Microsoft

Resilience by design for cloud services

A structured methodology for prioritizing engineering investments

May 2013

Contents

Overview

Microsoft Trustworthy Computing (TwC) has collaborated with a number of cloud service teams across Microsoft to develop an approach for increasing cloud service resiliency by identifying and analyzing potential failures. This paper briefly frames the motivation and benefits of incorporating robust resilience design into the development cycle. It describes Resilience Modeling and Analysis (RMA), a methodology for improving resiliency adapted from the industry-standard technique known as Failure Mode and Effects Analysis (FMEA)[1], and provides guidance for implementation.

The primary goal of this paper is to equip cloud service engineers with a detailed understanding of RMA, including the steps and templates used to complete the process, to enable easy and consistent adoption.

Background

Software development has traditionally emphasized fault prevention, and because customers operated the software, any failures were isolated to customers' on-premises deployments. Today, cloud services typically run as highly complex, distributed, "always available" systems that serve many customers. Cloud systems are globally distributed, often built using commodity hardware, and have inherent dependencies on third-party and partner services. The nature of the Internet and global networking is that transient and even prolonged failures are quite common. So engineers need to make a necessary mind shift to adopt Recovery-Oriented Computing (ROC) practices,[2] fully embrace the idea that failures <u>will</u> happen, and therefore incorporate coping strategies into their service design and software to minimize the harmful effects of such failures.

The following figure shows a spectrum of failures, ranging from infrequent (natural environmental disasters or man-made disasters) to common (software imperfections, hardware failures, or human errors). Because common failures are inevitable, their occurrence and impact need to be factored into the service during the design phase so that the software can be designed and built in a more resilient way, and impact to users can be minimized.

[1] Failure Mode and Effects Analysis is, also referred to as Failure Mode, Effects, and Criticality Analysis (FMECA).
[2] "Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies" http://roc.cs.berkeley.edu/papers/ROC_TR02-1175.pdf

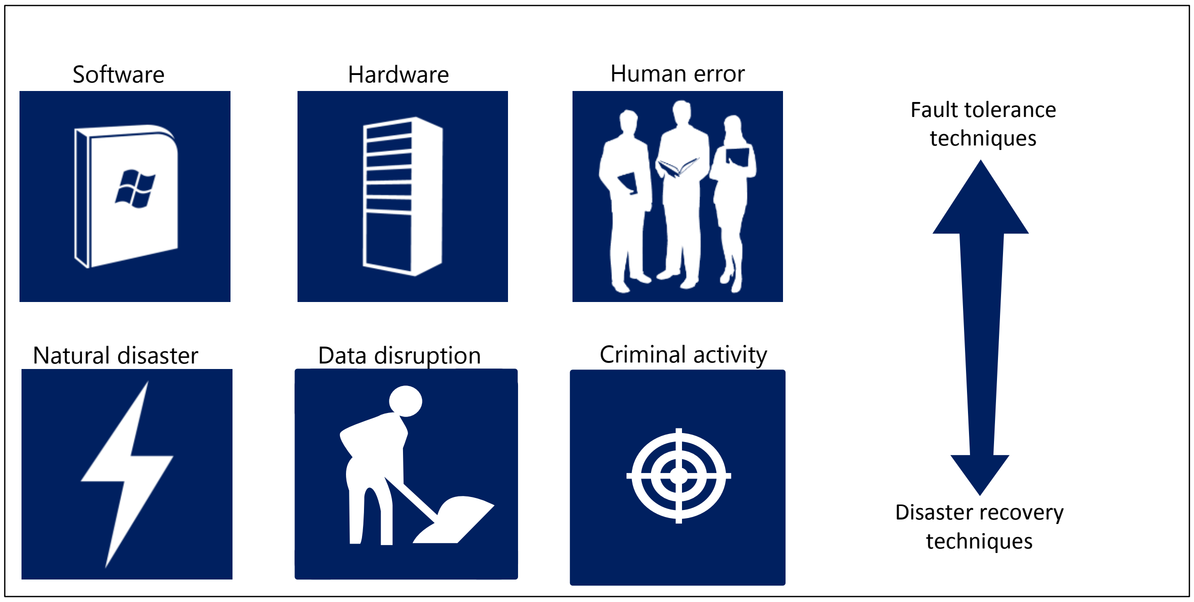


Figure 1. The spectrum of failure ranges from common failures to infrequently occurring failures. Failure is inevitable, so fault tolerance techniques should be incorporated into the service design to reduce the impact when failures occur.

If we embrace the idea that failures are expected in the world of cloud services, engineers must change their emphasis from designing to extend time between failures (TBF) to designing to reduce time to recover (TTR) from failures. If failures are commonplace, the most important objective is to detect them quickly and develop coping strategies that minimize their effects on customers.

Concepts from the industry-standard process known as FMEA have been adapted to create a Resilience Modeling and Analysis (RMA) methodology for cloud services teams at Microsoft. The purpose of this methodology is to more effectively prioritize work in the areas of detection, mitigation, and recovery from failures, all of which are significant factors in reducing TTR. By completing RMA, an engineering team will have thought through many of the reliability issues in depth and be better equipped to ensure that when failures occur, the impacts to customers are minimized.

FMEA is a flexible framework for performing qualitative failure analysis in industrial and computing systems. Potential failures are identified and the consequences of those failures are analyzed to assess risk.

Benefits

Adopting RMA benefits cloud service engineering teams by encouraging the following best practices, all of which can improve service reliability.

Address reliability issues early in design

The primary goal and benefit of RMA is to discover resilience gaps and explicitly design for their detection and mitigation before code is committed to production (where it becomes costlier to change and update). RMA can be codified in the development lifecycle and practiced by cloud service engineering organizations to become an important core competency that will instill the principles of ROC into engineering teams so they are consistently focused on reducing the time to detect, mitigate, and recover from failures.

After the service is in production, continuing to apply the RMA methodology will enable the engineering organization to apply any knowledge that is gained to the next engineering cycle.

