

Cost-Effective Logging at Scale: ShareChat's Journey to WarpStream

Kafka Current, Bangalore '25

About Us



Vivek Chandela

- Current
  - Software Engineer working on Live Streaming Infra at ShareChat, Bangalore
- University
  - BTech in Computer Science, IIIT Hyderabad
- Technical Interests:
  - Databases, stream processing and distributed systems



Shubham Dhal

- Current
  - Staff software engineer at Sharechat working on recommender systems infrastructure
- University
  - Bachelor's in Engineering from Indian Institute of Technology, Kharagpur
- Technical Interests:
  - Streaming, streaming and more streaming 😀

Joint Work with....



Brijesh Savaliya



Prakhar Joshi



Arya Ketan



Srijan Saket

Table of Contents

01 Motivation

02 Issues with Logging

03 Why WarpStream?

04 Deployment

05 Best Practices

06 Cost Comparison

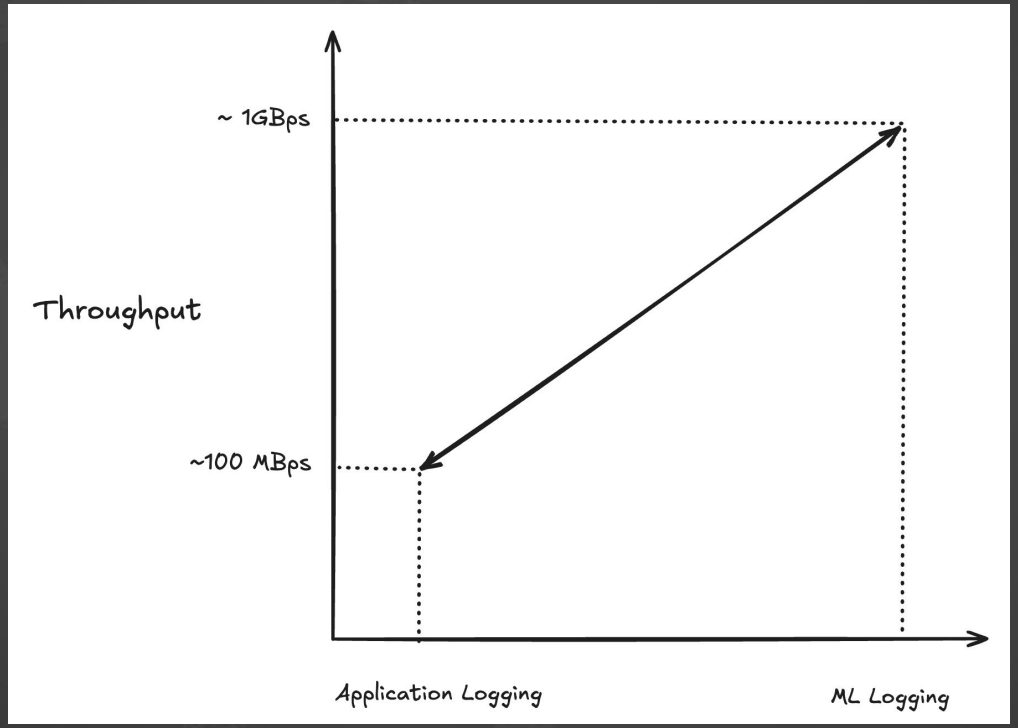
07 Spark Configs

08 Conclusion

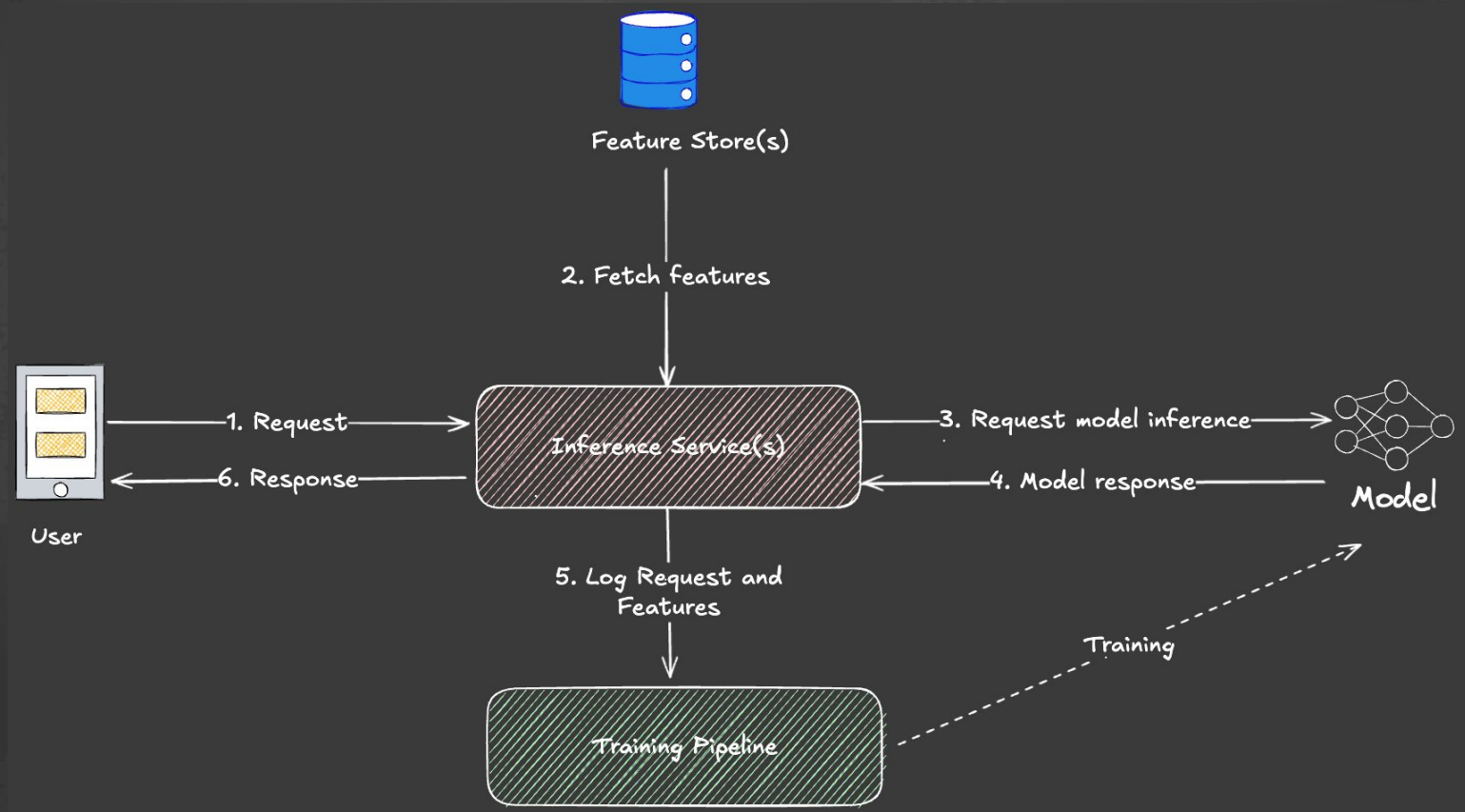


Motivation

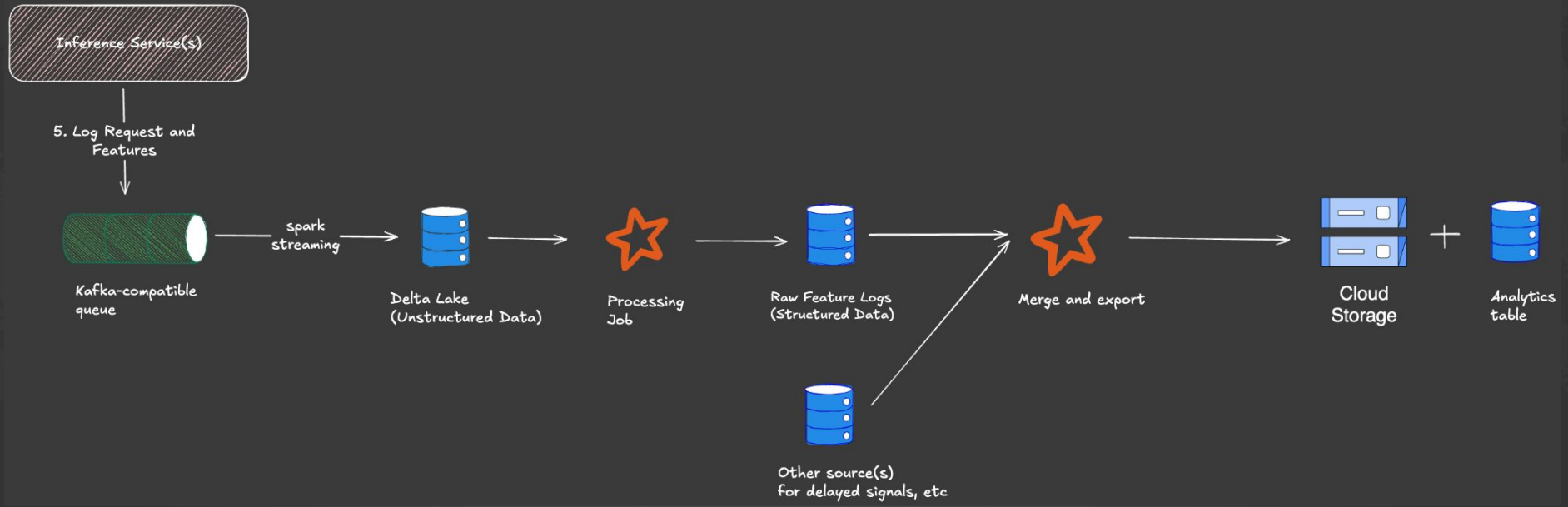
ML Logging vs Application Logging



Classical ML Inference + Training Pipeline



Training Pipeline





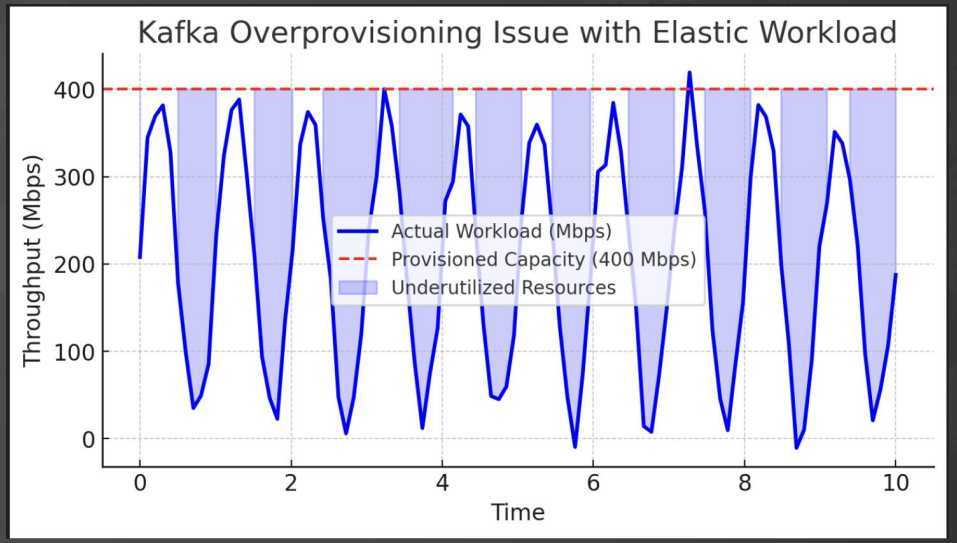
Issues With Logging

#1 Highly elastic workloads



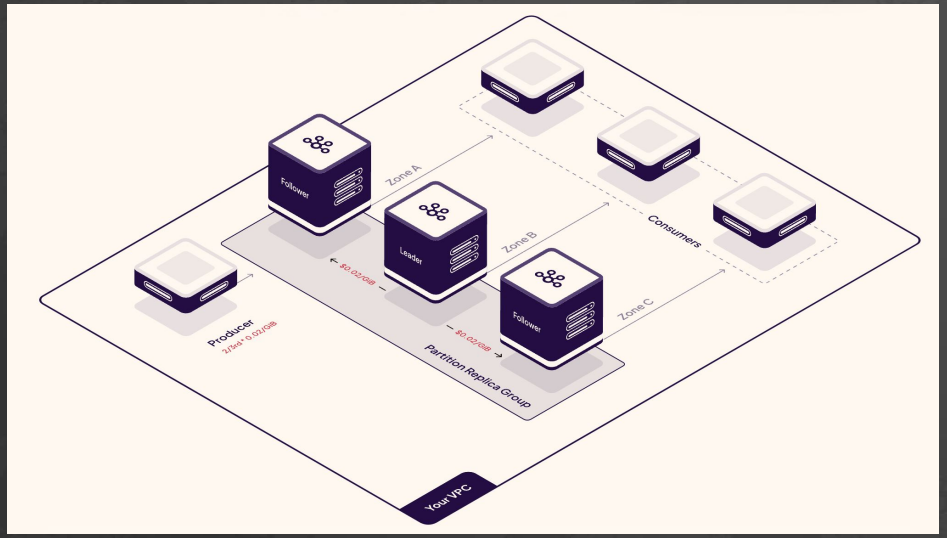
#1 Highly elastic workloads

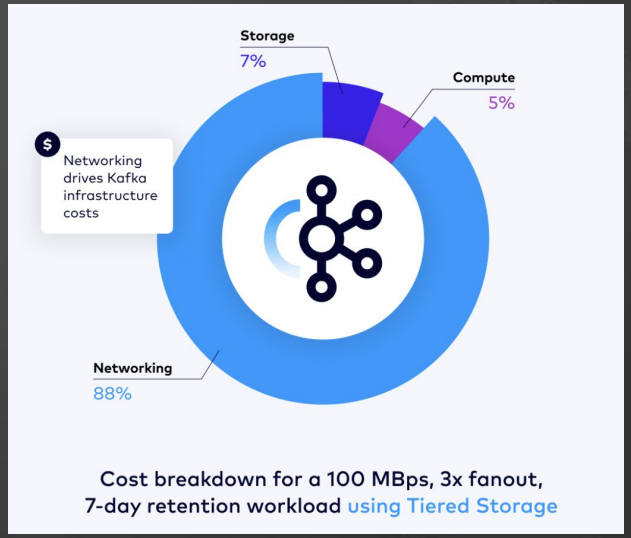
- Over Provisioning of Resources
- Auto-Scaling Limitations
- Partition Rebalancing Overhead
- Producer and Consumer Throttling
- Disk I/O Bottlenecks
- and some more...



