Cognito

Understanding Cognito, Cognito User Pools and the Mobile Hub Helper and IOS SDK

Cognito

- Understanding Cognito is initially confusing for a variety of reasons:
  - Authentication, Authorization and Identity Management in a distributed system is complex. There are many parties with different roles and elaborate prescribed interactions with keys, tokens and signatures. The end user, the relying party (RP) the Identity Provider (IdP) the resource being used (RS) and the resource owner (RO). This terminology is used by the OpenId Connect and OAuth2.0 standards documents. For reasons that will become clear, this terminology is not consistently used by AWS. But they concepts and the entities are all there when using Cognito.
    - There are some good OpenID connect overviews online (ex: http://nordicapis.com/api-security-oauth-openid-connect-depth/) and a review of those will help.
  - Cognito allows non OpenID Connect Identity Providers, this is an advantage (ex: allowing the non-OpenID Facebook identity) but it also means that Cognito is playing a "federating" role. This role is assumed by but outside the scope of OpenID Connect standards documents.

Cognito Naming

- Cognito is a single name created by AWS to cover many functionalities and roles.
  1. There is the RESTFUL web API to Cognito, but there is also the Cognito SDK. The SDK calls and the API messages are not named the same, and SDK calls make multiple and conditional API calls.
  2. Cognito can federate identity providers. It can persist and association betwene authenticated users from different identity providers (So it can remember your google+ and your facebook identities and associated them with a single Cognito identityId.)
  3. Cognito can provide persistent identityId (which, if anonymous, follow an IOS device using keychain data) for users as well as authenticated users. These are stored in what is called an Identity Pool (NOT to be confused with a User Pool). Your app receives the same identityId for a user on different devices for authenticated users. Unauthenticated (Guest) identityId's follow a single device.
  4. Cognito can store (known as "Sync") state data IdentityId's (on the AWS server), which works for authenticated and unauthenticated users.
  5. Cognito has a AWSCredentialsProvider (a source for AWS Credentials for using AWS Services (Cognito but also S3, DynamoDB, etc)
  6. Cognito can create an OpenID Connect server called a User Pool, which can be used by Cognito Identity to Authenticate users.
  7. Cognito is new, but AWS Federated Identities and AWS Identity Management and AWS Credentials are not, so there are lots of classes with overlapping responsibility. And the naming conventions are confusing (consider the name AWSCognitoIdentityCognitoIdentityProvider). The use of the "cognito" brand name for userpools, really is a nightmare. An AWSCognitoIdentity thing is Cognito Federated Identity CFI but an AWSCognitoIdentityProvider thing is a thing like userpools an authentication provider also called an identity provider. I have a glossary of synonyms later in the presentation.
  - Classes starting with AWSCognitoIdentity (but NOT AWSCognitoIdentityProvider) are about the credentialsProvider/IdentityProvider, classes starting with AWSCognitoIdentityProvider relate to Oauth/Open Id Connect providers and other distributed identity providers (facebook).

SOME OBJECTS

- AWSServiceConfiguration – a thing that lets you request AWS Services
- AWSCredentialsProvider – a thing that provides credentials to AWSServiceConfiguration
- AWSCognitoCredentialsProvider – a thing that queries an AWSIdentityProvider for a logins list that contains the Identity Provide name and an ID Token that indicates that the user that is authenticated (the token is an OAuth/OpenId Connect token that results from an authentication with that login provider). It provides temporary credentials to use AWS Services (each services configuration has a credentials provider registered that it uses).
- AWSIdentityProvider – is a thing that conforms to a protocol that allows the Authentication of a user. It can return a logins dictionary.
- AWSIdentityProviderManager is an entity that can provide logins to the AWSCognitoCredentialsProvider for one or **multiple simultaneous** AWSIdentityProvider's. This allows the merging of identities so that (for example) one IdentityId can represent the same user on facebook and google+ and UserPools
- AWSCognitoUserPools – is AWSIdentityProvider and a (single identity)AWSIdentityProviderManager (so it can provide the logins dictionary but it will always just list the signle login that AWSUserPools authenticated. So while it can get you logged in with an identity, and credentials. But it cannot get you logged in with multiple AWSIdentityProviders simultaneously. To do that you need to have a separate AWSIdentityProviderManager.
- AWSCredentials - A thing that AWSServiceConfiguration can use to decide on access rights to AWS Services. Constructed by a CredentialsProvider.

