Failure Modes

A resilient system continues to operate successfully in the presence of failures. There are many possible failure modes, and each exercises a different aspect of resilience. The system needs to maintain a safety margin that is capable of absorbing failure, and failure modes need to be prioritized to take care of the most likely and highest impact risks. In addition to the common financial calculation of risk as the product of probability and severity, engineering risk includes detectability. Failing silently represents a much bigger risk than the same failure that is clearly and promptly reported as an incident. Hence, one way to reduce risk is to make systems more observable. Another problem is that a design control, intended to mitigate a failure mode, may not work as intended. Infrequent failures tend to cause bigger problems in practice than expected, so it's important to carefully exercise the system to ensure that design controls are well tested and operating correctly. Staff should be familiar with recovery processes and the behavior of the system when it's working hard to mitigate failures. Disaster recovery testing, game days, and chaos engineering tools are all important components of a resilient system.

However, there are many possible failure modes, and since they aren't all independent, there can be a combinatorial explosion of permutations to consider. While it's not possible to build a perfect system, here are five good tools and techniques that can focus attention on the biggest risks and minimize impact on successful operations.

The first technique is the most generally useful. Concentrate on rapid detection and response. In the end, when you've done everything you can do to manage failures you can think of, this is all you have left when that weird complex problem that no-one has ever seen before shows up! Figure out how much delay is built into your observability system, it may be taking samples once a minute, processing them for a minute or two, then watching for several bad samples in a row before it triggers an alert. It commonly takes 5-10 minutes after the problem occurred, then people have to notice and respond to emails or pager text messages, dial into a conference call, and log in to monitoring dashboards before any human response can start. Try to measure your mean time to respond (MTTR) for incidents. If your system is mitigating the initial failure, but it's getting worse, and your team responds and prevents a customer visible incident from happening, then you can record a negative MTTR, based on your estimate of how much longer it would have taken for the problem to consume all the mitigation margin. It's important to find a way to record "meltdown prevented" incidents, and learn from them, otherwise you will eventually drift into failure [Decker].

The second technique starts with the system constraints that need to be satisfied to maintain safe and successful operation and works in a top down manner using System Theoretic Process Analysis (STPA), which is described in the book Engineering a Safer World by Nancy G. Leveson. STPA is based on a functional control diagram of the system, and the safety constraints and requirements for each component in the design. There are two main steps: First identify the potential for inadequate control of the system that could lead to a hazardous state, resulting from inadequate control or enforcement of the safety constraints. These could occur if a

control action required for safety is not provided or followed; an unsafe control is provided; a potentially safe control action is provided too early, too late or in the wrong sequence; or a control action required for safety is stopped too soon or applied for too long. For the second step each potentially hazardous control action is examined to see how it could occur. Evaluate controls and mitigation mechanisms, looking for conflicts and coordination problems. Consider how controls could degrade over time, including change management, performance audits and how incident reviews could surface anomalies and problems with the system design.

The third technique is lineage driven fault detection [Alvaro]. The idea is to start with the most important business driven functions of the system and follow the dependency tree or value chain that is invoked when it is working correctly. You can then ask what happens from a business perspective when component level failures occur. Most online services have a sign-up flow that acquires new customers, and a value flow that is the main purpose of the service. For Netflix, the primary value flow is “streaming starts”, where someone starts watching a show, and the global total rates for new customers and streaming starts were made into a dashboard that was the starting point for most people during an incident, to see how big the effect on the business was. At some point Netflix also figured out how to measure the number of customers dialed into their call centers globally and add it to the same dashboard, and this became a somewhat noisy but extremely valuable and sensitive metric for understanding whether customers were unhappy. Starting with the purpose for the system, we can walk through all the steps that provide value to its users, see what might go wrong with each step, come up with an observability and mitigation strategy, and find ways to run chaos experiments to validate our design controls. In a later section, we’ll dig deep into an example that provides a good generic starting point for lineage, based on a sign-up-flow example. This is effectively a top-down approach to failure mode analysis, and it avoids the trap of getting bogged down in all the possible things that could go wrong using a bottoms-up approach.