#2 High inter-AZ networking fees

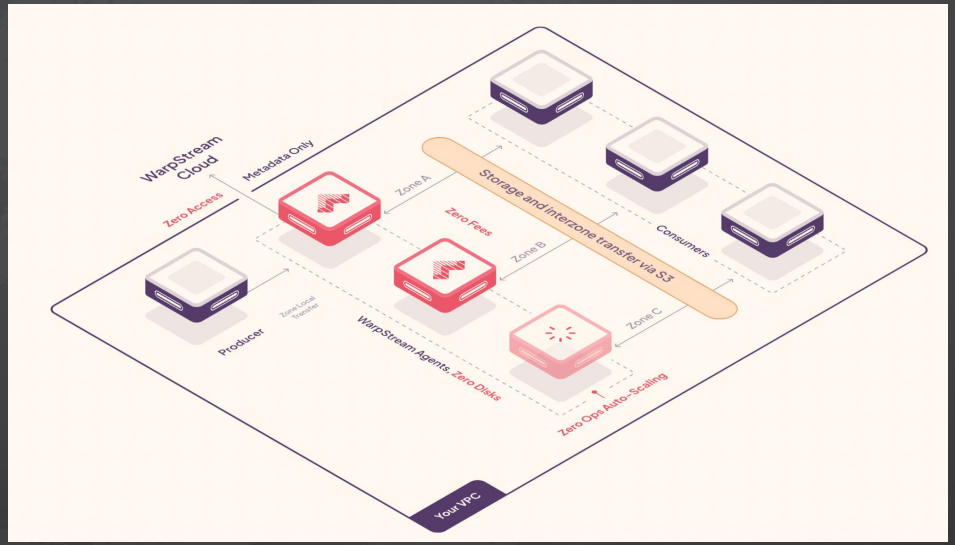
- Inter-AZ networking fees make ~70-90% of infra cost [Ref]
- For every 1 GiB data and replication factor = 3, we pay inter-AZ costs for 2/3 GB (sending data to leader partitions in other zones) + 2 GB (replication)
  - While networking costs $0.053/GiB, it costs just $0.023/GiB to store that data in S3 for a whole month !!





#2 High inter-AZ networking fees

- In WarpStream, networking fees are ~0 because:
  - Agents are zone-aware.
  - Object storage acts as the storage, network and replication layer.



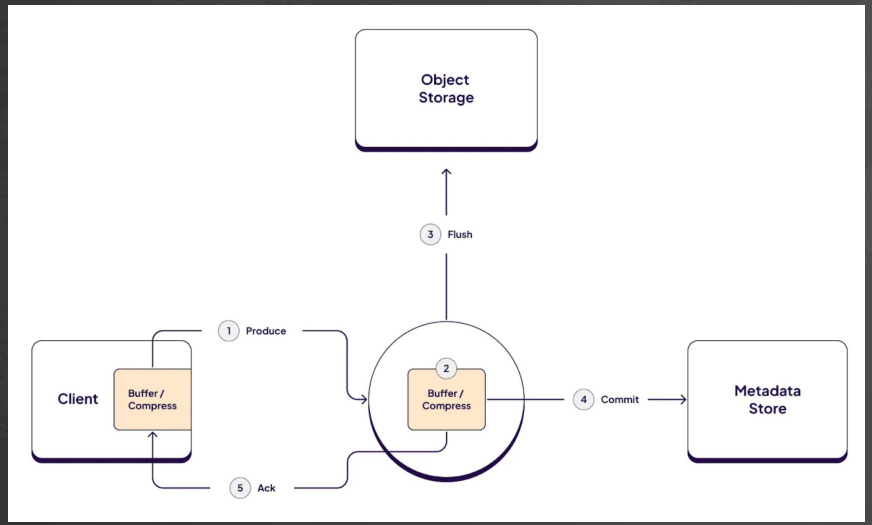


Why WarpStream?

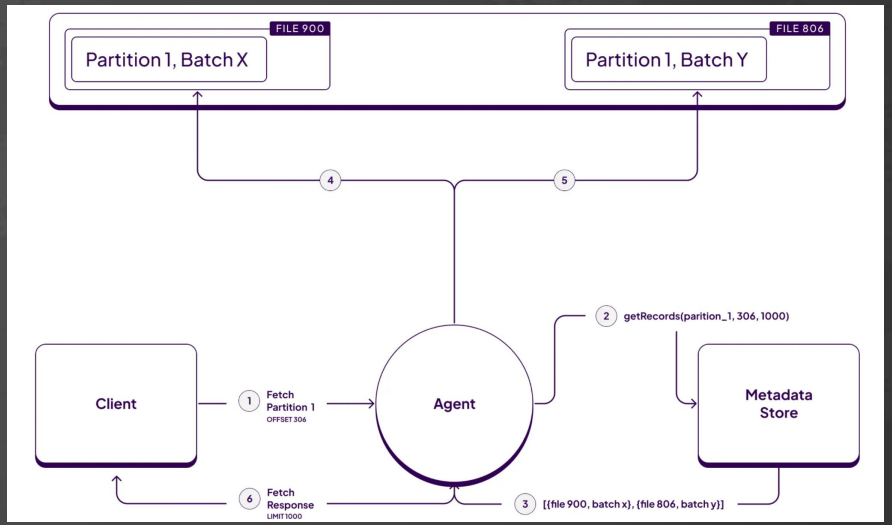
WarpStream Basics



WarpStream Basics



Write Path



Read Path

Why WarpStream?

Kafka		WarpStream
very high (70% of infra costs)	Inter-AZ networking costs	negligible
	Autoscaling	
Replicated Disks	Durability and Replication	Object Storage
	Operational Simplicity	
low	Latency	high
	Networking heavy instances	

Why WarpStream?

Kafka		WarpStream
very high (70% of infra costs)	Inter-AZ networking costs	negligible
	Autoscaling	
Replicated Disks	Durability and Replication	Object Storage
	Operational Simplicity	
low	Latency	high
	Networking heavy instances	

Why WarpStream?

Kafka		WarpStream
very high (70% of infra costs)	Inter-AZ networking costs	negligible
	Autoscaling	
Replicated Disks	Durability and Replication	Object Storage
	Operational Simplicity	
low	Latency	high
	Networking heavy instances	

Why WarpStream?

Kafka		WarpStream
very high (70% of infra costs)	Inter-AZ networking costs	negligible
	Autoscaling	
Replicated Disks	Durability and Replication	Object Storage
	Operational Simplicity	
low	Latency	high
	Networking heavy instances	

Why WarpStream?

Kafka		WarpStream
very high (70% of infra costs)	Inter-AZ networking costs	negligible
	Autoscaling	
Replicated Disks	Durability and Replication	Object Storage
	Operational Simplicity	
low	Latency	high
	Networking heavy instances	

Why WarpStream?

