Architectural Decision Records (ADRs)

UNSW SYDNEY

These lecture slides are from the books:

- "*Head First Software Architecture*", by Raju Gandhi, Mark Richards, Neal Ford, O'Reilly Media, Inc., March 2024
- "*Fundamentals of Software Architecture*", 2nd Edition, by Mark Richards, Neal Ford

Architectural Decision Records (ADRs)

- "Why is more important than how."
- Architectural decisions must be justified
- Future team members need to understand the rationale
- Without context, good decisions may seem arbitrary or incorrect
- Architectural Decision Records (ADRs) capture the what, why, and impact
- An ADR has seven sections: Title, Status, Context, Decision, Consequences, Governance, and Notes.
- Important aspects of an architectural decision are documented, including the decision itself.

ADR

ADR Structure Overview

Main Sections:

- Title: Numbered and concise
- Status: Proposed, Accepted, Superseded, or Request for Comments (RFC)
- Context: Forces and constraints
- Decision: What was chosen and why
- Consequences: Trade-offs and impacts
- Compliance: Governance and enforcement
- Notes: Metadata (author, date, approval)

ADR Section - Title

- Purpose: Identify and summarize the decision

- Best Practices:
  - Number sequentially (e.g., ADR 001)
  - Short, descriptive, and unambiguous

- Example:
  - “ADR 17: Asynchronous Messaging Between Order and Payment Services”
  - “ADR 21: Transition to PostgreSQL for Inventory Management”
  - “ADR 34: Enable OAuth 2.0 for Internal APIs”

ADR Section - Status

- Types:
  - Proposed: Pending approval
  - Accepted: Approved and active
  - Superseded: Replaced by another ADR
  - RFC: Open for feedback until a deadline

- Examples:
  - ADR 12: Status: Accepted
  - ADR 17: Status: Superseded by ADR 21
  - ADR 34: Status: RFC, Deadline 30 May 2025

ADR Section - Context

- Purpose: Explain what situation led to this decision

- Include:
  - Problem or force requiring a decision
  - Alternatives under consideration

- Examples:
  - “The Order service must transmit payment info. Options include REST or messaging.”
  - “Inventory updates are inconsistent across services; central DB vs. event-based sync considered.”
  - “Increased phishing attempts require re-evaluating access token validation approach.”

ADR Section - Decision

- Purpose: Describe what was chosen

- Best Practices:
  - Use clear, assertive language: “We will use...”
  - Justify with rationale

- Examples:
  - “We will use asynchronous messaging due to latency and decoupling benefits.”
  - “We will migrate inventory management to PostgreSQL to ensure consistency and performance.”
  - “We will adopt OAuth 2.0 using *IdentityServer* for access control.”

ADR Section - Consequences

- Purpose: Describe outcomes and trade-offs

- Include:
  - Positive and negative impacts
  - Known limitations

- Examples:
  - “Improves performance, however adds complexity in error handling.”
  - “Enables real-time updates; requires Kafka infrastructure.”
  - “Strengthens security, but introduces user reauthentication challenges.”

ADR Section - Compliance

- Purpose: Define how decision enforcement is measured

- Types:
  - Manual review
  - Automated tests (e.g., fitness functions)

- Examples:
  - Static code analysis rules for package structure compliance
  - Integration test that validates token expiration and renewal workflows