The fourth technique is to apply the no single point of failure (SPOF) principle. If something fails, there should be another way for the system to succeed. For high resiliency systems, it’s even better to use the “rule of three” and quorum based algorithms. This is why most AWS regions have three availability zones. When there’s three ways to succeed, we still have two ways to succeed when a failure is present, and if data is corrupted, we can tell which of the three is the odd one out. When safely storing data, it’s very helpful to have three locations to write to, because once a majority have succeeded, you can move on. There is no need to retry and no extra time taken when a failure is present. For systems that are latency sensitive, creating two independent ways to succeed is an important technique for keeping the 99th percentile latency under control. Chaos tests are an important technique to validate the hypothesis that two mechanisms are independent.

The fifth technique is risk prioritization, and there is an industry standard [ISO] engineering technique called Failure Mode and Effects Analysis which uses a set of conventions for getting an estimated risk priority number (RPN) of between 1 and 1000, by ranking probability, severity and observability on a 1-10 scale, where 1 is good and 10 is bad, and multiplying them. A perfectly low probability, low impact, easy to measure risk has an RPN of 1. An extremely

frequent, permanently damaging impact, impossible to detect risk has an RPN of 1000. By listing and rating failure modes, it's easy to see which one to focus on. Next you record what effect you expect your mitigation strategy to have, which should drop its RPN and then let you focus on the new highest RPN, until there aren't any high values left. In practice, the easiest way to reduce RPN is to add observability, so you aren't working blind. You can then get some empirical measurements of probability, as once it's visible, you can see how often it occurs. We'll use FMEA to work through the sign-up flow example in a later section.

Taking a top down approach we can divide the failure modes into four general categories. Each category is centered around the responsibilities of a different team of people, and should be discussed and developed in partnership with those teams.

Following the value chain from the business perspective, the first team is the developers who build the unique code that makes up the business logic of the system. The system itself could be a single microservice with a small team, which makes it easier to reason about, or a large monolithic application. The point is to focus on faults that are in the scope of control of the developers of the system, and that are tied directly to the business value of the application.

The second team is the software platform team, who provide standardized supported libraries, open source projects, packaged software, operating systems, build pipelines, language runtimes, databases, external high level services etc. that are used across multiple applications. They are indirectly supporting business value for multiple teams and use cases, and have to manage components that they aren't able to modify easily, but do have to manage versions and external supply chains.

The third team is the infrastructure platform team, who deal with datacenter and cloud based resources. They are concerned with physical locations and cloud regions, networking failures, problems with infrastructure hardware, and failures of the control planes used to provision and manage infrastructure.

The fourth team provides observability, incident management and operations for services in general, and includes the developers of specific highly resilient services in a DevOps role. Failures of observability and incident management can compound a small problem into a large one, and make the difference between a short and a long time to fix problems.

In all four cases, there is a common starting point and structure to the failure modes which should be extended to take account of a particular situation. The criticality and potential cost of each failure mode is context dependent, and drives the available time and budget for prioritized mitigation plans. The entire resiliency plan needs to be dynamic, and to incorporate learnings from each incident, whether or not the failure has noticeable customer impact.

Failure Modes and Effects Analysis

The FMEA spreadsheet is used to capture and prioritize risks based on Severity, Probability and Detectability where each is rated on a 1 to 10 scale. A standard model for each follows, the exact values chosen are somewhat arbitrary, and some forms of FMEA use a 1 to 5 scale, but all we are trying to do is come up with a rough mechanism for prioritization, and in practice this is good enough for the purpose.

Severity starts with several high levels that destroy things, in other words irreversible failures like death or incapacitation of a person, destruction of machinery, flood and fire in a datacenter. The next few levels are temporary incapacitation, recoverable with degradation of performance, and finally ratings of minor or no effect.

Effect	SEVERITY of Effect	Ranking
Hazardous without warning	Very high severity ranking when a potential failure mode affects safe system operation without warning	**10**
Hazardous with warning	Very high severity ranking when a potential failure mode affects safe system operation with warning	**9**
Very High	System inoperable with destructive failure without compromising safety	**8**
High	System inoperable with equipment damage	**7**
Moderate	System inoperable with minor damage	**6**
Low	System inoperable without damage	**5**
Very Low	System operable with significant degradation of performance	**4**
Minor	System operable with some degradation of performance	**3**
Very Minor	System operable with minimal interference	**2**
None	No effect	**1**

For probability, we use an exponential scale, from almost inevitable and repeated observed failures down through occasional failures to failures that haven't been seen in practice. The probabilities are guesses during the design phase, but should be measured in real life when a system is operating, and the risk updated based on what is seen in practice.

