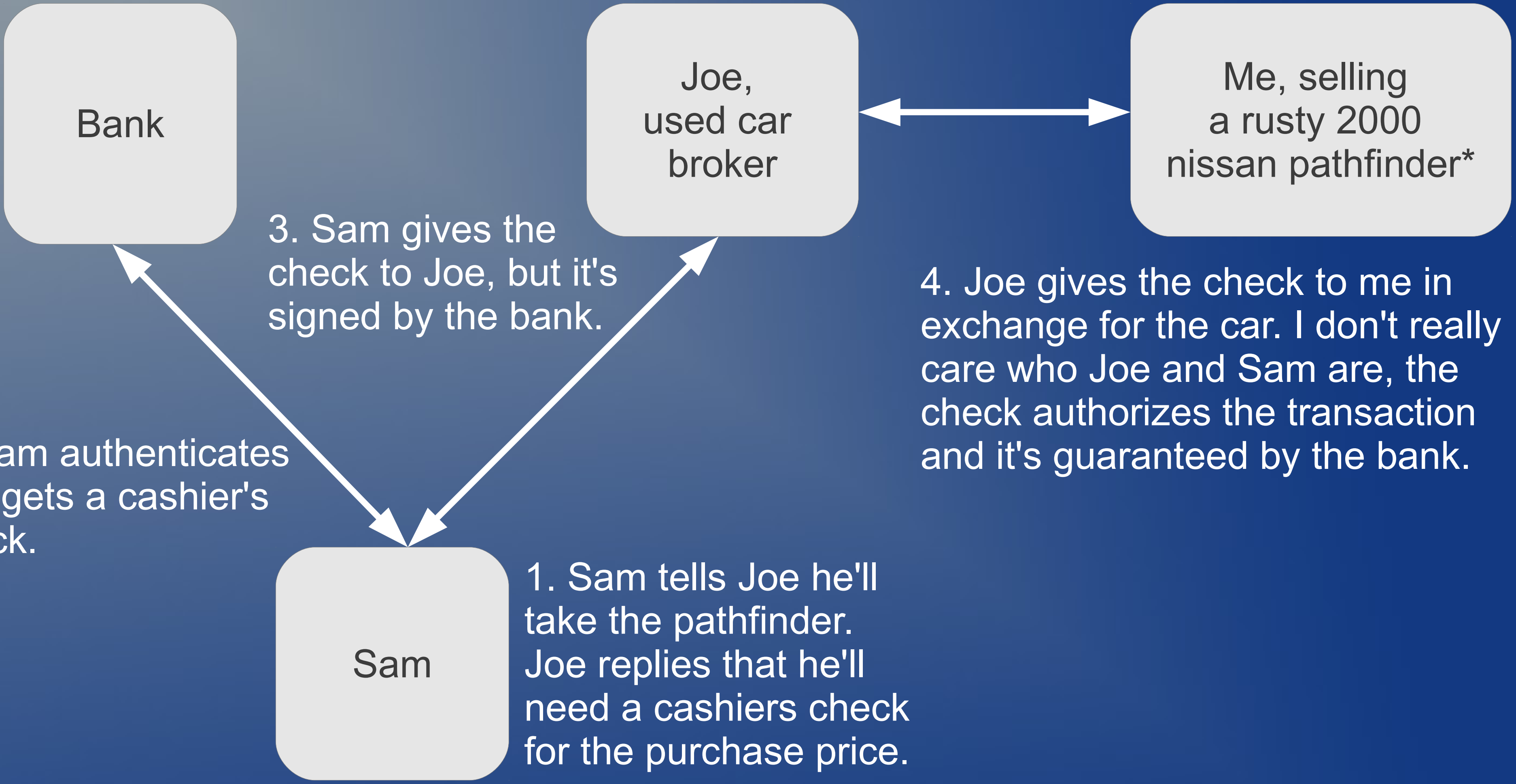
UAA Concepts and Overview

25 Feb 2013

introduction

- too many passwords are making us crazy: http://www.youtube.com/watch?v=2tJ-NSPES9Y
- how many passwords vs bank/credit accounts do you have and why?
- password managers are single point of failure: http://www.miamiherald.com/2012/09/13/3000801/miami-federal-jury-convicts-man.html
- more importantly, we need to control authorization not just reduce passwords: http://www.schneier.com/blog/archives/2008/07/disgruntled_emp.html
- the uaa aims to conveniently manage proof of authorization: http://xkcd.com/149/

