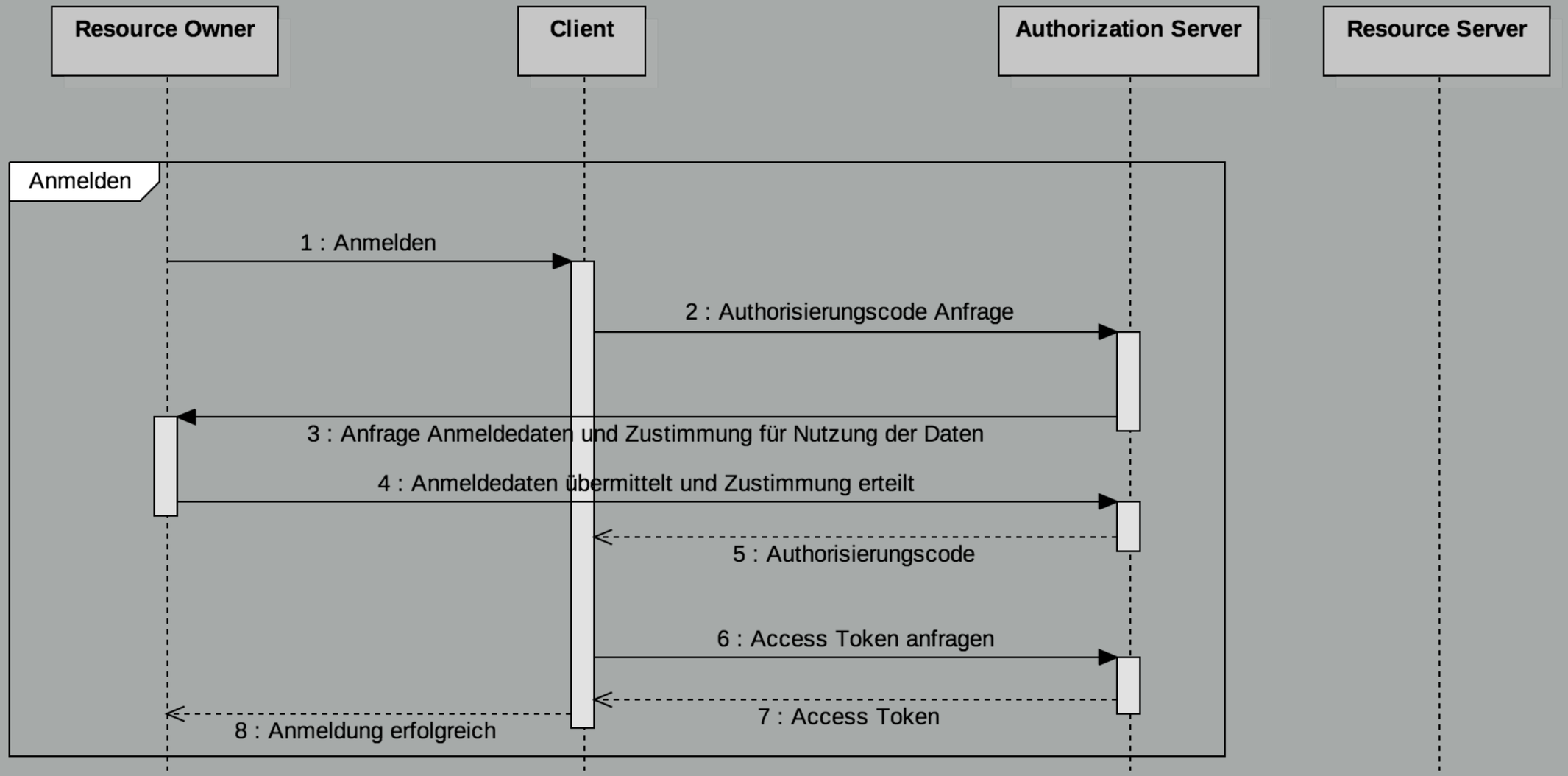
CHRISTIAN KELLER

CLOUD NATIVE APPS SICHERN MIT OAUTH 2/OPENID CONNECT

GLIEDERUNG

- OAuth 2.0 - Flows
  - Authorization Code Grant
  - Implicit Grant
  - Ressource Owner Credentials Grant
  - Client Credentials Grant
- OpenID Connect
- Vergleich
- Beispiel-Implementierung