Prioritize reliability-related work efforts

The key objective of RMA is to identify and produce a prioritized list of common reliability failures relevant to a specific service. Participants gain an understanding that recovery from failures must be emphasized over prevention of those failures. Complex cloud services are often subjected to myriad failure conditions, and it is often difficult for teams to know where they should invest their efforts to reduce the impact of those failures. This question is particularly troublesome for teams whose services consume components or services from multiple first or third-party component owners or service providers. RMA helps to prioritize and inform investment decisions in the areas of detection, mitigation, and recovery.

Provide tangible outputs for other reliability efforts

The RMA process produces tangible outputs that can be used for other reliability-focused efforts.

Partner teams, service operations, and support personnel can gain a better understanding of how the production software is instantiated by leveraging the component interaction diagram (CID), created during the initial phase of the RMA process. It provides a different pivot than the standard architectural or design diagrams that are frequently found in traditional development specifications.

The *prioritized failures list* maps concretely to work items and code bugs. It is also an excellent resource for the test organization to help develop test cases. These maps provide insight to places in the system where fault injection[3] practices could be best applied to validate the effectiveness of failure mitigations.

[3] The canonical example of fault injection testing is the Netflix resiliency tool known as Chaos Monkey.

How resilience modeling and analysis works

The resilience modeling and analysis (RMA) process is completed in four phases, which are shown and described in the following figure and bullet points:

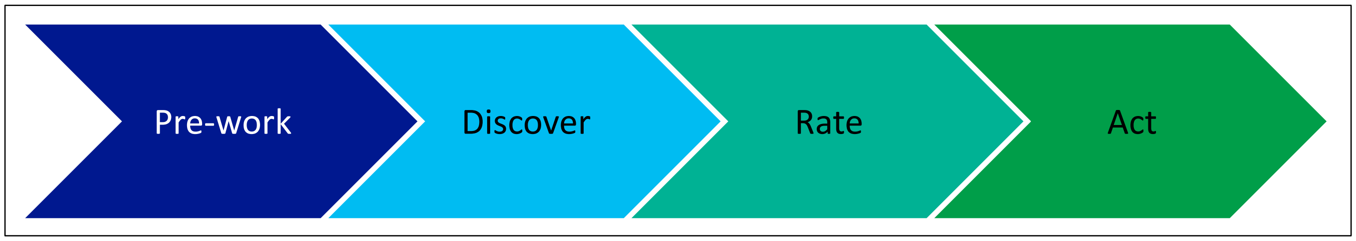


Figure 2. Phases of the RMA process

- Pre-work. Creates a diagram to capture resources, dependencies, and component interactions.
- Discover. Identifies failures and resilience gaps.
- Rate. Performs impact analysis.
- Act. Produces work items to improve resilience.

Pre-work
Two tasks are performed during this phase of the analysis. The first task is to create a component interaction diagram (CID); the second task is to transfer all of the interactions from the diagram into the RMA workbook template. The primary purpose of this phase is to capture all resources and the interactions between those resources. These interactions will be used to focus the failure enumeration in the Discover phase.

The two tasks in this phase must be completed before continuing to the Discover phase.

Task 1: Create the CID

It is critically important to the success of the RMA process to generate a high-quality and comprehensive diagram in this task. If resources or interactions are missing from the diagram, failures may be missed and the value of the exercise may be diminished. The developer(s) of the modeled component resources in the diagram are the primary personnel who are needed for this task.

Engineers may question how much detail is needed for the diagram, but there is no clear-cut rule to follow. The answer will depend on whether the team has chosen to scope the exercise to an end-to-end scenario, cloud service boundaries, or components.[4] However, some general guidance applies:

[4] See the "Approach" subsection later in this paper for more information on scoping the exercise.

- Do not include physical hardware. Cloud services are typically composed of server roles, of which there are usually multiple instances. In most cases, it is not productive to depict server components such as disks, network cards, processors, and so on. Although failures do occur with these components, both the impact and the frequency of these failures are well understood. Should a failure of this type affect a resource's ability to function, the effects will be seen in the interaction from the caller of this resource in the form of an error or simply no response. Similarly, network components such as routers, switches, and load balancers are all sources of failure, but they need not be drawn on the diagram. More detail is provided in the following text about how to capture the relevant failure information for these device/component types.
- Enumerating instances is important. The number of functional units is extremely important for reliability modeling. Redundancy is one of the primary resiliency techniques applied at all layers of a cloud service. The number of geographic regions where your service exists, the number of data centers within a region, and the number of server roles and instances are all important attributes to capture. This information is used to determine the likelihood that a given interaction failure will affect customers.
- Include all dependencies. Cloud services have many dependencies, from name resolution, to authentication, to data processing and storage. Often these services are provided by systems that are not owned by the cloud service team but are critical to the proper functioning of the service. Each of these dependency systems should be represented on the diagram, with the appropriate interactions between them and your cloud service components clearly depicted. However, the composition of dependency systems can be hidden by drawing them on the diagram as a single shape outside of the data center (if the service is available over the Internet) or as a single shape within each data center (if the service is provided locally in the data center). If service level agreement (SLA) information is known for the dependency system, this should be noted on the diagram as well.

Most teams have a service diagram that is based on design or architecture documents. In addition, teams that already practice security threat modeling as described in the Microsoft Security Development Lifecycle (SDL)[5] will have Level 0 data flow diagrams to refer to. Both types of diagrams are good starting points; however, they often lack some—or all—of the interactions required for a complete CID. Trustworthy Computing has created sample CID documents[6] to help teams build their own CID. These sample documents include many shapes and connections to provide visual cues that will help teams analyze failures later in the process. The following figure is a sample CID for a simple cloud service named *Contoso*,[7] a cloud service that collects information from Internet clients, transforms the information using a third party service, and stores the final data in the cloud.

[5] Microsoft Security Development Lifecycle (SDL) www.microsoft.com/sdl
[6] See the "Additional resources" section at the end of this paper for links to sample documents.
[7] This diagram does not use all of the shapes that are available in the CID Visio stencil; however, the full Visio stencil contains a complete legend with a description of every shape.