illustration: selling a car



3. Sam gives the check to Joe, but it's signed by the bank.

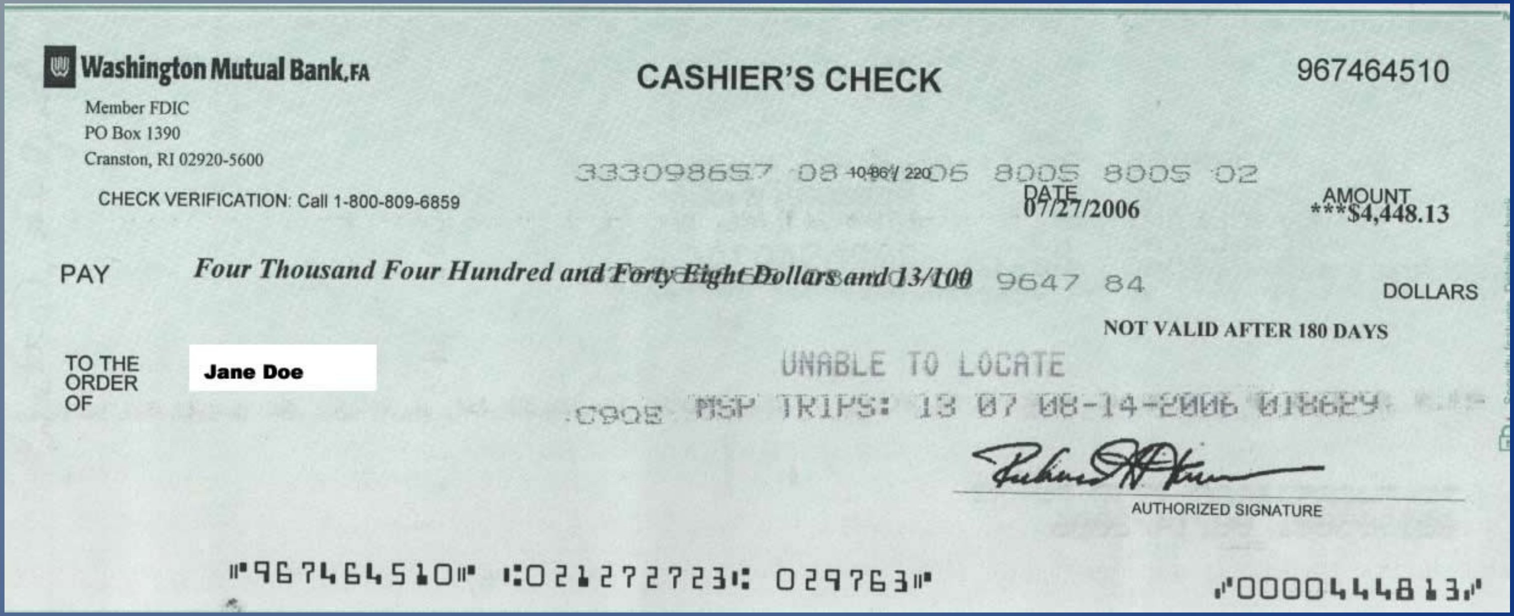
4. Joe gives the check to me in exchange for the car. I don't really care who Joe and Sam are, the check authorizes the transaction and it's guaranteed by the bank.

2. Sam authenticates and gets a cashier's check.

1. Sam tells Joe he'll take the pathfinder. Joe replies that he'll need a cashiers check for the purchase price.

* the pathfinder is actually for sale.

cashier's check



Washington Mutual Bank, FA
Member FDIC
PO Box 1390
Cranston, RI 02920-5600

CHECK VERIFICATION: Call 1-800-809-6859

CASHIER'S CHECK

967464510

333098657 08 10-86/220 2006 8005 8005 02

DATE 07/27/2006

AMOUNT ***$4,448.13

PAY *Four Thousand Four Hundred and Forty Eight Dollars and 13/100* 9647 84 DOLLARS

NOT VALID AFTER 180 DAYS

TO THE ORDER OF **Jane Doe**

UNABLE TO LOCATE

C90E MSP TRIPS: 13 07 08-14-2006 018629

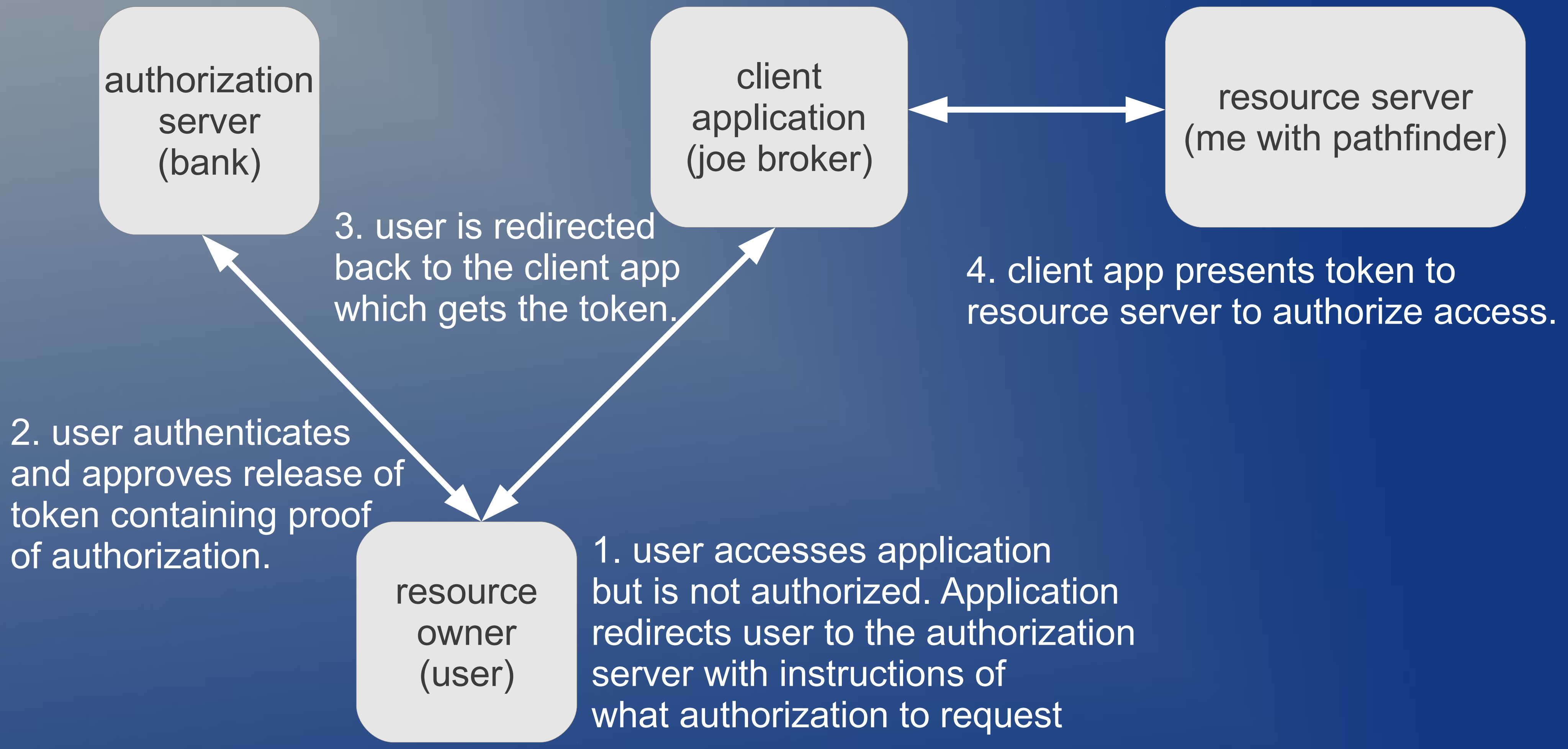
[illegible]