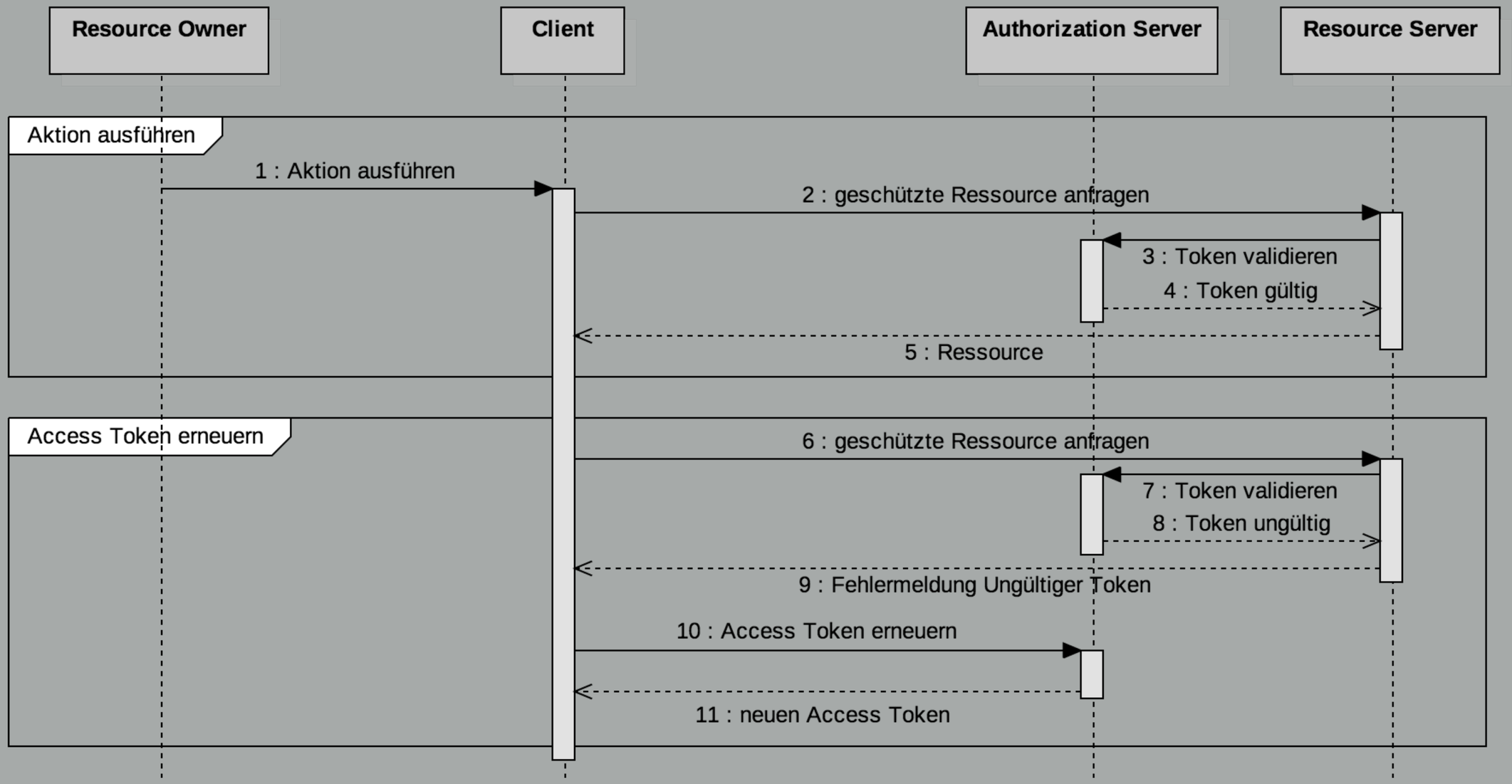
OAUTH 2.0 – AUTHORIZATION CODE GRANT FLOW



EIGENSCHAFTEN

- Client verarbeitet nie Zugangsdaten (Benutzername, Passwort)
- Token als Schlüssel für Zugang nur Client bekannt
- Viele Nachrichten notwendig
- Anbieter kann Zwei-Faktor-Authentifizierung verwenden
- Beispiel: Google Playground