Other Objects

- CognitoIdentity – provides the identity portion of Cognito's functions. Accessed using the AWSCognitoCredentialsProvider object (not sure if there is another way).
- IdentityId – this is a unique identifier provided by Cognito Identity. It can be unauthenticated (anonymous) (in which case it is assocated with a **single** IOS device by a keychain entry) or it can be authenticated in which case it is associated with an **Identity** (see below for ex: username) on one or more Identity Providers.
- **Identity** – This is some way of uniquely identifying a user that is authenticated by an identity provider (OpenID does not specify how this authentication is done but it is usually just a username, as it is in Cognito User Pools). This identity is returned in the claims in the ID Tokens in the entry returned in the logins dictionary (you can grab it from the log and paste it into ttps://jwt.io to decode it and see the id.

Synonyms

- Identity provider, authentication provider, Login provider, federated identity provider(s)
- Amazon Cognito, Cognito credentials provider, cognito identity (all seem to refer to the same class/process)
- Cognito user pool, Cognito Your User Pools, user pool.
  - Called an identity provider, authentication provider etc
- Cognito identity pool, pool, cognito pool, identity pool.
  - Occasionally called an identity provider (which seems incorrect) – never called an authentication provider
- Developer identity, developer authenticated identities, developer provider, developer identity provider (and incorrectly BYOI)
- IdentityId, Identity ID, id (as in get-id), identity (as in get-credentials-for-identity but not as in a specific identity on an identity provider)
- Identity – sometimes this term is used not to mean IdentityId but instead to refer to a unique username or other specifier on an identity provider.
- Federation means multiple things. I think this may be historical as maybe Cognito developed after WIF.
  - Web identity federation
  - Cognito federated identities
- BYOI (bring your own idenity) – recognized to mean a user may use google, facebook or another identity provider (perhaps a developer provided identity). But AWS Doc often confuses BYOI with developer authenticated identities.

IdentityId Behaviors

- An identity id looks something like this: **us-east-1:982396fs-841e-3cdd-9r43-e7ac41bhbcb28**
- Note: The bold items in this page are to help clarify when I am talking about an Identity Provider ***identity*** and when I am talking about a Cognito Federated Identity Pool IdentityId.
- The IdentityId is maintained on an IOS device in a keychain entry. For an unauthenticated IdentityId it remains the same until you clear the keychain (This can be done in simulator by Simulator -> Reset Content and Settings…). At that point that IdentityId is abandoned. It is not disabled, it is just never used again.
- Authenticating disables the unauthenticated IdentityId (the IdentityId will be marked with DISABLED in the Logins array in the identityPool entry. You can see this in the Cognito console.) that is currently on the device and merges it's data into the authenticated IdentityId. There is one exception: If this is the first time the authentication takes place for this ***Identity*** *(meaning username not IdentityId)* then the unauthenticated IdentityId is not abandoned but is associated with the ***Identity*** and used as the authenticated IdentityID going forward.