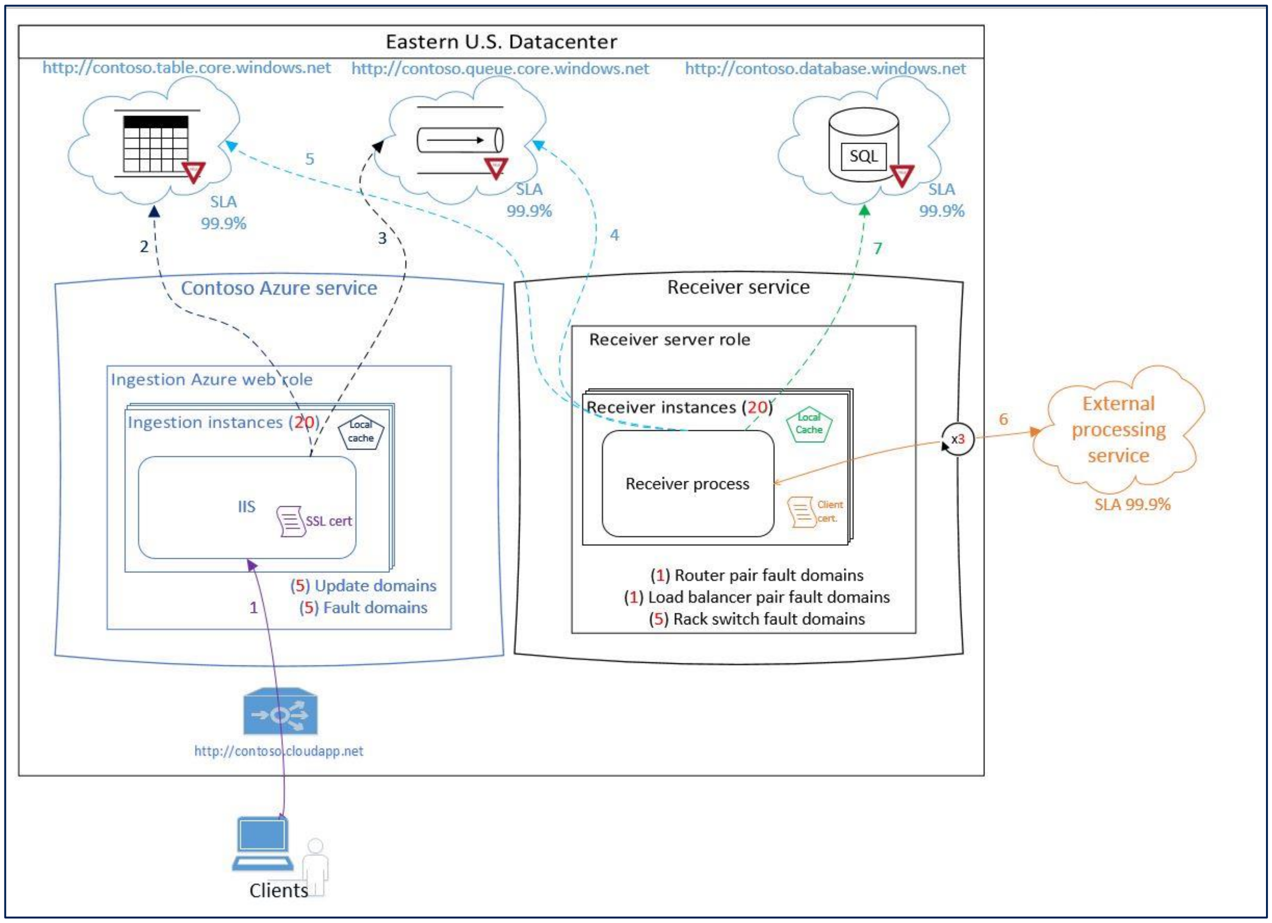
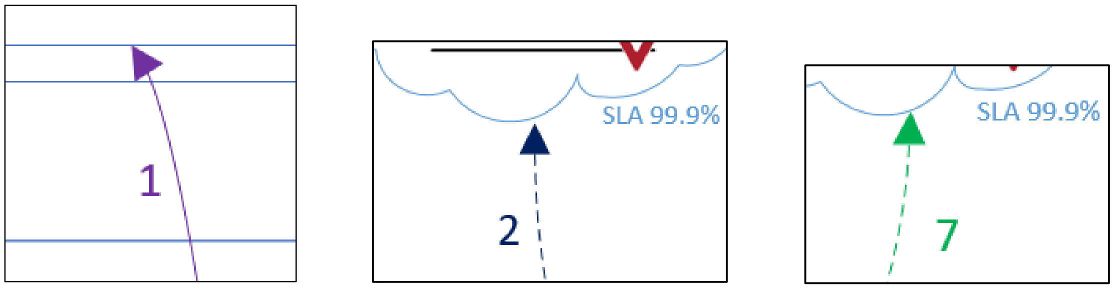
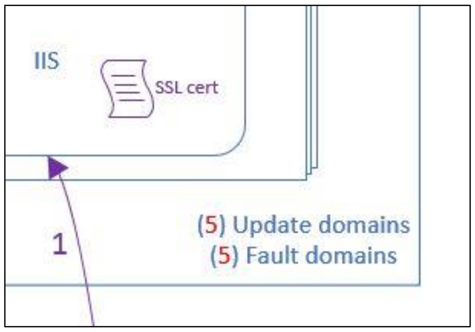


Figure 3. Sample component interaction diagram (CID)

The following figures provide a closer look at some of the shapes that provide visual cues for brainstorming about failures during the Discover phase:

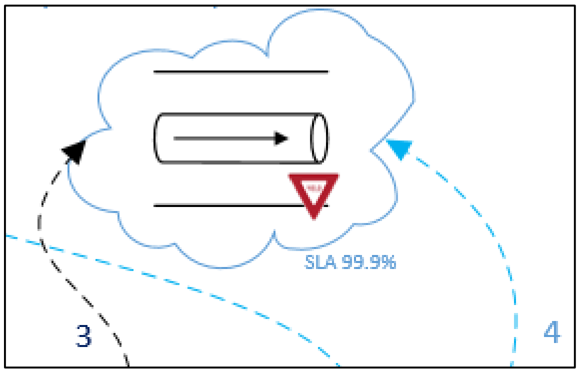
- Interaction arrows and numbers. The most important pieces of information are the component interactions, which are analyzed in the Discover phase to explore all of the different failures that might be encountered. The interactions are all labeled with a number that will be transferred into the RMA workbook.

