作し機能概要を理解するPetBattleの動作を支える関連アプリケーションを調べる

1-1. PetBattleアプリケーションの動作確認

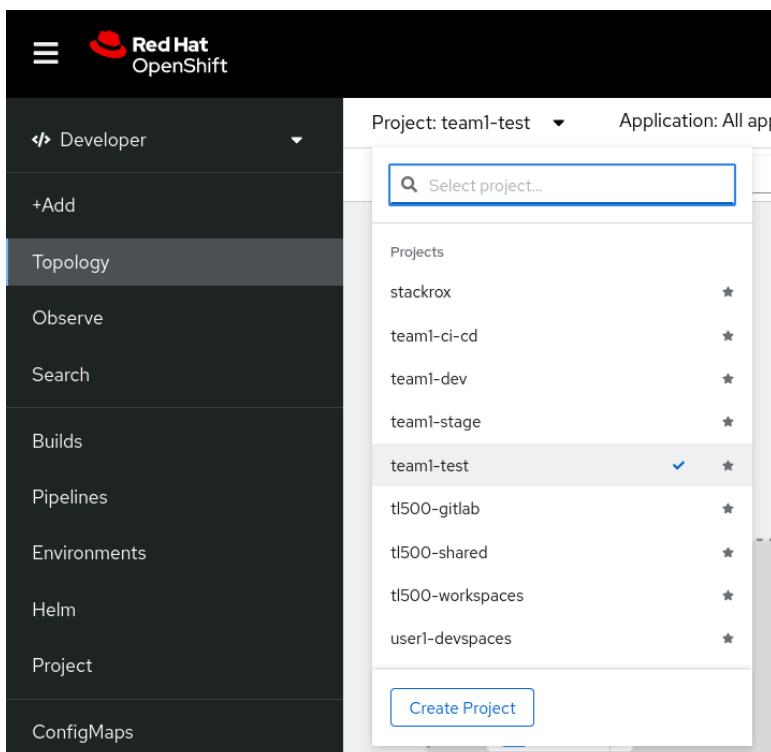
OpenShift上にインストールされたPetBattleのアプリケーションGUI (Graphical User Interface) を表示します。

1-1-1. PetBattleアプリケーションへのアクセス

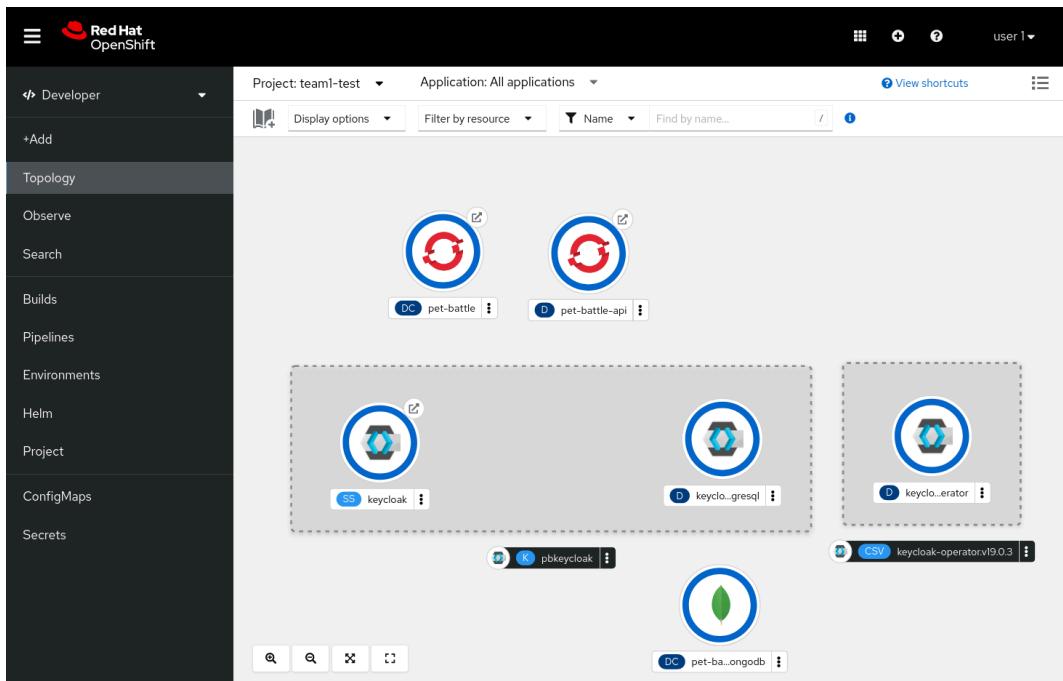
- OpenShift Webコンソールにログインし、左上のRed Hat OpenShiftロゴの下のメニューで、開発者向け表示である、[Developer]を選択してください。



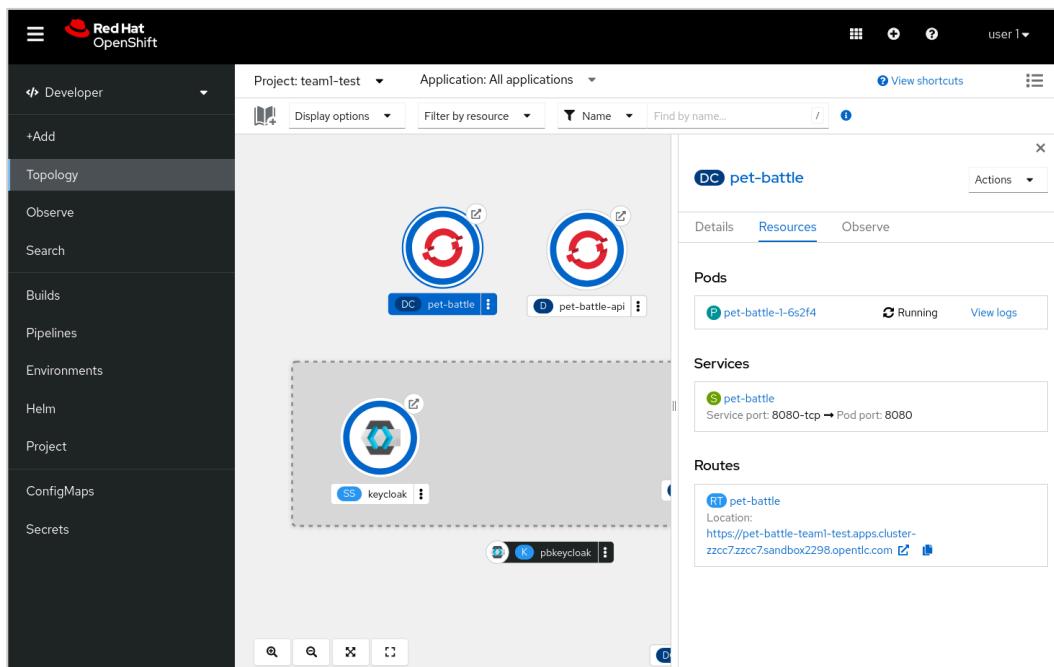
2. ナビゲーションメニューから [Topology] を選択し、右側上部のプロジェクト選択メニューから、<チーム名>-testプロジェクトを選択します。以下の例では、チーム名がteam1の場合で、プロジェクト名はteam1-testプロジェクトになります。



3. <チーム名>-testプロジェクトを選択すると、右側にアプリケーションを表すアイコンが表示されますので、pet-battleアプリケーションを探してください。



4. pet-battleアプリケーションをクリックすると、右側にPods、Services、Routesといったアプリケーションに関連するリソース情報が表示されます。



TIPS

Developerパースペクティブは、アプリケーション開発者がプロジェクト内のアプリケーションに関連するリソースにアクセスしやすいうように情報を配置します。一方、Administratorパースペクティブは、クラスター管理者がクラスター全体の状況を調べやすいうように、



PodsやServicesのような特定のリソースをプロジェクト横断的に表示するように情報を配置します。

- 右側に表示された [Routes] の [Location] リンクを選択します。

The screenshot shows the Red Hat OpenShift web interface for the 'pet-battle' application. At the top, there's a header with the application name 'pet-battle'. Below it, a navigation bar has tabs for 'Details', 'Resources' (which is selected), and 'Observe'. The main content area is divided into sections: 'Pods', 'Services', and 'Routes'. In the 'Pods' section, one pod named 'pet-battle-1-6s2f4' is listed as 'Running'. In the 'Services' section, a service named 'pet-battle' is shown with a port mapping from 'Service port: 8080-tcp' to 'Pod port: 8080'. In the 'Routes' section, a route named 'pet-battle' is listed with its location as 'https://pet-battle-team1-test.apps.cluster-zzcc7.zzzc7.sandbox2298.opentlc.com'. There are also icons for refreshing and viewing logs.

1-1-2. PetBattle アプリケーションのGUI画面

- Webブラウザーの新しいタブにPetBattleアプリケーションのGUI画面が表示されますので、好みの猫に投票してみてください。投票すると写真が別の猫に切り替わります。中央の [Refresh the Leaderboard] を押すと画面が再表示され、投票結果が画面に反映されることを確認してください。



Pet Battle [HOME](#) [RANDOM CAT](#) [CAT OF THE WEEK](#) [TOURNAMENT](#) [LOGIN](#)

It's Cat Vs Cat in this purrfect competition.

The challenge is tough as these cats are all very fur-midable! There can be only one winner in this a-paw-ling competition, could you pawsibly choose just one cat?? You must be kitten-me!!

[Add your cat the competition](#)



🥇 Current TopCat 5



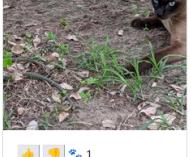
🥈 Second Place Cat 5



🥉 Third Place Cat 4

[Refresh the Leaderboard](#)

Upvote or Downvote your favourite cat to reveal another one ...



1



2



5



1

このPetBattleアプリケーションはGUI (Graphical User Interface) を提供するアプリケーションです。PetBattleが持っている機能は、PetBattle APIという別のアプリケーションが提供します。猫のリストを表示したり、猫に投票したり、投票数の多い猫を表示するなどの機能は、PetBattle APIへそれぞれの処理のリクエストを送信することで実現しています。

2. 画面上部の [Add your cat the competition] ボタンを押して新しい画像をアップロードしてください(猫以外の画像でもOK)。

1-2. PetBattleアプリケーションを支えるサービス

PetBattleアプリケーションはGUIを提供するアプリケーションです。PetBattleは複数のアプリケーションの連携によって実現されます。

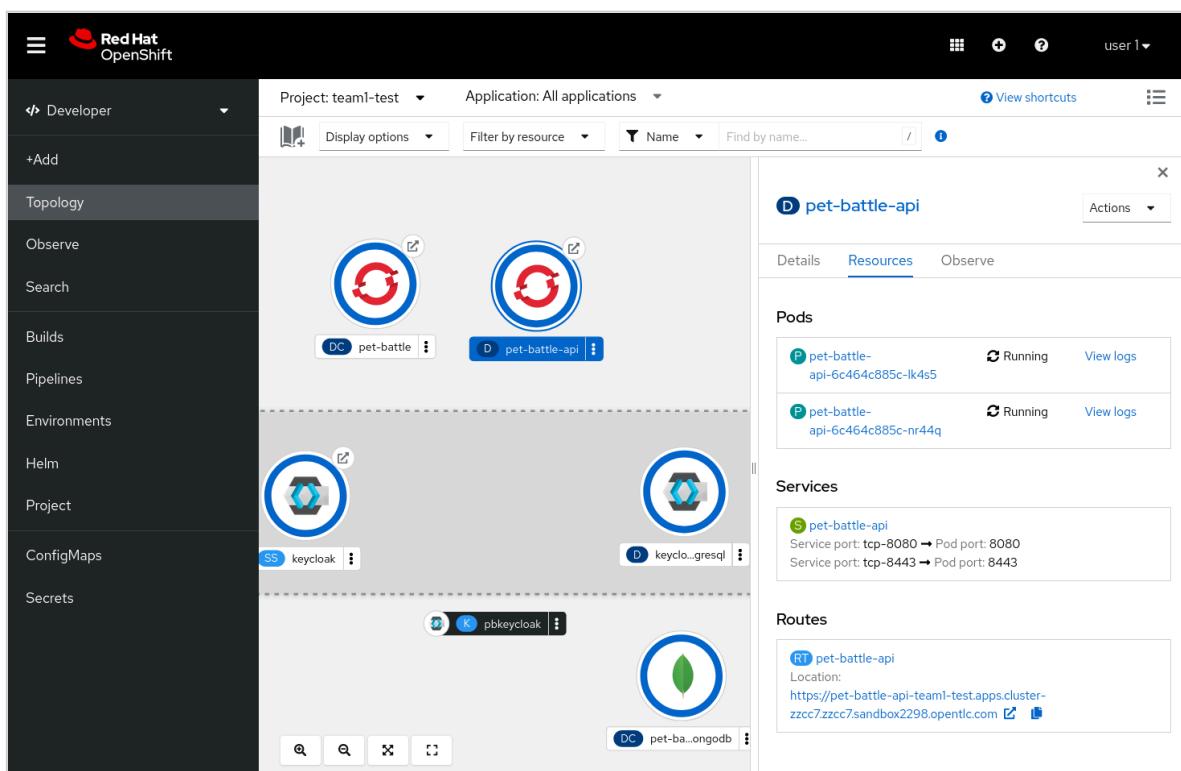
- PetBattleアプリケーション
PetBattleのGUIを提供する
- PetBattle API

PetBattleのAPI (Application Programming Interface) を提供する

- MongoDBデータベース
猫や投票などのデータを保存する
- Keycloak
ユーザー認証を行う(今回未使用)

1-2-1. PetBattle APIへのアクセス

3. OpenShift Webコンソールの [Topology] で <チーム名>-test プロジェクトを選択し、pet-battle-apiアプリケーションをクリックします。



4. pet-battle-apiアプリケーションをクリックすると、右側にリソース情報が表示されます。[Routes] の [Location] リンクをクリックすると、Webブラウザの新しいタブにPetBattle APIアプリケーションのUI画面が表示されます。

Welcome to Pet Battle API !

This page is served by Quarkus

- [Open API Documentation](#)
- [Health](#)
- [Metrics](#)

Next steps

- [Setup your IDE](#)
- [Getting started](#)
- [Quarkus Web Site](#)

Here are the cats.

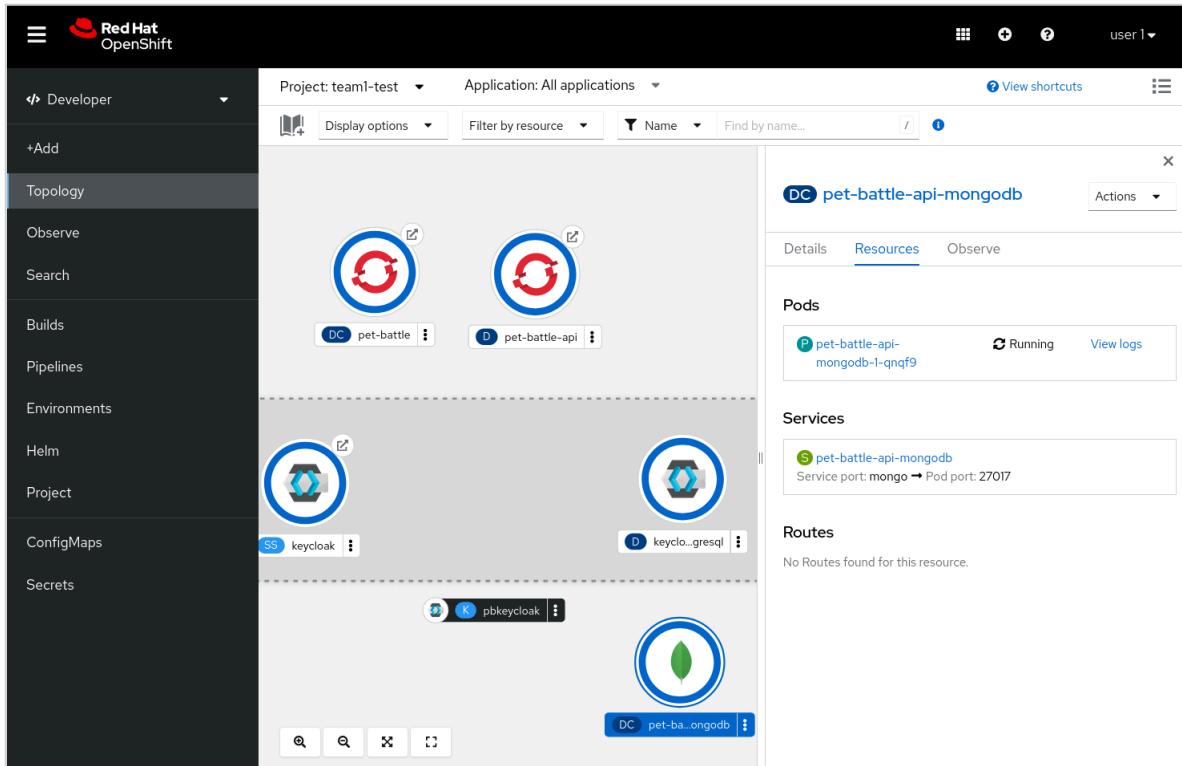
id	count	issff	image
668e18ba36c4605ec8c9f394	3	true	
668e18bb36c4605ec8c9f395	3	true	
668e18bc36c4605ec8c9f396	1	true	
668e18bc36c4605ec8c9f397	1	true	
668e18bc36c4605ec8c9f398	3	true	
668e18bc36c4605ec8c9f399	6	true	
668e18bd36c4605ec8c9f39a	5	true	
668e18bd36c4605ec8c9f39b	2	true	
668e18be36c4605ec8c9f39c	2	true	
668e18be36c4605ec8c9f39d	3	true	

