Technical Challenges of CDR Compliance

Version 1

Voruna Pty Ltd

CDR Compliance, Simplified.

voruna.com.au

Contents

1.0 Introduction and Overview

Key Objective: provide technical decision-makers of non-bank lenders (NBLs) an insight into the technical requirements of CDR compliance, with a focus on implementation of the data-sharing API.

The Consumer Data Right (CDR) is an Australian regulatory framework empowering consumers to control and share their data. Initially launched in banking and then energy, CDR is now extending to non-bank lenders (NBL). This means that **NBLs will become "Data Holders" (DHs) under CDR**, required to securely share product and customer data with accredited third parties at the consumer's request and consent. With the earliest obligations materialising mid-2026 – see our whitepaper, Obligations and Opportunities of CDR for more – **the clock is ticking** for NBLs to plan, acquire resources, and execute on a strategy to meet these requirements.

Failure to implement CDR properly carries serious risks. Regulators have already demonstrated zero tolerance for non-compliance – for instance, a major bank was fined over $750k for failing to correctly provide required data (omitting credit limit information) (ACCC, 2025). Beyond fines, poor CDR implementation can damage a lender's reputation and cause customer attrition as consumers and fintech partners lose trust in unreliable data services. On the positive side, a well-executed CDR program can enhance customer trust, enable new partnerships, and keep lenders competitive in a data-driven finance landscape.

Implementing CDR is more than a compliance checkbox – it requires coordinated technical systems, architectural foresight, and rigorous execution. This paper outlines the core components required for CDR compliance, with a particular focus on the design and implementation of the data-sharing API. In doing so, it highlights critical architectural decisions and offers practical insights to help non-bank lenders build a future-ready CDR capability.

This whitepaper is intended for the Chief Technical Officer and other key technical decision-makers of non-bank lenders who are planning their participation in the CDR.

2.0 Key Technical Systems

Participating in CDR as a Data Holder requires a multi-faceted technical implementation. In this section we'll discuss the key systems and some of their complexities.

2.1 OIDC-Compliant Authorisation Server

CDR security profiles require each Data Holder to operate an **OAuth 2.0/OIDC server** for authorising data sharing. When a Consumer wants to share data through a Data Recipient, they will be redirected to the Data Holder's authorisation service. The Consumer authenticates (e.g. logs into the internet banking or portal) and consents to specific data scopes through a CDR-compliant consent flow. The OIDC server then issues an authorisation code and tokens as per the standard. Implementing this involves integrating with the existing authentication system - this is more straightforward if there is an existing OIDC infrastructure, though CDR-specific requirements like Financial-grade API (FAPI) compliance may still require additional work. Lenders with no OIDC expertise might consider outsourcing this component, given its criticality and complexity. Those with sufficient internal resources should involve their security architects early to stand up a secure authorisation service that meets the CDR **Information Security Profile** standards (DSB, 2025 version) (which specify requirements like MTLS, JWT signing, encryption algorithms, etc.).

2.2 Consent Model

Each Data Holder needs to build out both parts of the Consent Model, being the Consent Flow and the Consumer Dashboard. These are the only DH systems that a Consumer directly interacts with as all other requests related to consents and their data are made by a Software Product via a Data Recipient. As such, access is usually through an existing portal/app and company branding is important. They must adhere to a sizeable list of Customer Experience standards that cover areas such as language, authentication flow, sending notifications, accessibility and user interfaces (DSB, 2025 version). From a data integration perspective, implementing these systems may involve extending an existing customer database with new consent tables, or creating a separate, dedicated **Consent database**.

The Consent Flow allows a Consumer to share their data with a Software Product. I.e., a customer of "Lightning Loans Co." can share their data with "LenderList App" to see if there's a better loan option for them. This process requires the Software Product (LenderList App) to redirect their authenticated user (the Consumer) to the start of a Data Holder's (Lightning Loans Co.) Consent Flow. The flow begins with the Consumer authenticating with the Data Holder's OIDC authorisation server, and CDR mandates that this must involve using a One-Time Password sent via email or mobile. The authenticated Consumer then selects the parts of their data that they wish to share, known formally as scopes, and agrees to the period for which the data is shared. To finish the flow, the user is redirected back to the Software Product where they can now access their shared data. All records of a Consumer's interaction with this flow are stored in the consent database.