AUTHORIZED SIGNATURE

⑈967464510⑈ ⑆021272723⑆ 029763⑈ ⑈0000444813⑈

- issuing bank
- check number
- account and name
- authorized amount
- issued at date
- expiration date
- recipient
- authorized signature

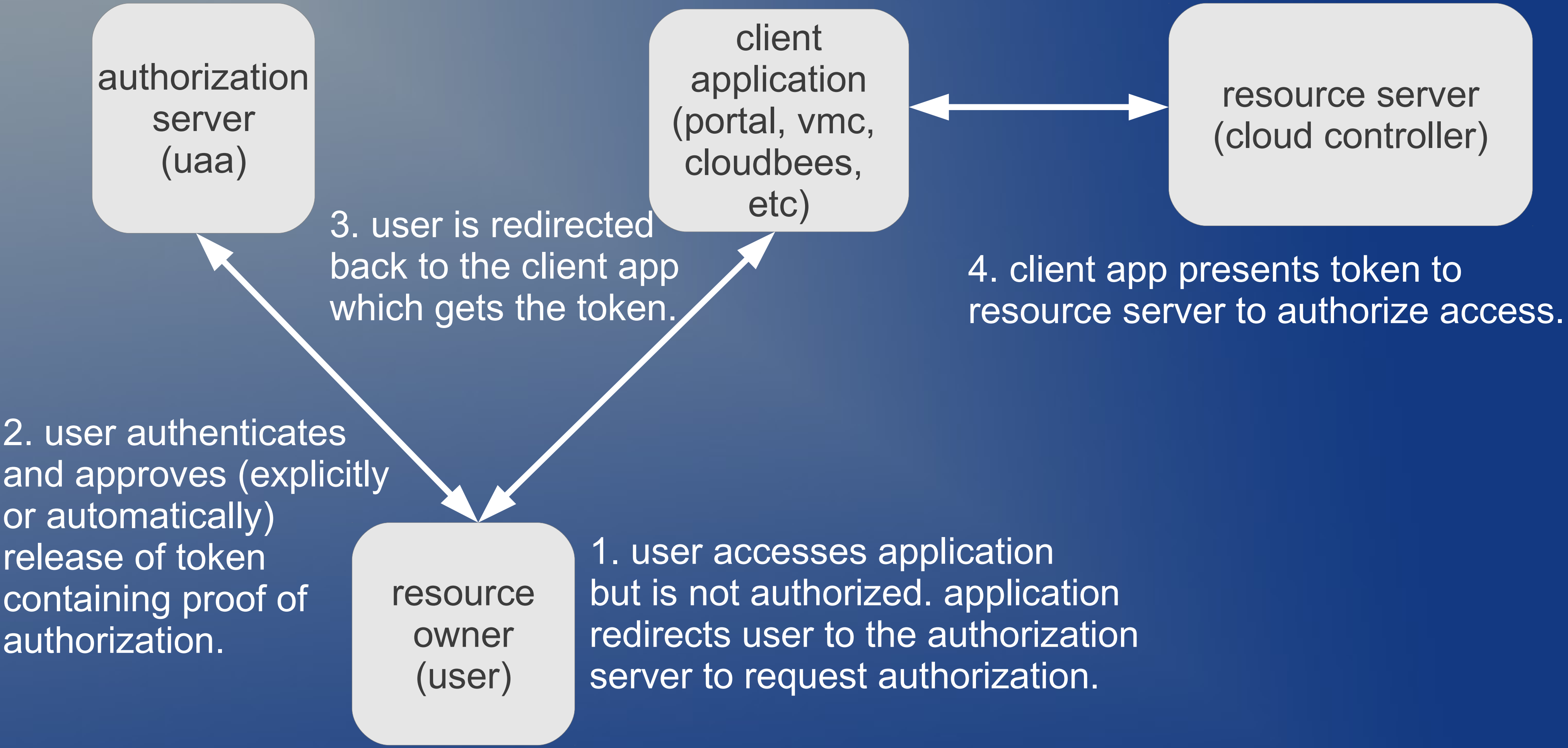
illustration in oauth2 terms


authorization server (bank)
client application (joe broker)
resource server (me with pathfinder)
resource owner (user)
1. user accesses application but is not authorized. Application redirects user to the authorization server with instructions of what authorization to request
2. user authenticates and approves release of token containing proof of authorization.
3. user is redirected back to the client app which gets the token.
4. client app presents token to resource server to authorize access.

json web token – jwt (uaa cashier's check)

- jwt is an ietf standard, pronounced jot
- base64 encoded parts separated by periods
- header.content.signature
- header indicates signature algorithm
- contents:
  - iss: issuer
  - jti: token id
  - sub: user id
  - scope: authorization
  - iat: issued at
  - exp: expiration
  - aud: audience
  - others: user name, email, client id

illustration in cloud foundry terms



authorization server (uaa)
client application (portal, vmc, cloudbees, etc)
resource server (cloud controller)
resource owner (user)

