The game is afoot:

Using Cloud Insights to help uncover issues in your stream-processing Kubernetes application

NetApp

Contents

The game is afoot:

Using Cloud Insights to help uncover issues in your stream-processing Kubernetes application

Application development has gone through incredible changes in the last couple of years. Today's microservice, event-driven architecture adoption, while opening up new horizons for application scale, introduced a level of complexity that is still not easy to navigate. With this complexity comes the need for monitoring, troubleshooting, and planning solutions to help us understand how all the components of our application are behaving and to help us keep them running healthily.

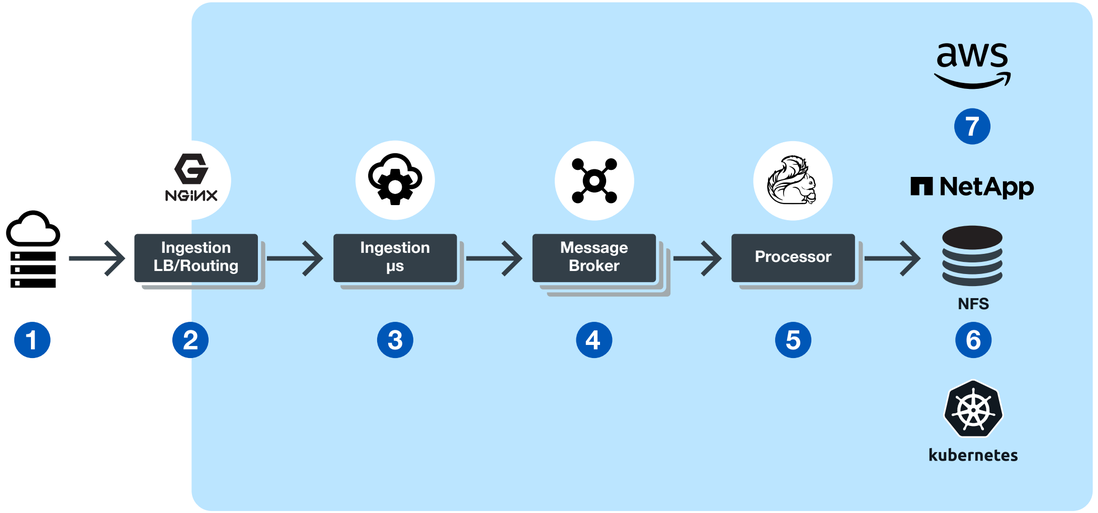
At NetApp we eat our own dogfood, using NetApp® Cloud Insights to help us develop the next versions of Cloud Insights, and also to develop other new products and ensure that they are running as they were designed to.

In this series we will follow a specific example of using Cloud Insights to help us troubleshoot issues in our system. This is a "whodunnit" story with a crime (poor performance), several suspects (architectural components), and a "the butler did it" ending (the faulty configuration causing the problem). It's a story of load balancers (NGINX), microservices (K8s pods), message brokers (Kafka), stream processing (Flink), underlying storage (NetApp Cloud Volumes ONTAP® FlexVol® technology), and even AWS EC2 virtual machines. A story in which the culprit comes from a side you typically don't correlate to your environment (storage and apps outside your K8s environment that impact the workings of that environment). A story in which Cloud Insights is our tool for following the clues. By the end of this series, we'll have collected our full environment and examined all of its components.

Follow me, the game is afoot!

The characters

Let’s introduce the main characters of this story, the components of our architecture. We are developing a stream-processing application. Because this document is based on a true story, names have been changed to protect privacy and size is trimmed a bit to fit your screen. This app receives requests from the outside world and data flows throughout a series of processes, ending with data stored in the file system.



Our application is a typical stream-processing application running in Kubernetes (K8s) in an AWS environment. The main components are:

1. The agent.
An external tool that captures data points and sends them to our system.

2. The router.
A router/load balancer for incoming data. We are using a NGINX ingress controller for this purpose. NGINX receives requests, routes them based on URL patterns, and load balances them across pods of an ingestion service.