The Consumer Dashboard is the second part of the Consent Model where a Consumer can view the consents they have granted to different Software Products and revoke any active consents at any time. This triggers a revocation event to be sent to the Data Recipient who is required to delete all related data such that it can no longer be requested by the Software Product. The CDR ecosystem actually involves two Consumer Dashboards: one by the Data Holder and the other by the Data Recipient. So similarly, if a Consumer revokes a consent through a Data Recipient's Consumer Dashboard, the Data Holder will receive a notification and must immediately ensure that data can no longer be requested. Consumers also have access to consent dashboards through each Data Recipient.

2.3 API Layer

Each Data Holder must implement the set of **standard APIs** defined by the Consumer Data Standards for Non-Bank Lending (DSB, 2025). This can be a considerable undertaking as it necessitates the mapping of existing data to the standardised CDR schemas. This includes endpoints for authenticated, Consumer-specific data such as account data, transaction data, customer profile data, etc. It also includes unauthenticated endpoints for product data, as well as metadata like **GET Status**, **GET Outages**, and **GET Metrics**, which report system availability and performance.

Data Holders must also ensure strict adherence to the JSON schemas and URL structures defined in the CDR standards – the uniformity and accuracy across all data holders is core to the purpose of CDR and non-adherence can be penalised. There are also non-functional requirements which include performance requirements such as **response times under 1000ms** for high priority endpoints. When using a compliance service for the infrastructure components of this system, Data Holders will still need to map internal data structures to those required by the compliance service as well as ensure that their systems meet the non-functional requirements. **The bespoke mapping and provision of data is Voruna's expertise**; we specialise in translating complex data models into CDR-compliant APIs with guaranteed performance and accuracy.

3.0 API Architectural Design Strategies

This next section will delve deeper into the most complex technical system, the API layer. This is the least generic system as it depends directly on the structure of a Data Holder's existing data and thus there is no one-size-fits-all architectural solution. One of the biggest technical challenges that should inform the system design, is **preventing it from impacting on core business operations**. The APIs must serve requests from multiple Data Recipients, and handle high continuous load, as well as respond to large spikes. There is a **significant risk of overwhelming core databases** if the API layer is not appropriate designed and implemented; either limiting accessibility to core systems, or at worst, bringing them down entirely. This section will explore solutions to this challenge and discuss their merits and weaknesses.

3.1 Larger Servers and Databases

The first strategy for overcoming this issue is to simply build an API that directly accesses the core databases and APIs. While this lacks complexity, this strategy adds the highest load to core systems and is highly susceptible to failing the non-function requirement of a fast response time and high availability. It is almost always the case that a single CDR endpoint must fetch data from several different database tables, if not several separate databases or systems. Under this strategy, every source of data must be able to return its data fast enough for the data to be compiled and served, taking into account the overhead for authentication and handling of the request and response. This means a single weak link in the system, which could simply be a core system that already has high traffic from the Data Holder's core software, can see this strategy completely fail the performance requirement. It also means all data sources must be available at all times. For this strategy to be successful, all core systems would need to run on highly performant modern servers. It may require increasing the size and power of the current systems, i.e. scaling up the RAM and CPUs of existing databases and servers. This could have the advantage of being quick to implement and deploy for monolithic, cloud-hosted systems, as it may only require a few configuration settings to be changed. However, it's not necessarily a trivial configuration change and may not even be possible for certain types of data stores, e.g. third-party hosted CRMs with inflexible rate limiting or fixed hard drives in data centres. It will also incur a higher baseline operational cost, which could be substantial depending on the number and type of databases and servers. Clearly this strategy poses a serious risk of wasting development resources to build a non-compliant system.