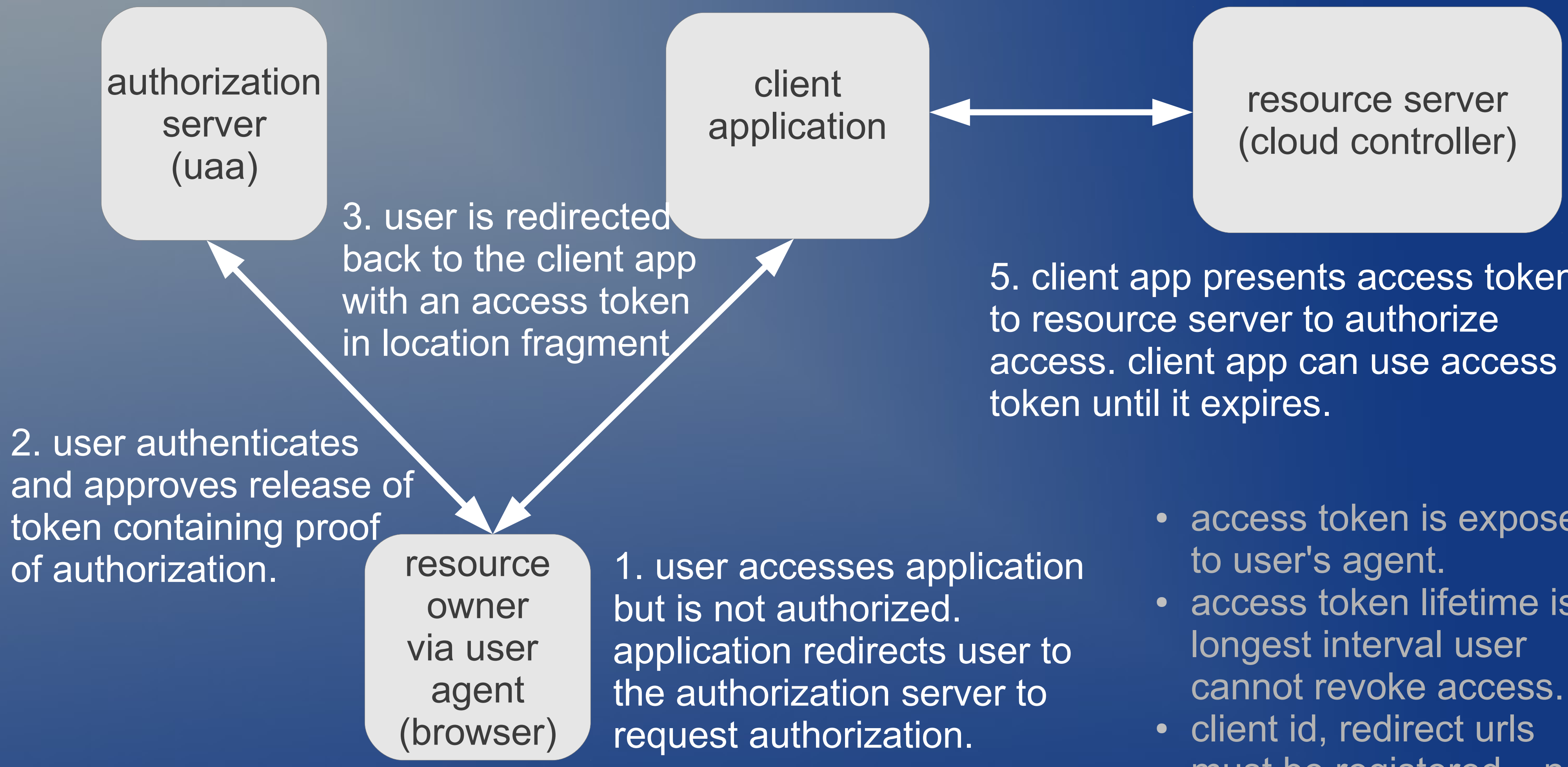
1. user accesses application but is not authorized. application redirects user to the authorization server to request authorization.

2. user authenticates and approves (explicitly or automatically) release of token containing proof of authorization.

3. user is redirected back to the client app which gets the token.

4. client app presents token to resource server to authorize access.

implicit grant



authorization server (uaa)
client application
resource server (cloud controller)
resource owner via user agent (browser)

1. user accesses application but is not authorized. application redirects user to the authorization server to request authorization.

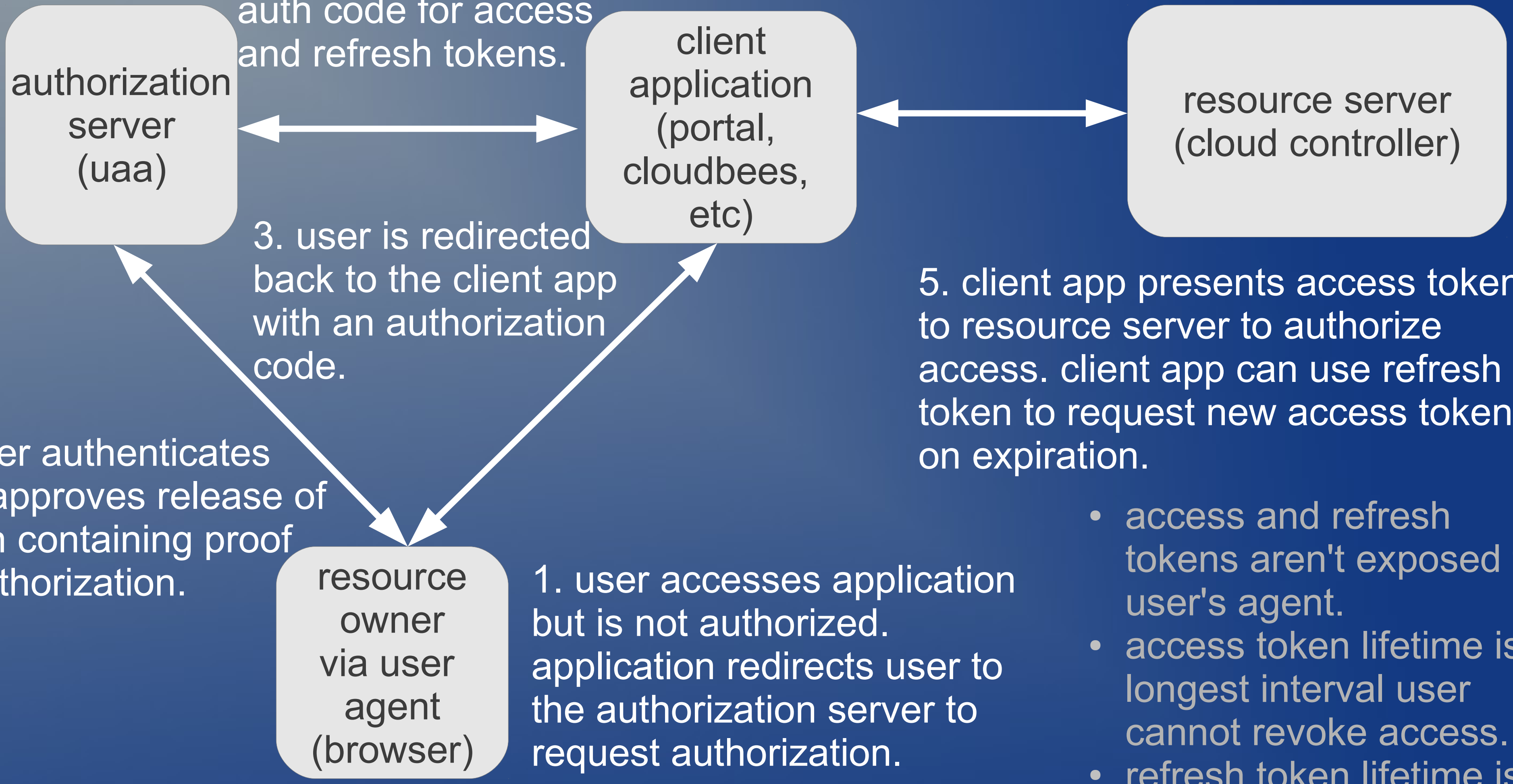
2. user authenticates and approves release of token containing proof of authorization.