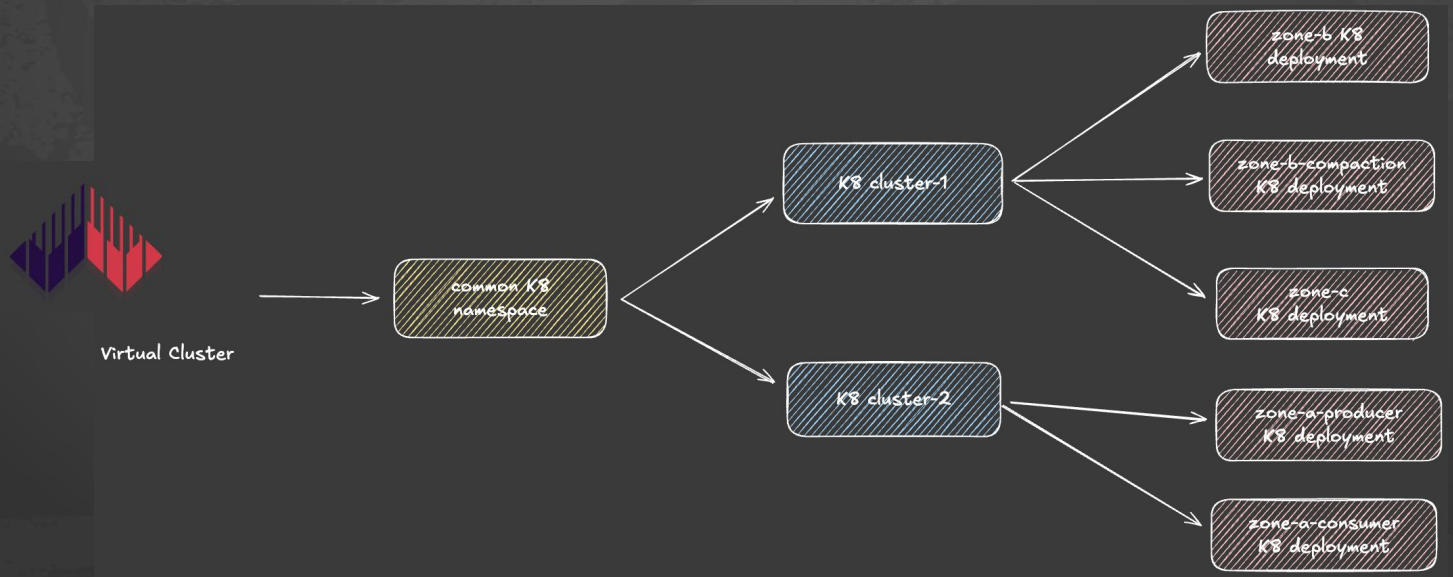
Kafka		WarpStream
very high (70% of infra costs)	Inter-AZ networking costs	negligible
	Autoscaling	
Replicated Disks	Durability and Replication	Object Storage
	Operational Simplicity	
low	Latency	high
	Networking heavy instances	



Deployment

Deployment

- Each BU (Ads, SC, Moj, Live, etc) → separate WarpStream Virtual Cluster (Metadata Store) for logical separation.
- All agents in a WarpStream Virtual Cluster cluster will share a common K8 namespace
- Separate deployments for agents in each zone of the K8 cluster so they can scale independently of the agents in other zones





Best Practices

Agent

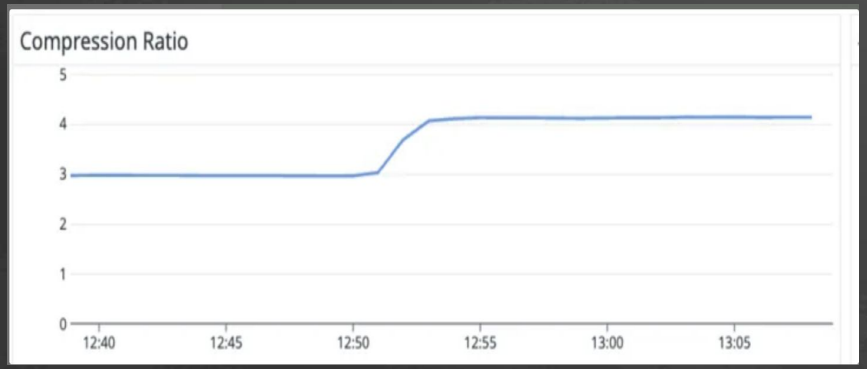
- There is no added efficiency for writing and reading data in the same zone with WarpStream, as the file cache is read-through and not write-through.
- Agent roles:
  - **proxy** → reads, writes, background jobs (e.g compaction)
  - **proxy-produce** → write-only
  - **proxy-consume** → read-only
  - **jobs** → background jobs only
- Run agents on spot instances instead of on-demand instances.
  - Try this with consumers first as even if they go down, we can clear the lag.

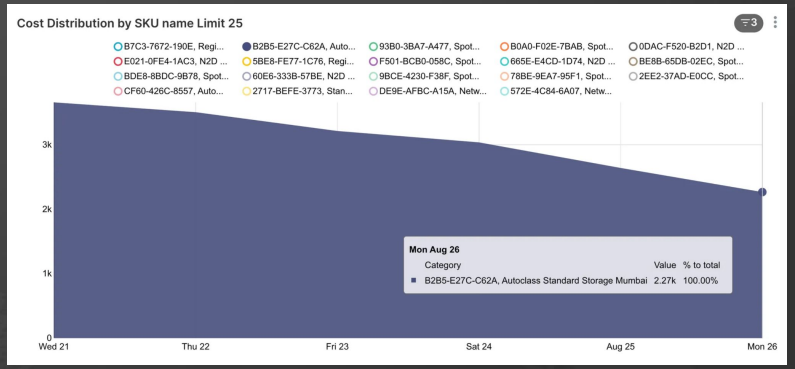
Agent

- A small no. of large agents is more efficient than a large no. of small agents.
  - Large no. of small agents:
    - 40 Agents (1 vCPU each) → 40 files every 250 ms → more PUT requests → more costs (Class A operations).
    - Each agent will run its own set of compactions to reduce the file count → more CPU and networking usage.
  - Small no. of large agents
    - 10 Agents (4 vCPU each) → 10 files every 250 ms for the same throughput.
    - Fewer consumer agents with more cores would lead to more efficient RPCs between the agents to power their in-memory cache.
    - The cache size defaults to 0.5GB/core, so the more the cores, the more the in-memory cache size.
    - Chances of being underutilized if size is too big

Agent

- **storageCompression** flag (**WARPSTREAM_STORAGE_COMPRESSION** env variable) controls the compression algorithm for data stored in object storage.
  - Switch from lz4 (default) → zstd for better compression at a slightly higher CPU
  - Network cards will be saturated slowly in zstd compared to lz4.
- **WARPSTREAM_ZSTD_COMPRESSION_LEVEL** env variable helps to tune zstd with compression levels ranging from negative 7 (fastest) to 22 (slowest in compression speed, but best compression ratio).
- Impact: Increased compression ratio and ~35% cost reduction !!





Producer Agents

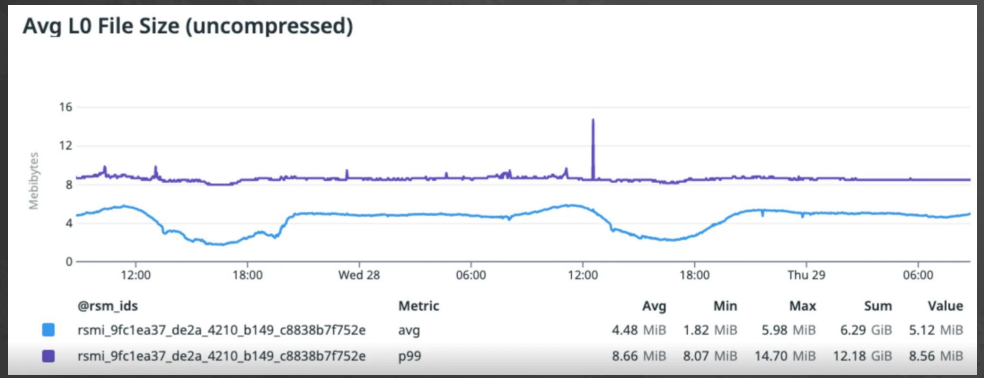
- Larger batches is more efficient than smaller batches
  - batch size doubles -> PUT requests cut in half -> reduced Class A operations → reduced cost
  - smaller batches have the following issues:
    - It increases the amount of load on the metadata store/control plane because more batches have to be tracked and managed.
    - It increases CPU usage because more batches usually mean worse compression, which involves copying more bytes around the network.
    - It increased end-to-end latency, because the Agents have to read more batches and perform more IO to transmit data to consumers.

