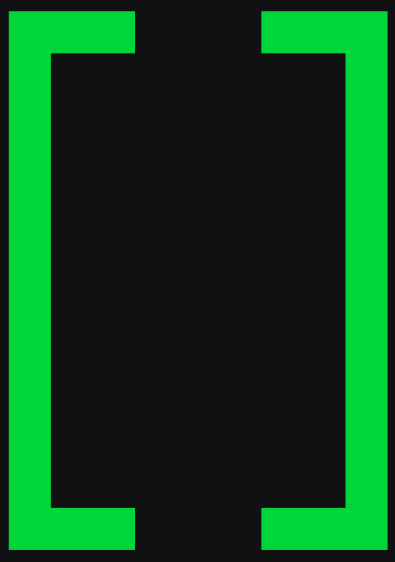
proto[docs]

FIELD GUIDE

The Architecture Documentation Map

Which architecture documents to write, who reads each one & which ones you can skip.



Matt Hensley · November 2026

proto[docs]

CONTENTS

What's in Here

01 **One Document, Too Many Readers**
Why architecture documentation goes unread · The four modes

02 **Who Reads Architecture Documentation**
Five readers & the question each one brings

03 **The Map**
Readers × modes · What goes in each cell

04 **Using the Map**
A one-afternoon audit · What makes each document go stale

05 **Resources**
Frameworks & further reading

CHAPTER 01

One Document, Too Many Readers

Most architecture documentation goes unread because it tries to serve everyone at once. Each reader shows up with one question & has to dig for the answer.

You have probably seen this page. It's called "Architecture Overview." It opens with the history of the move off the monolith, turns into setup steps halfway down, lists every environment variable in a table & ends with a diagram from two reorgs ago.

I've worked at far too many places that have "An Architecture Diagram" that outlines how half of their systems looked 2 years ago. Documents like that are worse than useless; they actively misinform. Nobody trusts them, so the real answers end up living in someone's head.

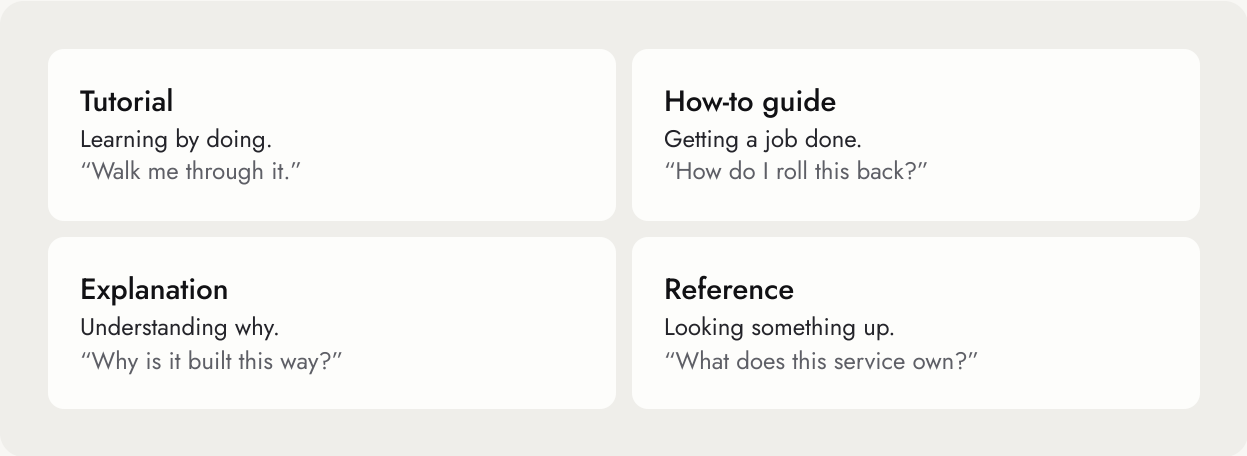
I get it, writing documentation is tedious. It's a thankless job & seems to be out of date the moment you click save. But we have all been on the receiving end of bad documentation. Documents that try to say everything, but end up saying nothing.

In this guide we are going to dive into what makes good documentation by answering two simple questions: who is reading & what they are trying to do right now.

The Four Modes

We are actually going to start by answering the second question. And to do that, we are going to look at the **Diátaxis** framework by Daniele Procida. It sorts documentation into four modes based on what the reader needs at that moment.

FIGURE 1 **The Diátaxis modes**



The top row is for doing & the bottom row is for knowing. The left column serves someone learning; the right serves someone at work.

Architecture documentation mostly lives in the bottom row. Diagrams are reference & decision records are explanation. I think that's the right home for most of it, but it leaves out the tutorials & how-tos that new & on-call engineers reach for first.

[ ]

A document should answer one kind of question for one kind of reader. If it answers two, it should be two documents.

CHAPTER 02

Who Reads Architecture Documentation

Five kinds of reader show up. Each brings a different question & a different amount of time.