Showing 1 to 10 of 13 entries

Previous 1 2 Next

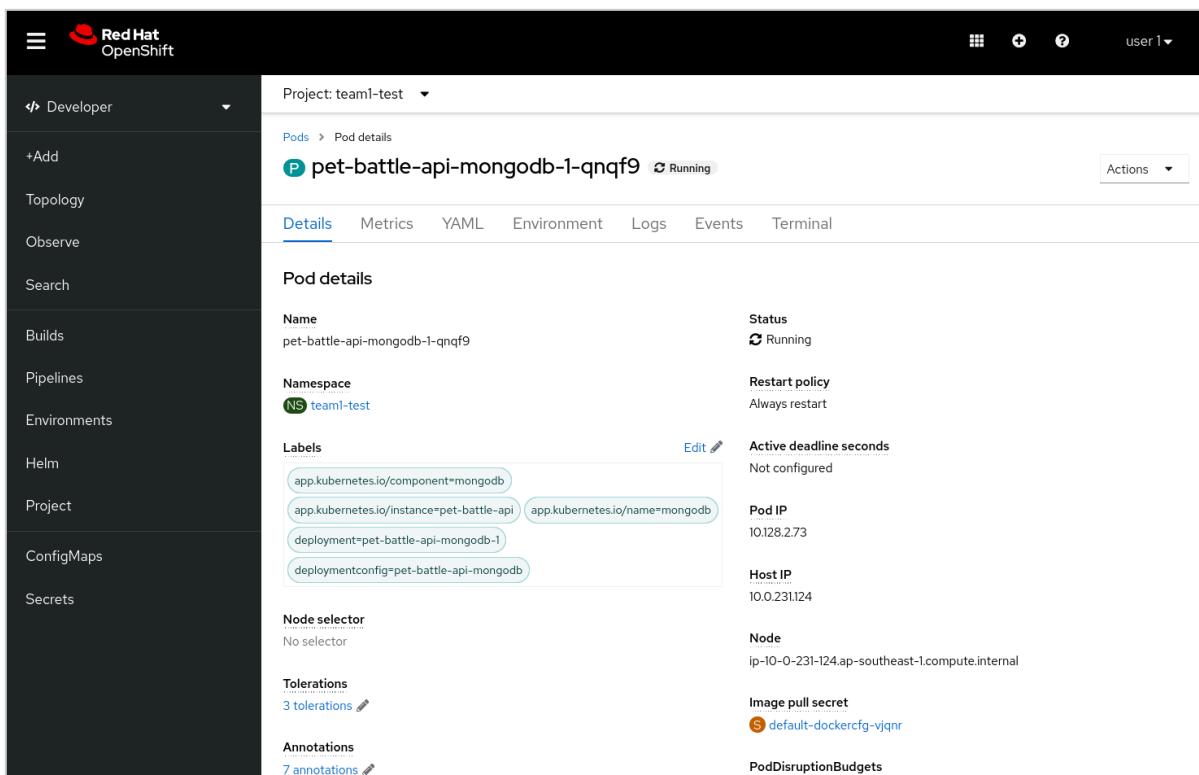
1-2-2. MongoDBへのアクセス

5. OpenShift Webコンソールの [Topology] に戻り、pet-battle-api-mongodbアプリケーションをクリックします。



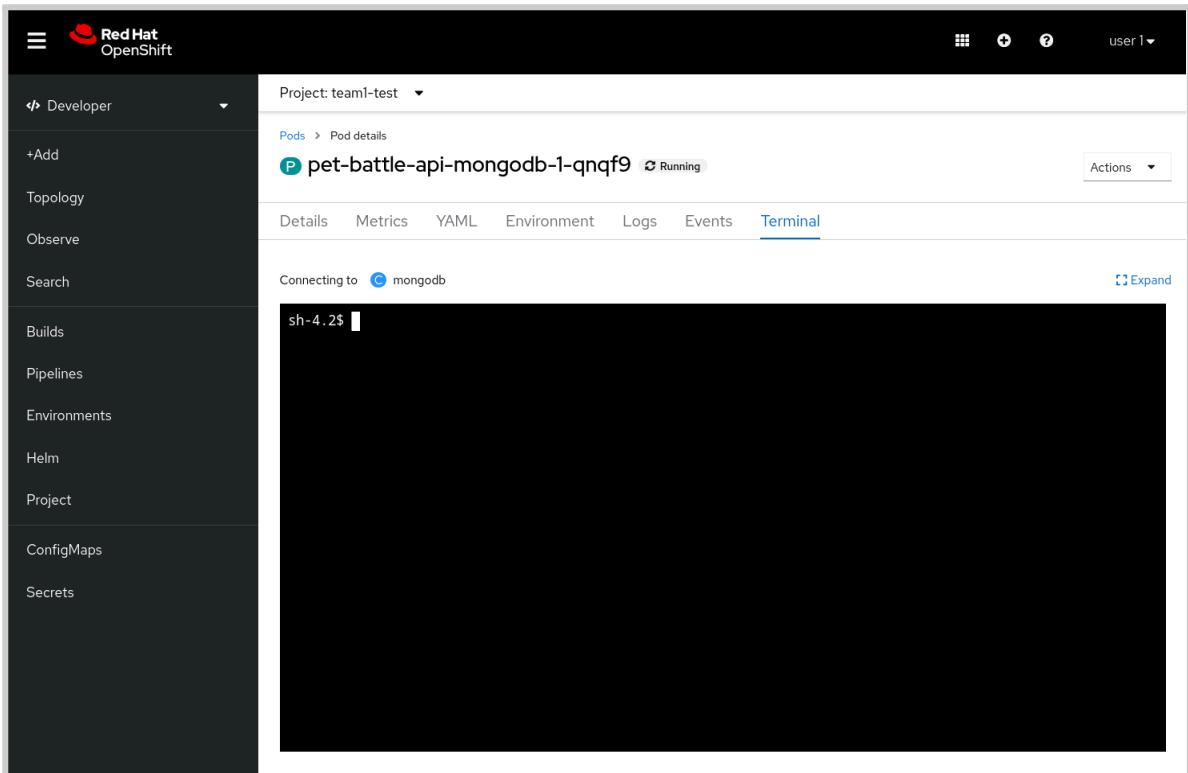
The screenshot shows the Red Hat OpenShift web interface. On the left, a sidebar lists various developer tools: Topology, Observe, Search, Builds, Pipelines, Environments, Helm, Project, ConfigMaps, and Secrets. The main area displays a cluster topology with several services and pods. Services shown include pet-battle-api-mongodb, keycloak, pbkeycloak, and pet-battle-ongodb. Below the services are two pods: pet-battle-api-mongodb-1-qnqf9 and pet-battle-api-mongodb-1-qnqf9. A detailed view of the pet-battle-api-mongodb-1-qnqf9 pod is open on the right, showing its status as 'Running' and providing options to 'View logs'.

6. 右側に表示されたリソース情報から [Pods] の下の
pet-battle-api-mongodb-1-XXXXXXという名前のリンクをクリックすると、MongoDBの
 Podの詳細画面が表示されます(XXXXXXの部分は自動生成された文字列なので実際
 のものとは異なります)。



The screenshot shows the detailed view of the pet-battle-api-mongodb-1-qnqf9 pod. The top navigation bar indicates the project is team1-test. The sidebar on the left remains the same. The main content area is titled "Pod details" for the pod pet-battle-api-mongodb-1-qnqf9, which is currently "Running". The "Details" tab is selected. The pod details section includes fields for Name (pet-battle-api-mongodb-1-qnqf9), Status (Running), Namespace (team1-test), Restart policy (Always restart), Active deadline seconds (Not configured), Pod IP (10.128.2.73), Host IP (10.0.231.124), Node (ip-10-0-231-124.ap-southeast-1.compute.internal), Image pull secret (default-dockercfg-vjqnr), and PodDisruptionBudgets. The Labels section lists several labels: app.kubernetes.io/component=mongodb, app.kubernetes.io/instance=pet-battle-api, app.kubernetes.io/name=mongodb, deployment=pet-battle-api-mongodb-1, and deploymentconfig=pet-battle-api-mongodb.

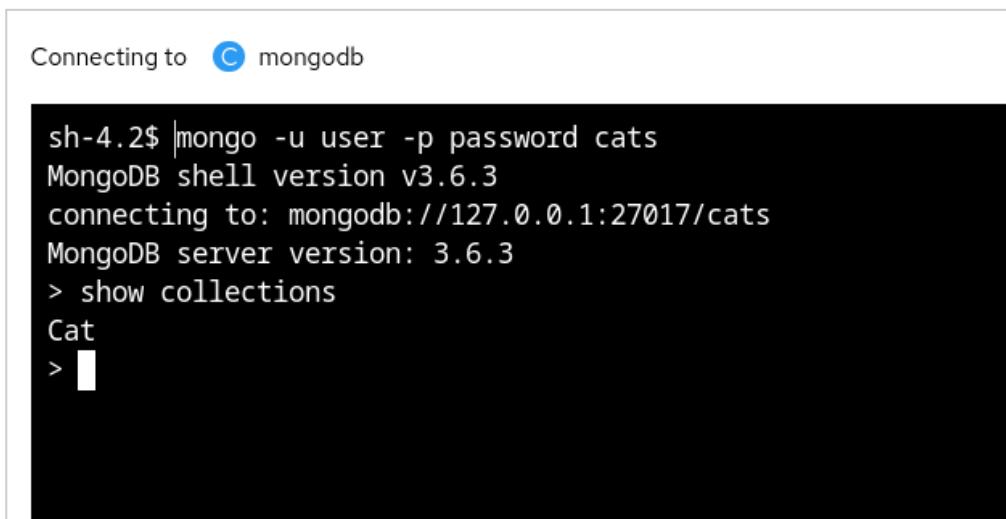
7. Pod詳細画面の[Terminal]タブをクリックすると、MongoDBのコンテナー内でターミナルを開くことができます。このターミナルから、MongoDBのCLIコマンドを実行することができます。



The screenshot shows the Red Hat OpenShift web interface. On the left is a sidebar with various developer tools: Developer (+Add), Topology, Observe, Search, Builds, Pipelines, Environments, Helm, Project, ConfigMaps, and Secrets. The main area is titled "Project: team1-test". It shows a "Pods > Pod details" section for a pod named "pet-battle-api-mongodb-1-qnqf9" which is "Running". Below this are tabs for Details, Metrics, YAML, Environment, Logs, Events, and Terminal, with Terminal being the active tab. A status message "Connecting to C mongodb" is displayed above a terminal window. The terminal window itself is mostly black with a single line of text: "sh-4.2\$".

8. ターミナル内でMongoDB内のデータにアクセスします。以下のコマンドを実行することで、MongoDBに接続することを確認します。

```
sh-4.2$ mongo -u user -p password cats
```



The screenshot shows a terminal session. The prompt "sh-4.2\$" appears, followed by the command "mongo -u user -p password cats". The output shows the MongoDB shell version (v3.6.3) and a connection message: "connecting to: mongodb://127.0.0.1:27017/cats". It then displays the MongoDB server version (3.6.3) and a command: "show collections". The response shows a collection named "Cat". The terminal ends with a prompt "sh-4.2\$".

```
sh-4.2$ mongo -u user -p password cats
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017/cats
MongoDB server version: 3.6.3
> show collections
Cat
>
```

1-2-3. Keycloakへのアクセス

Keycloakは、ユーザー認証やアプリケーションの認証を行うためのサービスです。今回の技術演習では使用しません。

2. Everything As Code (70分)

目的	<ul style="list-style-type: none">Gitリポジトリを使ったPetBattle APIの開発手順を理解する
目標	<ul style="list-style-type: none">GitLabアプリケーションを操作するGitコマンドを使ってソースコードを管理する手順を理解するPetBattle APIのソースのビルド、コンテナイメージ作成、コンテナー実行の各ステップを体験するPetBattle APIをWebブラウザー上で実行するPetBattle APIをSwagger UIを使って操作する

2-1. 開発環境の準備 (10分)

アプリケーションをリリースするまでの間にいくつかの異なる環境を使います。

このような環境は通常、最低でも以下の3種類のものが用意されることが多いです。

- 開発環境：開発者個人が使う環境（自分の担当部分を開発）
- 検証環境：本番環境にリリースする前の検証を行う環境
- 本番環境：ユーザーにサービスを公開するための環境

重要な注意

この演習では、開発者がPetBattle APIアプリケーションを開発しているつもりになって、Workstationを開発環境として使います。

2-1-1. 開発環境の設定

開発に必要な以下のパッケージをWorkstationにインストールします。

- Git
- Make
- Java 17
- Maven
- Podman
- Podman-plugins

1. ソフトウェアパッケージをインストールするため、Workstationのターミナルを開いて以下のコマンドを実行してください。コマンドは、改行せずに1行で入力してください。

```
[student@workstation ~]$ sudo dnf install git make git java-17-openjdk-devel  
maven podman podman-plugins -y
```

2-1-2. Java環境設定

1. OSに2つ以上のJavaがインストールされている場合は、Java 17を使うように指定します（下図では1番がJava 17、2番がJava 1.8.0の指定になります）。以下のJavaのバージョンの詳細は環境によって異なりますので全く同一でなくともJava 17の方を指定していれば大丈夫です。

```
[student@workstation ~]$ sudo alternatives --config java  
There are 2 programs which provide 'java'.  
Selection Command  
-----  
1      java-17-openjdk.x86_64  
(/usr/lib/jvm/java-17-openjdk-17.0.11.0.9-1.fc39.x86_64/bin/java)  
*+ 2      java-1.8.0-openjdk.x86_64  
(/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.412.b06-1.fc39.x86_64/jre/bin/java)  
Enter to keep the current selection[+], or type selection number: 1
```

2. 環境変数JAVA_HOMEの設定のため、以下のコマンドを実行してください。コマンドは、改行せずに1行で入力してください。

```
[student@workstation ~]$ export  
JAVA_HOME=/etc/alternatives/java_sdk_17_openjdk/
```

この環境変数はJavaがインストールされているディレクトリを示しています。

TIPS

この環境変数は設定したシェルでのみ有効です。他のターミナルでJavaを使う場合にこの環境変数を自動設定するには ~/.bashrc ファイルに上記の設定を書いてください。

3. Javaのバージョンがopenjdk 17であることを確認します。バージョンの詳細は環境によって異なりますが、メジャー・バージョンとして17が表示されていればOKです。

```
[student@workstation ~]$ java -version
```



```
openjdk version "17.0.2" 2022-01-18 LTS  
<略>
```

- Javaの環境変数が設定されていることを確認します。

```
[student@workstation ~]$ echo $JAVA_HOME  
/etc/alternatives/java_sdk_17_openjdk/
```

2-1-3. VSCodeのインストール

統合開発環境としてVSCodeをインストールします。VSCodeは、オープンソース版のVSCodeです。VSCodeはRHELの標準コマンドではないために、リポジトリの設定から開始する必要があります。

- 最初にGPG公開鍵をシステムにインポートします。次のコマンドは改行を入れずに1行で入力してください。

```
[student@workstation ~]$ sudo rpmkeys --import  
https://gitlab.com/paulcarroty/vscodium-deb-rpm-repo/-/raw/master/pub.gpg
```

- 次にリポジトリの設定ファイルを作成します。

```
[student@workstation ~]$ sudo sh -c "cat > /etc/yum.repos.d/vscodium.repo" << EOF  
[vscodium]  
name=download.vscodium.com  
baseurl=https://download.vscodium.com/rpms/  
enabled=1  
gpgcheck=1  
gpgkey=https://gitlab.com/paulcarroty/vscodium-deb-rpm-repo/-/raw/master/pub.gpg  
metadata_expire=1  
repo_gpgcheck=1  
EOF
```

