Real time failure point detection and operational health monitoring of Cloud Infra

Vittalkumar Mirajkar
VitalkumarrMirajkar@gmail.comm

Srivatsan P
srivatsan.p@skyhighsecurity.com

Abstract

Cloud Infra down time directly results in lost Engineering productivity time. Infra resilience is every engineering team's responsibility. In a complex cloud software solution, there are interconnected dependencies. This exposes us to multi point failure.

Current cloud infra monitoring process has challenges, the tools monitor individual parameters, and a real time co-relation is missing. In a complex interconnected services infra, a component failure has cascading effect. In case of an infra outage, reaching actual issue causing component is major portion of the troubleshooting time.

Proactive health monitoring is limited to checking feature running status post deployment. There is a gap how the system behaves during the actual deployment.

Our proposed solution to building robust software health monitoring system draws parallel from the health care process. During a critical operation, the entire patient monitoring is kept active to keep the overall situation under control.

On a similar analogy, in this paper we present how we addressed infra failure issues and real time monitoring during active deployment and regular time. We corelate Dependency tree, AWS Realtime logs, db queries, Jenkins and Harness logs, network connectivity, CPU consumption and many other attributes. We take inputs from multiple monitoring systems and corelate the data for better infra health status.

Biography

Vittalkumar Mirajkar is a Sr Manager at Skyhigh Security, with 18+ years of testing experience ranging from device driver testing, application testing and server testing. He specializes in testing security products and building CI/CD systems. His area of interest is performance testing, soak testing, data analysis and exploratory testing.

Srivatsan Parthasarathy is an SDE at Skyhigh Security, with 10+ years of experience in Automation Framework and tool Development. He specializes in building end-to-end frameworks and tools for automation testing for desktop, web, and mobile applications. He has expertise working with Cloud Providers and a deep understanding of Cloud FinOps.

1. Introduction

In cloud world, infrastructure outages are costly. They not only cause service disruption, outage cause reputation loss, and monetary loss as well. Not to account for the endless time spent by engineering teams to troubleshoot and customer sentiment management.

Cloud outages can be classified into broadly, but not limited into following categories.

External outage categories:

- Cascading failure of dependencies
- Outage caused by service providers.
- Cloud infrastructure failure due to resource starvation
- Network infrastructure issues

Internal outages categories

- Product issues
- Deployment related failures
- Resource starvation of nodes due to resource consumption due to prolonged running time.

In following chapters, we understand in some detail the cause for each of these failure categories.

Not all factors can be controlled, however as a SAAS provider some of these failure points can be predicted and mitigated.

2. Cloud Outage Classification

Below is a quick look at some of the reason for both External Outages and Internal outages,

2.1 External Outages reasons:

Factors contributing to external Outages, though not in control can be broadly classified into (but limited to) following categories.

