openMDM™ 5 Authentication and Roles

Canoo Engineering AG

(Contact: Sibylle Peter)

Situation

The openMDMT[M] Working Group recognized the need to provide user authentication and user authorization as part of the openMDM[TM] 5 framework.

An initial workshop on user authentication and authorization in the context of openMDM[TM] was held in April 2016. Its results are documented at https://openmdm.atlassian.net/browse/ORGA-98.

Objectives

Objectives – User Authentication

- Integration with existing enterprise authentication mechanisms and standards (such as PKI and Kerberos for example).
- When LDAP or Active Directory are available, use the information provided by those systems instead of maintaining independent copies of the information.
- When single sign-on mechanisms are available (such as Kerberos), openMDM™ must integrate seamlessly, without forcing the user to present their credentials once again.
- When the authenticated entity must be passed form openMDM™ to external systems then established standards for identity propagation must be used, such as CSIv2/SAS, SAML, JWT (where appropriate).
- The identity (trace) of the creator of an openMDM™ object must be stored in relationship with the object. Such identity must be stored in a form that allows reconstruction even when the identity has been removed from the central authentication database (i.e, the employee has left the company).

Objectives – User Authorization

- Support a base data container (known as “data pool”) on which roles apply to all openMDM™ objects related to it (such as Measurements, Tests, etc).
- Access rights must be assignable per data pool. If the data pool is physically spread dover multiple servers then those rights must be honored by all servers.
- Access rights must be primarily handled by roles, allowing the concepts of user groups. In exceptional cases access rights may be assigned to individuals.
- Access rights must be stored along with the data entities to which they pertain.
- Access checks must be performed at the data server level. (This is to ensure that access checks cannot be bypassed by login into the data server directly).

Design

Design Guidelines

- Data must be protected at the source, i.e., either directly at the data store level or within a service layer that completely encapsulates the data store and cannot be bypassed.

- Data protection must work application-independent, i.e., must produce the same results irrespective of the specific application used to access the data. If access to a particular data item is permitted, it should be visible in every application designed to work with that data; if access is not permitted, the data item must not be exposed through any application.

- Any action on protected data must be attributable to a specific entity ("principal") that can be held responsible for the action. The principal can be a human or (in the case of batch processing) a program.

Design Guidelines

- Data must be protected at the source, i.e., either directly at the data store level or within a service layer that completely encapsulates the data store and cannot be bypassed.

- Data protection must work application-independent, i.e., must produce the same results irrespective of the specific application used to access the data. If access to a particular data item is permitted, it should be visible in every application designed to work with that data; if access is not permitted, the data item must not be exposed through any application.

- Any action on protected data must be attributable to a specific entity ("principal") that can be held responsible for the action. The principal can be a human or (in the case of batch processing) a program.

Solution

Solution – User Authentication

- Use existing authentication system already available at the deploy location (such as LDAP, AD, Kerberos) as long as the openMDM™ requires network access and remote data access.

- In cases where the openMDM™ application is deployed on a standalone fashion or used in offline mode then the local authentication mechanism (such as operating system login) is enough.

Solution – User Authorization

- In cases where the openMDM™ application is deployed on a standalone fashion or used in offline mode then the user may have access rights to every operation the application provides. However access rights must be checked when the application connects with a remote server or attempts data transfer.
- For all other cases authorization may be handled in 3 different approaches

  - Native
  - Hybrid
  - Delegate

User Authorization - Native

- Defines an openMDM™ specific Access Rights API and implementation of that API (i.e., of administration functions and runtime access checking) within openMDM™ as a library or infrastructure component.

- This means that all security checking is carried out at the openMDM™ level and not in the underlying data server (e.g., the ODS server).

- openMDM™ would access the data server with superuser privileges to bypass all checking within the server itself and would then carry out its own access checks according to the access rights defined at the openMDM™ level.

User Authorization - Hybrid

- Define an openMDM™ specific Access Rights API, but implementation of that API is delegated to the data server.
- This means that openMDM™ would provide a common API to view and administrate access rights, but each server adapter would map calls to this API into calls to the security API of the underlying data server.
- For example, an ODS adapter would map the assignment of a particular access right to a particular group via the openMDM™ API into the creation of a corresponding ACL entry in the ODS server. The actual access checks would then be performed by the data server.

User Authorization - Delegate

Complete delegation to the data server.

This means that openMDM™ provides neither an API nor an implementation for access rights management, but simply relies on the access checks carried out by the data server.

In other words, access rights are administrated and checked at the data server level exclusively; openMDM™ simply passes the current user identity along with each server call and relies on the data server to expose only the data items the current user is entitled to see.

