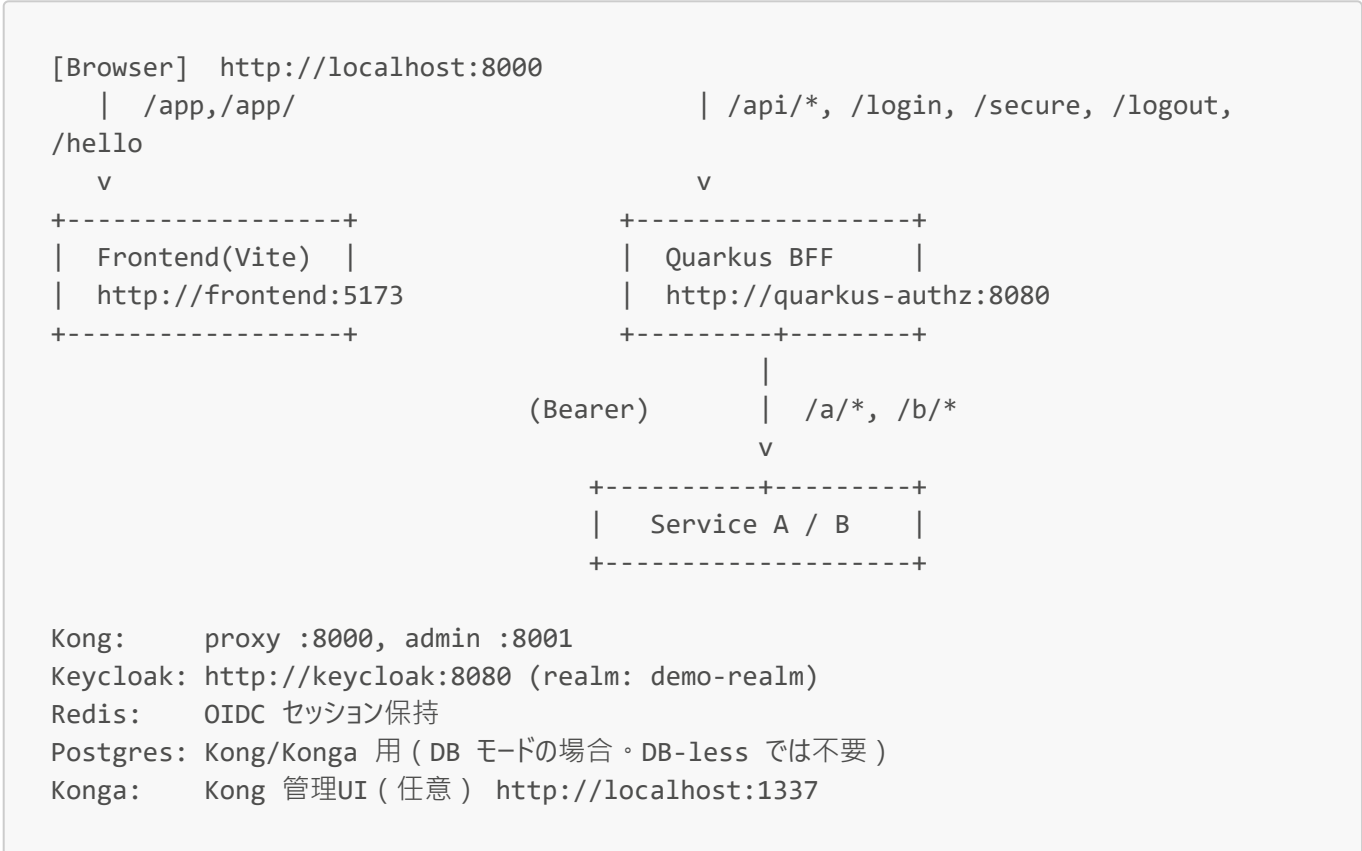
多段認証・多段認可デモ

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 側の実装で最終的に `/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



ログインの流れ（要点）

1. `http://localhost:8000/app/` を開く
2. フロントの「ログイン」 → `window.location.assign('/api/login')`
3. BFF の `/login` ハンドラが **未認証なら `/secure` に 302** `/secure` は `@Authenticated` なので Quarkus OIDC の**コードフロー**開始
4. Keycloak 認証 → **コールバックは `/login`**（BFF が受ける）
5. BFF がセッション確立後、**`/app/` に 302** で戻す
6. 以降、`/api/me` / `/api/mashup` が利用可能

> フロントの「ログイン/ログアウト」は**トップレベル遷移**（`window.location.assign(...)`）にしてあります。

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 設定を宣言的に適用する場合）

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-deck.yaml         # deck 用の宣言的設定（DB/DB-less 両対応）
│  └─ kong-nginx-http.conf   # 追加の nginx http コンテキスト
├─ 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）

> Dockerfile がローカルビルド成果物を参照する想定のため、事前にビルドしておくのが安全です。

```
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（Kong **DB モード**：既定）

`docker-compose.yaml` 既定は **DB モード**（Postgres を使用）です。

```
# 基盤系
docker compose up -d --build kong-database redis keycloak
# Keycloak 起動待ち（数秒～十数秒）