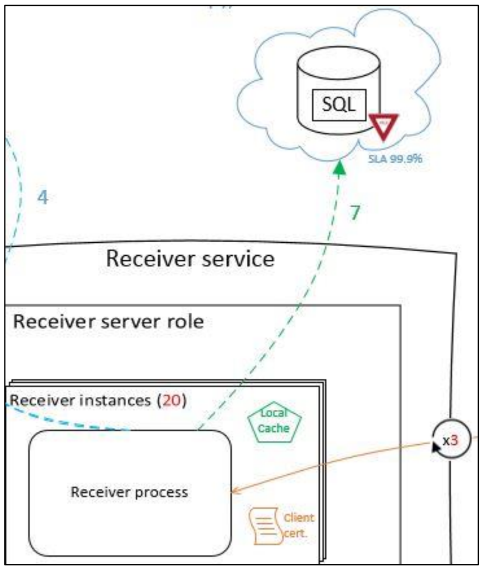




- Certificates. The certificate shape is used to highlight instances where certificates are required. Certificate related failures are frequent sources of failure. Notice how the certificates in the diagram are color-coded to match the corresponding interaction.

- Yield sign. The yield sign denotes that this resource employs throttling, which indicates a caller may encounter failures on interactions with this resource when there is an intentional slowing of the service.



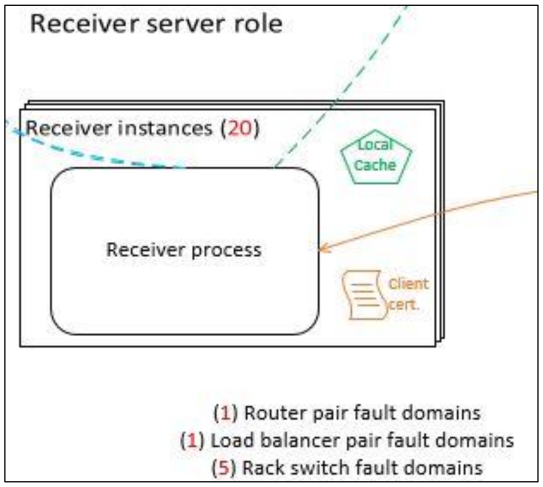


- Cache. Notice the local cache shape in green, which is included inside the receiver instances. Caching is a common mitigation against failures, and in this case, if the Receiver Service cannot succeed (via interaction #7) to store results in the database, it will cache the data locally until the connection to the database is restored.

- Fault domains. The following sample diagram depicts server roles of various types. Each role type uses special labeling to capture information about different *fault domains*. The concept of fault domains will be familiar to anyone who has developed cloud services on Windows Azure. A user chooses the number of fault domains for their role instances, and the underlying Windows Azure infrastructure ensures the role instances are striped across server racks, switches, and routers such that a failure in a lower-level infrastructure layer does not affect more than one fault domain. When discussing failures for role instances, this information is important because it directly influences the breadth of impact that a failure of this type will have on an Azure role type.

For cloud services built in a traditional data center hosting model, it is possible to apply this same concept of fault domains to the underlying infrastructure to assess impacts to the server role. In addition, it is possible to gauge the level of impact of failures in those components by simply noting the number of fault domains for each type.

In this diagram, notice all receiver server role instances are connected to one router pair, one load balancer pair, and are striped across five rack switches. This information helps convey the impact to the receiver role should a failure occur in any of these infrastructure layers.


Receiver server role
Receiver instances (20)
Local Cache
Receiver process
Client cert.
(1) Router pair fault domains
(1) Load balancer pair fault domains
(5) Rack switch fault domains

After the component interaction diagram is completed, the engineering team can move on to the second task in the Pre-work phase.

Task 2: Transfer the interactions from the diagram to the RMA workbook

The second task in the Pre-work phase transfers the interaction numbers from the CID into the RMA workbook to create the master list of interactions, which is used during the Discover phase to enumerate various failure types that might be encountered during each interaction.

The numbered interactions are entered into the RMA workbook. The required information includes the ID, a short name (usually specifying the caller and the responder), and a description of the interaction itself. The following workbook example includes information from the diagram for the Contoso service, which was discussed earlier in Task 1.

ID	Component/ Dependency Interaction	Interaction description
1	Internet Client -> Ingestion Service	End-user clients connect via the internet-facing endpoint of the Contoso Azure Service Ingestion web roles and upload data.
2	Ingestion Roles -> Azure Storage Table	Ingestion Instances store the client's uploaded data in Azure Storage Table.
3	Ingestion Roles -> Azure Storage Queue	Ingestion Instances send a message to the Azure Storage Queue with a pointer to the client's data stored in Azure Storage Table.
4	Receiver Roles -> Azure Storage Queue	Receiver Instances pull the message off the Azure Storage Queue which contains a pointer to the client's data stored in Azure Storage Table.
5	Receiver Roles -> Azure Storage Table	Receiver Instances pull the client's uploaded data from the Azure Storage Table.
6	Receiver Roles -> External Processing	Receiver Instances push the client's uploaded data to the External Processing Service for data transformation.
7	Receiver Roles -> Azure SQL Database	Receiver Instances push the client's transformed data to Azure SQL Database.

Figure 4. Interactions columns of the RMA workbook (sample)

After completing the CID and recording all of its component interaction information into the RMA workbook, the engineering team is ready to begin the next phase of the RMA process, the Discover phase.

Discover

The purpose of the Discover phase is to enumerate and record potential failures for every component interaction. This phase is a facilitated brainstorming session that is generally most effective when all engineering disciplines actively participate. Developers are critical participants for this exercise because of their intimate knowledge of system behavior when failures occur.

Information that should be recorded in the RMA workbook during this exercise includes:

- Interaction ID. Determined in the Pre-work phase.
- Interaction name. Determined in the Pre-work phase.
- Failure short name. A short name for the failure type.
- Failure description. A longer description of the failure.
- Response. Information about error handling, alerting, and mitigation(s)/restoration efforts associated with this failure.