3. The ingestion.
A microservice that receives the data from the agent, performs very lightweight validation, and writes accepted data points into a specified Kafka topic.

4. The message broker.
A Kafka cluster running across several brokers. Owns a Kafka topic that stores requests waiting to be processed.

5. The processor.
A stream-processing engine. We are using a Flink job to process incoming data from Kafka. This job is running on a Flink cluster with several task managers. The job performs transformation on data and eventually writes results to disk.

6. The storage.
The disk that is used by the Flink job, an NFS mount on Flink task managers. We are using Cloud Volumes ONTAP for our underlying storage.

7. The compute.
We run our application in Kubernetes (K8s). Our Kubernetes nodes are deployed as AWS EC2 instances.

The crime

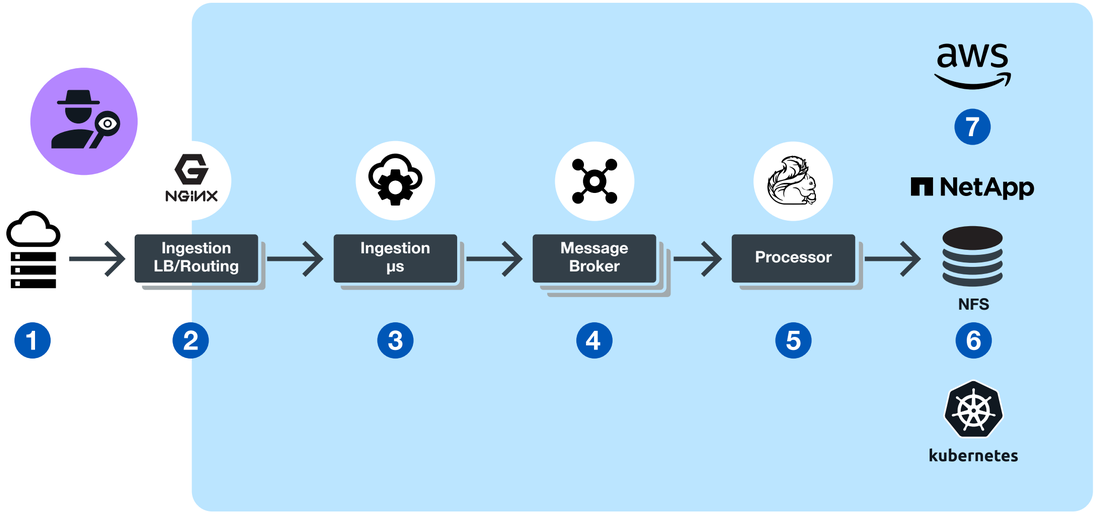
The day started like so many others:
Data was flowing and life was good. But in the middle of the day we begin to observe some weird behavior. Our most recent data was not being displayed. It would show up eventually, but with a noticeable delay. Something was preventing those results from showing up in our system on time. But with so many components, we didn't know what was causing the misbehavior. This went on for a couple of days, and we observed that at about 40 minutes after every hour data was taking a bit longer to be processed and made available. The system would eventually pick up and get back on track, but every hour at that time something was off.

We decided to look closely at each component of our architecture. Let's bring them for questioning.

