多段認証・多段認可デモ

Quarkus × Kong × Keycloak × Redis × React(Vite)

ユーザー → **Kong** → **Quarkus(BFF/Aggregator)** で OIDC ログイン（セッションは **Redis**）。 Quarkus はユーザーの **Access Token** を **Service A/B** に転送し、それぞれの認可ポリシーで判定 → 結果をマージして返します。

- Kong がフロントの**ルーティング**を担当（`/app` は Frontend、`/api/*` は BFF）。
- OIDC コールバックは `/login` に固定し、BFF 側で最終的に `/app/` に戻します。
- フロント（React + Vite dev）は「ログイン」「/api/me」「/api/mashup」などを操作できます。

アーキテクチャ

```
[Browser]  http://localhost:8000
   |  /app,/app/                          | /api/*, /login, /secure, /logout,
/hello
   v                                      v
+------------------+                +------------------+
|  Frontend(Vite)  |                |  Quarkus BFF     |
|  http://frontend:5173             |  http://quarkus-authz:8080
+------------------+                +---------+--------+
                                              |
                                 (Bearer)     |  /a/*, /b/*
                                              v
                                    +---------+---------+
                                    |   Service A / B   |
                                    +-------------------+

Kong:     proxy :8000, admin :8001
Keycloak: http://keycloak:8080 (realm: demo-realm)  ※realms/から自動import
Redis:    OIDC セッション保持
```

前提

- JDK 17、Maven
- Docker / Docker Compose
- Node.js 18+（推奨 20）/ npm（nvm 推奨）
- `jq`（確認に便利）
- `deck` 1.36+（Kong 設定の宣言的適用に使用）

リポジトリ

```
git clone https://github.com/h-i500/multi-authentication.git
cd multi-authentication
```

主要構成（抜粋）：

```
multi-authentication/
├─ docker-compose.yaml
├─ kong/
│  ├─ kong.yml             # DB-less 起動で読み込まれる宣言的設定
（KONG_DECLARATIVE_CONFIG）
│  ├─ kong-deck.yaml       # deck 用 state（DB/DB-less 双方に適用可能）
│  └─ kong-nginx-http.conf # 追加の nginx http コンテキスト（大きめヘッダなどの調整）
├─ realms/                 # Keycloak の realm 定義（自動 import）
├─ frontend/               # Vite + React (base=/app/)
│  ├─ vite.config.ts       # base: '/app/'
│  └─ src/App.tsx          # /api/login, /api/logout, /api/me, /api/mashup
├─ quarkus-authz/          # BFF (OIDC web-app + Redis セッション)
└─ service-a, service-b    # 下流サービス (Bearer)
```

ビルド（初回/変更時）

```
cd quarkus-authz && mvn clean package && cd ..
cd service-a     && mvn clean package && cd ..
cd service-b     && mvn clean package && cd ..
cd frontend      && npm ci && npm run build && cd ..
```

起動（デフォルト：**Kong は DB-less**）

`docker-compose.yaml` は **DB-less** 起動向けです。Kong は `kong/kong.yml` を読み込みます。

```
docker compose up -d --build
```

公開ポート

- Kong 8000/8001, Keycloak 8080, Frontend 5173（任意公開）, Quarkus(BFF) 8081（ホスト公開）

> **注意（DB-less）** `KONG_DECLARATIVE_CONFIG` は **ファイルパス** を指します。 ディレクトリをマウントすると「`.../kong.yml: Is a directory`」エラーで起動しません。

使い方（ブラウザ）

1. `http://localhost:8000/app/` を開く

2. 「ログイン」 → Keycloak 認証 → BFF が `/app/` に戻す
3. `/api/me` と `/api/mashup` が 200 で返ってくるのを確認
4. 「ログアウト」 → `/app/` に戻る（再び `/api/me` で未認証に）

Kong 設定の更新方法

DB-less（推奨の簡単手順）

- `kong/kong.yml` を編集 → **Kong 再起動**（または `POST /config`）で反映
- もしくは deck で現在実体と差分/反映も可能（DB-less でも Admin API 経由で適用できます）

```
# 差分確認
deck gateway diff kong/kong-deck.yaml --select-tag stack-multi-authn

# 反映（DB-lessでも使用可：メモリへ適用）
deck gateway sync kong/kong-deck.yaml --select-tag stack-multi-authn --yes
```