Producer Agents

- There are 2 ways to increase batch size:
  - Cut the no. of producer agents in half by doubling the number of cores available to them
    - Alternatively, we can double the value of `batchTimeout` from 250ms (default) -> 500ms
    - Having fatter agents by doubling the cores is slightly better as we can avoid the latency penalty (for waiting on `batchTimeout`), while increasing the L0 file size
  - Increase `batchMaxSizeBytes` to 16 MB (current maximum)
    - It is a tradeoff between cost and latency.
    - Only do this for agents with roles = `proxy_produce` / `proxy` as agents with role = `jobs` already have batch size = 64 MB
- Recommended Configs:
  - CPU: 8-16 vCPU, Ram: 32-64 GiB (4 GiB/vCPU)
  - batchMaxSizeBytes: 16 MB

Producer Agents

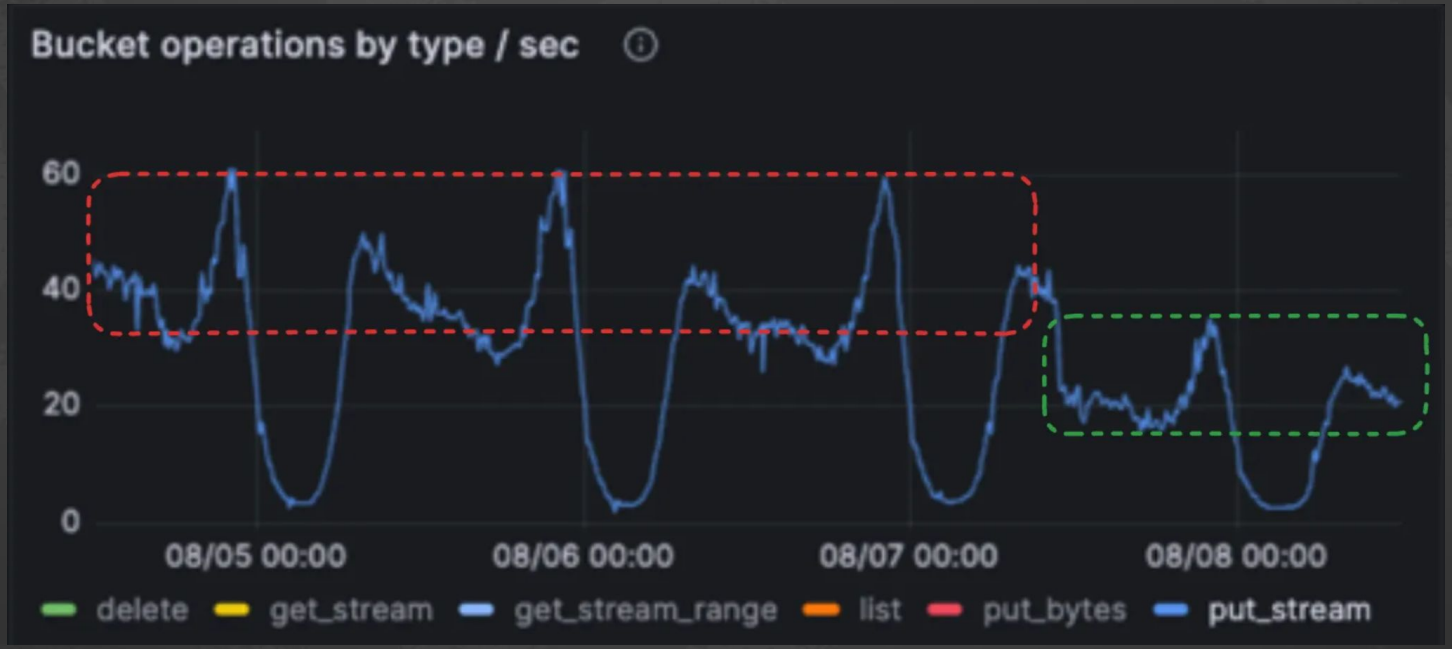
- To know if the batch size is good enough, check the **p99 uncompressed size of L0 files**.
  - If ~**batchMaxSizeBytes**, double **batchMaxSizeBytes** to halve PUT calls -> reduced Class A operations → reduced cost.
  - If < **batchMaxSizeBytes**, make the agents fatter or increase batch timeout to increase the size of L0 files. Now, double **batchMaxSizeBytes** to halve PUT calls.
- In our case, we noticed that the p99 uncompressed size of L0 files is ~8MB, so we doubled **batchMaxSizeBytes** to 16 MB





Doubling batchMaxSizeBytes to 16 MB

- PUT requests cut by nearly half.



Doubling batchMaxSizeBytes to 16 MB

- Latency for put bytes increased from ~99 ms to ~240 ms



Doubling batchMaxSizeBytes to 16 MB

- Produce latency also increased from ~270 ms to ~340 ms



Jobs Agents

- They need to be throughput optimized so we can run them hotter than other agents.
  - We set CPU-based HPA at 70% instead of 50% for jobs agents.
- They're very networking hungry so if you run them very hot on non-network optimized instances, you may end up saturating the network before they reach a high enough CPU to auto-scale.
- Recommended configs:
  - CPU: 4-8 vCPU
  - RAM: 16-32 GiB (4 GiB/vCPU)

Fetch Agents

- For spark-based use cases, we don't read data continuously. We wake up every 1m and try to read tons of data at once.
- So we used 8GiB/vCPU because our fetch workload is intense.
- Recommended configs (spark-based use cases):
  - CPU: 4-8 vCPU
  - RAM: 32-64 GiB (8 GiB/vCPU)

Client

- To eliminate inter-AZ costs, append `warpstream_az=<your-az>` to WarpStream client's `ClientID` for both producer and consumer.
- For agent roles, append:
  - `warpstream_proxy_target=proxy-produce` to `ClientID` in producer client
  - `warpstream_proxy_target=proxy-consume` to `ClientID` in consumer client
- Set `RECORD_RETRIES` = 3 and use compression (e.g GZIP)

