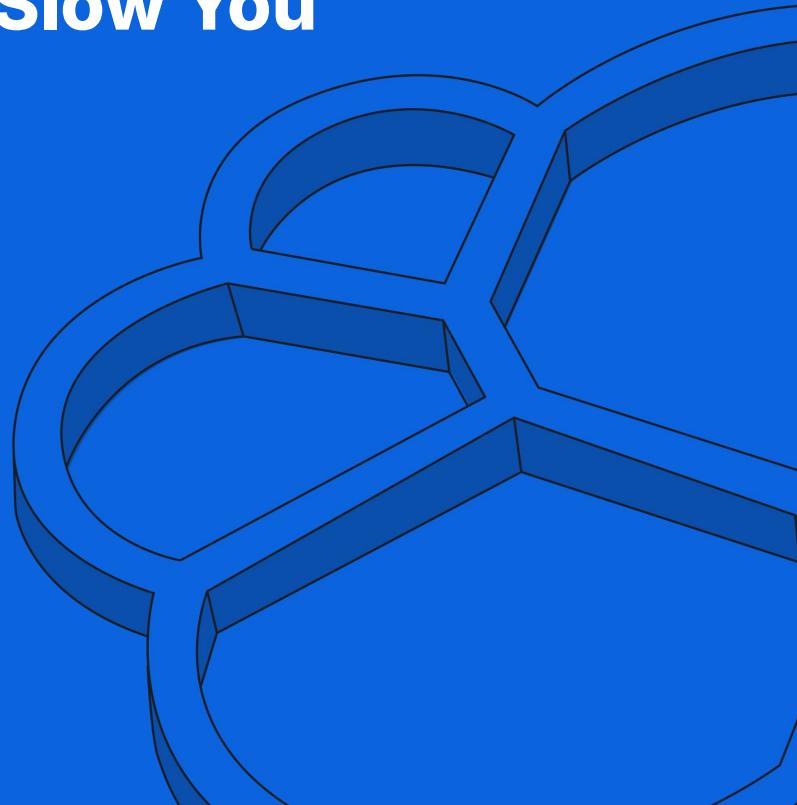


The Performance Paradox: When Slower Code Doesn't Slow You Down

22nd Nov 2025

Anderson Queiroz

Sr Software Engineer, Elastic





**Anderson
Queiroz**
**Sr Software Engineer,
Elastic**

Ingest GZIP-compressed logs

**“The concatenation of a sequence of gzipped files
is equivalent to the gzip of the concatenation of the
sequence.”**

Go's compress/gzip docs

Ingest GZIP-compressed logs

in other words, read a GZIP files, decompress it and ingest the plain data



On-the-fly decompression: no need for extra storage or loading the whole file into memory



Supports offset tracking / partial reading:
can resume if the file wasn't fully read. Only one in the market ;)



Identifies files by fingerprint of the decompressed data - can identify a file 1st seem as plain, then seem as a GZIP file



Integrity Check, checksum (CRC32) and size verification, happens at the end and just logs an error

Filebeat and Elastic Agent

Filebeat

Filebeat is a lightweight shipper for logs

"Whether you're collecting from security devices, cloud, containers, hosts, or OT, Filebeat helps you keep the simple things simple by offering a lightweight way to forward and centralize logs and files."

It's the application which actually does the heavy lifting: reading, processing and shipping the logs to Elasticsearch or to other storages ot for further processing.

The Elastic Agent configures and runs a filebeat under the hood. So new features for log collection are, mostly, developed on Filebeat and the agent just needs to configure/enable them.



Lightweight shipper for logs

Whether you're collecting from security devices, cloud, containers, hosts, or OT, Filebeat helps you keep the simple things simple by offering a lightweight way to forward and centralize logs and files.

[Download](#)[Filebeat documentation →](#)

Aggregate, "tail -f" & search

After you start Filebeat, open the Logs UI and watch your files being tailed right in Kibana. Use the search bar to filter by service, app, host, datacenter, or other criteria to track down curious behavior across your aggregated logs.

The screenshot shows the Elasticsearch Logs interface. The search bar at the top contains the query 'container.name:"metricbeat" and ecs.version:0.2'. The results list several log entries from a file named 'metricbeat'. Each entry includes a timestamp, a log level (INFO), and a detailed log message. The log messages describe various HTTP requests and responses, including connections to 'http://4.8.8.8:8080', 'http://0.0.0.0:8080', and 'http://127.0.0.1:8080'. Some entries mention 'error handling request' and 'block in perform'. The interface includes a search bar, a date range selector, and a 'Stream live' button.

Elastic Agent: all in “one”

Elastic Agent is a single, unified way to add monitoring for logs, metrics, and other types of data to a host.

The Elastic Agent, or “agent” as we say, is a centrally managed, easy to deploy and configure solution.

Users use [integrations](#) to configure it to observe and protect all sorts of endpoints.

