

Red Hat トレーニング

DevOps Culture & Practice (TL500)

技術演習手順書

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

0-2. 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

0-3-3. CodeReady Workspaces

1. コンテナー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 (60分)

2-1. Gitコマンドの基礎 (20分)

2-1-1. GitLab アプリケーション

2-1-2. Gitリポジトリの作成

2-1-3. Gitコマンドの基本操作

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

2-2-1. PetBattle API 開発環境の設定

2-2-1. PetBattle API ソースコードのダウンロード

2-2-1. PetBattle API ソースコードのビルド

2-2-3. PetBattle API コンテナーイメージ作成

2-2-4. PetBattle API MongoDB コンテナーの起動

2-2-4. PetBattle API コンテナーの起動

2-2-5. PetBattle API OpenAPI UI の表示

2-3. PetBattle API Swagger UI の操作 (20分)

2-3-1. GET /cats: すべての猫の情報を返す

2-3-2. GET /cats/ids: すべての猫のIDを返す

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

2-3-4. POST /cats: 猫データを生成または更新する

2-4. 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

TODO: 以下の図を描いて概要を説明する。

- OpenShiftクラスター
- PetBattleを支えるインフラ

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。

0-2. Red Hat Online Learning Portal

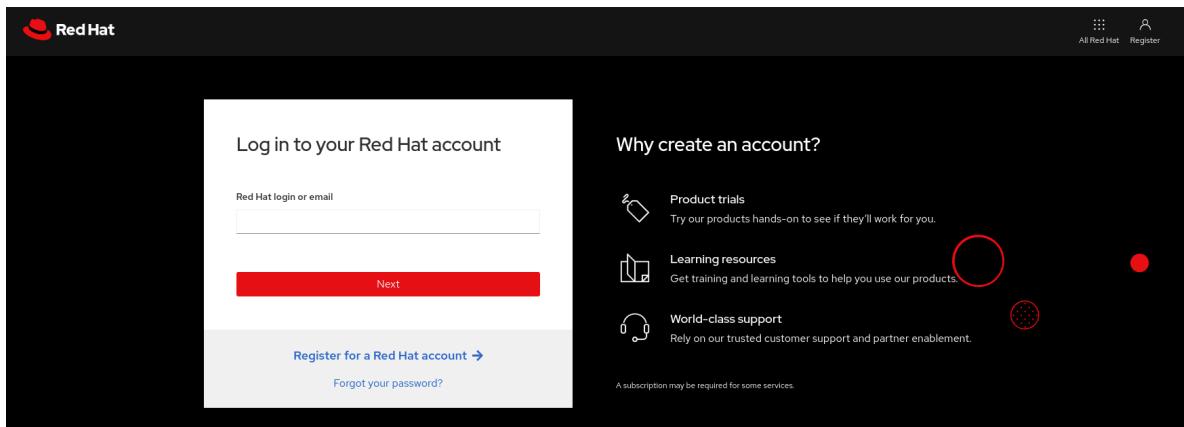
技術演習環境で使うOpenShiftクラスターは、Red Hatのクラウド環境で動作し、Red Hat Online Learning Portal(以後、ROLポータル)からアクセスします。

0-2-1. TL500 ダッシュボードへのアクセス

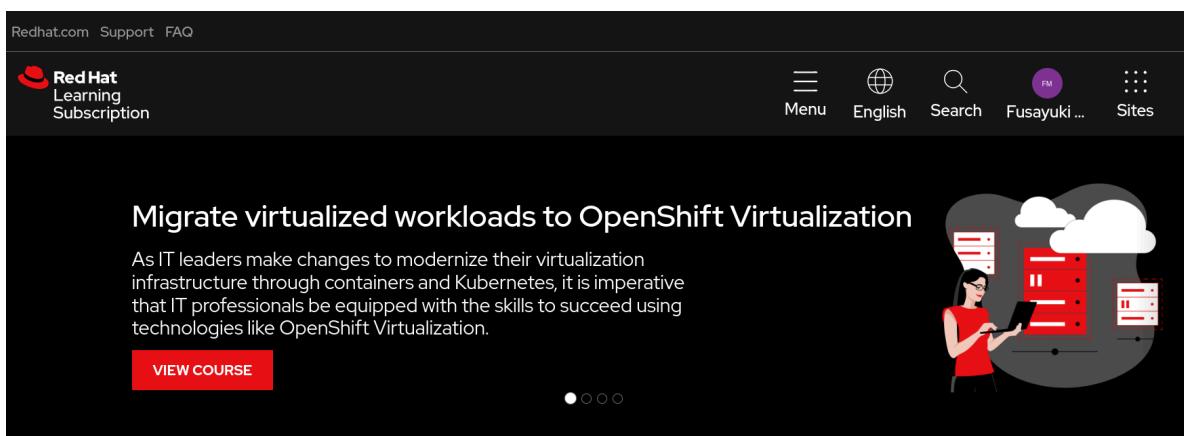
- ROLポータルにアクセスするには、Webブラウザーから以下のURLを開きます。

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

- 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

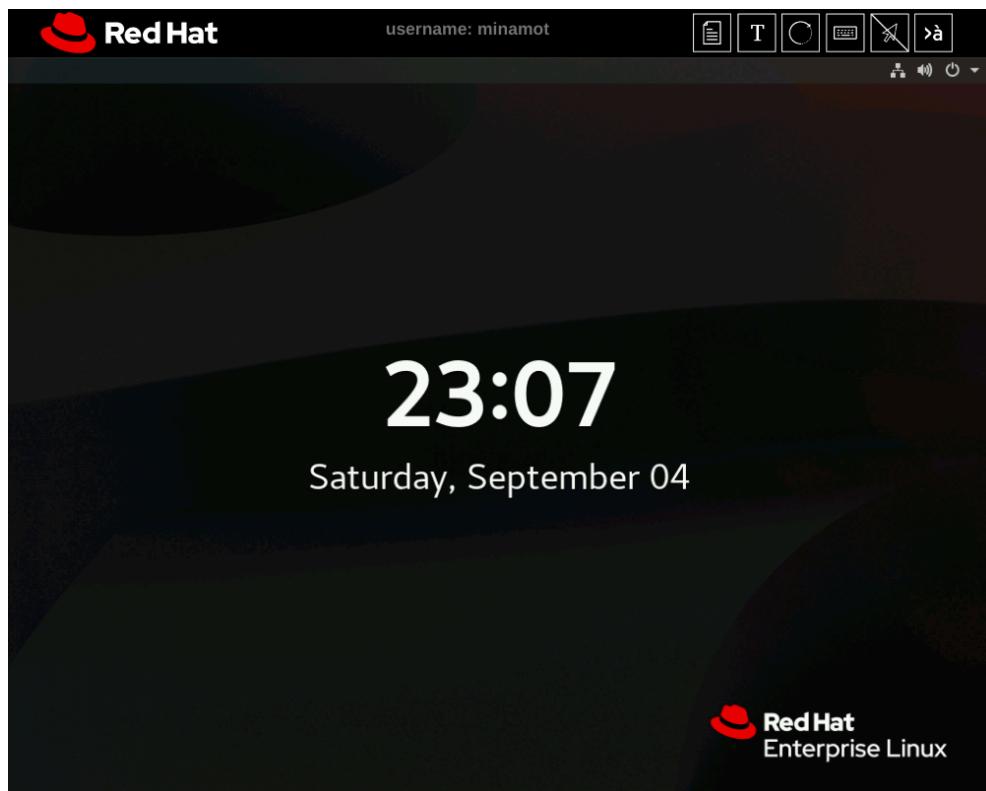
Auto-stop in 16 days.
Auto-destroy in 16 days.

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。

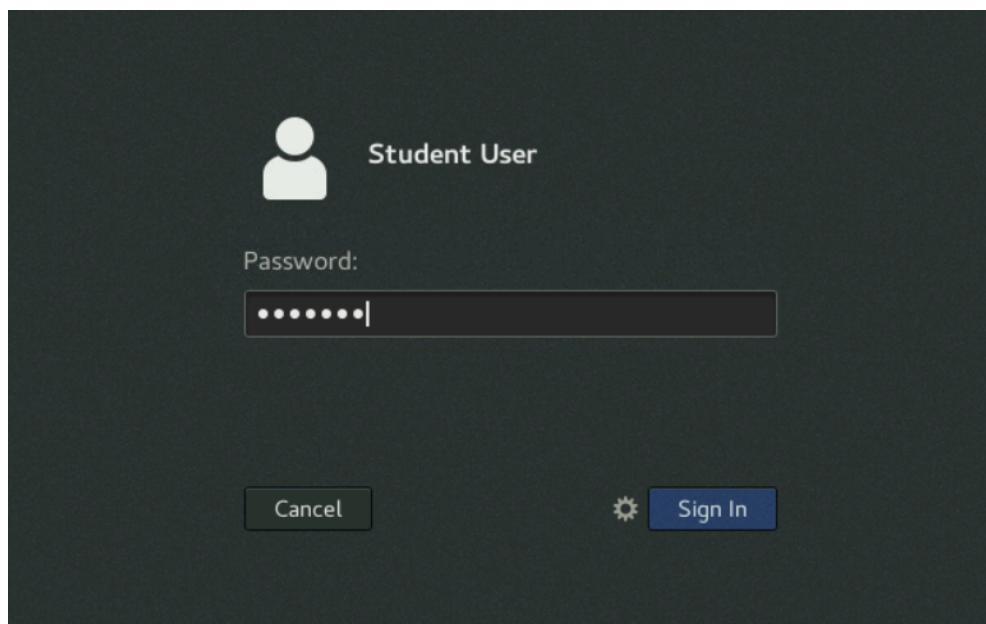
4. workstationと表示された行の[OPEN CONSOLE] ボタンを押すと、Webブラウザーの新しいタブにWorkstation仮想マシンの画面が開きます。

0-2-3. Workstationへのログイン

1. 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。



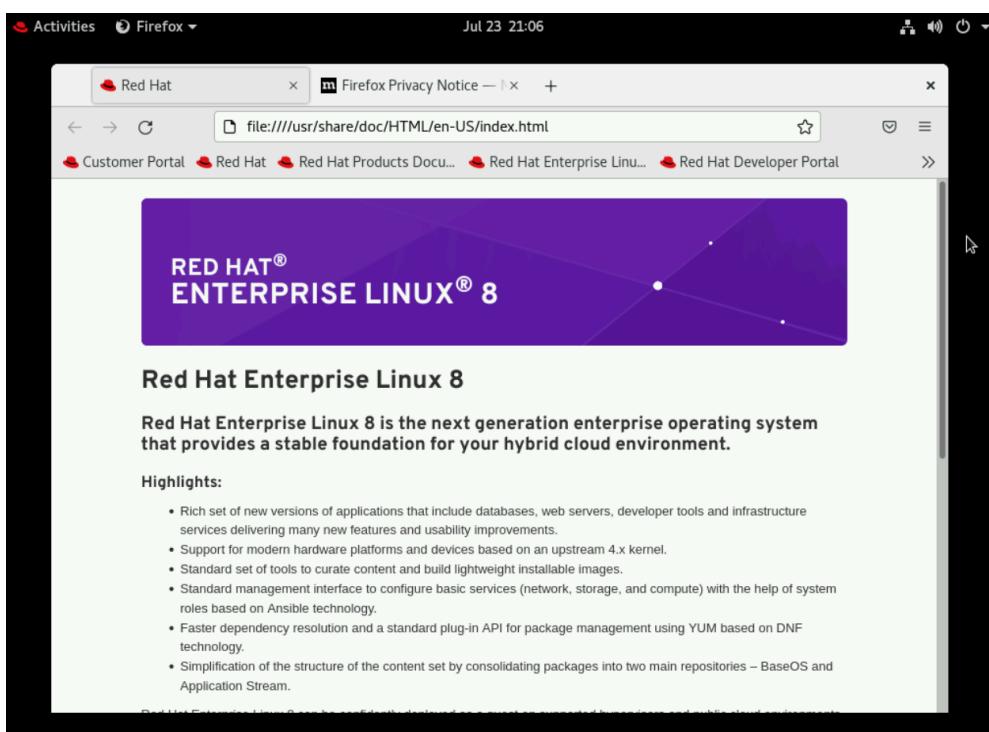
2. 次にログイン画面が現れます。[Student User] を選択して、パスワードとしてstudentを入力して [Sign in] ボタンを押してください。



3. Red Hat Enterprise Linux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が現れます。

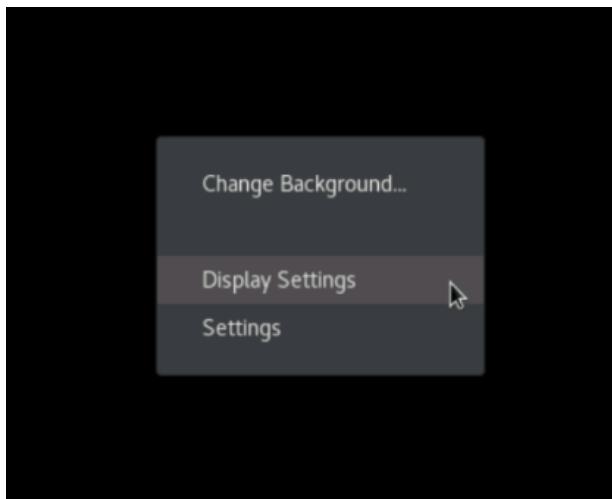


4. アイコンの一番上のFireFoxをクリックして、Webブラウザーを起動してください。このWebブラウザーからOpenShiftクラスターにアクセスします。

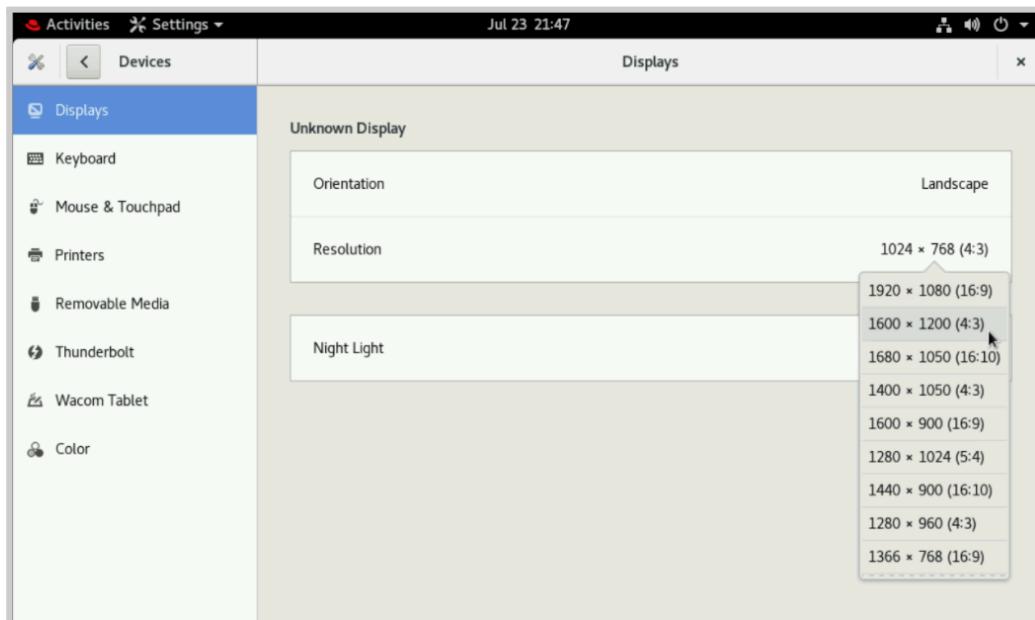


0-2-4. ディスプレイ解像度の変更

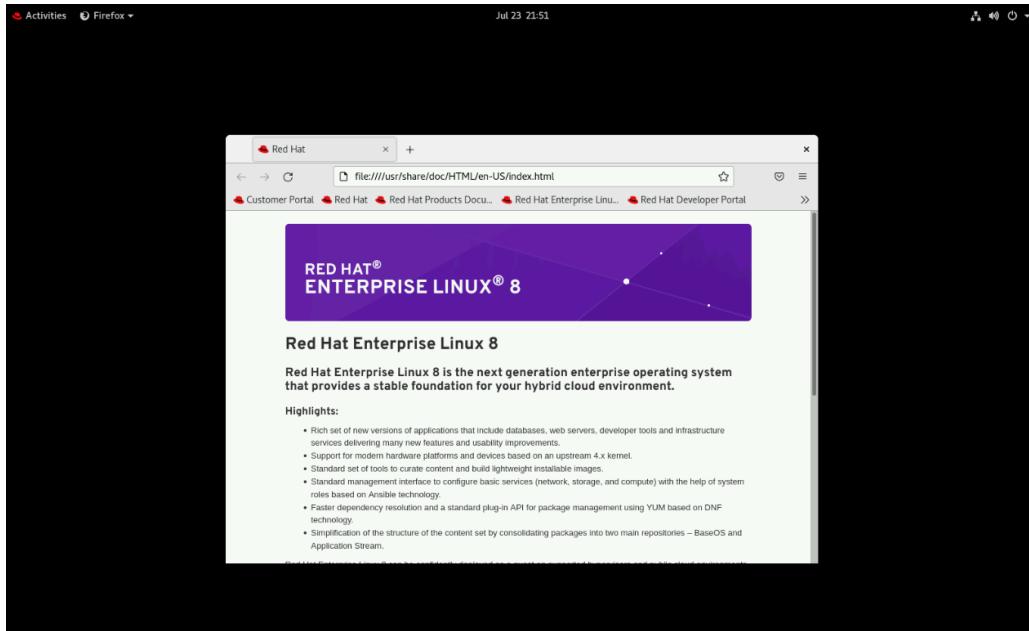
- 初期状態ではWorkstationの画面が狭いので、RHELのディスプレイ設定で解像度を高めに設定し直します。バックグラウンドで右ボタンメニューを表示して、[Display Settings] を選択します。



- [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。

例: 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。team2の場合は、lab02/lab02のように番号だけが異なります。team3、team4も同様です。