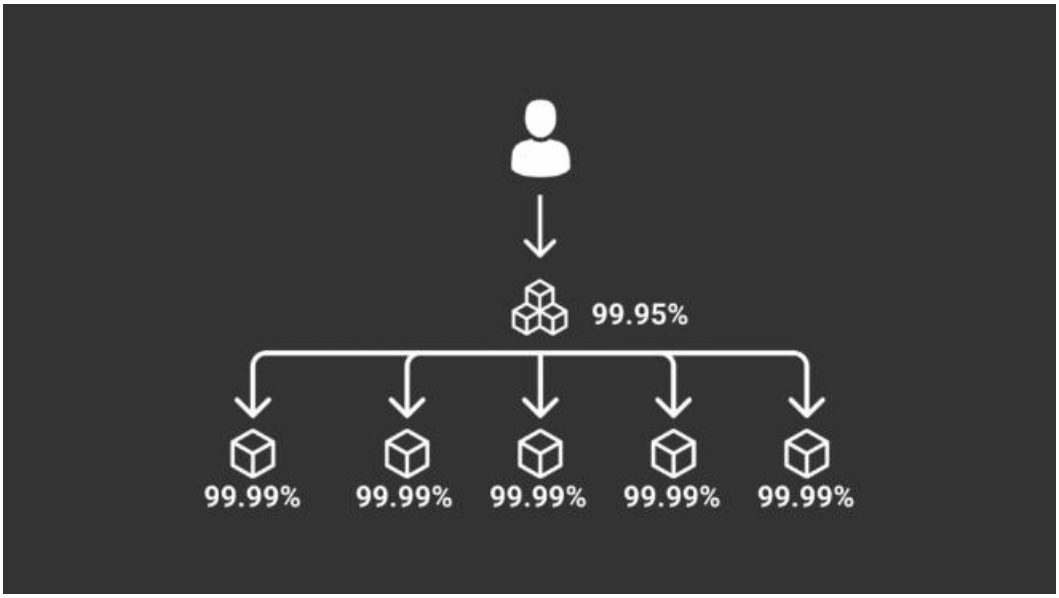
2.1.1 Cascading failure of dependencies

A SaaS solution is built on other apps. On an average digital business relies on 137 different service providers to power their software (Martens 2023). This introduces a large set of external failure points. Even when the SLA is 99.99% availability for each of the dependencies, in case of cascading failures, we need to move beyond SLA and determine a composite SLA.

*Composite SLA = No Of Dependencies * 0.9999 (assuming 99.99% availability)*

For example, an app with 5 dependencies, all with four nines of availability, can only offer a 99.95% SLA itself.

Composite SLA = 5* 0.9999 = 0.9995, this is 99.95 % availability.



Service Availability: Even when you have Service Level Availability for all upstream dependencies at the highest level of 99.99%, it does not mean Service uptime (Neumeier 2018).

(System) Uptime != (Service) Availability

Availability Level	Uptime	Downtime per Year	Downtime per Day
1 Nine	90%	36.5 days	2.4 hours
2 Nines	99%	3.65 days	14 minutes
3 Nines	99.90%	8.76 hours	86 seconds
4 Nines	99.99%	52.6 minutes	8.6 seconds
5 Nines	100.00%	5.25 minutes	0.86 seconds

The higher the service availability opted or redundant fail-safe plan, would directly result in cost increases. Added, we need to have contingency plan of having multiple service providers and real time roll over playbook.

2.1.2 Outage caused by service providers:

Almost all cloud services providers have had outages (Korolov 2022) (Hicks 2022) (Taylor 2022). These are caused by various reasons, some popular one causes are.

- Network upgrades causing issues.
- Routine maintenance causing service disruption.
- CSP services failing and causing cascading effects on end users.

And many many more. While we do not have control in such a case till services resume at a CSP end. One potential de-risking factor to host services on multiple CSP. This though have a cost factor associated.

2.1.3 Resource Starvation:

Insufficient resource sizing could lead to processing choke points, compute starvation or network choke points or just not enough IOP's being factored in. Any of these could lead to service disruption.

While right sizing is the simplest solution, get the right sizing is not simple.

2.1.4 Network Outages:

In last couple of years Network outages are seen among even the biggest cloud solutions provider. A wrong Network routers configuration causing network shutdown, a surge in network activity (Taylor 2022), causing the entire service to collapse. Network outages led to cascading failures for downstream service to be left in unreachable state.

There is limited action where Downstream SaaS product can do in such a case. Visibility into upstream dependencies help us know if the failure is "Is It Me or Them?" **(Martens 2023)**. Identifying the root cause is saves a lot of time lost in debugging. However, once the fault line is established as "Them", there is very little to do except for waiting for the upstream services to resume.

Though these outages constitute a major section of outages, this paper does not refer to addressing this issue. The focus of this paper is addressing areas which is under the control of SaaS application development team a.k.a internal outages.

2.2 Internal Outages reasons:

Internal outages can be anything where the root cause originates from within the company. Broad categories (but not limited to) are:

- Product issues
- Deployment related failures
- Resource starvation due to resource consumption of prolonged running time.

2.2.1 Product Issue:

Product feature failure directly resulting in service failure. While service monitoring will flag a service failure, but it is after the outage or service disruption. This disruption broadly can be controlled with better test infrastructure mimicking production load, production conditions.

Our Paper focus area is for below mentioned TWO Internal Failure Points.