> DB-less で deck による適用を行った場合、その内容は**揮発（再起動で消える）**です。 永続させたい設定は `kong/kong.yml` にも反映しておきましょう。 （例：`deck gateway dump > kong/kong.yml` で現在の実体を落として保存）

DB モードで動かす場合（将来用のメモ）

DB モードに切り替えると Kong の設定は Postgres に永続化されます。 `docker-compose.yaml` の `kong` サービスを DB モード用に戻し、以下を実施します。

```
# （DBモードのみ初回）Kong DB マイグレーション
docker compose run --rm kong kong migrations bootstrap

# deck で反映（DBモード推奨の適用方法）
deck gateway diff kong/kong-deck.yaml --select-tag stack-multi-authn
deck gateway sync kong/kong-deck.yaml --select-tag stack-multi-authn --yes
```

> **DB モードの注意点**
>
> - Postgres には **Kong の全エンティティ（services/routes/plugins/consumers など）**が格納されます。
> - Konga を使う場合は、Konga 用 DB も必要です（同一 Postgres サーバ内の別 DB で運用可）。
> - DB から DB-less へ切り替えるときは、`deck gateway dump > kong/kong.yml` で宣言ファイルを書き出してから移行すると安全です。

Kong のサービス/ルートを **手動（curl）**で作る（idempotent）

> すでに `kong/kong.yml` や `kong-deck.yaml` に同等の定義が入っています。 手で作りたい/検証したい場合は以下を流せば同じ状態になります。

Services

```
# Quarkus BFF (API)
curl -sS -X PUT http://localhost:8001/services/api-svc \
  -d url=http://quarkus-authz:8080 \
  -d tags[]=stack-multi-authn

# Frontend (Vite dev)
curl -sS -X PUT http://localhost:8001/services/frontend-dev \
  -d url=http://frontend:5173 \
  -d tags[]=stack-multi-authn
```

Routes（BFF 側）

```
# /api → BFF。/api は剥がす（/api/* → /*）
curl -sS -X PUT http://localhost:8001/routes/api-route \
  -d service.name=api-svc \
  -d paths[]=/api \
  -d strip_path=true \
  -d preserve_host=true \
  -d path_handling=v0 \
  -d tags[]=stack-multi-authn

# /login（OIDC コールバック）は剥がさない
curl -sS -X PUT http://localhost:8001/routes/login-route \
  -d service.name=api-svc \
  -d paths[]=/login \
  -d strip_path=false \
  -d preserve_host=true \
  -d path_handling=v0 \
  -d tags[]=stack-multi-authn

# /logout も剥がさない
curl -sS -X PUT http://localhost:8001/routes/logout-route \
  -d service.name=api-svc \
  -d paths[]=/logout \
  -d strip_path=false \
  -d preserve_host=true \
  -d path_handling=v0 \
  -d tags[]=stack-multi-authn

# /hello も剥がさない（BFF 直通）
curl -sS -X PUT http://localhost:8001/routes/hello-route \
  -d service.name=api-svc \
  -d paths[]=/hello \
  -d strip_path=false \
  -d preserve_host=true \
  -d path_handling=v0 \
  -d tags[]=stack-multi-authn
```

```
# /secure（OIDC 開始の踏み台）も剥がさない
curl -sS -X PUT http://localhost:8001/routes/secure-route \
  -d service.name=api-svc \
  -d paths[]=/secure \
  -d strip_path=false \
  -d preserve_host=true \
  -d path_handling=v0 \
  -d tags[]=stack-multi-authn
```

Routes（Frontend 側）

```
# /app と /app/ の両方を Frontend へ。Vite dev は strip_path=false & path_handling=v1
が安全
curl -sS -X PUT http://localhost:8001/routes/frontend-dev-route \
  -d service.name=frontend-dev \
  -d paths[]=/app \
  -d paths[]=/app/ \
  -d strip_path=false \
  -d preserve_host=true \
  -d path_handling=v1 \
  -d tags[]=stack-multi-authn
```

確認

```
curl -s http://localhost:8001/services | jq '.data[] | {name, host, port, path}'
curl -s http://localhost:8001/routes    | jq '.data[] | {name, paths, strip_path,
preserve_host, path_handling, service: .service.id}'

curl -i http://localhost:8000/hello
curl -I http://localhost:8000/app/
```

