できます！やり方は2通りありますが、ここではご要望どおり **Redis にサーバーサイド保存**するパターン（“クッキーには参照用キーだけ入れる”）の**サンプル実装**を示します。Quarkus には拡張ポイントがあり、`io.quarkus.oidc.TokenStateManager` を自作 CDI Bean で差し替えるだけで実現できます。

> ポイント
>
> - **セッションの実体（ID/Access/Refresh トークン等）を Redis に保存**
> - **ブラウザのクッキーには短いキー（例: `sid`）のみ**を入れる
> - 既存の “state クッキー” とは**別管理**（= クッキー肥大化の回避 & サーバーサイド失効が容易）

1) 依存関係の追加（quarkus-authz モジュール）

`quarkus-authz/pom.xml` に Redis クライアントを追加：

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-redis-client</artifactId>
</dependency>
```

> すでに Quarkus BOM を入れているので version 指定は不要です。

2) 接続設定（quarkus-authz）

`quarkus-authz/src/main/resources/application.properties` に追記：

```
# Redis 接続（Compose のサービス名に合わせる）
quarkus.redis.hosts=redis://redis:6379

# （任意）トークンをクッキーに入れない前提なので暗号化は不要
# ＝クッキーには参照キーのみ。ネットワークが安全でないなら true に戻してください
quarkus.oidc.token-state-manager.encryption-required=false
```

> すでに `REDIS_HOST=redis` を渡しているなら環境変数展開でもOKです
> （`quarkus.redis.hosts=${REDIS_URL:redis://redis:6379}` など）。

3) 実装：Redis バックエンドの TokenStateManager

`quarkus-authz/src/main/java/org/example/security/RedisTokenStateManager.java`（新規）：

```java
package org.example.security;

import java.time.Duration;
import java.util.Map;
import java.util.UUID;

import jakarta.annotation.Priority;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Alternative;
import jakarta.inject.Inject;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;

import io.quarkus.oidc.TokenStateManager;
import io.quarkus.oidc.Tokens;
import io.quarkus.redis.datasource.RedisDataSource;
import io.quarkus.redis.datasource.string.SetArgs;
import io.quarkus.redis.datasource.string.StringCommands;

/**
 * OIDC のセッション（ID/Access/Refresh トークン）を Redis に保存し、
 * クッキーには参照キー（state id）のみを入れる実装。
 *
 * 注意：Quarkus 3.11.1 時点の TokenStateManager のメソッド名/引数に
 *       若干の差分がある場合は、IDE の補完に合わせて微調整してください。
 */
@Alternative
@Priority(1) // ← 組み込み実装より優先させる
@ApplicationScoped
public class RedisTokenStateManager implements TokenStateManager {

    private static final String KEY_PREFIX = "q:oidc:sess:";
    private static final Duration DEFAULT_TTL = Duration.ofMinutes(30); // 適宜

    private final StringCommands<String, String> str;
    private final ObjectMapper mapper = new ObjectMapper();

    @Inject
    public RedisTokenStateManager(RedisDataSource ds) {
        this.str = ds.string(String.class);
    }

    /**
     *「トークンをクッキーに詰める代わりに」Redis に保存し、
     * クッキーにはこのメソッドの戻り値（＝stateId）だけを入れてもらう。
     */
    @Override
    public String createTokenState(Tokens tokens) {
        // 参照キー（state id）を生成
        String stateId = UUID.randomUUID().toString();

        // 保存する中身（必要に応じて項目追加）
```

```
        Map<String, Object> payload = Map.of(
            "id_token", tokens.getIdToken(),
            "access_token", tokens.getAccessToken(),
            "refresh_token", tokens.getRefreshToken(),
            "expires_at", tokens.getExpiresIn() != null ?
System.currentTimeMillis() + tokens.getExpiresIn() * 1000L : null
        );

        try {
            String json = mapper.writeValueAsString(payload);
            str.set(KEY_PREFIX + stateId, json, new SetArgs().ex(DEFAULT_TTL));
        } catch (Exception e) {
            throw new IllegalStateException("Failed to write token state to
Redis", e);
        }
        // ← クッキーにはこの stateId が保存され、以降の復元で使われる
        return stateId;
    }