2.2.2 Deployment related failures:

This is a unique case where maintenance window or a deployment window turns into an outage. This is due to not all the factors being considered during the product deployment. There could be unforeseen conditions which arise at run time, causing service outages.

Visibility into infrastructure, upstream dependency mitigates to some degree, but not fully covering this scope.

2.2.3 Resource starvation

Under resource planning, be it compute, network bandwidth, or storage creates bottlenecks resulting is service failures. This can be resolved by rightsizing. But as mentioned earlier is not that easy. Being dynamic in meeting resource demand is the best-case scenario.

Again, an comprehensive visibility into infrastructure and services helps us to scale up or down the resources. However, this does not trigger until a certain threshold is reached.

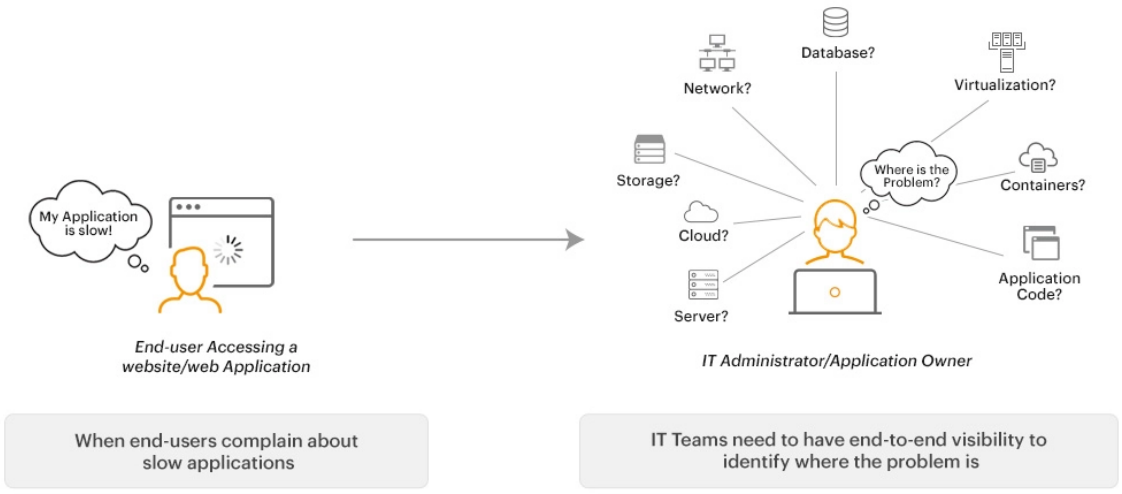
Not much has been proposed to mitigate Internal Outages reasons. Visibility into services and infrastructure is wildly consider as the go to in addressing this.

In the next chapter, we will look at what is the current visibility approaches and its shortcoming which we propose to solve.

3. Current Visibility approach and challenges

Effortless monitoring and Seamless troubleshooting, visibility from infrastructure to application both on external dependencies and internal dependencies is the key to having outages under control. This is easier said than done, as current tools used have limitation.

What we expect as SaaS solution provider when we talk about Visibility (Ellis 2022)



Despite the best planning, current monitoring and visibility challenges are,

- There are multiple tools used in monitoring.
- Still 1/3rd issues detected manually.
- 1/4th SaaS solution provider don't have full visibility (Basteri 2022)

There are just too many tools to be monitored and any real time correlation needs to be done by the engineer, introducing human elements into decision-making. The proposed solution to building a robust infra monitoring is a **2-track** problem.

- Address the correlation of existing multiple tools for better real time decision making.
- Proactively determine node failure by introducing real time monitoring, avoid maintenance windows flipping into outage windows.

4. Visibility across the spectrum and proactive node failure detection

There are multiple tools to do multiple monitoring, and all give out data into different tangents. These data points need to be corelated to make information readily accessible in decision making. This will greatly help in avoiding failures during maintenance windows where unforeseen issue arises turning a maintenance activity into managing outage.

Drawing our analogy from patient monitoring system, where entire health of patient is monitoring before operating on the patient, irrespective of which area is being operated. This is done to ensure we don't have cascading failures when the focus is on fixing / correcting a certain issue. If this is not done, a simple exercise can turn into crisis management.