The screenshot shows the Elastic website's integration page. At the top, there's a navigation bar with links for Elasticsearch, Solutions, Enterprise, Resources, Docs, Pricing, and a search bar. Below the navigation is a breadcrumb trail: Features > Elasticsearch > Integrations > Docs. The main content area is titled "Elastic integrations" and features a search bar labeled "Search integrations". Below the search bar are filters for "All Solutions", "Search", "Security", and "Observability". A legend indicates that blue icons represent "Only agentless integrations" and orange icons represent "Display beta integrations". A note says "If an integration is available for Elastic Agent and Beats, show:". There are two radio buttons: "Elastic Agent only" (selected) and "Beats only".

This screenshot shows detailed views of several integrations. It includes sections for ".NET", "Abnormal AI", "1Password", "Abuse.ch", "Active Directory Entity Analytics", and "ActiveMQ". Each section contains the integration logo, name, and a brief description of what it does. For example, the ".NET" section shows "Logs Metrics Traces" and the "Abuse.ch" section shows "Threat Intelligence".

This screenshot shows a grid of over 40 available integrations. The grid is organized into four columns. The first column lists integrations like APM, AWS, Azure, Cloud, Config management, Containers, CRM, Custom, Database, Elastic Stack, Elasticsearch SDK, Enterprise Search, Google Cloud, Network, Observability, OpenTelemetry, Operating Systems, Productivity, and Security. The second column lists abuse.ch, Active Directory Entity Analytics, Admin By Request EPM, Airflow, Akamai, AllenVault OTX, Amazon CloudFront, Amazon Data Firehose, Amazon EBS, Amazon EC2, Amazon EMR, Amazon GuardDuty, Amazon Kinesis Data Stream, Amazon Managed Streaming for Apache Kafka (MSK), and Amazon MQ. The third column lists 1Password, ActiveMQ, Airlock Digital, Amazon Bedrock, Amazon DynamoDB, Amazon Inspector, and Amazon ECS. The fourth column lists Abnormal AI, ActiveMQ, Airlock Digital, Amazon GuardDuty, Amazon Inspector, and Amazon MQ. Each integration entry includes a small icon, the integration name, and a brief description of what it does. A sidebar on the right says "Can't find an integration? Create a custom one to fit your requirements" and "Create new integration".



Filebeat inputs: it reads data from

- [AWS CloudWatch](#)
- [AWS S3](#)
- [Azure Event Hub](#)
- [Azure Blob Storage](#)
- [Benchmark](#)
- [CEL](#)
- [Cloud Foundry](#)
- [CometD](#)
- [Container](#)
- [Entity Analytics](#)
- [ETW](#)
- [**filestream**](#)
- [GCP Pub/Sub](#)
- [Google Cloud Storage](#)
- [HTTP Endpoint](#)
- [HTTP JSON](#)
- [journald](#)
- [Kafka](#)
- [Log \(deprecated in 7.16.0, use \[filestream\]\(#\)\)](#)
- [MQTT](#)
- [NetFlow](#)
- [Office 365 Management Activity API](#)
- [Redis](#)
- [Salesforce](#)
- [Stdin](#)
- [Streaming](#)
- [Syslog](#)
- [TCP](#)
- [UDP](#)
- [Unified Logs](#)
- [Unix](#)
- [winlog](#)

Filebeat outputs: it sends data to

- [Elastic Cloud Hosted](#)
- [Elasticsearch](#)
- [Logstash](#)
- [Kafka](#)
- [Redis](#)
- [File](#)
- [Console](#)
- [Discard](#)

The screenshot shows a section of the Elastic Docs website under the 'Reference' category. The left sidebar lists various output options: General settings, Project paths, Config file loading, Output (with sub-options: Elastic Cloud Hosted, Elasticsearch, Logstash, Kafka, Redis, File, Console, Discard, Change the output codec), Kerberos, SSL, Index lifecycle management (ILM), Elasticsearch index template, Kibana endpoint, Kibana dashboards, Processors, Autodiscover, Internal queue, and Logging. The main content area is titled 'Configure the output' and discusses how to set up Filebeat outputs. It includes a list of links corresponding to the items in the sidebar: Elastic Cloud Hosted, Elasticsearch, Logstash, Kafka, Redis, File, Console, and Discard. Navigation buttons at the bottom allow for 'Live reloading' and moving between 'Previous' and 'Next' pages.

Development process

Reading GZIP files

Development workflow

```
filebeat.inputs:  
- type: filestream  
  id: "test-filestream"  
  paths:  
    - /var/some-app/app.log*  
  gzip_experimental: true
```



RFC + PoC: ensure it's viable



Development, testing and review



Merge → Further tests on CI



Benchmark and final review

Benchmarks

We have two different types of benchmark

End-to-end

We have a internal tool called “benchbuilder” to perform end-to-end test of the beats (filebeat, metricbeat, and so on) and the Elastic Agent.

It focus on user experience, using events per second (EPS) as its key metric

More complex and requires extensive setup

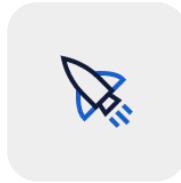
Go / micro-benchmarks

Regular Go benchmarks, testing a function or small component to access a new implementation is indeed better or we aren’t degrading performance with new features or refactors

Easy and cheap to develop and run

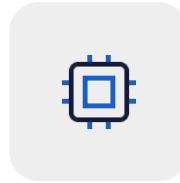
What we want to measure?

