


Revised Authentication Design

Documentation related to

 **NDS-1074** - Authentication design **RESOLVED**

Background

We currently support a simple local authentication process. User accounts are stored in etcd with apr1 hashed passwords with associated per-namespace secrets. The apr1 format was used to support the NGINX ingress controller, which uses htpasswd files to enable basic authentication into services running in Kubernetes. The current authentication approach is undesirable for a variety of reasons, including double sign-in (custom login screen and basic auth) and lack of support for SSO. We've long discussed supporting Oauth, but didn't have a driving use case (until now). This documentation supersedes any other Oauth2 discussions (e.g., WSO2, etc).

We have three (or four) basic requirements for authentication/authorization going forward:

1. Oauth: We've been asked to support InCommon authentication and no longer store usernames and passwords. This is primarily for the public beta system, but would also be useful for hosted systems.
2. Local: We still need the ability to quickly provision accounts in demo and hackathon systems. Ideally we can eliminate the "basic auth" scenario
3. LDAP: The TERRA-REF use case is driving the need for tighter integration with local LDAP systems, specifically using common UID/GID for shared filesystems.
4. External: Not a clear priority, but the ability to integrate with other external authentication systems (e.g., Clowder)

Use cases

Oauth2 authentication (Globus auth)

User accesses www.workbench.nationaldatascience.org and selects "Sign-in" using Oauth (Globus). If user is not authenticated, user is redirected via Oauth. Once authenticated, if user has no account, an account record is created. No email verification is required. If approval is enabled, account goes through approval workflow. User starts Jupyter notebook. When user accesses Jupyter endpoint, if already authenticated they are not prompted to authenticate again. If they have not authenticated, user is prompted to authenticate using their Workbench credentials. If another user tries to access this user's Jupyter notebook, they are not permitted.

Local authentication ("cauth")

User accesses www.workbench.nationaldatascience.org and selects "Sign-up". User enters registration information and submits. User is required to verify email address. Once verified, account goes through approval process (if enabled). Once approved, user can login to Workbench to start services. User starts Jupyter notebook. When user accesses Jupyter endpoint, if already authenticated they are not prompted to authenticate again. If they have not authenticated, user is prompted to authenticate using their Workbench credentials. If another user tries to access this user's Jupyter notebook, they are not permitted.

Notes:

- We've demonstrated the "cauth" capability – using the NGINX ILB external auth with a secondary login component to eliminate the basic-auth paradigm
- This would allow us to move to a better password hashing strategy and no longer require double-login

NCSA LDAP

User accesses www.workbench.nationaldatascience.org and selects "Sign in". They are prompted to enter their NCSA LDAP credentials. If they do not have an account, they can sign up via NCSA identity. When they sign in (similar to oauth), they will go through the approval workflow, if configured. The story is otherwise the same (single sign-on for Workbench services and containers, unable to access resources in another namespace/account).

Admin user

An admin user can login and access grafana/kibana/dashboards.

CLI user

User – particularly admin user – can login via CLI. Will likely require API token or spoofing oauth workflow via CLI.

Possible implementation

- Create custom authorization component (similar to cauth) that can be used as first-level authorization. This component would control whether to use the standard workbench authentication or oauth and know how to handle the Workbench auth token. If a token exists and is valid for the user /namespace, the user would be authorized to access the requested resource. If the token doesn't exist, the user would be routed through the authentication flow (i.e., /oauth2)

- Under this approach the "cauth" module would proxy all requests to the oauth2_proxy and need to be able to handle the /oauth2 endpoint calls.
- Create an account/token creation component that is downstream from the oauth2_proxy. This would receive the oauth2 headers (user, email, token) and be responsible for account creation/update and creating/setting the token cookie.
 - This may be the same as the cauth component, but would be "chained" as an upstream to the oauth2_proxy.