The router: The investigation begins

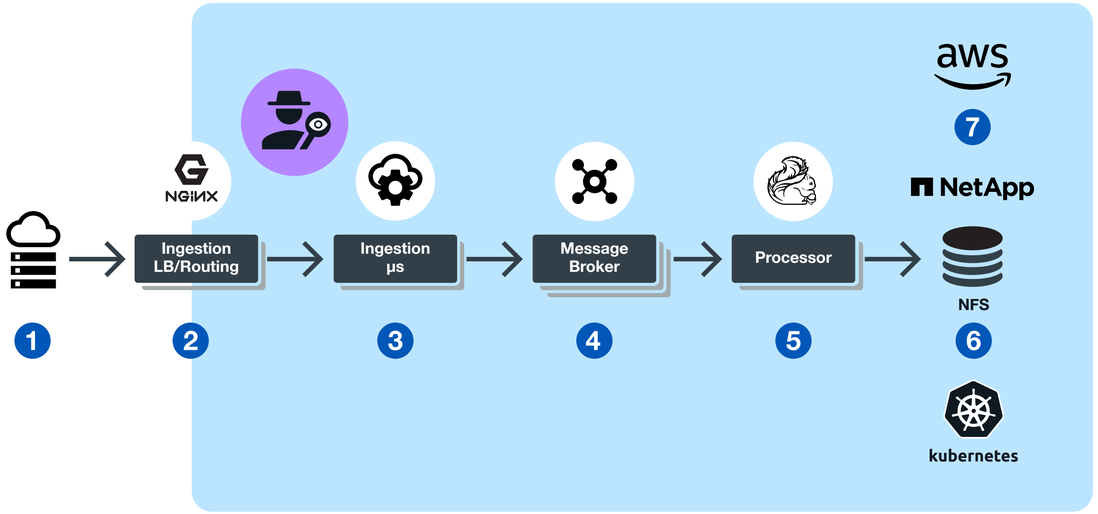
We need an overall picture of our environment, so let's start at the beginning. The first entry point to our application is our ingress controller (router/load balancer), so we'll start by collecting information from our NGINX controller.

Cloud Insights allows us to capture infrastructure and application data. To figure out what's going on with our NGINX component, we first look at captured NGINX data. Our working assumption was that maybe something was preventing data from getting into our NGINX controller.



Hmmm… Our NGINX numbers clearly show that is the wrong assumption. There is no degradation or significant change of behavior in our NGINX controller. Data is coming in steadily at the same rates throughout the day. You're free to go, NGINX, we have no more questions for you. But don't leave town just yet.

The ingestion: A new suspect



With NGINX out of the picture, we advance to the next suspect— our ingestion microservice. The ingestion service is deployed as a Kubernetes deployment, following all the best practices for resilience, fault tolerance, and scalability. But could this microservice be misbehaving at some point and causing the problem? We need to look deeper into this one. Because it's a microservice deployed as a pod, we decide to start watching Kubernetes to see how much traffic these pods are receiving and sending out. Maybe data is being dropped there?

In the Kubernetes cluster explorer in Cloud Insights, we click on our cluster to see the vital statistics.

We can see our cluster state, including the top users of the cluster and overall usage of the cluster. We can even dive into specific nodes to see what's running there.

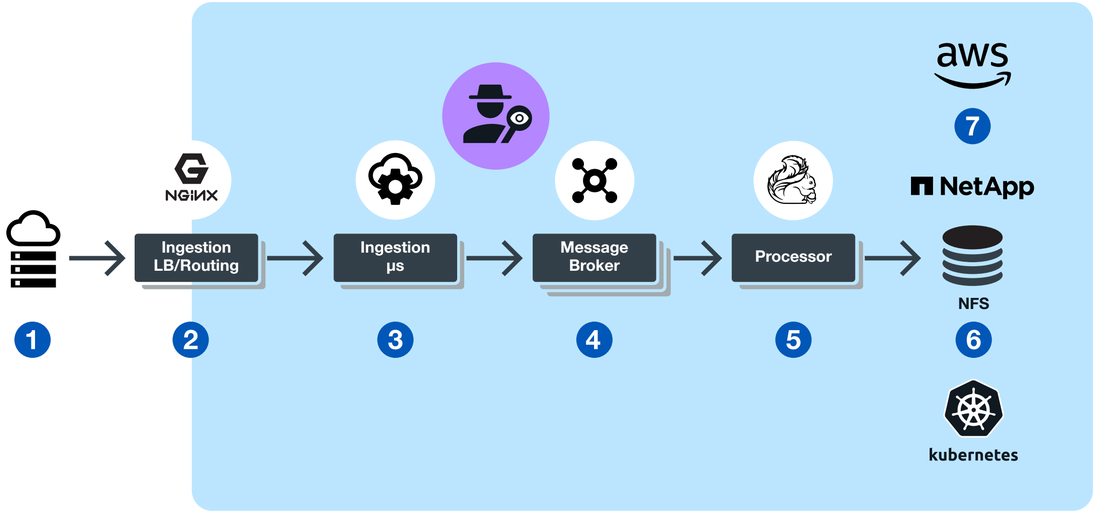
Now we can see all pods and containers running on our pod and get an overall picture of usage of our node. We can also dive deeper into one pod and see the specific information for that pod along with some simple historical CPU and memory usage.