- many small files 1000s of 10MB files (Kubernetes log rotation)
- 1 huge file, several GBs (ensure it can work with files larger than the memory available)



Is it slower, the same or faster?

EPS:
Events per second



How much does this increase the system load:

CPU usage



What's the new memory footprint?

Memory usage

Common sense: A Clear Trade-Off



Decompression = CPU Work



More CPU Work = Slower Performance



Slower reading ⇒ Events Per Second
(EPS) will be reduced

PoC benchmarks

Before starting developing the feature we wanted to make sure it was viable

```
› benchstat plain.mac.out gzip.mac.out
```

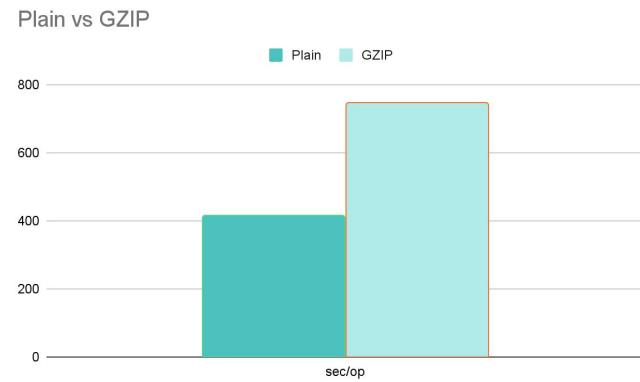
```
goos: darwin
```

```
goarch: arm64
```

```
pkg: github.com/elastic/beats/v7/filebeat/input/filestream
```

```
cpu: Apple M1 Max
```

	plain.mac.out	gzip.mac.out	
	sec/op	sec/op	vs base
Gzip/static-line/1.0_MB-10	12.27m ± 1%	14.78m ± 0%	+20.54% (p=0.000 n=10)
Gzip/static-line/5.2_MB-10	60.05m ± 1%	72.95m ± 0%	+21.48% (p=0.000 n=10)
Gzip/static-line/26_MB-10	300.0m ± 0%	362.5m ± 0%	+20.82% (p=0.000 n=10)
Gzip/static-line/52_MB-10	598.0m ± 0%	723.1m ± 0%	+20.92% (p=0.000 n=10)
Gzip/static-line/105_MB-10	1.199 ± 0%	1.450 ± 2%	+20.90% (p=0.000 n=10)
Gzip/static-line/262_MB-10	3.010 ± 0%	3.616 ± 1%	+20.15% (p=0.000 n=10)
Gzip/static-line/524_MB-10	6.162 ± 0%	7.236 ± 0%	+17.42% (p=0.000 n=10)
Gzip/static-line/1.0_GB-10	12.05 ± 0%	14.53 ± 0%	+20.53% (p=0.000 n=10)
Gzip/random-line/1.0_MB-10	4.890m ± 0%	13.017m ± 0%	+166.18% (p=0.000 n=10)
Gzip/random-line/5.2_MB-10	23.86m ± 0%	63.93m ± 0%	+167.94% (p=0.000 n=10)
Gzip/random-line/26_MB-10	119.7m ± 0%	321.3m ± 0%	+168.47% (p=0.000 n=10)
Gzip/random-line/52_MB-10	240.5m ± 0%	643.9m ± 0%	+167.69% (p=0.000 n=10)
Gzip/random-line/105_MB-10	481.5m ± 0%	1287.3m ± 0%	+167.38% (p=0.000 n=10)
Gzip/random-line/262_MB-10	1.204 ± 0%	3.220 ± 0%	+167.50% (p=0.000 n=10)
Gzip/random-line/524_MB-10	2.411 ± 0%	6.423 ± 0%	+166.46% (p=0.000 n=10)
Gzip/random-line/1.0_GB-10	4.828 ± 0%	13.047 ± 2%	+170.21% (p=0.000 n=10)
geomean	415.9m	746.6m	+79.49%



Reading GZIP files

Development workflow

```
filebeat.inputs:  
- type: filestream  
  id: "test-filestream"  
  paths:  
    - /var/some-app/app.log*  
  gzip_experimental: true
```



