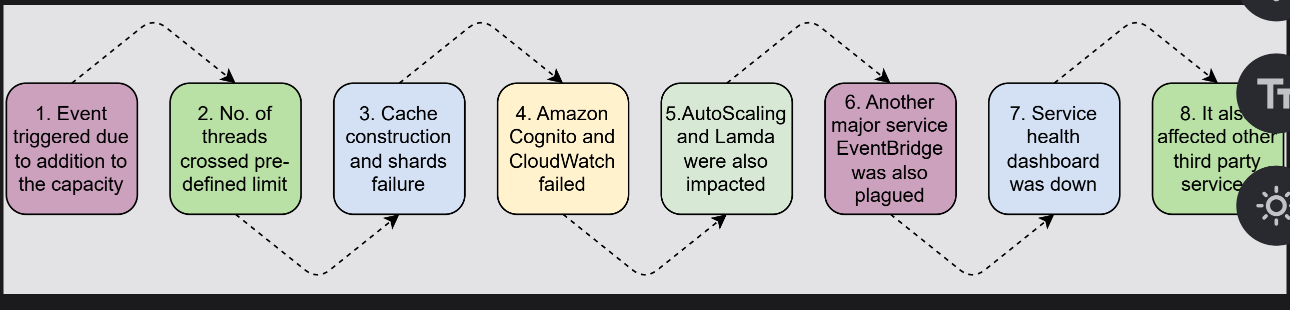
AWS Kinesis Outage Affecting Many Organizations

Learn the causes and analysis of a major AWS Kinesis outage.

Amazon Kinesis allows us to aggregate, process, and analyze real-time streaming data to get timely insights and react quickly to the information it provides. It continuously captures gigabytes of data from hundreds of thousands of sources per second. The Kinesis service's frontend handles authentication, throttling, and distributes workloads to its back-end "workhorse" cluster via database sharding. On November 25, 2020, the Amazon Kinesis service was disrupted in the US-East-1 (Northern Virginia) region, affecting thousands of other third-party services. The failure was significant enough to take out a large portion of Internet services.

Sequence of events

- According to Amazon, the event was triggered by adding a small capacity to the AWS front-end fleet of servers, scheduled from 2:44 a.m. PST to 3:47 a.m. PST.
- The addition of the new capacity caused all the servers in the fleet to exceed the maximum number of threads that are allowed by an operating system configuration.
- Due to exceeding the limit on threads, cache construction was failing to complete, and front-end servers were ending with useless shard maps that left them unable to route requests to back-end clusters.
- Other primary Amazon services also stopped working, including Amazon Cognito and CloudWatch.
    - Kinesis Data Streams is used by Amazon Cognito to gather and analyze API usage patterns. Because of the long-running issue with Kinesis Data Streams, a hidden error in the buffering code—which is required for Cognito services—caused the Cognito web servers to start blocking on the backlogged Kinesis Data Stream buffers. As a result, Cognito customers witnessed an increase in API failures and latencies for Cognito user pools and identity pools, making it impossible for external users to authenticate or receive temporary AWS credentials.
    - CloudWatch uses Kinesis Data Streams to process metrics and log data. The `PutMetricData` and `PutLogEvents` APIs in CloudWatch encountered higher error rates and latencies, and alerts were set to `INSUFFICIENT DATA`. The great majority of metrics were unable to be processed due to higher error rates and latencies. When CloudWatch was experiencing these greater issues, internal and external clients couldn’t persist all metric data to the CloudWatch service.

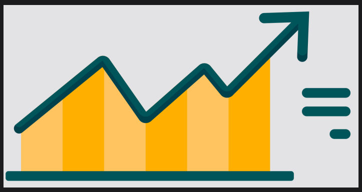

1. Event triggered due to addition to the capacity
2. No. of threads crossed pre-defined limit
3. Cache construction and shards failure
4. Amazon Cognito and CloudWatch failed
5.AutoScaling and Lamda were also impacted
6. Another major service EventBridge was also plagued
7. Service health dashboard was down
8. It als affected other third party service

- Due to the problems with CloudWatch metrics, two services were also impacted. First, AutoScaling policies that rely on CloudWatch measurements suffered delays. Second, Lambda experienced the effect. Currently, posting metric data to CloudWatch is required as a part of Lambda function invocation. If CloudWatch is unavailable, Lambda metric agents are meant to buffer metric data locally for a period of time. The metric data buffering became so large that it generated memory congestion on the underlying service hosts utilized for Lambda function invocations, leading to higher error rates.
- Increased API failures and event processing delays plagued CloudWatch Events and EventBridge. EventBridge is used by Elastic Container Service (ECS) and Elastic Kubernetes Service (EKS) to drive internal processes for managing client clusters and jobs. This has an impact on new cluster provisioning, existing cluster scaling, and task deprovisioning.
- Apart from the service issues, Amazon also experienced delays in communicating service status to their customers. Amazon uses two dashboards for communication with their customers, Service Health Dashboard and Personal Health Dashboard. The Service Health Dashboard informs all customers of events, such as the current one. It was down because of its dependency on Cognito, which was impacted by this event.
- Apart from the disruption in the Amazon services, this also had a ripple effect on thousands of third-party online services, applications, and websites. This includes Adobe Spark, Acorns, Coinbase, *Washington Post*, and hundreds of such services.

