

Couchbase NoSQL Developer Workshop

ラボハンドブック

演習 0 : セットアップ

ラボのセットアップ

このセクションでは、Web UI と API の *Docker* コンテナが作成され、開始されます。以下の手順に従い、手順に問題がある場合は、ヘルプを求めます。

ステップ 1: 作業ディレクトリを作成する

作業ディレクトリを作成すると、ディレクトリ名はラボ参加者に対して行われます。作業ディレクトリは、Web UI および API ラボのリポジトリが存在するディレクトリになります。

```
$ mkdir ~/Documents/cbDevDays/
```

ステップ 2: Web UI をセットアップする

[ステップ 1](#) で作成した作業ディレクトリに移動します。

```
$ cd ~/Documents/cbDevDays/
```

2a: Web UI git リポジトリのクローンを作成する

cb-dev-days-Web リポジトリには、ラボで使用する Web UI を実行するために必要なすべてのリソースが含まれています。 *git CLI* を使用してリポジトリのクローンを作成します。

```
$ git clone https://github.com/YoshiyukiKono/cb-dev-days-web.git
```

リポジトリのクローンが作成されると、現在の作業ディレクトリに *cb-dev-days-web* ディレクトリが存在する必要があります。作業ディレクトリを *cb-dev-days-web* に変更します。

```
$ cd cb-dev-days-web
```

2b: Web UI *Docker* コンテナを作成します。

Web UI 用の新しい *Docker* コンテナを開始します。 *docker run* コマンドに提供される引数の簡単な説明を以下に示します。

- **--rm** このオプションを使用すると、コンテナが停止したときにコンテナを削除できます。
- **--name** 作成するコンテナのオプション名。このコンテナには、「web」という名前を付けます。
- **-d** *Docker* コンテナをデタッチ モードで実行し、制御がターミナルに戻されるようにします。
- **-v** ボリュームのマッピングに使用されます。

- `-v /app/node_modules` は、上書きしないようにコンテナの `app/node_modules` ディレクトリをブックマークしています
- `-v$(pwd)/app` は現在の作業ディレクトリをコンテナ作業ディレクトリにマッピングしています。これにより、ホストマシン上でコードを変更することができ、その変更はコンテナ内に反映されます。
 - `$(pwd)` は **macOS** および **Linux** オペレーティング・システム用です。
 - **Windows** 環境で、**PowerShell** を使用している場合は `${PWD}` を、コマンドラインを使用する場合は `%cd%` を試す事ができます。
 - Linux スタイルのコマンドを可能にする **Windows** 上で `git bash` を使用することをお勧めします
 - **PWD** コマンドの代わりにフルパスを使用することもできます。**※Windows** で実行していて、上記のどの方法でも「invalid reference format」というエラーが表示された場合は、フルパスを利用してください。
- **-p** ポート転送に使用されます。コンテナのポート **8080** をホストのポート **8080** に転送します

Mac/Linux

```
$ docker run --rm --name web -d -v /app/node_modules -v $(pwd)/:/app -p 8080:8080 thejcfactor/cbdd-web:initial-release
```

```
Unable to find image 'thejcfactor/cbdd-web:initial-release' locally
initial-release: Pulling from thejcfactor/cbdd-web
cbd31ae33279: Pull complete
4e8f99b4e7b4: Pull complete
777e8f2fd0f3: Pull complete
a831261d2b41: Pull complete
e78ee2a0a70a: Pull complete
327132737c89: Pull complete
d6a6a16d1425: Pull complete
7949923d73f7: Pull complete
5eec14690faa: Pull complete
a0ab7354739e: Pull complete
Digest: sha256:5194ceee3665ba29e1c9dfdcc26163c70198f982e5204f000b6202d55a11d697
Status: Downloaded newer image for thejcfactor/cbdd-web:initial-release
0f1b0e668f753ae757b10d26a8962efb45e8021f891fd524c9f175f176897d63
```

Windows PowerShell

```
> docker run --rm --name web -d -v /app/node_modules -v ${PWD}/:/app -p 8080:8080 thejcfactor/cbdd-web:initial-release
```

コンテナが実行されている場合、**Docker** はコンテナ ID を表示します。

注 - コンテナ ID は、ドキュメント内のものとは異なります。

次の **Docker** コマンドを使用して、コンテナが実行されていることを確認します。

```
$ docker container ls --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0f1b0e668f75	thejc...	"docker-e..."	13 minutes ago	Up 13 minutes	0.0.0.0:8080->8080/tcp	web

次のコマンドを実行して、コンテナログを表示できます。 ログ出力の内容は、下のウィンドウで表示するために切り捨てられますが、主なポイントはアプリが実行されているのを見る事、つまり強調表示された行です。

