Introduction to the WSO2 Identity Server & Contributing to an OS Project

Michael Geiser
PhillyJUG
June 24, 2015

Agenda

- Overview of WSO2 Company and Platform
- Summary of WSO2 Identity Server
- Demo of Identity Server Main Features
- Demo of Single Sign On with SAML2 and OAuth
- Development of Feature Extending OS Product
- Process and Status of Contribution

- I added the Code Commenting discussion notes to the end of the deck.

What is WSO2?

Site: http://wso2.com

Company Overview: http://www.slideshare.net/wso2.org/wso2-platform-introduction?related=2

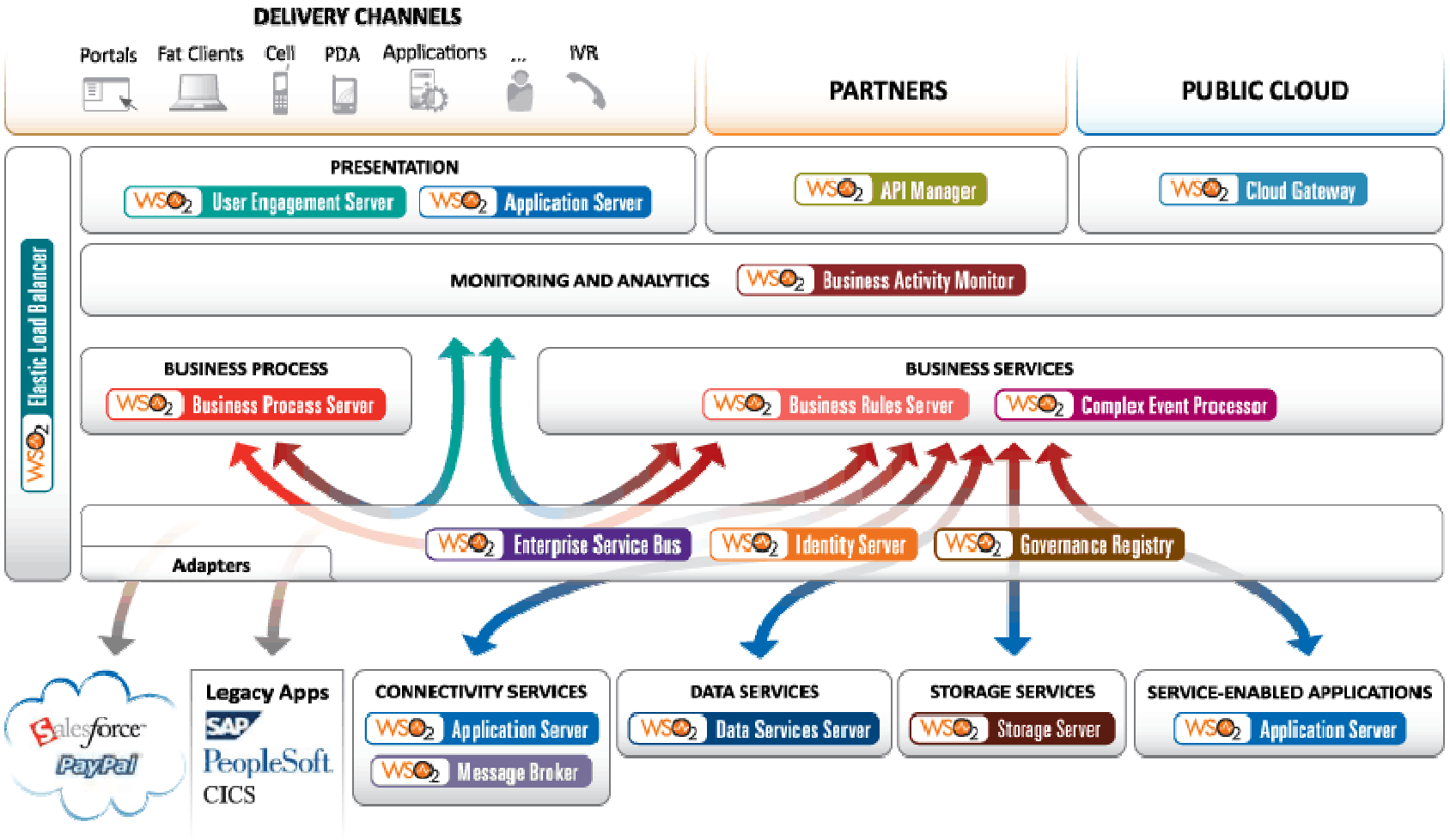
The suite of WSO2 products are 100% Open Source and based on Open Standards. WSO2 monetizes the product by selling support much like RedHat (but without the subscription-only Enterprise Edition).

Developers can extend the platform and customize code (we'll see an example of this later).

WSO2 says their key advantage is all WSO2 products are built on a common foundation – "WSO2 Carbon"; a modular, reconfigurable, elastic, OSGi-based architecture. This creates a strong stable base for building as well as integrating with existing large-scale enterprise applications.

What is WSO2?

Site: http://wso2.com



What is WSO2 Identity Server?

Product Site: http://wso2.com/products/identity-server/

WSO2 Identity Server enables enterprise architects and developers to

- Deliver an inter-Enterprise Single Sign-On/Signout environment.
- Simplify identity provisioning
- Guarantee secure online interactions

The WSO2 Identity Server decreases identity management and entitlement management administration burden by:

- Including role based access control (RBAC) convention
- Fine-grained policy based access control - XACML
- SSO bridging to internal and external destination

WSO2 Identity Server is an entitlement management server for security and identity management of :

- Enterprise Web applications
- Services
- APIs

What is WSO2 Identity Server?

Product Site: http://wso2.com/products/identity-server/

WSO2 Identity Server enables enterprise architects and developers to

- Improve customer experience by reducing identity provisioning time
-
-

Th nt
ma
-
-
-

An open source Identity & Entitlement management server

WSO2 Identity Server is an entitlement management server for security and identity management of :