~~RFC + PoC: ensure it's viable~~



~~Development, testing and review~~



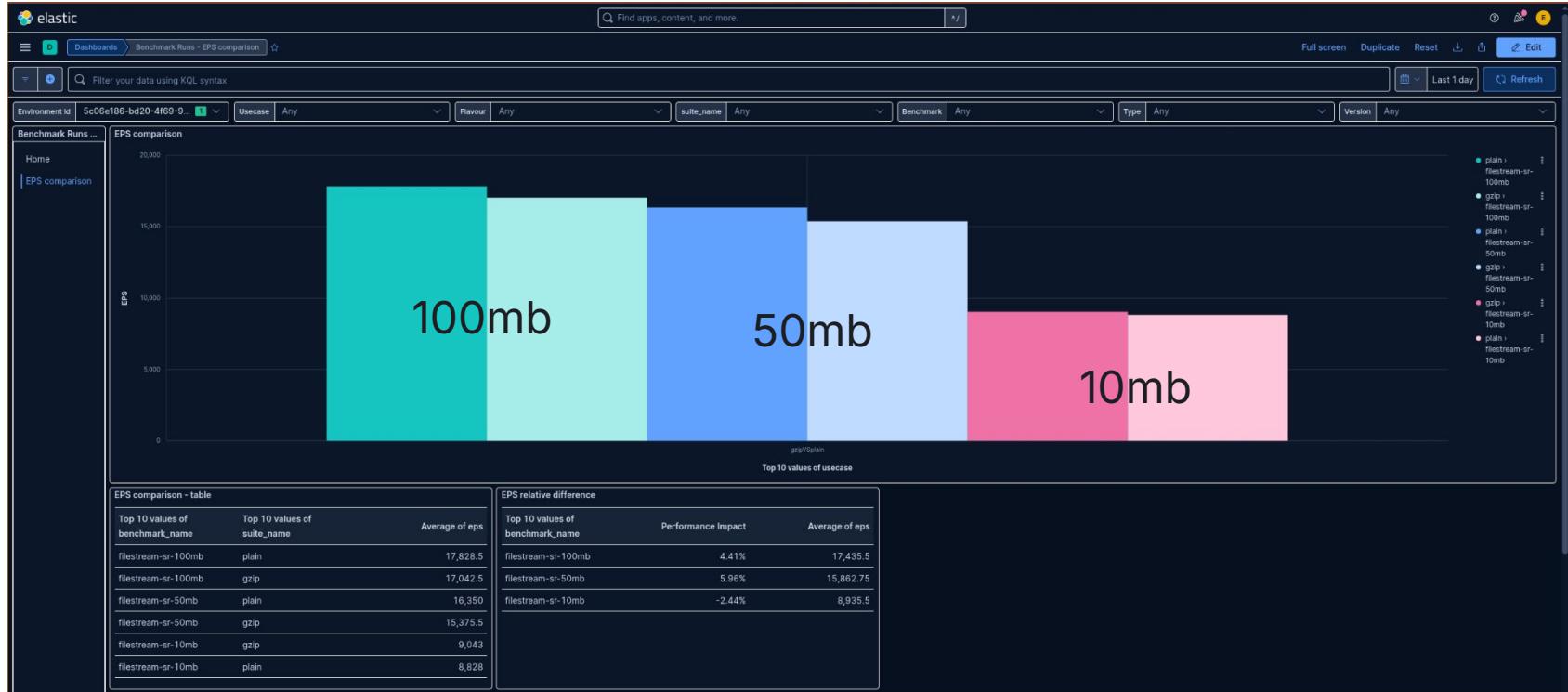
~~Merge → Further tests on CI~~



Benchmark and final review

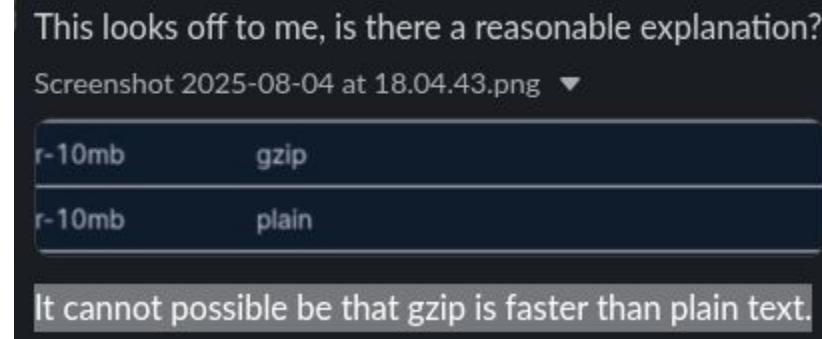
Benchmarks

1st EPS comparison



Small difference between EPS output

Odd, GZIP was faster. WHY?!