```
$ docker logs web
```

```
...
```

```
App running at:
```

```
- Local: http://localhost:8080/
```

```
It seems you are running Vue CLI inside a container.
```

```
Access the dev server via http://localhost:<your container's external mapped port>/
```

```
Note that the development build is not optimized.
```

```
To create a production build, run npm run build.
```

ブラウザを開きます(Chrome または Firefox をお勧めします。インターネット エクスプローラを使用しないでください。Web UI の URL に移動します。: <http://localhost:8080/>)

次の Web ページが表示されます。

注意 - もしも他のページ (ログインページ等) にリダイレクトされた場合は、左上隅にある「Couchbase|NoEQUAL」をクリックすることでホームページ (下記画面) に移動する事ができます。

ステップ 3: REST API の作業ディレクトリを作成する

[ステップ 1](#) で作成した作業用ディレクトリに移動します。

```
$ cd ~/Documents/cbDevDays/
```

API ラボ用の作業ディレクトリを作成し、そのディレクトリに移動します。

```
$ mkdir node3x
$ cd node3x
```

ステップ 4: REST API のセットアップ

4a: API git リポジトリのクローンを作成

cb-dev-days-node3x リポジトリには、ラボで使用する **Node.js API** を実行するために必要なすべてのリソースが含まれています。 **git CLI** を使用してリポジトリのクローンを作成します。

```
$ git clone https://github.com/YoshiyukiKono/cb-dev-days-node3x.git
```

repository が複製されたら、現在の作業ディレクトリに **cb-dev-days-node3x** ディレクトリが存在する必要があります。作業ディレクトリを **cb-dev-days-node3x** に変更します。

```
$ cd cb-dev-days-node3x
```

4b: API 構成設定を編集

付録 に詳細な **Node.js API** のプロジェクト構造に [注意してください](#)。

config.json ファイルを表示/編集するには、構成ディレクトリに移動します。

```
$ cd configuration/
```

任意のテキストエディタを使用して **config.json** ファイルを（必要に応じ）編集します。

以下に強調表示されている次の値を編集します。

- **host** - Couchbase Server クラスターのホスト名の一覧。自分で準備した **Couchbase Server** を使う場合はそのホスト（**Docker** 上で動かしている場合は「**host.docker.internal**」、直接ローカル環境へインストールした場合は「**localhost**」）を、または **Couchbase** チームによって提供されたホストを指定します。（**Couchbase** チームによって提供された環境が、セキュア接続に設定されている場合、**secure** を「**true**」に変更します。）

```
{
  "dev": {
    "appName": "node_3x_api",
    "port": "3000",
    "database": {
      "host": "host.docker.internal",
      "secure": false,
      "bucket": "retailsample",
      "username": "svc-devdays",
      "password": "password"
    },
    "secret": "4*@L$VzRp&E%cYZDSO^E6IPyygYpKHTXJtlOlt!zKTZF@@6l81^rJ1xmCPN%MD2",
    "sessionTTL": 300
  },
  "prod": {}
}
```

4c: API Docker コンテナ作成

メイン API 作業ディレクトリに戻ります。

```
$ cd ..
$ ls -ltra

total 240
drwxr-xr-x   3 root   root    96 Sep 10 09:08 ..
-rw-r--r--   1 root   root    13 Sep 10 09:08 .dockerignore
-rw-r--r--   1 root   root  1791 Sep 10 09:08 .gitignore
-rw-r--r--   1 root   root   866 Sep 10 09:08 Dockerfile.dev
-rw-r--r--   1 root   root  1987 Sep 10 09:08 README.md
drwxr-xr-x   5 root   root   160 Sep 10 09:08 controllers
drwxr-xr-x   6 root   root   192 Sep 10 09:08 library
-rw-r--r--   1 root   root 96678 Sep 10 09:08 package-lock.json
-rw-r--r--   1 root   root   648 Sep 10 09:08 package.json
drwxr-xr-x   3 root   root    96 Sep 10 09:08 repository
drwxr-xr-x   5 root   root   160 Sep 10 09:08 resources
-rw-r--r--   1 root   root  1779 Sep 10 09:08 server.js
drwxr-xr-x   4 root   root   128 Sep 10 09:08 service
drwxr-xr-x  12 root   root   384 Sep 10 09:08 .git
drwxr-xr-x   4 root   root   128 Sep 10 09:17 configuration
drwxr-xr-x   2 root   root    64 Sep 10 09:36 node_modules
drwxr-xr-x  17 root   root   544 Sep 10 09:36 .
```

API 用の新しい *Docker* コンテナを開始します。 `docker run` コマンドに提供される引数の簡単な説明を以下に示します。

- **--rm** このオプションを使用すると、コンテナが停止したときにコンテナを削除できます。
- **--name** 作成されたコンテナの名前を"api"にします。
- **-d** *Docker* コンテナをデタッチ モードで実行し、制御がターミナルに戻されるようにします。
- **-v** ボリュームのマッピングに使用されます。
 - **-v /app/node_modules** は、上書きしないようにコンテナの `app/node_modules` ディレクトリをブックマークしています
 - **-v\$(pwd)/app** は現在の作業ディレクトリをコンテナ作業ディレクトリにマッピングしています。これにより、ホストマシンでコードを変更することができ、その変更はコンテナ内に反映されます。
 - **\$(pwd)** は macOS および Linux オペレーティング・システム用です。
 - **PowerShell** を使用している場合は **\$(PWD)** を試み、コマンドラインを使用する場合は **%cd%** を試します。
 - Linux スタイルのコマンドを許可する Windows で `git bash` を使用することをお勧めします
 - **PWD** コマンドの代わりにフルパスを使用することもできます。
- **-p** ポート転送に使用されます。コンテナのポート 3000 をホストのポート 3000 に転送します