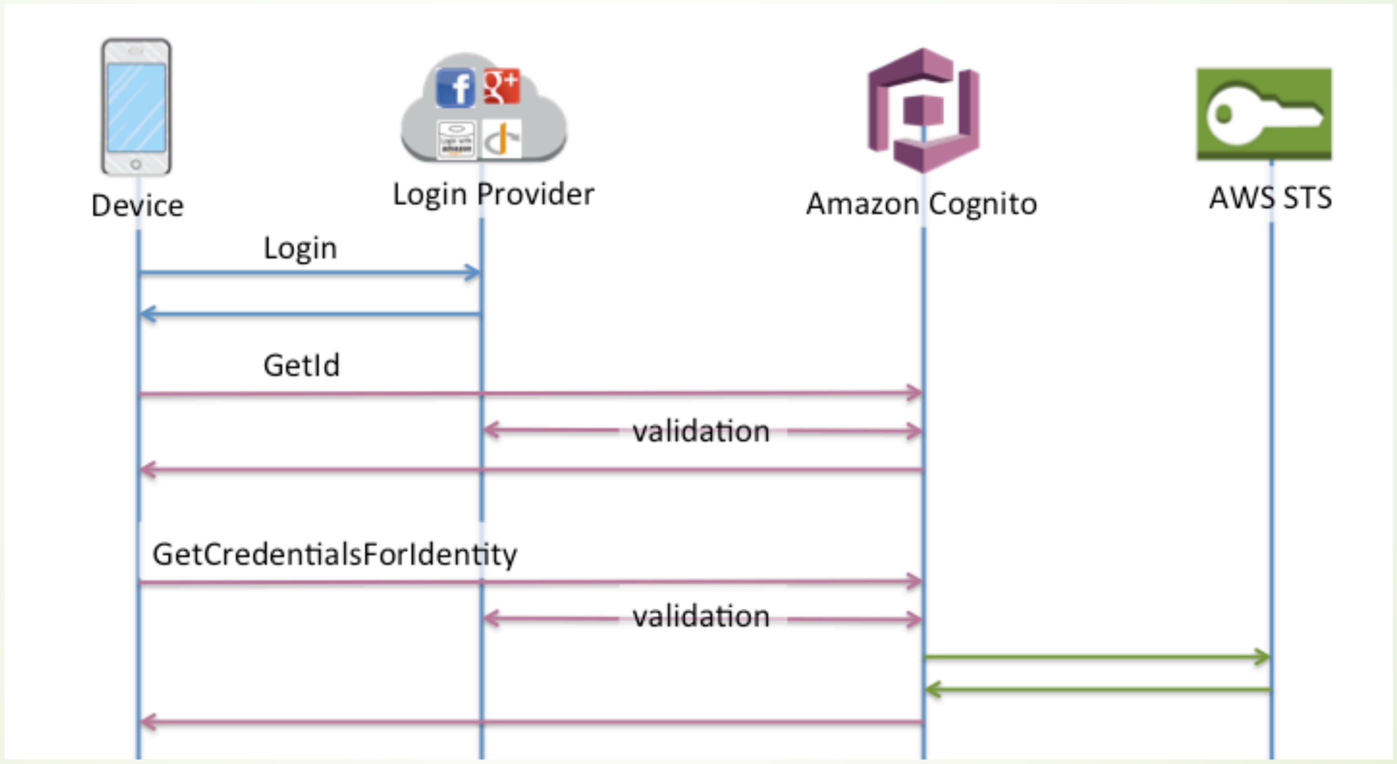
IdentityId Behaviors

- Merging multiple ***Identities*** (meaning usernames not IdentityId's) from different identity providers disables the IdentityId of one, and associates the two ***Identities*** with the other IdentityId. Disabled Id's get created whenever you merge an Id's synced data into a dataset. These ID's are marked with DISABLED in the Logins array in the identityPool entry for that ID.
- In practice this process creates a reasonable use of unique id's with disabled id's only getting created when a user authenticates on a new device (It can be bothersome in testing as it creates a barrage of disabled and unused identityId's as the tester logs out and in multiple times with multiple id's). In practice a device user would:
  - Connect – get an unauthenticated id - authenticate – and use the same ID. No abandoned id is created.
  - Connect on another device – here he/she would get a new unauthenticated id – and when he/she authenticated and got the identityId for his/her ***identity***, one unauthenticated id would be disabled and abandoned.
  - Each merging of ***identities*** from two identity providers would also create a disabled and abandoned identityId.