Pros / Cons

User Authorization - Native

- PROS
  - Clean room design and implementation.
  - Integration with existing authentication mechanism need to be implemented once at the openMDM™ level, not at every data backend level.

- CONS
  - Major undertaking with all stakeholders in order to design the model (like it happened with ODS for example).
  - Direct access to the data level is forbidden, thus all existing tools must migrate to openMDM™ as soon as possible.
  - Access rights cannot be used for optimized queries at the data level.

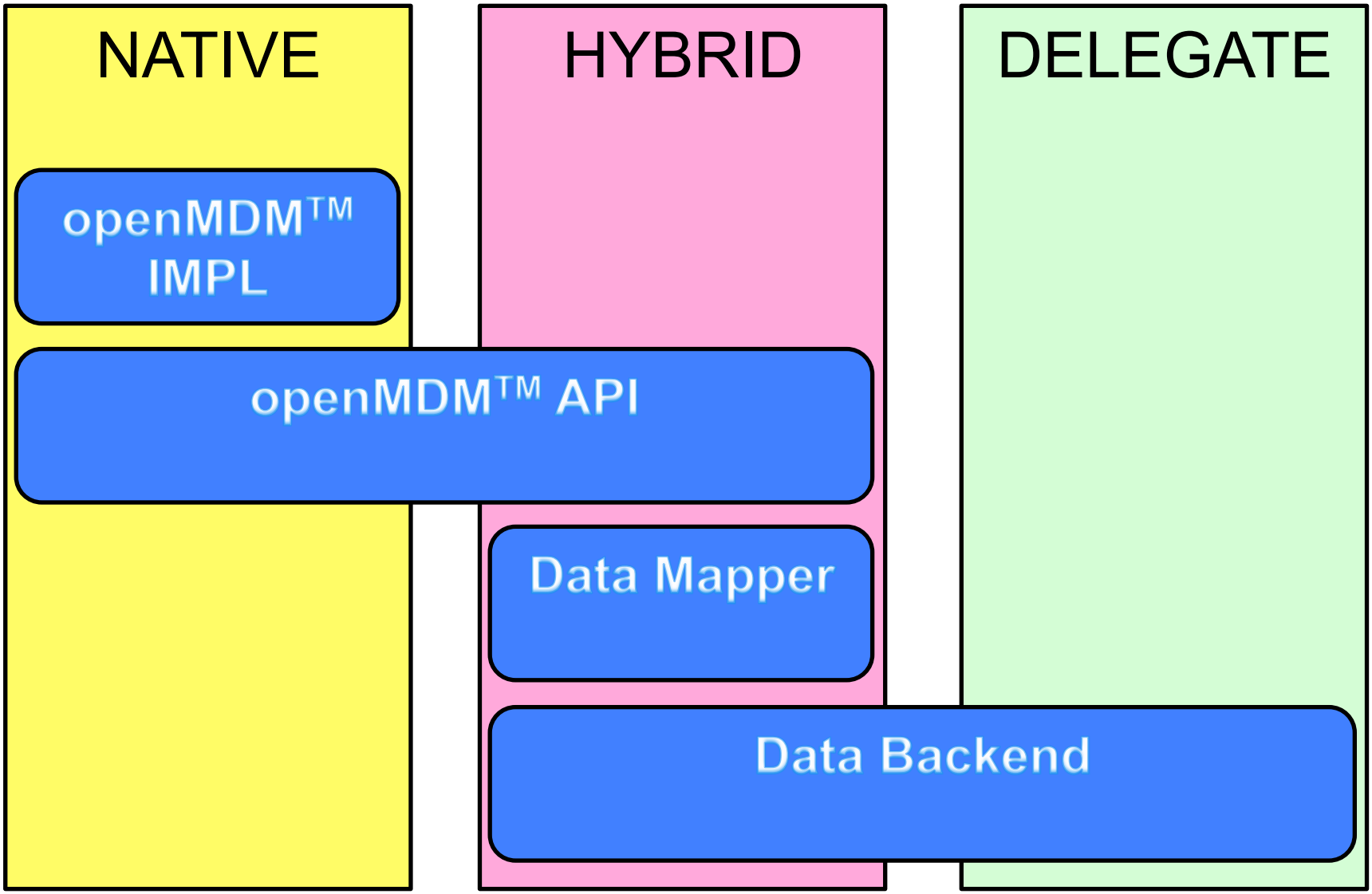
User Authorization - Hybrid

- PROS
  - Clean room design.
  - Data protection occurs at the source (data level) thus openMDM™ and non openMDM™ applications can rely on it.