PROBABILITY of Failure	Failure Prob	Ranking
Very High: Failure is almost inevitable	>1 in 2	**10**
	1 in 3	**9**
High: Repeated failures	1 in 8	**8**
	1 in 20	**7**

Moderate: Occasional failures	1 in 80	**6**
	1 in 400	**5**
	1 in 2,000	**4**
Low: Relatively few failures	1 in 15,000	**3**
	1 in 150,000	**2**
Remote: Failure is unlikely	<1 in 1,500,000	**1**

For detectability we are thinking in terms of a design control, metrics that track behavior of the system, and alerting rules that can initiate an incident to investigate a fault. If there is no way to detect the failure it gets a high score, if we have a robust and well tested alert for the issue and a clear incident handling process in place, it gets the lowest score.

Detection	Likelihood of DETECTION by Design Control	Ranking
Absolute Uncertainty	Design control cannot detect potential cause/mechanism and subsequent failure mode	**10**
Very Remote	**Very remote chance the design control will detect potential cause/mechanism and subsequent failure mode**	**9**
Remote	**Remote chance the design control will detect potential cause/mechanism and subsequent failure mode**	**8**
Very Low	**Very low chance the design control will detect potential cause/mechanism and subsequent failure mode**	**7**
Low	**Low chance the design control will detect potential cause/mechanism and subsequent failure mode**	**6**
Moderate	**Moderate chance the design control will detect potential cause/mechanism and subsequent failure mode**	**5**
Moderately High	**Moderately High chance the design control will detect potential cause/mechanism and subsequent failure mode**	**4**
High	**High chance the design control will detect potential cause/mechanism and subsequent failure mode**	**3**
Very High	**Very high chance the design control will detect potential cause/mechanism and subsequent failure mode**	**2**
Almost Certain	Design control will detect potential cause/mechanism and subsequent failure mode	**1**

The spreadsheet is organized into sections, listing failure modes for each function. There is also a recommended action, listing who is responsible and when, actions taken and the updated severity, occurrence and detectability that lead to a planned reduction in the RPN. The rows are shown split below for readability. The only formula needed is RPN=Sev*Prob*Det.

Item / Function	Potential Failure Mode(s)	Potential Effect(s) of Failure	Sev	Potential Cause(s)/ Mechanism(s) of Failure	Prob	Current Design Controls	Det	RPN

Recommended Action(s)	Responsibility & Target Completion Date	Action Results				
		Actions Taken	New Sev	New Occ	New Det	New RPN

Application Layer FMEA

The first FMEA models the application layer assuming it is implementing a web page or network accessed API. Each step in the access protocol is modelled as a possible failure mode, starting with authentication, then the access itself. This is followed by some common code related failure modes. For a specific application team, these should be discussed, prioritized and have additional failure modes added.

Item / Function	Potential Failure Mode(s)	Potential Effect(s) of Failure	Sev	Potential Cause(s)/ Mechanism(s) of Failure	Prob	Current Design Controls	Det	RPN	Recommended Action(s)
Authentication	Client can't authenticate	Can't connect application	5	Certificate timeout, version mismatch, account not setup, credential changed	3	Log and alert on authentication failures	3	**45**	
	Slow or unreliable authentication	Slow start for application	4	Auth service overloaded, high error and retry rate	3	Log and alert on high authentication latency and errors	4	**48**	
								0	
Client Request to API Endpoint	Service unknown, address un-resolvable	Delay while discovery or DNS times out, slow fallback response	5	DNS configuration error, denial of service attack, or provider failure	1	Customer eventually complains via call center	10	**50**	Dual redundant DNS, fallback to local cache, hardcoded IP addresses. Endpoint monitoring and alerts
	Service unreachable, request undeliverable	Fast fail, no response	4	Network route down or no service instances running	1	Autoscaler maintains a number of healthy instances	1	**4**	Endpoint monitoring and alerts