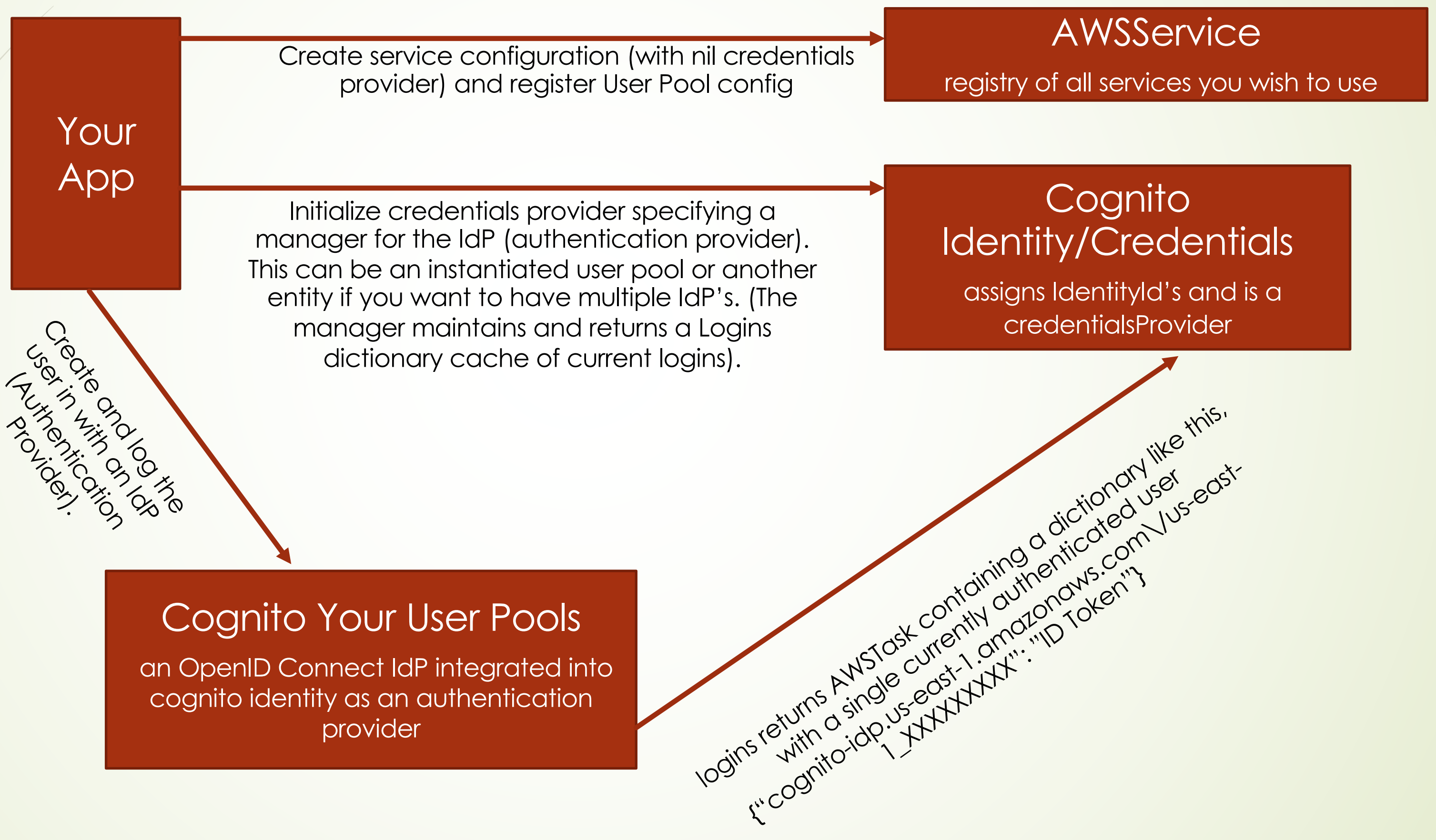
Where Cognito buries it's acorns.