One person can wear several of these hats. Go by the question they bring rather than their job title.

READER	THE QUESTION THEY BRING	WHEN THEY SHOW UP
New engineer	“How does this fit together, & how do I make my first change without breaking it?”	First two to six weeks. Reads a lot, in order.
Maintaining engineer	“What does this service depend on, & why did we build it this way?”	Before a change or a review. Reads one section & leaves.
On-call / operations	“What is broken, who owns it & what do I do right now?”	During an incident. Has minutes.
Product & leadership	“What do we have, who owns it & what does it stop us from doing?”	Planning, budgeting, due diligence. Wants one page.
Security & compliance	“Where does sensitive data go, & what protects it?”	Audits, reviews, vendor questionnaires. Reads everything & checks it.

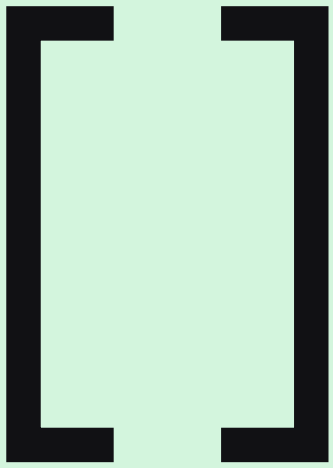
Different Slices

It’s tempting to write one long document with the simple part on top & the detail at the bottom. That breaks down fast. A security reviewer needs more detail on data flows than a new engineer does, & none of the setup steps. Each reader needs their own slice of the system.

03

The Map

Five readers across four modes gives you twenty cells. Fifteen of them are worth writing. Here is what goes in each one.



CHAPTER 03

The Map

Rows are readers. Columns are what they are trying to do.

READER	TUTORIAL	HOW-TO	REFERENCE	EXPLANATION
New engineer	Laptop to first merged change, end to end	Add an endpoint, add a service, run one test suite	Glossary, container diagram, repo map	System context & the handful of decisions that shaped it
Maintaining engineer	Skip	Release, roll back, rotate a secret, migrate data	API contracts, data model, config keys, dependencies	Decision log & known trade-offs
On-call / operations	One guided incident drill, run once	A runbook per alert	Dependency map, owners, dashboards, SLOs	Skip
Product & leadership	Skip	Skip	System landscape: what exists & who owns it	One-page context diagram with its risks
Security & compliance	Skip	How to pull audit evidence	Data flows, trust boundaries, data classes	Threat model & the controls chosen against it

Write It Once & Leave the Gaps

The context diagram serves new engineers & leadership. Draw it once & link to it from both rows; if you keep two copies, one is already wrong. The skipped cells are just as deliberate. Every document costs maintenance for as long as it exists, so spend that on readers you actually have.

CHAPTER 04

Using the Map

You can audit your current documentation against the map in an afternoon. Keeping it accurate takes a habit.

The Audit

1. Cross out the rows you don't have. A ten-person startup with no compliance requirement can drop the last row for now.
2. For each remaining cell, find the document that answers it today. Write its link in the cell or leave it blank.
3. Look for links that show up in more than one column. Those documents mix modes. Split them.
4. Rank the blank cells by how often someone asks that question in chat or in meetings. Write the top three first.
5. Give every document an owner. The ones without an owner go stale first.

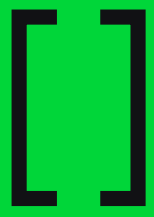


Run the audit with the people who field the questions in chat. They already know which cells are empty.

System Changes Trigger Document Changes

Documentation rarely goes stale from neglect alone. Something specific changes & nobody connects that change to the page. For each document on the map you can name the event that makes it wrong.

DOCUMENT	GOES STALE WHEN	WHO NOTICES FIRST
Context diagram	A system is added, retired or starts talking to a new third party	A new hire, or an auditor
Container diagram	A service is split, merged or renamed	The next engineer to onboard
Runbooks	An alert, threshold or piece of infrastructure changes	On-call, mid-incident
API reference	A contract changes	The consuming team, in production
Data flows	A new data store or vendor receives sensitive data	The security review, late
Setup tutorial	The toolchain or a dependency version changes	The next new hire, on day one



If you can name the event that makes a document wrong, you can catch it when it happens. Use this list as a springboard to create your own.

REFERENCE

Resources

The frameworks this guide leans on, if you want to go deeper.

Diátaxis: the four documentation modes, by Daniele Procida

diataxis.fr

The C4 model: context, container & component diagrams, by Simon Brown

c4model.com

Documenting Architecture Decisions: the original ADR post, by Michael Nygard

cognitect.com/blog/2011/11/15/documenting-architecture-decisions

arc42: a template for architecture documentation

arc42.org

Write the Docs: a community & guides on documentation practice

writethedocs.org

Plain language guidelines: writing for readers outside engineering

plainlanguage.gov

proto[docs]

prototypd.dev