Engineer

Architecture decision record

a short record of a design choice and its reasons

What it is

An architecture decision record, often abbreviated ADR, is a short account of a consequential technical choice. It says what the team decided, which conditions made the decision necessary, what credible options it considered and what consequences it accepted. The choice might concern how to store facts, where a system boundary lies, how teams integrate or what happens during a failure. A reader arriving later can use the record to understand why this design exists and which assumptions still apply.

A few fields carry most of the work. An identifier and a date make the record easy to cite. A short status indicates whether a decision is proposed, accepted or superseded. The context identifies the problem and constraints. The decision describes what the team will actually do. Consequences cover benefits and obligations, such as another service to operate, a slower transition or a migration that must be rehearsed. Include enough detail about options to explain the tradeoff without recreating the full discussion.

Use the same publication path reviewers already trust. A team may keep Markdown records beside its code and link them from the change that implements the decision. A team designing a service across several repositories may keep a shared index and reference the same record from each change. The record’s value comes from readable reasons and a stable address. Its template is a means of capturing those reasons consistently.

Use the concept in a decision

Put “What is an architecture decision record?” to work in a real decision. We can help relate the concept to the system you are building or buying. The shape of a system worked out on paper first, so the parts fit and the cost is known before anyone writes code.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. One infrastructure plan for 14 departments — 14 research departments on one infrastructure plan

One infrastructure plan for 14 departments Get in touch →

Why it matters

Code is good at preserving what exists. It explains the history only partially. A reader may see two databases and assume that one should be retired, when the split was a temporary measure to isolate a risk while a supplier changed its export. The written decision can clarify what the team was protecting, why the alternative was viable and what event makes it time to reconsider. That avoids repeating an investigation from scratch.

The record also helps people disagree productively. A proposal can say that a new dependency would spare the team from developing a scheduler but require an operator who can restore its state. A reviewer can question the cost or the availability of that operator. The tradeoff is visible to people outside the immediate implementing pair, including the team that will operate the service. Once accepted, the same account gives a handover reader a way to understand where responsibility belongs.

The time spent writing should match the effect of revisiting the choice. Changing a label normally belongs in the change history. Changing which service owns the ledger or how an application handles duplicate payments is costly to undo and has consequences for several readers. A short record written during the decision preserves those reasons before the team forgets earlier alternatives. Keep the evidence close enough that another engineer can confirm which reading was available on that day.

What it looks like in practice

The portfolio's account of a platform's founding technical decisions describes choosing core technologies while shaping the product and its technical direction. That case shows the kind of decision worth preserving. The following fictional example illustrates how an ADR could record a similar tradeoff about ownership. It is written for this article.

Suppose a shipping service has a live order database and a read-only status page. Teams have been debating whether the status page should read the order database directly. The context is that dispatch receives late status events and

support needs a public view that cannot disclose another customer's order. The decision is to serve the view from a separate read model, updated from approved fields only. A direct read would start with fewer components and usually show fresher data, but would put public traffic and a new permission boundary next to the system of record. Reusing a nightly export would be simpler still, but would leave same-day delivery changes invisible.

The consequences should drive work. A lag indicator tells the support team when to fall back to its internal procedure. A contract check catches an accidental new field. Someone owns the update job and its alert. The team agrees to revisit the choice if lag exceeds the agreed tolerance in two busy weeks, or if the separation requirement changes. The record links to the design review and the test. A maintainer can see the cause of the extra component and the evidence needed before replacing it.

When replacing the choice, create another record, link both directions and make the old entry's status clear. Preserve the dated reason that was valid for the previous design. People investigating an older system version still need that history. An index with short titles and statuses lets readers find the live decision without reading a chain of reviews.

Before returning a status, the public endpoint verifies that the viewer has access to that specific order. Test rejection of another customer's order as well as the exact allowed response fields.

Where it goes wrong

A record is weak when it lists one preferred option and a caricature of everything else. Give alternatives the fairest version the team considered. Readers should understand why an option would have been attractive in a different situation. Ground important claims in the benchmark, standard or constraints that actually informed the decision. A copied template should leave room to describe a decision that depends on legislation, capacity or previous commitments.

A second mistake is to list advantages and leave the costs unowned. A new database, a second application or a shared queue changes someone else's working day. Include the operational handover, security boundary or maintenance cost that follows. Assign important obligations to people and checks where appropriate. If the consequences cannot be tested immediately, name what would be observed and when the team expects to learn.

Finally, keep the record small and timely. A folder of uniform documents whose milestones no longer match the work is harder to trust than a handful of precise decisions. Separate the decision's reasoning from a general tutorial and link to changing operational procedures. When a continuation becomes a new choice, preserve both decisions and give readers a reliable route to the current one.

Further reading

- AWS guidance on the architecture decision record process
- Microsoft Azure Well-Architected guidance on decisions
- UK government architectural decision record framework

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is an architecture decision record?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/architecture-decision-record/