Musing on similar approach we propose solution for the following TWO use cases.

1. During upgrade / maintenance window how to ensure there are no unforeseen failures leading to outages
2. Proactively detect failure of node based on its node health (resource consumption)

4.1 During product deployment / upgrade window:

Monitor overall health of the infra to know if the systems which are to be patched are operating under near ideal condition. This ideal state is decided based.

- Current CPU consumed.
- Monitoring memory consumed vs available memory.
- Current event in processing queue
- Are the upstream dependencies fully functional and nodes in healthy condition for the component under question to undergo maintenance.

There are multiple monitoring utilities which are scattered and capture individual parameters only. Realtime co-relation is missing.

Irrespective of any activity (patching or upgrading), following checklist could provide a guideline.

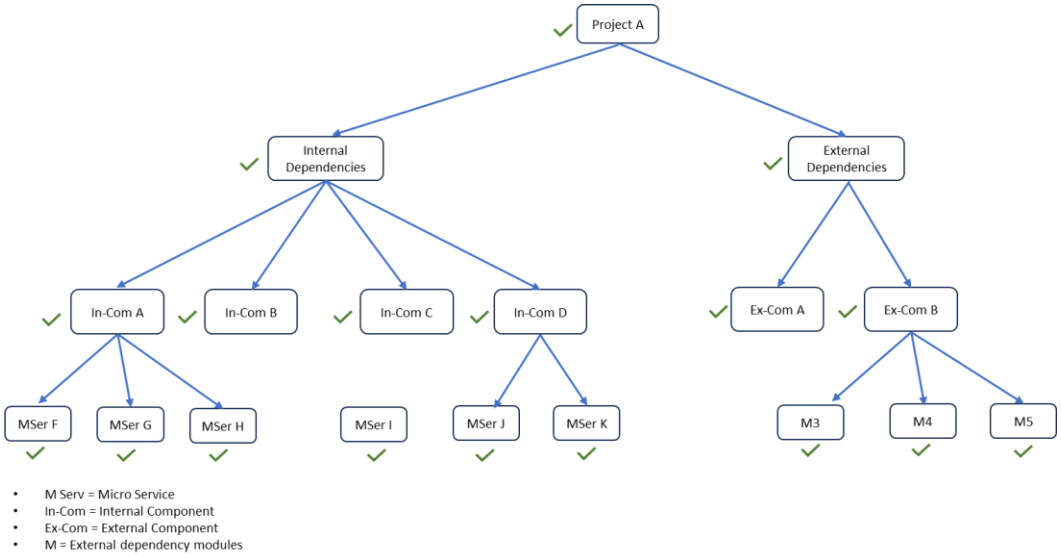
Ensure current state of the infra is heathy, this can be ensured by

- Current state of services running condition.
- CPU consumption < 60 %
- Memory consumption < 60 % of overall capacity
- Network availability < 50 %

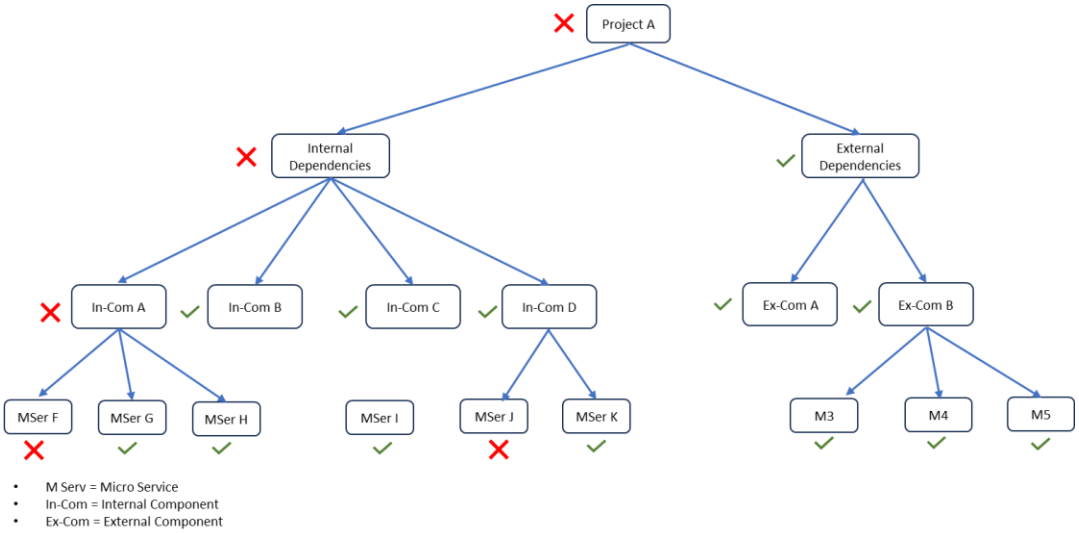
Once a stable state is established, infrastructure is ready for upgrade or patching. Beyond infrastructure monitoring, we need to monitor product upstream / downstream dependencies and their real time running states.