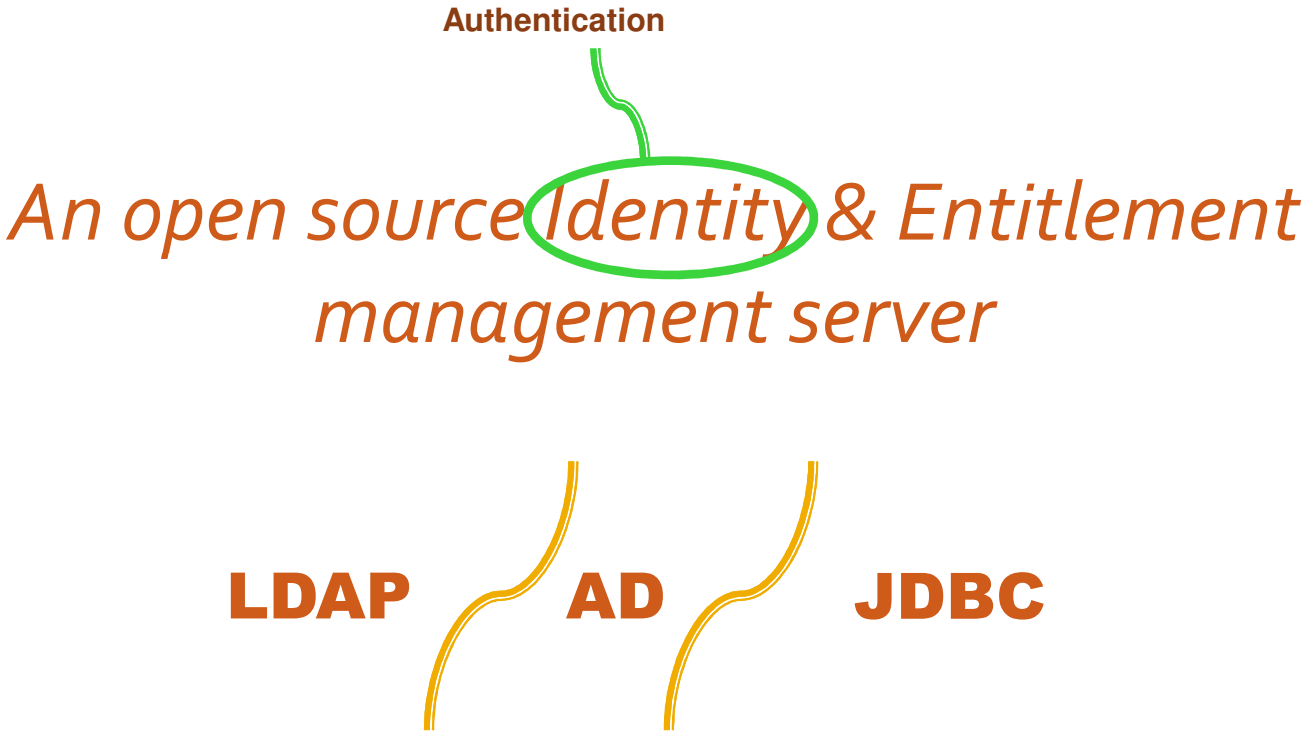
- Enterprise Web applications
- Services
- APIs

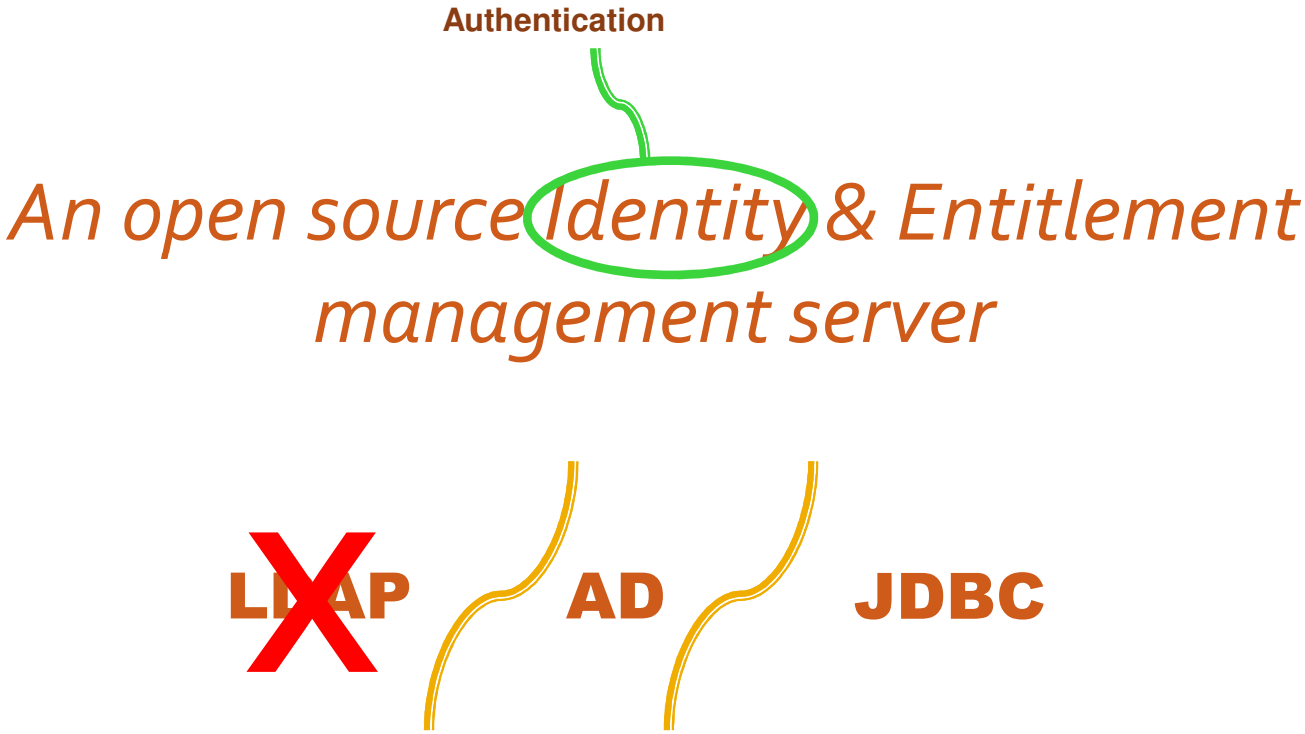
WSO2 IS 5.0.0 SP1 Main Feature Set

- User Stores and configurations in LDAP/AD/JDBC/others
- Multiple integrated user stores
- Multi-tenant
- Multiple Identity Standards
  - OpenID
  - SAML2
  - OAuth 1.0a/2.0
  - Security Token Service with WS-Trust
  - Kerberos
  - Integrated Windows AD Authentication
  - WS-Fed Passive
- XACML 2.0/3.0
- SCIM 1.1
- WS-XACML

What is WSO2 Identity Server?

An open source Identity & Entitlement management server


Authentication
An open source Identity & Entitlement management server
LDAP
AD
JDBC