- VSCodeをインストールします。

```
[student@workstation ~]$ sudo dnf install codium -y
```

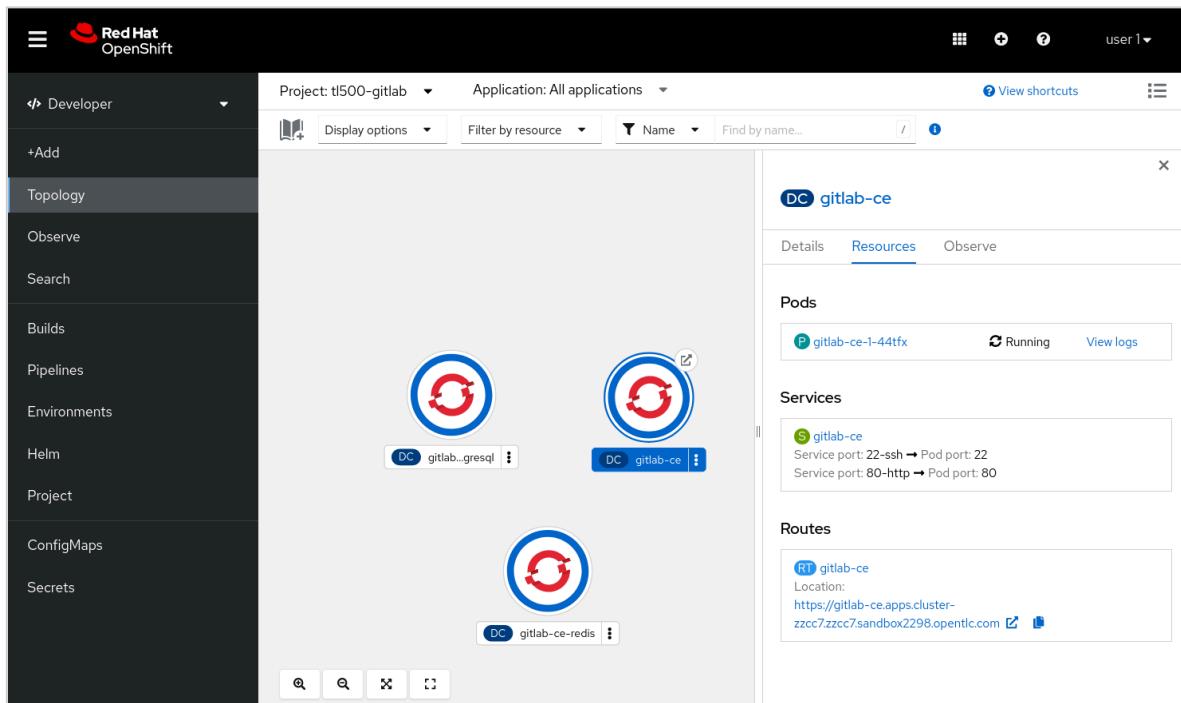
- VSCodeのコマンド名は codium です。codium <ファイル名 or ディレクトリ名> で起動することも可能です。ここでは codium を使って実際にファイルを開くことはせずに、起動確認だけします。起動できたらcodiumを終了してください。

```
[student@workstation ~]$ codium
```

2-2. Gitコマンドの基礎 (20分)

2-2-1. GitLabアプリケーション

1. OpenShift WebコンソールのDeveloperペースペクティブにおいて [Topology]を選択します。上部のプロジェクト選択メニューで tl500-gitlabプロジェクトを選択し、gitlab-ce アプリケーションをクリックします。



2. gitlab-ceアプリケーションをクリックすると、右側にリソース情報が表示されます。[Routes]の[Location]リンクをクリックすると、Webブラウザの新しいタブにGitLabアプリケーションのサインイン画面が表示されます。チームのドライバーにファシリテーターから提供された、チーム名とパスワードを入力し、[Sign in]ボタンを押してください。



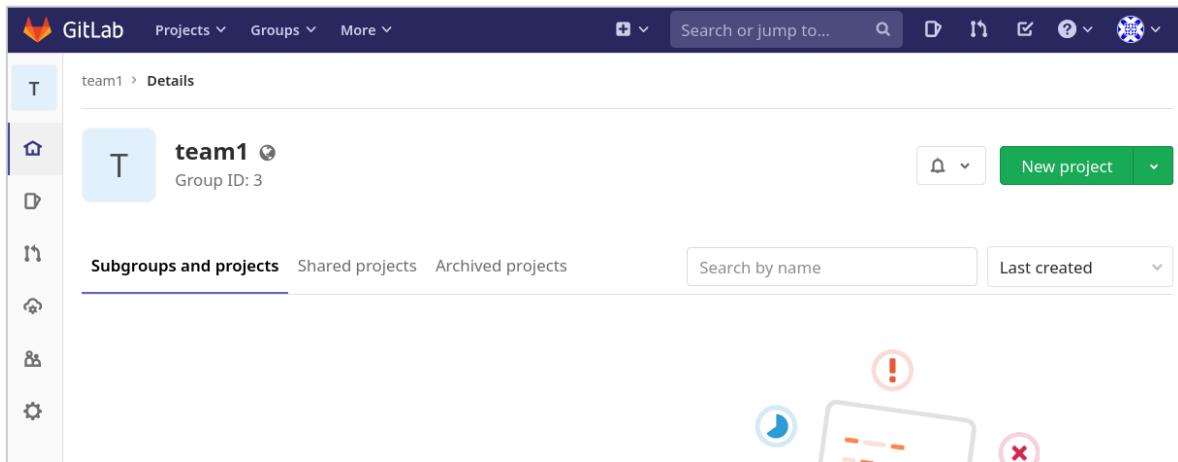
The screenshot shows the GitLab Community Edition login page. At the top, there's a banner with the text "GitLab Community Edition" and "Open source software to collaborate on code". Below the banner, there's a brief description: "Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki." On the right side of the page, there's a login form for "LDAP" authentication. The form fields are: "LDAP Username" (containing "lab01"), "Password" (containing "*****"), and a "Remember me" checkbox. A green "Sign in" button is at the bottom of the form.

3. サインイン後にGitLabプロジェクト画面が表示されます。GitLabにはすでに2つのプロジェクトが作られた状態になっています。以下の図ではteam1グループの下にpet-battle-apiプロジェクトとtech-exerciseプロジェクトが登録されています。

The screenshot shows the GitLab "Projects" page. The header includes the GitLab logo, navigation links for "Projects", "Groups", and "More", and a search bar. Below the header, the "Projects" section is titled "Your projects 2". It shows two projects: "team1 / pet-battle-api" and "team1 / tech-exercise". Both projects are listed as "Owner". The "pet-battle-api" project was updated "1 day ago". The "tech-exercise" project was also updated "1 day ago".

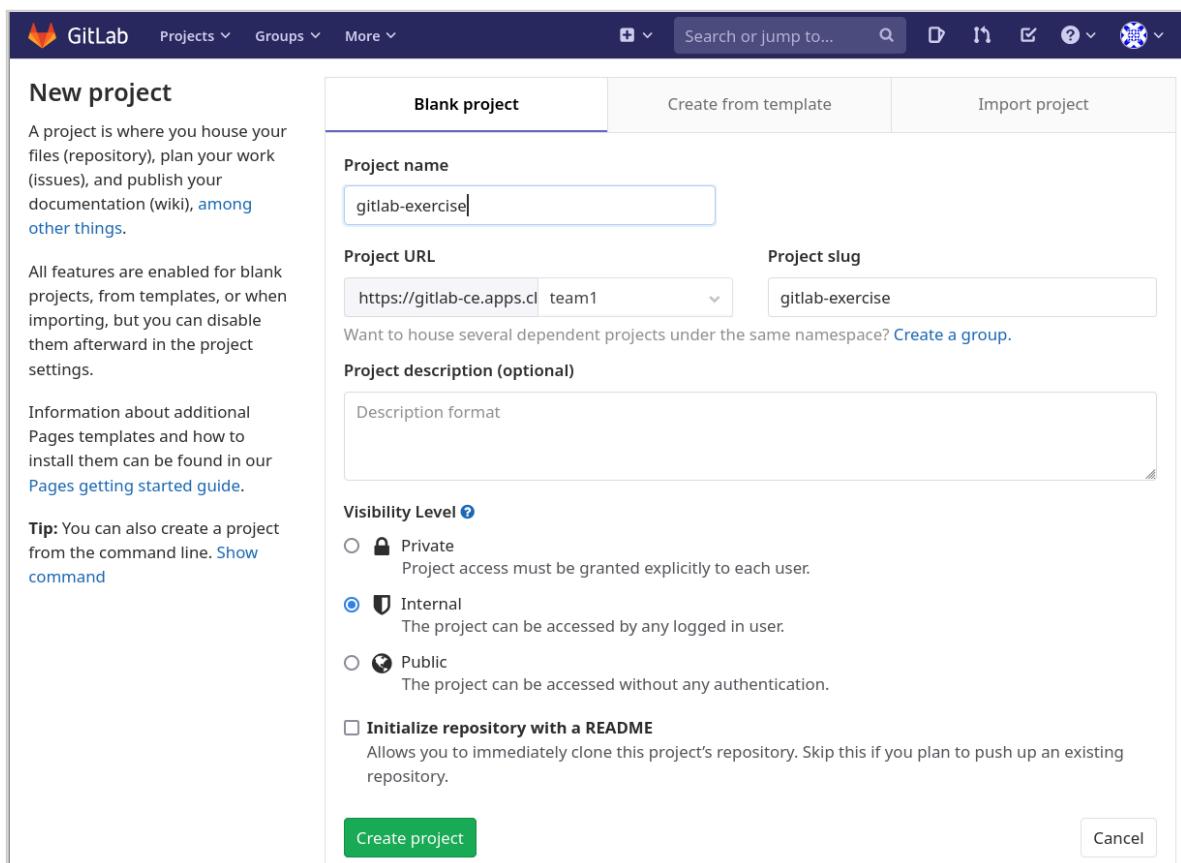
2-2-2. Gitリポジトリの作成

1. [Projects] のページで、緑色の [New project] ボタンを押します。



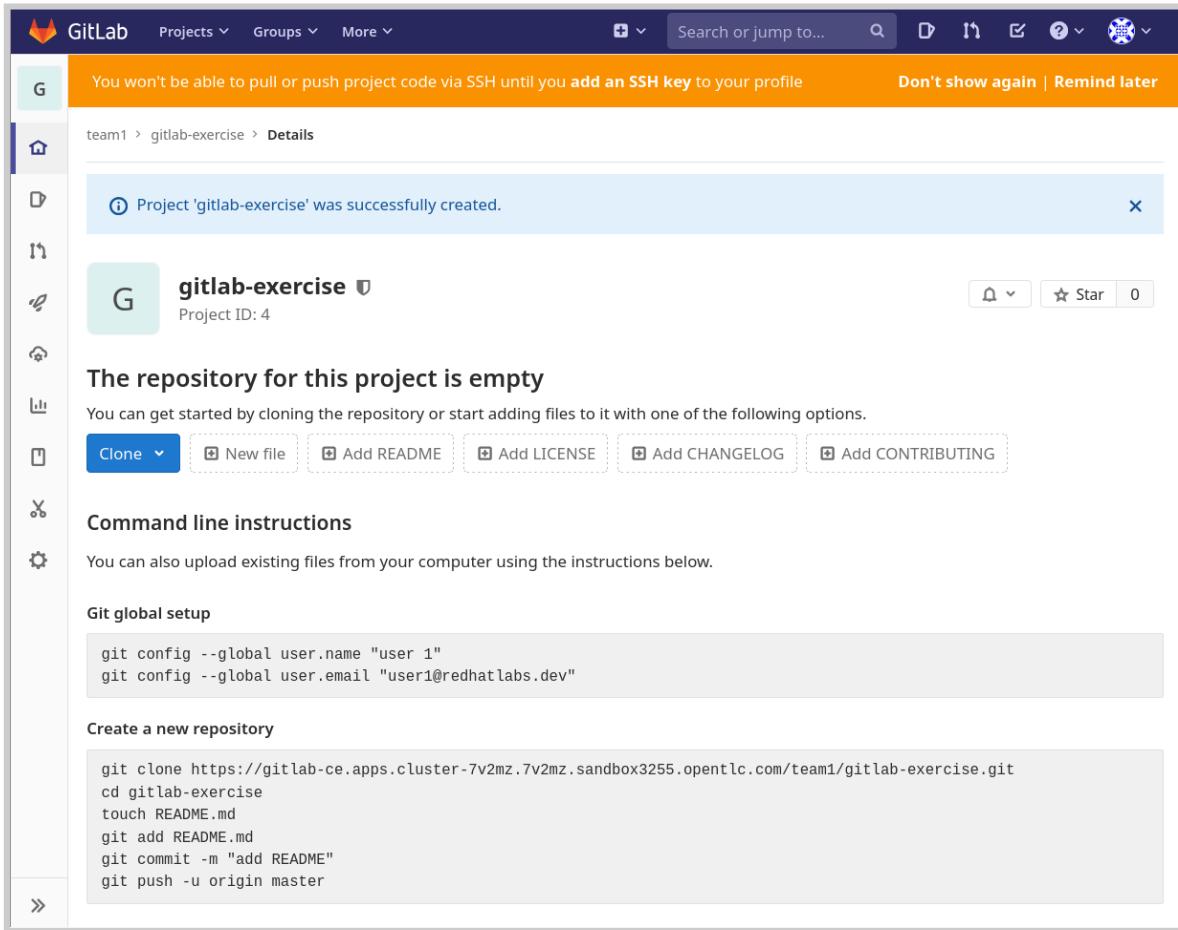
The screenshot shows the GitLab interface for the 'team1' group. The top navigation bar includes 'Projects', 'Groups', and 'More'. The main area displays the group details with a 'Subgroups and projects' section. A search bar at the bottom right allows filtering by 'Last created'.

[New project] の画面で、[Project name] に **gitlab-exercise** と入力し、[Visibility Level] で Internal を選択後、緑色の[Create project] ボタンを押します。



The screenshot shows the 'New project' dialog. It has three tabs: 'Blank project' (selected), 'Create from template', and 'Import project'. The 'Project name' field contains 'gitlab-exercise'. The 'Project URL' field shows 'https://gitlab-ce.apps.cl team1'. The 'Project slug' field is 'gitlab-exercise'. Below these fields is a note about creating dependent projects under the same namespace. The 'Project description (optional)' field is empty. Under 'Visibility Level', the 'Internal' option is selected. There is also an unchecked checkbox for 'Initialize repository with a README'. At the bottom are 'Create project' and 'Cancel' buttons.

これで teamX グループの下に gitlab-exercise プロジェクトが作成されました。



You won't be able to pull or push project code via SSH until you [add an SSH key](#) to your profile

team1 > gitlab-exercise > **Details**

Project 'gitlab-exercise' was successfully created.

gitlab-exercise

Project ID: 4

The repository for this project is empty

You can get started by cloning the repository or start adding files to it with one of the following options.

Clone New file Add README Add LICENSE Add CHANGELOG Add CONTRIBUTING

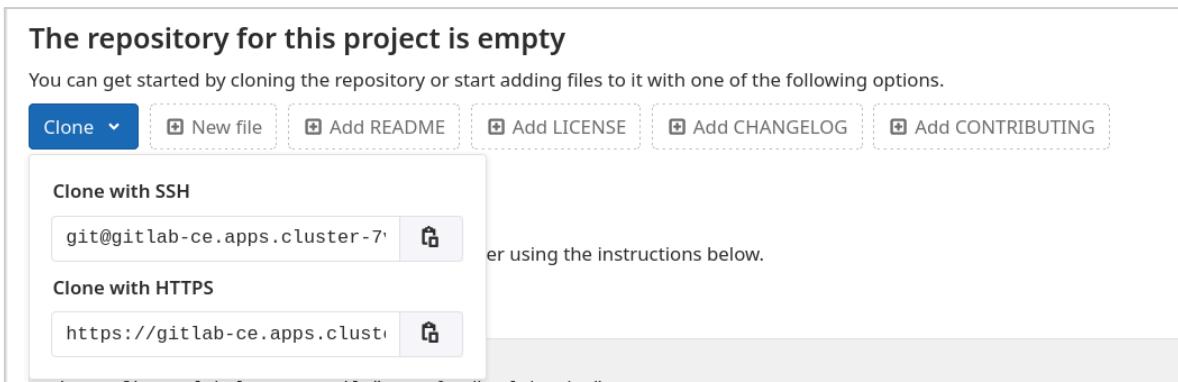
Command line instructions

```
git config --global user.name "user 1"
git config --global user.email "user1@redhatlabs.dev"
```

Create a new repository

```
git clone https://gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com/team1/gitlab-exercise.git
cd gitlab-exercise
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

青い[Clone]ボタンを押して、[Clone with HTTPS]の下に表示されたリポジトリのURLをコピーします。



The repository for this project is empty

You can get started by cloning the repository or start adding files to it with one of the following options.

Clone New file Add README Add LICENSE Add CHANGELOG Add CONTRIBUTING

Clone with SSH
git@gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com:team1/gitlab-exercise.git

Clone with HTTPS
https://gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com:team1/gitlab-exercise.git

- Gitローカルリポジトリを作成します。ローカルのPCでターミナルを開き、git cloneコマンドを実行すると、ローカルリポジトリが作成されます。git cloneの引数のURLには前のステップでコピーしたURLを指定します（以下のgit cloneのURLは環境によって異なります）。UsernameとPasswordはファシリテーターから指定されたものを使ってください。



```
[student@workstation ~]$ git clone  
https://gitlab-ce.apps.ocp4.example.com/team1/gitlab-exercise.git  
Cloning into 'gitlab-exercise'...  
<略>  
warning: You appear to have cloned an empty repository.  
total 0
```

2-2-3. Gitユーザーの設定

最初にリポジトリにコミットを記録する際のユーザ情報を設定する必要があります。これは初回に1回だけ実施すればよいものです。

1. git configコマンドを使ってユーザー名とメールアドレスを設定します。下記の"lab01"の部分は、実際のユーザー名に置き換えてください。