```
$ docker run --rm --name api -d -v /app/node_modules -v $(pwd)/:/app -p
3000:3000 thejcfactor/cbdd-node3x-api:initial-release
```

```
Unable to find image 'thejcfactor/cbdd-node3x-api:initial-release' locally
initial-release: Pulling from thejcfactor/cbdd-node3x-api
cbd31ae33279: Already exists
4e8f99b4e7b4: Already exists
```

```
777e8f2fd0f3: Already exists
a831261d2b41: Already exists
e78ee2a0a70a: Already exists
2cceca9cefaa: Pull complete
4be2a08602ec: Pull complete
7c02142cc0bf: Pull complete
1b74b3c71fc7: Pull complete
b51933a772d5: Pull complete
4120a3913840: Pull complete
99317e52396d: Pull complete
a277f3ba3ad2: Pull complete
d2cdacc930ec: Pull complete
11391cc82c3c: Pull complete
0e694068ea02: Pull complete
Digest: sha256:7a38fa5925083a7a367bf1ce6fd136976eee72807f2abc347c05e018f5ea0fc8
Status: Downloaded newer image for thejcfactor/cbdd-node3x-api:initial-release
e3fe36fcd3436daaa7a178811e23d8a2855bc791db96068d4836c9d46ef1bb78
```

次の *Docker* コマンドを使用して、コンテナが実行されていることを確認します。

```
$ docker container ls --all
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
0f1b0e668f75	thejc...	"docker-e..."	33 minutes ago	Up 33 minutes	0.0.0.0:8080->8080/tcp	web
e3fe36fcd343	thejc...	"docker-e..."	5 minutes ago	Up 5 minutes	0.0.0.0:3000->3000/tcp	api

次のコマンドを実行して、コンテナ ログを表示します。サーバーはポート **3000** でリッスンしています。

```
$ docker logs api
```

```
> node-3x-api@1.0.0 dev /app
> nodemon
```

```
[nodemon] 2.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
2020-04-24 02:35:34: server.js - listening on port 3000
2020-04-24 02:35:34: repository.js:connect() - connected to bucket:  retailsample
2020-04-24 02:35:34: repository.js:connect() - ping response {
  config_rev: 37,
  id: '0x31e6810/5cef0178dcd3f603',
  sdk: 'libcouchbase/3.0.6-njs couchnode/3.0.7 (node/12.18.2; v8/7.8.279.23-node.39; ssl/1.1.1g)',
  services: {
    fts: [ [Object] ],
    kv: [ [Object] ],
    n1ql: [ [Object] ],
    views: [ [Object] ]
  },
  version: 1
}
```