In this case, however, we want to look at the overall ingestion behavior of our pods. To do this we can view our Kubernetes performance metrics alongside the data from the rest of our application, all in one convenient view.

We also want to keep tabs on CPU and memory utilization for the containers for those pods. In all of these views, we have grouped our metrics by `kubernetes_cluster, namespace, pod_name`, and `container_name` to get one line per container, so we can easily spot anything that looks suspicious.

It looks like our ingestion pods are not to blame after all. We see that the amount of data flowing in and out is consistent throughout the day, and we don't see any significant degradation. We also don't see any significant memory or CPU impact during the times that we have issues.

The message broker: The man in the middle



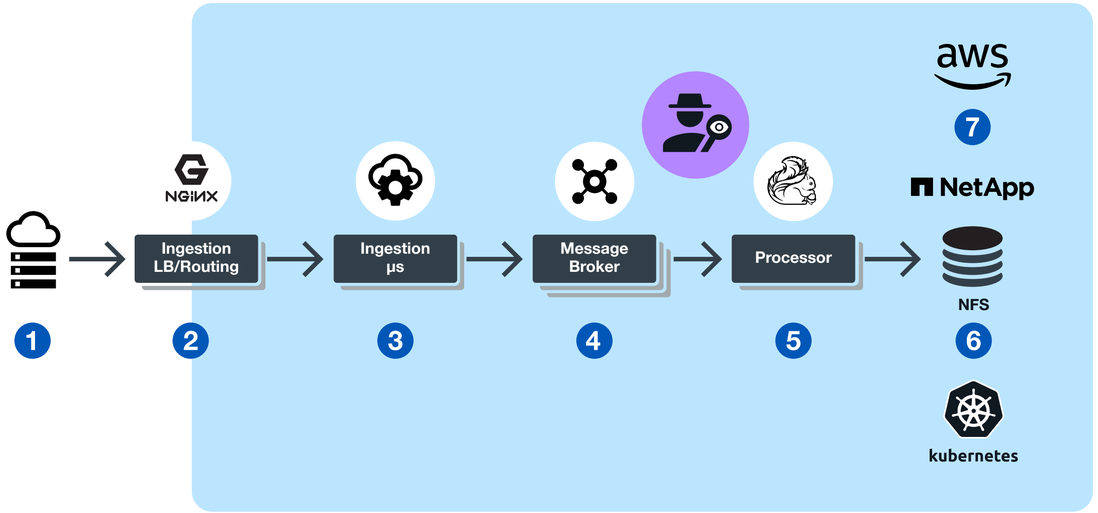
Next in line for questioning is our Kafka broker. Our microservice does some lightweight processing of incoming data and then writes it to a Kafka topic. Maybe something is misbehaving in Kafka at the times we observe slow processing?

We can view critical Kafka performance metrics in our application dashboard.

Now that's interesting… There is not a significant change in the amount of data coming into Kafka, but we do see some weird behavior around the time that data stops coming in. We definitely see a difference in the amount of data coming out of Kafka. At least we see that data coming into Kafka is not the issue. That's another data point. Now why are we seeing those changes in the amount of data read from Kafka around those times?

In any case, we think that Kafka can be discarded from the list of suspects.

