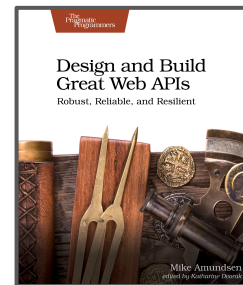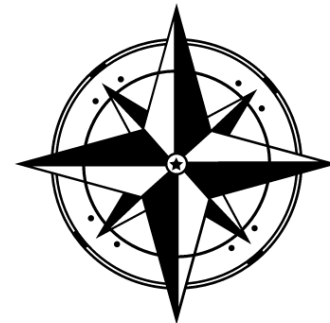# API Consumption Compass

Mike Amundsen
@mamund

# The API Consumption Compass
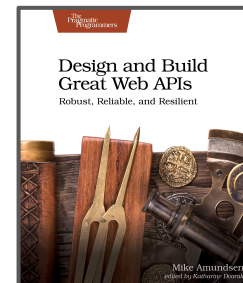
- What is it?
- The Seven Rs of API Consumption
- Summary

# The API Consumption Compass
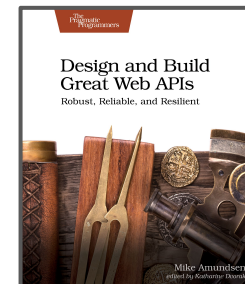
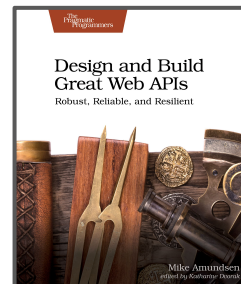# What is the API Consumption Compass?

- Addresses concerns that are important for maximizing loose coupling between components.
- Helps to treat the dependencies created by API consumption more responsibly
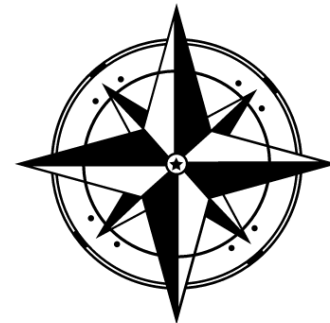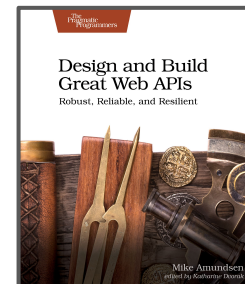- Provides a checklist for reviewing API consumers

# The Seven Rs

# The Seven Rs of API Consumption
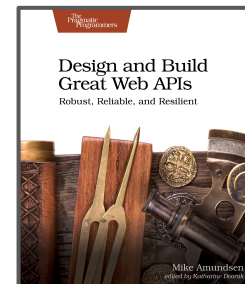
- Risk
- Replacement
- Redundancy
- Resilience

- Rightsizing
- Representation
- Reporting

# Risk
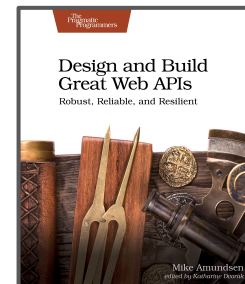
- Each API is a possible dependency risk
- Dependency failures can cascade
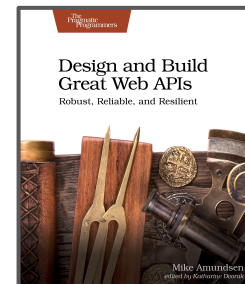- Runtime dependency is especially risky

# Risk Checklist

- Do you have a list of all APIs consumed by a single service?
- Do you have a plan for replacing each dependency (API) w/ another solution?
- What runtime protection do you have in place in case the API becomes too slow, or unreachable?
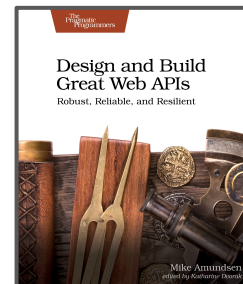
# Risk Checklist

- If you consume data from an API, do you have a plan to manage a cache or duplicate set of that data?
- If you write data to an API, do you have runtime protection in place if the service fails to confirm writes/updates/deletes?
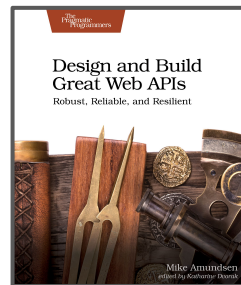- Do you monitor the "health status" of the APIs you are consuming?