3.2 Change Data Capture

The second strategy is to use Change Data Capture to track changes to core databases (for customers, accounts, loans, transactions) and **replicate those changes in real-time** to a secondary database, which we'll refer to as the CDR database. The CDR API is then built to only use this database to serve requests, which completely isolates the core systems from the additional load of CDR. Business-critical systems are protected from traffic spikes and the CDR API can be scaled in a more fine-tuned, cost-effective way. This strategy also means that any API endpoints that fetch data from multiple primary sources does not need to rely on their individual performances or them all being available at the time of the request, as the data for CDR requests is compiled in advance. The key to minimizing costs with this strategy is that the CDR database is kept to the bare minimum, instead of replicating entire core databases, it duplicates only the values that are required for CDR. This strategy does introduce the complexity of keeping the CDR database in sync with the primary data sources, which can be achieved using Change Data Capture. This approach proved successful in a major Australian bank's Open Banking implementation, where CDC enabled real-time data replication for CDR APIs serving over 400,000 customers **without requiring modifications to core banking infrastructure** (Ramesh et al., 2025). By capturing changes from existing systems and streaming them to the CDR database, they avoided disrupting critical operations.

The implementation involves establishing a data pipeline that monitors the production databases transaction log or uses event triggers for relevant tables (customers, loans, accounts, transactions). Each change event is placed in a queue, filtered, batched, and then applied to the CDR database. Many modern databases support CDC natively (e.g. SQL Server Change Tracking, Oracle GoldenGate, etc.), there are also open-source tools (Debezium, Kafka Connect) or cloud services (AWS DMS, Azure Data Factory, GCP Datastream) to stream changes. To optimise the pipeline, filtering the events is necessary so that they capture only what's needed for servicing the API. Not every column on every table is part of the CDR specification. Focusing on the minimum necessary data reduces noise and processing load. Filtering can either happen before the events are triggered (by only monitoring changes to certain tables) or while they are being processed in the queue. The process typically involves:

1. **Initial Data Load:** First, perform a one-time extraction of all existing in-scope data (all customers, open loan accounts, and at least two years of transactions as required by CDR regulations) from the core system. This establishes the baseline dataset in the new CDR database.
2. **Ongoing Change Stream:** Next, enable continuous CDC from that point forward. Each new transaction (loan drawdown, repayment, fee, etc.) or update (address change, account status change) in the core will generate an event.
3. **Data Mapping:** As events are received, their data must be transformed to the CDR schema before being saved into the CDR database. This way, the queries made by the API can be optimised for minimum response times.

The CDR database should be designed for **fast reads and high concurrency**. If using a relational database, proper indexing on customer IDs, account IDs, date fields (for transaction range queries) is critical. It can be supplemented by an in-memory cache for serving static or infrequently changing data. This could include data such as product reference data (like product offerings, rates, fees) and customer/account details which are often not changed after their creation.

The strategy of using CDC is far more resilient and cost-effective and has become the industry standard for CDR. For most Data Holders, it would be Voruna's recommended approach.

4.0 Non-functional Requirements

This section discusses a few of the main non-functional requirements. The full list can be found at the DSB website https://consumerdatastandardsaustralia.github.io/standards.

4.1 Ensuring High Availability

CDR infrastructure must be highly reliable – consumers and accredited data recipients expect to fetch data on demand at any time. The Consumer Data Standards include **strict non-functional requirements** for data holder APIs, such as a minimum **99.5% availability per month** (DSB, 2025). In practical terms, this

allows for at most roughly 3.6 hours of unplanned downtime in a month. Moreover, this uptime requirement covers both **authenticated endpoints** (customer data APIs that require a valid consent/auth token) and **unauthenticated endpoints** (such as public product data).

Data holders must architect their systems with redundancy and failover capabilities to meet this: for example, deploying APIs across multiple availability zones or data centers, using load balancers, and having backup instances to eliminate single points of failure. **Planned outages** (for maintenance) are permitted but must be reasonable in frequency/duration and communicated to ecosystem participants in advance; standards require publishing of this outage information via a "Get Outages" endpoint and giving at least one week's notice for normal maintenance windows (DSB, 2025).

