手順

全体像（最終到達形）

ユーザー → **Kong** `/mashup` → **Quarkus(集約API)** が認証（OIDC） → Quarkus が **ユーザーの Access Token をそのまま下流へ転送** → **Service A / Service B で各自のポリシーで認可** → 結果をマージして返却

1. 前提・ディレクトリ

```
.
├─ docker-compose.yaml
├─ kong-nginx-http.conf
├─ init-konga-db.sql
├─ quarkus-authz/           # 集約API (web-app + service テナント)
├─ service-a/
└─ service-b/
```

2. ビルド（各サービスを fast-jar で）

各モジュールで:

```
cd quarkus-authz && mvn clean package && cd ..
cd service-a     && mvn clean package && cd ..
cd service-b     && mvn clean package && cd ..
```

> Dockerfile は `target/quarkus-app/...` と `quarkus-run.jar` 前提なので、**必ず** `mvn clean package` **実行**してください。

3. 起動（Docker Compose）

```
docker compose up -d --build kong-database redis keycloak
# 数秒待機（Keycloak起動待ち）

# 初回のみ Kong DB のマイグレーションを実行：
docker compose run --rm kong kong migrations bootstrap
```

```
docker compose up -d --build kong konga
docker compose up -d --build service-a service-b quarkus-authz
```

> `version` 警告は無視可。`quarkus-authz/service-a/service-b` は 8080 で Listen、Compose で 8081/9081/9082 に公開。

---

4. Keycloak 設定（demo-realm）

(以下手動で実施する場合です。docker-compose起動時、./realms/demo-realm.jsonの内容を読み込むようになっています)

4.1 Realm/ユーザー/クライアント作成（管理コンソール）

- 管理UI: http://localhost:8080/ Admin: `admin` / `admin`

1. **Realm**: `demo-realm` 作成
2. **User**: `testuser` 作成
    - Credentials: `password`（Temporary 解除）
3. **Clients** 3つ作成
    - `quarkus-client`（集約APIブラウザ用）
        - Access Type: Public（Keycloak 24 では「Client authentication = Off」）
        - Standard Flow（Authorization Code）= On
        - Direct Access Grants（Password）= On（開発テスト用）
        - Valid Redirect URIs: `http://localhost:8081/*`
        - Web Origins: `http://localhost:8081`（必要なら `+` でもOK）
    - `service-a`（下流A）
        - Access Type: **Bearer-only**（ログイン画面なし）
    - `service-b`（下流B）
        - Access Type: **Bearer-only**
4. **Client Roles**
    - `service-a` の Roles: `read`, `user`
    - `service-b` の Roles: `read`, `user`
5. **ユーザーへロール付与**
    - `testuser` → Role Mapping → Client Roles → `service-a: read, user` / `service-b: read, user` を付与
6. **Audience(aud) に A/B を含める**

- `quarkus-client` → Client Scopes → Add Mapper → **Audience**
  - Included Client Audience: `service-a`, `service-b`
  - Add to access token: ON （または `quarkus-client` の「Mappers」で Audience マッパを2つ追加しても良い）

> これで、`quarkus-client` で取得したトークンに `aud: ["service-a","service-b","account"]` と `resource_access.service-a.roles / service-b.roles` が入るようになります。

---

5. Kong のルーティング

> OIDC は **Quarkus に任せる**ので、Kong 側は**素通しルート**だけ作ります。 （Konga UI で作っても良いですが、ここでは Admin API 例）

Admin API: http://localhost:8001

```
# Quarkus 集約API用の Service（Docker 内名に向ける）
curl -sS -X POST http://localhost:8001/services \
  -d name=mashup-svc \
  -d url=http://quarkus-authz:8080

# /mashup の Route
curl -sS -X POST http://localhost:8001/services/mashup-svc/routes \
  -d name=mashup-route \
  -d paths[]=/mashup \
  -d strip_path=false
```

Kong のルート設定を確認：

```
curl -s http://localhost:8001/routes | jq '.data[] | {id, paths, strip_path,
service_id: .service.id}'
```

paths に "/mashup" があるルートの id を控えて、strip_path=false に更新します。

```
ROUTE_ID=<さっき控えたID>

curl -sS -X PATCH http://localhost:8001/routes/$ROUTE_ID \
  -d strip_path=false
```