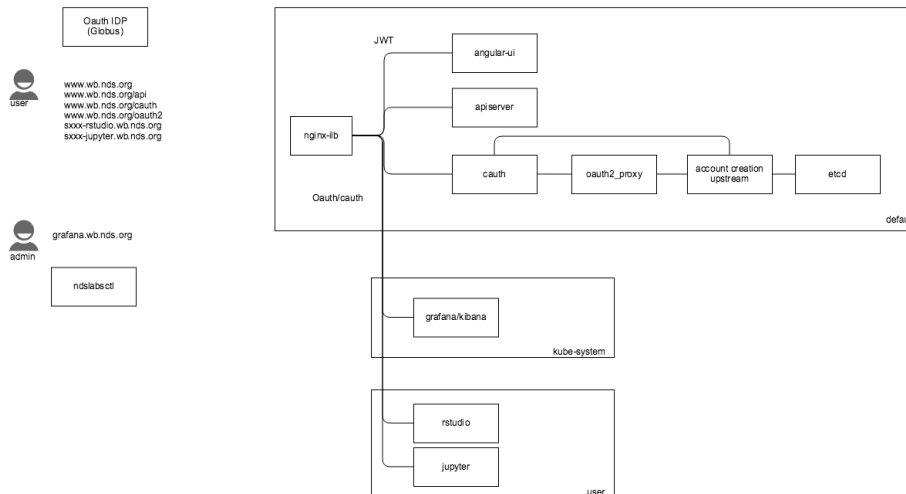
Notes about oauth2_proxy

- There is a working example of oauth2_proxy working with nginx ILB here: <https://github.com/craig-willis/oauth2-k8s>
- OAuth2 information (token, username, email, etc) are only available to configured oauth2_proxy upstreams.
- The upstream response codes are ignored (e.g., 401/403), so this cannot be used for authorization
- The upstream can set a cookie (i.e., token used by other applications)
- The upstream must have it's own path (i.e., something other than the already-taken start, sign_in, auth endpoints).

Other considerations/questions:

- Need to consider signup/approval process
- Namespaces can no longer be based on username (need unique ID and can use labels instead)
- Need to handle updated information from IDP (what if I change my email address?)
- Authorization must be namespace aware – can't let other users access my services
- Account record will change, for example: idp=globus, id=globusid, email=email, namespace=unique)
- Need to consider protected v unprotected routes (what needs auth/oauth, what doesn't)

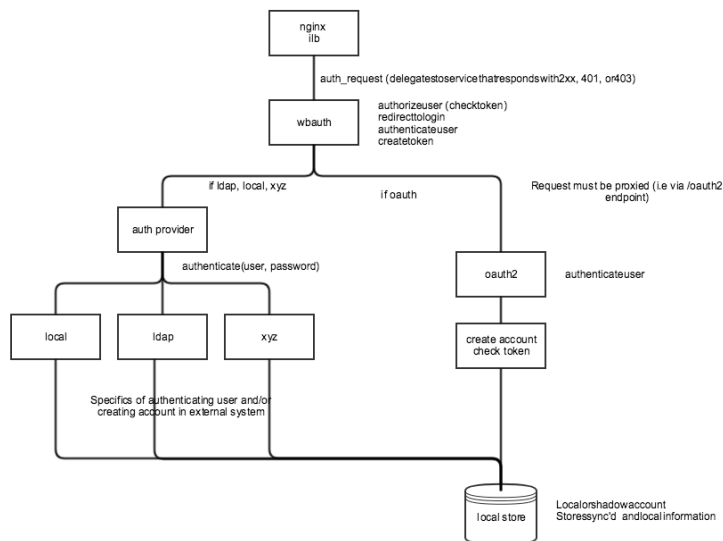
Overview



Authentication/authorization component

The wbauth component handles the nginx auth_request and simply returns 2xx, 401 or 403. It can set and check for the presence of a cookie (e.g., wb_auth) and delegate to another configured authentication module.

- wb_auth cookie
 - Same as JWT
 - Encodes user
- Checks for the presence of wb_auth cookie (e.g., token)
 - If cookie present
 - Confirms that token is valid
 - Confirms that user has access to requested resource
 - e.g., sxxx-studio – must lookup stack ID and confirm user has access to namespace
 - If valid
 - returns 200 (updating token)
 - If not valid
 - Returns 403
 - If cookie not present
 - Delegates (proxies) request to supported auth methods



LDAP provider

The LDAP provider would support authentication using an existing LDAP server, such as NCSA LDAP.

- wbauth is configured to use LDAP provider
 - server: ldap.ncsa.illinois.edu
 - port: 636
 - use SSL
 - user base dn: ou=People,dc=ncsa,dc=illinois,dc=edu
 - search filter : (&(objectClass=inetorgperson) ((!(memberOf=cn=prj_isda,ou=groups,dc=ncsa,dc=illinois,dc=edu) (memberOf=cn=isda_external,ou=groups,dc=ncsa,dc=illinois,dc=edu))))
- Need to define what it means to sign-up for workbench or if this is basically the oauth scenario (i.e., anyone who logs in is signed-up, pending approval). Options include:
 - LDAP configured with allowable groups only, members of the group are allowed in.
 - Configurable sign-in URL (<https://identity.ncsa.illinois.edu>)
 - We could have a sign-in page to push the user through the usual approval process. This would overcome the "opensource" problem where you can sign-up for an opensource account but need to know to ask to be added to JIRA, HipChat, etc. In this case, the user still signs in with their NCSA identity, but it pushed through an approval process.
- Instead of storing the password in etcd, we would delegate authentication to LDAP. This would allow us to synchronize information (e.g., groups) at each login
- Could also follow the pattern of automatically sync'ing LDAP accounts (ala Atlassian or OMD)
- Using the posixAccount object, we can optionally get UID/GID information and use this to run containers. The runAsUser feature appears to work without question. However, groups are a different problem. The fsGroup feature appears to be limited to certain storage options (e.g., not hostPath), so would force us to move to another storage model (which we want, anyway).