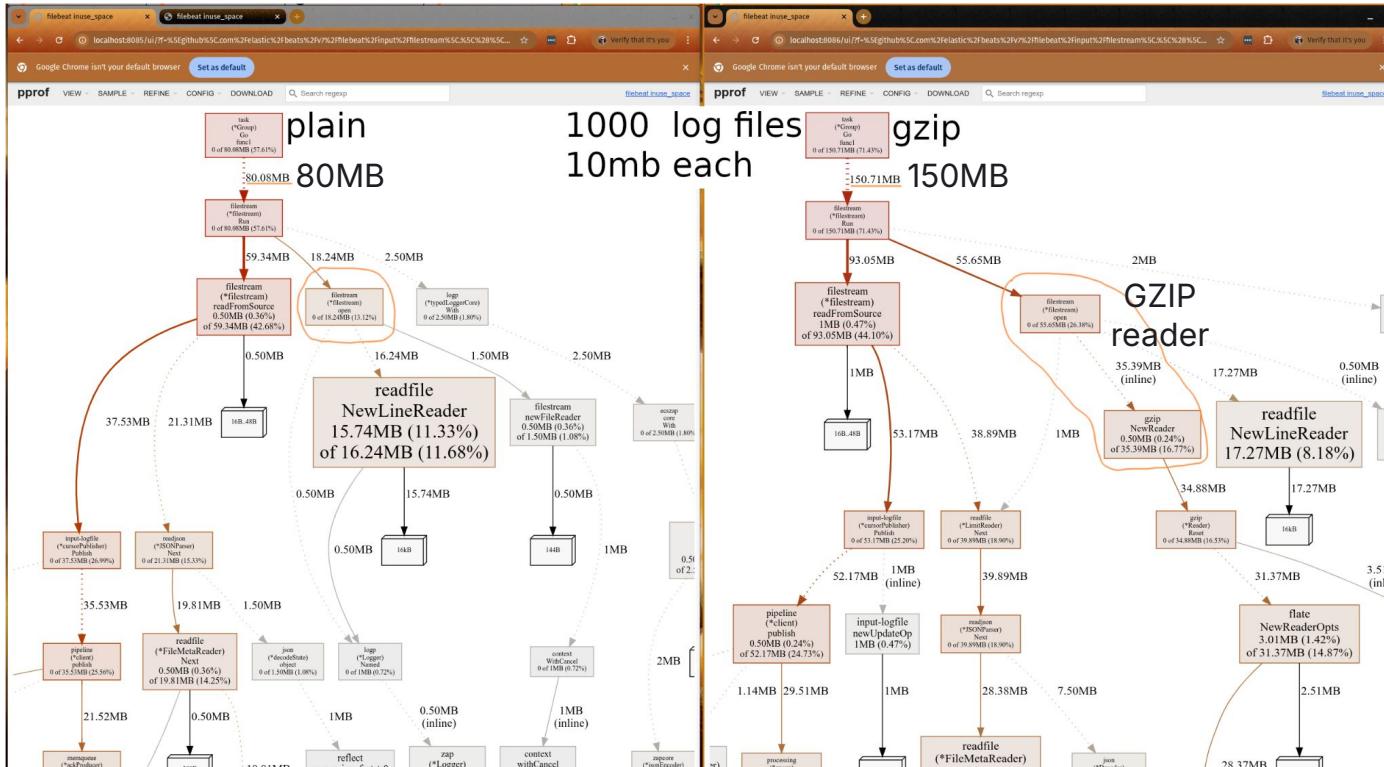


filestream-sr-10mb	gzip	9,043
filestream-sr-10mb	plain	8,828

Memory analysis:

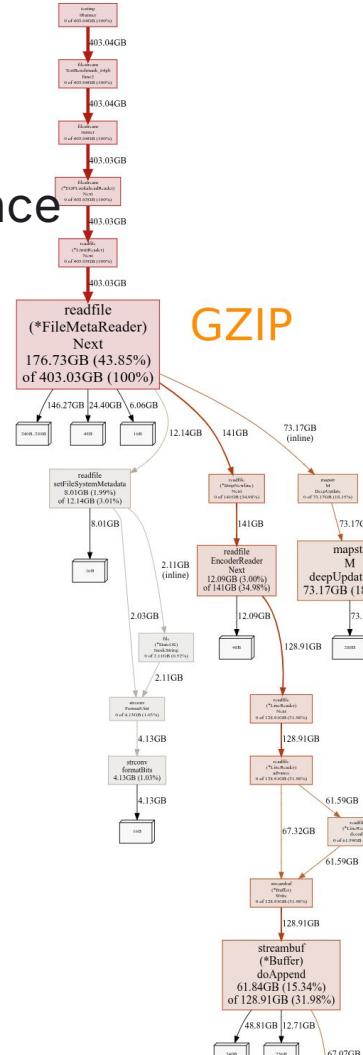
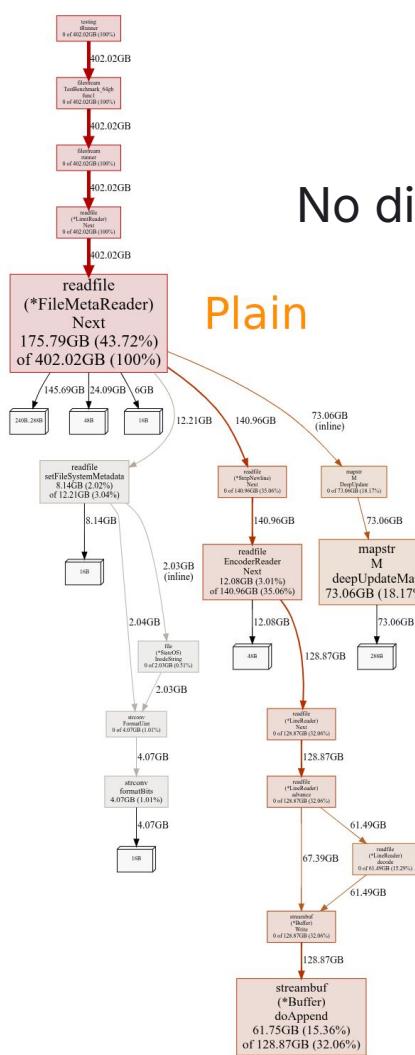
Go benchmark

Memory usage: 1000×10MB file



Memory usage: 1x64GB file

No difference



Memory usage: conclusion

- 100Kb increase in memory usage per file.

Mitigation:

- **limiting the number of files read concurrently.** On Filebeat terms, limit the number of *harvesters*.
- **closing files as soon as then end is reached.** It's the behaviour for GZIP files as no data should be appended to them.