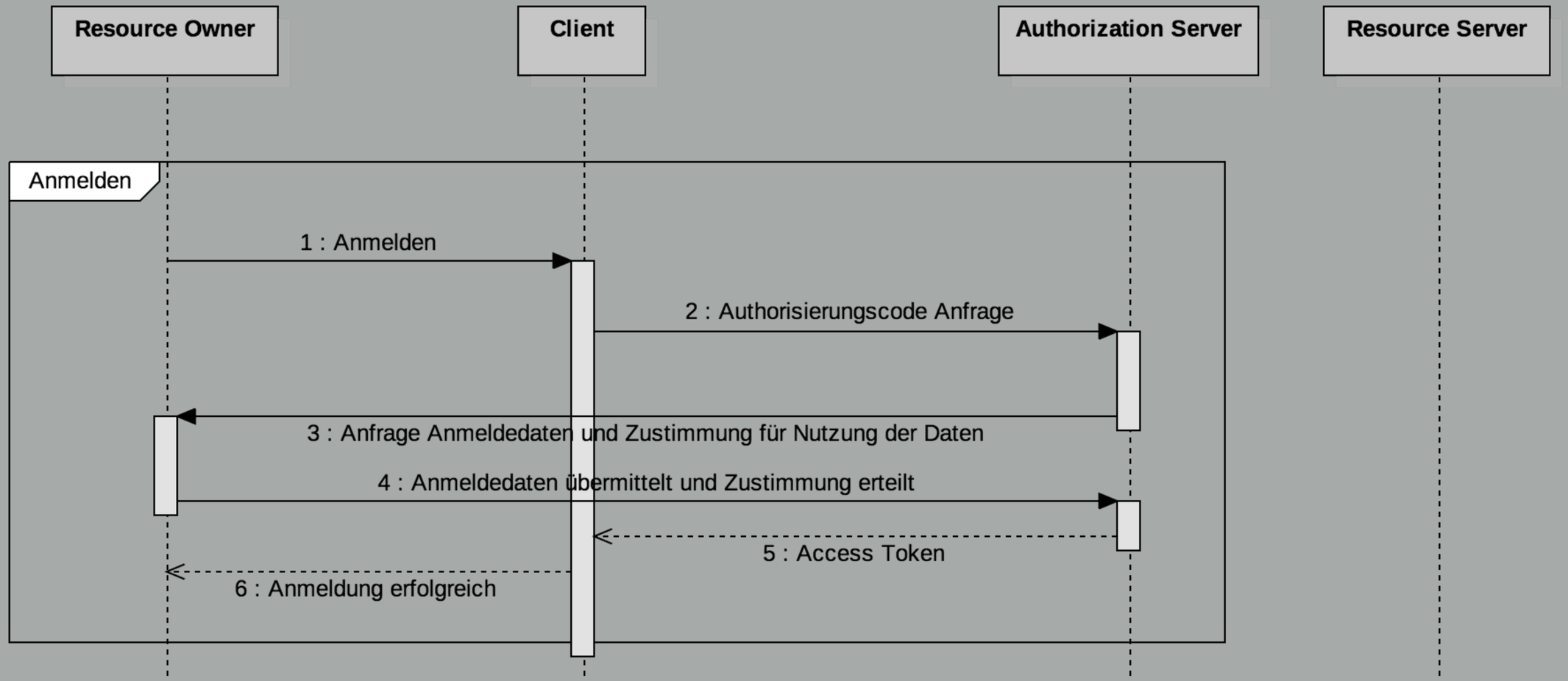
OAUTH 2.0 – RESSOURCENZUGRIFF



EIGENSCHAFTEN

- Token kann validiert werden
- Token enthält keine Benutzerinformationen
- Token kann jederzeit für ungültig erklärt werden
- Refresh Token als Mechanismus um Token zu erneuern