> これで **http://localhost:8000/mashup**（Kong 経由）→ `quarkus-authz:8080/mashup` に到達。

以下ご参考

> `authz-service` は **Quarkus /hello** を裏に向けます。 `/secure` へ来たリクエストを Quarkus `/hello` にルーティング。 Keycloak からのリダイレクト `/hello` にも備えて **/hello ルート**を追加します。

```
# 1) Service 作成（最初はベースURL）
curl -i -X POST http://localhost:8001/services \
  --data name=authz-service \
  --data url=http://quarkus-authz:8080

# 2) Service の URL を /hello に変更（/secure → /hello に届くように）
curl -i -X PATCH http://localhost:8001/services/authz-service \
  --data url=http://quarkus-authz:8080/hello

# 3) /secure ルート作成（初回アクセス入口）
curl -i -X POST http://localhost:8001/services/authz-service/routes \
  --data paths[]=/secure

# 4) /hello ルート作成（Keycloakのredirect_uri先を受ける）
curl -i -X POST http://localhost:8001/services/authz-service/routes \
  --data paths[]=/hello

# 5) Host 情報を上流に渡す（リダイレクト復元に有利）
SECURE_ROUTE_ID=$(curl -s http://localhost:8001/routes | jq -r '.data[] |
select(.paths|index("/secure")) | .id')
HELLO_ROUTE_ID=$(curl -s http://localhost:8001/routes | jq -r '.data[] |
select(.paths|index("/hello")) | .id')

curl -i -X PATCH http://localhost:8001/routes/$SECURE_ROUTE_ID --data
preserve_host=true
curl -i -X PATCH http://localhost:8001/routes/$HELLO_ROUTE_ID  --data
preserve_host=true

# （任意）Kong セッション・プラグイン（保存先=Kong DB）
# ※ OSS の session プラグインは storage=kong or cookie のみ（redisは不可）
curl -i -X POST http://localhost:8001/routes/$SECURE_ROUTE_ID/plugins \
  --data name=session \
  --data config.storage=kong \
  --data config.secret=$(openssl rand -hex 32) \
  --data config.cookie_samesite=Lax \
  --data config.cookie_http_only=true \
  --data config.cookie_secure=false

curl -i -X POST http://localhost:8001/routes/$HELLO_ROUTE_ID/plugins \
  --data name=session \
  --data config.storage=kong \
  --data config.secret=$(openssl rand -hex 32) \
  --data config.cookie_samesite=Lax \
  --data config.cookie_http_only=true \
  --data config.cookie_secure=false
```

> 🔎 補足

> - `/secure` は「入口」用の見せパス。Kong が upstream の `/hello` に繋ぎます。
> - `/hello` ルートは **Keycloak の `redirect_uri` を直接受けるために必要です。**
> - `preserve_host=true` で `Host: localhost:8000` が Quarkus へ伝わり、`quarkus.http.proxy.proxy-address-forwarding=true` と相まって、正しい外部 URL に復元されます。
> - 大きなトークン等で 502/大きいヘッダ系のエラーが出たら `kong-nginx-http.conf` の値を少し増やしてください。

---

6. Quarkus 側の設定（最終確認）

`quarkus-authz/src/main/resources/application.properties`（あなたの現行でOK）

- デフォルトテナント（web-app）
- **Named tenant `"service"`（application-type=service）**
- `/mashup` を `authenticated`
- Rest Client の URL は `http://service-a:8080` / `http://service-b:8080`
- `MashupResource` に `@Tenant("service")`（＝**/mashup は Bearer 専用**）

`MashupResource`（あなたの現行でOK）

- AccessToken を `SecurityIdentity` から取得し、そのまま **Authorization: Bearer …** で下流へ転送

---

7. 下流サービス（A/B）設定

`service-a` / `service-b` の `application.properties`（現行でOK）

```
quarkus.oidc.auth-server-url=${QUARKUS_OIDC_AUTH_SERVER_URL}
quarkus.oidc.client-id=${QUARKUS_OIDC_CLIENT_ID}
quarkus.oidc.application-type=service

# アクセストークン内ロールを使用
quarkus.oidc.roles.source=accesstoken
quarkus.oidc.roles.role-claim-path=resource_access["service-a"].roles  # (A)
# B のほうは ["service-b"]
```