CPU usage: Go benchmark

CPU usage

Just as for the memory benchmarks:

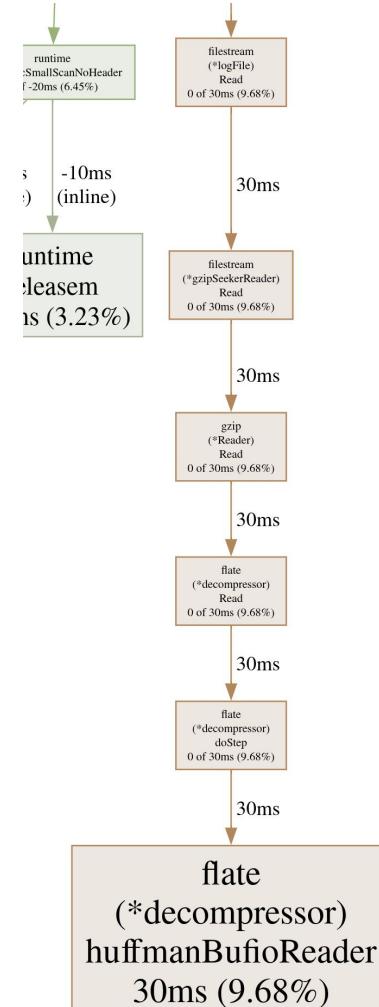
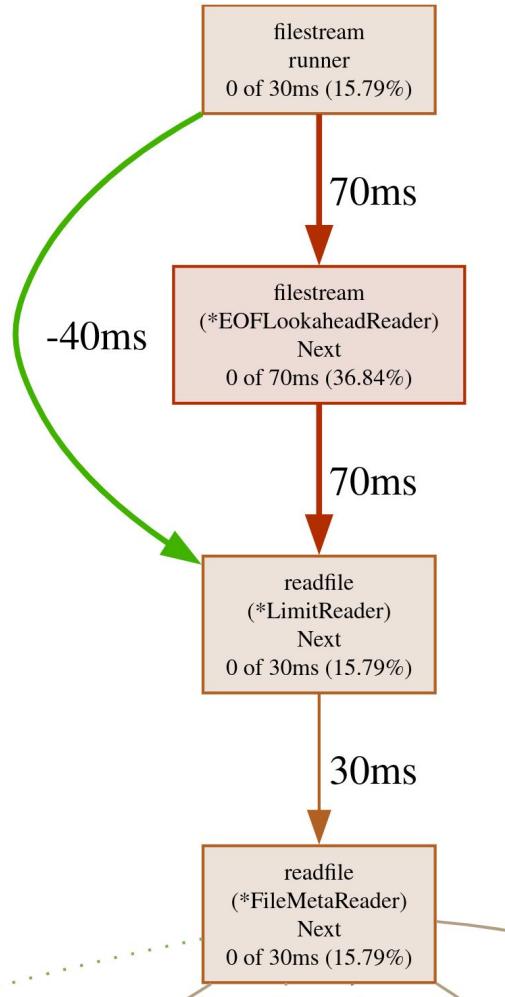
A local benchmark was conducted to evaluate the file reading process in isolation.

This test instantiated the reader abstraction used by filestream, excluding overhead from other Filebeat components like processors and outputs.

We see a increase in CPU usage if we look only at reading the files.

```
go test -v -run TestBenchmark_64gb -timeout=0
    === RUN   TestBenchmark_64gb
    === RUN   TestBenchmark_64gb/plain
    === RUN   TestBenchmark_64gb/gzip
    --- PASS: TestBenchmark_64gb (1130.28s)
    --- PASS: TestBenchmark_64gb/plain (251.57s)
    --- PASS: TestBenchmark_64gb/gzip (329.39s)
PASS
```