3. user is redirected back to the client app with an access token in location fragment

5. client app presents access token to resource server to authorize access. client app can use access token until it expires.

- access token is exposed to user's agent.
- access token lifetime is longest interval user cannot revoke access.
- client id, redirect urls must be registered – no secret
- generally a less secure and less convenient grant – avoid if possible

authorization code grant (and refresh token grant)


authorization server (uaa)
client application (portal, cloudbees, etc)
resource server (cloud controller)
resource owner via user agent (browser)

4. client redeems auth code for access and refresh tokens.

3. user is redirected back to the client app with an authorization code.

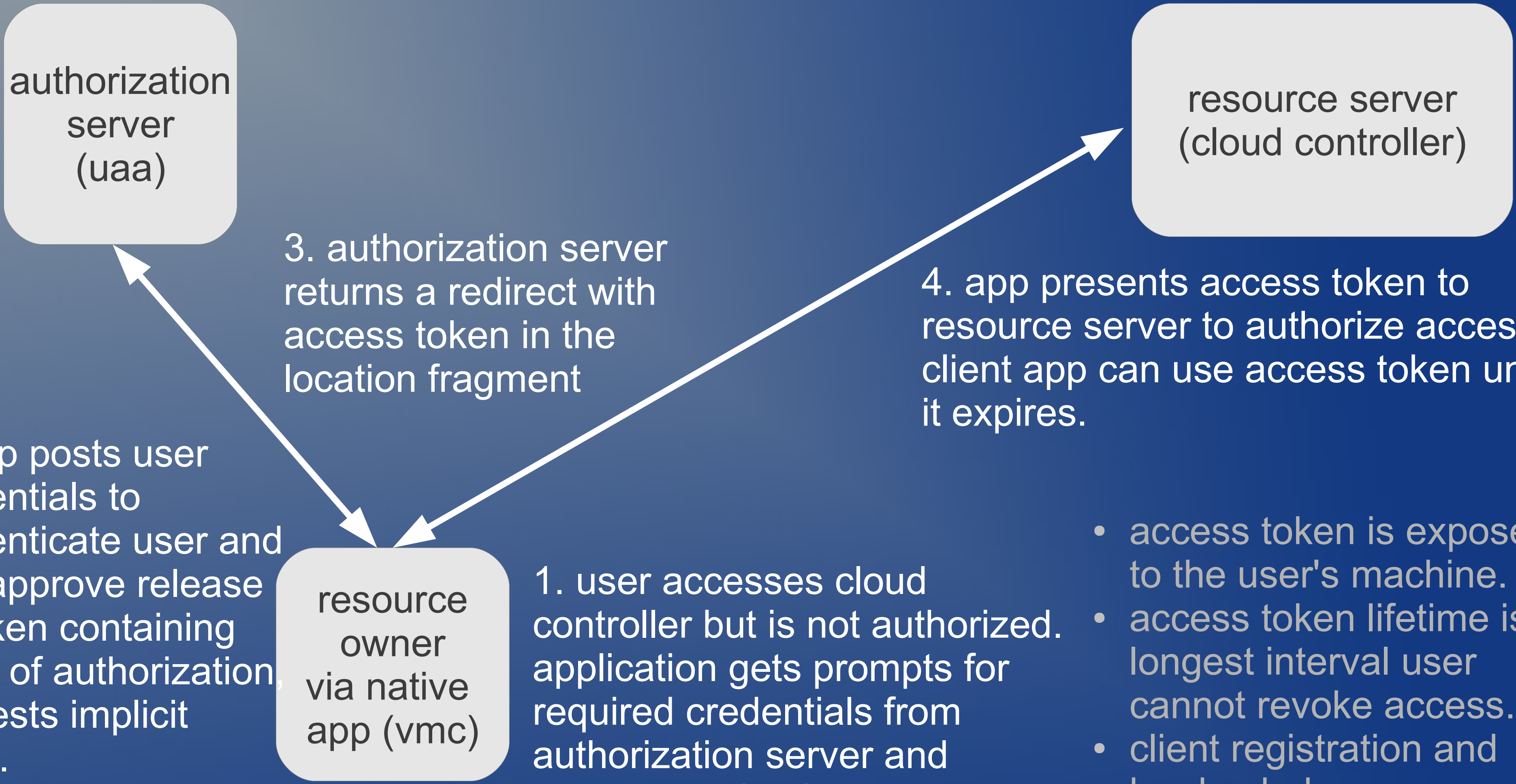
2. user authenticates and approves release of token containing proof of authorization.