- CONS
  - Mapping of security model to every data backend (ODS, PAK Cloud, etc). This include interfaces with existing enterprise authentication mechanisms.
  - The API must be designed as the least common denominator between existing data backends. Backend specific extensions must be designed and implemented.

User Authorization - Delegate

- PROS
  - Data protection occurs at the source (data level) thus openMDM™ and non openMDM™ applications can rely on it.
- CONS
  - Absence of a coherent openMDM™ security model as each data backend implements security in its own way.
  - Integration with existing enterprise authentication mechanisms must be executed by each data backend.


NATIVE
HYBRID
DELEGATE
openMDM™ IMPL
openMDM™ API
Data Mapper
Data Backend

Recommendation

Delegate

This is the path of least resistance, as it only requires propagation of the user identity to the data server on each call.

Starting with Delegate does not exclude the Hybrid approach, as delegation (via mapping) must happen as well.

Only the Native approach requires a separate design and implementation. It also requires a major undertaking by all stakeholders to agree on design.

This time and cost-wise (in the short term) Delegate is the preferred approach on which Hybrid can be built later. Native makes sense when all stakeholders and participants can move to openMDM™ together.

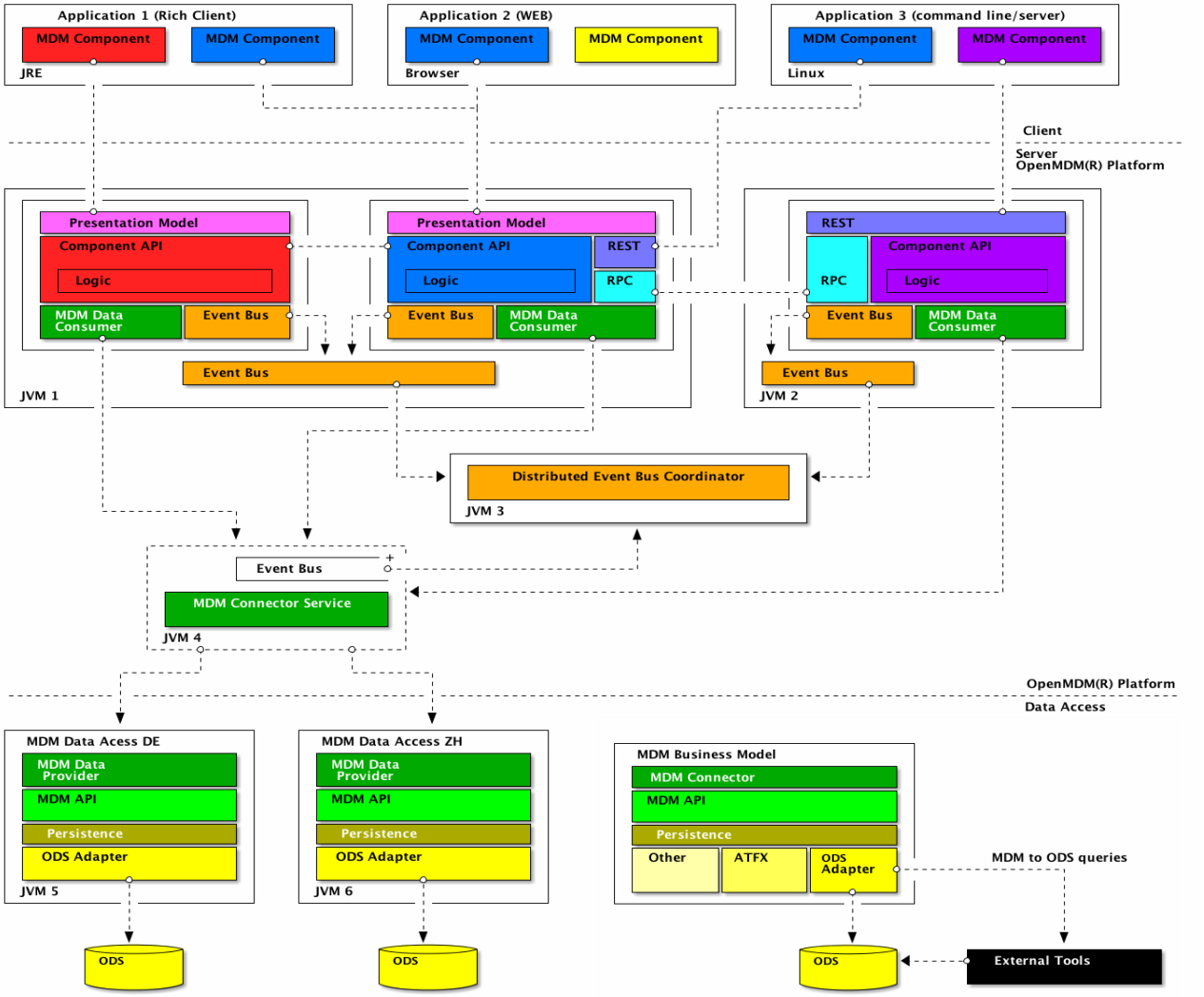
Delegate

The openMDM™ API requires two changes for this proposal to work:

- A login request that can be forwarded to the backend. The result of this request is either an error or an identity token.
- All other requests (such as queries, mutators) must include the identity token as part of its arguments. If the token is sent in the request header, no API change is needed.

In the case of a **search request**, the identity token could be used to reconstruct the real identity attached to an openMDM™ object, allowing faster and “native” searches at the backend level, instead of a two-pass search & filter alternative at the openMDM™ API level.

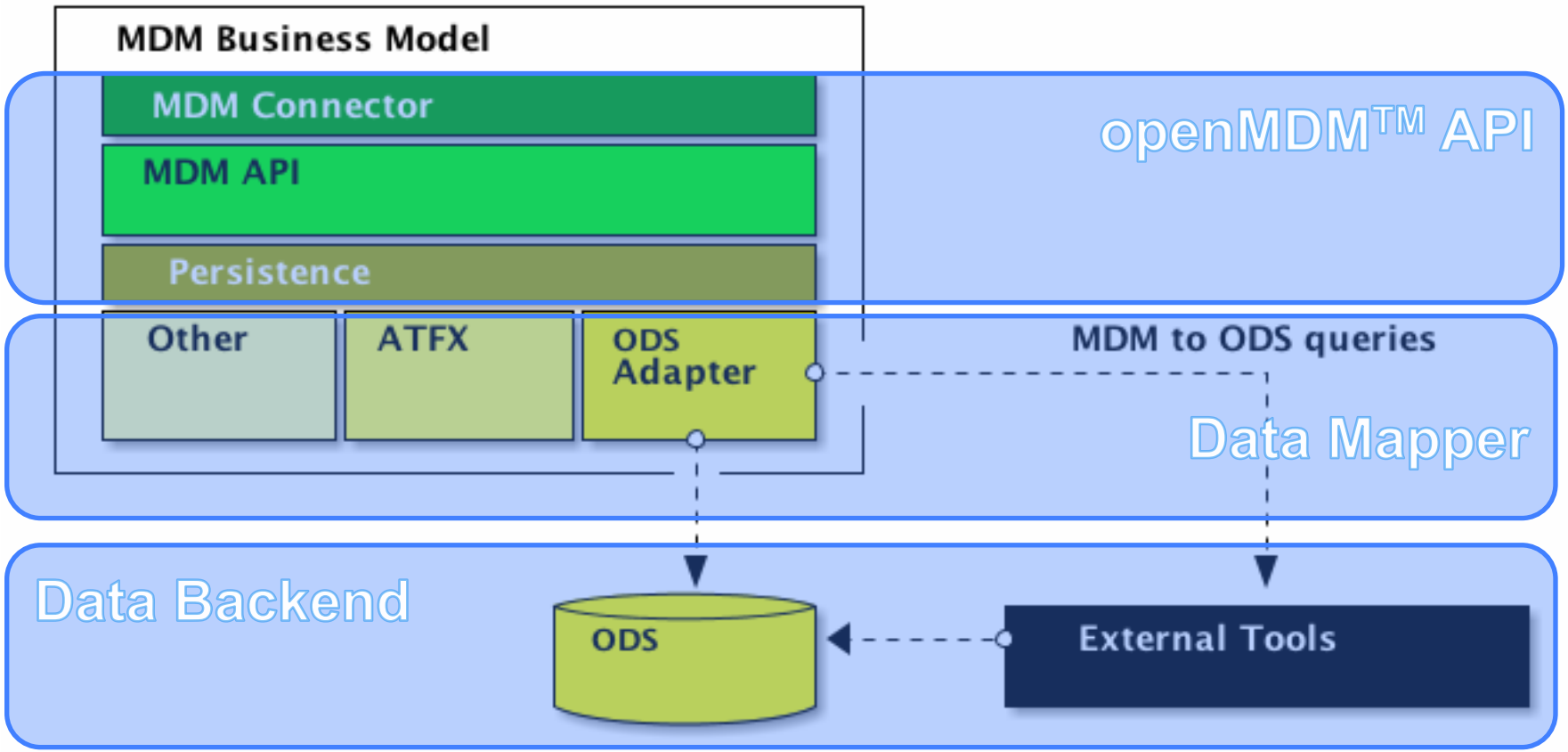
A possible distributed OpenMDM 5 system

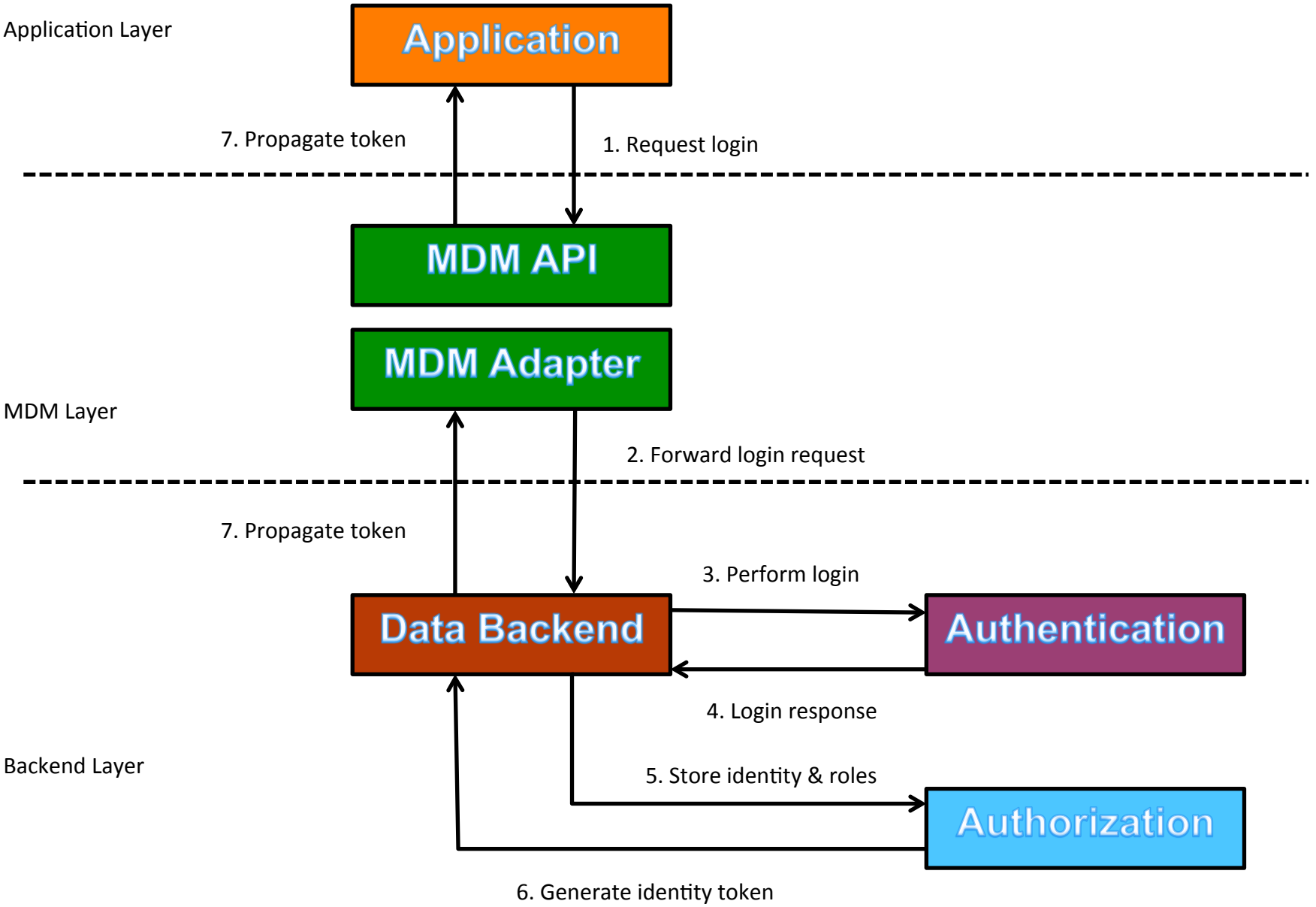


[delivering end-user happiness]