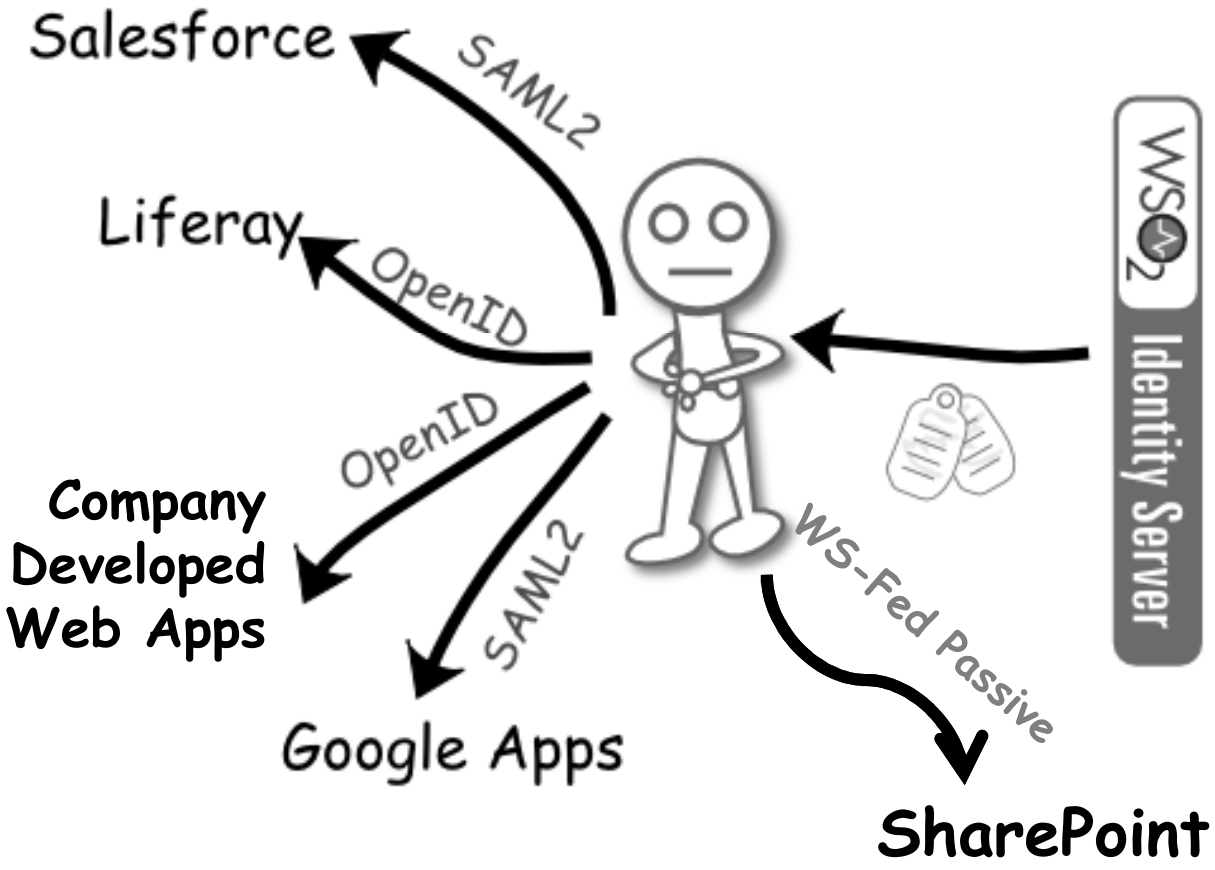

Authentication
An open source Identity & Entitlement management server
LDAP
AD
JDBC

Single Sign On

Authentication

An open source Identity & Entitlement management server

Single Sign-On



SAML2

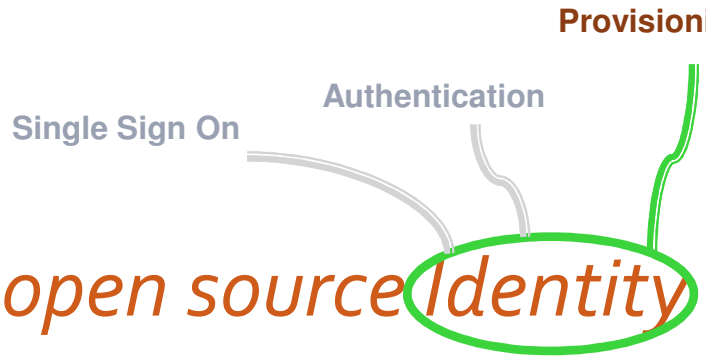
- Single Sign On / Single Logout
- Widely used by *aaS providers [Google Apps, Salesforce]
- SAML2 Web SSO Profile
- SAML2 Attribute Profile
- Distributed Federated SAML2 IdPs

OAuth

- Identity Delegation
- Securing RESTful services
- 2-legged & 3-legged OAuth 1.01
- XACML integration with OAuth
- OAuth 2.0 support with Authorization Code, Implicit, Resource Owner Credentials, Client Credentials

OpenID

- Decentralized Single Sign On
- Single user profile
- Widely used for community & collaboration aspects
- Multifactor Authentication
- OpenID relying party components



An open source Identity & Entitlement management server

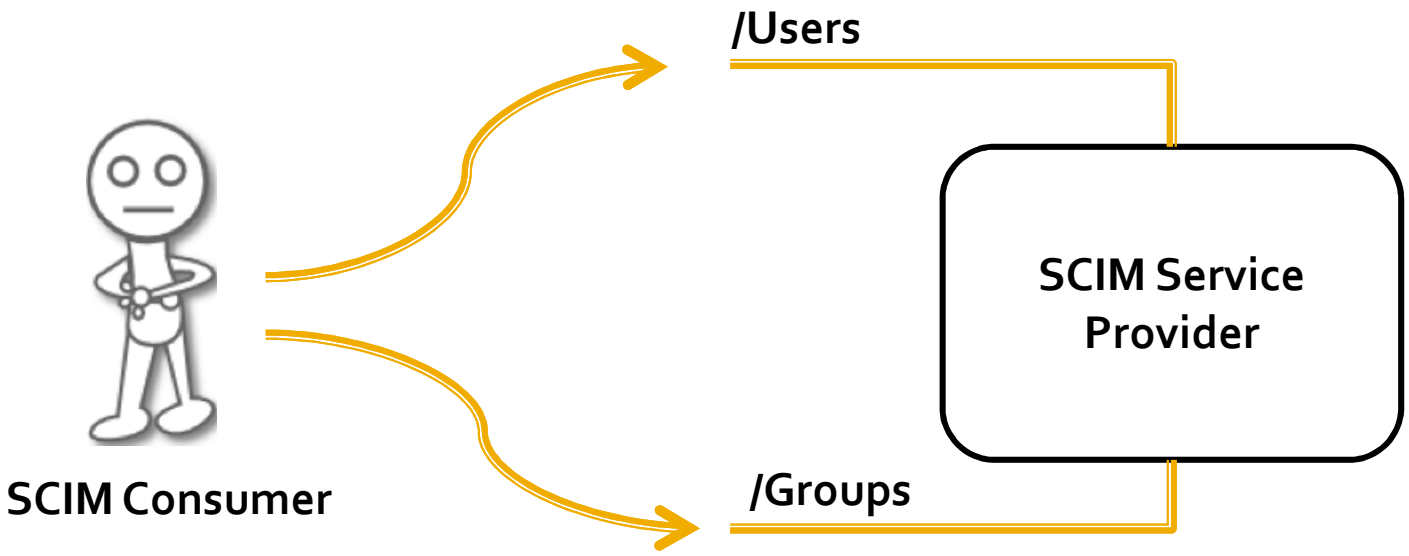
WSO2 APIs

SPML

SCIM

Service Provisioning Markup Language

System for Cross-domain Identity Management



REST services for SCIM Provisioning

add-user.json

```
{
 "schemas":[],
 "name":{"familyName":"Geiser","givenName":"Michael"},
 "userName":"mgeiser","password":"correcthorsebatterystaple",
 "emails":[{"primary":true,"value":"phillyjug@gmail.com","type":"jugmaster"},
{"value":"mgeiser@mgeiser.net","type":"personal"}]
}
```

curl command

curl -v -k --user admin:admin -d @add-user.json --header "Content-Type:application/json" https://localhost:9443/wso2/scim/Users