1. user accesses application but is not authorized. application redirects user to the authorization server to request authorization.

5. client app presents access token to resource server to authorize access. client app can use refresh token to request new access token on expiration.

- access and refresh tokens aren't exposed to user's agent.
- access token lifetime is longest interval user cannot revoke access.
- refresh token lifetime is interval before user has to re-authenticate.
- client id, secret, redirect urls must be registered

implicit grant with credentials (deprecated)


authorization server (uaa)
resource server (cloud controller)
resource owner via native app (vmc)

1. user accesses cloud controller but is not authorized. application gets prompts for required credentials from authorization server and collects credentials

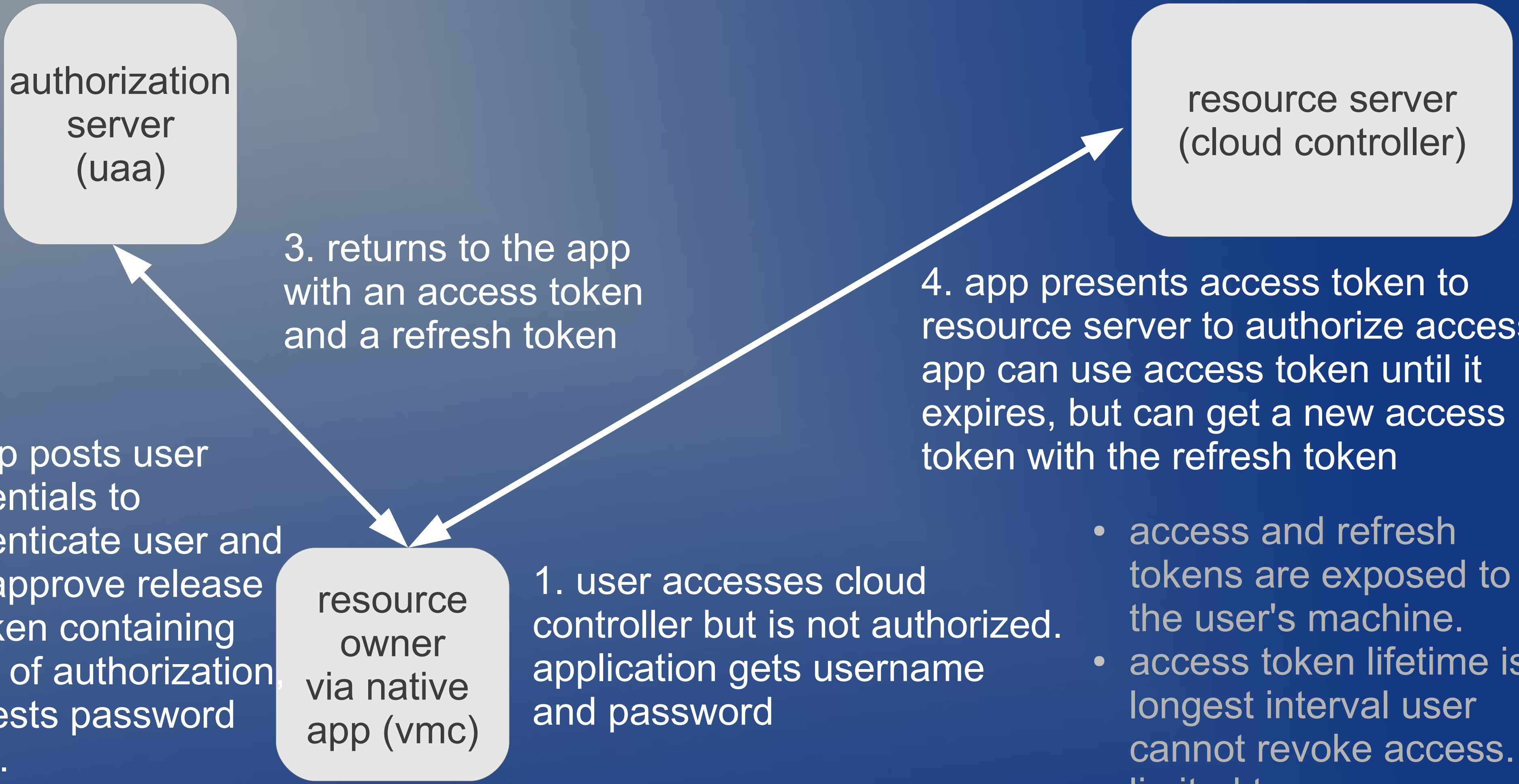
2. app posts user credentials to authenticate user and autoapprove release of token containing proof of authorization, requests implicit grant.

3. authorization server returns a redirect with access token in the location fragment

4. app presents access token to resource server to authorize access. client app can use access token until it expires.

- access token is exposed to the user's machine.
- access token lifetime is longest interval user cannot revoke access.
- client registration and hardcoded, unnecessary redirect urls were problem
- our extension to oauth that is now deprecated.

resource owner password grant (& refresh token grant)


authorization server (uaa)
resource server (cloud controller)
resource owner via native app (vmc)

1. user accesses cloud controller but is not authorized. application gets username and password

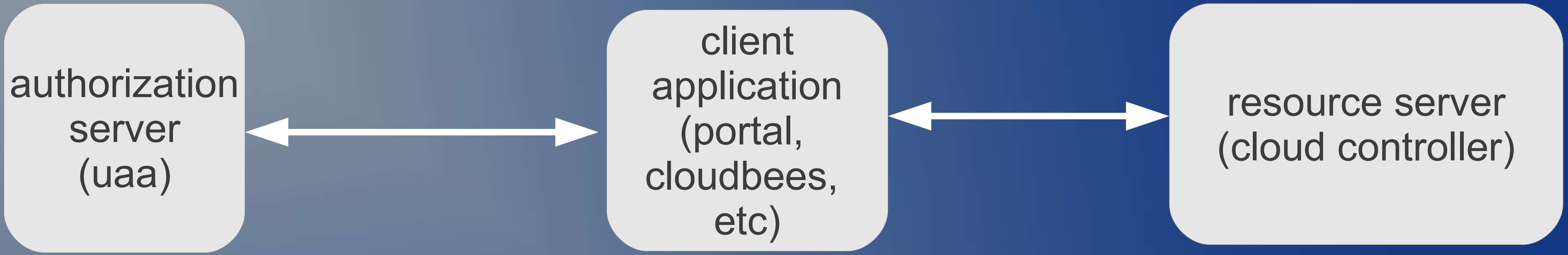
2. app posts user credentials to authenticate user and autoapprove release of token containing proof of authorization, requests password grant.