```
[student@workstation ~]$ git config --global user.name "lab01"  
[student@workstation ~]$ git config --global user.email "lab01@redhatlabs.dev"
```

設定できているかの確認は、git config --global --listで行えます。

```
[student@workstation ~]$ git config --global --list  
user.email=lab01@redhatlabs.dev  
user.name=lab01
```

1. パスワードのキャッシュ期間を設定します。キャッシュに保存されている間は、git pushのときにパスワードを入力する必要はありません。

```
[student@workstation ~]$ git config --global credential.helper 'cache  
--timeout=3600'
```

2-2-4. Gitコマンドの基本操作

Gitローカルリポジトリで変更した内容をGitリモートリポジトリに反映する手順を確認します。

1. gitlab-exerciseディレクトリに移動して、git statusコマンドを実行します。git statusコマンドは、ローカルプロジェクトにおけるファイルの状態を示します。



```
[student@workstation ~]$ cd gitlab-exercise/
[student@workstation gitlab-exercise]$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

- gitlab-exerciseディレクトリでREADME.mdファイルを作成します。ターミナルでgeditエディタを起動し以下の1行を入力し、[Save] ボタンを押して保存してください。保存が終わったら、geditのアプリケーションを終了します。

README

ファイル作成後にgit statusコマンドでリポジトリの状況を確認すると、Untracked filesとしてREADME.mdが表示されます。

```
[student@workstation gitlab-exercise]$ gedit README.md # エディタを起動する
[student@workstation gitlab-exercise]$ cat README.md # ファイルの内容を確認
# README

[student@workstation gitlab-exercise]$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  README.md

nothing added to commit but untracked files present (use "git add" to track)
```

- ステージングエリアにREADME.mdを追加するため、git addコマンドを実行します。git statusを実行すると、README.mdがコミット対象である(Changes to be committed)と表示されます。

```
[student@workstation gitlab-exercise]$ git add README.md
[student@workstation gitlab-exercise]$ git status
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file: README.md
```

4. 同様に、**gitedit**を使って、gitlab-exerciseディレクトリに以下の内容を持つhello.pyというファイルを作成します。

```
print("Hello world!")
```

hello.pyは、Python言語で書かれた1行プログラムです。python hello.pyを実行すると "Hello world!"と表示します。

```
[student@workstation gitlab-exercise]$ gedit hello.py
[student@workstation gitlab-exercise]$ cat hello.py
print("Hello world!")
[student@workstation gitlab-exercise]$ python hello.py
Hello world!
```

ファイル作成後にgit addコマンドでステージングエリアに登録します。ステージングエリアにREADME.mdとhello.pyの2つのファイルが登録されました。

```
[student@workstation gitlab-exercise]$ git add hello.py
[student@workstation gitlab-exercise]$ git status
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
new file: README.md
new file: hello.py
```

5. ステージングエリアに2つのファイルが登録されたところで、git commitコマンドを使ってコミットを作成します。git commitの-mオプションでコミットメッセージを指定します。このコミットメッセージは次のステップのgit logで確認することができます。

```
[student@workstation gitlab-exercise]$ git commit -m 'initial commit'
[master (root-commit) 7c1b5f8] initial commit
 2 files changed, 2 insertions(+)
 create mode 100644 README.md
 create mode 100644 hello.py
```

6. **git log**コマンドによってコミットの履歴を確認できます。コミット履歴には、誰がコミットを実行したのかが一緒に記録されています。

```
[student@workstation gitlab-exercise]$ git log
commit 7c1b5f8ae5c3255c3a29931e8edbb9a31a976160 (HEAD -> master)
Author: user 1 <user1@redhatlabs.dev>
Date: Thu Jul 18 15:13:30 2024 +0900

  initial commit
```

7. **git push**コマンドによってローカルリポジトリにコミットがリモートリポジトリへアップロードされます。UsernameとPasswordはファシリテーターから渡されたものを使ってください。

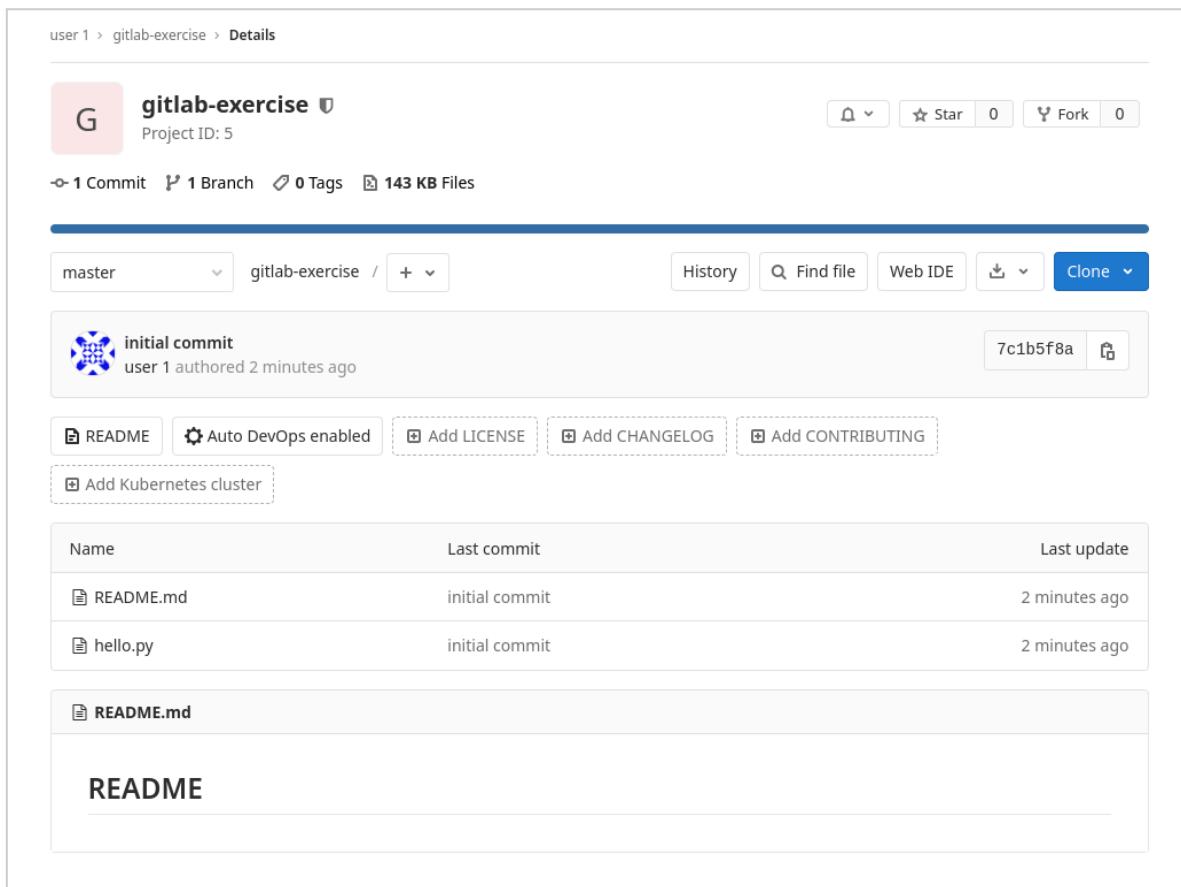
```
[student@workstation gitlab-exercise]$ git push -u origin master
Username for 'https://gitlab-ce.apps.ocp4.example.com': lab01
Password for 'https://lab01@gitlab-ce.apps.ocp4.example.com':
warning: redirecting to
https://gitlab-ce.apps.ocp4.example.com/lab01/gitlab-exercise.git/
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 20 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 283 bytes | 283.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://gitlab-ce.apps.ocp4.example.com/lab01/gitlab-exercise
 * [new branch] master -> master
branch 'master' set up to track 'origin/master'
```

8. **git push**を行ったターミナルでカレントディクトリをホームディレクトリに変更します。

```
[student@workstation gitlab-exercise]$ cd
```

[student@workstation ~]\$

9. git pushが成功したらGitLab上のgitlab-exerciseプロジェクトを確認します。ローカルリポジトリと同じソースコードがリモートリポジトリに見えるはずです。



user 1 > gitlab-exercise > Details

gitlab-exercise  Project ID: 5

1 Commit 1 Branch 0 Tags 143 KB Files

master / gitlab-exercise / +

History Find file Web IDE ⌂ Clone

initial commit
user 1 authored 2 minutes ago

7c1b5f8a ⌂

README Auto DevOps enabled Add LICENSE Add CHANGELOG Add CONTRIBUTING
 Add Kubernetes cluster

Name	Last commit	Last update
README.md	initial commit	2 minutes ago
hello.py	initial commit	2 minutes ago

README.md

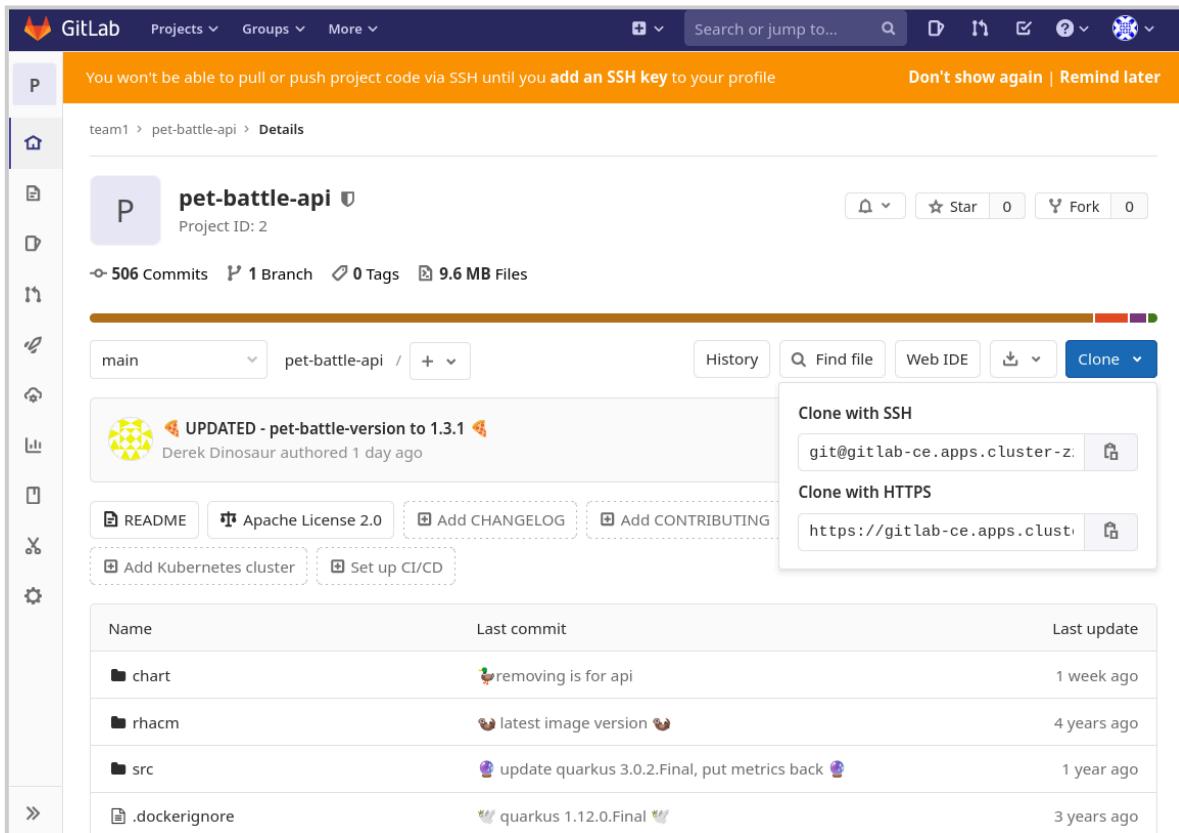
README

2-3. PetBattle APIアプリケーションのビルドから実行まで(20分)

ここでは開発者のタスクを理解するため、GitLabからソースコードをダウンロードして、PetBattle APIアプリケーションが動作するまでを実行していきます。

2-3-1. PetBattle API ソースコードのダウンロード

1. GitLab上のpet-battle-apiプロジェクトをクリックすると、プロジェクトのソースコード一式が表示されます。青い[Clone]ボタンを押すと、このpet-battle-apiプロジェクトのURLが表示されます。[Clone with HTTPS]のURLをコピーします。



2. GitLabからソースコードをダウンロードするために、Workstationのターミナル上で、以下のgit cloneコマンドを実行します。git cloneコマンドの引数として直前のステップでコピーしたURLを指定します。git cloneは、引数を含めて1行で入力してください。

```
git clone https://gitlab-ce.apps.ocp4.example.com/<チーム名>/pet-battle-api.git
```

git cloneの実行途中でUsernameとPasswordの入力が求められます。これらの値としては、ファシリテーターから与えられたユーザー名とパスワードを入力してください。

```
[student@workstation ~]$ git clone
https://gitlab-ce.apps.ocp4.example.com/team1/pet-battle-api.git
Cloning into 'pet-battle-api'...
remote: Enumerating objects: 3230, done.
remote: Counting objects: 100% (936/936), done.
remote: Compressing objects: 100% (328/328), done.
remote: Total 3230 (delta 650), reused 840 (delta 563), pack-reused 2294
Receiving objects: 100% (3230/3230), 9.64 MiB | 10.04 MiB/s, done.
```



```
Resolving deltas: 100% (1869/1869), done.
```

2-3-2. PetBattle API ソースコードのビルド

1. **git clone**を実行したローカルPC上のターミナルにおいて、**pet-battle-api**ディレクトリにおいて**make compile** コマンドでJavaソースコードのビルドを実行します。このコマンドにより、Javaソースコードのコンパイルが行われ、**target**ディレクトリにアプリケーションの成果物であるjarファイルが生成されます。

初回のビルドは、ネットワークからJavaライブラリが大量にダウンロードされるために時間がかかりますが、これらのライブラリはローカルディスクに保存されるので、次回のビルドは比較的短時間に実行できます。

```
[student@workstation ~]$ cd pet-battle-api
[student@workstation pet-battle-api]$ make compile
mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] -----< org.acme:pet-battle-api >-----
[INFO] Building pet-battle-api 1.1.1
[INFO]   from pom.xml
[INFO] -----[ jar ]-----
```

<略>

```
[INFO] Scanning for projects...
Downloading from central:
https://repo.maven.apache.org/maven2/io/quarkus/platform/quarkus-maven-plugin/3.0.2.Final/quarkus-maven-plugin-3.0.2.Final.pom
Downloaded from central:
https://repo.maven.apache.org/maven2/io/quarkus/platform/quarkus-maven-plugin/3.0.2.Final/quarkus-maven-plugin-3.0.2.Final.pom (10 kB at 17 kB/s)
Downloading from central:
https://repo.maven.apache.org/maven2/io/quarkus/platform/quarkus-bom/3.0.2.Final/quarkus-bom-3.0.2.Final.pom
<略>
```

```
[INFO] --- quarkus:1.13.7.Final:build (default) @ pet-battle-api ---
[INFO] [org.jboss.threads] JBoss Threads version 3.2.0.Final
[INFO] [io.quarkus.deployment.QuarkusAugmentor] Quarkus augmentation completed
in 3146ms
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 12.420 s
[INFO] Finished at: 2024-07-26T00:10:43+09:00
[INFO] -----
```

2-3-3. PetBattle API コンテナーイメージ作成

1. **make podman-build**コマンドを使ってpet-battle-apiアプリケーションからコンテナーアイメージを作成します。コンテナーアイメージとは、コンテナーの元になるバイナリーのファイルで、コンテナーを実行するのに必要なコマンドやライブラリを含みます。

```
[student@workstation pet-battle-api]$ make podman-build
mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] -----< org.acme:pet-battle-api >-----
[INFO] Building pet-battle-api 1.1.1
[INFO]  from pom.xml
[INFO] -----[ jar ]-----

<略>

STEP 15/17: EXPOSE 8080
--> 851afedeb8c8
STEP 16/17: USER 1001
--> f8f20092c089
STEP 17/17: ENTRYPOINT [ "/deployments/run-java.sh" ]
COMMIT quay.io/petbattle/pet-battle-api:latest
--> 453dd48863aa
Successfully tagged quay.io/petbattle/pet-battle-api:latest
```



```
453dd48863aa5695a5e785ed8820bfd2f4056dae849ffd5c3493803d574447cf  
[student@workstation pet-battle-api]$
```

2-2-4. PetBattle API MongoDB コンテナーの起動

1. MongoDBはPetBattle APIからの要求を受けてデータを検索したり、保存したりするデータベースです。PetBattle APIが扱う猫や投票データはMongoDB内に保存されますので、PetBattle APIのUI画面を実行するためにはMongoDBを起動しておく必要があります。

新しいターミナルを開いて以下のコマンドを実行してください。`podman network create pet-battle` は最初に1回だけ実行すれば次回から実行する必要はありません。`podman run`コマンドは、改行せずに1行で入力してください。

```
[student@workstation pet-battle-api]$ podman network create pet-battle  
[student@workstation pet-battle-api]$ podman run --rm --name mongo -p  
27017:27017 --network pet-battle  
quay.io/mongodb/mongodb-community-server
```