canoo

Aggregation of components to simplify…

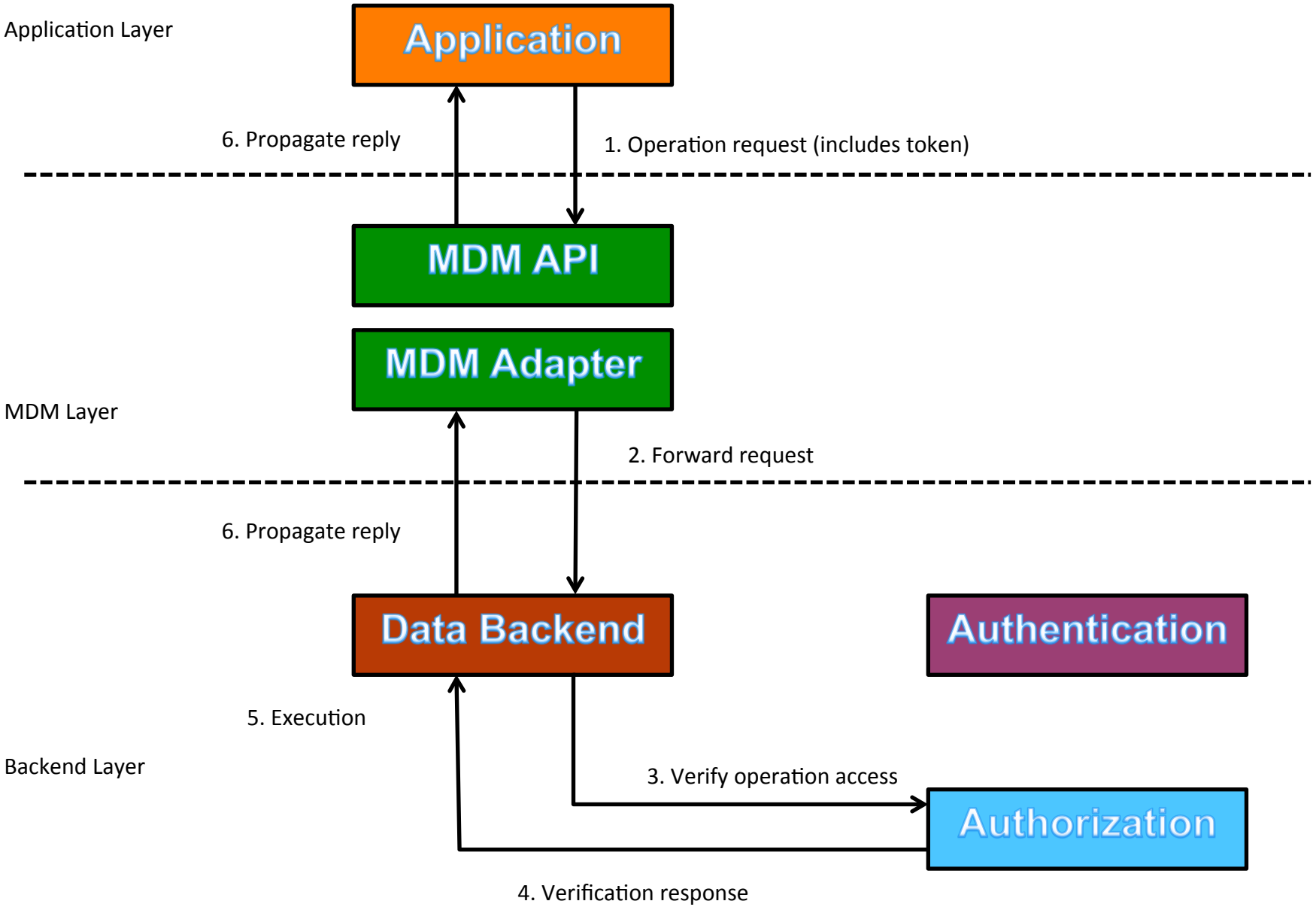



Application Layer
Application
7. Propagate token
1. Request login
MDM API
MDM Adapter
MDM Layer
2. Forward login request
7. Propagate token
3. Perform login
Data Backend
Authentication
4. Login response
Backend Layer
5. Store identity & roles
Authorization
6. Generate identity token

User Authentication– Data Flow

1. Application issues a login request.
2. The MDM API forwards the login request to the MDM Adapter.
3. The Data backend (ODS for example) receives the request and forwards it to the Authorization module (LDAP/AD/etc).
4. Authorization module either accepts the login credentials and returns an identity with roles or fails.
5. If login was successful then identity and roles are stored temporarily (think of a session).
6. An authentication token is generated. The token must be used in all other incoming requests in order to grant access to operations.
7. The token is forwarded all the way to the application.

Tools such as Matlab skip the MDM layer of course.