Analysis

- **Complexity of enhancing scalability**: We've been stressing the need fo horizontal scalability in all of our design problems. This outage event shows that in practice, adding more capacity to a serving cluster so that we don deny any client requests can be challenging and can have unintended side effects.

- **The need for a trained team**: Training the production team to deal with such unforeseen situations is a challenging task, but it's worth it and can result in expedited recovery. In addition, randomly restarting the front-end servers and suspecting that the memory pressure causes the problem alludes to the challenges of reaching the root causes under the stress of time.
- **Reading from authoritative servers during bootstrap**: During the bootstrap process, it's a good idea to take data from the authoritative metadata store rather than from the front-end servers to reduce the impact of such failures.
- **Identification of faults in initial stages**: There's a need for automated processes to identify the causes of failure within the initial stages of its occurrence.
- **Proper testing mechanisms can reduce the severity of the fault**: The new capacity that's causing the servers in the fleet to exceed the maximum number of threads is a potential bug that should have been fixed earlier than before restarting the servers. This reflects the inability to properly test the system. There should be a kind of simulator to test all cases before deploying additional capacity into the fleet.

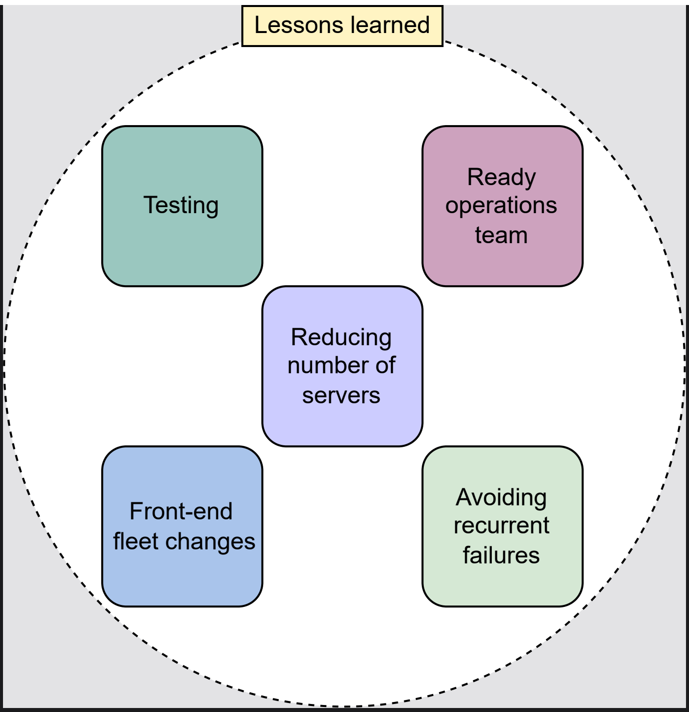


- **Finding potential bugs before planned events**: The issue with Kinesis Data Streams triggered a latent bug in the buffering code that caused the Cognito web servers to begin to block on the backlogged Kinesis Data Stream buffers. The potential bugs should be identified and fixed before critical planned events, such as adding capacity or software and hardwar maintenance.

- **Adopting automated processes for resource allocation**: The Cognito team sought to reduce the effect of the Kinesis faults in the early phases of the event by providing extra capacity and therefore boosting their ability to buffer calls to Kinesis. Instead of manually assigning capacity, there should be an automated system to increase or reallocate capacity in such events.

Lessons learned

- **Testing**: Proper testing and identifying the potential bugs are essential. In this case, the number of threads exceeding a maximum limit defined by the operating system seems to be a possible bug.
- **Ready operations team**: A bug bringing an overall system to a halt is the single point of failure, which is possible in a complex system. The production team should be trained and ready for such events.
- **Reducing number of servers**: To get significant headroom in the thread count used as the total threads, there's a need to move to more powerful CPU and memory servers. This will reduce the total number of servers and the threads required by each server to communicate across the fleet. Having fewer servers means that each server maintains fewer threads. Amazon is adding fine-grained alarming for thread consumption in the service.
- **Front-end fleet changes**: Several changes are required to radically improve the cold start time for the front-end fleet. Moreover, the front-end server cache needs to be moved to a dedicated fleet.
- **Avoiding recurrent failures**: To avoid recurrent failures in the future, extensive AWS services, like CloudWatch and others, must be moved to a separate, partitioned front-end fleet.



Lessons learned during the Kinesis outage

Point to Ponder

Question

What possible measures should Amazon have adopted to safeguard against the kind of failures they faced in Kinesis Data Streams?