<略>

```
{"t": {"$date": "2024-07-25T15:12:35.417+00:00"}, "s": "I", "c": "INDEX", "id": 20345,  
"ctx": "LogicalSessionCacheRefresh", "msg": "Index build: done  
building", "attr": {"buildUUID": null, "collectionUUID": {"uuid": {"$uuid": "82d0b529-fb8b-4c  
d5-933a-120ccb1183f1"}}, "namespace": "config.system.sessions", "index": "lsidTTLIndex",  
"ident": "index-6-3885138141738700255", "collectionIdent": "collection-4-3885138141738  
700255", "commitTimestamp": null}}
```

このコマンドはデータベースとして実行を続けてターミナルを占有します。PetBattle API アプリケーションはデータを保存・検索するためにMongoDBを使いますので、ターミナルは閉じずにそのままにしておいてください。終了させるときはCtrl-Cを押してください (Ctrl+Cとは、Ctrlキーを押しながら、Cキーを押すことを意味します)。

2-3-4. PetBattle API コンテナーの起動

1. [2-3-3](#) で作成したコンテナーイメージ (`quay.io/petbattle/pet-battle-api`) を元に、PetBattle APIコンテナーを起動します。新しいターミナルを開き、`pet-battle-api` のディレクトリで`podman run`コマンドを実行してください。このコマンドは、改行せずに1行で入力してください。

```
[student@workstation pet-battle-api]$ podman run --rm --name pet-battle-api -p 8080:8080 --network pet-battle -e quarkus.mongodb.connection-string=mongodb://mongo:27017 -e QUARKUS_PROFILE=dev quay.io/petbattle/pet-battle-api:latest
```

<略>

```
2024-07-25 15:13:07,249 INFO [io.quarkus] (main) pet-battle-api 1.0.1 on JVM (powered by Quarkus 1.13.7.Final) started in 5.941s. Listening on: http://0.0.0.0:8080 and https://0.0.0.0:8443
```

```
2024-07-25 15:13:07,256 INFO [io.quarkus] (main) Profile dev activated.
```

```
2024-07-25 15:13:07,256 INFO [io.quarkus] (main) Installed features: [cdi, micrometer, mongodb-client, mongodb-panache, mutiny, rest-client, resteasy, resteasy-jsonb, resteasy-mutiny, smallrye-context-propagation, smallrye-health, smallrye-openapi, swagger-ui]
```

このコマンドは終了せずにターミナルを占有します。終了させるとときはCtrl+Cを押してください(Ctrl+Cとは、Ctrlキーを押しながら、Cキーを押すことを意味します)。

注意

mvn quarkus:devコマンドの起動中にエラーになった場合は、MongoDBが起動に失敗した可能性があります。MongoDBを起動したターミナルを見てログにエラーが出力されていないことを確認してください。

2-3-5. PetBattle API OpenAPI UIの表示

- pet-battle-apiの起動ログにはhttp://0.0.0.0:8080でアクセスできると書かれています。Webブラウザーで、<http://localhost:8080>を開くとPetBattle APIのUI画面が開きます。PetBattle APIの画面の上部にある[Open API Documentation]のリンクを開くとSwagger UIの画面が開きます。

Welcome to Pet Battle API !

This page is served by Quarkus

- [Open API Documentation](#)
- [Health](#)
- [Metrics](#)

Next steps

- [Setup your IDE](#)
- [Getting started](#)
- [Quarkus Web Site](#)

Here are the cats.

id	count	isSFF	image
669714ba5e5a300aa7f85679	2	true	
669714bb5e5a300aa7f8567a	4	true	
669714bb5e5a300aa7f8567b	5	true	
669714bb5e5a300aa7f8567c	5	true	
669714bb5e5a300aa7f8567d	2	true	

2-4. PetBattle API Swagger UIの操作 (20分)

Swagger UIはPetBattle APIが提供するAPIをUI画面で呼び出せるようにしたものです。

Swagger UIが提供されることで、これを利用したGUIアプリケーションの開発が容易になります。このSwagger UIはOpenAPIという規格に従って作成したソースコードから自動生成されたものです。

GUI画面を提供するPetBattleと、APIを提供するPetBattle APIを別々のアプリケーションとして分離することで、いくつかのメリットがあります。同じAPIを使用して新しいGUI画面を作ることができますし、CLIコマンドのようなWeb UI以外の手段でAPIを利用することができます。さらに、利用者の数が増えた場合、GUI画面のアプリケーションの数やAPIアプリケーションの数を増やすことで柔軟に対処することができます。

ここでSwagger APIを使って、PetBattle APIの代表的ないくつかのAPIを呼び出してみましょう。



Swagger UI /q/openapi Explore pet-battle-api 1.0.1 (powered by Quarkus 1.13.7.Final)

Generated API 1.0 OAS3

/q/openapi

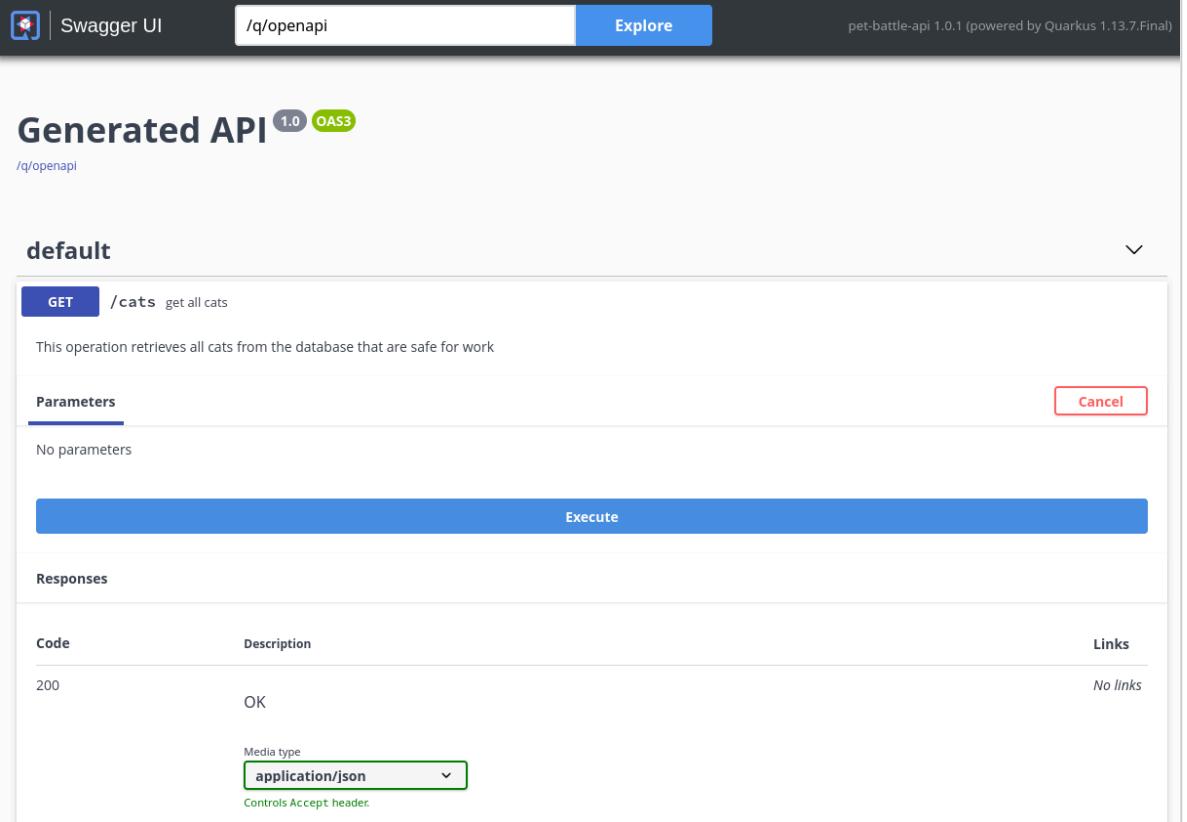
default

▼

GET	/cats	get all cats
POST	/cats	create or update cat
GET	/cats/count	count all cats
GET	/cats/datatable	datatable cats by id
GET	/cats/ids	get all cat ids
DELETE	/cats/kittyciller	⚡ remove all cats ⚡
GET	/cats/loadlitter	preload db with cats
GET	/cats/topcats	get sorted list of top 3 cats by count descending
GET	/cats/{id}	get cat by id
DELETE	/cats/{id}	delete cat by id

2-4-1. GET /cats: すべての猫の情報を返す

1. [GET] /catsをクリックすると以下のように展開されて、[Try it out] をというボタンが現れますので押してください。さらに [Execute] という青いボタンが表示されます。これはAPIを呼び出すボタンです。APIの種類によっては、必要に応じてパラメーターを設定することもできます。



Swagger UI | /q/openapi | Explore | pet-battle-api 1.0.1 (powered by Quarkus 1.13.7.Final)

Generated API 1.0 OAS3

/q/openapi

default

GET /cats get all cats

This operation retrieves all cats from the database that are safe for work

Parameters

No parameters

Responses

Code	Description	Links
200	OK	No links

Media type: application/json
Controls Accept header.

Execute

Cancel

2. [Execute] ボタンを押すと画面中程の [Responses] にAPIの呼び出し結果がJSON形式で表示されます。



Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/cats' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8080/cats
```

Server response

Code	Details
200	<p>Response body</p> <pre>[{"id": "669714ba5e5a300aa7f85679", "count": 2, "image": "data:image/jpeg;base64,/9j/4AAOSkZJgqABAQAAAQABAAAD/2wBDAAgGBqcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHrofHh0aHbwgJ4nICIsIxwckDcpLDAxhD00Hy5PTgyPC4zNDL/2wBDAQkJCQwLDbgNDRgyIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjI/wAARCAEAAEDEA5IAAEBAxE9/8QAHwAAQUBAQEBQEAQAAAAAAAECwQFBgCICo0/8QAtRAAqEDAwIEAwUFBQAAMF9AQIDAAQRBRThNWEGE1FhByJxFDK8kaEII0KxwRVsEfakW2yygkKFhYgrollokSo@NTY30Dk60RFkklSUpTVFWV1hZwmhKzWznaGlc3k1dn4dExphIMh41jjpKTJhWl51zmqjkpMm61pqrKztLW2t7i5usLdxXgXg8xjyjLT1Nw19jz2uH14+T15uf06erx8vP09fb3+Pn6/QAHwEAEBQAEBAQAAAECwQFBgCICo0/8QAtREAgEBAAQDABCFBAQAAQJ3AAECAXEEBSexbhBQdhrRMkmoIEFkRobhBCSMzluVnRLChykNOE18RCYGRomLygpkjU2Nz50kHNERUZHSE1UK1URV1ldWVpjjZGMz2hpanv0dXZ3eH16go0EhyAhilmKkpOU1zaXmJmaoQOkpaangkomsr00tha3uLm6wsPEcbiyhNk0tPU1b2h2N4u4P5e5ben6OngvP09fb3+Pn6/9oADAMBAIRAXeApwDjnObVNRHcegI+ppcDg9sfwCMYBwDVkCLSTSjZX6ChwuSPz0SYIX9RTxkGNvLY7ngG1IHt8BFHV018AjNUJ2CcRk44yal3jquwPsivDER+fb6l7h1BjvvJ0hkVFIlfkKYoc8/Q0DJS0g9ajZiV0+3SnjOP8ajYXx3L/AIUA712v4vmmntg4+vPan0jYPHPCQqKdjpb6jFAhGRVGW9vSmij1I1981IIFBONwA9xQxz05p2Ah2jgl1Pc1lCNbxjxriXgD0Bx1iQR2+1PgqkAY6vE/3ye9Pg41EJhnfYM0Ute467pd4X14+nenMjK4F1CEz1kNymwdzxUg9zxUfJBG3+PyRzZ9KAHS5mtjBw10py0009Mehpgrkce8DFJtw0e1stwQQQKh1JJGKvhE2OeaeuMg8+9rbckjP609gjeuR+FY/P+0tFRfu/wiQdzCkzn1jvTxhR3zeimBsg7cA+ppPuBu4PHlqICQZmwuMdpTHQDQAc84o3KGz1APY1FLMT9zGM96AJAbtzwSemKkzQF13VU3uMs69017AsG1YDHtAyJtWPrntc1Zv0Bz6vx4lPfLXMcw52N1/D1xJ57bbfTRnnDHe9Pv7Ro/3gBvhyCo+1y87TNeRN038P6RFqjXhn1iFUBSpwQx7/gfzxrIPs2hmlU19ip70hB/WVjfdxxlb2ouI+jupGLQynpkCfd9j/OvTMbDP1FVzqzJutDSFOLjqcI/gbUJ2Sw0gPSH8xUEngzhsjbHCR7SiVQsVU9BSMR644pe3m2MTz-Dtaxq1600/ex/jTP+EL1kYThi9z5or08Ywe1Nz2J+o+TD2MTzQ+c9bh/LKM/sAbVc0n/C667j/AFElw/7f8a9L196bk51o+s1D2MTzP8A40/xjcNvFg9f9Nj/xpreEdzJA8i1f8a61x14zs/dzG2177qDD110fkNwPbdffpXm/7gGg65e/vZXBOREp2poog41Kj1d0dOK1wLrxEnQxOM9p19af8A8IbrxaZ/v8AL/]/X167q14e1keGz1TCNog9MwceV7znIBHO80p1zXNOLPMv8AnDdbGMg2/z2T/Gn/wDCia0MfKp9/NT/GvScdkTcmo+sSL9j8ybm8MazCpd7M7QCxkup4HPY1iEjPbz4ycE8V5Tdh/Zus3NtggBtye6nkfpWtk5tpmdsCj9j06inPrUdeCenNT0CAkY1+nWm8adb7nv7uYk1jwe319Kt17jPbvTyhJIPBhfStEcZGM0D1dq/3D+dF5+TiiipswhMgAdovG5aj5bdu4xnLgsCz15/Cg85YvnirEH [Download] @0YlenOKYxzgnqPTsKBdMJJ71JuIwR06VM55GepqbldgDBI9RjNAFKYpHfrX1yOcN3xSvhjuI8ZctjI5qrJuvhjxMGPB9RRYr5qCJuWHAb+1zotb...]</pre> <p>Response headers</p> <pre>content-type: application/json transfer-encoding: chunked</pre>

2-4-2. GET /cats/ids: すべての猫のIDを返す

- 同様に [GET] /cats/ids を呼び出してみます。呼び出し結果には、id, count, voteという組が繰り返し出力されています。次のAPIの呼び出しに使うので、最初の id の値をコピーしてファイルに保存しておいてください(下図では669714ba5e5a300aa7f85679が最初のidになりますが、実行環境によってidの値は異なります)。

GET /cats/ids get all cat ids

This operation retrieves all cat ids from the database

Parameters

No parameters

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/cats/ids' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8080/cats/ids
```

Server response

Code	Details
200	Response body
	[{ "count": 2, "id": "669714ba5e5a300aa7f85679", "vote": false }, { "count": 4, "id": "669714bb5e5a300aa7f8567a", "vote": false },]

2-4-3. GET /cats/{id}: 指定されたIDの猫の情報を返す

1. [GET] /cats/{id} は具体的なIDを引数として渡すとその猫のデータを返すAPIです。直前の/cats API呼び出しで得た id の値をパラメーターとして指定して、このAPIを呼び出してみましょう。



GET /cats/{id} get cat by id

This operation retrieves a cat by id from the database

Parameters

Name Description

id * required
string
(path)

669714ba5e5a300aa7f85679

Cancel

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/cats/669714ba5e5a300aa7f85679' \
-H 'accept: application/json'
```

Request URL

http://localhost:8080/cats/669714ba5e5a300aa7f85679

Server response

Code Details

200 Response body

```
{
  "id": "669714ba5e5a300aa7f85679",
  "count": 2,
  "image": "data:image/jpeg;base64,/9j/4AAQSkZJ... (large JSON object containing image data and other details)"}
```

2. API呼び出し結果として、指定した id の猫情報が [Response body] にJSON形式で表示されています。この猫情報は次のAPI呼び出しで使うのでファイルとしてダウンロードしておいてください。猫情報のダウンロードには、[Response body] の右下に表示されているダウンロードボタンを使うと便利です。ファイルはホームディレクトリのDownloadsディレクトリに保存されます。

TIPS

この結果の猫データには、JPEG形式のイメージデータが含まれています。このデータはBase64エンコーディングで符号化された文字列ですので、以下のようにしてイメージファイルを作成することができます。

```
$ echo "イメージ文字列" | base64 --decode > mycat.jpg
```

イメージの文字列は、"data:image/jpeg;base64,XXXXXX"のXXXXXXの部分になりますので、その部分だけをechoコマンドの引数として渡してください。

2-4-4. POST /cats: 猫データを生成または更新する

- [POST] /cats はPOSTのデータとして猫のJSONデータを渡すと新規の猫を作成します。その際、猫データに既存IDを含めておくと、その猫の情報を更新します。/cats/{id}呼び出しの結果として得られた猫データを使って、猫の投票をしてみましょう。

猫データのJSON形式文字列は、具体的には、以下のような構造をしています。現在の投票値(count)は2になっています。[POST] /cats APIを実行する際に、voteをtrueに指定すると投票値がカウントアップし、falseにするとカウントダウンします。