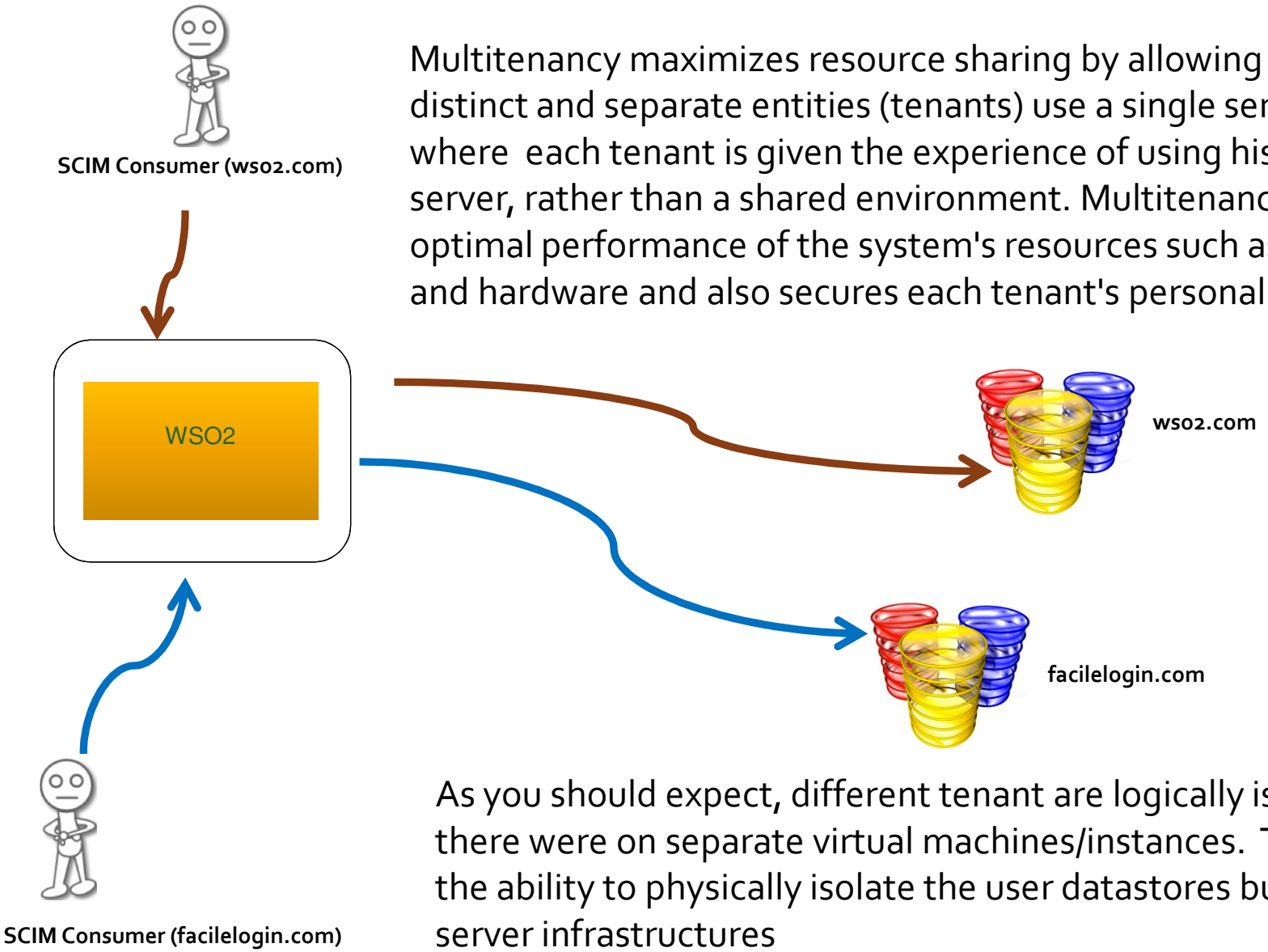
add-group.json

```
{
  "schemas": ["urn:scim:schemas:core:1.0"],
  "id": "PhillyJUG_Members",
  "displayName": "PhillyJUG Members",
}
```

curl command

curl -v -k --user admin:admin -d @add-group.json --header "Content-Type:application/json" https://localhost:9443/wso2/scim/Groups