Information in the Response column will be used to analyze the effects of each failure in the Rate phase, so it is very important to capture all of the information described in the preceding list.

The following example is for the Contoso service, which is referenced throughout this paper.

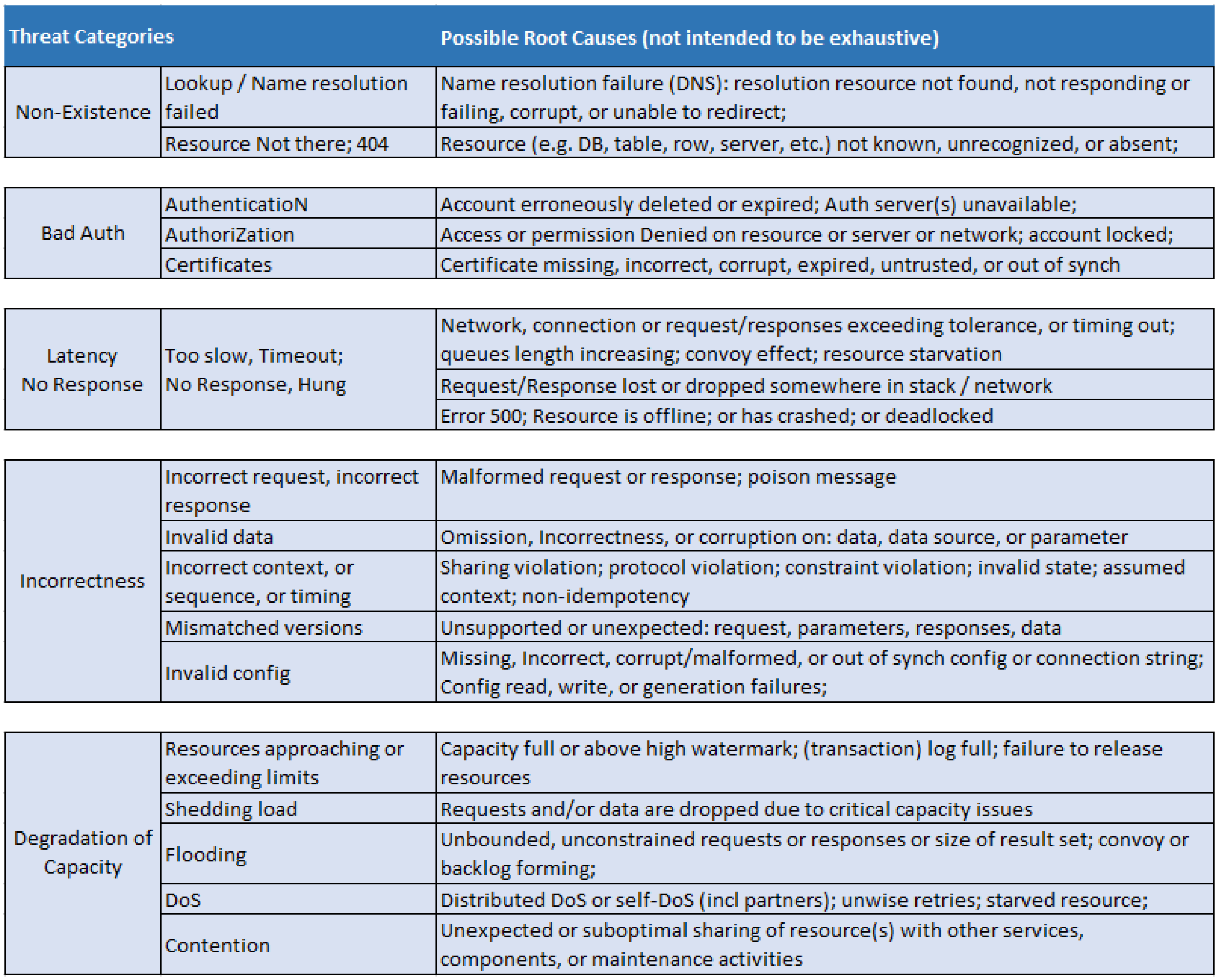
ID	Component/ Dependency Interaction	Failure Short Name	Failure Description	Response
2	Ingestion Roles -> Azure Storage Table	Existence::Name resolution	Azure Storage Table may fail to resolve in DNS for prolonged periods.	Ingestion Instances will cache client data locally on virtual disk until name resolution is restored for Azure Storage Table. Local cache will persist during VM reboot but would be lost upon VN destruction. ContosoMon alerting system will fire a SEV1 alert for name resolution errors to Azure Storage table. Human interaction will be required if condition persists beyond local queue lengths.
2	Ingestion Roles -> Azure Storage Table	Latency::No response	Azure Storage Table may fail to respond for prolonged periods to all Ingestion Role instances.	Ingestion instances will cache client data locally on virtual disc until Azure Storage Table becomes responsive. Local cache will persist during VM reboot but would be lost upon VM destruction. ContosoMon alerting system will fire a SEV1 alert for name resolution errors to Azure Storage Table. Human Interaction will be required if condition persists beyond local queue lengths.

Figure 5. Columns of the RMA workbook used during the failure brainstorming exercise.

A facilitator is needed to ensure the failure brainstorming meeting is productive and captures the right level of detail. The CID will be used and referenced throughout the meeting and will typically be the only thing shown to participants during this phase. The facilitator will record the various failures that are identified during the meeting in the RMA workbook.

A good practice to follow for the brainstorming meeting is to limit the time to <u>no more than 90 minutes</u> to avoid fatigue and a resulting decline in quality. If after 90 minutes not all of the interactions have been examined for failure, experience has shown it is better to stop and schedule a second session to complete this phase.

Although the brainstorming can begin on any interaction (such as one that everyone agrees routinely suffers from reliability issues), it is best to begin at the logical starting point of a scenario and then follow the natural order of interactions from there. The facilitator will lead the participants through the brainstorming activity of identifying the <u>large majority</u> of possible failures. A little structure in this phase will go a long way toward ensuring the exercise is efficient and still comprehensive. To help the facilitator, Trustworthy Computing developed a list (shown in the following figure) of common failures that can be used to help guide the conversation in a repeatable manner for each interaction.