```
{  
  "id": "669714ba5e5a300aa7f85679",  
  "count": 2,  
  "image":  
"data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAAgGBgcGBQg  
HBwcJCQgKDBQNDAsLDBkSEw8UHRofHh0aHBwgJC4nIClsIxwcKDcpLDAxNDQ0H  
yc5PTgyPC4zNDL  
<略>  
  "vote": false  
}
```

この猫データの文字列をコピーして、[POST] /cats の [Request body] にペーストします。この際、"vote": true に変更してみてください。[Execute] ボタンを押すと成功するはずです。



POST /cats create or update cat

This operation creates or updates a cat (if id supplied)

Parameters

No parameters

Request body

application/json

```
{
    "name": "Fluffy",
    "age": 3,
    "breed": "Siamese",
    "color": "Brown and white"
}
```

Execute

2. 次に、この猫の投票結果を[Get]/cats/{id} APIを使って確認します。直前で使用した猫の id を指定して猫データを取得すると次のように投票結果が2から3にカウントアップしていることがわかります。

Response body

```
{  
  "id": "669714ba5e5a300aa7f85679",  
  "count": 3,  
  "image": "data:image/jpeg;base64,/9j/4AAQSkZJRgABAgAAAQAB  
wcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2wBDAQkJCQwLDBgNDRgyIRwhMjIyM  
RCAEsAOEDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAECawQFBg  
I0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY30Dk6Q0RFRkdISupTVFV  
LW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Tl5uf06erx8vP09fb3+Pn6  
QAAQJ3AAECAxEEBSEExBhJBQdhdRMiMoEIFEKRobHBcSMzUvAVYnLRChYkN  
6goOEhYaHiImKkpOUlZaXmJmaoq0kpaanqKmqsr0tba3uLm6wsPExcbHyM  
cegI+ppckDg9sfWmsCMYBwDVkClST5jZX6ChwuCSPzoY5IX9TTXkGNvLY7m
```

2-5. [オプション] PetBattleとPetBattle APIの間の連携

PetBattleからはHTTPプロトコルによってPetBattle APIにリクエストが送信されます。

PetBattle APIにリクエストが到達したとき、アクセスがあったことをログ記録することができます。これをアクセスログと言います。アクセスログを出力することで、GUI画面の操作をした結果、どのようなAPIが呼び出されたかを確認することができます。PetBattle APIのアクセスログの設定はデフォルトでは無効になっていますので、有効にしてアクセスログへの出力を確認します。



1. PetBattle APIアプリケーションを [2-2-4](#) の手順に従って起動しておきます。
2. PetBattle APIのアクセスログを有効にするため、
src/main/resources/application.propertiesファイルの最後に以下の1行を追加し、
Ctrl+Sキーを押して保存します。

```
quarkus.http.access-log.enabled=true
```

3. [2-2-3](#) の手順に従ってpet-battle-apiコンテナーイメージを作り直します。
4. [2-2-4](#) の手順に従ってpet-battle-apiコンテナーを起動します。
5. PetBattle APIのUI画面からGET /cats APIを呼び出して、PetBattle APIのログに以下のようなアクセスログが記録されることを確認します。

```
2024-07-15 18:28:56,467 INFO [io.qua.htt.access-log] (executor-thread-1)  
0:0:0:0:0:0:1 - - 15/Jul/2024:18:28:56 +0900 "GET /cats HTTP/1.1" 200 244573
```

参考リンク

QuarkusアプリケーションのHTTPアクセスログを制御するためのプロパティは以下のドキュメントに説明があります。

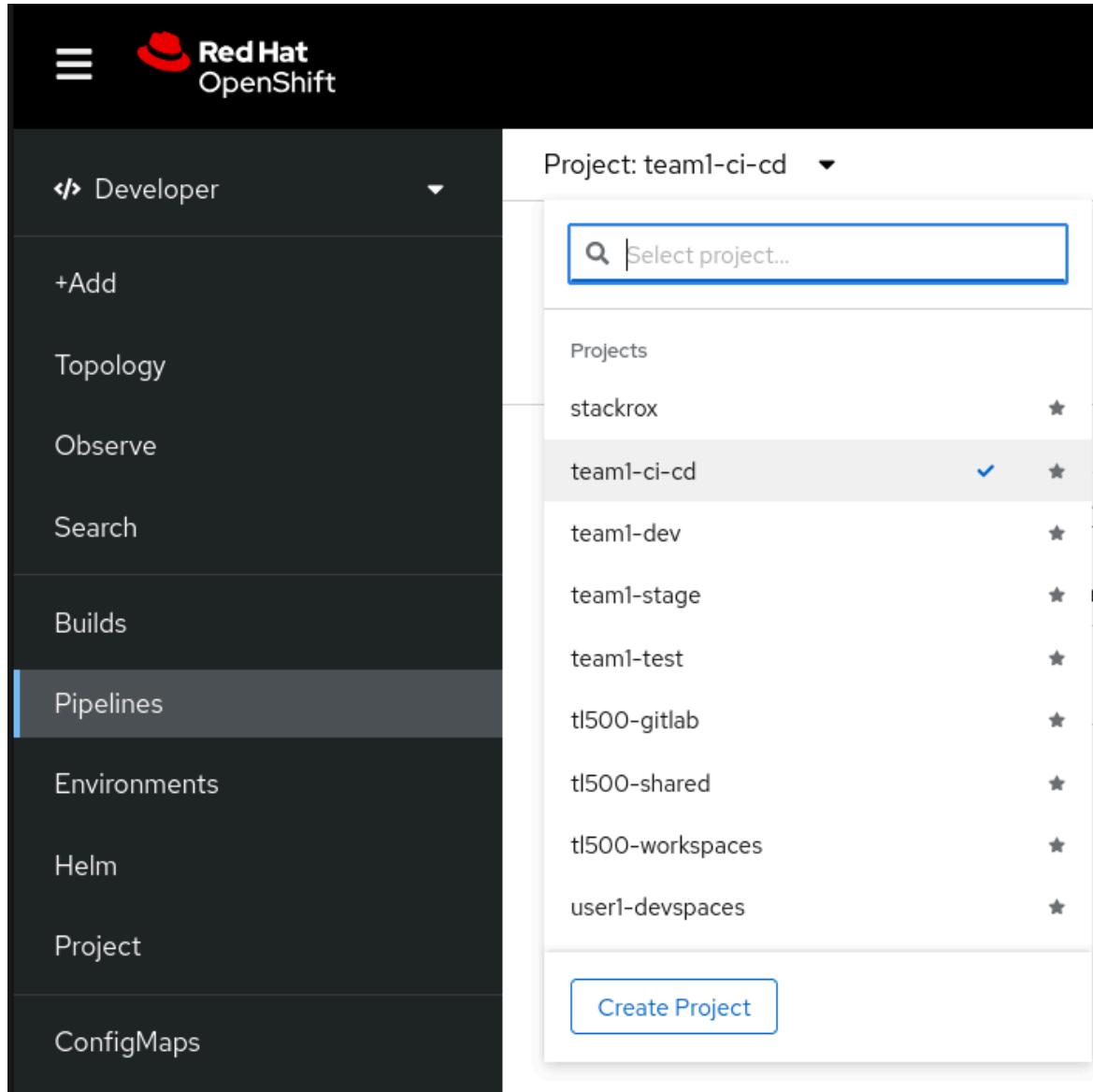
<https://ja.quarkus.io/guides/http-reference#configuring-http-access-logs>

3. CI/CDパイプライン (40分)

目的	<ul style="list-style-type: none">• CI/CDパイプラインの目的と構造を理解する
目標	<ul style="list-style-type: none">• パイプラインを構成するタスクを調べる• パイプラインを実行してログを調べる

3-1. PetBattle API パイプライン全体の構造を調べる (10分)

1. OpenShift WebコンソールのDeveloperペースペクティブにおいて [Pipelines]を選択します。上部のプロジェクト選択メニューで <チーム名>-ci-cdプロジェクトを選択します。



The screenshot shows the Red Hat OpenShift Pipelines interface. On the left, a sidebar menu includes options like Developer, +Add, Topology, Observe, Search, Builds, Pipelines (which is selected and highlighted in blue), Environments, Helm, Project, and ConfigMaps. On the right, a main panel titled "Project: team1-ci-cd" displays a list of projects. A search bar at the top of the list says "Select project...". Below it, the "team1-ci-cd" project is listed with a checkmark next to it. Other projects listed include stackrox, team1-dev, team1-stage, team1-test, tl500-gitlab, tl500-shared, tl500-workspaces, and user1-devspaces. At the bottom of the list is a "Create Project" button.

2. [Pipelines]の画面の[Pipelines]タブにパイプライン一覧が表示されます。
maven-pipelineという名前のリンクをクリックします。



The screenshot shows the Red Hat OpenShift web interface. The left sidebar has a 'Developer' section with options like '+Add', 'Topology', 'Observe', 'Search', 'Builds', 'Pipelines' (which is selected), 'Environments', 'Helm', 'Project', 'ConfigMaps', and 'Secrets'. The main area is titled 'Pipelines' and shows a table with a single row for 'maven-pipeline'. The table columns are 'Name', 'Last run', 'Task status', 'Last run status', and 'Last run time'. A 'Create' button is located in the top right corner.

3. [Pipelines] > [Pipelines details] の画面でmaven-pipelineパイプラインの全体の構成図が表示されます。

The screenshot shows the 'Pipeline details' page for 'maven-pipeline'. At the top, it says 'Project: team1-ci-cd'. Below that, it shows the pipeline name 'maven-pipeline' and a 'Actions' dropdown. The main area is titled 'Pipeline details' and contains a diagram of the pipeline tasks: 'fetch-app-repository' → 'maven' → 'bake' → 'helm-package' → 'fetch-argocd-configuration' → 'deploy-test' → 'verify-deployment'. Below the diagram are search and filter icons. To the left, there are sections for 'Name' (maven-pipeline), 'Namespace' (team1-ci-cd), and 'Labels' (rht-gitops.com/team1-ci-cd+tekton-pipeline). To the right, there are sections for 'TriggerTemplates' (pet-battle-api-trigger-template) and 'Tasks' (git-clone, maven, bake, helm-package, git-clone, deploy, verify-deployment).

パイプラインは複数のタスクの連鎖によって構成されます。このパイプラインは、(1) **fetch-app-repository** タスク、(2) **maven**タスク、(3) **bake**タスクのように左から実行されています。

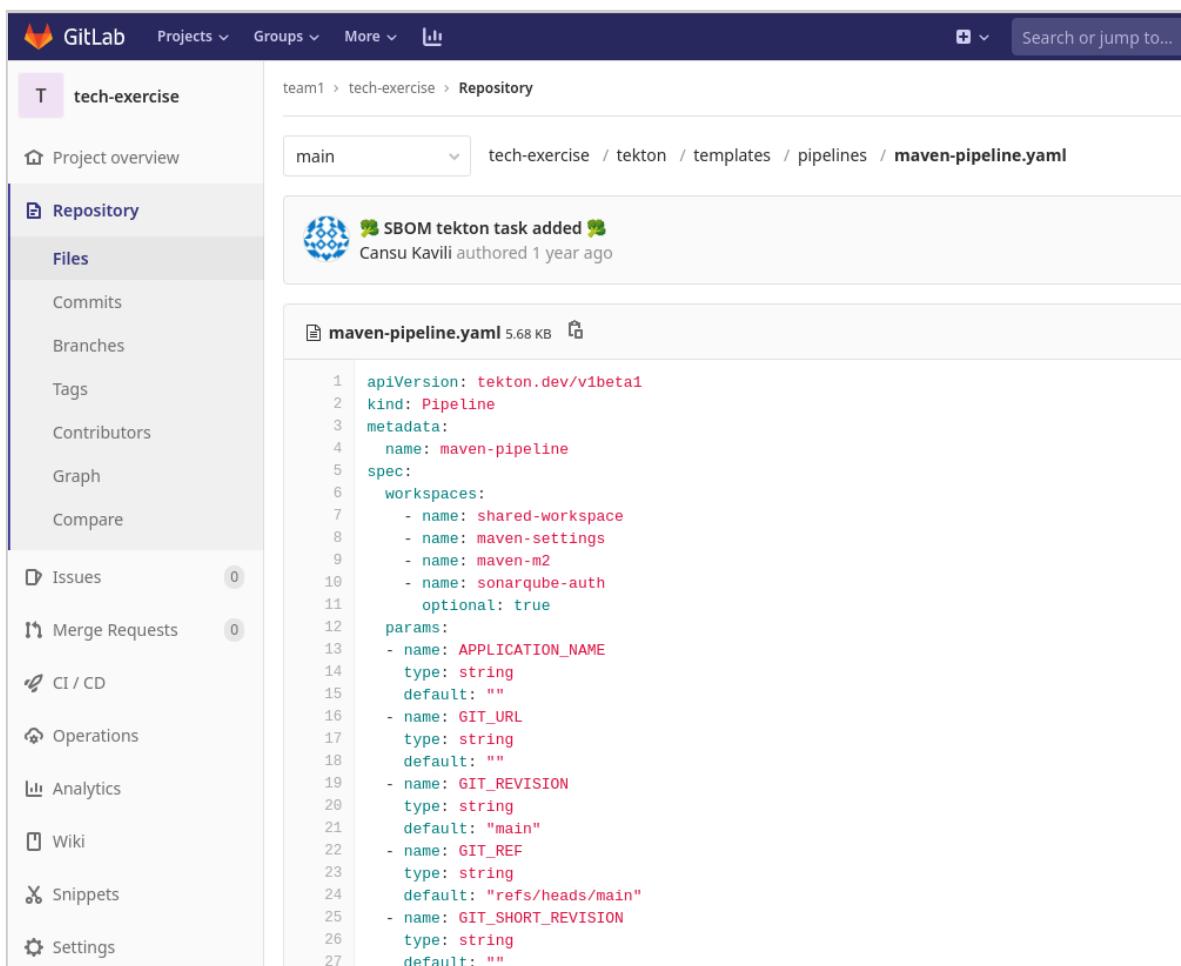
PetBattle APIパイプラインは、GitLabのpet-battle-apiプロジェクトにgit pushされたことがトリガーとなって開始されます(この仕組みをWebHookと言います)。WebHookでは、GitLabからOpenShiftに対してソースコードの変更が発生したことが通知されます。この際、変更のあったGitプロジェクト情報がOpenShiftに渡されます。

OpenShiftはWebHookの通知を受けるとパイプラインの処理を開始します。すべてのタスクが成功すると、最終的にPetBattle APIアプリケーションがOpenShiftにデプロイされます。途中で処理に失敗したタスクがあると、パイプライン全体が停止します。

3-1-1. PetBattle API パイプラインを構成するタスクを調べる。

PetBattle API パイプライン定義は、GitLab tech-exchangeプロジェクト内の tekton/templates/pipelines/maven-pipeline.yamlというファイルで保存されています(つまり、パイプライン定義もソースコードとして管理されています)。[2-1](#) の手順に従い、OpenShift上のGitLabアプリケーションの画面を開いて、パイプライン定義のファイル maven-pipeline.yamlを開いてください。パイプラインの設定内容について大まかな構造を確認していきましょう。

3-1-2. workspaces宣言



The screenshot shows the GitLab interface for the 'tech-exercise' project. The left sidebar has 'Repository' selected. The main area shows a commit message: 'SBOM tekton task added' by Cansu Kavili 1 year ago. Below it is the 'maven-pipeline.yaml' file content:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: maven-pipeline
spec:
  workspaces:
    - name: shared-workspace
    - name: maven-settings
    - name: maven-m2
    - name: sonarqube-auth
      optional: true
  params:
    - name: APPLICATION_NAME
      type: string
      default: ""
    - name: GIT_URL
      type: string
      default: ""
    - name: GIT_REVISION
      type: string
      default: "main"
    - name: GIT_REF
      type: string
      default: "refs/heads/main"
    - name: GIT_SHORT_REVISION
      type: string
      default: ""
```



maven-pipeline.yaml の7行から11行は、パイプライン全体で使用するワークスペースの宣言になります。ワークスペースは、パイプラインを実行中の状態を保持するストレージです。タスク間で同じファイルを参照する場合にはこのワークスペースに置きます。

3-1-3. params宣言

maven-pipeline.yaml の12行から48行は、パイプライン全体で使用するパラメーターの宣言になります。パラメーターによってパイプライン起動時に外部から値を設定することが可能になる仕組みです。パイプラインで定義されている各種パラメーターは、パイプラインを起動するWebHookリクエストに含まれた情報を元に設定されます。

```
params:  
- name: APPLICATION_NAME  
  type: string  
  default: ""  
- name: GIT_URL  
  type: string  
  default: ""  
- name: GIT_REVISION  
  type: string  
  default: "main"  
- name: GIT_REF  
  type: string  
  default: "refs/heads/main"  
- name: GIT_SHORT_REVISION  
  type: string  
  default: ""  
- name: GIT_BRANCH  
  type: string  
  default: ""  
<略>
```