- Cognito stores a keychain on the device that contains the last identityId that was used. I think this is used by the credentialsProvider/identityProvider object upon a call to credentialsProvider.credentials (IOS SDK name) to re-use an existing identity (for example unauthenticated) and avoid creating unused identities unless the user truly is not going to log in or resume.
- Mobile-Hub-Helper's AWSSignInProvider's and AWSIdentityManager store an indication of an open session state in NSUserDefaults. These are used to re-start the session if the app is terminated and restarted.
- The original AWSIdentityManager did not support identity merging, did not fully support providers other than hard-coded google and facebook. Now (with this modified version) you can control the names of those NSUserDefaults keys and associate them with AWSSignInProvider class names, and all of those classes will get resumed if they have open sessions.

Confused?


Device
Login Provider
Amazon Cognito
AWS STS
Login
GetId
validation
GetCredentialsForIdentity
validation

Single IdPAuthentication flow using IOS SDK and User Pools (SDK manages much of the state and flow)


Your App
AWSService
registry of all services you wish to use
Cognito Identity/Credentials
assigns IdentityId's and is a credentialsProvider
Cognito Your User Pools
an OpenID Connect IdP integrated into cognito identity as an authentication provider

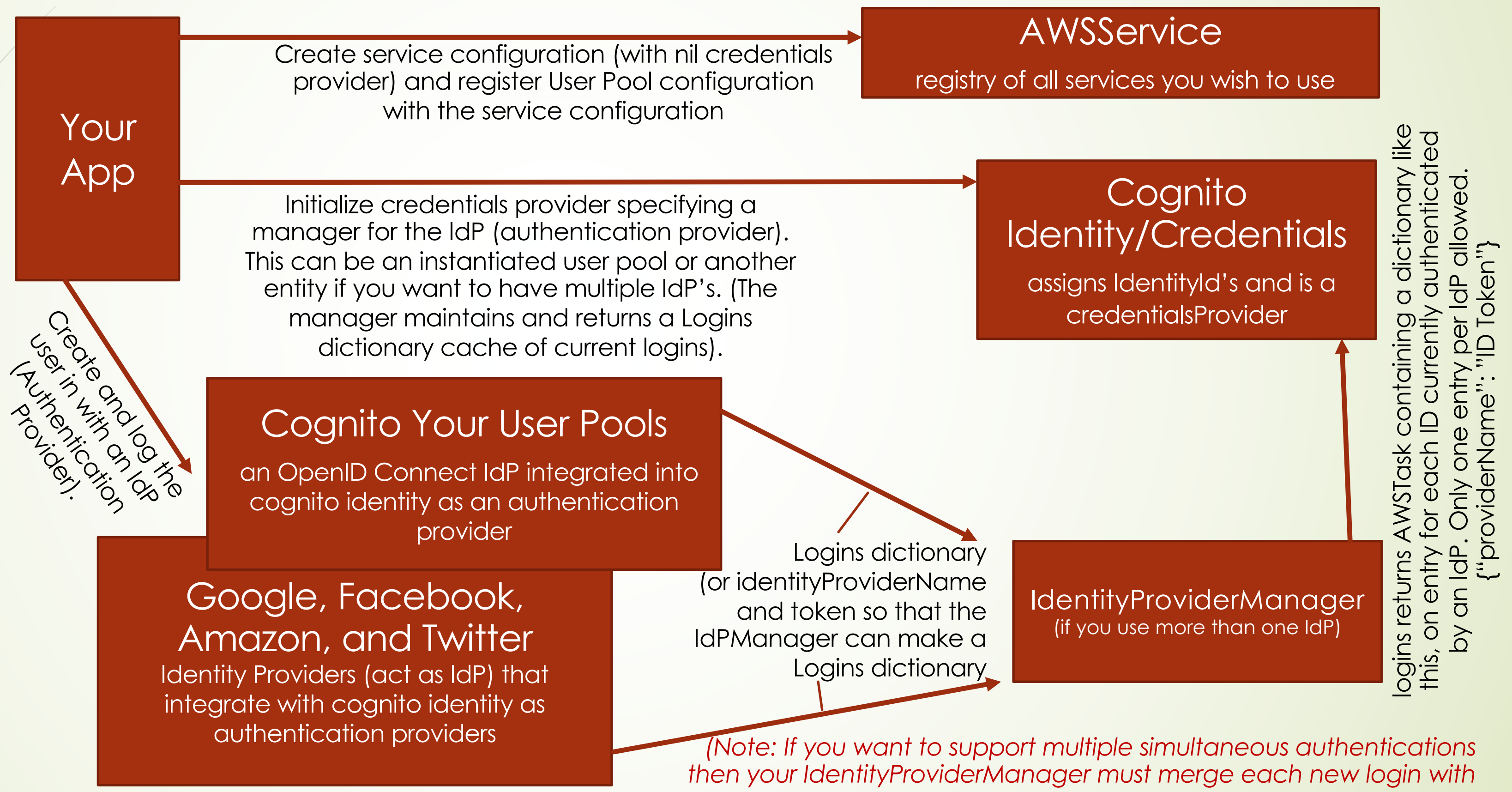
Create service configuration (with nil credentials provider) and register User Pool config