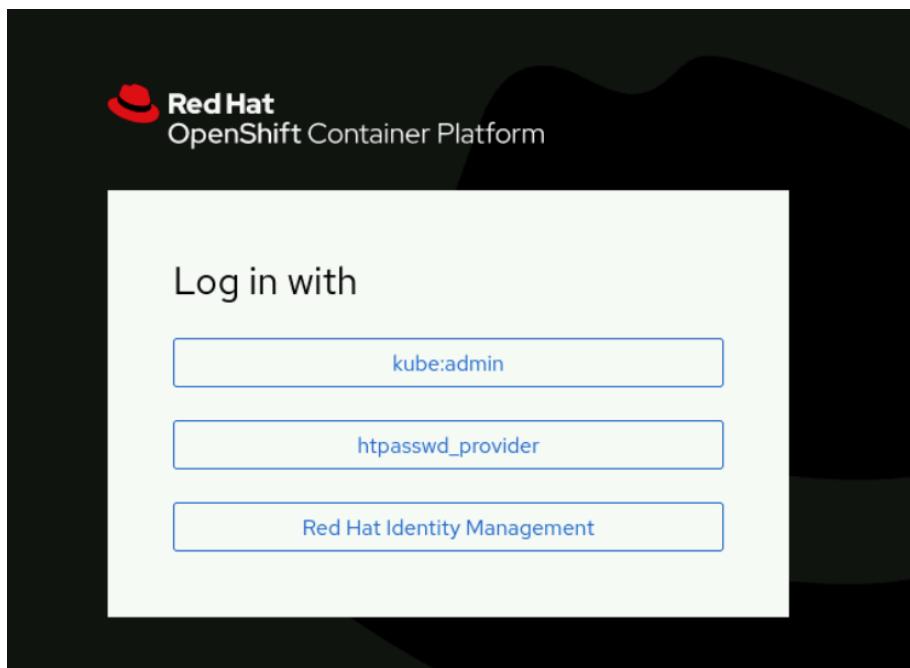
チーム名	team1
------	-------

ユーザー名	lab01
パスワード	lab01

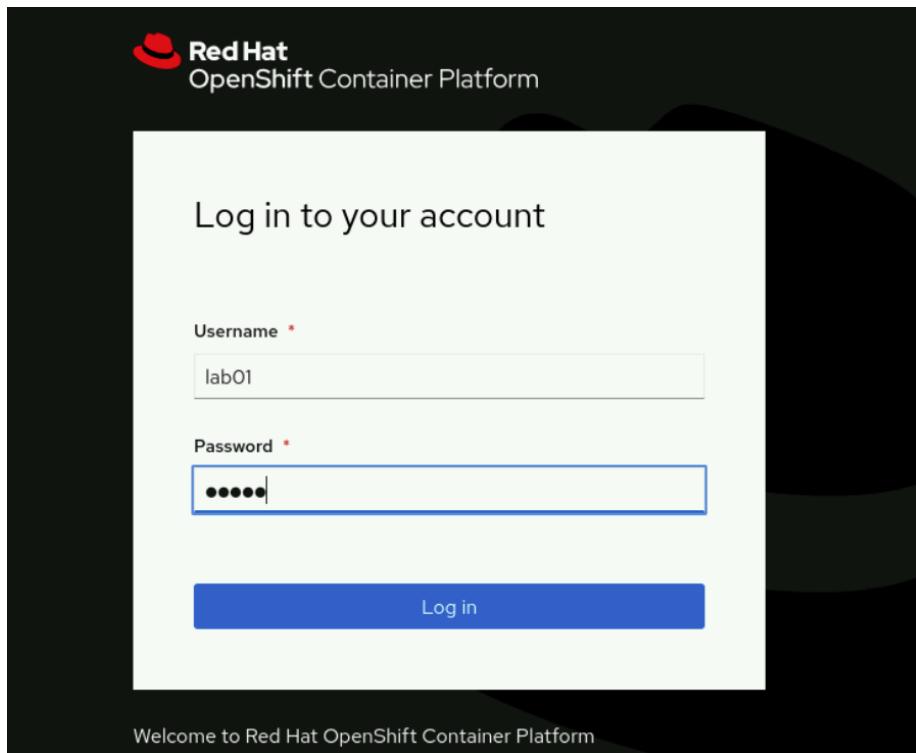
0-3-2. OpenShift Webコンソール

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。

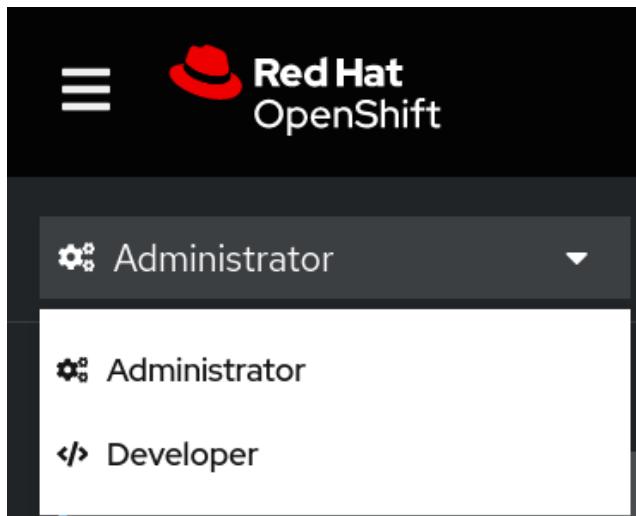
1. Webブラウザーで <https://console-openshift-console.apps.ocp4.example.com> を開くと、OpenShift Webコンソールのログイン画面が表示されます。Log in withの下の [Red Hat Identity Management] を選択します。



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。



The screenshot shows the Red Hat OpenShift Container Platform interface. The top navigation bar includes the Red Hat logo, the platform name, and a user dropdown. The left sidebar has a 'Projects' section selected, along with other options like Home, Search, API Explorer, Events, Operators, Workloads, Networking, Storage, and Builds. The main content area is titled 'Projects' and displays a table of existing projects. The columns are Name, Display name, Status, Requester, and Created. The table lists four projects: stackrox, tl500-gitlab, tl500-shared, and tl500-workspaces. All projects are active and were created on November 3, 2023, at 12:32 PM.

0-3-3. CodeReady Workspaces

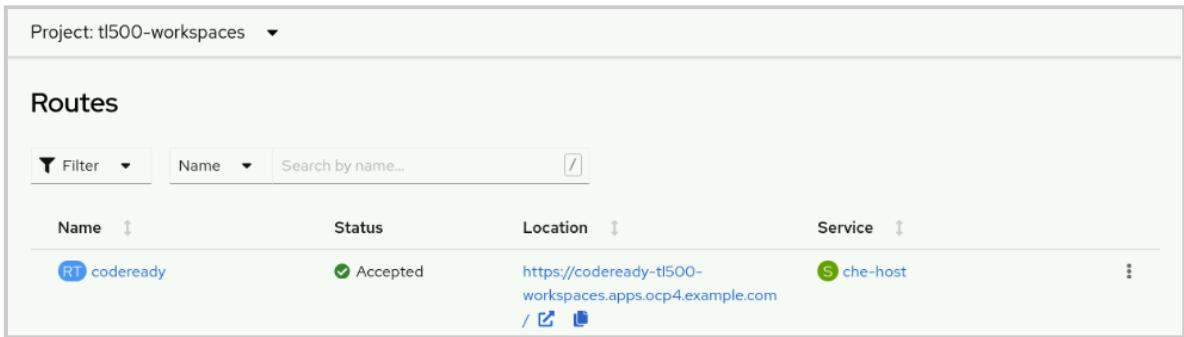
CodeReady Workspacesは、OpenShift上で動作するアプリケーション開発環境です。本技術演習では、アプリケーションのソースコードの閲覧や編集、コマンド操作をする環境としてCodeReady Workspacesを使います。

1. 左側のメニューから [Home] - [Project] を選択し、プロジェクトリストから、tl500-workspacesプロジェクトを開きます。画面をスクロールすると [Inventory] が現れます。Inventoryに表示されている数は、そのときの演習の状況によって異なりますので、下図とまったく同一にはなりません。

The screenshot shows the Red Hat OpenShift Container Platform interface. The left sidebar has a 'Projects' section selected. The main content area is titled 'Inventory' and displays various cluster metrics. The data shown is:

- 7 Deployments
- 0 DeploymentConfigs
- 0 StatefulSets
- 7 Pods
- 1 PersistentVolumeClaim
- 6 Services
- 5 Routes
- 10 ConfigMaps
- 0 VolumeSnapshots

2. [Inventory] から [Routes] のリンクを開き、tl500-workspacesプロジェクト内のRouteリソースを表示します。Routeリソースはアプリケーションの画面にアクセスするURL情報(Location)を提供します。



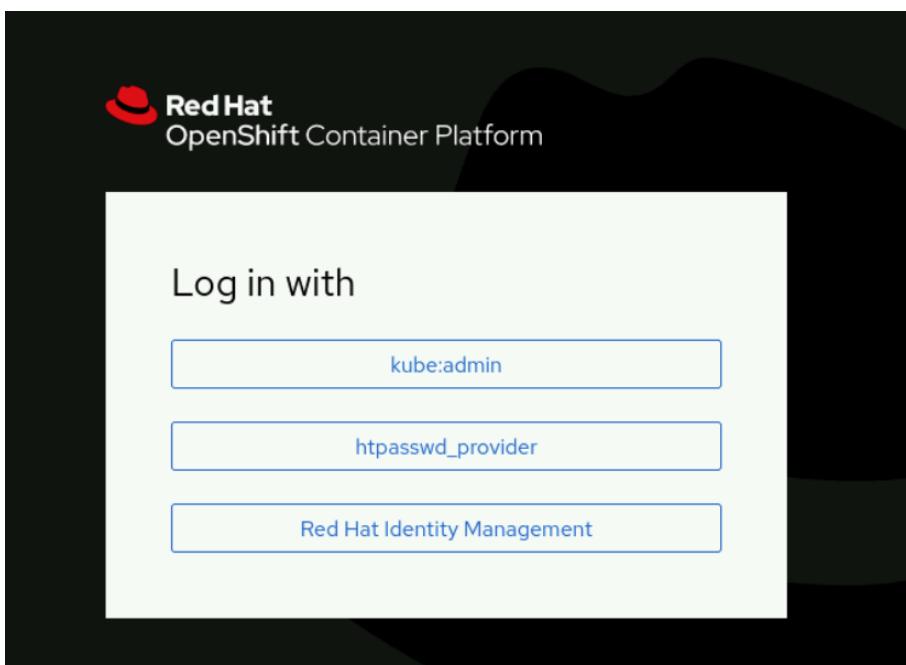
The screenshot shows the 'Routes' list page within the OpenShift web interface. At the top left, it says 'Project: tl500-workspaces'. Below that is a search bar with 'Name' and a 'Search by name...' input field. A filter dropdown is also present. The main table has columns: Name, Status, Location, and Service. One row is visible:

Name	Status	Location	Service
RT codeready	Accepted	https://codeready-tl500-workspaces.apps.ocp4.example.com	 che-host /  

TIPS

左側のメニューにおいて [Networking] - [Routes] を選択し、プロジェクトとして tl500-workspaces を選択しても同じRouteの画面を表示することができます。

3. codereadyルートの[Location]のリンクを開くと、CodeReady Workspaceのログイン画面が開きます。[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。





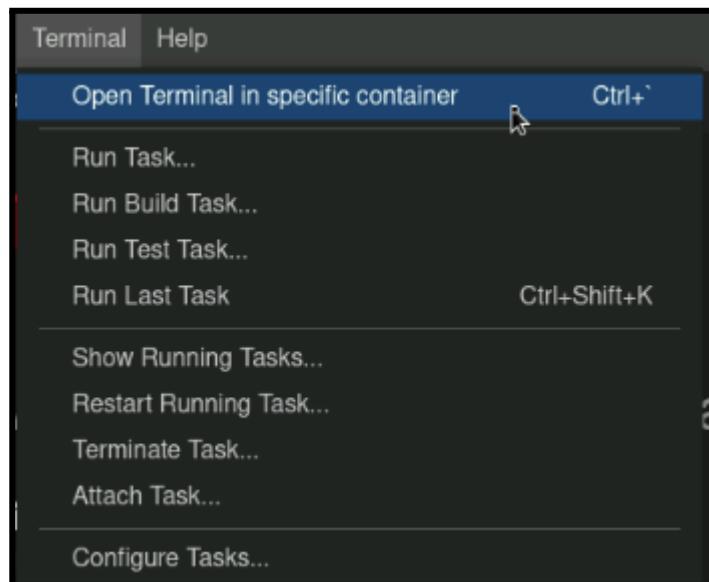
4. CodeReady Workspaceを開きます。[RECENT WORKSPACES]に表示されているワークスペースの[Open]リンクを開きます。

The screenshot shows the Red Hat CodeReady Workspaces interface. On the left, there's a sidebar with 'Create Workspace', 'Workspaces (1)', and 'RECENT WORKSPACES' sections. Under 'RECENT WORKSPACES', there's a single entry: 't1500-ec4g'. The main area is titled 'Workspaces' and contains a brief description: 'A workspace is where your projects live and run. Create workspaces from stacks that define projects, runtimes, and commands.' Below this is a search bar and a table listing workspaces. The table has columns for 'Name', 'Last Modified', and 'Project(s)'. One workspace, 't1500-ec4g', is listed with a timestamp of '21 minutes ago' and the project 'tech-exercise'. There are buttons for 'Delete' and '+ Add Workspace'.

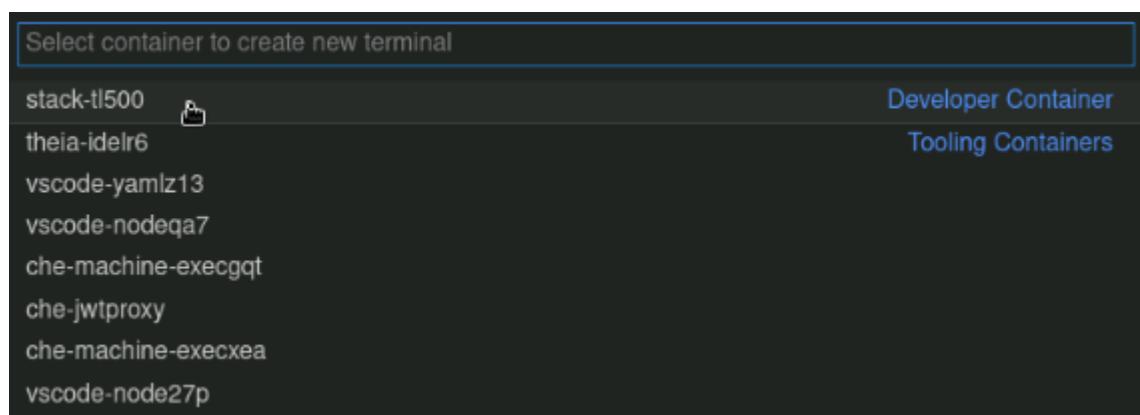
5. [Red Hat CodeReady Workspaces] の表示が見えれば、これでアプリケーションの設定やソースコードを変更できる環境の準備は完了です。これはVS Code for the Webというアプリケーションの開発環境です。

The screenshot shows the VS Code for the Web interface. The top navigation bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', 'Terminal', and 'Help'. The left sidebar is labeled 'EXPLORER' and shows a tree view of a workspace named 'CHE (WORKSPACE)'. The tree includes folders like '.github', 'docs', 'images', 'config', 'facilitation', 'slides', 'pet-battle', 'quick-starts', 'tekton', 'templates', 'tests', and 'TIMELINE'. The main area is titled 'Welcome To Your Workspace' and contains sections for 'New' (with 'New File...' and 'Git Clone...'), 'Open' (with 'Open Files...', 'Open Command Palette...', and keyboard shortcuts), 'Settings' (with 'Open Preferences' and 'Open Keyboard Shortcuts'), and 'Help' (with 'Discover CodeReady Workspaces' and 'Browse Documentation'). A status bar at the bottom shows 'main' and 'Preview' tabs, memory usage 'Mem: 0.59/8.59 GB 6%', and CPU usage 'CPU: 11 m'. A message 'Workspace: t1500-ec4g' is also visible.