Authorization Delegate

[delivering end-user happiness]

canoo

User Authorization – Data Flow

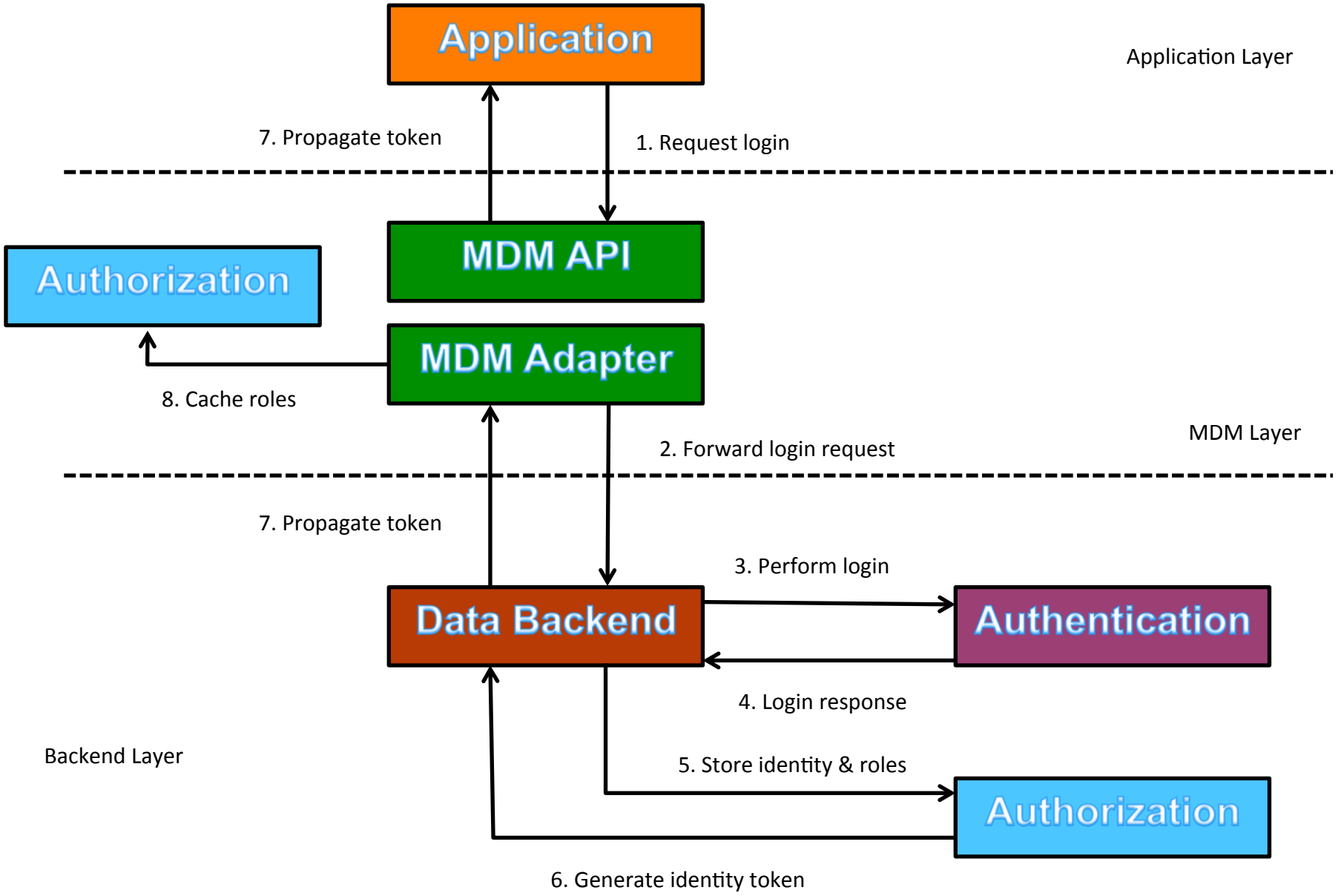
1. Application issues an operation request (store data for example). This request contains the authenticated identity token.
2. The MDM API forwards the operation request to the MDM Adapter.
3. The Data backend (ODS for example) validates the identity token and checks access rights.
4. Valid token and correct access rights grants green light to the operation to continue. Failures result in denied access; possible authentication workflow if no token or expired token.
5. Execute the requested operation.
6. Propagate results to the application.

Comparison of Hybrid Approach

Hybrid

- Can reuse code and components built for the Delegate solution.