Below is the design of our design for Cloud Resource Monitoring and Service Dependency tree monitoring.

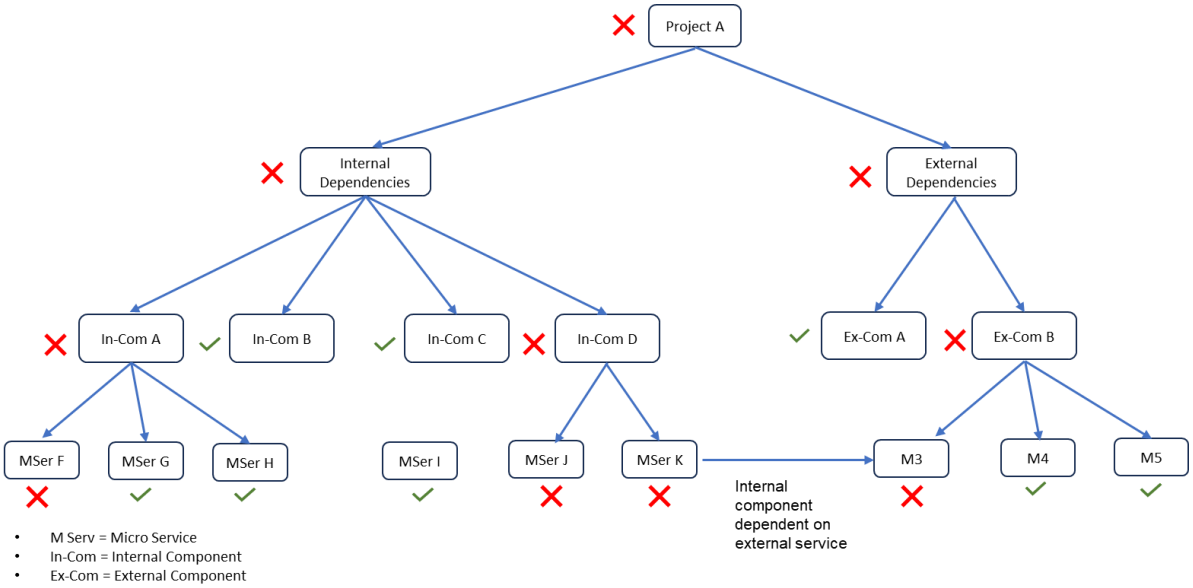
When all internal and external dependencies are Operational



When some internal and external dependencies are Non-Operational (Specifically when an internal dependency is down)



This logic can be extended to external dependencies as well. This tracking mechanism is true for both Infrastructure dependencies as well as Software Service dependencies, provided this is an clear method to establish its operation status.



Once Infra stability state is achieved, handling service dependency order is the is the next critical item in the flow. The sequence of draining the queue and downstream services shutdown is critical. Similarly, once the upgrade is completed, top-down sequence of restarting the flow is critical to getting back a working infra back online.

One thumb rule to be followed:

- Before upgrade: Shutdown Downstream dependencies 1st
- Upgrade
- After upgrade: Restart Downstream dependencies.

4.2 Proactive failure detection

A service failure could be because of following two issues.

- Software failure
- Infrastructure failure on which the software is hosted.