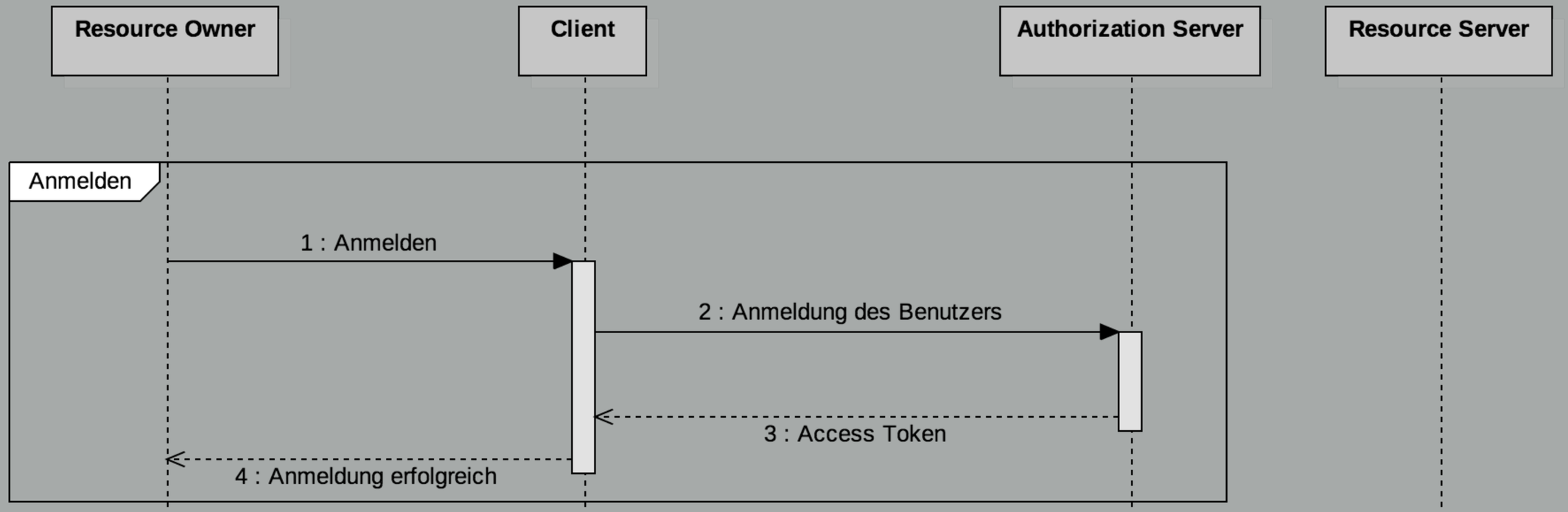
OAUTH 2.0 – IMPLICIT GRANT FLOW



EIGENSCHAFTEN

- Optimiert für Webanwendungen
- Daten in der URL
- Unsicherer als „Authorization Code Grant Flow“
  - Gefahr: Austausch der Zugangsdaten mit Dritten
- Keine Aktualisierung des Token möglich

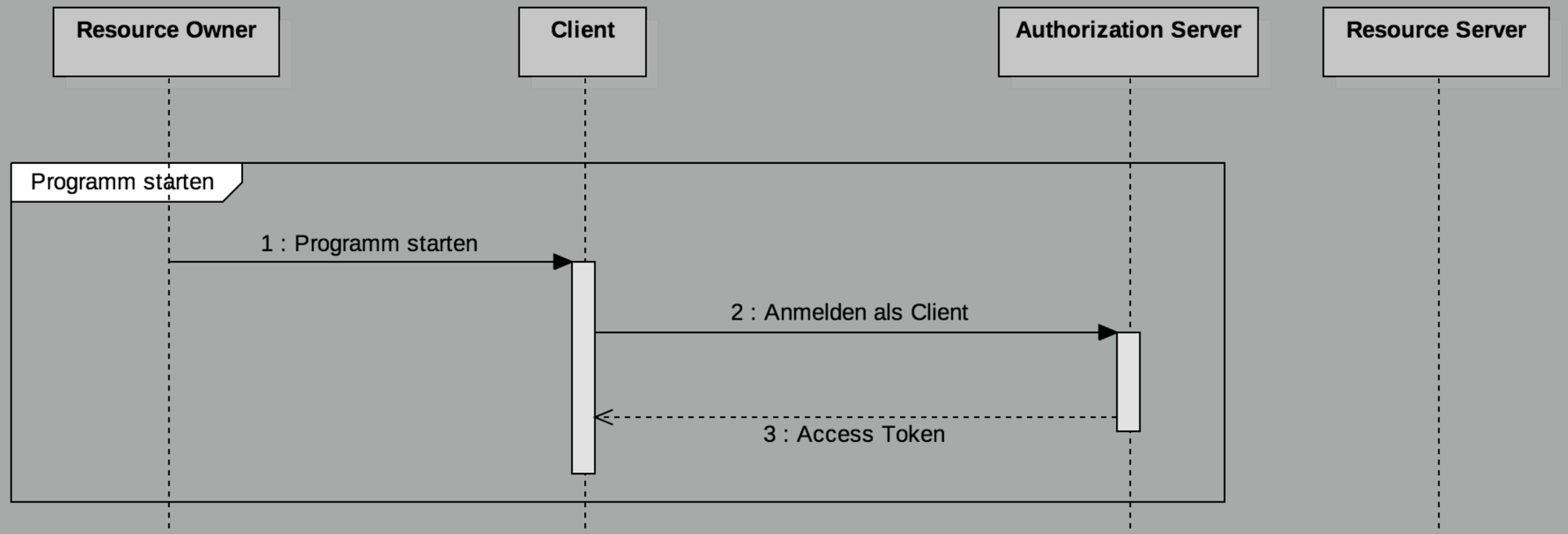
OAUTH 2.0 – RESOURCE OWNER CREDENTIALS GRANT FLOW