Multitenancy

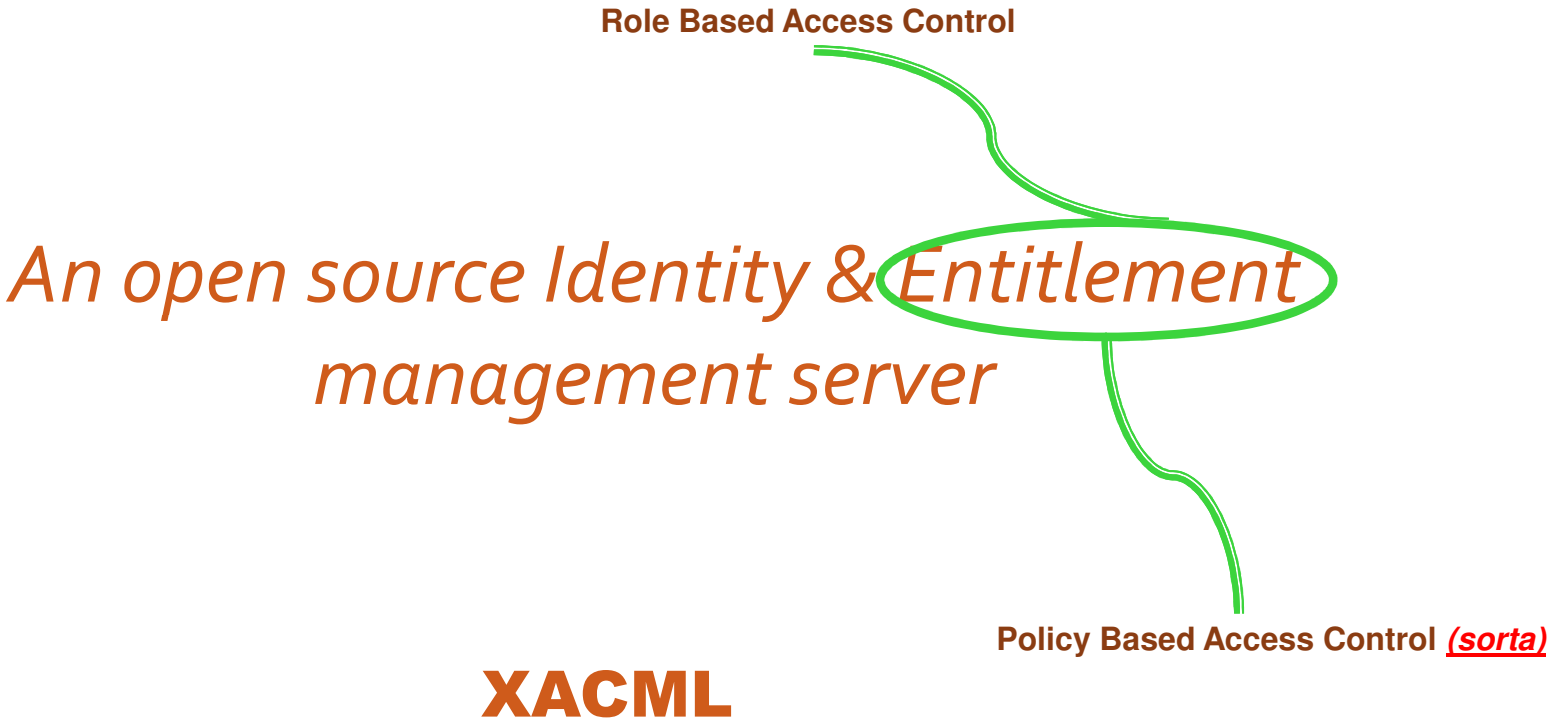


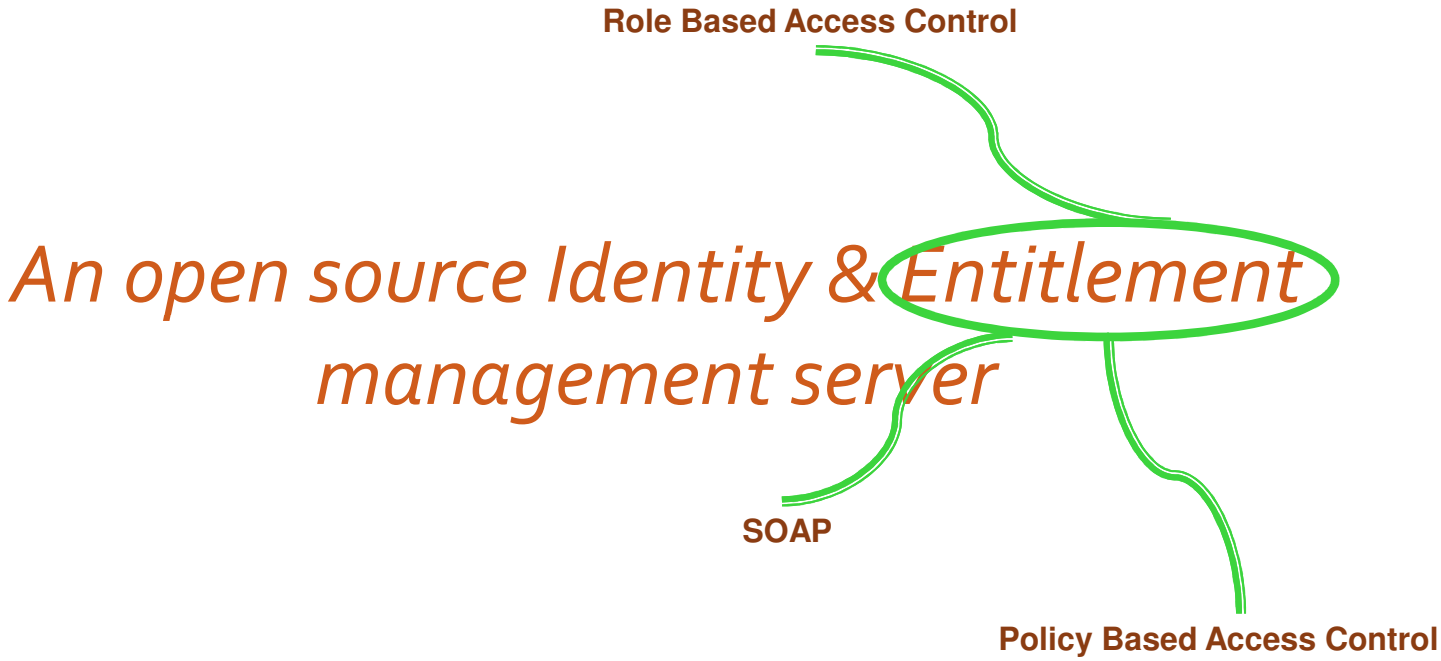
Multitenancy maximizes resource sharing by allowing multiple distinct and separate entities (tenants) use a single server/cluster where each tenant is given the experience of using his/her own server, rather than a shared environment. Multitenancy ensures optimal performance of the system's resources such as memory and hardware and also secures each tenant's personal data.

As you should expect, different tenant are logically isolated as if there were on separate virtual machines/instances. There is even the ability to physically isolate the user datastores but use the same server infrastructures