Client

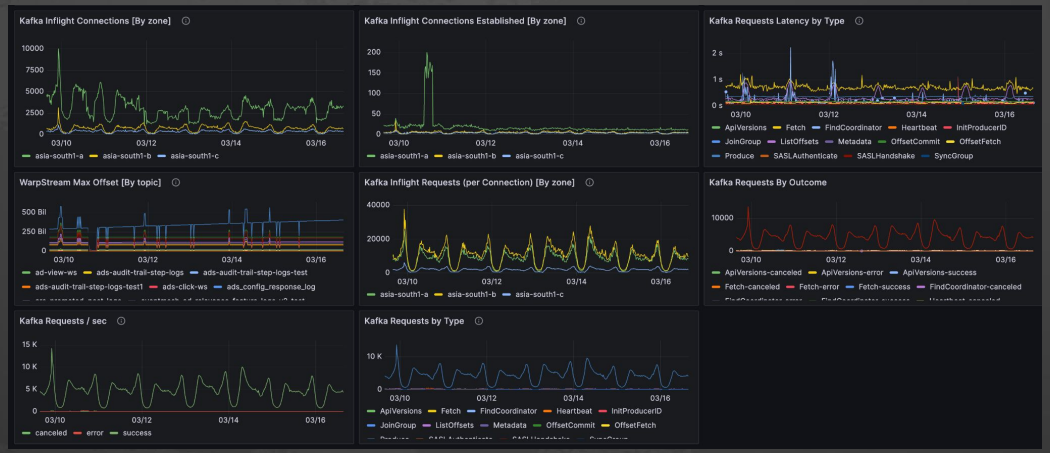
- metaDataMaxAge sets the maximum age for the client's cached metadata (default = 5m)
  - It is set to 60 sec currently.
  - Removing this will lead to traffic imbalances in the agents within the same zone.
- To be doubly sure that metadata is refreshed properly:
  - In the client, set metaDataMaxAge to 10 sec
  - In the agents, set kafkaLoadBalancingInterval flag (WARPSTREAM_KAFKA_LOAD_BALANCING_INTERVAL env variable) to 5m.
- Use Sticky partitioner instead of Round-Robin partitioner

Tweaking Latency

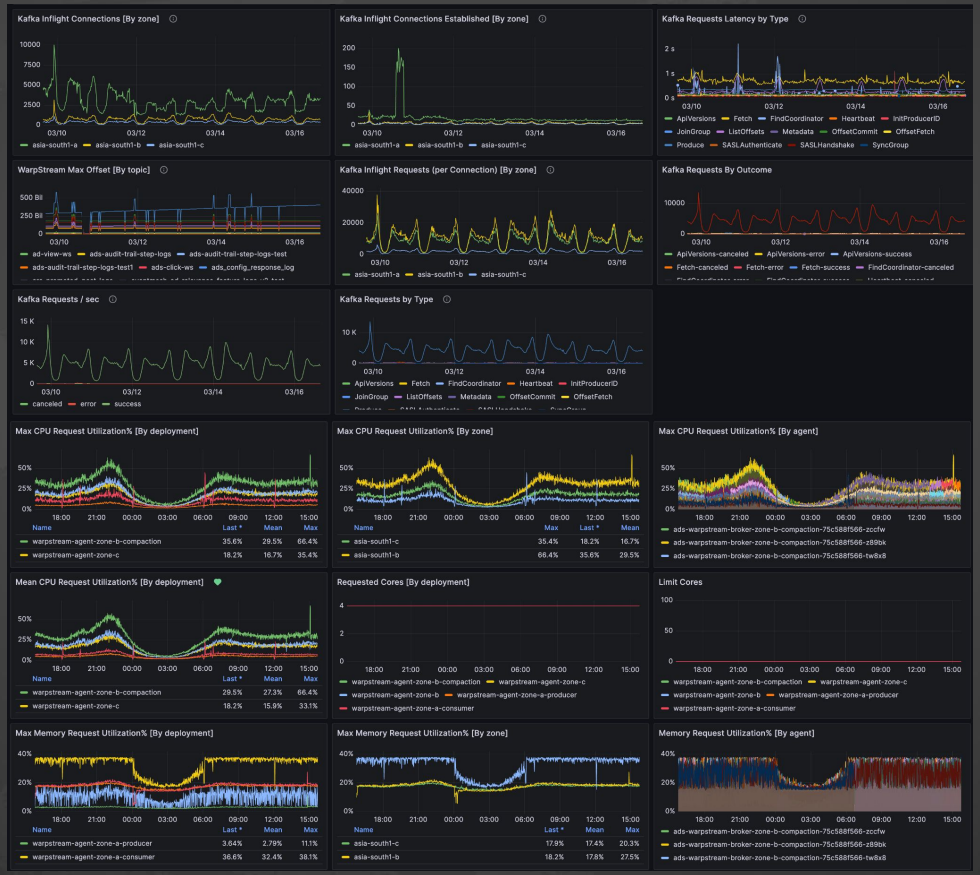
- WARPSTREAM_BATCH_TIMEOUT can be decreased from 250 ms (default) to upto 50 ms !!
  - It controls how long the Agents will buffer data in-memory before flushing it to object storage.
  - It is a tradeoff b/w latency and cloud infra costs.
  - Lowering this value leads to lower latency but more infra costs as Agents will have to create more files in object storage and incur higher PUT call API fees.
  - However, this doesn't affect durability as produce requests are never acknowledged to the client before data is durably persisted in object storage.
- Decrease Control plane latency.
- There's nothing similar to S3 Express One in Google Cloud, so this option was out of bounds for us.

Alerts

- CPU > 50% and Memory > 50%
- Add alerts on the topic's lag as per SLA, L0 file size and inter-AZ networking traffic
- Alerts at various granularity -- including agent, topic, and zone
- Z-Score based alerts on compressed throughput



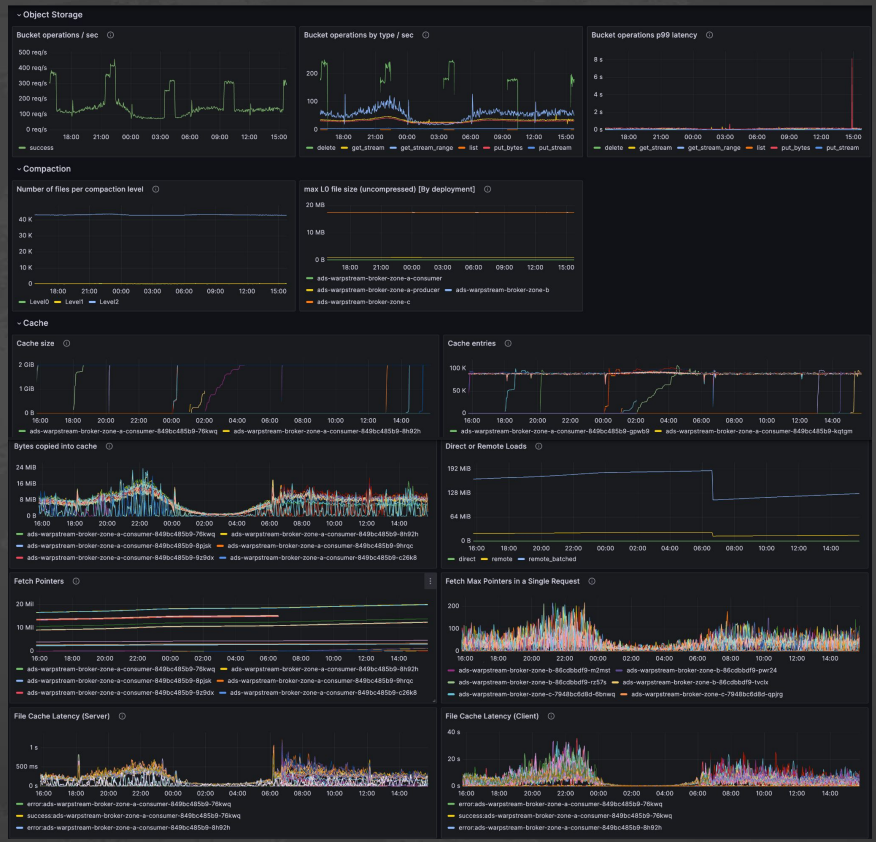
Alerts


Kafka Inflight Connections [By zone]
Kafka Inflight Connections Established [By zone]
Kafka Requests Latency by Type
WarpStream Max Offset [By topic]
Kafka Inflight Requests (per Connection) [By zone]
Kafka Requests By Outcome
Kafka Requests / sec
Kafka Requests by Type
Max CPU Request Utilization% [By deployment]
Max CPU Request Utilization% [By zone]
Max CPU Request Utilization% [By agent]
Mean CPU Request Utilization% [By deployment]
Requested Cores [By deployment]
Limit Cores
Max Memory Request Utilization% [By deployment]
Max Memory Request Utilization% [By zone]
Memory Request Utilization% [By agent]