> **スペルは `accesstoken`（ハイフン/アンダースコア無し）** ここが間違うと起動時に enum 変換エラーになります。

> エンドポイントは `A: /a/data` / `B: /b/data`。 認可を厳しくしたい場合は `@RolesAllowed("read")` を復活させれば、 トークン内の `resource_access["service-?"].roles` が使われます。

---

8. 動作確認

8.1 直接（集約API 8081 に対して）

- ブラウザで `http://localhost:8081/mashup` → **初回は 302 で Keycloak ログイン** → ログイン後に `/mashup` へ戻り、結果が表示 （`MashupResource` を web-app で受けてから **内部で service テナントのトークンを使用**する構成）
    - もし **401** が欲しい（リダイレクトしたくない）なら、`MashupResource` を `"service"` テナントのみで受ける構成に変更し、`/mashup` を web-app 側のパーミッションから外す運用にします。今回はログイン要求が要件なので現行のままでOK。
- CLI（パスワードグラントでトークン取得→Bearer で呼ぶ）

```
TOKEN=$(curl -s -X POST \
  'http://localhost:8080/realms/demo-realm/protocol/openid-connect/token' \
  -d 'grant_type=password' \
  -d 'client_id=quarkus-client' \
  -d 'username=testuser' \
  -d 'password=password' | jq -r .access_token)

curl -i http://localhost:8081/mashup -H "Authorization: Bearer $TOKEN"
```

期待レスポンス：

```
{
  "fromServiceA": { "message": "ok-from-A", "detail": "helloA" },
  "fromServiceB": { "message": "ok-from-b", "detail": "helloB" }
}
```

8.2 Kong 経由（プロキシ 8000）

```
# ブラウザ or curl
curl -i http://localhost:8000/mashup
# ブラウザなら Quarkus→Keycloak へリダイレクトしてログイン → /mashup 表示
```

> Kong は**素通し**なので、認証・セッション管理は Quarkus が担当。 将来、Kong 側で WAF/RateLimit や API キーなどの“入口制御”も足せます。

---

9. よくあるハマり & TIPS

- **roles.source の値** `accesstoken` 以外（`access_token`/`access-token`）は**起動エラー**になります。
- **ロールが見つからない** `role-claim-path` のキー（`"service-a"` / `"service-b"`）が正しいか再確認。

- **aud が足りない** Audience マッパーで `service-a` / `service-b` を追加したか確認。 `echo $TOKEN | cut -d . -f2 | tr '_-' '/+' | base64 -d | jq .aud`
- **トークン期限** 403/401 やログに “The JWT is no longer valid” が出たら再取得。
- **var がコンパイル不能** JDK 17 を使う（pom.xml で `<maven.compiler.release>17</maven.compiler.release>`）。 それでもダメなら `var` を明示型に置き換え。
- **セッションクッキーが大きい警告**（集約側） 開発中は無視でもOK。気になる場合は：

  ```
  quarkus.oidc.token-state-manager.split-tokens=true
  ```

  （あるいは `strategy=id-refresh-tokens` で Access Token をクッキーに含めない運用に変更）

以下で、接続が成功する場合もあります。

```
docker compose down
docker compose up -d
```

---

10. 期待する“多段認可”の流れ（再掲）

1. **集約API**：Keycloak でユーザー認証（OIDC）
2. **集約API内認可**：そのユーザーが A/B を呼ぶ要件を満たすか（必要なら `@RolesAllowed` 等で）
3. **トークン伝搬**：ユーザーの Access Token を `Authorization: Bearer` で A/B に転送
4. **下流 A/B 認可**：それぞれ `roles.source=accesstoken` + `role-claim-path` で**自前のポリシー**
5. **結果マージ**：`{ fromServiceA, fromServiceB }` を統合して返却

以上です。