CPU profile



CPU profile

Plain



GZIP

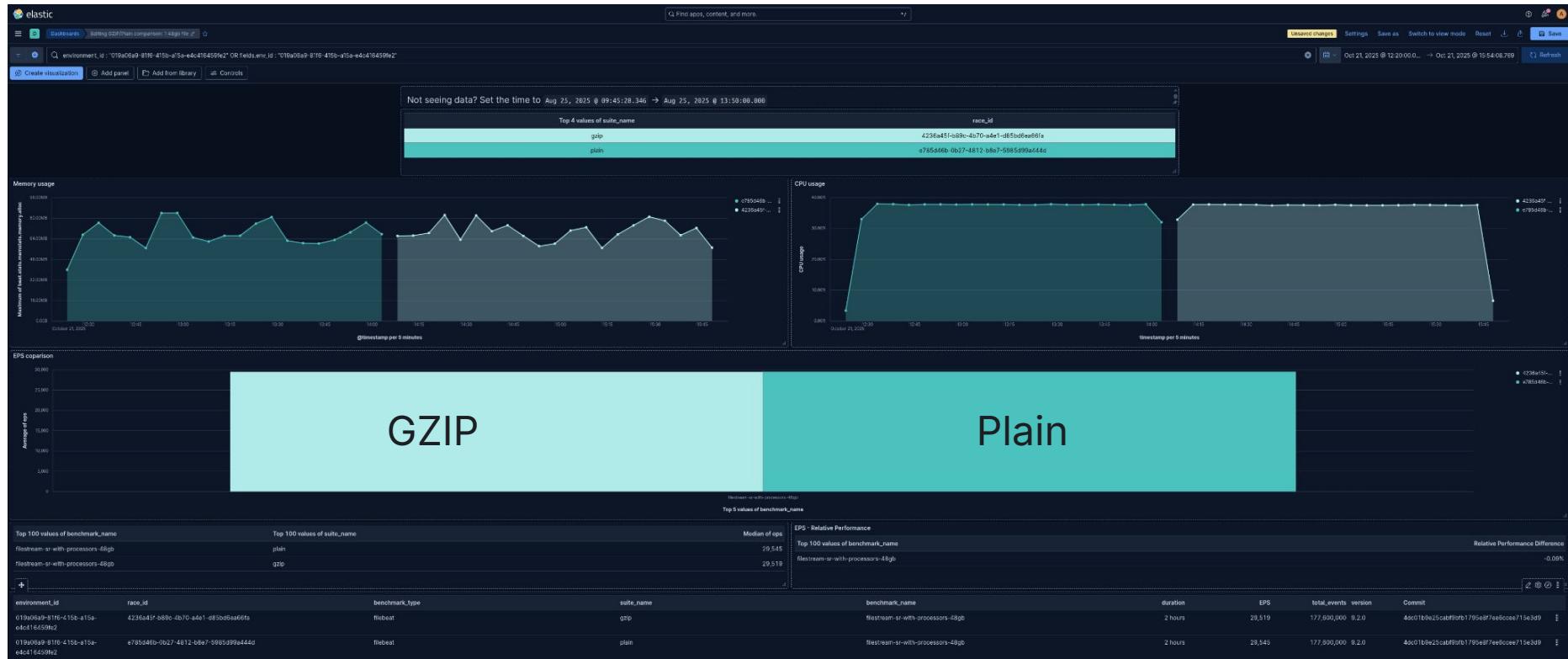


CPU usage: conclusion

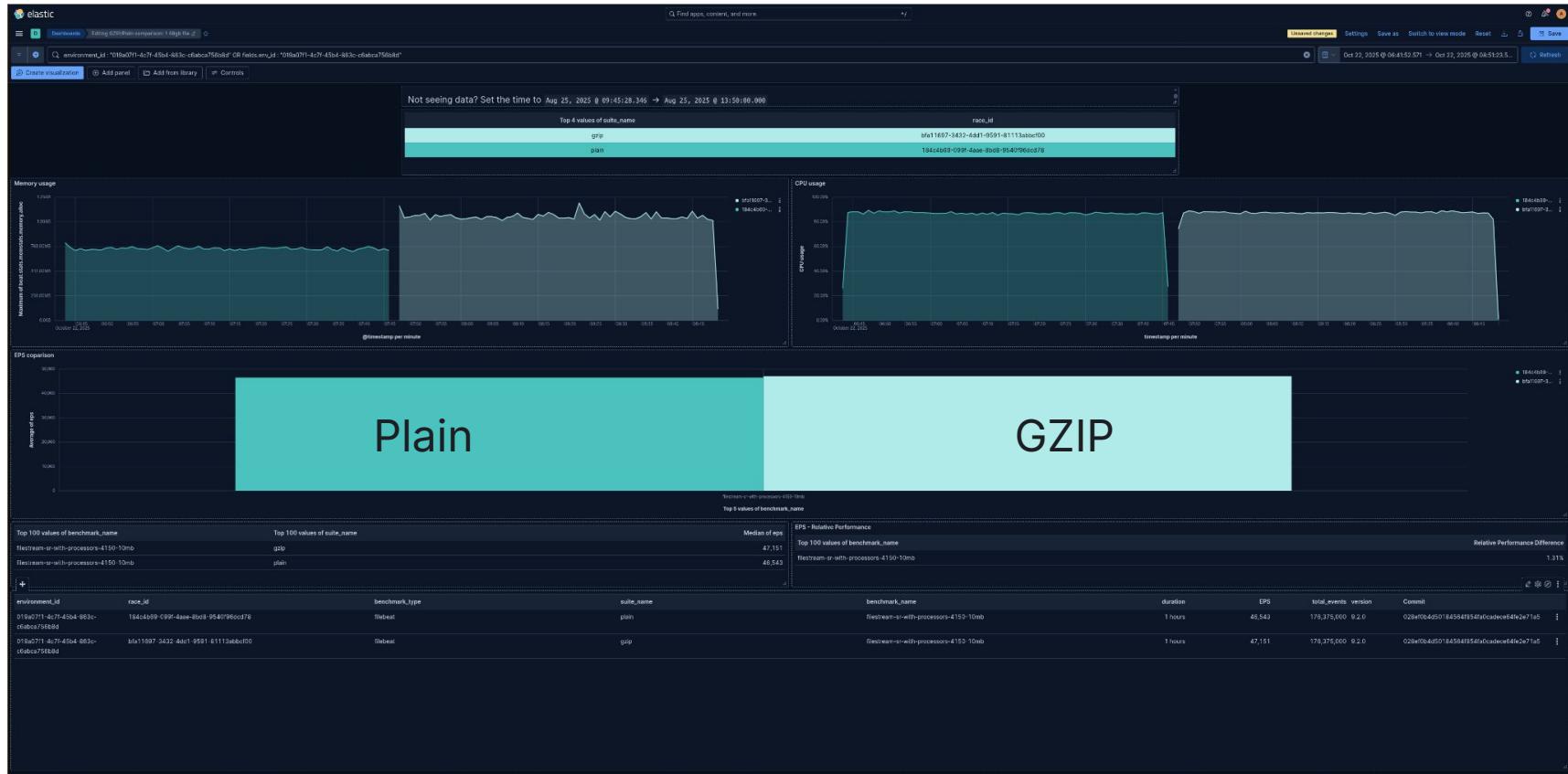
- Increase due to the *EOFLookaheadReader* -
signals EOF for GZIP files
- Decompression also increases CPU usage
what a surprise!
- Decompression isn't really expensive