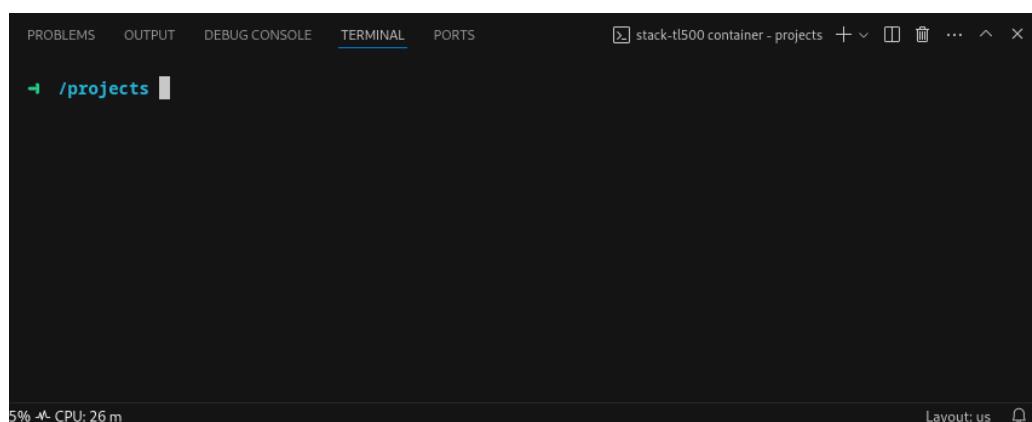
6. 次に左上のメニューから [Terminal] - [Open Terminal in specific container] を選択して、VSCode内でターミナルを開きます。ターミナルとは、Linuxコマンドを操作するアプリケーションです。



コンテナ選択メニューから stack-tl500 を選択します。



7. ターミナルは画面右下に表示されます。cd /projects コマンドを実行してカレントディレクトリを /projects に変更します。/projects という表示は、プロンプトと呼ばれるもので、ここで Linux コマンドを入力することができます。





TIPS

この`/projects`という表示は現在のディレクトリ(カレントディレクトリ)の場所を示しています。ディレクトリを変更すると、プロンプトの表示も変わります。`pwd`というコマンドを実行することで、現在のディレクトリの名前を知ることができます。プロンプトにカレントディレクトリを表示しているので、自分の作業している場所がすぐにわかります。

8. ターミナル内でLinuxコマンドを叩いてみましょう。ls コマンドはカレントディレクトリ内に存在するファイルやディレクトリを表示します。以下の表示から、カレントディレクトリである /projects の下に3つのディレクトリがあることがわかります(ディレクトリは青いファイルとして表示されています)。



The screenshot shows a terminal window within a code editor interface. The top bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined, indicating it's active), and PORTS. To the right of the tabs is a search bar containing "stack-tl500 container - projects" and a set of icons for navigating between tabs, closing, and more. The main area of the terminal shows the following command history and output:

```
→ /projects pwd
/projects
→ /projects ls
pet-battle-api  tech-exercise  tl500-setup
→ /projects ls -l
total 12
drwxrwxr-x.  6 developer 1000800000 4096 Jul 10 05:22 pet-battle-api
drwxrwxr-x. 11 developer 1000800000 4096 Jul 10 05:09 tech-exercise
drwxrwxr-x.  4 developer 1000800000 4096 Jul 10 04:54 tl500-setup
→ /projects |
```

9. 続けてターミナル内で次のコマンドを叩いてみましょう。cd コマンドでtech-exerciseの下にカレントディレクトリを移動すると、プロンプトがtech-exerciseと表示されていることがわかります。また、lsコマンドを実行するとtech-exerciseディレクトリに含まれるファイルやディレクトリが表示されます。

The screenshot shows a terminal window with the following content:

```
→ /projects cd tech-exercise
→ tech-exercise git:(main) ls
CODE_OF_CONDUCT.md    LICENSE           docs          tekton      ubiquitous-journey
CONTRIBUTING.md       README.md        pet-battle    templates   values.yaml
Chart.yaml            argocd-values.yaml quick-starts  tests
→ tech-exercise git:(main) █
```

1. コンテナー化されたアプリケーション (20分)

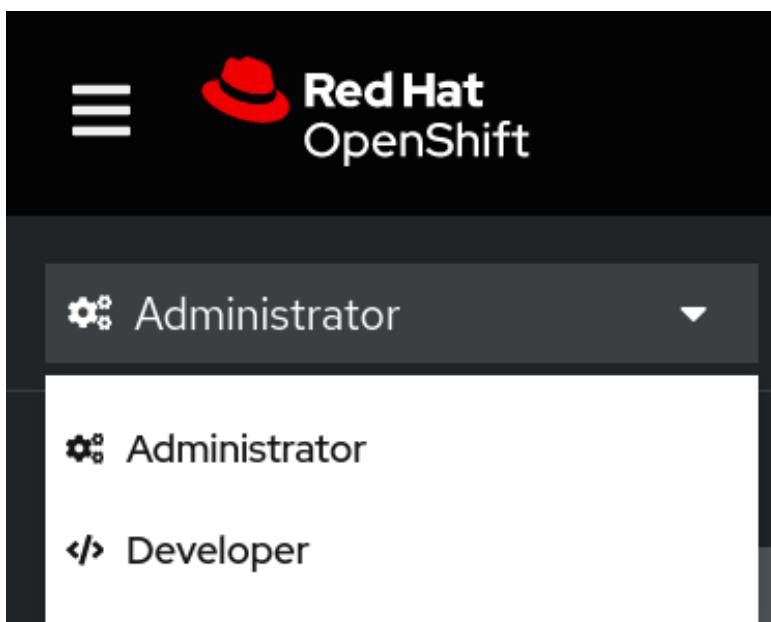
目的	<ul style="list-style-type: none">PetBattleアプリケーションの概要を理解する
目標	<ul style="list-style-type: none">OpenShift 上にインストールされたPetBattleを確認するPetBattleのUIを操作し機能概要を理解するPetBattleの動作を支える関連アプリケーションを調べる

1-1. PetBattleアプリケーションの動作確認

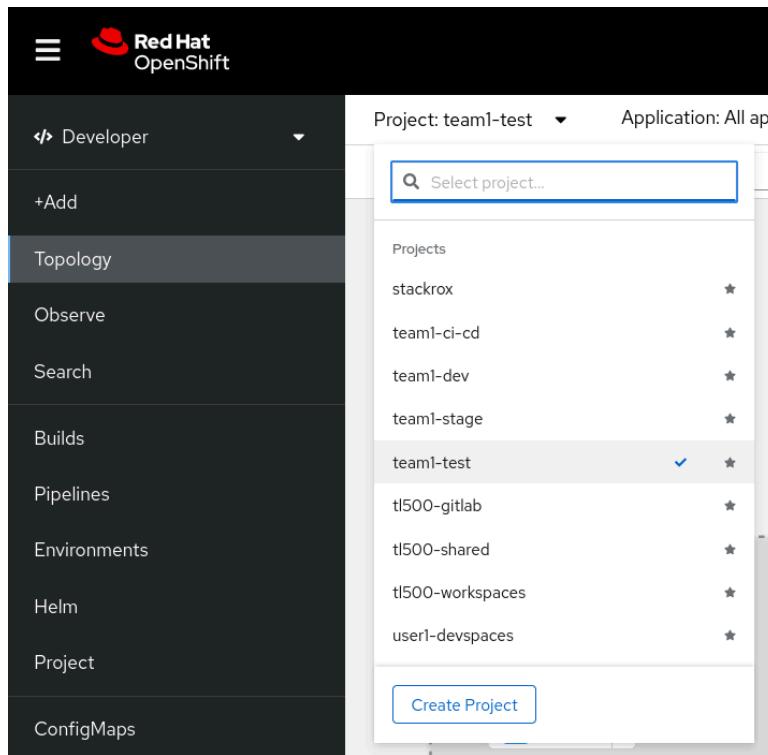
OpenShift上にインストールされたPetBattleのアプリケーションGUI (Graphical User Interface) を表示します。

1-1-1. PetBattleアプリケーションへのアクセス

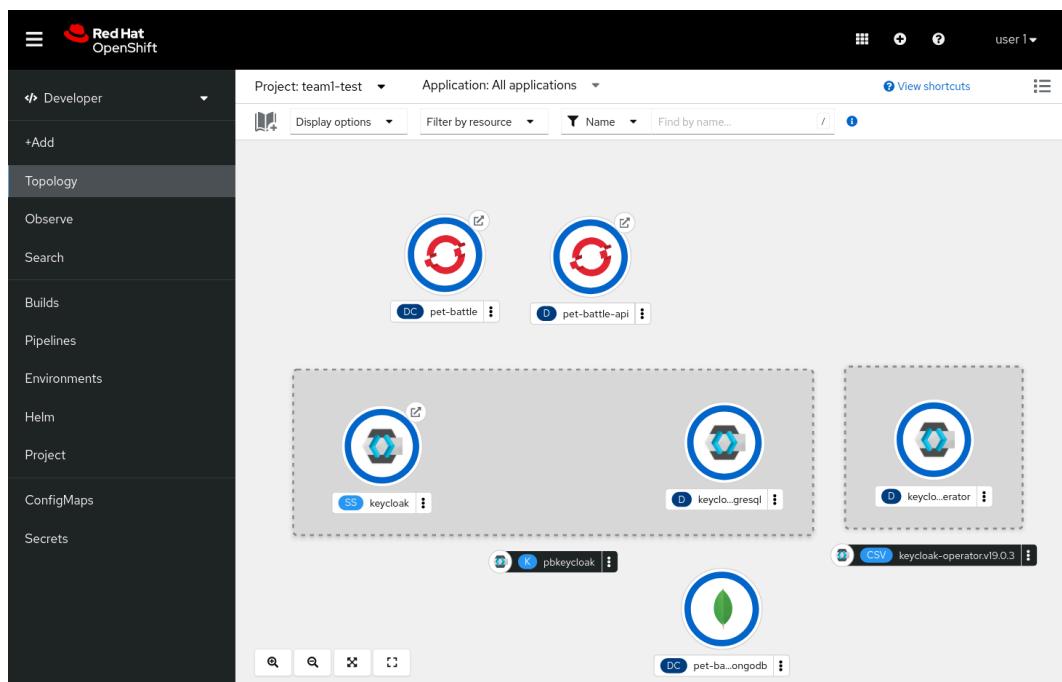
- OpenShift Webコンソールにログインし、左上のRed Hat OpenShiftロゴの下のメニューで、開発者向け表示である、[Developer]を選択してください。



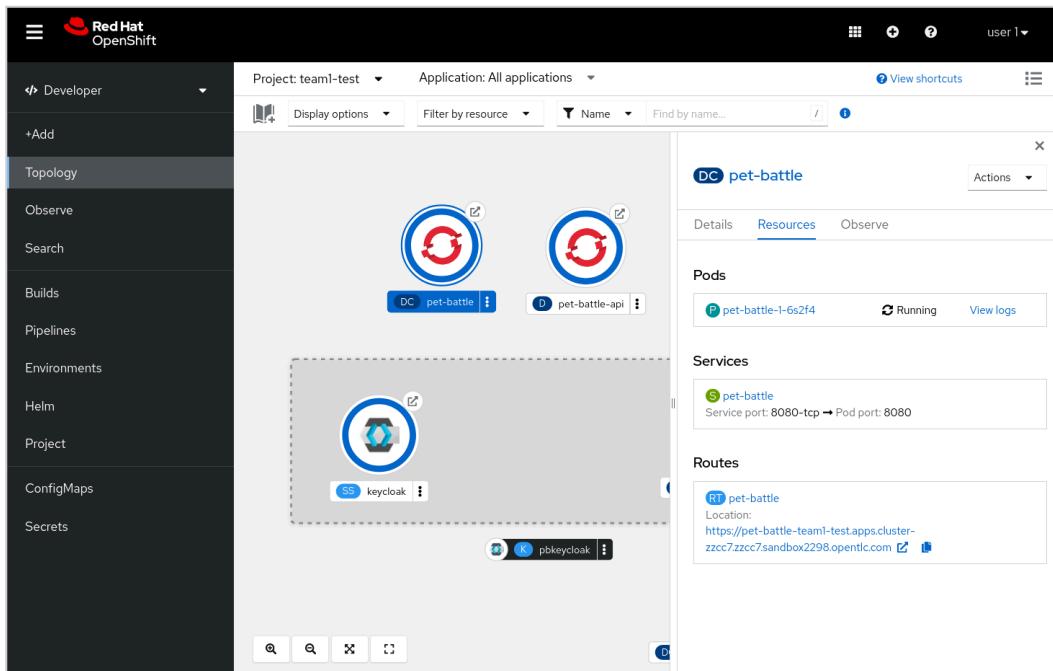
- 左側のメニューから [Topology] を選択し、右側上部のプロジェクト選択メニューから、<チーム名>-testプロジェクトを選択します。以下の例では、チーム名がteam1の場合で、プロジェクト名はteam1-testプロジェクトになります。



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。



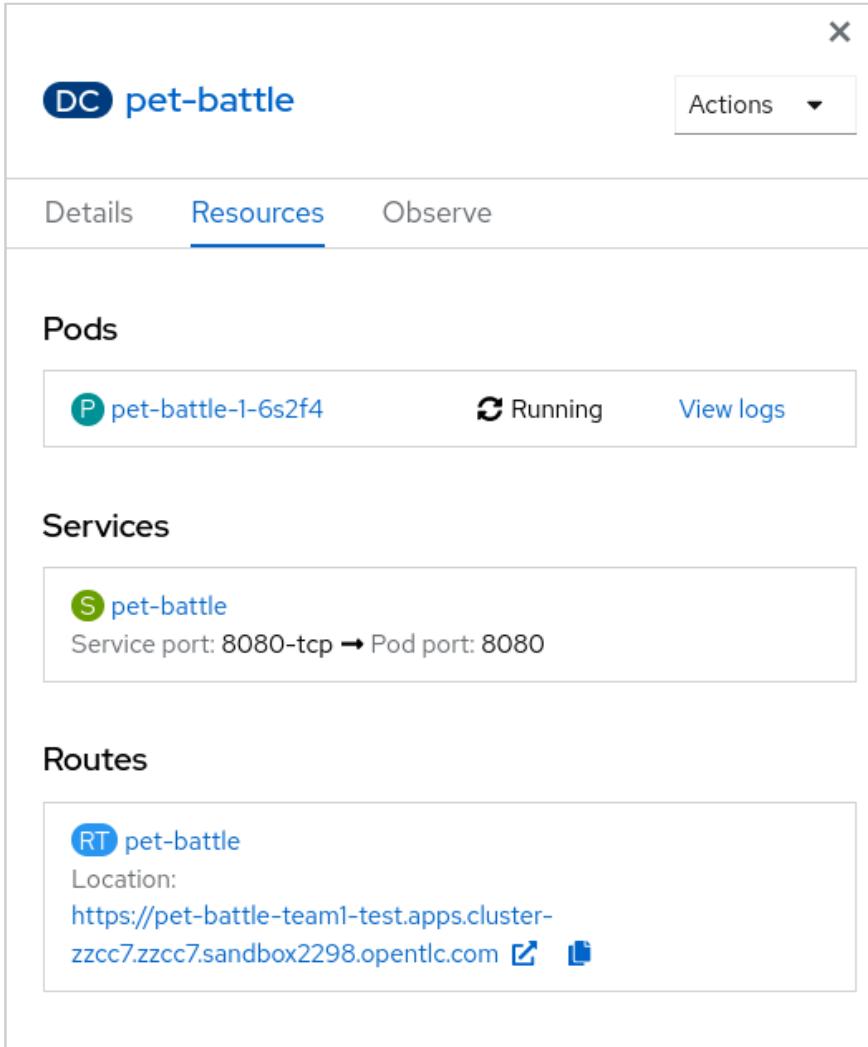
4. pet-battleアプリケーションをクリックすると、右側にPods、Services、Routesといったアプリケーションに関連するリソース情報が表示されます。



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。

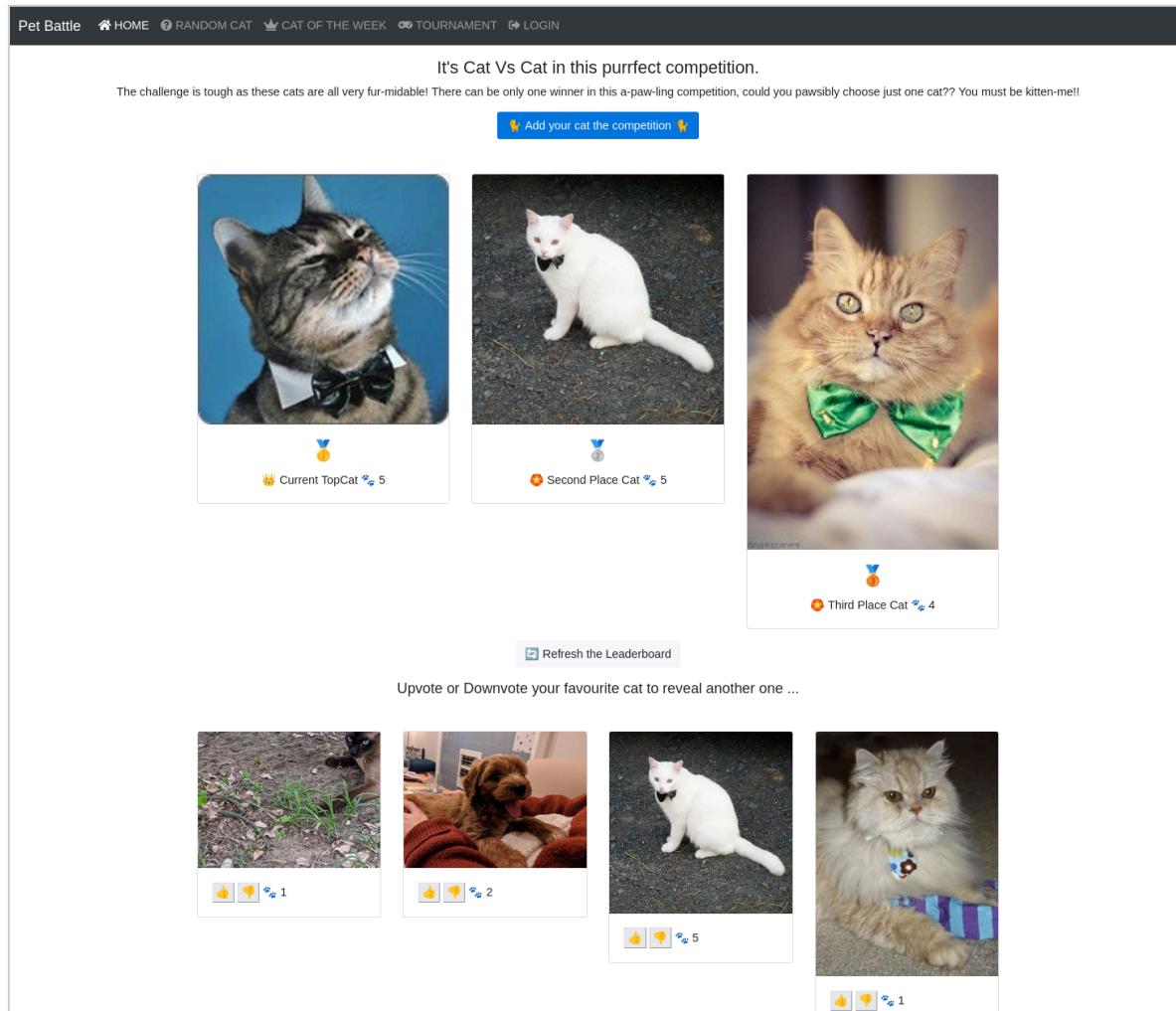
5. 右側に表示された [Routes] の [Location] リンクを選択します。



The screenshot shows the Red Hat OpenShift web console interface for the 'pet-battle' application. At the top, there's a header with a 'DC' icon and the text 'pet-battle'. To the right is an 'Actions' dropdown menu. Below the header, there are three tabs: 'Details', 'Resources' (which is currently 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 green status icon, indicating it is healthy. It maps the service port 8080-tcp to the pod port 8080. In the 'Routes' section, a route named 'pet-battle' is listed with the location <https://pet-battle-team1-test.apps.cluster-zzcc7.zzzcc7.sandbox2298.opentlc.com>. There are also icons for a refresh button and a clipboard.