Threat Categories		Possible Root Causes (not intended to be exhaustive)
Non-Existence	Lookup / Name resolution failed	Name resolution failure (DNS): resolution resource not found, not responding or failing, corrupt, or unable to redirect;
	Resource Not there; 404	Resource (e.g. DB, table, row, server, etc.) not known, unrecognized, or absent;
Bad Auth	AuthenticatioN	Account erroneously deleted or expired; Auth server(s) unavailable;
	AuthoriZation	Access or permission Denied on resource or server or network; account locked;
	Certificates	Certificate missing, incorrect, corrupt, expired, untrusted, or out of synch
Latency No Response	Too slow, Timeout; No Response, Hung	Network, connection or request/responses exceeding tolerance, or timing out; queues length increasing; convoy effect; resource starvation
		Request/Response lost or dropped somewhere in stack / network
		Error 500; Resource is offline; or has crashed; or deadlocked
Incorrectness	Incorrect request, incorrect response	Malformed request or response; poison message
	Invalid data	Omission, Incorrectness, or corruption on: data, data source, or parameter
	Incorrect context, or sequence, or timing	Sharing violation; protocol violation; constraint violation; invalid state; assumed context; non-idempotency
	Mismatched versions	Unsupported or unexpected: request, parameters, responses, data
	Invalid config	Missing, Incorrect, corrupt/malformed, or out of synch config or connection string; Config read, write, or generation failures;
Degradation of Capacity	Resources approaching or exceeding limits	Capacity full or above high watermark; (transaction) log full; failure to release resources
	Shedding load	Requests and/or data are dropped due to critical capacity issues
	Flooding	Unbounded, unconstrained requests or responses or size of result set; convoy or backlog forming;
	DoS	Distributed DoS or self-DoS (incl partners); unwise retries; starved resource;
	Contention	Unexpected or suboptimal sharing of resource(s) with other services, components, or maintenance activities

Figure 6. Table of failure categories for use during the Discover phase

The categories are listed in a logical sequence of potential component interactions. For example, when Component A makes a request to Resource B, issues may be encountered in the following logical sequence:

1. Resource B may not exist or cannot be found. If it is found,
2. Component A may not successfully authenticate with B. If it does authenticate,
3. Resource B may be slow or not respond to the request Component A issued,

and so on.

The preceding list is not intended to be an exhaustive catalog of all possible failures. However, if participants think about these failures for every interaction and review each category in this structured manner, recording the results is easier and entire classes of failures will less likely be missed.

One very important point to consider during this exercise is that failures in RMA *do not equate* to root cause. There may be multiple root causes that can lead to a failure of a resource and cause it to appear to be offline. However, in all such cases, the behavior that is encountered by the caller is the same—the resource is offline. The underlying root cause has no bearing on how the caller will respond. Nevertheless, it may be beneficial to record various root causes of a failure type in the Failure Description column for later use when determining the likelihood of the failure type.

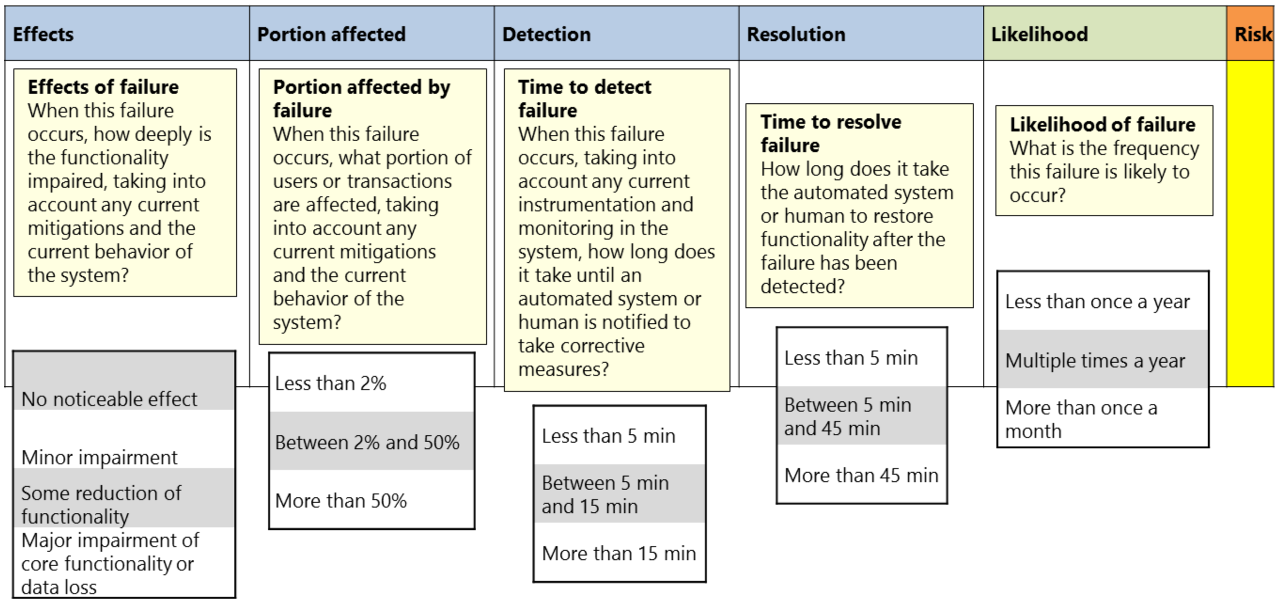
After failures are fully enumerated for all component interactions, the engineering team is ready to move on to the Rate phase.

Rate

The purpose of the Rate phase is to analyze and record the effects that could result from each of the enumerated failures in the Discover phase. The details that were recorded for each failure type in the Response column will provide most of the context needed to complete this task. This exercise will result in a list of calculated risk values for every failure type, which will be used as input for the Act phase.

Again, as in the Discover phase, the facilitator will lead a meeting that includes the same people that took part in enumerating the failures. As before, we recommend this meeting be no more than 90 minutes in length. Typically, the facilitator will display the workbook on a projection screen as they populate the remaining data for each failure type using information from this meeting.