3-1-4. git-cloneタスク

maven-pipeline.yaml の50行目からfetch-app-repositoryというタスク定義です。これはtaskRefでgit-cloneという既定義のタスクを参照しています。このタスクを動かすための各種パラメーターはparamsの下に定義されています。\$(params.GIT_URL)のような書き方はパイプラインの先頭のparamsで定義されたGIT_URLパラメーターを参照するという意味です。

```
tasks:
```

```
- name: fetch-app-repository
  taskRef:
    name: git-clone
    kind: ClusterTask
  workspaces:
    - name: output
      workspace: shared-workspace
  params:
    - name: url
      value: "${params.GIT_URL}"
    - name: revision
      value: "main"
    - name: subdirectory
      value: "${params.APPLICATION_NAME}/${params.GIT_BRANCH}"
    - name: deleteExisting
      value: "true"
    - name: sslVerify
      value: "false"
```

TIPS

git-cloneタスクの定義はこちらのページを参照してください。

<https://hub.tekton.dev/tekton/task/git-clone>

3-1-5. mavenタスク

maven-pipeline.yaml の78行目からはmavenというタスク定義です。これはmavenコマンドを使ってJavaソースコードのビルドを実施します。

```
- name: maven
  taskRef:
    name: maven
  runAfter:
    - fetch-app-repository
  params:
    - name: WORK_DIRECTORY
      value: "${params.APPLICATION_NAME}/${params.GIT_BRANCH}"
    - name: GOALS
      value: "package"
    - name: MAVEN_BUILD_OPTS
      value: "-Dquarkus.package.type=fast-jar -DskipTests"
  workspaces:
    - name: maven-settings
```

```
workspace: maven-settings
- name: maven-m2
  workspace: maven-m2
- name: output
  workspace: shared-workspace
- name: sonarqube-auth
  workspace: sonarqube-auth
```

taskRefでmavenという既定義のタスクを参照しています。実際のmavenコマンドを呼び出す手順についてはそのタスク定義に書かれています。

runAfterによってどのタスクの後にこのタスクが呼び出されるかが指定されています。

TIPS

mavenタスクの定義はこちらのページを参照してください。

<https://hub.tekton.dev/tekton/task/maven>

3-2. PetBattle API パイプラインを実行する(30分)

PetBattle APIのソースコードをGitLabのpet-battle-apiリポジトリにgit pushすることによってパイプラインが起動します。実際にPetBattle APIのソースコードの一部を変更してパイプラインを起動し、修正を反映したアプリケーションがデプロイされることを確認しましょう。

3-2-1. PetBattle APIのソースコードを修正する

1. 2-3-1と同様の方法で、GitLabからpet-battle-apiプロジェクトをgit cloneします。

```
git clone https://gitlab-ce.apps.ocp4.example.com/<チーム名>/pet-battle-api.git
```

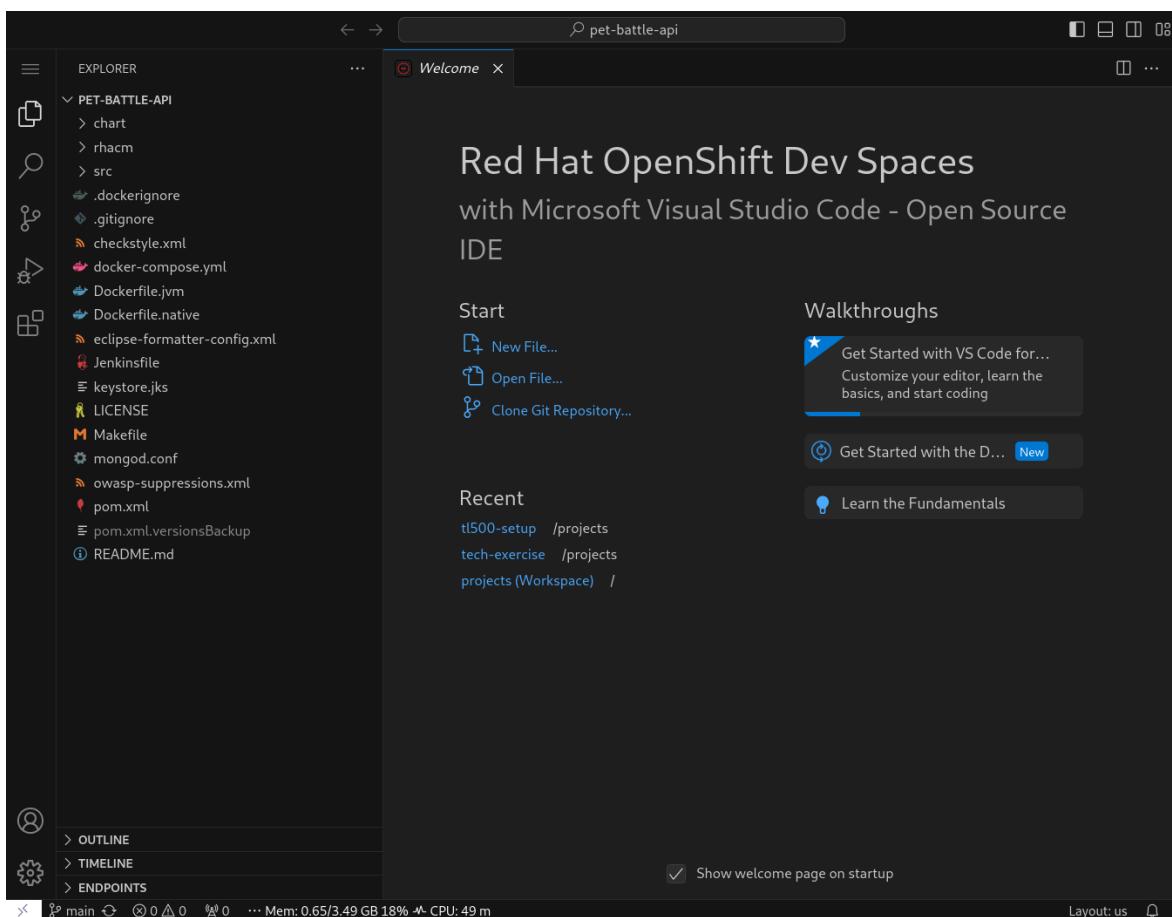
以下のコマンドは、team1の場合の例になります。

```
[student@workstation ~]$ git clone  
https://gitlab-ce.apps.ocp4.example.com/team1/pet-battle-api.git
```

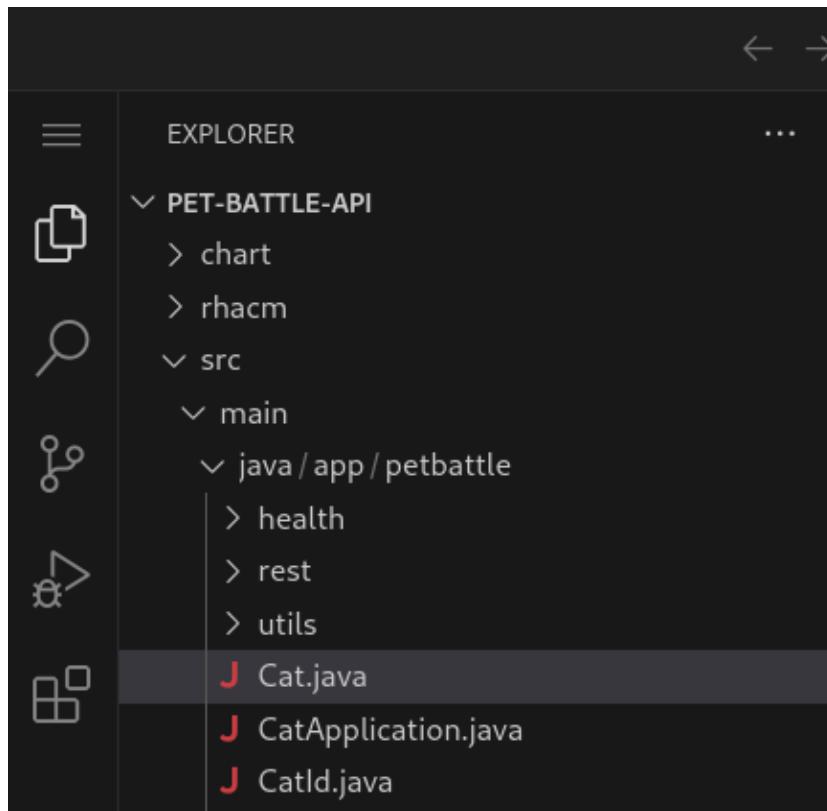
2. Workstationでターミナルを開き、VSCodeを起動します。

```
[student@workstation ~]$ code pet-battle-api
```

3. VSCodeが起動するとpet-battle-apiのファイル階層が左側に表示されます。



4. ディレクトリ階層の中でCat.javaファイルを探してください。ディレクトリ名をクリックするとサブディレクトリが展開されます。PET-BATTLE-API > src > main > java / app / petbattleの下にCat.javaが見つかるはずです。Cat.javaが見つかったらファイル名をクリックしてください。
5. 画面右側にCat.javaのソースコードが表示されています。56行-62行にvote()という名前のメソッド定義がありますので探してください。これは猫に投票するときのロジックをコーディングしている箇所です。getCount()で現在の投票値を調べ、それに対してGoodで+1、Badで-1しています。



```
public void vote() {  
    if (vote) {  
        setCount(getCount() + 1);  
    } else {  
        setCount(getCount() - 1);  
    }  
}
```

6. このコードを以下のように修正します。Goodで+10、Badで-10に変更しています。修正が完了したら、Ctrl+S キーを押してファイルを保存してください。

```
public void vote() {  
    if (vote) {  
        setCount(getCount() + 10);  
    } else {  
        setCount(getCount() - 10);  
    }  
}
```

3-2-2. PetBattle APIリポジトリへの変更のコミットとプッシュ

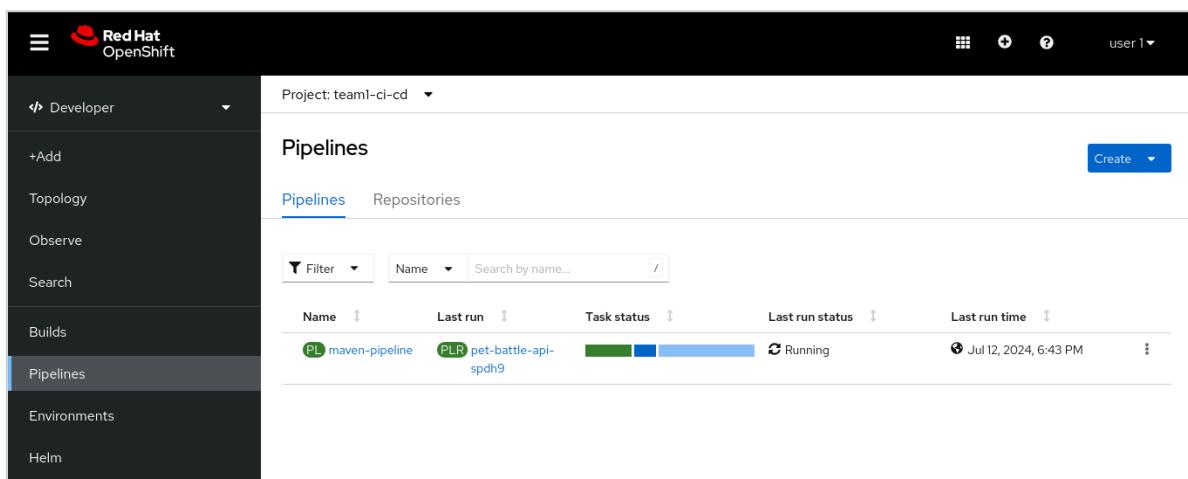
1. Workstationのターミナルを開きます。ターミナル内で以下のコードを実行して修正をGitリポジトリにpushしてください。

```
[student@workstation ~]$ cd pet-battle-api  
[student@workstation pet-battle-api]$ git add .  
[student@workstation pet-battle-api]$ git commit -m 'Changed vote logic'  
[student@workstation pet-battle-api]$ git push
```

git pushのコマンドで、UsernameとPasswordの入力をします。これらの値はGitLabのログインのときに使ったものと同じです。ファシリテーターから与えられたものを使ってください。

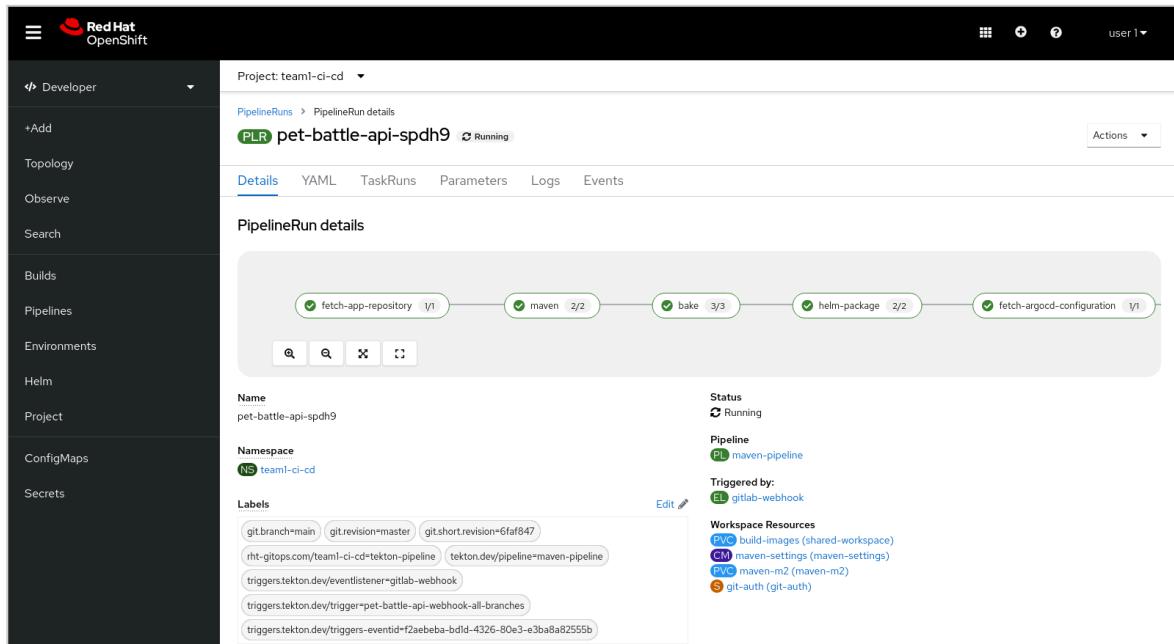
3-2-3. PetBattle APIパイプラインの起動確認

1. [3-1](#) の手順に従って、OpenShift Webコンソールからパイプラインの表示をします。[Last run] の下のリンクをクリックしてください。



Name	Last run	Task status	Last run status	Last run time
PL maven-pipeline	PLR pet-battle-api-spdh9	<div style="width: 100px; height: 10px; background-color: #007bff;"></div>	Running	Jul 12, 2024, 6:43 PM

2. [PipelineRuns] > [PipelineRuns details] の画面で、現在実行中のパイプラインの状態を観察することができます。パイプラインの処理がすべて終了するまで10分～20分くらいかかります。



The screenshot shows the Red Hat OpenShift web interface. On the left, a sidebar menu includes options like Developer, Topology, Observe, Search, Builds, Pipelines, Environments, Helm, Project, ConfigMaps, and Secrets. The main content area displays a PipelineRun titled 'pet-battle-api-spdh9' under the 'team1-ci-cd' project. The PipelineRun status is 'Running'. The pipeline consists of five sequential steps: 'fetch-app-repository' (1/1), 'maven' (2/2), 'bake' (3/3), 'helm-package' (2/2), and 'fetch-argocd-configuration' (1/1). All steps are marked as completed with green checkmarks. The 'Details' tab is selected. A 'Workspace Resources' section lists several shared resources.