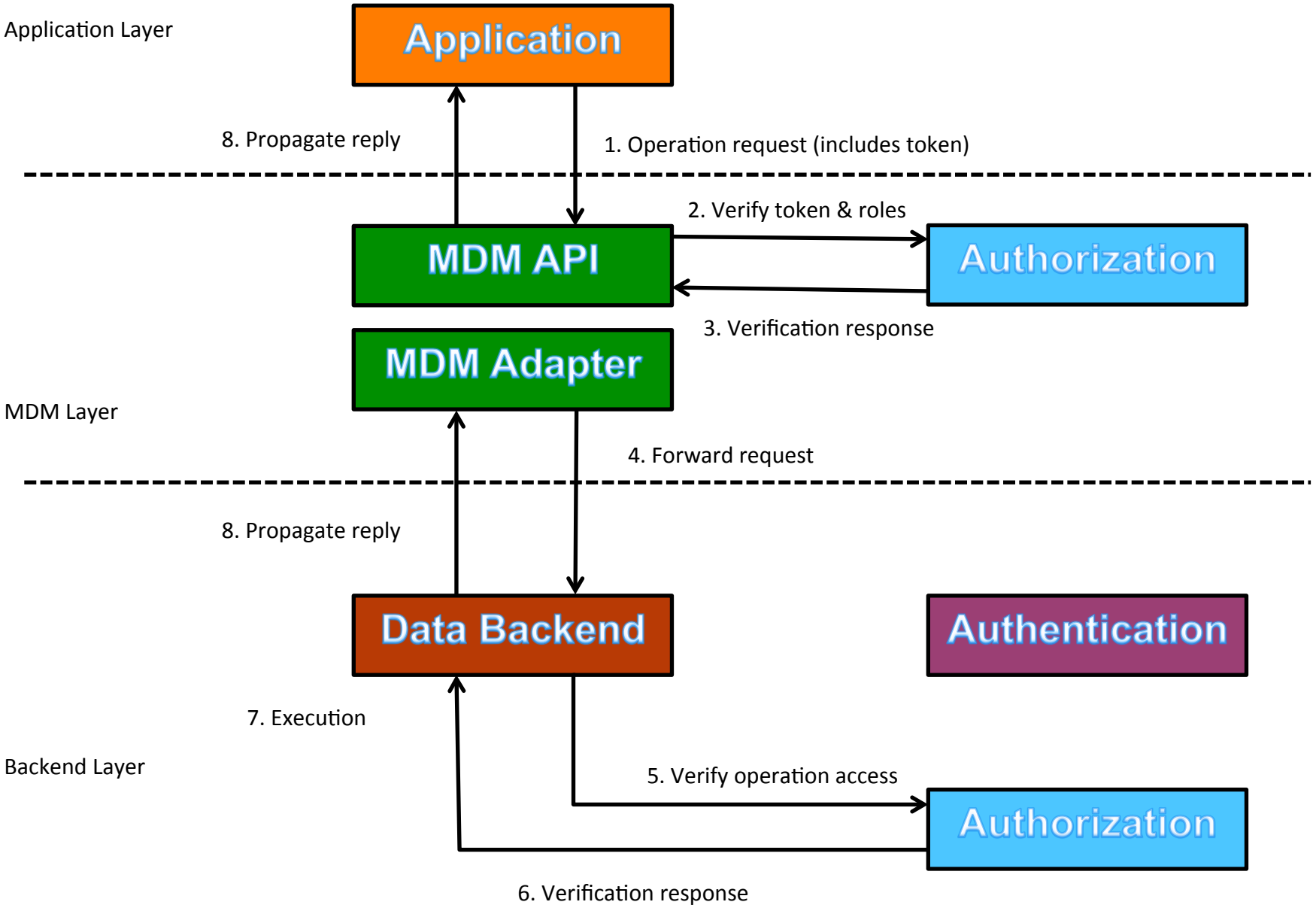
- Adds a new API on top of the openMDM™ API to cache, validate, and verify the security token and roles & rights.



Authentication Hybrid

[delivering end-user happiness]

canoo


Application Layer
Application
8. Propagate reply
1. Operation request (includes token)
2. Verify token & roles
MDM API
Authorization
3. Verification response
MDM Adapter
MDM Layer
4. Forward request
8. Propagate reply
Data Backend
Authentication
7. Execution
Backend Layer
5. Verify operation access
Authorization
6. Verification response

Required API Changes (Hybrid Approach)

- API changes like delegate approach

+

- openMDM™ API requires new types to handle Identities and Roles, alongside a Verification module/service.

  → All requests (except login) must be handled by the Verification module to check access rights before sending down the stack through the MDM adapter.