ADR Section - Notes

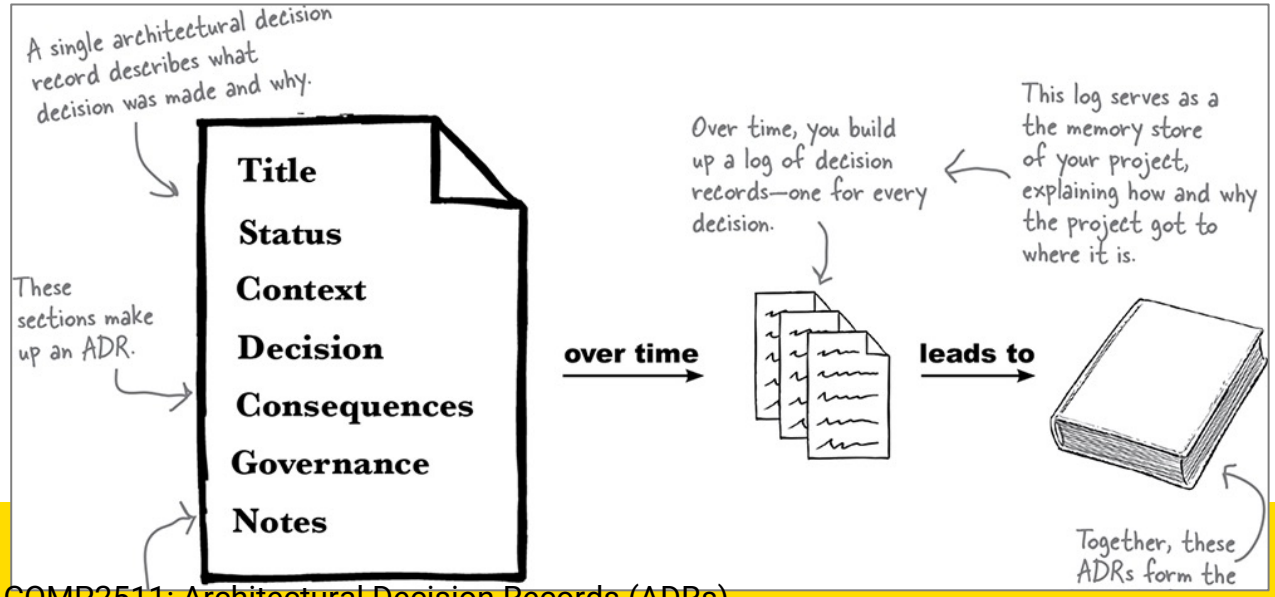
- Purpose: Capture metadata

- Typical Fields:
  - Author, approval date, approver
  - Last modified, superseded reference

- Examples:
  - Author: A. Johnson, Approved by: Arch Review Board, 15 May 2025
  - Author: M. Singh, Modified on: 10 May 2025, Supersedes ADR 12
  - Author: L. Chen, RFC Deadline: 30 May 2025

Benefits of Using ADRs

- Serves as a memory log for decisions
- Helps new team members understand context
- Improves consistency and governance
- Supports continuous evolution and learning



Example: ADR

Title

012: Use of queues for asynchronous messaging between order and downstream services

Status

Accepted

Context

The trading service must inform downstream services (namely the notification and analytics services, for now) about new items available for sale and about all transactions. This can be done through synchronous messaging (using REST) or asynchronous messaging (using queues or topics).

Decision

We will use queues for asynchronous messaging between the trading and downstream services.

Using queues makes the system more extensible, since each queue can deliver a different kind of message. Furthermore, since the trading service is acutely aware of any and all subscribers, adding a new consumer involves modifying it—which improves the security of the system.

Consequences

Queues mean a higher degree of coupling between services.