4.2.1 Software failure:

A service failure during deployment is easier to detect, whereas when a functional system fails, it takes time to isolate the problem. A real time service running status dashboard with clear upstream and downstream dependencies outlined helps quickly isolate the problem origin.

A node could be in running condition, however nonfunctional. Service running state alone might not be enough to establish infrastructure being functional. A periodic smoke test running across all critical paths covered, can aid a long way to determine if the nodes are fully functional.

4.2.2 Infrastructure failure:

During the normal working, because of any of the following (but not limited to) reason we may experience service disruption or node failure.

- Processing threads are hung some time from prolonged duration of usage.
- There is un-anticipated network spike / processing spike, choking the queues.
- Due to gradual resource consumption, the node is heading towards a failure, but not yet failed. Current monitoring system wait till the node under question reaches RED zone before triggering an alarm. Can this be preempted and highlighted when the node is in the YELLOW zone of resource consumption. This approach will enable us preempt node failures.

5. Implementation details and results

We implemented our model on one of our QA cloud infrastructures having more than 400+ nodes and running 140+ microservices.

We monitored,

- Infrastructure node level resource consumption
- 140+ micro service functional status

This greatly helped in real time health monitoring of the infrastructure. If one of the nodes under observation, health starts deteriorating, leading to failure, we can detect this condition real time and mitigate the same. Also, during the regular operational status, periodic sweeps of the environment with a smoke test ensured all the servers are not just running but also in functional state.

By this dual approach, we were able achieve.

- Environment stability and service availability, functional up time increased from 90% to 95%.
- We were able to eliminate resource starvation-based outage by >95%

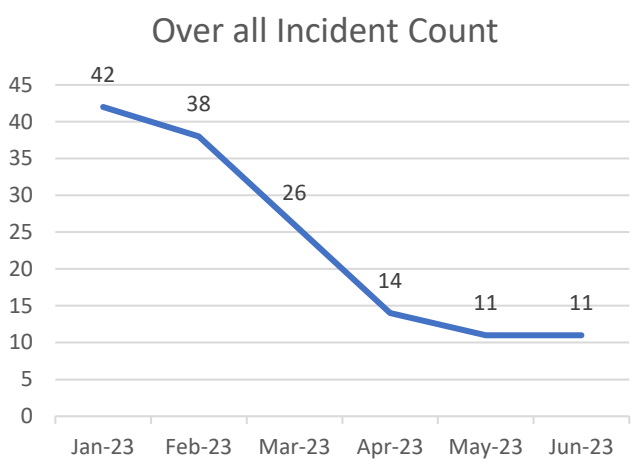
Following data parameters was measured spanning across 2 quarters.

- Total Outage Incident Count
- Post deployment failure
- Internal Service dependency related failures
- External Dependency failures
- Resource starvation related failures
- Time taken for debugging service failures (in hrs)

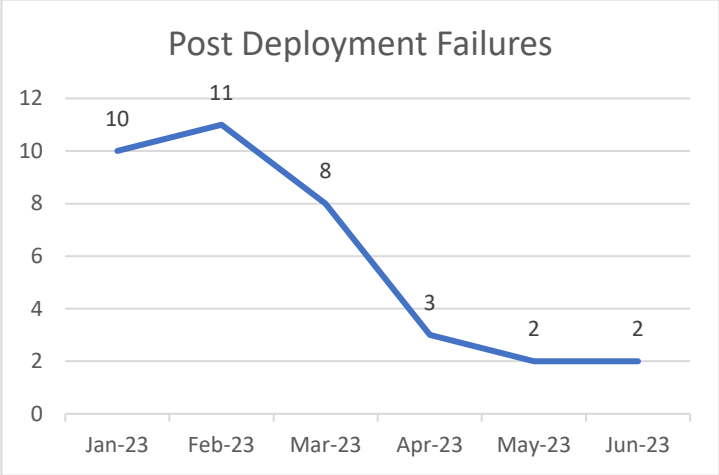
Q1 (Jan – Feb – March) this data represent condition before applying our monitoring framework.

Q2 (April – May – June), new monitoring system was implemented. We noticed considerable reductions in the time taken to identify and isolate the failure. The time taken to troubleshoot the failures also reduced.