# Replacement

- What if your API dependency "goes away"?
- Temporarily unavailable (network, service)
- Long-term unavailability (deprecation, cancellation)
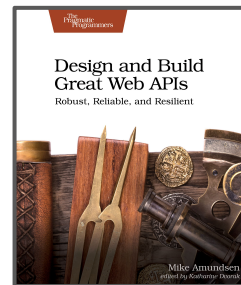
# Replacement Checklist

- Do you have a replacement plan in place for each consumed API?
- Have you identified at least one replacement for each API you consume?
- Do you have protection in place when an API becomes unavailable at runtime (circuitbreaker, etc.)?
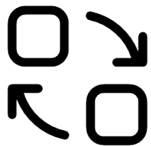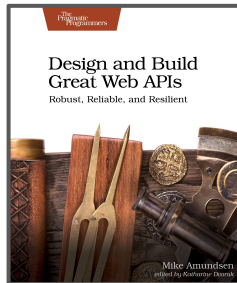
# Replacement Checklist

- Do you have protection in place when you can no longer write to an existing API (queues, etc.)?
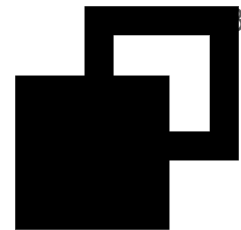- Do you have tests defined that can validate your replacement implementation?

**@mamund**

# Redundancy

- Functionality "back up"
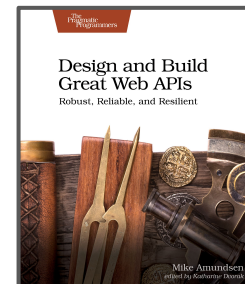- Complete copy/mirror
- Partial/Degraded functionality
- Can be costly to set-up/maintain

Design and Build
Great Web APIs
Robust, Reliable, and Resilient

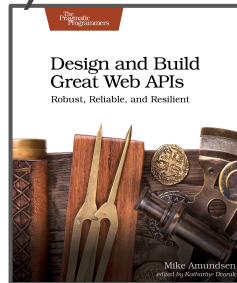Mike Amundsen

# Redundancy Checklist

- Do have at least one alternative provider for each consumed API (might include changing API providers in the future)?
- Have you considered identifying a runtime "failover" plan with a different provider for each consumed API?
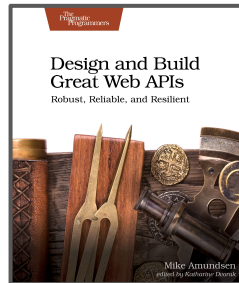- Do you have tests in place (to run before deployment) to validate your redundancy implementation?

# Resilience

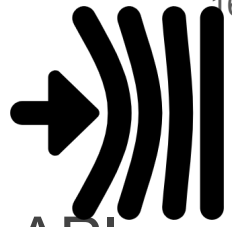- Networks can affect availability
- Too slow
- Too much traffic
- Faulty connections
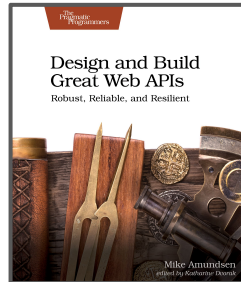
# Resilience Checklist

- Do you have runtime protection in place in case the API becomes too slow, or unreachable?
- Do you have runtime support in place for failed API state changes that write data?
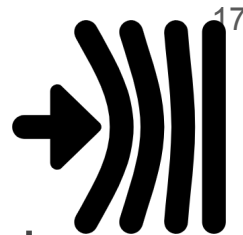- Do you have timeouts in place to prevent waiting too long for an API response?