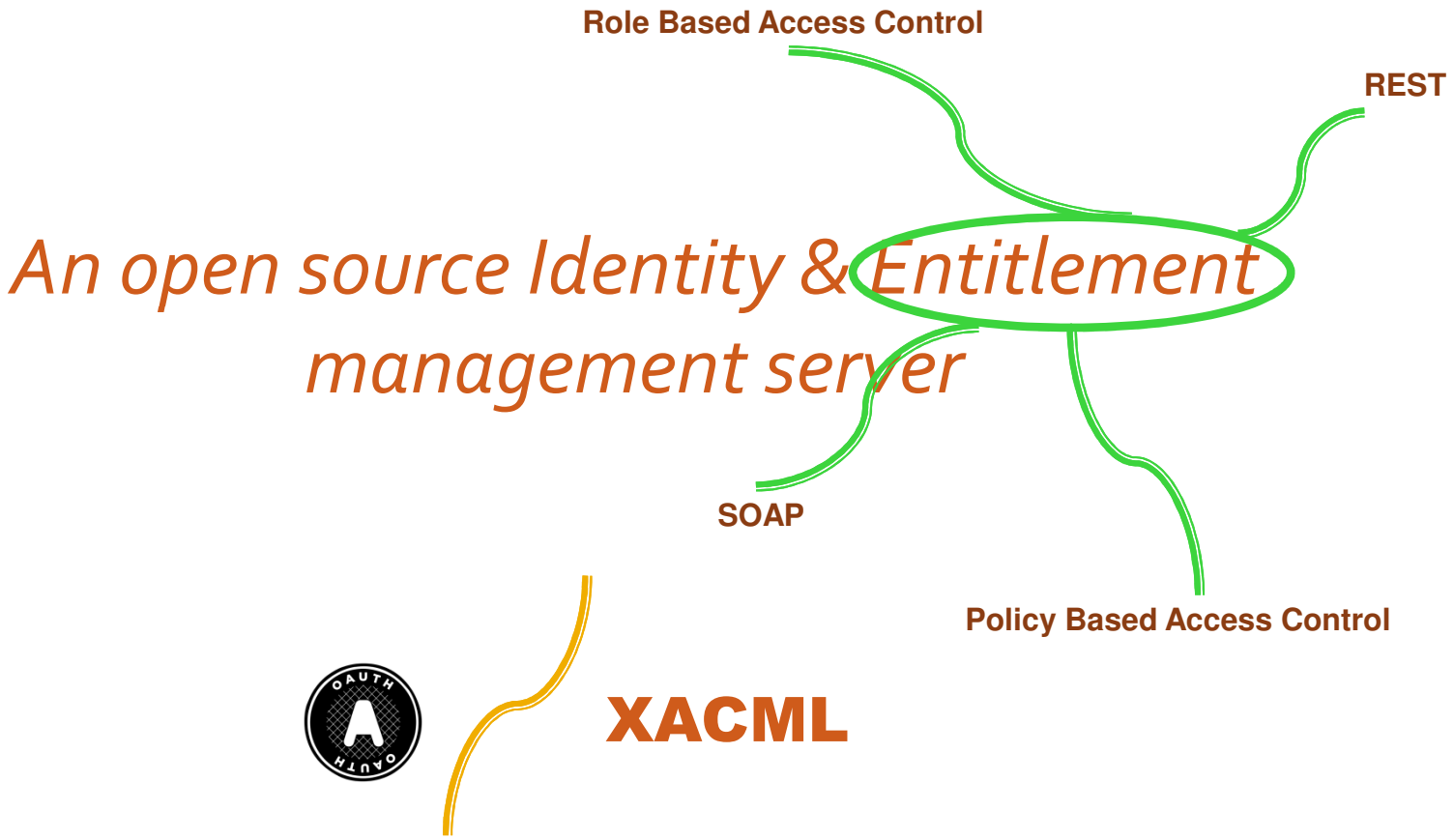
Role Based Access Control *(sorta)*

An open source Identity & Entitlement management server


Role Based Access Control
An open source Identity & Entitlement management server
Policy Based Access Control (sorta)
XACML



XACML / WS-XACML


Role Based Access Control
REST
An open source Identity & Entitlement management server
SOAP
Policy Based Access Control
OAUTH
XACML

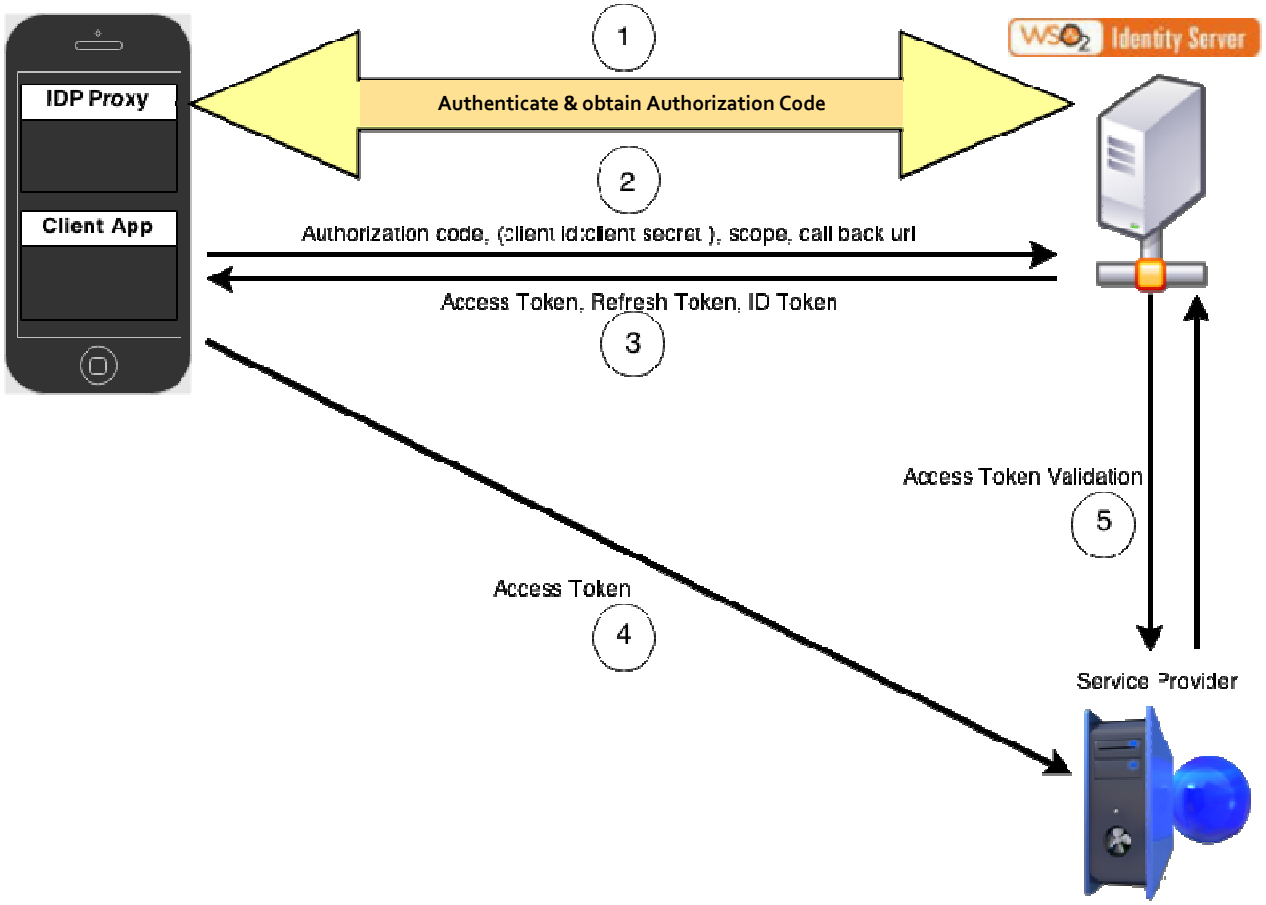
XACML ("zack-mul")

- e**X**tensible **A**ccess **C**ontrol **M**arkup **L**anguage
- Implements XACML 3.0
- Support for multiple PIPs
- Policy distribution
- Decision / Attribute caching
- UI wizard for defining policies
- Notifications on policy updates
- TryIt tool to test WIP

WSO2 Demo

- Management Console
- Single Sign On between Multiple Apps

Mobile IdP Proxy



Setting up a Playground: VM vs. AWS

Laptop Based VMs

- Need a ~~good~~ great laptop: quad core i7, 32Gb RAM, 500GB SSD = ~$1500 (I have a ThinkPad w540)
- Maybe can get away with ~$425 for 32Gb RAM and a big (750Gb) SSD if your current laptop can take the RAM (16Gb OK)
- Requires you to do provisioning and set up networking
- Must decide between VirtualBox & VMware (Linux VMM is like LDAP, Open Office or Isla Nublar; best avoided when possible)
  - If everyone doesn't standardize , you can 't easily share VMs
- Availability and access for others limited to when your laptop is online
- Networking and Port security can cause problems even with NAT
- Linux networking with multiple VMs on a laptop that changes network connections is problematic.
- Almost need to have Linux host OS
- Docker and Vagrant are your friend

Setting up a Playground: VM vs. AWS (2)

AWS

- Can use "normal" laptop
- Everyone can access anywhere anytime
- Per hour costs are quite reasonable; especially if you shut down when not working;
  - Is $25-$50/month recurring OpEx for AWS better than $2000 CapEx? For a new laptop (but you should lease your laptop regardless so you can refresh your tech more often…)
- AWS provisioning and networking abstracts many PITA details for you
- Having ***REAL*** AWS experience is a HUGE plus on your resume…

Want to learn Linux? This is a great start (plus you should be taking courses on EdX anyway)
https://www.edx.org/course/introduction-linux-linuxfoundationx-lfs101x-2

Contributing to OS - WSO2

- Feature Gap
- Set up Dev environment
- Mapped requirements to features and planned implementation

Oh sh*t! We Have a Feature Gap!