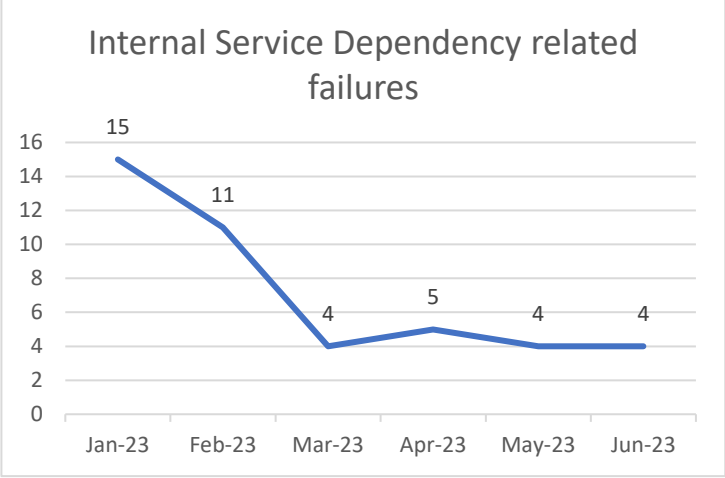
Below is data set captured for Q1 vs Q2 data pointers for the above said areas. There is an improvement observed.



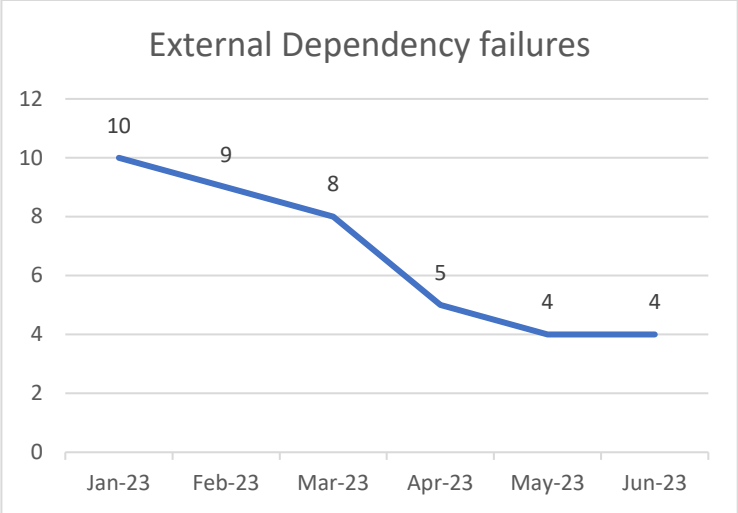
- In 2 Quarters we have observed overall drastic reduction in Incident count.
- Some of this reduction is direct related to the proactive monitoring system implemented.



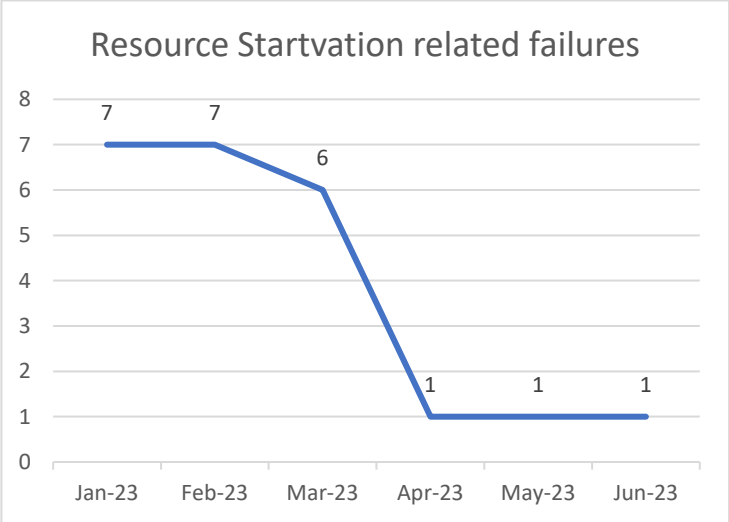
- Post deployment failure which are detected real time as there is real time 360degree monitoring of services.
- Post deployment failures are now handled in deployment window only.