Couchbase Server に接続できていない場合、下記のログが表示されます。

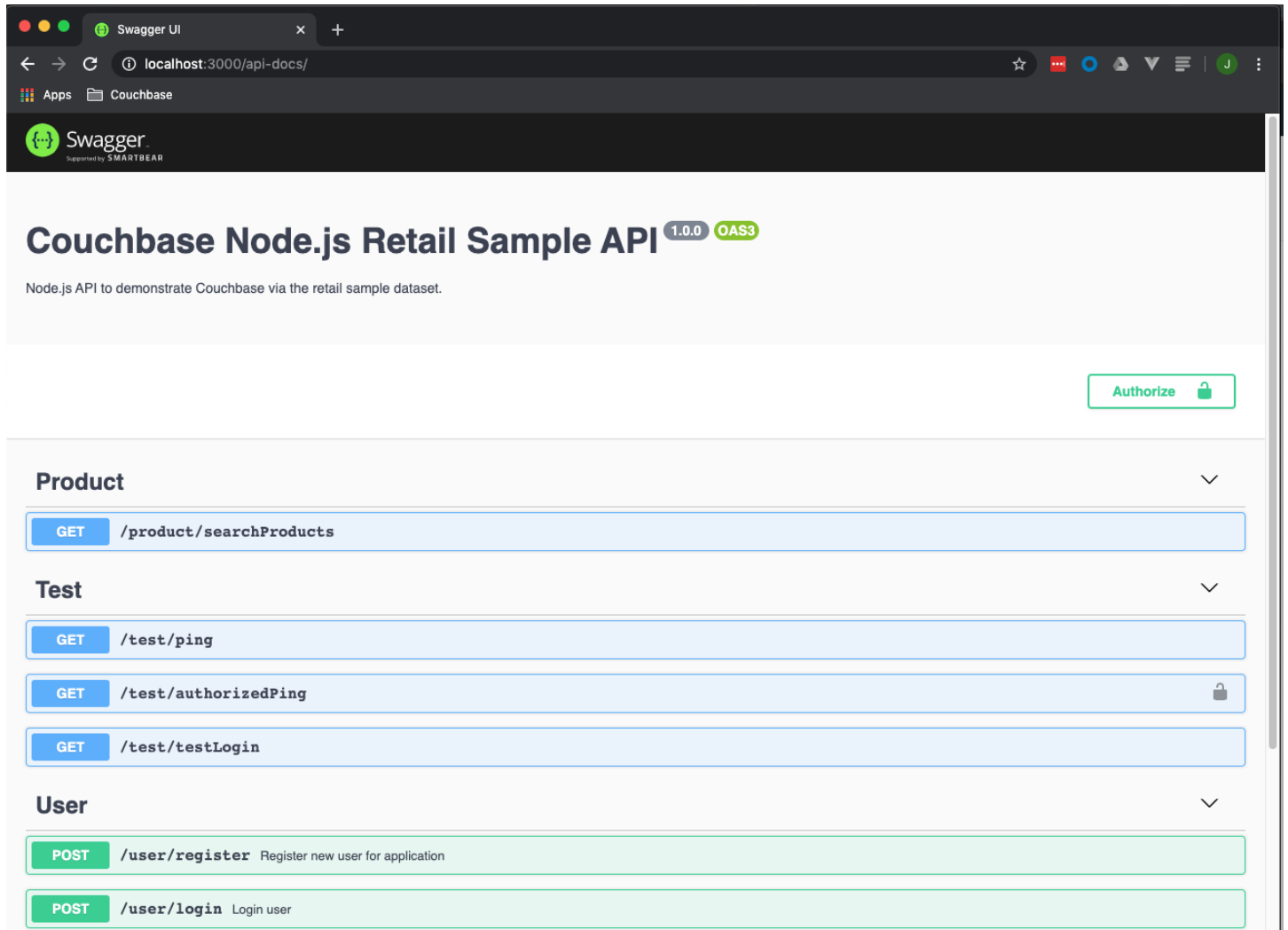
```
2021-05-11 02:47:05: repository.js:connect() - error pinging to bucket. Error:
cluster object was closed
    at Connection.close (/app/node_modules/couchbase/lib/connection.js:265:24)
```

```
at /app/node_modules/couchbase/lib/cluster.js:605:14
```

4d: API Docker コンテナ確認

ブラウザを開きます(Chrome または Firefox をお勧めします。インターネット エクスプローラを使用しないでください)。API の SwaggerUI URL に移動してください: <http://localhost:3000/api-docs/>

次の Web ページが表示されます。



[Swagger] ページでは、API を直接操作できます。残りの演習では、製品とユーザーのパスの下にある特定のエンドポイントの機能を構築します。


4e: Couchbase 接続をテストする

[Swagger API] ページの[テストパス]の下にある ping エンドポイントを使用して、Couchbase の接続を確認できます。ping エンドポイントは SDK のヘルスチェック API を利用します。以下のステップに従って実行します(ステップのビデオについては、後掲のクリップを参照してください):

1. [Swagger UI]ページに移動します: <http://localhost:3000/api-docs/>
2. `/test/ping` エンドポイントをクリックします。


3. パネルが展開されたら、[Try it out]ボタンをクリックします。
4. [Execute] ボタンをクリックします。
5. 応答コードは 200、応答本文は次のようになります。

```
{
  "data": "Connected to Couchbase server.",
  "message": "Successfully pinged database.",
  "error": null,
  "authorized": null
}
```

 **Swagger**
powered by SMARTBEAR

Couchbase Node.js Retail Sample API 1.0.0 OAS3

Node.js 2.x API to demonstrate Couchbase via the retail sample dataset.


Authorize 

Product

GET /product/searchProducts

Test

GET /test/ping

GET /test/authorizedPing 

GET /test/testLogin

User

POST /user/register Register new user for application

POST /user/login Login user