- The WSO2 JDBC UserStore doesn't have a Password Reuse Policy (I know…*REALLY*??!!)
- This is a must-have feature for almost anyone
- WSO2 is Java & Open Source; we do Java.
  - How hard could it be?
  - What could go wrong?
- We already scheduled WSO2 onsite for "Quick Start Week " engagement and just added this to the agenda and breakout sessions

Password Reuse Policy Overview

The high level requirements for Password Reuse Policy.

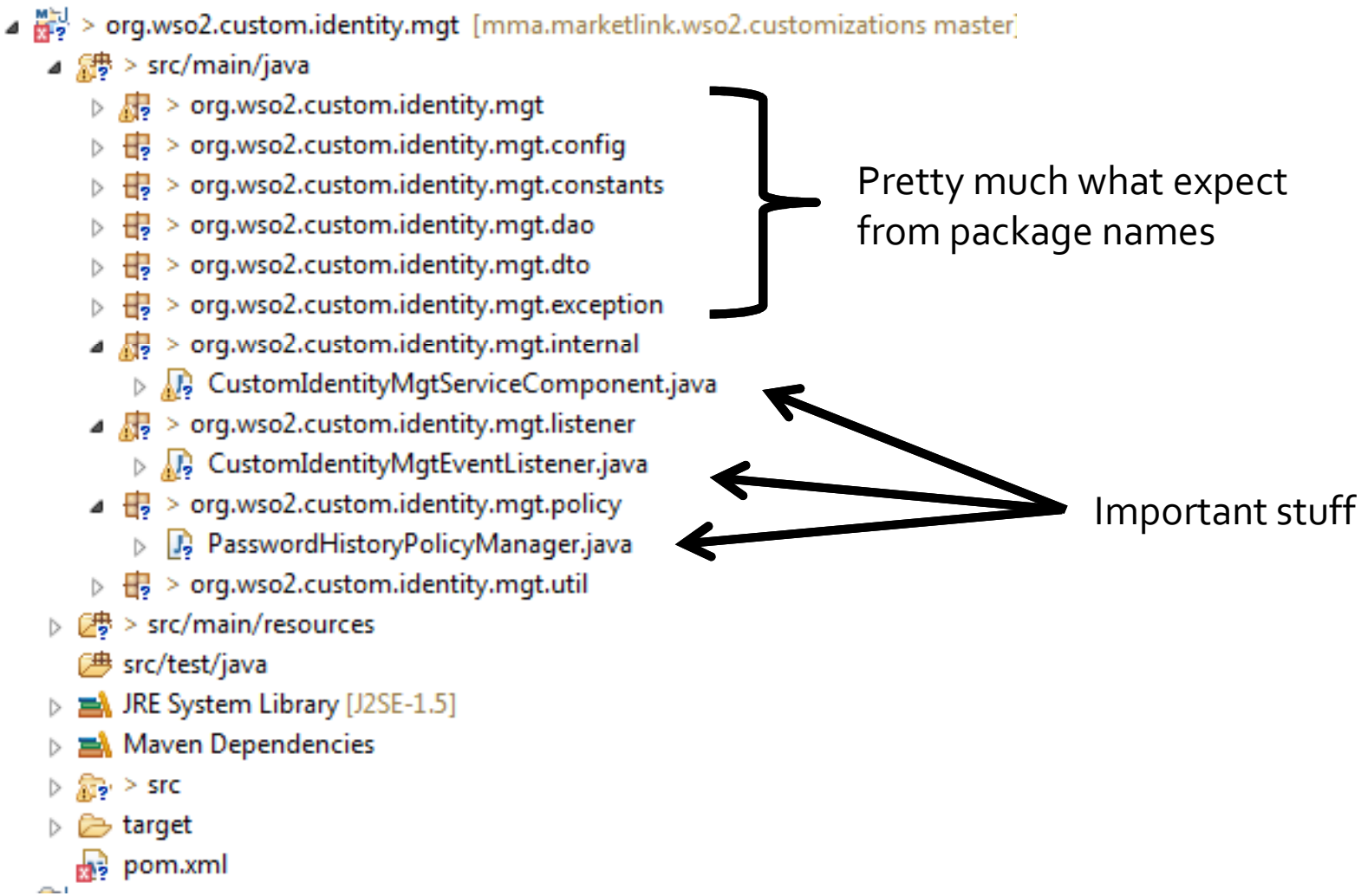
- Settings properties file will allow configuration of:
  - Time-based Password Reuse (# of days before reuse)
  - Frequency-based Password Reuse (# of interim passwords before reuse)
- Admins will be allowed to chose either Time-based and/or Frequency-based Password Reuse on/off and settings
- These will be additive; if the admin sets the timeThreshold=90 and frequencyTheshold=10 then a password cannot be reused for 90 days regardless of how many times it changes ***and*** a password cannot be reused for 10 password changes (even if the user only changes the password once every 90 days).
- Data must be persisted securely to support this policy
- Data maintenance will be implement to remove unneeded records and must happen during the password change event
- More…

(this was a short, low resolution summary of the User Stories describing the feature)

WSO2 Architecture and Extensibility

- WSO2 uses Apache Felix and OSGi framework for a dynamic component based application
- The OSGi specification defines modular systems and a service platforms for Java that implement a complete and dynamic component model.
- Applications or components, packaged the form of bundles for deployment (i.e. jars) , can be remotely installed, started, stopped, updated, and uninstalled without requiring a JVM restart.
- The OSGi specifications have evolved beyond the original focus of service gateways, and are now used in applications ranging from mobile phones to the Eclipse IDE.

Project Structure



CustomIdentityMgtServiceComponent.java

- The component class contains the annotations needed to specify the component and when it is activated

```
@scr.component name="org.wso2.custom.identity.mgt.internal.CustomIdentityMgtServiceComponent"
immediate="true"
```

- The class implements activate() and deactivate() methods; think JUnit's setUp() and tearDown() methods

CustomIdentityMgtEventListener.java

- Our WSO2 component model extends the existing IdentityMgtEventListener class and override functionality where the behavior has to change.
- The original IdentityMgtEventListener essentially updates the Password for a user if the Password Composition Policy and other existing tests pass.

Example: a user has to provide the correct currentPassword and a newPassword and the newPassword must comply with all Password Composition Policy tests before a user's password is changed.