The RMA workbook has several drop-down selection columns that will be populated during this phase. The following figure provides a closer look at each of these columns.



Effects	Portion affected	Detection	Resolution	Likelihood	Risk
Effects of failure When this failure occurs, how deeply is the functionality impaired, taking into account any current mitigations and the current behavior of the system?	**Portion affected by failure** When this failure occurs, what portion of users or transactions are affected, taking into account any current mitigations and the current behavior of the system?	**Time to detect failure** When this failure occurs, taking into account any current instrumentation and monitoring in the system, how long does it take until an automated system or human is notified to take corrective measures?	**Time to resolve failure** How long does it take the automated system or human to restore functionality after the failure has been detected?	**Likelihood of failure** What is the frequency this failure is likely to occur?	
No noticeable effect	Less than 2%	Less than 5 min	Less than 5 min	Less than once a year	
Minor impairment	Between 2% and 50%	Between 5 min and 15 min	Between 5 min and 45 min	Multiple times a year	
Some reduction of functionality	More than 50%	More than 15 min	More than 45 min	More than once a month	
Major impairment of core functionality or data loss					

Figure 7. Columns of the RMA workbook used during the failure effects analysis exercise

The Risk column is a calculated value derived from the other five columns. The idea behind this calculation is to assess risk as the product of the failure *impact* and *likelihood*. Likelihood is a straightforward concept represented by a single value in the workbook (selected from the Likelihood drop-down list). However, the impact of a failure can be decomposed into several elements.

The RMA process is used to evaluate the impact of potential failures during product design in much the same way that a problem management team might assess the impact of an outage for a cloud service already in production. The following key questions are routinely asked about the impact of an outage:

- What were the discernible effects to the user or business-critical process? Was this outage a minor annoyance or performance impact, or was this outage something that prevented key user scenarios from completing? What was the *depth* of impact?
- How large was the scope of the impact? Were only a few customers or transactions affected, or was the scope of impact widespread? What was the *breadth* of impact?
- How long did it take a person or automated system to become aware of the failure? What was the time to detect (TTD)?
- Once detected, how long did it take to recover the service and restore functionality? What was the time to recover (TTR)[8]?

These questions apply to potential failures in the RMA workbook, and they map respectively to the columns Effects, Portion affected, Detection, and Resolution.

[8] TTR is the time it takes to restore functionality to the customer, not necessarily the time it takes for the failure to be completely fixed.

It is important for the engineering team to remember that all of the columns are completely independent of one another. The facilitator should ensure that evaluations of the Effects and Portion affected columns in particular are not conflated during the exercise. For example, it is perfectly reasonable to expect a team to analyze a failure that would result in only a slight degradation of service quality while affecting practically every user of the system. Conversely, the team could be analyzing a failure that affects only a single user (or tiny fraction of the total transactions being processed) in the service, but such a failure might have very serious consequences in terms of data integrity, including data loss.

The numerical calculation of the Impact and Likelihood columns is fixed in the model; the drop-down values are fixed to three choices[9] (four in the case of Effects). However, teams can change the associated text of the drop-down selections to better reflect characteristics of their cloud service. For example, the Detection column defaults of *Less than 5 min*, *Between 5 min and 15 min*, and *More than 15 min* can be changed by a team with a service failure detection time that is nearly instantaneous to *Less than 50 milliseconds*, *Between 50 milliseconds and 500 milliseconds,* and *More than 500 milliseconds,* respectively. At the beginning of the Rate phase, the drop-down selections for all of these columns should be reviewed.

When the effects analysis is done, the Rate phase is complete. The team now has a prioritized list of failures to use as input for the Act phase.

Act

The purpose of this final phase is to take action on the items discovered during the resilience modeling portion of the RMA process and make tangible investments to improve the reliability of the cloud service.

The risk ranking of the failures captured in the RMA workbook during the Rate phase provides guidance on the priority of potential engineering investments. You can create tracking items by evaluating all of the high and medium-risk failures, and then generate work items for engineering resources. Sometimes it is possible to transform high-risk failures into medium-risk failures by reducing the time to detect (TTD) or time to recover (TTR) values. Many improvements in these two areas can be achieved by making investments in, or changes to, monitoring systems. Work items may also be called out for instrumentation and telemetry, detection and monitoring, and mitigations to address specific root causes or accelerate recovery. Work items can also introduce new test cases, which will need to be factored into the test harness for the service. Work items may also result in requests for architectural changes if RMA is performed early enough in the design stage.

As mentioned earlier, one of the ancillary benefits of RMA is that it produces artifacts that can be beneficial to those beyond the feature crew, such as on-call support personnel or operations

[9] The drop-down choices are expressed as low, medium, high to enable teams to make the selection quickly without becoming mired in precision calculations.

personnel. By compelling the engineering team to revalidate the architecture and design during the Pre-work phase, the CID will often provide additional insights beyond the current design, architecture, or as-built diagrams that are typically in use today.

The test organization can use the RMA workbook to target fault injection testing, because it captures metadata about the quality of a component interaction's response to failure in the cloud service. Any component that incorporates failure mitigations could be a target for fault injection to verify the quality and effectiveness of those mitigations.

Implementation considerations

RMA is a streamlined and simple approach for teams to use to increase the resilience of their cloud service by identifying and analyzing potential failures. Although implementation of the process is flexible in a number of areas, teams should carefully consider the topics of timing, approach, roles, and responsibilities before beginning.

Timing

If your team is accustomed to performing security threat modeling as described in the Microsoft Security Development Lifecycle, you will already have a good sense of the cadence of RMA. Similar to the recommendations for security threat modeling, Trustworthy Computing recommends revisiting your resilience model every six months (no less than, and definitely at least once annually), whenever making significant architectural or functionality changes, or when suffering live site issues that prevent you from consistently meeting your customer availability goals.