> `/app` と `/app/` を**両方**作る & `strip_path=false`、`path_handling=v1` が Vite dev では重要（301 ループ/アセット 404 を防止）。 `preserve_host=true` は外向きホスト/パス情報を BFF/Frontend に正しく伝えるのに有効です。

Keycloak（demo-realm）

- 管理 UI: `http://localhost:8080/`（Admin: `admin` / `admin`）
- realm は `realms/` から**自動インポート**

代表的なクライアント設定

- `quarkus-client`（BFF 用）: OIDC/Confidential、Authorization Code、Valid Redirect URIs に `http://localhost:8000/*`、`http://localhost:8081/*` 等、Web Origins は `http://localhost:8000`, `http://localhost:8081`。 Credentials の **Secret** を BFF に設定。
- `service-a` / `service-b`（下流）: Bearer-only

ロール & Audience

- `service-a`: `read`, `user` / `service-b`: `read`, `user` を `testuser` に付与
- `quarkus-client` に Audience マッパーで `service-a` と `service-b` を `aud` に含める

BFF（Quarkus）の要点

`quarkus-authz/src/main/resources/application.properties`（抜粋）

```
# OIDC
quarkus.oidc.auth-server-url=http://keycloak:8080/realms/demo-realm
quarkus.oidc.client-id=quarkus-client
quarkus.oidc.credentials.client-secret.value=...   # Keycloak の Secret
quarkus.oidc.application-type=web-app
quarkus.http.proxy.proxy-address-forwarding=true

# コールバックを /login に固定し、戻り先はアプリ側で制御
quarkus.oidc.authentication.redirect-path=/login
quarkus.oidc.authentication.restore-path-after-redirect=false

# PKCE / state
quarkus.oidc.authentication.pkce-required=true
quarkus.oidc.authentication.state-secret=${STATE_SECRET:change-me-change-me-
change-me-1234}

# Redis セッション
quarkus.redis.hosts=redis://redis:6379
quarkus.oidc.token-state-manager.strategy=keep-all-tokens
# 必要なら分割:
# quarkus.oidc.token-state-manager.split-tokens=true

# API 保護の例
quarkus.http.auth.permission.authenticated.paths=/api/*
quarkus.http.auth.permission.authenticated.policy=authenticated
```

`LoginResource` の役割：

- `GET /login`（PermitAll）: 未ログイン→`/secure` に 302、ログイン済→`/app/` に 302
- `GET /secure`（Authenticated）: OIDC 開始 → 認証後 `/app/`
- `GET /logout`: セッション破棄 → `/app/`

Redis でセッション確認

```
docker exec -it <redis-container> redis-cli
127.0.0.1:6379> SCAN 0 MATCH * COUNT 100
127.0.0.1:6379> TTL "oidc:token:..."   # 残存時間
```

下流サービス（A/B）の要点

```
# 共通
quarkus.oidc.auth-server-url=http://keycloak:8080/realms/demo-realm
quarkus.oidc.application-type=service
quarkus.oidc.roles.source=accesstoken

# service-a:
quarkus.oidc.client-id=service-a
quarkus.oidc.roles.role-claim-path=resource_access["service-a"].roles

# service-b:
quarkus.oidc.client-id=service-b
quarkus.oidc.roles.role-claim-path=resource_access["service-b"].roles
```

よくあるハマりどころ

- **/app が 301 ループ/アセット 404** → Kong で `/app` と `/app/` の**両方**をルート登録、`strip_path=false`、`path_handling=v1`。 → Vite の `base='/app/'` を確認。
- **/login が 404** → `login-route` が `strip_path=false` か確認。剥がしてしまうと BFF の `/login` に届かず 404。
- **「State parameter can not be empty」ログ** → `/login` を直接開いた直後などに見えることがあります（導線として `/secure` 経由で問題なし）。
- **Keycloak 未起動 / 起動待ち不足** → Compose の `healthcheck` を入れるか、起動順に注意。
- **DB-less で起動しない** → `KONG_DECLARATIVE_CONFIG` が**ファイル**を指しているか確認（ディレクトリは不可）。

主要エンドポイント

- Frontend: `http://localhost:8000/app/`
- API(BFF): `http://localhost:8000/api/*`
  - ログイン: `/api/login`（→ `/secure` → Keycloak → `/login` → `/app/`）
  - ログアウト: `/api/logout`
  - 認証ユーザ: `/api/me`
  - マッシュアップ: `/api/mashup`
- Keycloak: `http://localhost:8080/`（realm: `demo-realm`）
