Evaluate your integration maintenance posture

A guide to performing best-in-class integration maintenance

MERGE

Contents

Introduction

This guide is written for engineering leaders who have built integrations in-house and need to get up to speed on maintenance best practices. It can also be a reference for those who are beginning to build integrations.

How to use this guide

- Reference our **integration maintenance must-haves** to understand what best-in-class integration maintenance should look like.
- Use the checklist in this guide to grade your approach (or planned approach) to performing integration maintenance.
- Learn how Merge can help you maintain integrations at scale.

SECTION 1.1

Background on integration maintenance

There’s a common misconception in the world of product integration: the initial build is harder than long-term maintenance. **In reality, this couldn’t be further from the truth**.

Integration maintenance is the work your engineers need to do to keep integrations operational after you’ve onboarded customers. Maintenance is critical, expensive, and persistent for the entire lifetime of an integration.

Integrations management, separately, is all of the business operations that focus on how your customer interacts with their integration. This includes customer success responses to customer-facing issues.

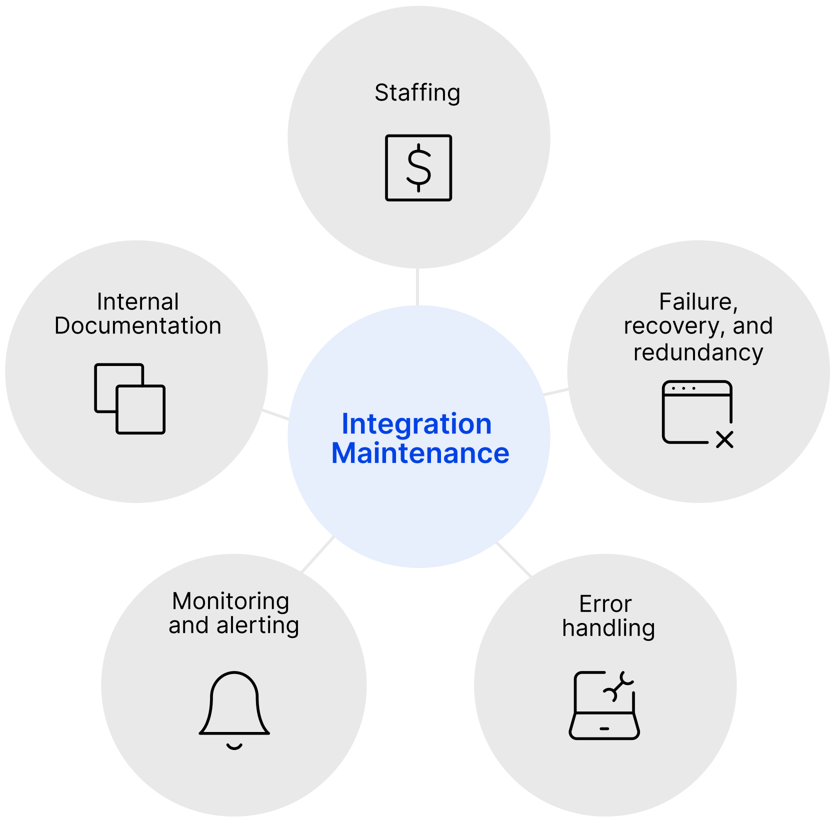
At Merge, our job is to make integrations painless for B2B companies. We’ve distilled our experience in helping Ramp, Drata, Guru, and thousands of other customers maintain integrations into this guide.

SECTION 2.0

Integration maintenance must-haves

Integration maintenance must-haves

The integration maintenance needs of modern SaaS companies fall into several buckets. Development teams with the strongest maintenance postures have processes in place for all of the items below.



SECTION 2.1

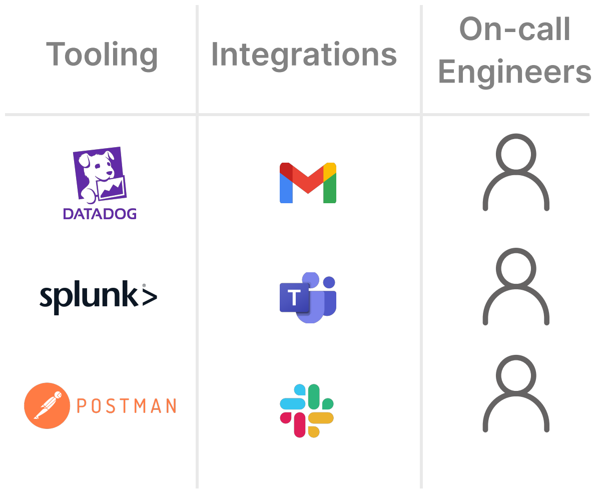
Identify integration issues quickly and easily with monitoring and alerting

Overview

Proper monitoring and alerting provides visibility on how your integrations are performing and gives your team time to respond to issues before they impact users.