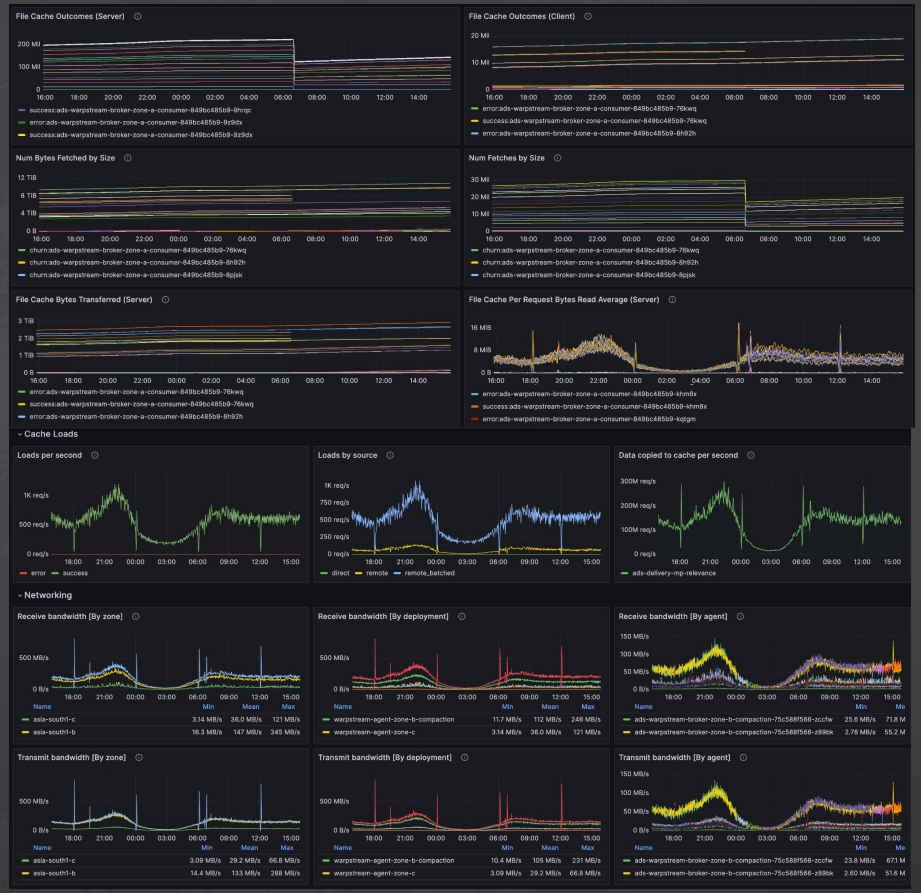
Alerts


Requested Memory [By deployment]
16 GiB
8 GiB
0 B
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
warpstream-agent-zone-b-compaction
warpstream-agent-zone-c
warpstream-agent-zone-b
warpstream-agent-zone-a-producer
warpstream-agent-zone-a-consumer
Limit Memory [By deployment]
16 GiB
8 GiB
0 B
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
warpstream-agent-zone-b-compaction
warpstream-agent-zone-c
warpstream-agent-zone-b
warpstream-agent-zone-a-producer
warpstream-agent-zone-a-consumer
Pod Restarts
100
50
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
warpstream-agent-zone-b-compaction
warpstream-agent-zone-c
warpstream-agent-zone-b
warpstream-agent-zone-a-producer
warpstream-agent-zone-a-consumer
Agent (pod) Availability
15
10
5
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
ads-warpstream-broker-zone-a-consumer
ads-warpstream-broker-zone-a-producer
ads-warpstream-broker-zone-b
ads-warpstream-broker-zone-b-compaction
ads-warpstream-broker-zone-c
#unavailable agents (pods)
6
4
2
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
ads-warpstream-broker-zone-a-consumer
ads-warpstream-broker-zone-a-producer
ads-warpstream-broker-zone-b
ads-warpstream-broker-zone-b-compaction
ads-warpstream-broker-zone-c
Pod Provisioning Type
20
15
10
5
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
Value
on-demand
spot
~ WarpStream Topics
Total Topics per K8 namespace
15
10
5
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
Name
Last *
ads-warpstream-brokers
13
Total Topics Limit per K8 namespace
4 K
2 K
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
Name
Last *
ads-warpstream-brokers
4.10 K
Total Partitions per K8 namespace
400
200
0
03/15 18:00
03/16 00:00
03/16 06:00
03/16 12:00
Name
Last *
ads-warpstream-brokers
516
Total Partitions Limit per k8s namespace
7.50 K
5 K
2.50 K
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
Name
Last *
ads-warpstream-brokers
8.19 K
~ Warpstream agent-jobs
Processing of background jobs
1.5
1
0.5
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
compaction_job_type-success
deadscanner_job_type-success
publish_virtual_cluster_metrics_job_type-success
Background jobs by type
1.5
1
0.5
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
compaction_job_type
deadscanner_job_type
publish_virtual_cluster_metrics_job_type
Latency of background jobs by type
30 s
20 s
10 s
0 s
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
compaction_job_type
deadscanner_job_type
publish_virtual_cluster_metrics_job_type
Dead Files Scanner: Checked vs Deleted Files
400
300
200
100
0
18:00 21:00 00:00 03:00 06:00 09:00 12:00 15:00
checked
deleted
error
invalid
scanned

Alerts


Object Storage
Bucket operations / sec
Bucket operations by type / sec
Bucket operations p99 latency
Compaction
Number of files per compaction level
max L0 file size (uncompressed) [By deployment]
Cache
Cache size
Cache entries
Bytes copied into cache
Direct or Remote Loads
Fetch Pointers
Fetch Max Pointers in a Single Request
File Cache Latency (Server)
File Cache Latency (Client)

Alerts


File Cache Outcomes (Server)
File Cache Outcomes (Client)
Num Bytes Fetched by Size
Num Fetches by Size
File Cache Bytes Transferred (Server)
File Cache Per Request Bytes Read Average (Server)
Cache Loads
Loads per second
Loads by source
Data copied to cache per second
Networking
Receive bandwidth [By zone]
Receive bandwidth [By deployment]
Receive bandwidth [By agent]
Transmit bandwidth [By zone]
Transmit bandwidth [By deployment]
Transmit bandwidth [By agent]



Cost Comparision

WarpStream

- Compressed Throughput → Max = 394 Mbps, Mean = 178 Mbps
- Cost → 13.4K INR per day
- Cost per GBps (using max throughput) = ((13.4k / 394) * 1024) = ~35k INR per day per GBps
- Cost per GBps (using mean throughput) = ((13.4k / 178) * 1024) = ~77k INR per day per GBps





Multi-Zone Kafka

- We've combined Kafka's throughput with WarpStream's throughput to get total throughput of Kafka before WarpStream was introduced.
    - Compressed Throughput → Max = 1504 Mbps (1.11 GBps + 394 MBps), Mean = 730 Mbps (552 Mbps + 178 Mbps)
- Cost → 130k INR per day
- Cost per GBps (using max throughput) = ((130k / 1504) * 1024) = ~88.5k INR per day per GBps
- Cost per GBps (using mean throughput) = ((130k / 730) * 1024) = ~182k INR per day per