1-1-2. PetBattle アプリケーションのGUI画面

1. 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ます。



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。

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。

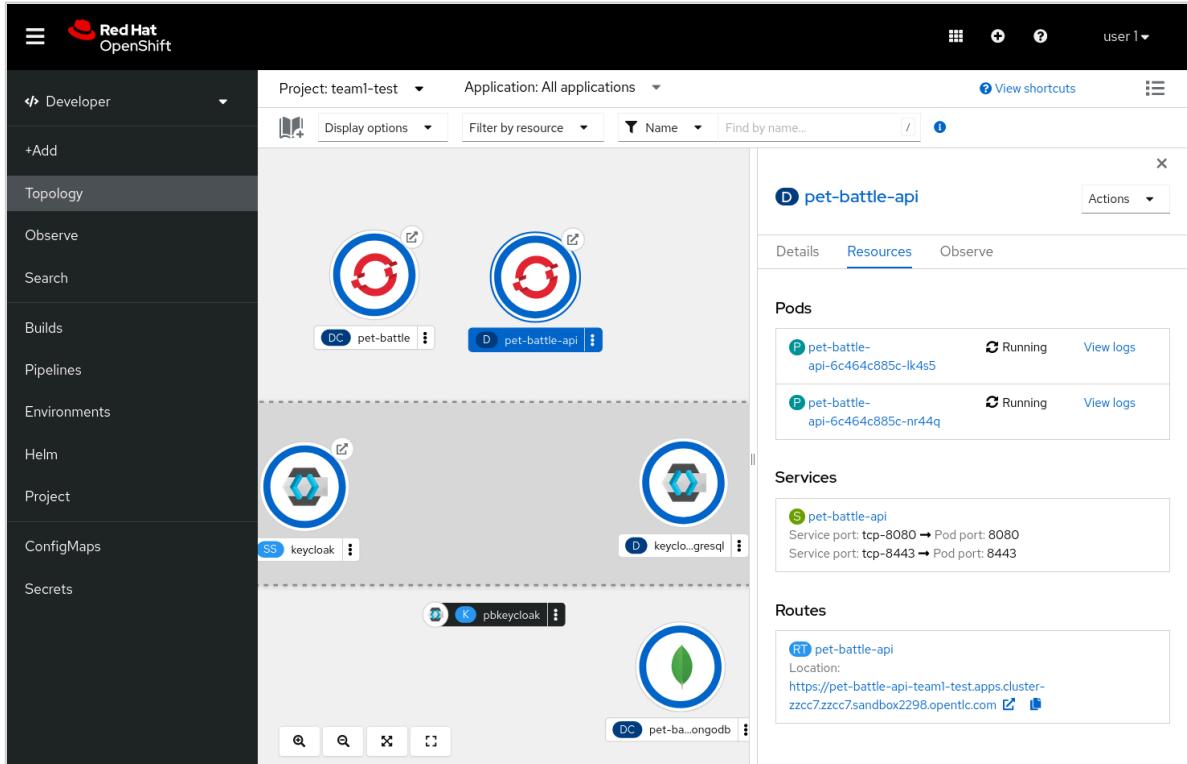
- PetBattleアプリケーション
PetBattleのGUIを提供する
- PetBattle API
PetBattleのAPI (Application Programming Interface) を提供する
- MongoDBデータベース
猫や投票などのデータを保存する

- Keycloak

ユーザー認証を行う

1-2-1. PetBattle APIへのアクセス

1. 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。



The screenshot shows the Red Hat OpenShift Web Console interface. The left sidebar has a 'Topology' tab selected. The main area displays a network diagram with several nodes: 'pet-battle' (DC), 'pet-battle-api' (D), 'keycloak' (SS), 'pbkeycloak' (D), and 'pet-ba...ongodb' (DC). The 'pet-battle-api' node is highlighted with a blue border. To the right of the diagram, detailed information for the 'pet-battle-api' application is shown under the 'Resources' tab. It includes:

- Pods:** pet-battle-api-6c464c885c-lk4s5 (Running), pet-battle-api-6c464c885c-nr44q (Running)
- Services:** pet-battle-api (Service port: tcp-8080 → Pod port: 8080, Service port: tcp-8443 → Pod port: 8443)
- Routes:** pet-battle-api (Location: https://pet-battle-api-team1-test.apps.cluster-zzcc7zzcc7.sandbox2298.opentlc.com)

2. 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.

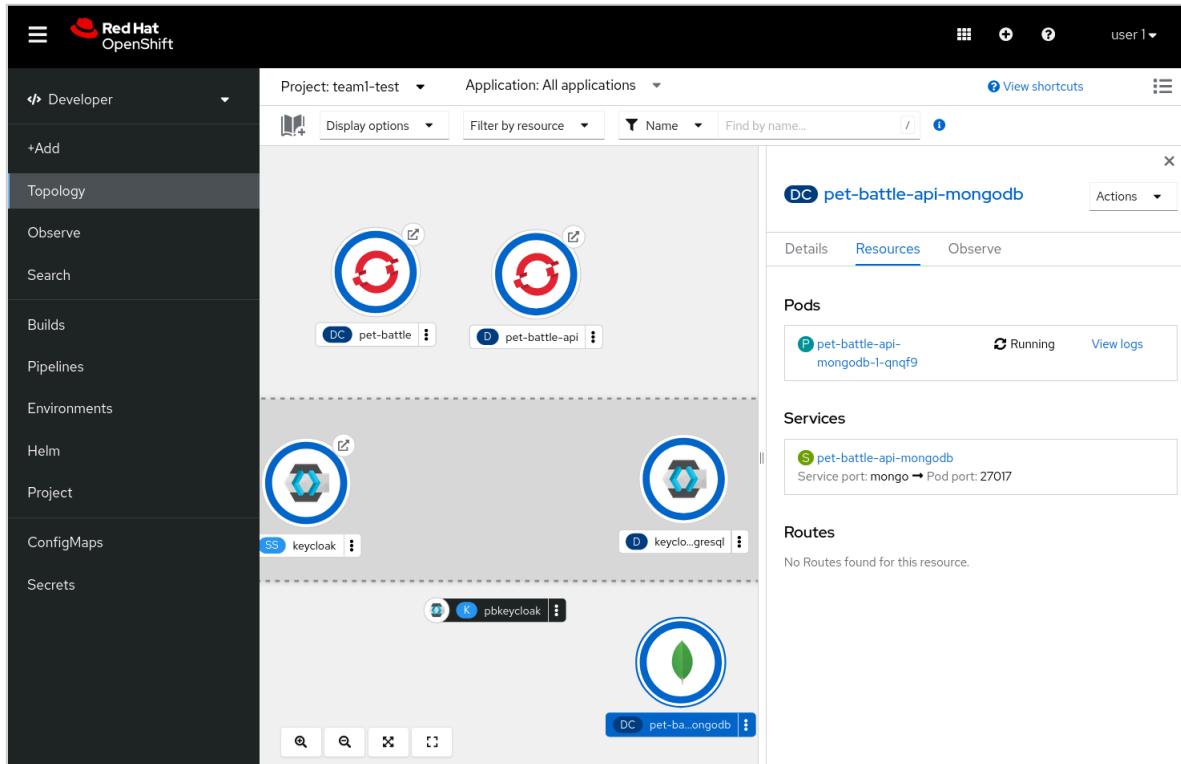
id	count	issff	image
668e18ba36c4605ec8c9f394	3	true	
668e18bb36c4605ec8c9f395	3	true	
668e18bc36c4605ec8c9f396	1	true	
668e18bc36c4605ec8c9f397	1	true	
668e18bc36c4605ec8c9f398	3	true	
668e18bc36c4605ec8c9f399	6	true	
668e18bd36c4605ec8c9f39a	5	true	
668e18bd36c4605ec8c9f39b	2	true	
668e18be36c4605ec8c9f39c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

Showing 1 to 10 of 13 entries

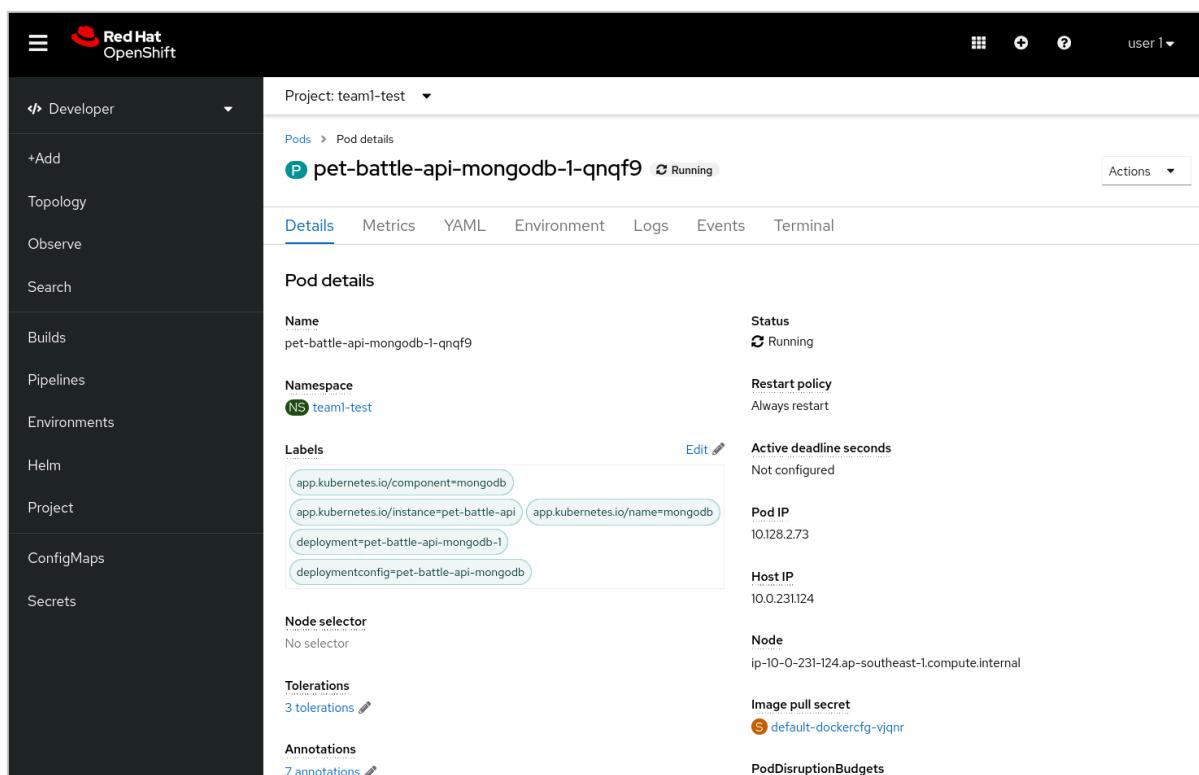
Previous 1 2 Next

1-2-2. MongoDBへのアクセス

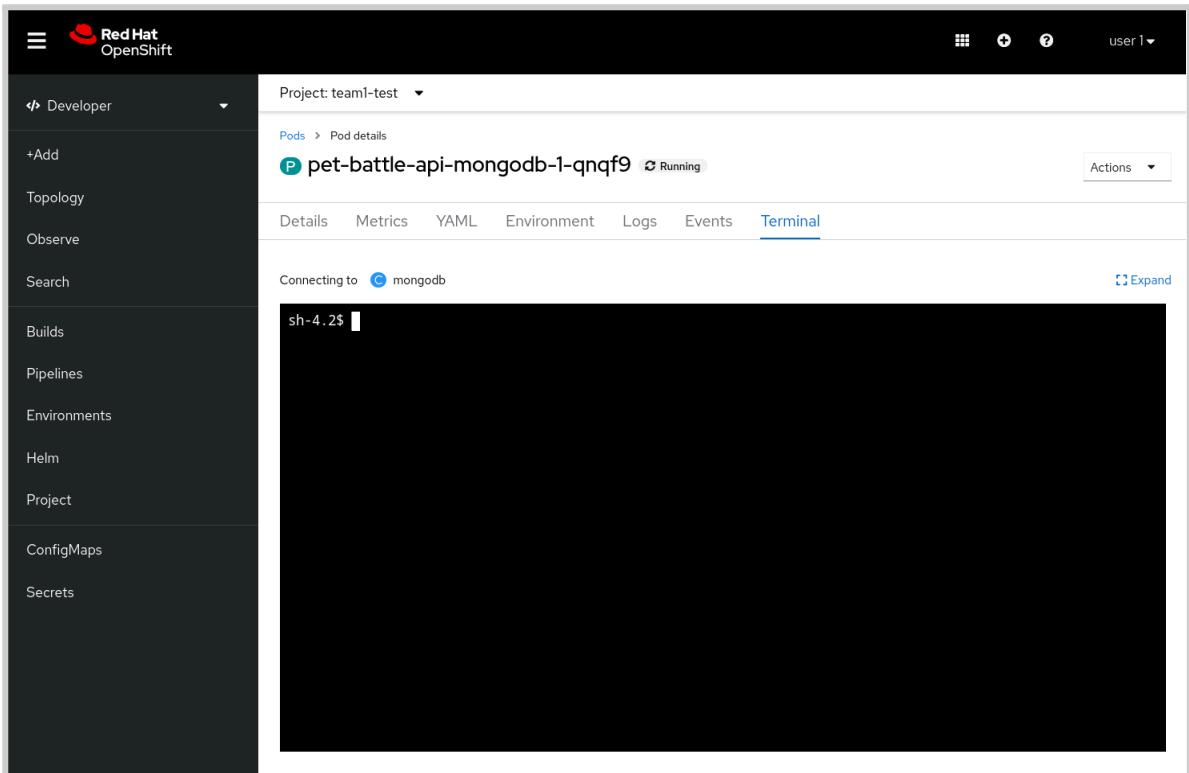
3. OpenShift Webコンソールの [Topology] に戻り、pet-battle-api-mongodbアプリケーションをクリックします。



4. 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)。



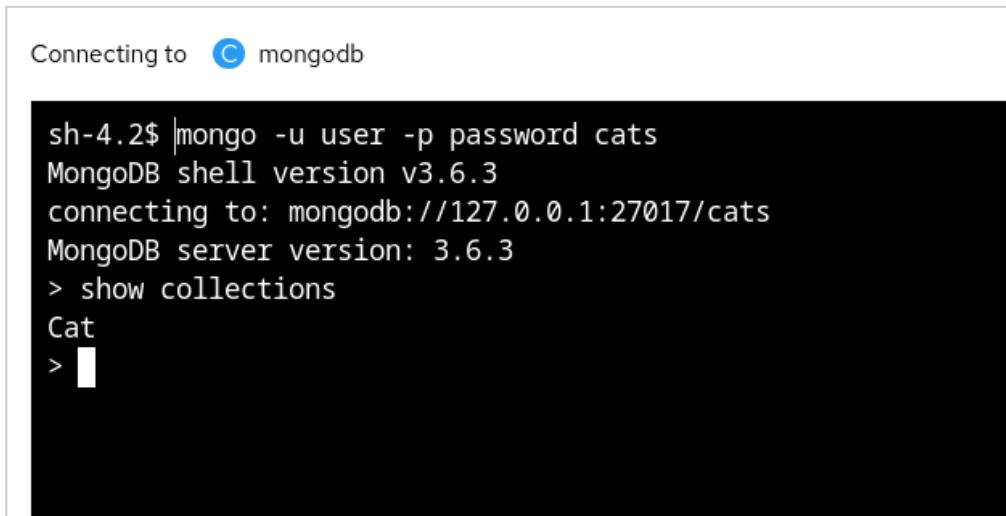
5. Pod詳細画面の[Terminal]タブをクリックすると、MongoDBのコンテナー内でターミナルを開くことができます。このターミナルから、MongoDBのCLIコマンドを実行することができます。



The screenshot shows the Red Hat OpenShift web interface. On the left, there's a sidebar with various developer tools like Developer, Topology, Observe, Search, Builds, Pipelines, Environments, Helm, Project, ConfigMaps, and Secrets. The main area is titled "Project: team1-test". It shows a "Pods" section with one pod named "pet-battle-api-mongodb-1-qnqf9" which is "Running". Below the pod list are tabs for Details, Metrics, YAML, Environment, Logs, Events, and Terminal. The Terminal tab is currently selected. A message "Connecting to mongodb" is displayed above a terminal window. The terminal window itself is mostly blank, showing only the prompt "sh-4.2\$".