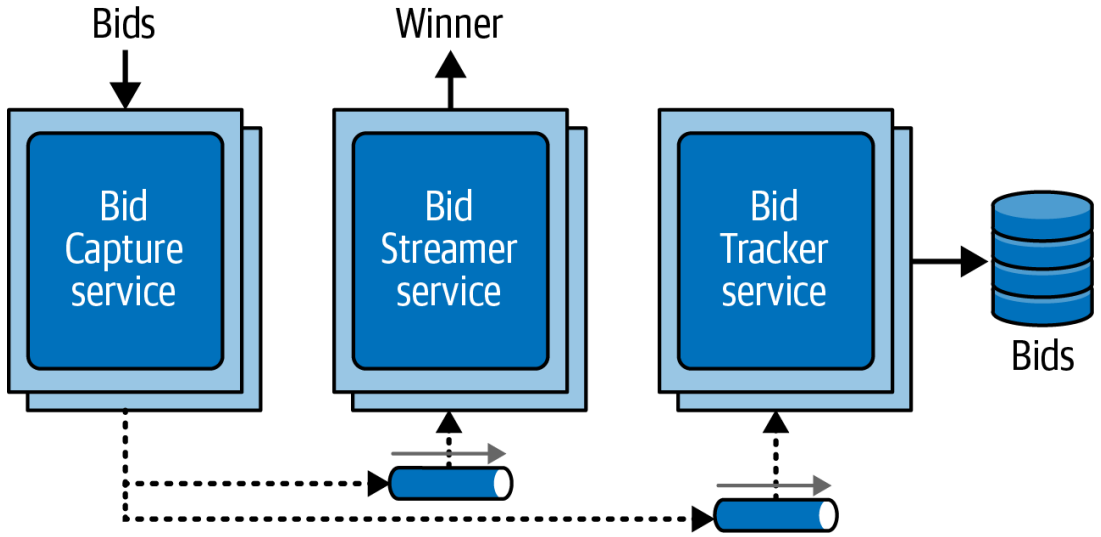
We will need to provision queuing infrastructure. It will require clustering to provide for high availability.

If additional downstream services (in addition to the ones we know about) need to be notified, we will have to make modifications to the trading service.

ADR – Auction System Example

Architectural Decision in the following auction system:

- use separate point-to-point queues between the bid capture, bid streamer, and bid tracker services **instead of** a single publish-and-subscribe topic (or even REST, for that matter)
- ADR needs to **justify the choice** to prevent confusion or disagreements among other designers or developers.



ADR – Auction System Example

ADR 76. Separate Queues for Bid Streamer and Bidder Tracker Services

STATUS

Accepted

CONTEXT

The `Bid Capture` *service, upon receiving a bid, must forward that bid to the* `Bid Streamer` *service and the* `Bidder Tracker` *service. This could be done using a single topic (pub/sub), separate queues (point-to-point) for each service, or REST via the Online Auction API layer.*

ADR – Auction System Example

DECISION

We will use separate queues for the `Bid Streamer` *and* `Bidder Tracker` *services.*

The `Bid Capture` *service does not need any information from the* `Bid Streamer` *service or* `Bidder Tracker` *service (communication is only one-way).*

The `Bid Streamer` *service must receive bids in the exact order they were accepted by the* `Bid Capture` *service. Using messaging and queues automatically guarantees the bid order for the stream by leveraging first-in, first out (FIFO) queues.*

Multiple bids come in for the same amount (for example, "Do I hear a hundred?"). The `Bid Streamer` *service only needs the first bid received for that amount, whereas the* `Bidder Tracker` *needs all bids received. Using a topic (pub/sub) would require the* `Bid Streamer` *to ignore bids that are the same as the prior amount, forcing the* `Bid Streamer` *to store shared state between instances.*

The `Bid Streamer` *service stores the bids for an item in an in-memory cache, whereas the* `Bidder Tracker` *stores bids in a database. The* `Bidder Tracker` *will therefore be slower and might require backpressure. Using a dedicated* `Bidder Tracker` *queue provides this dedicated backpressure point.*

ADR – Auction System Example

CONSEQUENCES

We will require clustering and high availability of the message queues.

This decision will require the `Bid Capture` *service to send the same information to multiple queues.*

Internal bid events will bypass security checks done in the API layer.
UPDATE: Upon review at the January 14, 2025, ARB meeting, the ARB decided that this was an acceptable trade-off and that no additional security checks are needed for bid events between these services.

COMPLIANCE

We will use periodic manual code reviews to ensure that asynchronous pub/sub messaging is being used between the `Bid Capture` *service,* `Bid Streamer` *service, and* `Bidder Tracker` *service.*

NOTES

Author: Subashini Nadella

Approved: ARB Meeting Members, 14 JAN 2025

Last Updated: 14 JAN 2025

Summary of ADR

- Each section contributes to clarity and traceability
- Together they provide context, rationale, and continuity
- Encourage consistent use across all teams and domains