3. returns to the app with an access token and a refresh token

4. app presents access token to resource server to authorize access. app can use access token until it expires, but can get a new access token with the refresh token

- access and refresh tokens are exposed to the user's machine.
- access token lifetime is longest interval user cannot revoke access.
- limited to user name and password.
- for vmc, requires public client (no client secret).

client credentials grant


authorization server (uaa)
client application (portal, cloudbees, etc)
resource server (cloud controller)

1. client authenticates and gets an access token with all its registered authorizations.

2. client app presents access token to resource server to authorize access.

- access token lifetime is longest interval uaa cannot revoke access.
- client id, secret must be registered

client registration contents

- client id: name, e.g. vmc or portal
- client secret: can be empty for a public client
- authorized grant types
- authorities: authority as the client, token scope for client credentials grant – think 'client scope'
- scope: authority that can be requested for a user via all other grant types – think 'maximum user scope'
- redirect uris
- access token validity
- refresh token validity
- autoapprove scope

calculating client token scope

- authenticate client id and secret via basic auth
- validate request is client credentials grant
- read client authorities
- put it in the scope field of the token

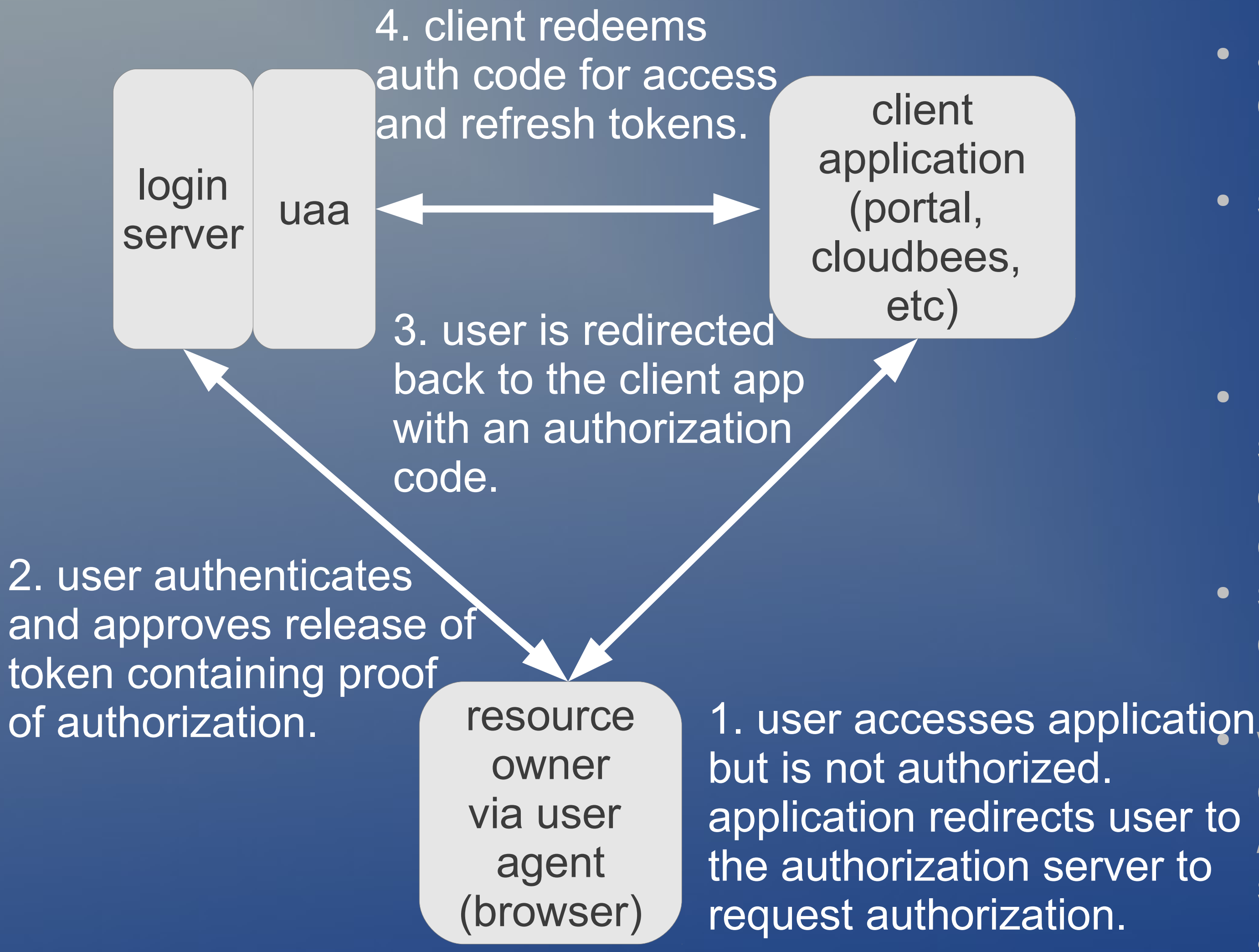
calculating user token scope

- if no scope is requested, the default is the scope registered for the requesting client
- for each scope: if the user is in the group and the scope is either autoapprove or the user explicitly approves it, put it in the token.