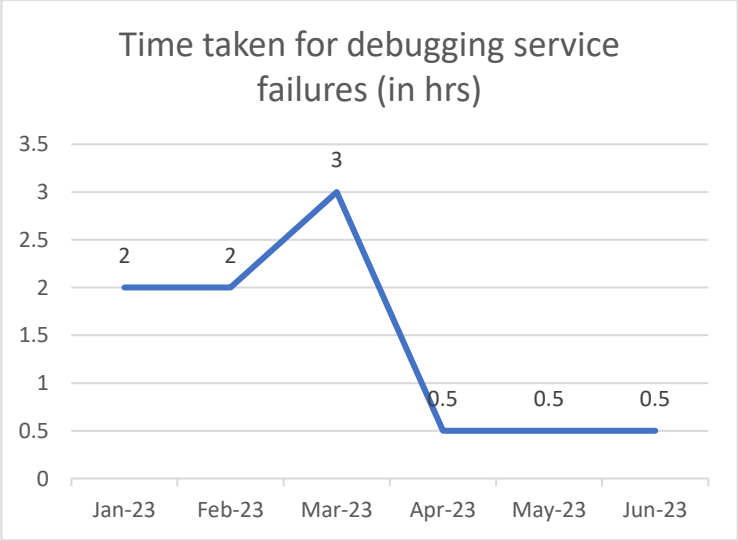
- Internal service-related failures are now instantly detected to the actual root cause.
- Service status check and interdependencies mapping is the key in helping in achieve ~80% reduction in incidents.



- There is little we can do when an External Failure is detected.
- Having visibility into 3rd party Infra Dependency / Software Dependency will help us address the question, "**Is it US or THEM**' as the root cause.
- Our monitoring system now gives instant insight into knowing the origin of the issue.



- Using the proactive monitoring, we have observed resource starvation related node failures are almost eliminated.
- We have been able Detect and Mitigate potential failures well within time.



- One major advantage we have seen is in time to reach root cause of issue.
- The monitoring system directly exposes the trigger service which has caused the failure.

6 Learnings and Take aways:

Some of our key take aways in our journey of getting our cloud infrastructure stable.

1. There is no one tool which can help you monitoring everything.
2. Correlation between data generated from different tools need to be automated.
3. Service status visibility is a must and most importantly service dependencies should be mapped.
4. Proactive hardware infra monitoring can help identify nodes which are leading towards failures, thereby giving us time to mitigate potential failures.
5. You will not be able to avoid all failures. When service failures happening how quickly we are isolating the root cause is equally important.
6. Periodic smoke tests on production environments helps us proactively check functional working status of all the services.

References

Basteri, Alicia. 2022. *2022 Observability Forecast.* September. Accessed July 20, 2023. https://newrelic.com/observability-forecast/2022/about-this-report.

Ellis, Brent. 2022. *SaaS Outages: When Lightning Strikes, Thunder Rolls.* April 12. Accessed July 25, 2023. https://www.forrester.com/blogs/saas-outages-when-lightning-strikes-thunder-rolls/.

Hicks, Mike. 2022. *Seven Outages That Shook Up 2021.* January 6. Accessed June 12, 2023. https://www.thousandeyes.com/blog/seven-outages-shook-up-2021.

Korolov, Maria Korolov and Alex. 2022. *Top 10 outages of 2021.* January 31. Accessed July 5, 2023. https://www.networkworld.com/article/3648352/top-10-outages-of-2021.html.

Martens, Jeff. 2023. *The Overlooked Culprit Behind 70% of SaaS Outages.* Feb 23. Accessed June 10, 2023. https://metrist.io/blog/the-overlooked-culprit-behind-70-of-saas-outages/`.

Neumeier, Sascha. 2018. *The Uptime Myth: Why Uptime Does Not Mean Availability.* October 8. Accessed June 25, 2023. https://blog.paessler.com/why-uptime-does-not-mean-availability.

Taylor, Twain. 2022. *7 Biggest Cloud Outages of the Past Year.* February 11. Accessed July 15, 2023. https://techgenix.com/7-biggest-cloud-outages-services-2021/.