# Resilience Checklist
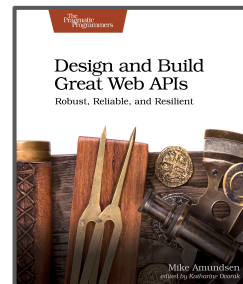
- Do you send Failfast timing budget values to APIs when you send a request to them?
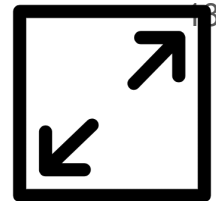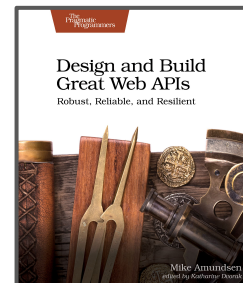- Do you use parallel requests where possible?

# Rightsizing

- Capacity needs change over time
- Seasonal traffic changes
- Planned increased consumption (new clients, etc.)

Design and Build
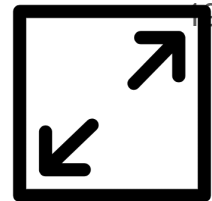Great Web APIs
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak
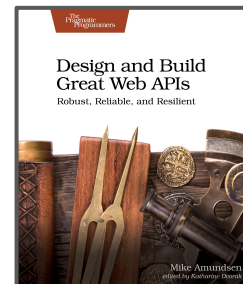
# Rightsizing Checklist

- Do you have a process for evaluating the performance of your API consumers?
- Do you use correlation IDs to track transactions throughout your ecosystem?
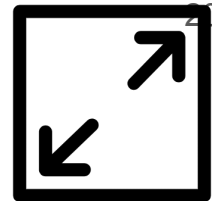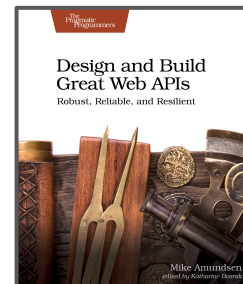
# Rightsizing Checklist

- Do you have a dashboard that records the typical transaction time (in length) as well as reach (# of components touching the transaction)?
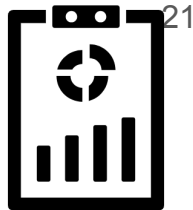- Do you have a process for evaluating the effectiveness of interactions with consumed APIs?

**@mamund**

Design and Build
Great Web APIs
Robust, Reliable, and Resilient

Mike Amundsen

# Representation
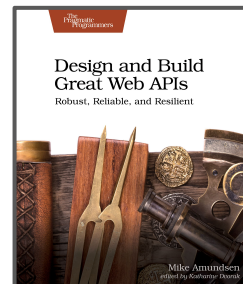
- Services send representations
- Negotiating representations is built into HTTP
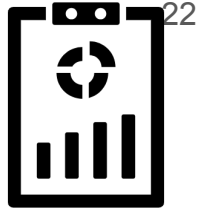- Coupling at representation level is more reliable

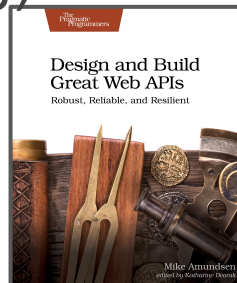**@mamund**

# Representation Checklist

- Do you convert the representations you are consuming into internal models before operating on that data?
- Do you have code that resspsonsibly handles invalid representations?
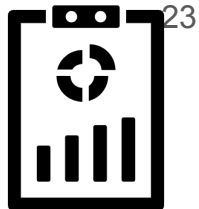- Do you robustly handle changes in representations over time (e.g. via evolution of the API you are consuming)?
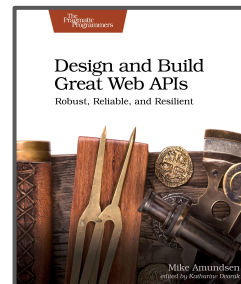
# Representation Checklist