EIGENSCHAFTEN

- Analoge Umsetzung zu HTTP-Authentifizierung (Basic Auth)
- Client kennt Benutzerdaten (ist angehalten diese nach dem Anmeldeprozess zu verwerfen)

OAUTH 2.0 – CLIENT CREDENTIALS GRANT FLOW



EIGENSCHAFTEN

- Authentifiziert wird die Anwendung selbst
- Nutzer hat meist keine Kenntnis

OPENID CONNECT

- Erweiterung zu OAuth 2.0
  - Zusätzliche Funktionalität durch ID Token
- Verwendet andere Begriffe
  - Client → Relaying Party
  - Authorization Server → Identity Provider
- Spezifikation beschreibt nur „Authorization Code Grant Flow“ und „Implicit Grant Flow“

ID TOKEN (JSON WEB TOKEN – JWT)

eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQ
ogImh0dHA6Ly9leGFt cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
.dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk

Base64 codiert

{"typ":"JWT",
 "alg":"HS256"}
{"iss":"joe",
 "exp":1300819380,
 "http://example.com/is_root":true}
AIyTb[P5Z[XXSy

Schlüssel	Beschreibung
iss	Aussteller des ID Tokens
exp	Ablaufzeitpunkt in s
name	Vollständiger Name
profile	URL zu Profilseite
picture	URL zu Benutzerbild
email	E-Mail Adresse

VERGLEICH

	OAuth 2.0	OpenID Connect
Authorization Code Grant	Ja	Ja
Implicit Grant	Ja	Ja
Resource Owner Credential Grant	Ja	Nein
Client Credentials Grant	Ja	Nein
Zusatzinformation mittels JWT	Nein	Ja
UserInfo-Endpunkt	Nein	Ja

VERGLEICH – TOKENBASIERTE VERFAHREN VS. HTTP BASIC AUTHENTICATION

- Basieren beide auf SSL/TLS Verschlüsselung der Nachricht
- Client speichert bei HTTP Basic Authentication Benutzerdaten während tokenbasierte Verfahren nur Token speichern

VERGLEICH – TOKENBASIERTE VERFAHREN VS. FORM-BASED LOGIN

- Form-Based Login ist nicht genauer spezifiziert. Allgemein Login über ein HTML-Formular.
- Abhängig gegenüber Client
- Tokenbasierte Verfahren sind unabhängig vom Client

BEISPIEL IMPLEMENTIERUNG

Github: https://github.com/keller-c/cnj-backend

FAZIT

- OAuth 2.0 - sicheres Framework für eine flexible Authentifizierung und Autorisierung
- OpenID Connect - Erweiterung um ID Token ermöglicht weitere Informationen über den Nutzer
- Stand der Technik und weit verbreitet
- Leicht in der Anwendung mit Spring -> Fokus auf Anwendungsentwicklung

QUELLEN

- OAuth 2.0 Spezifikation
  https://tools.ietf.org/html/rfc6749
- JSON Web Token (JWT)
  https://tools.ietf.org/html/rfc7519#section-3.1
- OpenID Connect – Core 1.0 Spezifikation
  http://openid.net/specs/openid-connect-core-1_0.html

VIELEN DANK FÜR EURE AUFMERKSAMKEIT!

Noch Fragen?