パイプラインの処理が完了したら、PetBattle APIアプリケーションを起動して変更が反映されていることを確認します。

重要な注意

パイプラインの実行中にエラーで停止した場合はファシリテーターまでお知らせください。クラウド上のマシンリソースの不足により稀にエラーが発生することがあります。

3-2-4. PetBattle APIアプリケーションの修正確認

- 1-1 の手順に従い、OpenShift上にデプロイされたPetBattleアプリケーションのGUI画面を開きます。
- 特定の猫の現在のカウントを確認し、その猫に投票します。
- 投票の結果としてカウントが10増えていることを確認します。

4. テストの自動化 (40分)

目的	<ul style="list-style-type: none"> パイプラインにおける自動テストの意味を理解する
目標	<ul style="list-style-type: none"> 静的テストをパイプラインに追加する 動的テストをパイプラインを追加する Allureを使ってテスト結果を検証する



パイプラインは、GitLabのtech-exchangeプロジェクトで定義されています。パイプラインの拡張をする場合はtech-exchange上のファイルを更新します。tech-exchangeプロジェクトにgit pushするとパイプラインが更新されるように設定済みです。

[3-2-1](#) で示した通り、パイプラインを開始するには、GitLabのpet-battle-apiプロジェクトにgit pushします。

4.1. PetBattle API パイプラインに静的テストを追加する (20分)

1. [2-3-1](#) と同様の方法で、GitLabからtech-exerciseプロジェクトをgit cloneします。

```
git clone https://gitlab-ce.apps.ocp4.example.com/<チーム名>/tech-exercise.git
```

以下のコマンドは、team1の場合の例になります。

```
[student@workstation ~]$ git clone  
https://gitlab-ce.apps.ocp4.example.com/team1/tech-exercise.git
```

2. Workstationでターミナルを開き、VSCodeを起動します。

```
[student@workstation ~]$ codium tech-exercise
```

3. パイプラインに静的解析 (code-analysis) のタスクを追加します。

tech-exercise/tekton/templates/pipelines/maven-pipeline.yamlファイルを編集し、mavenビルドの前にcode-analysisを追加します。

重要な注意

[YAML構文](#)は、インデントによってデータ構造を定義します。インデントの深さは以下とまったく同一にしないと動作しないので注意してください。インデントはスペースのみを使います。タブは使わないでください。

```
# Code Analysis
  - name: code-analysis
    taskRef:
      name: maven
    params:
      - name: WORK_DIRECTORY
        value: "${params.APPLICATION_NAME}/${params.GIT_BRANCH}"
      - name: GOALS
```

```
        value: "test sonar:sonar"
      - name: MAVEN_BUILD_OPTS
        value: "-Dsonar.host.url=http://sonarqube-sonarqube:9000
-Dsonar.userHome=/tmp/sonar" # <- ここは1行で入力
      runAfter:
        - fetch-app-repository
    workspaces:
      - name: maven-settings
        workspace: maven-settings
      - name: maven-m2
        workspace: maven-m2
      - name: output
        workspace: shared-workspace
      - name: sonarqube-auth
        workspace: sonarqube-auth
```

4. また、パイプラインの実行をトリガーするときに、sonarqube-authワークスペースをシークレットにバインドする必要があります。これを行うには
tekton/templates/triggers/gitlab-trigger-template.yamlファイルを編集し、# sonarqube-authプレースホルダーがあるworkspaces listの末尾にこのコードを追加します。

```
# sonarqube-auth
  - name: sonarqube-auth
    secret:
      secretName: sonarqube-auth
```

5. tech-exercise/tekton/templates/pipelines/maven-pipeline.yamlファイルを編集し、code-analysis-checkステップをパイプラインに追加します。ここから参照されている sonarqube-quality-gate-checkは、tekton/templates/tasks/の下の sonarqube-quality-gate-check.yamlに定義があります。

```
# Code Analysis Check
  - name: analysis-check
    retries: 1
    taskRef:
      name: sonarqube-quality-gate-check
    workspaces:
      - name: output
        workspace: shared-workspace
      - name: sonarqube-auth
```



```
workspace: sonarqube-auth
params:
- name: WORK_DIRECTORY
  value: "${params.APPLICATION_NAME}/${params.GIT_BRANCH}"
runAfter:
- code-analysis
```

6. mavenビルドのrunAfterをanalysis-checkに設定して、アプリケーションをコンパイルする前に静的分析が行われるようにします。

```
- name: maven
  taskRef:
    name: maven
  runAfter:
    - analysis-check # <- ここを更新します
  <略>
```

7. パイプライン定義をGitLabにプッシュします。

```
cd /home/student/tech-exercise
git add .
git commit -m "ADD - code-analysis & check steps"
git push origin main
```

8. パイプラインビルドをトリガーします。空のコミットをリポジトリにプッシュして、パイプラインをトリガーできます。

```
cd /home/student/pet-battle-api
git commit --allow-empty -m "TEST - running code analysis steps"
git push origin main
```

9. OpenShift WebコンソールのDeveloperペーススペクティブを選択し、[Pipelines] > [Pipelines details] の画面でmaven-pipelineパイプラインを確認します。パイプラインは以下のように拡張されました。



Project: team2-ci-cd

PipelineRuns > PipelineRun details

PLR pet-battle-api-k2hwt Succeeded

Actions ▾

Details YAML TaskRuns Logs Events

PipelineRun details

Name: pet-battle-api-k2hwt Status: Succeeded

Namespace: NS team2-ci-cd Pipeline: PL maven-pipeline

Labels: git.branch=main, git.revision=master, git.short.revision=8acbd25, rht-gitops.com/team2-ci-cd:tekton-pipeline, tekton.dev/pipeline=maven-pipeline, triggers.tekton.dev/eventlistener=gitlab-webhook, triggers.tekton.dev/trigger=pet-battle-api-webhook-all-branches, triggers.tekton.dev/triggers-eventid=85ec4521-2ea3-40fa-900a-389b0c0a2394

Triggered by: EL gitlab-webhook

Workspace Resources:

- PVC build-images (shared-workspace)
- CM maven-settings (maven-settings)
- PVC maven-m2 (maven-m2)
- S git-auth (git-auth)
- S sonarqube-auth (sonarqube-auth)

10. パイプラインが完了すると、SonarQube UI で結果を調べることができます。SonarQube UIのURLを得るには以下を実行します。

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

11. SonarQubeのUI画面は以下のように表示されます。

Search by project name or key Create Project ▾

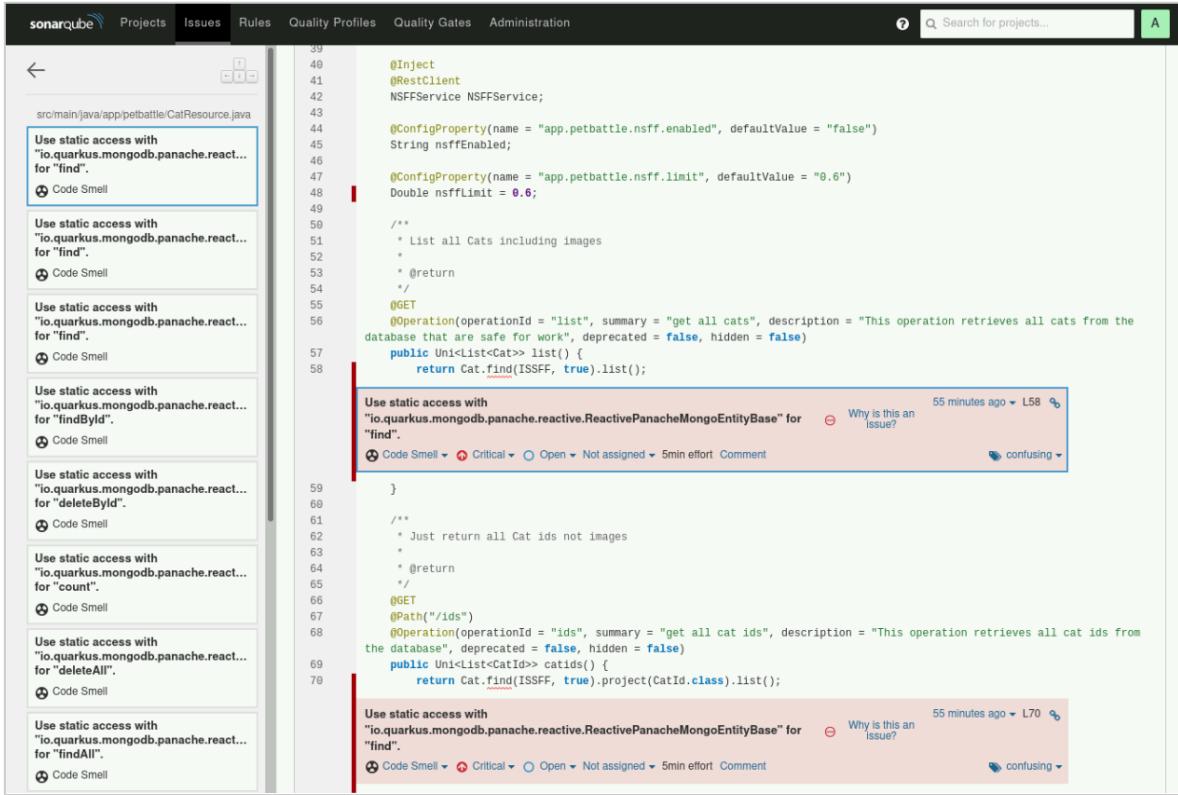
1 projects Perspective: Overall Status Sort by: Name

Last analysis: 51 minutes ago

Bugs: 1 D	Vulnerabilities: 0 A	Hotspots Reviewed: 0.0% E	Code Smells: 72 A	Coverage: 0.0% O	Duplications: 0.0% G	Lines: 1.3k S Java, XML
-----------	----------------------	---------------------------	-------------------	------------------	----------------------	-------------------------

1 of 1 shown

12. Issuesタブを開き、Criticalな箇所のソースコードを開きます。このように問題が検出されたソースコードを調べることができます。



The screenshot shows the SonarQube interface for a Java application. The main area displays a Java file with several static code analysis issues highlighted in blue boxes. These issues include:

- Use static access with "io.quarkus.mongodb.panache.react..." for "find". (Code Smell)
- Use static access with "io.quarkus.mongodb.panache.react..." for "find". (Code Smell)
- Use static access with "io.quarkus.mongodb.panache.react..." for "find". (Code Smell)
- Use static access with "io.quarkus.mongodb.panache.react..." for "findById". (Code Smell)
- Use static access with "io.quarkus.mongodb.panache.react..." for "deleteById". (Code Smell)
- Use static access with "io.quarkus.mongodb.panache.react..." for "count". (Code Smell)
- Use static access with "io.quarkus.mongodb.panache.react..." for "deleteAll". (Code Smell)
- Use static access with "io.quarkus.mongodb.panache.react..." for "findAll". (Code Smell)

Each issue has a detailed description, severity (e.g., Critical), status (e.g., Open), and a comment section. The interface also includes navigation buttons like back, forward, and search, as well as a top menu for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration.

4-1. PetBattle API パイプラインにテストタスクを追加する(20分)

1. Mavenパイプライン

(tech-exercise/tekton/templates/pipelines/maven-pipeline.yaml)に save-test-resultsステップを追加します。

```
# Save Test Results
- name: save-test-results
  taskRef:
    name: allure-post-report
  params:
    - name: APPLICATION_NAME
      value: "${params.APPLICATION_NAME}"
    - name: WORK_DIRECTORY
      value: "${params.APPLICATION_NAME}/${params.GIT_BRANCH}"
  runAfter:
    - maven
  workspaces:
    - name: output
      workspace: shared-workspace
```

2. MavenパイプラインのmavenタスクからskipTests引数を削除します。

変更前:

```
- name: maven
  params:
    - name: MAVEN_BUILD_OPTS
      value: "-Dquarkus.package.type=fast-jar -DskipTests"
```

変更後:

```
- name: maven
  params:
    - name: MAVEN_BUILD_OPTS
      value: "-Dquarkus.package.type=fast-jar"
```

3. パイプラインを更新するため、GitLabにプッシュします。

```
cd /home/student/tech-exercise
git add .
git commit -m "ADD - save-test-results step"
git push
```

4. 空のコミットで新しいPipelineRunをトリガーし、OpenShift Pipelines に移動して実行を確認します。

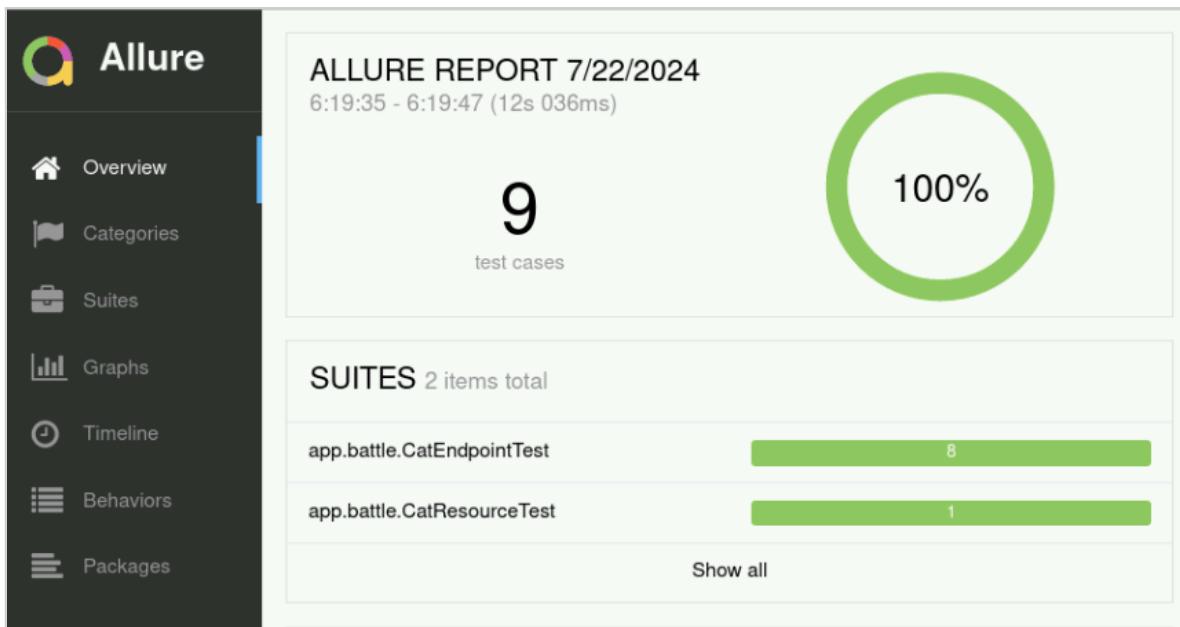
```
cd /home/student/pet-battle-api
git commit --allow-empty -m "test save-test-results step"
git push
```

5. OpenShift Webコンソールでパイプラインが拡張されたことを確認してください。

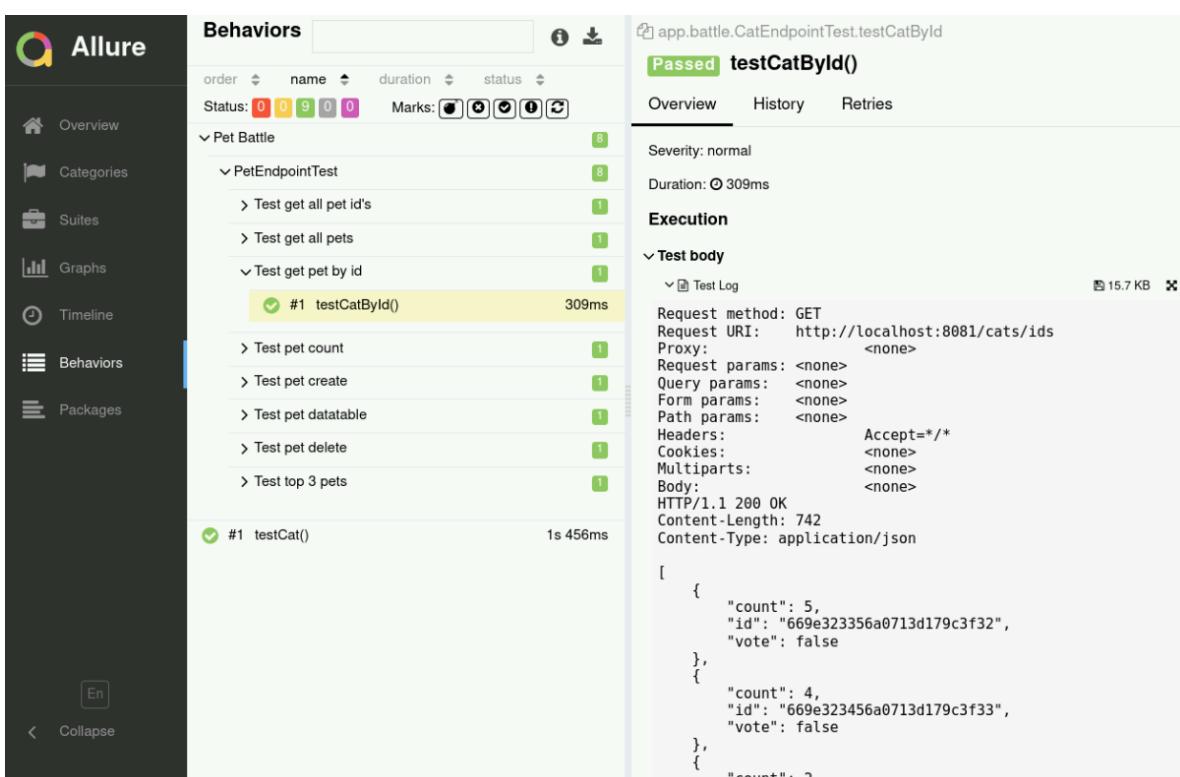
6. パイプラインを実行するとテスト結果はAllureに保存されます。Allure UIのURLを得るには以下を実行します(1行で入力してください)。

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

7. Allure UIのトップ画面です。



8. テスト結果の詳細表示もできます。



参考リスト

- [1] Tech Exercise <https://github.com/rht-labs/tech-exercise>
- [2] PetBattle <https://github.com/petbattle/pet-battle>
- [3] PetBattle API <https://github.com/petbattle/pet-battle-api>
- [4] Tekton Hub <https://hub.tekton.dev/>