# （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

# Kong / Konga
docker compose up -d --build kong konga

# アプリ群
docker compose up -d --build service-a service-b quarkus-authz frontend

# まとめて一括でもOK
# docker compose up -d --build
```

公開ポート:

- Kong 8000/8001, Keycloak 8080, Quarkus BFF 8081（ホスト公開）, Service-A 9081, Service-B 9082, Frontend 5173（任意）

Kong の設定を適用（deck 推奨）

`kong/kong-deck.yaml` は**いま動いている成功構成**に一致しています。 タグ `stack-multi-authn` を使って選択的に適用できます。

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

# 反映
deck gateway sync kong/kong-deck.yaml --select-tag stack-multi-authn --yes
```

> 旧 `deck diff/sync -s` は非推奨です。`gateway diff/sync` を使ってください。

手動（curl）で作る場合（idempotent）

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。strip_path=false, path_handling=v1 がポイント
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}'
```

> `/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）。

Kong を **DB-less** で動かしたい場合（オプション）

DB-less は起動時に **ファイル**から設定を読み込みます。`kong` サービスを切り替えてください。

```
# docker-compose.yaml の kong を差し替え（例）
kong:
  image: kong:3.6.0
  environment:
    KONG_DATABASE: "off"   # ★ DB-less
    KONG_DECLARATIVE_CONFIG: /usr/local/kong/declarative/kong.yml
    KONG_ADMIN_LISTEN: 0.0.0.0:8001
    KONG_PROXY_ACCESS_LOG: /dev/stdout
    KONG_ADMIN_ACCESS_LOG: /dev/stdout
    KONG_PROXY_ERROR_LOG: /dev/stderr
    KONG_ADMIN_ERROR_LOG: /dev/stderr
    KONG_PLUGINS: bundled
    KONG_NGINX_HTTP_INCLUDE: /usr/local/kong/nginx-http.conf
  volumes:
    - ./kong/kong-nginx-http.conf:/usr/local/kong/nginx-http.conf:ro
    - ./kong/kong.yml:/usr/local/kong/declarative/kong.yml:ro    # ★「ディレクトリ」ではなく「ファイル」をマウント！
  ports: ["8000:8000","8001:8001"]
```

> **注意**: `KONG_DECLARATIVE_CONFIG` は**ファイルパス**です。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。

`kong.yml` を deck から生成する例:

```
deck gateway dump --format yaml --select-tag stack-multi-authn > kong/kong.yml
```

Keycloak（demo-realm）

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

クライアント（代表）

- `quarkus-client`（BFF 用）
  - OpenID Connect / Confidential（Client Authentication ON）
  - 標準フロー（Authorization Code）ON
  - Valid Redirect URIs: `http://localhost:8000/*`, `http://localhost:8081/*` 等
  - Web Origins: `http://localhost:8000`, `http://localhost:8081`
  - Credentials の **Secret** を BFF に設定
- `service-a` / `service-b`（下流）
  - Bearer-only（ログイン画面なし）

ロール & Audience

- Client Roles
  - `service-a`: `read`, `user`
  - `service-b`: `read`, `user`
- `testuser` に上記ロールを付与
- `quarkus-client` に **Audience マッパー**で `service-a`/`service-b` を付与（access token の `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 でセッションを外出し（token-state-manager 関連）
quarkus.redis.hosts=redis://redis:6379
quarkus.oidc.token-state-manager.strategy=keep-all-tokens
# 必要に応じて：
# quarkus.oidc.token-state-manager.split-tokens=true
# quarkus.oidc.redis-token-state-manager.redis-client-name=session

# API 保護（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/`

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
```

必要に応じて `@RolesAllowed("read")` を付与してください。

Frontend（Vite/React）

- `vite.config.ts` は `base: '/app/'` を設定済み
- Kong の `/app` ルートは **`/app` と `/app/` の両方**を登録し、`strip_path=false`、`path_handling=v1`

`src/App.tsx`（抜粋）

```
<button onClick={() => window.location.assign('/api/login')}>ログイン</button>
<button onClick={() => window.location.assign('/api/logout')}>ログアウト</button>

<button onClick={() => apiGet('/me')}>/api/me</button>
<button onClick={() => apiGet('/mashup')}>/api/mashup</button>
<a href="/app/" style={{ marginLeft: 8 }}>トップへ</a>
```

動作確認

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

ブラウザで `http://localhost:8000/app/` → 「ログイン」 → 認証後 `/app/` へ戻る `/api/me` と `/api/mashup` が 200 で応答することを確認。

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` になっているか。誤って剥がすと BFF の `/login` へ届きません。
- **「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 未起動による OIDC 接続失敗** → Compose の起動順や `healthcheck` を確認。
- **DB-less で起動しない** → `KONG_DECLARATIVE_CONFIG` は**ファイル**をマウントしてください（ディレクトリは不可）。

Kong を DB-less に切り替える時の注意

- Admin API 経由の変更は**揮発**です（再起動で消える）。
- 設定は `kong/kong.yml` を**正**にして、`deck gateway dump/sync` で管理するのがおすすめ。
- 既存の Postgres は **Kong のコンフィグ（サービス/ルート/プラグイン/コンシューマ等）** と **Konga 用 DB** に使われています。 DB-less に移行するなら、Postgres/Konga は停止・削除して構いません（必要ならバックアップを）。

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`）
- Konga（任意）: `http://localhost:1337/`