6. 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. It starts with the message "Connecting to mongodb". Then it runs the command "mongo -u user -p password cats", which connects to the MongoDB server at 127.0.0.1:27017 and selects the "cats" database. The MongoDB shell version v3.6.3 is shown. The command "show collections" is run, and the output shows a single collection named "Cat".

```
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を使ってユーザー管理、ユーザー認証を行います。

Keycloakのアカウントはteam1-testプロジェクトの credential-pbkeycloakシークレット内にある。

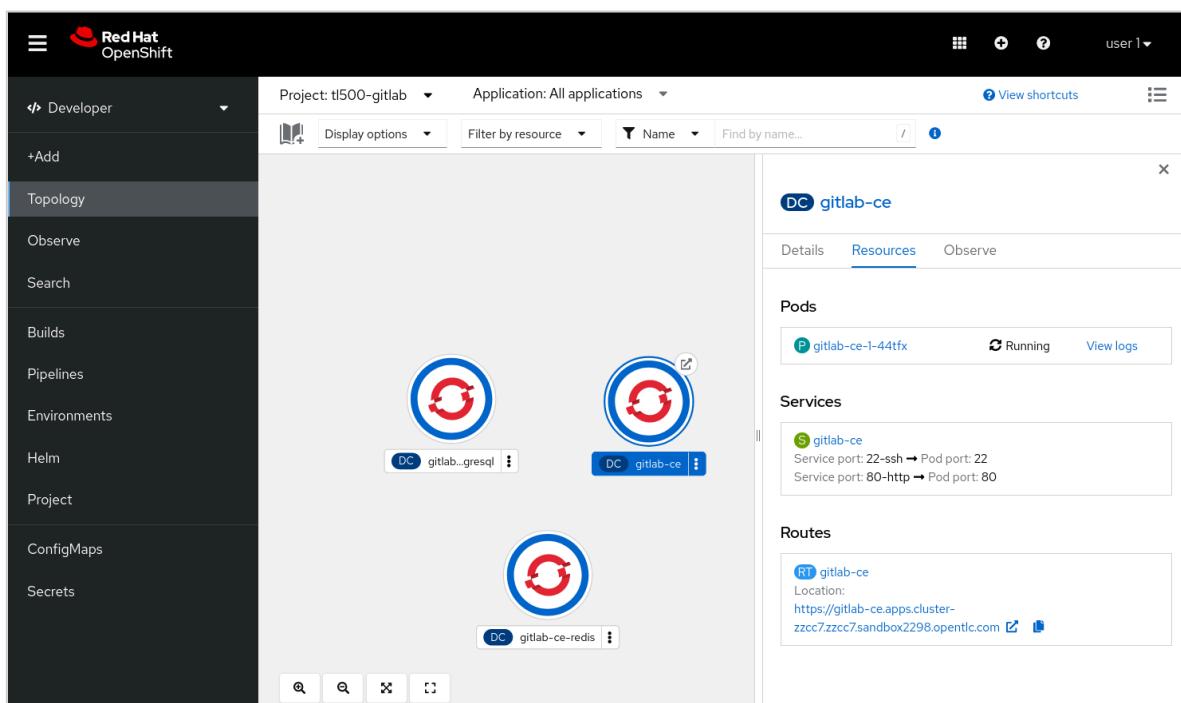
2. Everything As Code (60分)

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 ローカル環境でPetBattle APIのソースをビルドする PetBattle APIをWebブラウザー上で実行する PetBattle APIをOpen APIを使って操作する

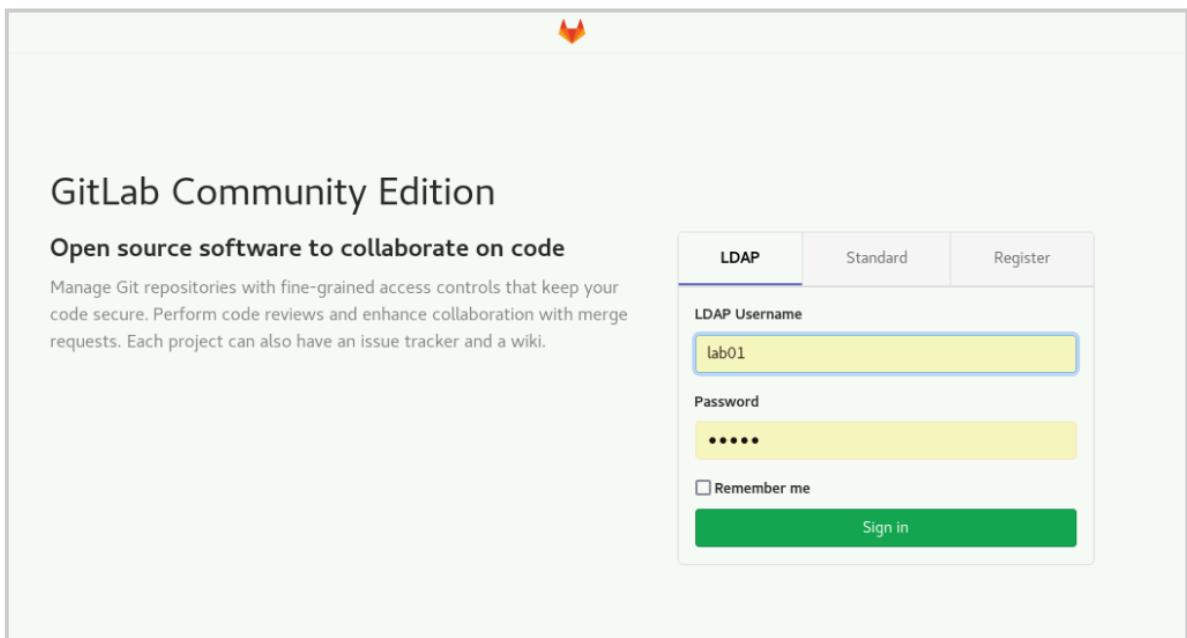
2-1. Gitコマンドの基礎 (20分)

2-1-1. GitLabアプリケーション

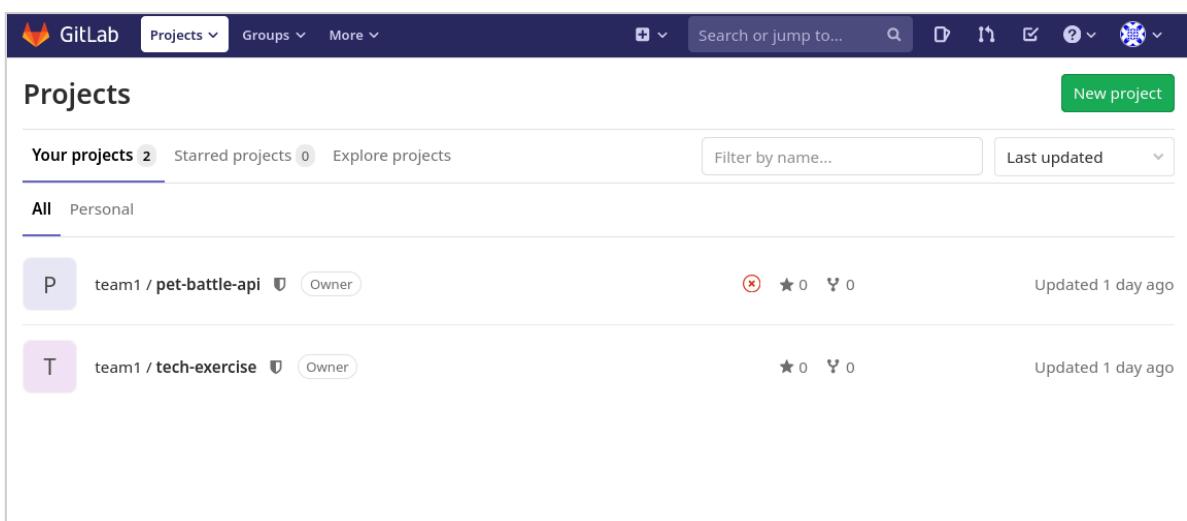
- OpenShift WebコンソールのDeveloperパースペクティブにおいて [Topology]を選択します。上部のプロジェクト選択メニューで tl500-gitlabプロジェクトを選択し、gitlab-ceアプリケーションをクリックします。



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。

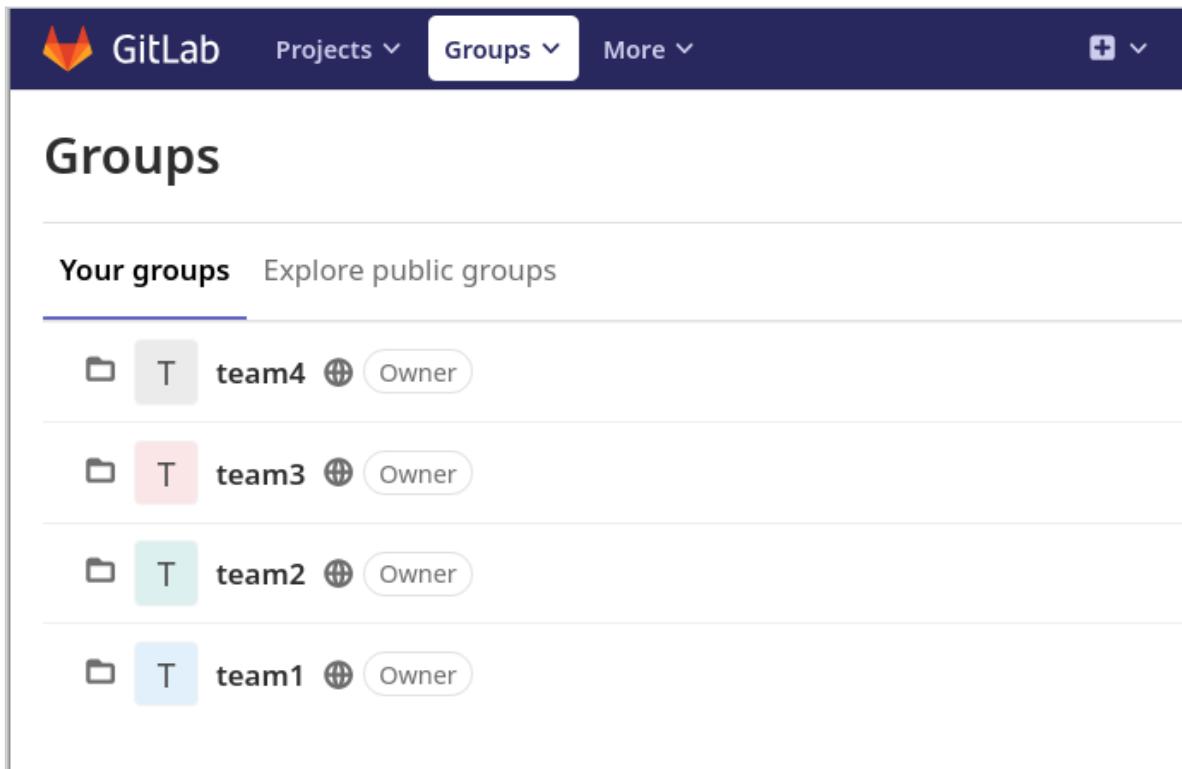


3. サインイン後にGitLabプロジェクト画面が表示されます。GitLabにはすでに2つのプロジェクトが作られた状態になっています。以下の図ではteam1グループの下にpet-battle-apiプロジェクトとtech-exerciseプロジェクトが登録されています。



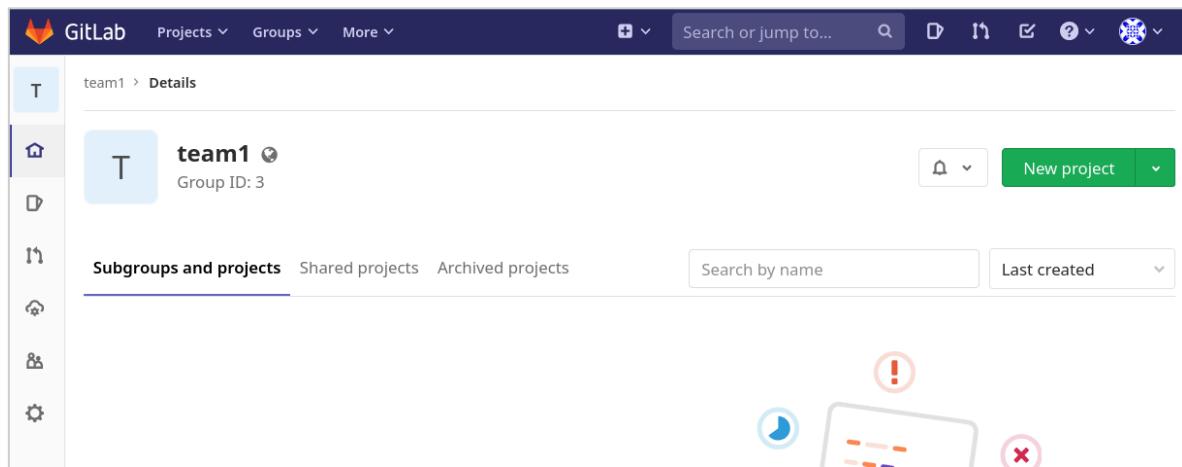
2-1-2. Gitリポジトリの作成

1. Gitリモートリポジトリを作成します。GitLab上に新規にgit-exerciseプロジェクトを作成します。まず、GitLabのUI画面上で、上部の [Groups] メニューから、[Your projects] を選択してGitLabグループのリストを表示します。



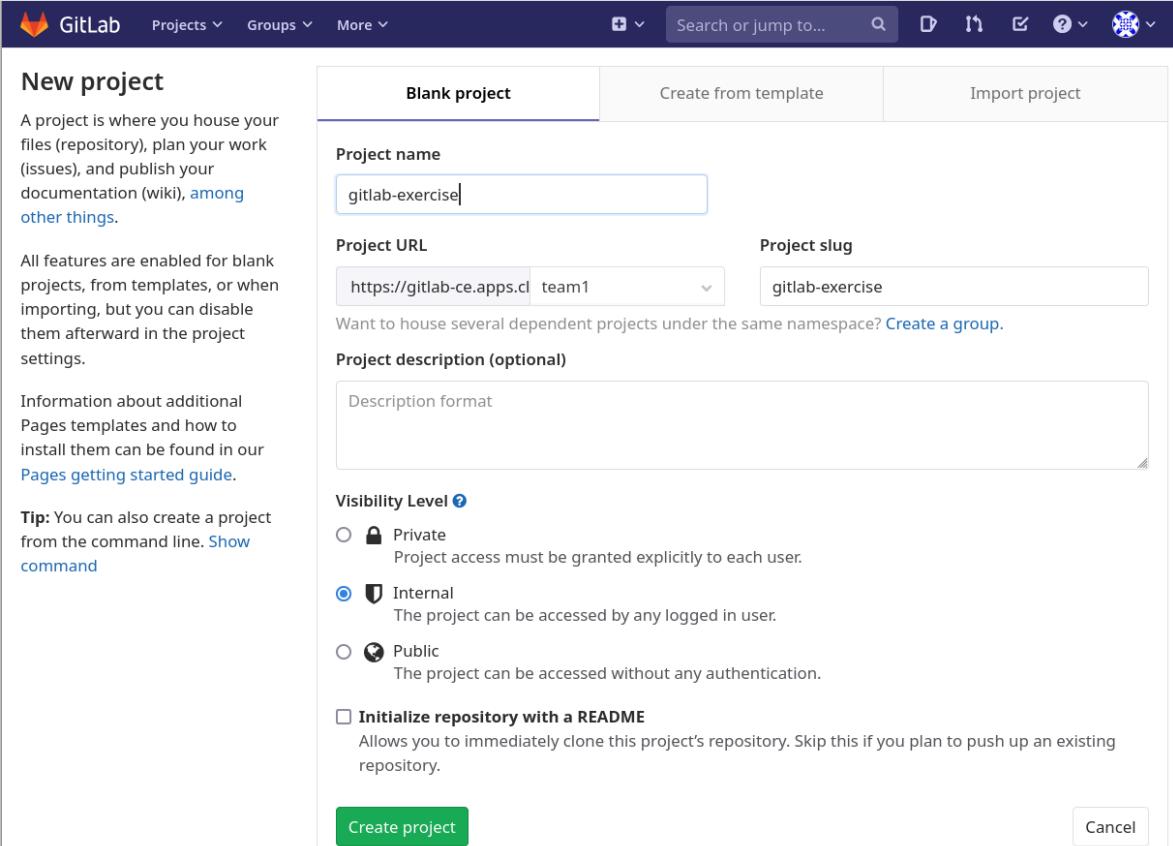
The screenshot shows the 'Groups' page in the GitLab UI. At the top, there are navigation links: 'Projects', 'Groups' (which is currently selected), and 'More'. A '+' button is also visible. Below the header, the title 'Groups' is displayed. Under the heading 'Your groups', there is a link to 'Explore public groups'. The main content area lists four groups: 'team4' (Owner), 'team3' (Owner), 'team2' (Owner), and 'team1' (Owner). Each group entry includes a folder icon, a team name, a person icon, and an 'Owner' badge.

自分のチームに割り当てられたチーム名のグループを選択し、緑色の [New project] ボタンを押します。



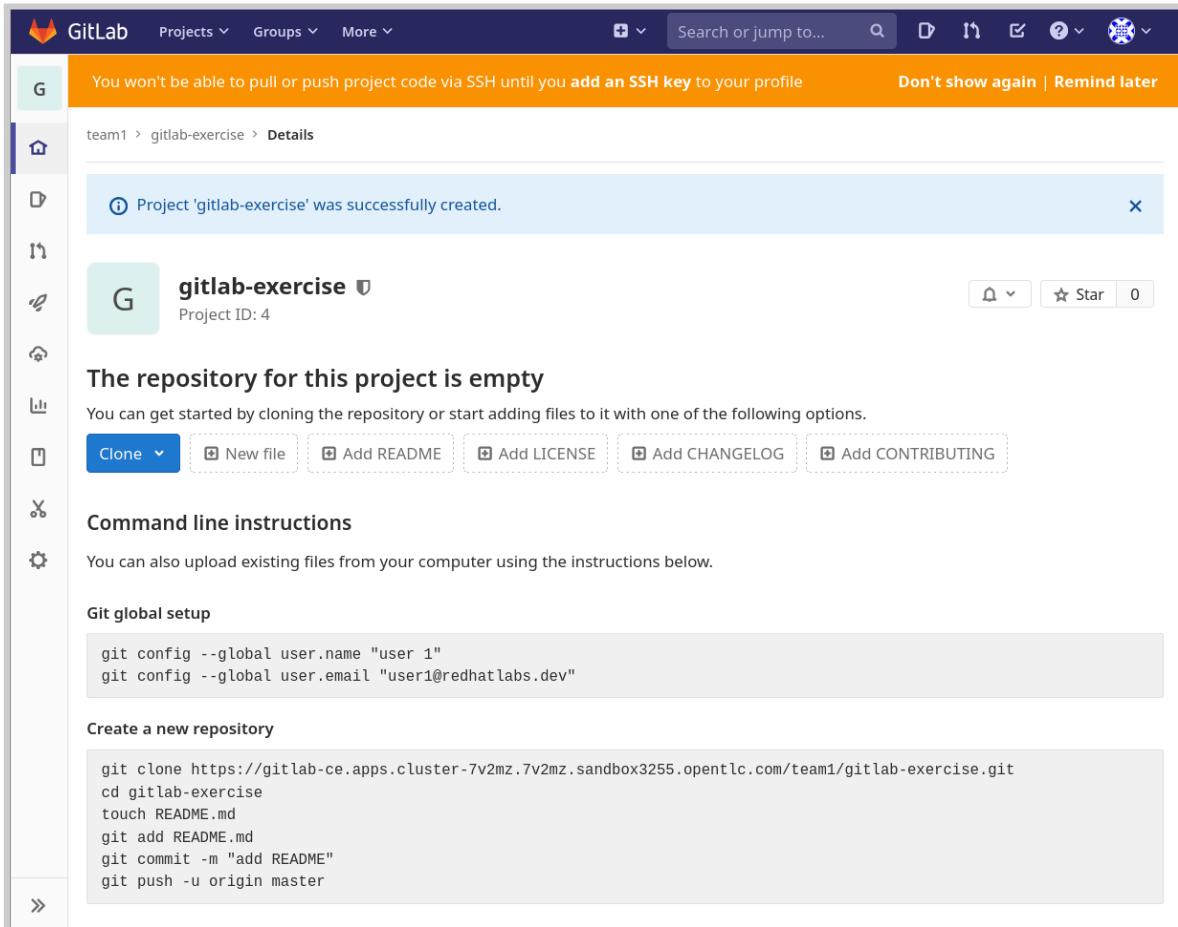
The screenshot shows the 'team1' details page in the GitLab UI. The top navigation bar includes 'Projects', 'Groups', 'More', and a search bar. On the right side of the header, there is a green 'New project' button. The main content area displays the 'team1' group details, including its name, Group ID (3), and a 'Details' link. Below this, there are sections for 'Subgroups and projects', 'Shared projects', and 'Archived projects'. A search bar and a 'Last created' dropdown are also present. In the bottom right corner, there are three small icons: a blue circle with a white exclamation mark, a red rectangle with a white minus sign, and a red circle with a white cross.

[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 GitLab 'New project' creation interface. On the left, there's a sidebar with information about projects, groups, and documentation. The main area has tabs for 'Blank project', 'Create from template', and 'Import project'. The 'Blank project' tab is selected. It contains fields for 'Project name' (gitlab-exercise), 'Project URL' (https://gitlab-ce.apps.cl team1), 'Project slug' (gitlab-exercise), and a 'Project description (optional)' field with 'Description format'. Below these are visibility options: 'Private' (radio button), 'Internal' (radio button, selected), 'Public' (radio button), and a checkbox for 'Initialize repository with a README' which is unchecked. 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

[Don't show again](#) | [Remind later](#)

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](#)

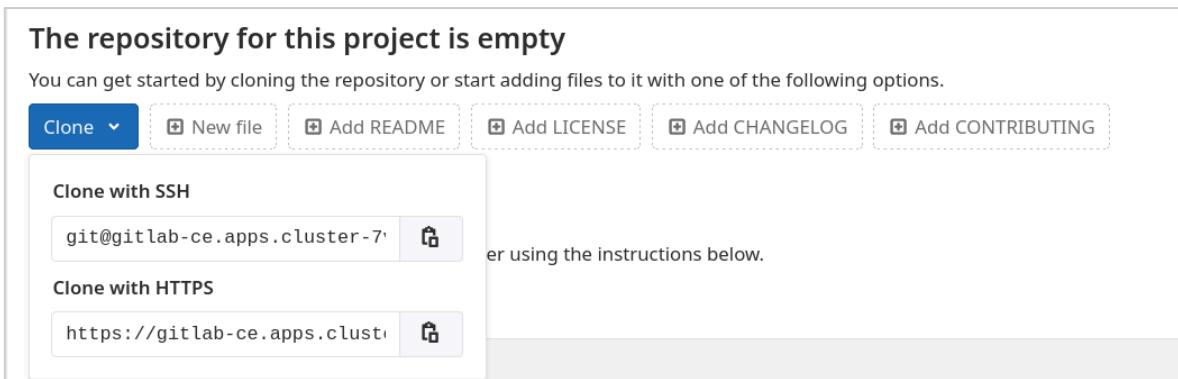
Command line instructions