	Service reachable, request undeliverable	Connect timeout, slow fail, no response	4	Service frozen/not accepting connection	1	Retry request on different instance. Healthcheck failure instances removed. Log and alert.	2	**8**	
	Request delivered, no response - stall	Application request timeout, slow fail, no response	4	Broken service code, overloaded CPU or slow dependencies	1	Retry request on different instance. Healthcheck failure instances removed. Log and alert.	2	**8**	
	Response undeliverable	Application request timeout, slow fail, no response	4	Network return route failure, dropped packets	1	Retry request on different instance. Healthcheck failure instances removed. Log and alert.	2	**8**	
	Response received in time but empty or unintelligible	Fast fail, no response	3	Version mismatch or exception in service code	2	Retry request on different instance. Healthcheck failure instances removed. Log and alert.	2	**12**	
	Request delivered, response delayed beyond spec	Degraded response arrives too late, slow fallback response	6	Service overloaded or GC hit, dependent services responding slowly	2	Retry request on different instance. Healthcheck failure instances removed. Log and alert.	2	**24**	
	Request delivered, degraded response delivered in time	Degraded timely response	2	Service overloaded or GC hit, dependent services responding slowly	2	Log and alert on high service latency and errors	2	**8**	
Time Bombs	Internal application counter wraparound								Test long running operations of code base
	Memory leak								Monitor process sizes and garbage collection intervals over time
Date Bombs	Leap year, leap second, epoch wrap around, "Y2K"								Test across date boundaries
Content Bombs	Incoming data that crashes the app								Fuzz the input with generated random and structured data to show it doesn't crash.
Configuration Errors	Configuration file syntax errors or incorrect values								Canary test deployments incrementally. Chaos testing.
Versioning Errors	Incompatible interface versions								Canary test deployments incrementally

Retry Storms	Too many retries, too large timeout values								Chaos testing applications under stress
Excessive Logging	Cascading overload								Chaos testing applications under stress

Software Stack FMEA

The software stack starts along the same lines, with authentication and a request response sequence, however the more specific failure modes relate to the control planes for services hosted in cloud regions. In general a good way to avoid customer visible issues caused by control plane failure modes is to pre-allocate network, compute and database structures wherever possible. The cost of failure should be weighed against the cost of mitigation.

Item / Function	Potential Failure Mode(s)	Potential Effect(s) of Failure	Sev	Potential Cause(s)/ Mechanism(s) of Failure	Prob	Current Design Controls	Det	RPN	Recommended Action(s)
Authentication to cloud services	Client can't authenticate	Can't connect application	5	Certificate timeout, version mismatch, account not setup, credential changed	3	Log and alert on authentication failures	3	**45**	
	Slow or unreliable authentication	Slow start for application	4	Auth service overloaded, high error and retry rate	3	Log and alert on high authentication latency and errors	4	**48**	
								0	
Client request to cloud service endpoint	Service unknown, address un-resolvable	Delay while discovery or DNS times out, slow fallback response	5	DNS configuration error, denial of service attack, or provider failure	1			**0**	
	Service unreachable, request undeliverable	Fast fail, no response	4	Network route down or no service instances running	1			**0**	

	Service reachable, request undeliverable	Connect timeout, slow fail, no response	4	Service frozen/not accepting connection	1			**0**	
	Request delivered, no response - stall	Application request timeout, slow fail, no response	4	Broken service code, overloaded CPU or slow dependencies	1			**0**	
	Response undeliverable	Application request timeout, slow fail, no response	4	Network return route failure, dropped packets	1			**0**	
	Response received in time but empty or unintelligible	Fast fail, no response	3	Version mismatch or exception in service code	2			**0**	
	Request delivered, response delayed beyond spec	Degraded response arrives too late, slow fallback response	6	Service overloaded or GC hit, dependent services responding slowly	2			**0**	
	Request delivered, degraded response delivered in time	Degraded timely response	2	Service overloaded or GC hit, dependent services responding slowly	2			**0**	
EC2 Control Plane	Instance request refused, direct or via autoscaler	Capacity limited or control plane failure		Limit reached, or Insufficient Capacity Exception					Service call for increased limit. Try a different instance type, different zone, or different region
	Instance created but fails to start	Bad instance hardware						0	Retry via autoscaler
	Instance slow to start								
EC2 Network Control Plane	Configuration request refused	Capacity limited or control plane failure		Limit reached, or Insufficient Capacity Exception					Service call for increased limit. Try a different zone, or different region
	Network creation started but operation fails							0	Pre-allocate all network structures in all regions