Our added functionality will:

- instantiate our new PasswordHistoryPolicyManager class where needed in the CustomIdentityMgtEventListener
- Call methods on the PasswordHistoryPolicyManager class as needed for new functionality
- Run updated or reimplemented code and call super as needed

PasswordHistoryPolicyManager.java

Actual Implementation of the New Policy

- Calls DAO to get Password History of a user
- Determines if new Password complies with the Frequency and Elapsed Time thresholds limiting reuse
- Calls DAO to insert new Password History data for a user
- Calls DAO to perform Password History database table data maintenance

Demo

- Toad to database
- Records change

How is New Feature Added to WSO2

WSO2 implements OSGi and Felix…

Deployment Procedure

Add the new table to the UserStore schema

- Run DDL in target database schema

Compile jar

- mvn clean install

Copy jar from maven repository to WSO2 "dropin" directory

- from **C:\Users\mgeiser\.m2\repository\org\wso2\sample\org.wso2.custom.identity.mgt\1.0.0-SNAPSHOT\org.wso2.custom.identity.mgt-1.0.0-SNAPSHOT.jar**
- to **<IS_HOME>/repository/components/dropins**

Copy configuration file from GIT to <IS_HOME>/repository/conf/security

Restart the WSO2 Identity Server service

Next Step: Refactoring

Code reviews, unit tests and functional testing have indicated a few improvements that are needed before this new component is fully "Production Ready"

- DDL Refactoring / Improvements
  - **DRI** - IDN_IDENTITY_PASSWORD_HISTORY.USER_NAME and IDN_IDENTITY_PASSWORD_HISTORY.TENANT_ID are FKs but no DRI is defined
  - **Covering Indexes** - SQL Execution Plans (and inspection) indicates additional indexes are needed
- Data Maintenance on User Delete Event
  - When a Password is changed, the Password History entries for a user is trimmed to only keep the number of passwords the are required to the Frequency and Time thresholds.
  - An additional component will be needed that will extend the User Management Service Component and delete all IDN_IDENTITY_PASSWORD_HISTORY entries for a user when the user is deleted.
  - Currently the records will be orphaned in the table. Eventually the orphaned records will affect the efficiency of the system and wastes storage. This needs to be fixed ASAP
  - Doing this data maintenance based on the delete user event is best; running a periodic job that finds orphaned records is "computationally expensive" and gets more resource intensive as the number of users in the system grows.

Final Step – Contribute Code to Product

- Requirement Definition and Dropin component submitted (pending final revisions) to WSO2 via our Account Manager
- WSO2 architects and reviewers will review code and artifacts and respond with questions
- WSO2 will add to roadmap based on capacity to update and test
  - Since requirements definition and working code exists as Dropin, this will help to minimize this effort and timeline
  - New DB table and DB traffic will slow down adoption slightly for testing
  - High complexity compared to other submitted features
- Code will be available as a dropin to other users until official adoption
- WSO2 will often include developers blogs and other contributions in the WSO2 documents site if it contributes to the community.
- I'll be writing up a page for this Dropin and attempt to get it on-line on late July or early August

Questions?

- Thanks for coming out!

Why I'm an LDAP Hater

Architecturally, an LDAP/Directory Server solution has drawbacks compared to an relational database solution:

- LDAP is designed for a high read-to-write ratio (10:1 or 100:1 is most often quoted as optimal for LDAP based directories). For any Password Policy that tracks attempted authentications, the Directory Server must update data once for every read that checks passwords (i.e. any authentication attempt). Idle and maximum (a.k.a soft and hard) timeouts are another required feature. Even though the Policy Server caches information whenever possible, implementing timeouts require updates to the User DataStore so that the timeout information can be shared among all Policy Servers.

- Directory Server data has limited Data Typing . There are **Strings, Numbers** (Integer), **Time, Telephone Numbers, Boolean, Binary, Distinguished Name** and **Bit Strings** data types in directory servers. Decimal (and all non-integer numeric) data and complex types (objects) must be stored as a string or serialized/deserialized and explicitly cast if used in any application (SQL, Java Visual basic…). And there are limits on searchability and indexability (and indexing in general); especially for non-native data types . Relational database (like Oracle) datatypes map to Java SQL datatypes without any casts.

- LDAP Connection Pooling support is non-existing or is very limited. This is a serious scalability and performance concern.

- LDAP is not a transactional protocol. Generally, Identity Management is closely coupled to other database transactions and the ability to have changes to the Identity Management user store and other schema participate in transactions is important.

- LDAP has no equivalent structure to stored procedures (and packages). It is desirable to have the SQL for data input and output abstracted from the calling applications to minimize the risk and impact to existing applications of future changes to the User DataStore. Decoupling the release cycles of the database and Business logic as much as possible is a more agile approach

- LDAP and Directory Servers do not generally have DRI, locking, or check constraints even if the relational database the LDAP implementation is built on supports them.

- A Directory Server has minimal Error Handling internally and externally error handlers must be coded and implanted in all code that calls into the Directory Server. Relational databases' Error Handling allows for better and more consistent exception handling, resolution, and logging and encapsulates these functions from the calling application.

Commenting Code Discussion

- Remember, you are writing code for other people (and your future self). They will have to debug or modify the code.
- Commenting is necessary because you need to tell people what you intended to do and why you chose to do it a specific way.
- While what the code does ***should*** be "self-evident" (an oxymoron usually) from inspection of the code, you need to communicate what you INTENDED to do; it is possible you made a logic error.
- Comments also do NOT slow you down; I've timed it. I spend about 10 to 20 minutes commenting an entire class. It is impossible to argue that is not time well spent.
- Comments are NOT counted as LOCs and absolutley do NOT add to maintainability costs; they do just the opposite.

Commenting Code Discussion (con't)

- Also, just as there are many ways to skin a cat (that is really an awful expression, isn't it?), there are many ways to implement the logic on how to implement a requirement.
  - You usually have an insightful and well reasoned thought process behind why you implement a bit of logic in a certain way. You need to ***document in comments*** why you chose a specific implementation.
  - Commenting will answer questions without future maintainers guessing and second-guessing your reasons and allow them to determine if the implementation is still the best implementation as the application evolves and business requirements are refined or changed over time

Commenting Code Discussion (con't)

- Not all classes need the same level of commenting
  - DTOs do not require much commenting, just a sentence or two that relates it to the other code
  - Implementations (like the 3 Password History classes in the previous example) usually should have more bytes of comments than code
  - Level of commenting in other classes vary

Commenting Code Discussion (con't)

Comment Content should be

- What you intended to implement (the code shows what you *did* implement)
- Explanations of how tricky and clever bits of logic work (for people not as smart)
- What User Stories have the requirements you're implementing
- Design Decisions (for example: you chose HashMap over TreeMap and LinkedHashMap)

Err on the side of too much commenting

Commenting Code Example

I have a simple one class that illustrates the point.

This is a quick "down and dirty" app that is meant to run from Eclipse.

Evaluate the **MontyHallProblem_NoComments.java** version of the class first and then evaluate the **MontyHallProblem.java** code.

GitHub Website: https://github.com/mgeiser/MontyHallProblem

Git Repo: https://github.com/mgeiser/MontyHallProblem.git