```
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
```

2. 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。

```
$ git clone  
https://gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com/team1/g  
itlab-exercise.git  
Cloning into 'gitlab-exercise'...  
Username for  
'https://gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com': user1  
Password for  
'https://user1@gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com':  
warning: You appear to have cloned an empty repository.  
<略>total 0
```

2-1-3. 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。

- 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。下記の"user1"の部分は、実際のユーザー名に置き換えてください。

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

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

```
$ git config --global --list  
user.email=user1@redhatlabs.dev  
user.name=user 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。

```
$ cd gitlab-exercise/  
$ git status  
On branch master  
  
No commits yet
```

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

3. 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。

```
$ gedit README.md
$ cat README.md
# README
$ 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)
```

4. ステージング環境に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。

```
$ git add README.md
$ git status
On branch master

No commits yet

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

5. 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。

```
$ gedit hello.py
$ cat hello.py
print("Hello world!")
$ 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環境に登録します。ステージング環境に README.mdとhello.pyの2つのファイルが登録されました。

```
$ git add hello.py
$ 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
```

6. ステージング環境に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。

```
$ 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
```

- 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。

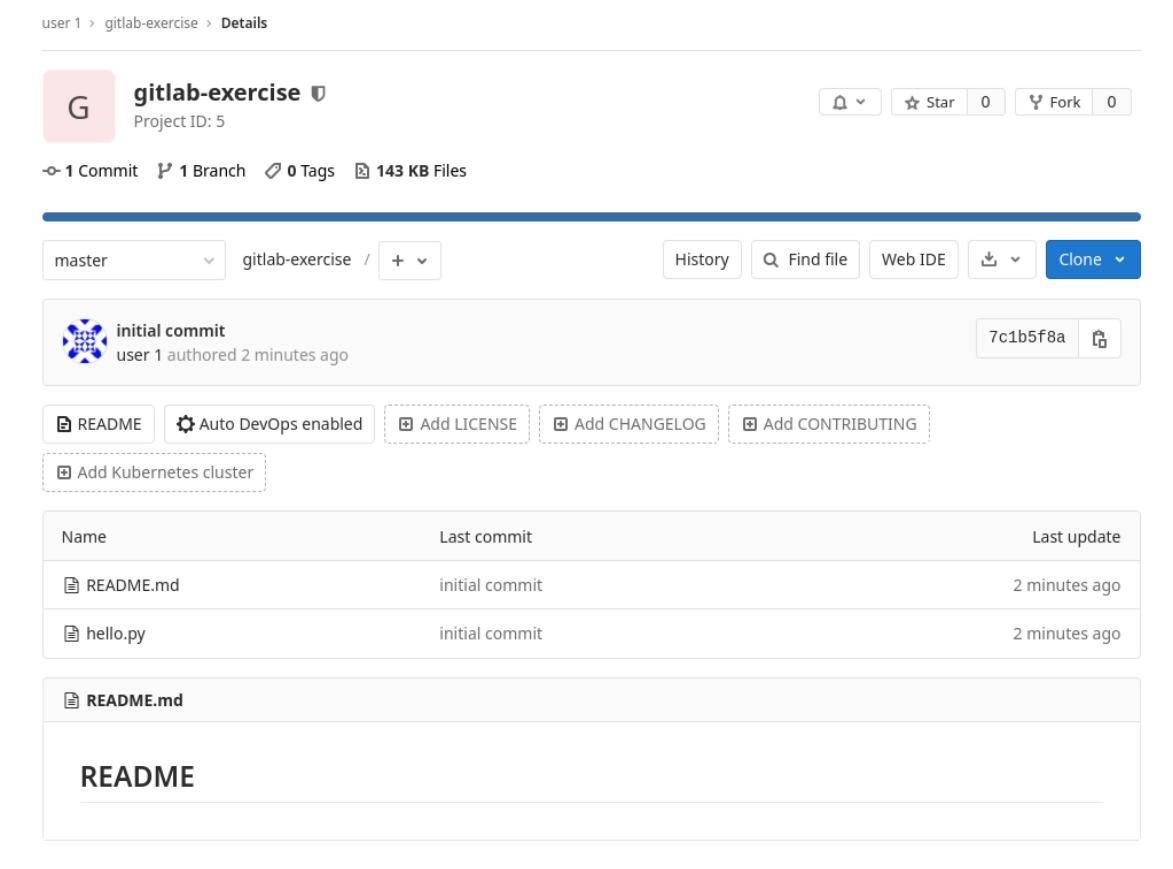
```
$ git log
commit 7c1b5f8ae5c3255c3a29931e8edbb9a31a976160 (HEAD -> master)
Author: user 1 <user1@redhatlabs.dev>
Date: Thu Jul 18 15:13:30 2024 +0900

    initial commit
```

- 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。

```
$ git push -u origin master
Username for
'https://gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com': user1
Password for
'https://user1@gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com':
warning: redirecting to
https://gitlab-ce.apps.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com/user1/gi
tlab-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.cluster-7v2mz.7v2mz.sandbox3255.opentlc.com/user1/gi
tlab-exercise
 * [new branch]  master -> master
branch 'master' set up to track 'origin/master'
```

- 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

2-2. 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。

開発者はアプリケーションの開発時には開発者は開発をするための環境を使います。このような環境は通常、以下の3種類のものがあります。

- 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

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。この章の残りは**CodeReady Workspaces**は使わないので注意してください。

2-2-1. PetBattle API 開発環境の設定

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のインストール

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。

```
sudo dnf install git make git java-17-openjdk-devel maven podman  
podman-plugins
```

次に、すでにインストール済みのJava 1.8.0 のパッケージを削除します。

```
sudo dnf erase java-1.8.0-openjdk
```

2. Javaを使うための環境変数設定

Javaを使うためには環境変数JAVA_HOMEの設定が必要です。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。

```
export JAVA_HOME=/etc/alternatives/java_sdk_17_openjdk/
```

2-2-1. 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。



The screenshot shows the GitLab interface for a project named 'pet-battle-api'. At the top, there's a message: 'You won't be able to pull or push project code via SSH until you add an SSH key to your profile'. Below the header, the project details are shown: 'team1 > pet-battle-api > Details'. The project summary includes '506 Commits', '1 Branch', '0 Tags', and '9.6 MB Files'. A navigation bar at the top right includes 'Clone' (SSH and HTTPS options), 'Find file', 'Web IDE', and other project management links. On the left, a sidebar provides navigation links like Home, Projects, Groups, and More. The main content area displays a commit history with one recent update: 'UPDATED - pet-battle-version to 1.3.1' by 'Derek Dinosaur' (1 day ago). Below the commit history is a table showing project files and their last commits:

Name	Last commit	Last update
chart	removing is for api	1 week ago
rhacm	latest image version	4 years ago
src	update quarkus 3.0.2.Final, put metrics back	1 year ago
.dockerignore	quarkus 1.12.0.Final	3 years ago

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 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.
```



2-2-1. 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mvn compile コマンドでJavaソースコードのビルドを実行します。

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] -----< 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] -- compiler:3.11.0:compile (default-compile) @ pet-battle-api --
[INFO] Nothing to compile - all classes are up to date
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:20 min
```



```
[INFO] Finished at: 2024-07-11T16:36:41+09:00  
[INFO] -----  
$
```

2-2-3. 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イメージを作成します。

```
[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 16/18: EXPOSE 8080  
--> e7ad1c64112d  
STEP 17/18: USER 185  
--> e60eea8f38b5  
STEP 18/18: ENTRYPOINT [ "/deployments/run-java.sh" ]  
COMMIT quay.io/petbattle/pet-battle-api:latest  
--> ec455a9b42cb  
Successfully tagged quay.io/petbattle/pet-battle-api:latest  
ec455a9b42cb273de4dde9c4a3bc6101146d48dcc336f7aa6a6034f4b620c0d3  
$
```

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。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専有しますので、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)。

```
[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 mongo:latest
<略>
```

```
{"t":{"$date":"2024-07-11T07:56:25.106+00:00"},"s":"I", "c":"CONTROL", "id":23138,
"ctx":"SignalHandler","msg":"Shutting down","attr":{"exitCode":0}}
```

2-2-4. PetBattle API コンテナーの起動

1. MongoDBが起動していれば、PetBattle APIのUI画面を表示することができます。
pet-battle-api のディレクトリにおいて、mvn quarkus:dev コマンドを実行してください。
これは開発モードでアプリケーションを実行します。

```
[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>
```

```
2024-07-20 06:36:20,253 INFO [org.mon.dri.cluster]
(cluster-ClusterId{value='669b5ae45f037c230ece47a5',
description='null'}-mongo:27017) Monitor thread successfully connected to server with
description ServerDescription{address=mongo:27017, type=STANDALONE,
state=CONNECTED, ok=true, minWireVersion=0, maxWireVersion=21,
maxDocumentSize=16777216, logicalSessionTimeoutMinutes=30,
roundTripTimeNanos=67339347}
```

```
2024-07-20 06:36:24,036 INFO [io.quarkus] (main) pet-battle-api 1.1.1 on JVM
(powered by Quarkus 3.0.2.Final) started in 5.603s. Listening on: http://0.0.0.0:8080
and https://0.0.0.0:8443
```

```
2024-07-20 06:36:24,037 INFO [io.quarkus] (main) Profile dev activated.
```

```
2024-07-20 06:36:24,037 INFO [io.quarkus] (main) Installed features: [cdi, micrometer,
mongodb-client, mongodb-panache, narayana-jta, rest-client-reactive, resteasy-reactive,
resteasy-reactive-jackson, smallrye-context-propagation, smallrye-health,
smallrye-openapi, swagger-ui, vertx]
```

このコマンドは終了せずにターミナルを専有し続けます。Javaソースコードを変更した場合は、ソースコードを自動的にコンパイルして実行に反映します。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)。

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。

2-2-5. PetBattle API OpenAPI UIの表示

1. 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	

2-3. 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

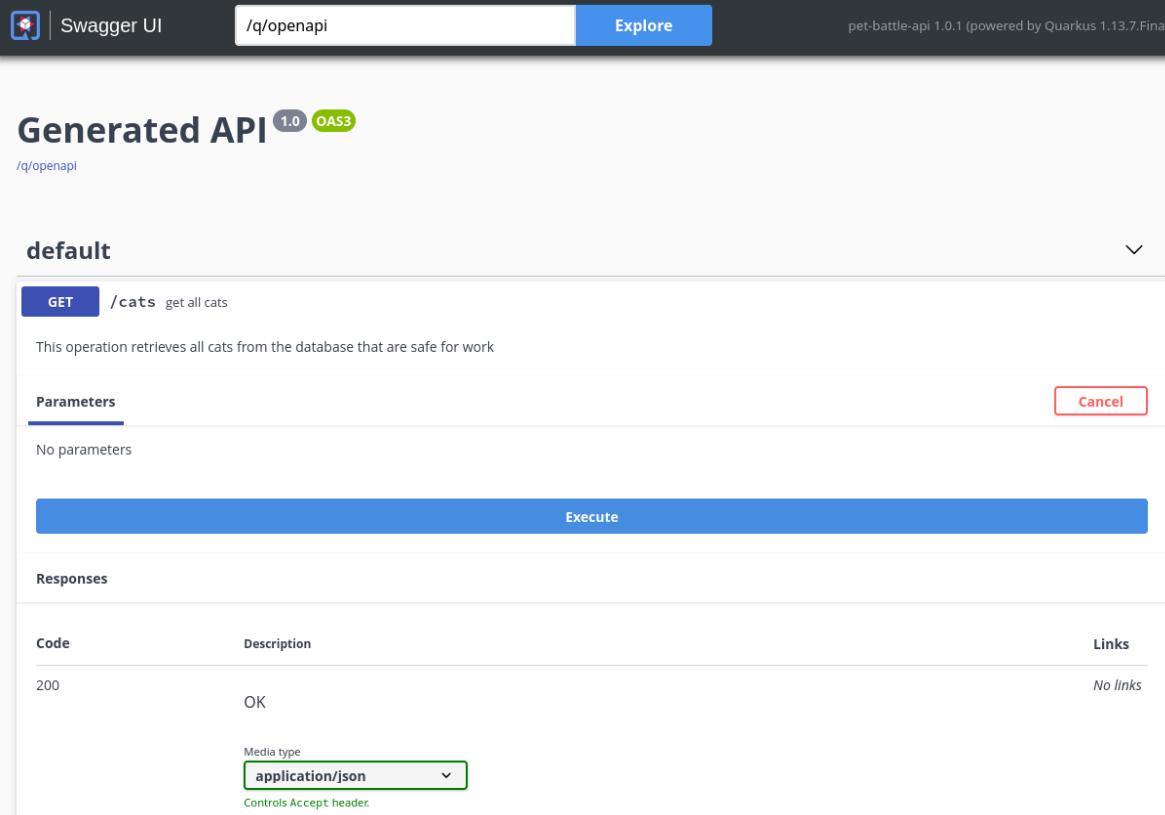
default

▼

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

2-3-1. 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。



The screenshot shows the Swagger UI interface for a generated API. At the top, it displays "Swagger UI" and the URL "/q/openapi". To the right of the URL is a blue "Explore" button. On the far right, it says "pet-battle-api 1.0.1 (powered by Quarkus 1.13.7.Final)". Below the header, the title "Generated API" is followed by "1.0" and "OAS3". A link to "/q/openapi" is also present. The main content area is titled "default". It shows a "GET /cats" operation with the description "get all cats". Below the description, it says "This operation retrieves all cats from the database that are safe for work". Under the "Parameters" section, it states "No parameters". To the right of this section is a red "Cancel" button. In the center, there is a large blue "Execute" button. Below the "Execute" button is a "Responses" section. It contains a table with three columns: "Code", "Description", and "Links". There is one row for the status code 200, which is labeled "OK". To the right of the "Links" column, it says "No links". Under the "Description" column for the 200 row, there is a dropdown menu set to "application/json". A small note below the dropdown says "Controls Accept header".