calculating user token scope: details

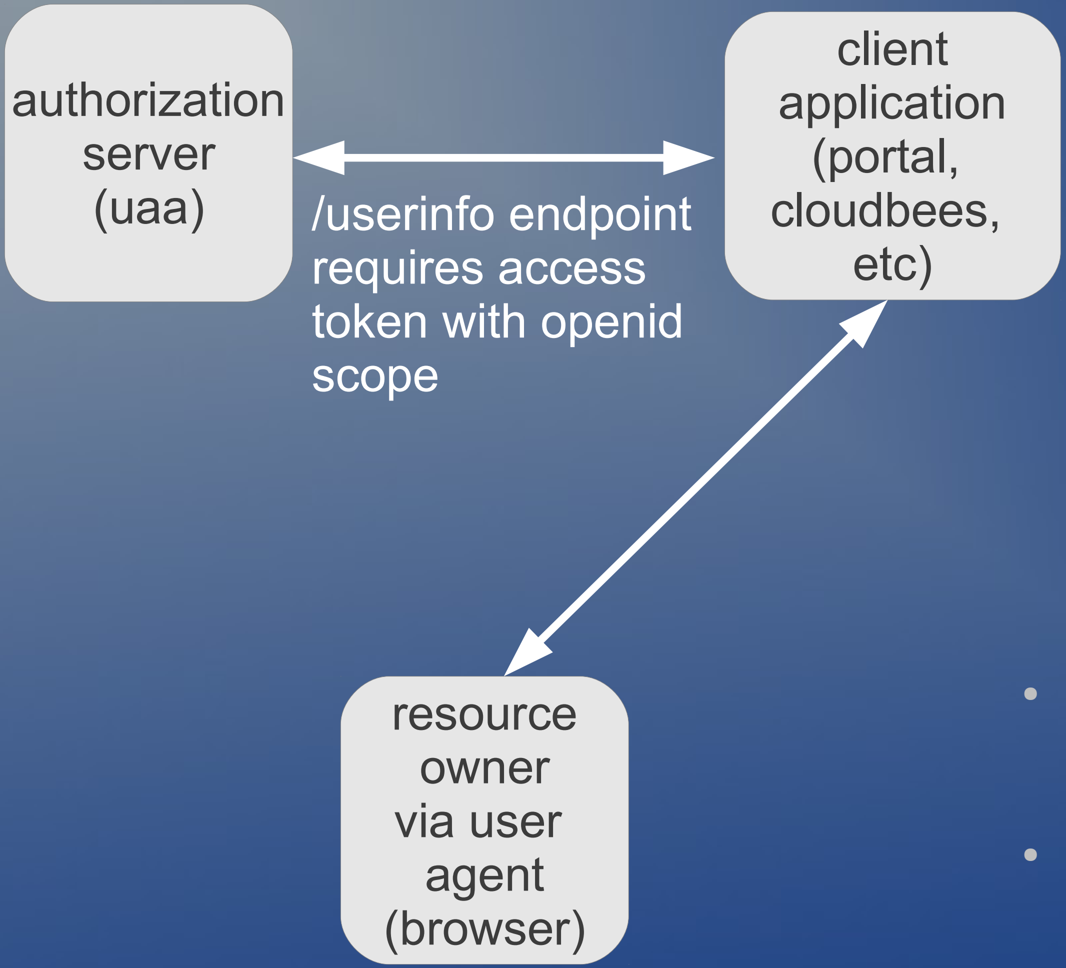
- given a list of scopes, a user, and a client
- return error if the list of scopes includes a scope not in the client registration
- if the list is empty, fill it with the scopes from the client registration
- remove all scopes for which the user is not a member of the corresponding group – this is the possiblescopes list.
- make a list of all remaining scopes that are not autoapproved (approvable)
- get the stored approved and denied lists for this user and client
- make a list of all scopes from the approvable list that are not on the approved or denied lists (unapproved)
- if it's a refresh token and unapproved list is not empty, return error (the client must send the user through a new grant).
- ask the user for approvals for all approved or denied scopes (with appropriate defaults), AND the unapproved scopes (as "new").
- make a list of the possiblescopes that are now approved or autoapproved -- that's what goes in the token.

extensible authentication with a login server


login server
uaa
client application (portal, cloudbees, etc)
resource owner via user agent (browser)
4. client redeems auth code for access and refresh tokens.
3. user is redirected back to the client app with an authorization code.
2. user authenticates and approves release of token containing proof of authorization.
1. user accesses application but is not authorized. application redirects user to the authorization server to request authorization.

- allows authentication to be other than user name and password in uaa db.
- supports graphics and branding for cloudfoundry rather than generic interface in uaa
- login server may support saml, ldap, openid2 (google), oauth 1.1a (twitter), facebook, etc.
- some login servers proxy the oauth2 token endpoint to the uaa.
- we are working on making endpoints available from the /info endpoint on the login server: oauth2 token & authorize, scim users & groups, openid connect userinfo, etc.

openid connect


authorization server (uaa)
client application (portal, cloudbees, etc)
/userinfo endpoint requires access token with openid scope
resource owner via user agent (browser)

client can use access token to authorize access to a resource server, including the uaa's /userinfo endpoint.

- openid connect also specifies another token type called an id_token which is not yet implemented.
- the id_token represents the authentication event and its audience is the client, not the resource server.
- it essentially contains information about the user's session with the uaa

simple cloud identity management - scim

- ietf draft changed name to system for cross-domain identity management.
- rest apis for managing user accounts, groups, and client registrations
- supports read, create, delete, full update, query
- partial update not yet implemented
- primarily designed for provisioning, but provides enough directory-like capabilities for what we need.

using uaac

- based on cf-uaa-lib, which is available from rubygems with damn fine documentation
- targets a uaa or login server similar to how vmc targets a cloud controller
- saves state in ~/.uaac.yml
- a context is relative to a target and contains an access token (and possibly a refresh token) for a specific user or client
- supports multiple targets, each with multiple contexts

demo uaac