Database Control Plane (DynamoDB or Aurora)	Configuration request refused							0	Service call for increased limit. Try a different zone, or different region
	Database table creation started but operation fails							0	Pre-allocate all database tables in all regions

Infrastructure FMEA

It's not generally useful to talk about "what to do if an AWS zone or region has an outage" because it depends a lot on what kind of outage and what subset of services might be impacted. Service specific control plane outages are part of the software stack FMEA. If a datacenter building is destroyed by fire or flood, we have a very different kind of failure than a temporary power outage or cooling system failure, and that's very different to losing connectivity to a building where all the systems are still running, but isolated. In practice, we can expect individual machines to fail randomly with very low probability, groups of similar machines to fail in a correlated way due to bad batches of components and firmware bugs, and extremely rare availability zone scoped events caused by weather, earthquake, fire and flood.

Item / Function	Potential Failure Mode(s)	Potential Effect(s) of Failure	Sev	Potential Cause(s)/ Mechanism(s) of Failure	Prob	Current Design Controls	Det	RPN	Recommended Action(s)
Availability Zone Durability	Permanent destruction of zone	Total data loss in zone	8	Fire or flood inside building or destruction of datacenter building	2	Cross zone synchronous replication to over 10Km away	1	**16**	Ensure that system can run on two out of three zones
	Temporary loss of zone	Loss of compute capacity and non-durable state in zone	5	Power or cooling outage causes reboot of part or all of a datacenter building	3	Cross zone synchronous replication to over 10Km away	1	**15**	Ensure that system can run on two out of three zones
								0	
Region Connectivity	Address un-resolvable	Delay while DNS times out, slow fallback response	5	DNS configuration error, denial of service attack, or provider failure	1			**0**	Dual redundant DNS, fallback to local cache, hardcoded IP addresses. Endpoint monitoring and alerts
	Unreachable, request undeliverable	Fast fail, no response	4	Network route down	1			**0**	Failover to secondary region

	Request undeliverable	Connect timeout, slow fail, no response	4	Router frozen/not accepting connection	1			**0**	Failover to secondary region
	Request delivered, no response - stall	Application request timeout, slow fail, no response	4	Broken router, overloaded network or slow dependencies	1			**0**	Failover to secondary region
	Response undeliverable	Application request timeout, slow fail, no response	4	Network return route failure, dropped packets	1			**0**	Failover to secondary region
	Response received in time but empty or unintelligible	Fast fail, no response	3	Network response failure	2			**0**	Failover to secondary region
	Request delivered, response delayed beyond spec	Degraded response arrives too late, slow fallback response	6	Network overloaded dependent services responding slowly	2			**0**	Failover to secondary region
	Request delivered, degraded response delivered in time	Degraded timely response	2	Service overloaded, dependent services responding slowly	2			**0**	Alert operators

Operations and Observability

Misleading and confusing monitoring systems cause a lot of failures to be magnified rather than mitigated. While some of the failure modes can be prioritized with an FMEA, these higher level failures are better modelled using Systems Theoretic Process Analysis (STPA), which also captures the business level criticality of the application.

Item / Function	Potential Failure Mode(s)	Potential Effect(s) of Failure	Sev	Potential Cause(s)/ Mechanism(s) of Failure	Prob	Current Design Controls	Det	RPN
Authentication	Montoring agent can't authenticate	Can't monitor application	5	Certificate timeout, version mismatch, account not setup, credential changed	3	Log and alert on authentication failures	3	**45**

	Montoring tool end user operator can't authenticate	Can't monitor system, increased MTTR	5	Certificate timeout, version mismatch, account not setup, credential changed	3	Log and alert on authentication failures	3	**45**
	Slow or unreliable authentication	Errors and delays in observability and alerts	4	Auth service overloaded, high error and retry rate	3	Log and alert on high authentication latency and errors	4	**48**