    /**
     * クッキーに入っていた stateId を使って Redis からトークンを復元。
     */
    @Override
    public Tokens getTokens(String tokenState) {
        try {
            String json = str.get(KEY_PREFIX + tokenState);
            if (json == null) {
                return null; // 期限切れ or 無効
            }
            Map<String, Object> payload = mapper.readValue(json, new
TypeReference<Map<String, Object>>() {});
            String id = (String) payload.get("id_token");
            String at = (String) payload.get("access_token");
            String rt = (String) payload.get("refresh_token");

            // TTL を延長したい場合はここで touch
            str.expire(KEY_PREFIX + tokenState, (int) DEFAULT_TTL.getSeconds());

            return new Tokens(id, at, rt);
        } catch (Exception e) {
            throw new IllegalStateException("Failed to read token state from
Redis", e);
        }
    }

    /**
     * ログアウト等のタイミングでサーバーサイドのセッションを消す。
     */
    @Override
    public void removeTokens(String tokenState) {
        str.del(KEY_PREFIX + tokenState);
    }

    /**
```

```
     * トークンを更新した際（リフレッシュ）に呼ばれる想定。
     * 実装が不要な場合は createTokenState を再利用しても良い。
     */
    @Override
    public String updateTokenState(String tokenState, Tokens newTokens) {
        // 既存キーを上書き（stateId は据え置き）
        try {
            Map<String, Object> payload = Map.of(
                "id_token", newTokens.getIdToken(),
                "access_token", newTokens.getAccessToken(),
                "refresh_token", newTokens.getRefreshToken(),
                "expires_at", newTokens.getExpiresIn() != null ?
System.currentTimeMillis() + newTokens.getExpiresIn() * 1000L : null
            );
            String json = mapper.writeValueAsString(payload);
            str.set(KEY_PREFIX + tokenState, json, new SetArgs().ex(DEFAULT_TTL));
            return tokenState;
        } catch (Exception e) {
            throw new IllegalStateException("Failed to update token state in
Redis", e);
        }
    }
}
```

> **注**：`TokenStateManager` のメソッド名/シグネチャは Quarkus のバージョンで微妙に変わることがあります。IDE の補完に従って上記のメソッド名（`createTokenState/getTokens/removeTokens/updateTokenState` 等）を合わせてください。 ログに出ていた「custom TokenStateManager を登録」というメッセージ通り、`@Alternative @Priority(1)` が肝です。

---

4) 使い方

- 実装を追加したら `quarkus-authz` を **再ビルド → 再起動**：

```
cd quarkus-authz && mvn clean package && cd ..
docker compose up -d --build quarkus-authz
```

- 初回アクセスでコードフロー完了後、**クッキーには `stateId` だけ**入り、 トークンの実体は `redis` **コンテナ**の中に `q:oidc:sess:<uuid>` で保存されます。

---

5) 失効・ログアウトの扱い

- サインアウト時に `removeTokens(stateId)` が呼ばれ、Redis のキーを削除 → **即時失効**できます。
- **強制ログアウト**（管理側）を作りたい場合は、Redis の該当キーを消すだけでも OK。 次回アクセス時に復元できず 401→再ログイン、という流れになります。

6) 動作確認のヒント

- Redis に入ったかを確認（コンテナ内）：

```
docker exec -it 3-redis-1 redis-cli
keys q:oidc:sess:*
ttl q:oidc:sess:<stateId>
get q:oidc:sess:<stateId>  # ← JSON が見えます（本番では見えない運用が望ましい）
```

- クッキーサイズの警告が消える（もしくは大幅に減る）はずです。

7) 代替案（参考）

- すぐ使える公式ストア：`quarkus-oidc-db-token-state-manager`（JDBC） → Postgres などに保存。Redis ではなく DB で良ければ最短です。
- 入口で 302 を避けたい：`/mashup` を Bearer 専用テナントだけに割り当て、401→フロントで `/authorize` に誘導する方式も可能。

必要なら、この Redis 実装を **抽象化（マルチテナント対応・JSON スキーマ厳格化・暗号化）**してもう少し堅牢にする版も用意できます。ログアウト API、TTL ルール、キー回転ポリシーなどもあわせて整備できますよ。