Initialize credentials provider specifying a manager for the IdP (authentication provider). This can be an instantiated user pool or another entity if you want to have multiple IdP's. (The manager maintains and returns a Logins dictionary cache of current logins).

Create and log the user in with an IdP (Authentication Provider).

logins returns AWSTask containing a dictionary like this, with a single currently authenticated user {"cognito-idp.us-east-1.amazonaws.com\/us-east-1_XXXXXXXXX": "ID Token"}

Multiple IdP Authentication flow using IOS SDK including User Pools (SDK manages much of the state and flow)


Your App
AWSService
registry of all services you wish to use
Cognito Identity/Credentials
assigns IdentityId's and is a credentialsProvider
Cognito Your User Pools
an OpenID Connect IdP integrated into cognito identity as an authentication provider
Google, Facebook, Amazon, and Twitter
Identity Providers (act as IdP) that integrate with cognito identity as authentication providers
IdentityProviderManager
(if you use more than one IdP)

Create service configuration (with nil credentials provider) and register User Pool configuration with the service configuration

Initialize credentials provider specifying a manager for the IdP (authentication provider). This can be an instantiated user pool or another entity if you want to have multiple IdP's. (The manager maintains and returns a Logins dictionary cache of current logins).

Create and log the user in with an IdP (Authentication Provider).

Logins dictionary (or identityProviderName and token so that the IdPManager can make a Logins dictionary

logins returns AWSTask containing a dictionary like this, on entry for each ID currently authenticated by an IdP. Only one entry per IdP allowed. {"providerName": "ID Token"}

(Note: If you want to support multiple simultaneous authentications then your IdentityProviderManager must merge each new login with the logins dictionary. To support one at a time there is no merging. Merging is support by the modified version of the mobile hub helper.)

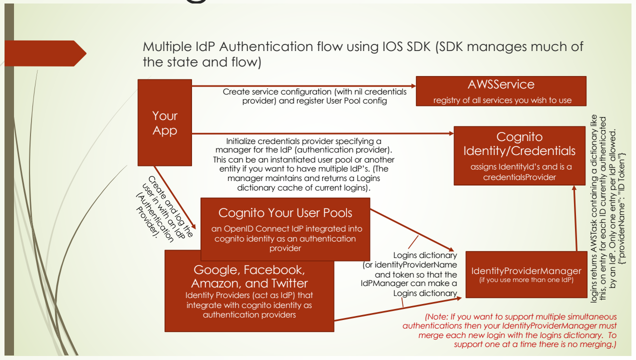
What does an AWSIdentityProviderManager do?