- Do you have runtime protection in place to inspect representations for malicious code injection?
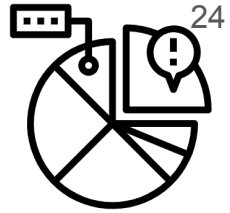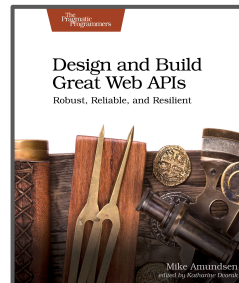- Do you have tests that validate these runtime protections?

# Reporting

- Runtime review of ecosystem performance
- Setting baseline, alerting on trends
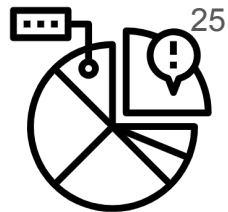- Real-time reporting vs historical analysis
- "Self-healing" recovery

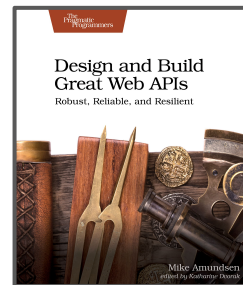**@mamund**

# Reporting Checklist

- Do you have an established set of metrics all API consumers must record?
  - Utilization, Saturation, Errors (USE)
  - Rate, Errors, Duration (RED)
  - Latency, Errors, Traffic, Saturation (LETS)
- Do you have the ability to correlate API consumption with application logic?
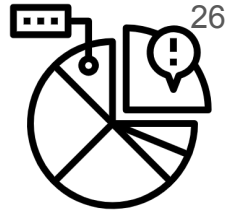
Design and Build
Great Web APIs
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak
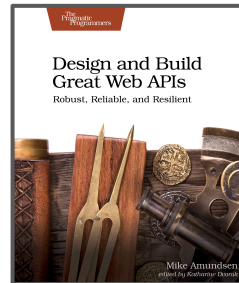
# Reporting Checklist

- Do you have a dashboard service in place to display API consumer metrics?
- Do you have an automated process that notifies you when metrics are unusual?
- Do you have an automated process that takes corrective actions when metrics are unusual?
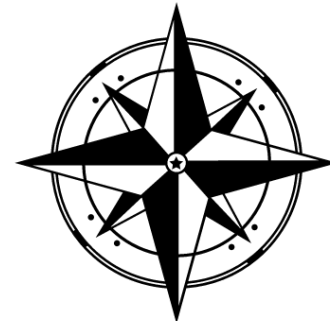
Design and Build
Great Web APIs
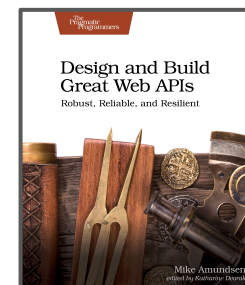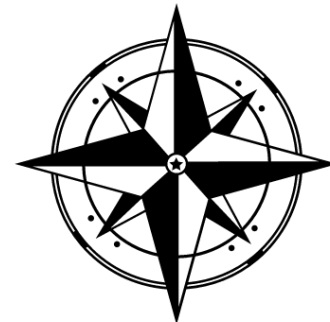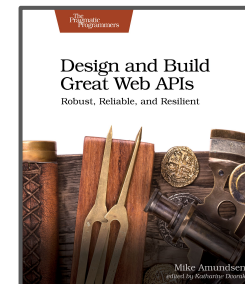Robust, Reliable, and Resilient

# Summary

# Seven Rs

- Reduce Dependency Risk
- Plan for Replacement
- Support Runtime Redundancy
- Survive Network w/ Resilience
- Rightsizing for Capacity Changes
- Coupling on Representations
- Use Performance Reporting

# API Consumption Compass

Mike Amundsen
@mamund