How to approach monitoring and alerting

- **Invest in the right tooling.** Your engineers shouldn't invest their limited time and resources on building and maintaining monitoring and alerting infrastructure; tools like Datadog or Splunk, while expensive, can meet your needs.
- **Integrate your monitoring tool(s) with the applications your engineers already work in.** To help engineers find issues quickly, you can connect your monitoring tool(s) with a platform like Slack, and build a workflow that alerts a designated Slack channel of any issues.
- **Establish an on-call roster for integration-specific issues.** Since integration issues can occur any time, an engineer will need to be on-hand 24/7.



Identify integration issues quickly and easily with monitoring and alerting, cont.

Examples of monitoring and alerting postures

🔴 Poor	Limited monitoring and real-time alerting in place. Ad hoc or inconsistent auditing and testing.
🟡 Functional	Real-time monitoring and proactive alerting. Clear response training across teams.
🟢 Excellent	Comprehensive monitoring across performance, availability, and error rates. Real-time alerts based on established thresholds including severity-based alerts. Automated self-healing operations if possible.

SECTION 2.2

Minimize the impact of any integration issue through error handling

Overview

Error handling consists of the tools and processes in place for engineers to detect and handle integration anomalies. Done well, error handling keeps your application functioning in as many potential error states as possible, even if an API is temporarily unavailable.

How to approach error handling

- **Identify potential points of failure and develop solutions for each scenario in advance.** Your team should dedicate time to listing out all possible errors, and, for each, decide how to troubleshoot and resolve it. If possible, try to build automated solutions to save your engineers time.
- **"Normalize" response errors so that your teams can diagnose and resolve them faster.** An application may provide a response error that's uniquely worded but effectively means the same thing as certain response errors from other applications. If you and your team come across this, you should categorize or unify errors that are frequently returned.
- **Establish processes for detecting and resolving unexpected errors.** While you can try to account for integration issues proactively, you don't know what'll break an integration until it's live. Therefore, it's worth developing a standardized issue resolution protocol for addressing novel issues.

Minimize the impact of any integration issue through error handling, cont.

Examples of error handling postures

🟥 Poor	Limited error detection, logging, and fault tolerance.
🟨 Functional	Catch critical errors, detailed and secure logging, error tracking, reporting, comprehensive testing.
🟩 Excellent	Comprehensive logging, customer-facing error messaging (if needed), fault tolerance, error reporting, and documentation.

Prevent knowledge gaps by building and maintaining comprehensive API documentation

Overview

Your internal resources can cover high-level overviews on integrations, API specs, details on how they impact customers, and testing practices. In many cases, they can and should also include code samples and tutorials.

How to approach internal documentation

- **Designate an engineer to project lead documentation development.** While the engineers who are most familiar with an integration should document it, a designated “project leader” can confirm whether they’re adding enough detail and aren’t including superfluous information.

- **Consistently document key information for each integration.** You should include details like who built it, how it was built, and why it was built the way it was. You can also include details like the edge cases that have cropped up previously and broken the integration and quirks that are specific to that integration.

- **Require your engineers to review and provide updates to your internal documentation.** You can share the documentation at a certain point in an engineer’s onboarding process; and you can task all of your engineers with reviewing the documentation and adding their feedback to it on a regular basis (e.g. once per quarter).

Prevent knowledge gaps by building and maintaining comprehensive API documentation, cont.

Examples of approaches to internal documentation

🔴 Poor	Limited to no written documentation. Knowledge is distributed between team members and decentralized across multiple internal systems.
🟡 Functional	Core knowledge documented, centralized, and included in onboarding.
🟢 Excellent	Comprehensive, up-to-date documentation shared during onboarding and maintained by members of the team as API reference changes. Includes procedures, tools, and tutorials.

SECTION 2.4

Allocate the optimal number of staff to integration projects

Overview

Dedicating the right amount of engineers toward maintaining integrations can be difficult. On the one hand, you need enough personnel available to address issues over time; and on the other hand, these engineers are expensive, and if you hire too many of them, you risk limiting their productivity.