The processor: We’re getting closer

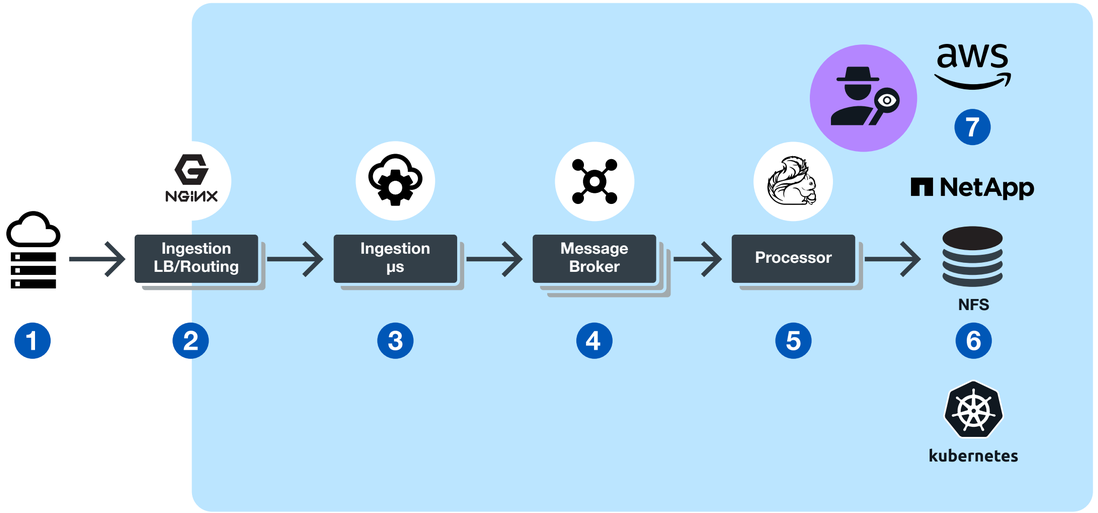


Next we move through the track to our Flink job. Our job is pretty well instrumented with custom metrics for disk writes, and we can take advantage of that in Cloud Insights.

We’ve added to our dashboard the number of ingested records per second for the Flink operator that writes to disk, as well as our custom metrics and write lag. Now things are getting interesting.

We do see some drop in writing to disk at around the time of the incident. That is consistent with the drop we see of data being read from Kafka. But we don’t see any significant difference in data being ingested by Kafka, so it looks like for some reason our Flink operator is getting hit when trying to write to disk. What if the problem is actually in writing to disk?

The storage: The plot thickens



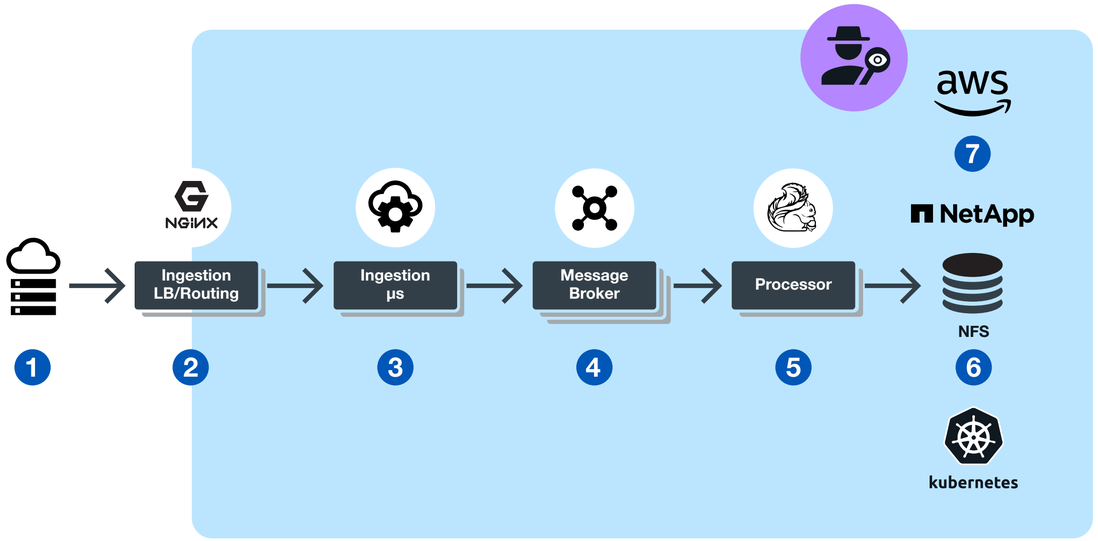
Our Flink job writes to storage, and that storage is accessible to the Flink TaskManager pods via the usual Persistent Volume Claim + Persistent Volume setup. It took quite a few hops to get to this information, but it turns out that when we were looking at our microservice pods and activated K8s monitoring in Cloud Insights, we started collecting part of the information we need. The part we're still missing is the actual storage behind the persistent volumes information. But this has traditionally been Cloud Insights bread and butter. Let's check out the information about our storage, NetApp Cloud Volumes ONTAP, that our Flink processing pod is using.