4.2 Performance and Scalability

The CDR standards also define performance tiers for API responsiveness and assert that 95% of calls per hour must respond within these thresholds. High priority endpoints must respond within 1000ms, while most other endpoints have a slightly more lenient threshold of 1500ms and above. It should be noted that these response time thresholds apply regardless of API load, hence Data Holders must design for scaling: ensuring the API layer and database can handle spikes (for example, end-of-month when many loan statements might be retrieved). The CDR API must handle concurrent requests from potentially many data recipients, especially for a large customer base and as more fin-tech apps integrate. Load testing should be used to validate throughput and mitigate obligation breaches.

One helpful practice is to implement **API rate limiting and throttling** to protect your backend. Voruna's CDR gateway solutions include built-in rate limiting, monitoring, and performance optimization to ensure SLA compliance under peak load conditions. The CDR rules allow data holders to reject or defer requests if capacity is exceeded, but only within certain thresholds (DSB, 2025). Use an API gateway to enforce reasonable limits and queue excess requests rather than letting your system overload. An API gateway can also serve as a first line of defence, quickly rejecting invalid requests and caching frequent queries (as noted, caching can be employed at the gateway level too).

4.3 Data Latency and Quality

The requirements of low data latency and high data quality are poorly defined by the DSB but are critical to the effectiveness of CDR. Low data latency means that whenever primary data is changed, that change should be reflected as soon as possible through the CDR API. The rule of thumb being that CDR data should always match what is accessible through the Data Holder's core software systems (e.g. customer portal, banking app, etc). Clients have an expectation of low data latency, as they should not be privy to any caching or intermediary databases involved with CDR to satisfy the response time requirements. High data quality refers to serving product data that accurately matches what is published elsewhere by the Data Holder. The CDR API should not serve information about a particular product that does not match what is advertised on the Data Holder's website and product disclosure statements.

5.0 Next Steps

Becoming a CDR data holder introduces a range of technical, regulatory, and operational challenges, but with the right architecture and implementation strategy, non-bank lenders can turn compliance into a strategic asset.

Voruna specializes in helping lenders navigate this journey. From system design to implementation, our team brings deep expertise in CDR data architecture, integration pipelines, and regulatory alignment. Whether you're just beginning your CDR roadmap or validating an existing design, we can help you build with confidence.

Action items to begin your CDR journey:

- **Perform a gap analysis** of your current systems: What components do you already have (e.g., an API gateway, an OAuth server), and what needs to be developed or integrated?
- **Engage with CDR solution providers**: A partner like Voruna can offer reference implementations, testing environments, and integration templates to reduce time-to-compliance.
- **Build integration interfaces**: Ensure core systems can export or stream data to your CDR layer, especially if CDC tools are unavailable.
- **Update your architecture documentation**: Map the data and consent flows end-to-end to align teams and support audit readiness.
- **Test early and often**: Simulate data recipient scenarios to surface bugs or bottlenecks before going live.

Contact Voruna to schedule a discovery session. Make CDR compliance a competitive advantage – not just a regulatory hurdle.

6.0 References

ACCC. (2025). *National Australia Bank pays $751,200 in penalties for alleged breaches of Consumer Data Right Rules*. Australian Competition & Consumer Commission (ACCC). https://www.accc.gov.au/media-release/national-australia-bank-pays-751200-in-penalties-for-alleged-breaches-of-consumer-data-right-rules

DSB. (2025). *Consumer Data Standards*. Data Standards Body. https://consumerdatastandardsaustralia.github.io/standards

Ramesh, H., Sumitha, M., Srinivasan R., & Vargas, T. (2025). *How TCS unlocked Open Banking with Amazon Kinesis Data Streams*. https://aws.amazon.com/blogs/apn/how-tcs-unlocked-open-banking-with-amazon-kinesis-data-streams

Wijesekara, D. (2020). *Non Performance Requirements of Consumer Data Standards Specification : Open Banking in Australia*. Medium. https://dassanawijesekara.medium.com/non-performance-requirements-of-consumer-data-standards-specification-open-banking-in-australia-f947ee013578