Single-Zone Kafka

- If we remove NIZE cost, we can roughly estimate the cost for single-zone RP.
- NIZE Cost → 422k /7 = 60k per day
- Cost per GiBps (single-zone RP, using max throughput) = ((70k / 1504) * 1024) = ~48k INR per day per GBps
- Cost per GBps (single-zone RP, using mean throughput) = ((70k / 730) * 1024) = ~98k INR per day per GBps



Total Savings

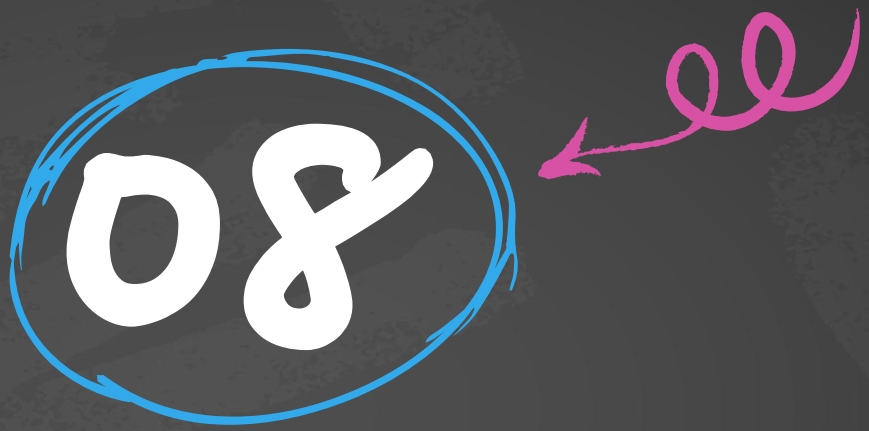
- WarpStream vs Multi-Zone Kafka
  - Based on Max Throughput = 1-(35/88.5) ~ 60%
  - Based on Mean Throughput = 1-(77/182) ~ 58%
- WarpStream vs Single-Zone Kafka
  - Based on Max Throughput = 1-(35/48) ~ 27%
  - Based on Mean Throughput = 1-(77/98) ~ 21.4%



Spark Configs

Spark Configs

- Tune the topic partitions to maximize parallelism:
  - We made sure that each partition processes not more than 1MB / sec.
  - Keep numbers of partitions as multiple of `spark.executor.cores` (we kept it `spark.executor.cores` * `spark.executor.instances`).
- Tune kafka client configs to avoid too many fetch requests while consuming.
  - Increase `kafka.max.poll.records` for topic with too many records but small payload size.
  - Increase `kafka.fetch.max.bytes` for topic with high volume data.
- With these tunings, we were able to reduce single spark micro batch processing time:
  - 22 minutes → ~50 seconds (for Processing throughput > 220 MB / second)
  - 6 minutes → ~30 seconds (for Processing rate > 200K records/ second)

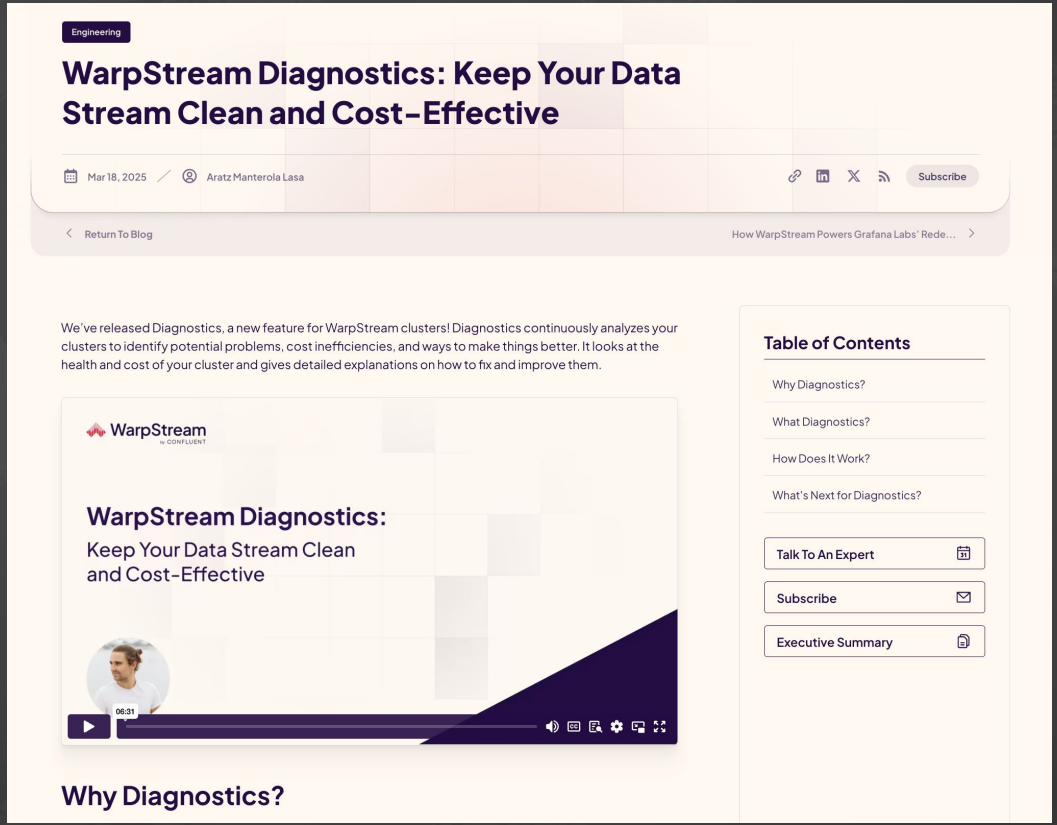


Conclusion

Conclusion

- WarpStream shines in a specific niche—logging, observability, and data lake feeding—where a slight increase in latency is a fair trade-off for substantial cloud cost savings and simplified operations.
- No need to worry about:
    - Brokers, Coordinators, Controllers, ZooKeeper (or KRaft), Leader elections, Partition Rebalancing, Failure recovery, Hotspots, Broker tuning, etc
- WarpStream offloads all the hard problems of data replication, durability, availability and networking to object storage.
- WarpStream Agent is a stateless, autoscaling and zone-aware Go binary with no local disks.
    - Any WarpStream Agent can act as the “leader” for any topic, commit offsets for any consumer group, or act as the coordinator for the cluster.
- Cost savings of ~58% vs Multi-Zone Kafka and ~21% vs Single-Zone Kafka

BTW there's an announcement....



Engineering

WarpStream Diagnostics: Keep Your Data Stream Clean and Cost-Effective

Mar 18, 2025 / Aratz Manterola Lasa

Subscribe

Return To Blog

How WarpStream Powers Grafana Labs' Rede...

We've released Diagnostics, a new feature for WarpStream clusters! Diagnostics continuously analyzes your clusters to identify potential problems, cost inefficiencies, and ways to make things better. It looks at the health and cost of your cluster and gives detailed explanations on how to fix and improve them.

Why Diagnostics?

Table of Contents

- Why Diagnostics?
- What Diagnostics?
- How Does It Work?
- What's Next for Diagnostics?

Talk To An Expert

Subscribe

Executive Summary

Thanks!

Questions?