How to approach staffing

- **Hire engineers who have experience with relevant 3rd-party APIs and your existing tooling.** Recruiting engineers who have limited experience with the relevant 3rd-party APIs and/or your monitoring and alerting tools can translate to a long, resource-intensive onboarding process.
- **Analyze integration performance in conjunction with momentum on core product initiatives frequently.** If integrations break more often and/or remain broken for longer stretches, it can signal that your engineers are under water. Similarly, if your engineers fail to meet key milestones on your product roadmap, it may be worth assessing if integration maintenance is the culprit.
- **Survey your engineers frequently to gauge their morale and their existing workload.** To prevent engineers from leaving and to keep them engaged, you can send them an anonymous survey every so often (e.g. once a month) to determine how they’re feeling and whether they need more support.

Allocate the optimal number of staff to integration projects, cont.

Examples of managing staffing

🔴 Poor	Limited staffing, coverage gaps, and regular customer impact due to a lack of resourcing.
🟡 Functional	Core knowledge documented, centralized, and included in onboarding.
🟢 Excellent	Sufficient staffing and systems in place to immediately respond to unforeseen errors, bugs, and edge cases – limiting impact to customers.

SECTION 2.5

Plan for issues that go beyond APIs to maintain high performance

Overview

Issues that extend beyond the API integrations can also lead them to break, whether it's your server going down, a network outage, or your monitoring infrastructure crashing. To accommodate these critical errors, you'll need to adopt the right set of service failover, disaster recovery, and redundancy processes.

How to approach service failover, disaster recovery, and redundancy

- **Identity the full gamut of potential issues and develop contingency plans for each.** Similar to error handling, you simply don't know which issues will occur. To ensure your team is equipped for any situation, you should develop and document incident response plans for each scenario.

- **Store key items related to your API integration in a secure location.** This includes credentials, scripts, version libraries, and anything else that you'd consider core to your API integrations.

- **Backup data in a separate location.** This ensures that the local disruption to the main data source isn't impacting the backup. Separately, you'll also want to test the backup on a frequent basis to ensure it can be restored successfully.

Plan for issues that go beyond APIs to maintain high performance, cont.

Examples of failover, disaster recovery, and redundancy postures

🔴 Poor	Infrequent backups, irregular testing, and undocumented restoration procedures.
🟡 Functional	Regular data backups, testing, and quick recovery procedures in event of API failure. System redundancies, such as backup services and cached data, are also in place.
🟢 Excellent	Regular full data backups, including metadata. Audits, RCAs and improvements for any integration failures. Fully tested system redundancies, with practiced recovery procedures.

“Integration maintenance is a **painful, unavoidable, and ongoing tax** you incur from the day you decide to begin building. However, the right tools and processes can empower your team to act fast and deliver the best experience for your customers.”

- Gil Feig, Co-Founder and CTO of Merge

SECTION 3.0

Diagnosing your approach to maintenance

Integration maintenance self-assessment

So, how are you performing? Rate yourself across each of these categories to better understand the overall strength of your maintenance approach.

Maintenance category	**Your rating** *(1 = poor, 2 = functional, 3 = excellent)*
Monitoring and alerting	
Error handling	
Internal documentation	
Cost management and staffing	
Failover, recovery, and redundancy	

Total: ____________

🟥 **1-5:** critical to poor	🟨 **6-10:** fair to good	🟩 **11-15:** good to excellent

Note: if you're already maintaining your integrations in-house and have a "1" or "2" in any of the above categories, we strongly recommend that you prioritize improving this on behalf of your users. You can also connect with us to learn how thousands of other B2B companies have leveraged Merge to maintain their integrations.

SECTION 3.2

Maintain your integrations with Merge

If you handle integrations in-house, or if you're planning to, you should be prepared to invest heavily on the right resources and commit to maintaining them long-term in the interest of your business and users. You're fully responsible for the entire integration lifecycle which carries with it a growing set of complexity to manage over time.

The costly financial and time investments required to maintain integrations is a core reason why we founded Merge.

We're the leading unified API solution that allows you to add an entire category of integrations — such as HRIS, CRM, ATS, or file storage — by building to single unified API.

Also, once you connect to one of our unified APIs, we'll handle all maintenance activities on your behalf. Our support and partner engineering teams have expertise in building and maintaining third-party APIs and work closely with our customers to ensure integrations are performing correctly — so your teams don't have to.

You can learn more about Merge's approach to maintenance by scheduling a demo with one of our integration experts.