- Manages federated AWSIdentityProviders
  - While really all it needs to do is return (to credentials provider) logins with the list of providers and ID Tokens you want merged, a reasonable implementation is to call most or all of the IdentityProvider functions through the Identity Provider Manager. That is what the mobile-hub-helper AWSIdentityManager does.
    - Unfortunately AWSIdentityManager only returns the providername and token for a single identity provider.
  - Instead what it can do (with this modification and if you set the Info.plist key "Allow Merged Identities" to YES) is maintain an NSDictionary called cachedLogins, which is added to when a new login call is made and which is shortened when a logout call is made. Then when it returns logins it always returns the loginCache.
    - When the credentials provider calls it's associated AWSIdentityProviderManager logins method, it may find a list of logins instead of just one. In that case it will merge those logins in it's database and disable the identityId of one of them. How does it know which ID goes with which login? The ID Token contains an encoded decryptable (paste the token into https://jwt.io to see for yourself) set of claims, one of which is the identity (ex: username)
    - Multiple provider logins are maintained (each one creating an NSUserDefaults key) and when you log out of one, any other active sessions are restored. (So you have to log out as many times as you log in). An enhancement would be a logoutAll.
    - Even though you have an identityId that as multiple related logins, you are only ever authenticated by one provider. That is the provide that you have access to (for instance, to ask it for your imageURL etc. That is another area for improvement.

About merging identities

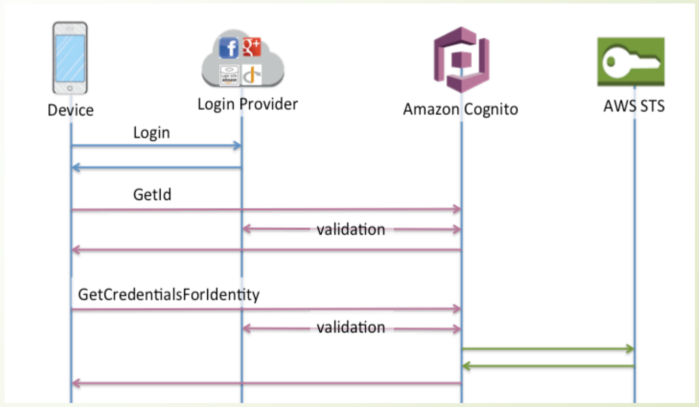
- You can probably think of all sorts of gotcha's when merging identities.
    - What if I try to merge two identities from the same provider (wouldn't the dictionary keys be the same?)
    - What if I try to merge two identities, each of which has a different identity from the same provider associated with it (and again they would create two entities with the same keys).
- Cognito manages this beautifully and rejects attempts to merge identites that cannot be merged. The rejection happens at login time (when you would try to merge in an identity with a different token).

The IOS SDK, behind the scenes communicates with with AWS via the AWS Cognito API.


Multiple IdP Authentication flow using IOS SDK (SDK manages much of the state and flow)
Your App
Create service configuration (with nil credentials provider) and register User Pool config
AWSService
registry of all services you wish to use
Initialize credentials provider specifying a manager for the IdP (authentication provider). This can be an instantiated user pool or another entity if you want to have multiple IdP's. (The manager maintains and returns a Logins dictionary cache of current logins).
Cognito Identity/Credentials
assigns IdentityId's and is a credentialsProvider
Create and log the user in with an IdP (Authentication Provider).
Cognito Your User Pools
an OpenID Connect IdP integrated into cognito identity as an authentication provider
Google, Facebook, Amazon, and Twitter
Identity Providers (act as IdP) that integrate with cognito identity as authentication providers
Logins dictionary (or identityProviderName and token so that the IdPManager can make a Logins dictionary
IdentityProviderManager
(if you use more than one IdP)
logins returns AWSTask containing a dictionary like this, on entry for each ID currently authenticated by an IdP. Only one entry per IdP allowed. {"providerName": "ID Token"}
(Note: If you want to support multiple simultaneous authentications then your IdentityProviderManager must merge each new login with the logins dictionary. To support one at a time there is no merging.)

- The Auth flow diagram below seems to imply that Cognito (the RP) will validate the token out of band with the IdP. But it is not clear if this is the way it is done or if it is using bearer tokens.
- Cognito can get the ID Tokens when it wants from the IdentityProviderManager logins dictionary
- The service configuration has an associated credentialsProvider (in our case Cognito). The credentials provider has an associated IdentityProvider or IdentityProviderManager if using multiple identity providers.
- So other than requesting services from the correct serviceConfiguration, we should not have to manage or provide tokens or credentials (the SDK does most of that).
- Cognito's credentials provider provides credentials that have an access token that expires after 3600 seconds (by default). You can make that time shorter but not longer. The expired tokens can be refreshed but I have not tested so there is some question as to who/what is responsible in the SDK when a token expires.