For new services, or services undergoing a major revision, teams should consider implementing RMA after the architecture has been mapped out and the initial design has been proposed but before most of the coding has been completed. It is more cost-effective to design resilience into the service rather than to react to failures after the service is already in operation. Cloud service teams that are already in production should consider implementing RMA immediately; there is no need to wait until the next major service revision to start applying the RMA methodology. The prioritized failure list that is generated by the RMA process provides a wealth of opportunities for making targeted investments in instrumentation, detection, mitigation, and recovery, many of which may be implemented quickly. Furthermore, the knowledge gained from conducting the analysis on a product currently in production will be highly valuable as input for future planning and development cycles. Many teams with services that are already in production are also interested in fault injection. The outputs of the RMA process provide excellent input for fault injection testing to validate the effectiveness of the current failure mitigation techniques.

Approach

RMA is flexible enough to apply to any facet of a cloud service. Consider each of the following variations in scope when determining the time investment you are willing to make in RMA:

- End-to-end scenario. Service reliability should be evaluated from the customers' perspective. Consequently, teams often want to focus on failures that affect key user work streams or ones in which the organization may be most affected by reliability-related incidents. If you apply this approach, you will prioritize reliability-related engineering investments, which are regarded as being important to the consumers of the service. However, end-to-end scenarios commonly traverse many components and service boundaries. To reduce the need to secure participation from so many people across many different teams, the scenario can be divided into sub-scenarios to achieve the desired level of efficiency.
- Cloud service boundaries. Teams are often most worried about reliability-related issues at the boundaries of their cloud service. These boundaries are the intersection points between their cloud service components and third-party or partner-supplied services. The RMA exercise can be scoped such that all service integration points are modeled for failures. The benefit of this approach is that these integration points are typically where services are often most susceptible to failures. In addition, this approach usually requires fewer participants to complete the exercise. The downside to this approach is that certain components comprising key user scenarios or business-critical work streams may not be modeled sufficiently.
- Component by component. A third option that teams can choose is to target RMA at a small number of components of the cloud service at one time. Typically, teams will start with components that are experiencing the most reliability issues (if the service is already in production), or with a component for which they want to gain reliability awareness early in the design phase. Scoping the RMA exercise to a single cloud service component has the benefit of requiring fewer participants and can be completed at lower cost. This approach is similar to modeling cloud service boundaries, but at the component level. Key work streams that span multiple components get complete coverage when each component team completes RMA over time. However, additional analysis of how the components interact is needed to ensure detection, mitigation, and recovery efforts are complementary at each integration point.

Roles and responsibilities

The RMA exercise is most effective when representatives from all engineering disciplines are included in the process, which enables varied perspectives from each role to be expressed during the various RMA phases. Including all engineering disciplines will help ensure a more comprehensive outcome.

There is one discipline that absolutely must participate in the RMA process, and that is the development discipline. Typically, developers (or component owners) are best suited to speak to the details of how a system will behave when a failure is encountered.

A facilitator should drive the process from start to finish and assign work items to others at the end of the process. This person can be from any discipline, but must possess the ability to carefully guide the conversations between the various engineering disciplines so the work items assigned at the conclusion of the exercise produce the greatest reliability-related gains while consuming the fewest possible resources to do so.

Call to action

Embrace the engineering principles associated with recovery-oriented computing.

Incorporate RMA into your software development lifecycle: model failures and the effects of those failures, decide which failures must be addressed by following the process described in this paper, and make the necessary engineering investments to mitigate high priority risks.

Demonstrate the value of applying RMA in your environment by refocusing the engineering resources that are reclaimed by virtue of not having to continuously respond to failures, and apply them to the design and subsequent delivery of customer-facing innovation at an even faster pace than before.

Share the RMA experience with other engineering teams who have not yet made the leap, and help them understand how to apply the methodology and benefit from the productivity gains just as you have.

Conclusion

Cloud computing can be characterized as a complex ecosystem that consists of shared infrastructure and loosely coupled but tightly-integrated dependencies between applications, many of which will be outside the provider's direct control. As the adoption of cloud-based services continues to grow, customer expectations for utility-grade service availability remain high, despite the obvious challenges associated with delivering a reliable experience 24 hours a day, seven days a week, and 365 days a year.

In spite of rigorously applying well-known enterprise-based software design practices, testing components during development, and implementing redundant infrastructure and replicated copies of data, interruptions will inevitably occur. There is mounting evidence that being able to avoid failure altogether is a flawed assumption; media articles continue to appear that describe failures of popular cloud services, and cloud service providers routinely supply explanations of what went wrong, why it went wrong, and how they plan to avoid future occurrences. Yet similar failures continue to happen.

Software architects must embrace the notion that underlying computing resources can—and most certainly will—simply disappear without warning, and potentially for long periods. Simply stated, software architects need to accept this new era of unpredictability and design accordingly.

The methodology described in this paper has proven to be highly effective in helping cloud service engineering teams understand how to cope with persistent reliability-related threats. It can also help prioritize the necessary engineering investments intended to reduce—or even eliminate—the impact of those threats from the customers' perspective.

The primary benefit of adopting RMA versus a more targeted approach comprised of only fault modeling and root cause analysis is that the cloud service design team emerges from the exercise with a more comprehensive analysis based on the deep exploration of every aspect of the service being built. The results of the RMA process provide the team with a deeper understanding of where the known failure points are, what the impact of the known failure modes is likely to be, and most importantly, the order in which to tackle these potential risks to produce the most reliable outcome in the shortest amount of time.

We know service interruptions are inevitable. We are designing and building our services using methodologies like RMA to help minimize the impact to customers when such interruptions occur.

Additional resources

Guidance for resilient cloud architecture on Azure

- Windows Azure architectural guidance: www.windowsazure.com/en-us/develop/net/architecture/
- FailSafe (on Channel 9): channel9.msdn.com/series/failsafe

Recovery-oriented computing: roc.cs.berkeley.edu/

Templates

- component interaction diagram (CID): aka.ms/CID
- Sample RMA workbook: aka.ms/RMAworkbook

Trustworthy Computing reliability: www.microsoft.com/reliability

Authors and contributors

DAVID BILLS – Microsoft Trustworthy Computing

SEAN FOY – Microsoft Trustworthy Computing

MARGARET LI – Microsoft Trustworthy Computing

MARC MERCURI – Microsoft Consulting Services

JASON WESCOTT – Microsoft Trustworthy Computing