In the cluster explorer, we see that our persistent volume is linked to the underlying storage and some key metrics associated with it. Let's click through.

The storage: The plot thickens

Do we see a probable cause? There is an I/O bump happening at the time of the incident, and a latency bump in accessing our storage. We're getting closer. But what is causing it? Who is actually creating this I/O? Our pod does not seem to be a good candidate, because we have already observed that we have consistent amounts of data coming
in throughout the day. It's got to be something else. To round out the picture, we can also check out AWS EC2 information from the same consolidated application view.

The compute: What are our AWS EC2 instances doing?



Taking a closer look into the linked FlexVol volume used by our Flink job, we find a candidate for the unexpected increase in I/O at the time of the event. Looks like virtual machine `frosa-system-backup` is generating some relevant I/O in the volume. Let's click along and look closer at that virtual machine.

The probable cause has turned into a solid accusation. This VM is doing some heavy work at around the time of the degradation.

Let’s dive onto this EC2 instance, log onto it, and dig around. Upon analysis of the virtual machine we find a process running every hour at around the 40-minute marker—a pesky cronjob.

```
[root@frosa-system-backup ~]$ crontab -l
40 * * * * /opt/backups/system_backup.sh
[root@frosa-system-backup ~]$
```

This process is doing data dumps and generating enough I/O to impact the performance of our internal volume shared with our processing pods. These pods are doing business-critical operations. We need to fix this EC2 instance to avoid writing to the same FlexVol volume behind our Flink job Kubernetes persistent volume. We ask our system administrator to have this EC2 instance attach to a different FlexVol volume.

Our application is now still flying through data. Looks like we solved it!

The aftermath: Elementary, my dear system

Typical crime stories have the unexpected culprit, the distant relative that our characters had lost track of. This story is no different. We have a culprit that is not easily remembered in Kubernetes monitoring tools. Storage. It's true – it's usually not easy to see the connection between your Kubernetes pods and underlying storage. As with all good investigations, finding the right clues and relations between suspects is crucial, and Cloud Insights makes those clues and relations visible to us. By the end of our investigation we created visibility into our whole infrastructure. While we used this visibility to help us find a problem, we now have a strong foundation to build on to help us keep our system healthy. And we should be able to find issues a lot more easily in the future.

We now collect all data for:

- NGINX ingress controller
- Our whole Kubernetes environment, including our ingestion microservice
- Kafka brokers and topics
- Flink job and its operators
- Persistent volumes with underlying NetApp Cloud Volumes ONTAP actual storage
- AWS EC2 virtual machines

We collected all the clues and with the whole picture in front of us solved our mystery.

There's a lot more to explore in Cloud Insights; this investigation just scratches the surface. We can set alerts on any of the components captured in Cloud Insights, and we can collect a lot more data on any other parts of the system. This is just the beginning of what you can do.

Give it a try for free. Just go to the Cloud Insights registration page and get started with your free Cloud Insights trial account.

About NetApp

In a world full of generalists, NetApp is a specialist. We're focused on one thing, helping your business get the most out of your data. NetApp brings the enterprise-grade data services you rely on into the cloud, and the simple flexibility of cloud into the data center. Our industry-leading solutions work across diverse customer environments and the world's biggest public clouds.

As a cloud-led, data-centric software company, only NetApp can help build your unique data fabric, simplify and connect your cloud, and securely deliver the right data, services, and applications to the right people—anytime, anywhere.

NetApp