- SDK flow is above, Authentication flow is on the right
- In the diagram on the right the device is your app, the login provider is the IdP or IdentityProvider, Amazon Cognito is Cognito Identity/Credentials and AWS STS is the Security Token Service
- The flow illustrated is the flow that occurs through the cloud to AWS Services from your App, not the flow of calls you make to the SDK (Which are very different)
- Not shown in the diagram above: There are two ways to log a user in using User Pools, one is using the userpool delegate (which will automatically be called to authenticate when needed) the other is to call userpool.getSession with a username and password. When allowing unauthenticated users it is not clear how to avoid having the authentication delegate kick off when you try to get credentials for the unauthenticated identity id. (One kludgy solution is to use an entirely separate serviceConfiguration with an entirely separate credentialsProvider created via AWSAnonymousCredentialProvider. This works because the UserPool is not connected to the credentialProvider or serviceConfiguration, so the delegate is not activated.



Configuration Errors

- You must configure Cognito User Pools as an authentication provider in your Cognito Federated Identity Pool. You do NOT need to configure it as an OpenID Connect provider as well. You must specify the correct User Pool App Client Id in the config or you will get **message=Token is not from a supported provider of this identity pool.**
- If you do configure it as an OpenID Connect provider as well, you must specify your "User Pool App Client Id" as the audience in the identity provider you create in IAM, if you do not the ID token in logins will contain an audience that does not match and you will get: **message=Invalid login token. Incorrect token audience.**

Errors

- OSStatus error: [-34018] Security error has occurred or OSStatus error: [-25299] Security error has occurred
  - Cognito uses keychains in a way that requires the "Keychain Sharing" capability in Target->Capabilities.
- "x-amzn-ErrorMessage" = "Invalid identity pool configuration. Check assigned IAM roles for this pool.".
  - IAM is the source of permissions to use the AWS Services. It has a console just like everything else.
  - Identity Pools are "trusted" in IAM, and each pool has roles associated typically with authenticated and unauthenticated users.
  - Roles have policies assocated with them.
  - You need to make sure that the roles specified in your Identity pool has policies and trust policy that uses the correct pool id. This can get disturbed if you recreate pools etc.

Errors

- "x-amzn-ErrorMessage" = "Logins don't match. Please include at least one valid login for this identity or identity pool."
  - This usually occurs because you attempt to log in as another user without logging out. The SDK recovers by retrying.
  - It can also happen if you wrote your own AWSSignInProvider and constructed identityProviderName or token() incorrectly.
- "x-amzn-ErrorMessage" = "Unauthenticated access is not supported for this identity pool."
  - You have an identityId and you are trying to use it, but your Identity Pool is set up without "Allow unauthenticated identities", and you have not yet authenticated.
- "x-amzn-ErrorMessage" = "Cannot merge these identities.";
  - The logins dictionary can only hold one entry for each identityProviderName (because it is a dictionary). So it is not possible to associate an identityId with multiple different users on a single identity provider.
  - Also, if an identityId is associated with an identity provider it cannot be associated with another identityId that is also associated with the identity provider. This is really just an indirect version of the logins dictionary restriction. Example: If identity Id **us-east-1:dbbcfd3b-e877-4870-a7a4-0391ae962cb2** is associated with a facebook username. It cannot be merged with an identityId that is associated with a different facebook username.

Errors

- "x-amzn-ErrorMessage" = "Token is not from a supported provider of this identity pool."
  - This means that the credentials provider (inside Cognito) has inspected the logins dictionary and found that the token listed is not from an authentication provider that is registered with the pool. You fix this in the console by going to authentication providers section and adding the missing provider.
- "x-amzn-ErrorMessage" = "Invalid login token. Issuer doesn't match providerName"z
  - I think this can happen when a refresh token fails. Not sure.Check Client Ids, Api keys, audiences etc.