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。



2-3-2. GET /cats/ids: すべての猫のIDを返す

1. 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

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 },]

2-3-3. 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。



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。

2-3-4. 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/2wBDAgGBgcGBQg  
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。



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  
QAAQJ3AAECAxEEBSEExBhJBUQdhcRMiMoEIFEKRobHBcSMzUvAVYnLRChYkN  
6go0EhYaHiImKkpOUlZaXmJmaoq0kpaanqKmqsro0tba3uLm6wsPExcbHyM  
cegI+ppckDg9sfWmsCMYBwDVkClST5jZX6ChwuCSPzoY5IX9TTXkGNvLY7m
```

2-4. 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。

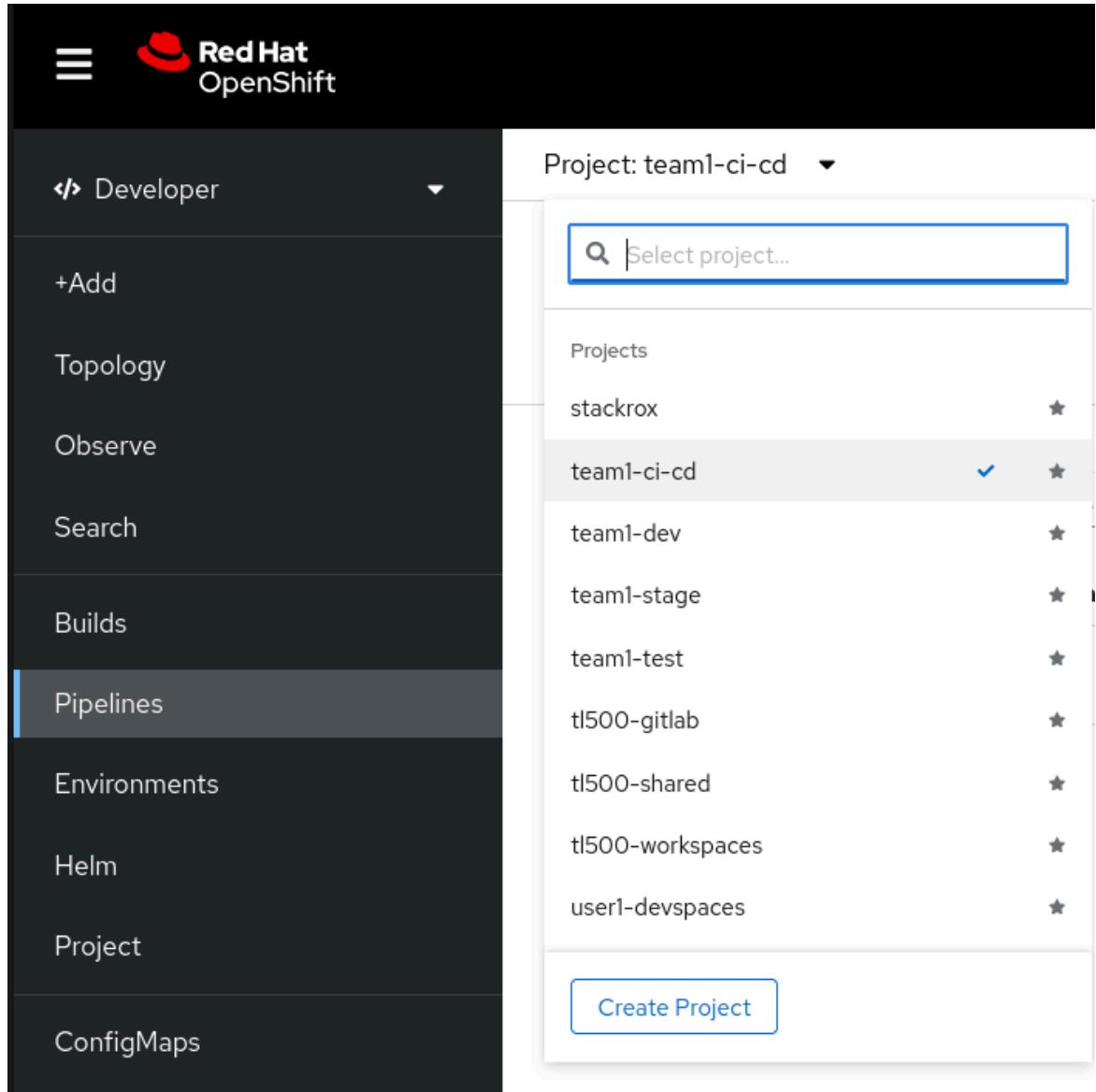
<https://ja.quarkus.io/guides/http-reference#configuring-http-access-logs>

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 displays a dropdown menu titled "Project: team1-ci-cd" with a search bar containing "Select project...". A list of projects is shown, with "team1-ci-cd" selected and marked with a checkmark. 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 panel 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 top navigation bar includes the Red Hat logo, user information (user1), and a search bar. The left sidebar has a 'Developer' section with various options like Topology, Observe, Search, Builds, Pipelines (which is selected and highlighted in blue), Environments, Helm, Project, ConfigMaps, and Secrets. The main content area is titled 'Pipelines' and shows a table with one row. The table columns are Name, Last run, Task status, Last run status, and Last run time. The single row is labeled 'maven-pipeline' with a green PL icon.

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 the 'maven-pipeline'. At the top, it shows the project 'team1-ci-cd' and the pipeline name 'maven-pipeline'. Below this, there are tabs for Details, YAML, PipelineRuns, Parameters, Resources, and Metrics. The 'Details' tab is selected. The main area displays the 'Pipeline details' graph with nodes: fetch-app-repository, maven, bake, helm-package, fetch-argocd-configuration, deploy-test, and verify-deployment. Below the graph are four sections: 'Name' (maven-pipeline), 'Namespace' (team1-ci-cd), 'Labels' (rht-gitops.com/team1-ci-cd+tekton-pipeline), and 'TriggerTemplates' (pet-battle-api-trigger-template). The 'Tasks' section lists the individual steps: git-clone (fetch-app-repository), maven, bake, helm-package, git-clone (fetch-argocd-configuration), deploy (deploy-test), and 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。

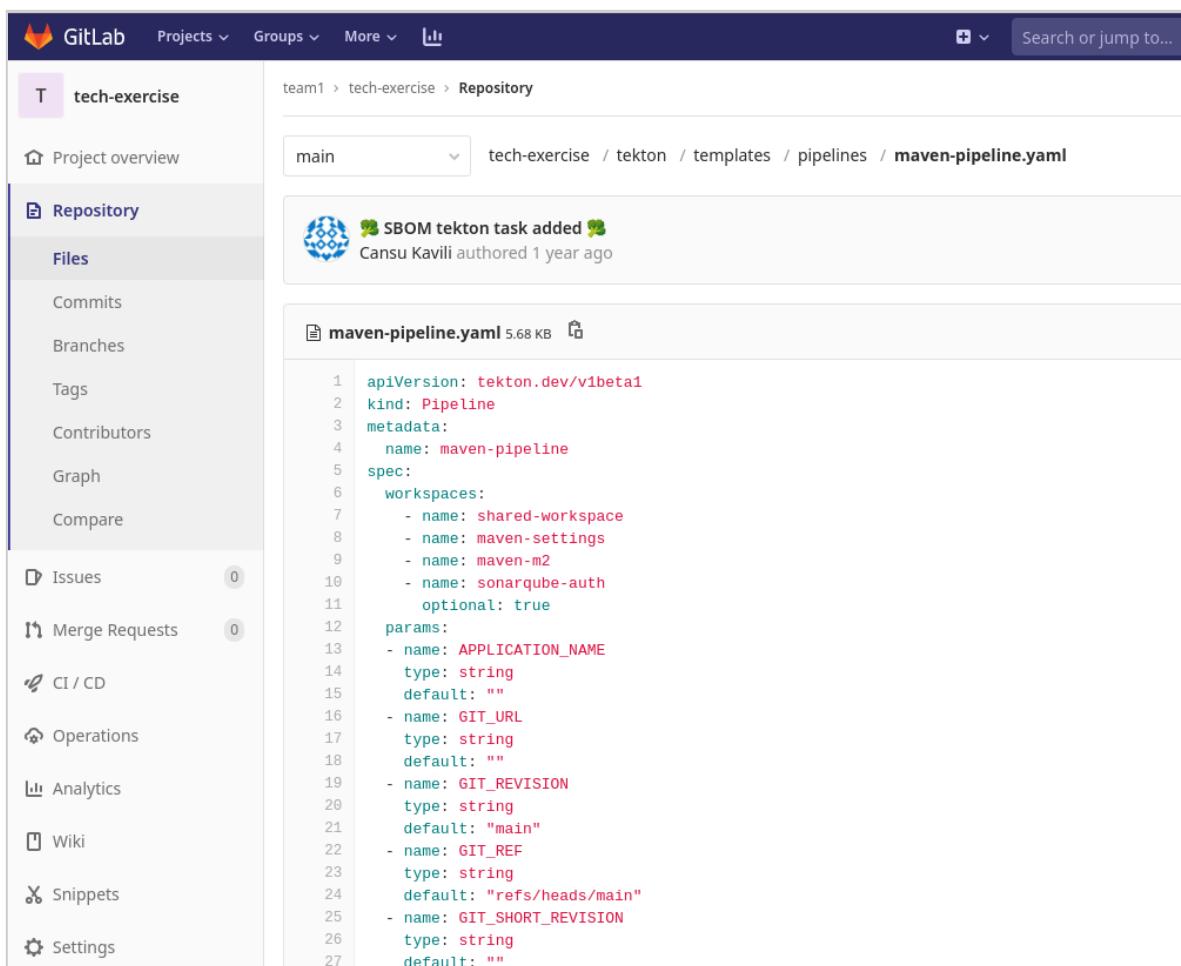
PetBattle APIパイプラインは、GitLabのpet-battle-apiプロジェクトにgit pushされたことがトリガーとなって開始されます(この仕組みをWebHookと言います)。WebHookでは、GitLabからOpenShiftに対してソースコードの変更が発生したことが通知されます。この際、変更のあったGitプロジェクト情報がOpenShiftに渡されます。

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。

3-1-2. 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>
```

3-1-3. 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>

3-1-3. 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。

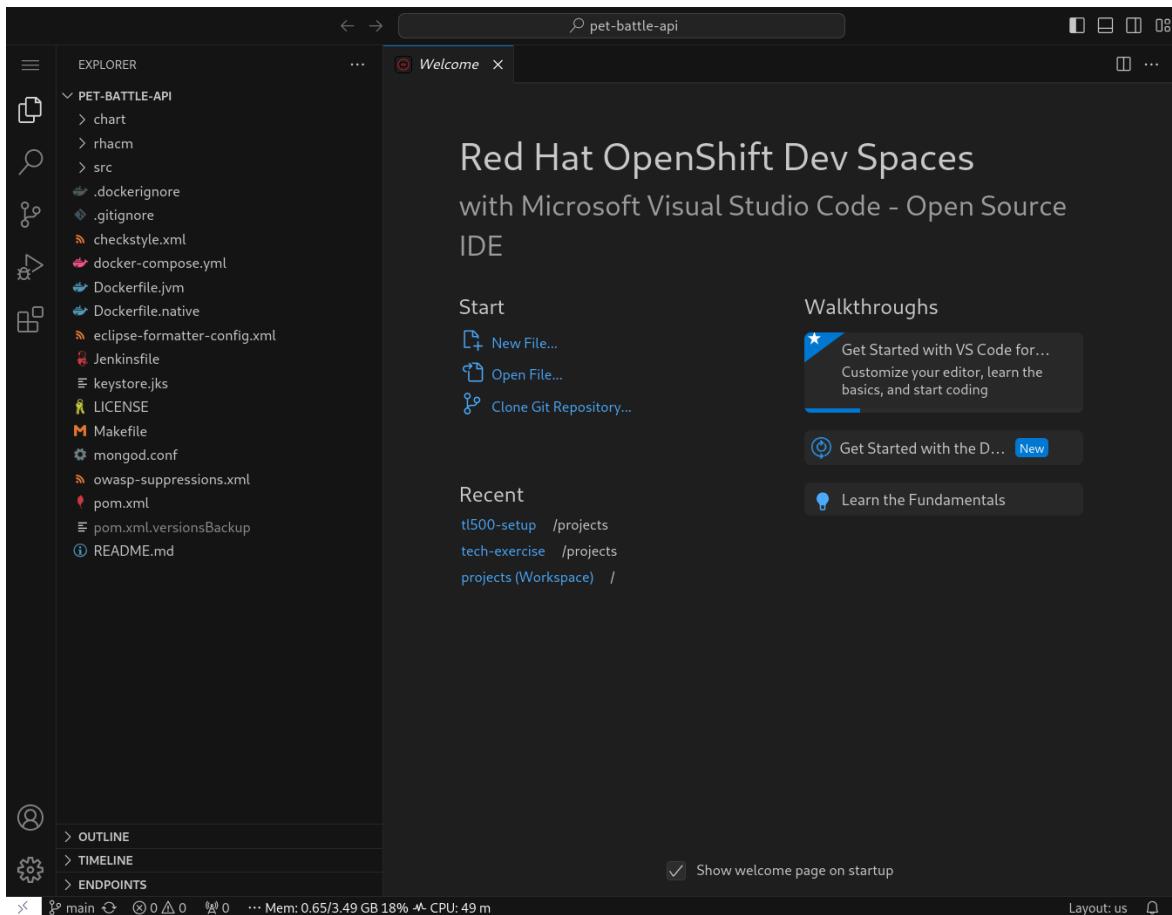
<https://hub.tekton.dev/tekton/task/maven>

3-2. PetBattle API パイプラインを実行する(30分)

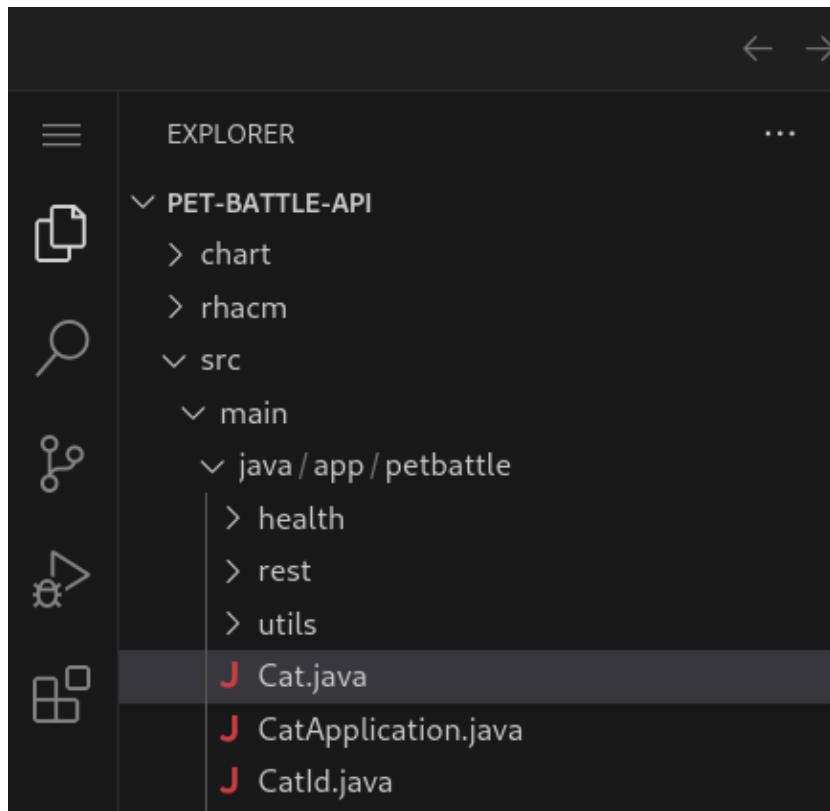
PetBattle APIのソースコードをGitLabのpet-battle-apiリポジトリにgit pushすることによってパイプラインが起動します。実際にPetBattle APIのソースコードの一部を変更してパイプラインを起動し、修正を反映したアプリケーションがデプロイされることを確認しましょう。

3-2-1. PetBattle APIのソースコードを修正する

1. [0-4](#) の手順に従って、CodeReady Workspaces内でVSCodeの環境を開きます。
VSCodeを開いた直後は、左側にファイル階層を表示するExploreの画面になっているはずです。
2. 左上にあるVSCodeのメニューから [File] > [Open Folder] を選び、ディレクトリのメニューから/project/pet-battle-api/ ディレクトリを選択します。以下のように pet-battle-apiのファイル階層が左側に表示されます。



3. 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。
4. 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);  
    }  
}
```

5. 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

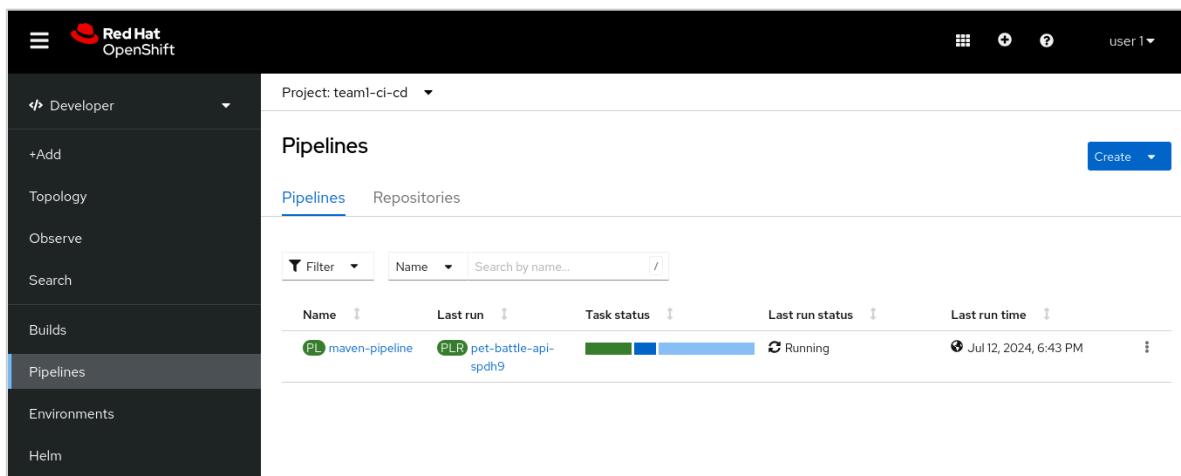
1. VSCodeのメニューから [Terminal] < [New Terminal (Select a Container)] を選択し、画面右下にターミナルを開きます。ターミナル内で以下のコードを実行して修正をGitリポジトリにpushしてください。

```
cd pet-battle-api
git add .
git commit -m 'Changed vote logic'
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。

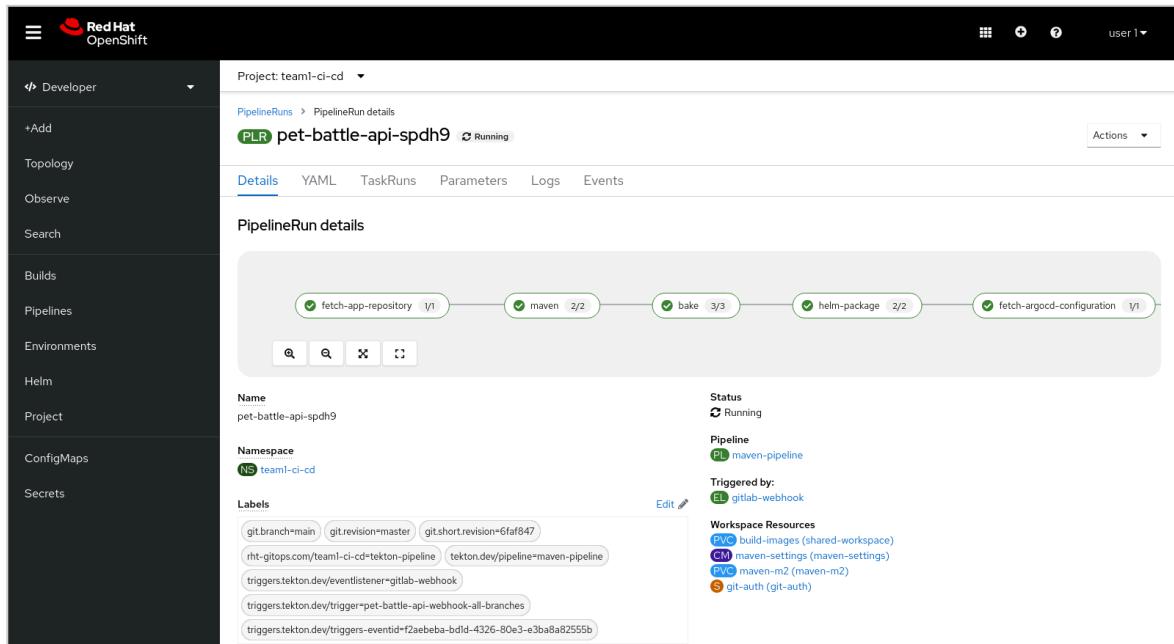
3-2-3. PetBattle APIパイプラインの起動確認

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
maven-pipeline	2024-07-12 14:43:23	Green	Success	2024-07-12 14:43:23
pet-battle-api-spdh9	2024-07-12 14:43:23	Blue	Running	2024-07-12 14:43:23

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 successful with green checkmarks. The 'Details' tab is selected. A 'Workspace Resources' section lists several Persistent Volume Claims (PVCs) and ConfigMaps (CMs) used by the pipeline.

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

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

1. 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。

```
# 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" # - org.owasp:dependency-check-maven:check
    - name: MAVEN_BUILD_OPTS
      value: "-Dsonar.host.url=http://sonarqube-sonarqube:9000
-Dsonar.userHome=/tmp/sonar"
  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
```

2. 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
```

3. 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
```

4. 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 # <- update this 💪💪
  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
```

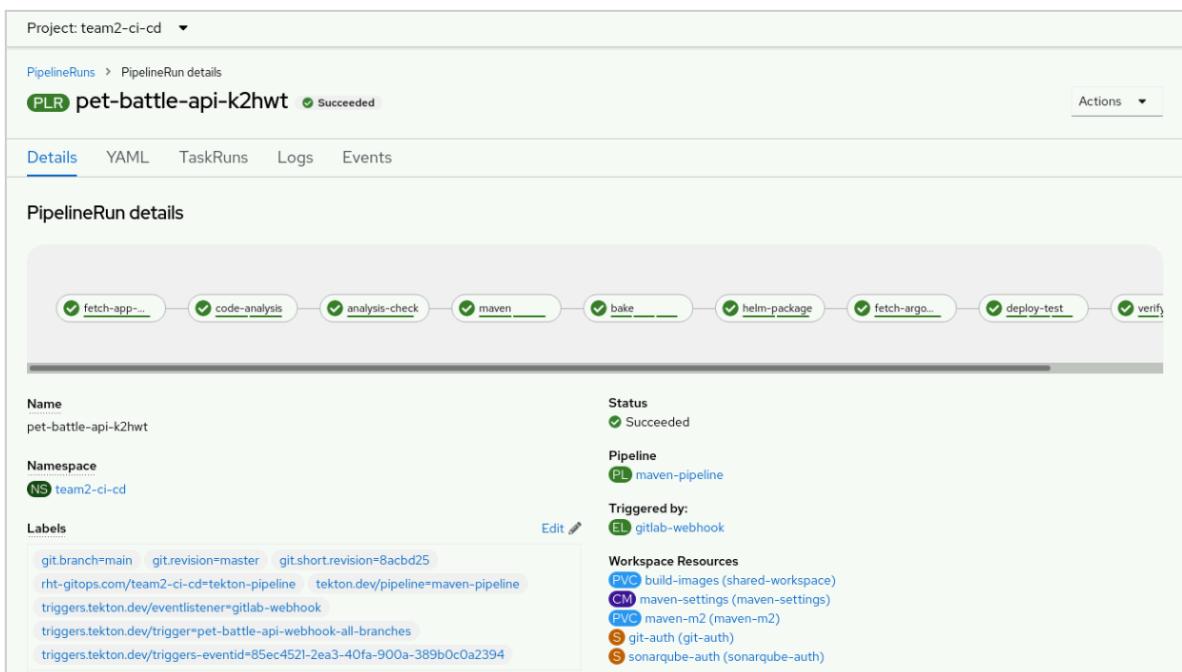
5. パイプライン定義をGitLabにプッシュします。

```
cd /projects/tech-exercise
git add .
git commit -m "ADD - code-analysis & check steps"
git push
```

6. パイプラインビルトをトリガーします。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 /projects/pet-battle-api
git commit --allow-empty -m "TEST - running code analysis steps"
git push
```

7. パイプラインの全体は以下のように拡張されました。



Name	Status
pet-battle-api-k2hwt	Succeeded

Namespace	Pipeline
team2-ci-cd	maven-pipeline

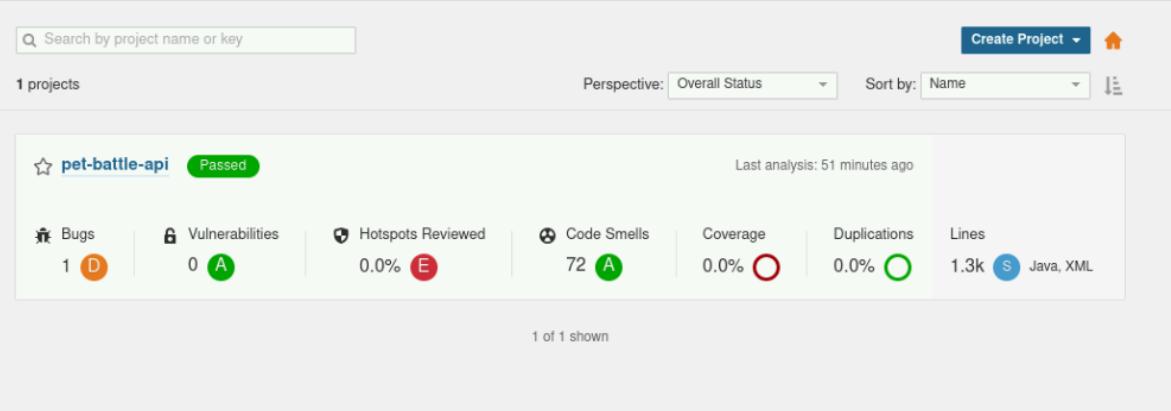
Labels	Triggered by:
git.branch=main git.revision=master git.short.revision=8acbd25 rht-gitops.com/team2-ci-cd=tekton-pipeline tekton.dev/pipeline=maven-pipeline triggers.tekton.dev/eventlistener=gitlab-webhook triggers.tekton.dev/trigger=pet-battle-api-webhook-all-branches triggers.tekton.dev/triggers-eventid=85ec4521-2ea3-40fa-900a-389b0c0a2394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

8. 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)
```

9. 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  Home 

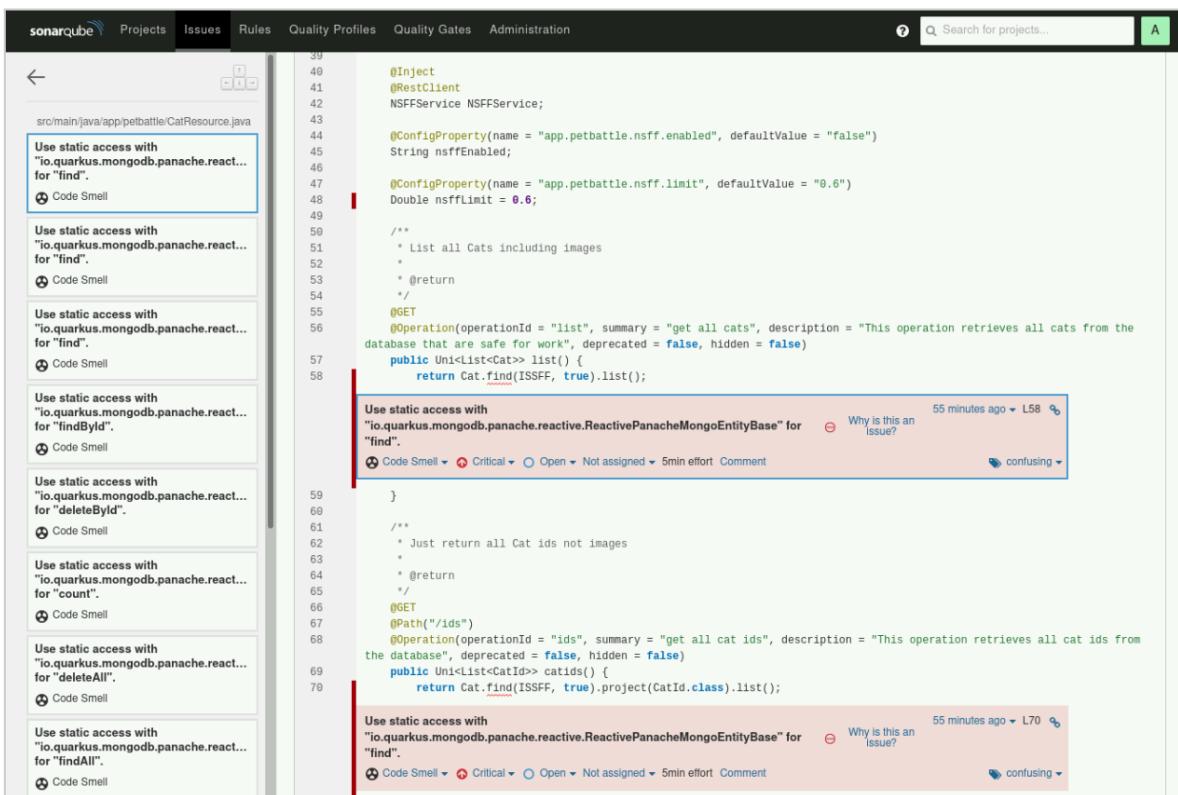
1 projects Perspective: Overall Status Sort by: Name

pet-battle-api Passed 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% O Lines: 1.3k S Java, XML

1 of 1 shown

10. 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。



Projects Issues Rules Quality Profiles Quality Gates Administration Search for projects... A

src/main/java/app/petbattle/CatResource.java

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 "findByld".
Code Smell

Use static access with "io.quarkus.mongodb.panache.react..." for "deleteByld".
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 "findById".
Code Smell

Line 58: Use static access with "io.quarkus.mongodb.panache.reactive.ReactivePanacheMongoEntityBase" for "find".
Code Smell - Critical - Open - Not assigned - 5min effort - Comment - 55 minutes ago - L58 %

Line 70: Use static access with "io.quarkus.mongodb.panache.reactive.ReactivePanacheMongoEntityBase" for "find".
Code Smell - Critical - Open - Not assigned - 5min effort - Comment - 55 minutes ago - L70 %

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

(/projects/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 /projects/tech-exercise
git add .
git commit -m "ADD - save-test-results step"
git push
```

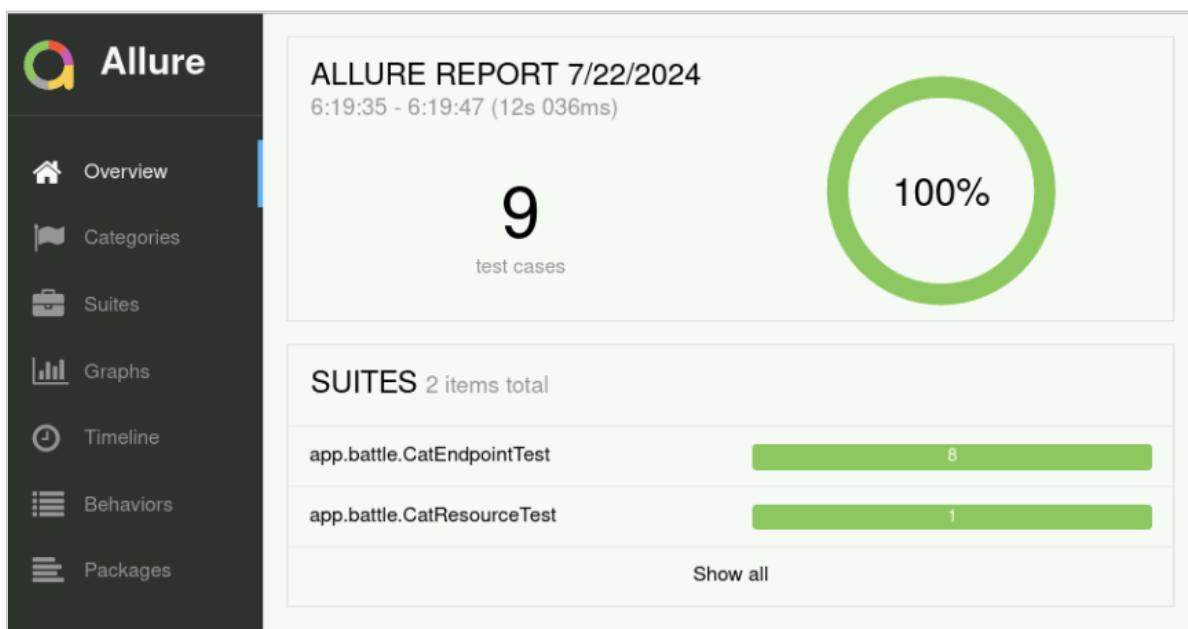
4. 空のコミットで新しいPipelineRunをトリガーし、OpenShift Pipelines に移動して実行を確認します。

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

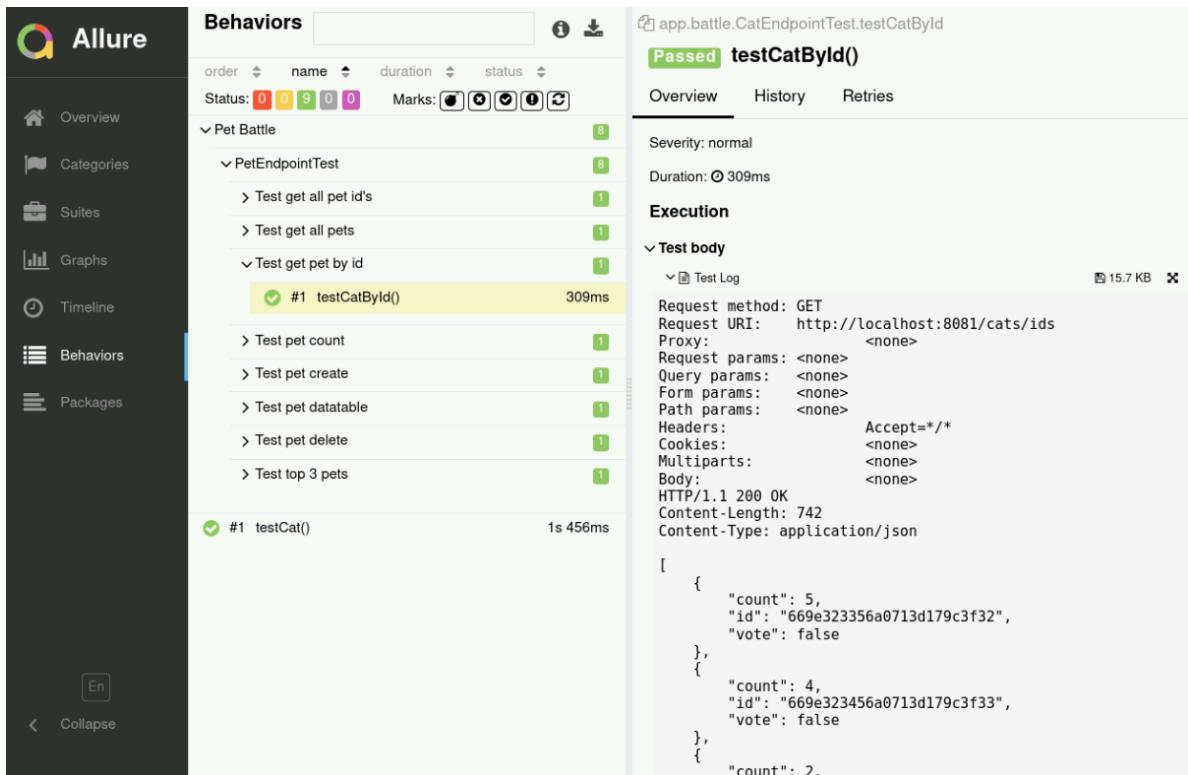
5. パイプラインを実行するとテスト結果はAllureに保存されます。Allure UIのURLを得るには以下を実行します

```
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
```

6. Allure UIのトップ画面です。



7. テスト結果の詳細表示もできます。



The screenshot shows the Allure Testops interface with the 'Behaviors' report for the PetBattle API. The left sidebar includes links for Overview, Categories, Suites, Graphs, Timeline, Behaviors (selected), and Packages. The main panel displays a tree structure of test cases under 'Pet Battle' and 'PetEndpointTest'. The 'testCatById()' test is highlighted in yellow and marked as passed. The right panel provides detailed execution information for this test, including the request method (GET), URI (http://localhost:8081/cats/ids), and the JSON response:

```

Request method: GET
Request URI: http://localhost:8081/cats/ids
Proxy: <none>
Request params: <none>
Query params: <none>
Form params: <none>
Path params: <none>
Headers: Accept=/*
Cookies: <none>
Multiparts: <none>
Body: <none>
HTTP/1.1 200 OK
Content-Length: 742
Content-Type: application/json

[
  {
    "count": 5,
    "id": "669e323356a0713d179c3f32",
    "vote": false
  },
  {
    "count": 4,
    "id": "669e323456a0713d179c3f33",
    "vote": false
  },
  {
    "count": 2,
    ...
  }
]
  
```

参考リスト

- [1] PetBattle <https://github.com/petbattle/pet-battle>
- [2] PetBattle API <https://github.com/petbattle/pet-battle-api>
- [2] Tekton Hub <https://hub.tekton.dev/>