Final end-to-end benchmarks

1 × 48GB file



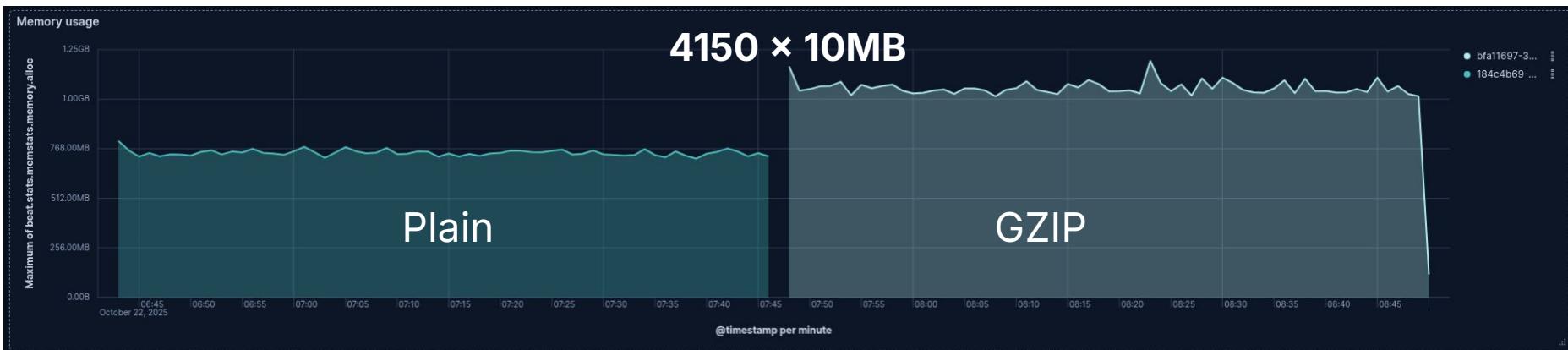
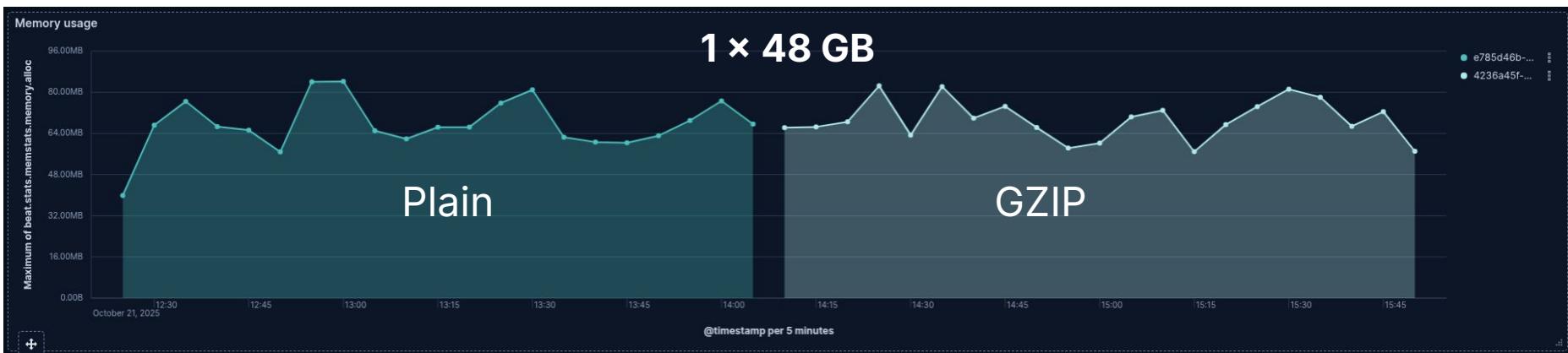
4150 × 10MB files



CPU usage



Memory usage



EPS comparison

1 × 48 GB



4150 × 10MB



EPS comparison

1 × 48 GB

EPS - Relative Performance							
Top 100 values of benchmark_name							Relative Performance Difference
filestream-sr-with-processors-48gb							-0.09%

environment_id	raco_id	benchmark_type	suite_name	benchmark_name	duration	EPS	total_events	version	Commit
019a0ca0-8116-415b-a15e- 6cd0164591e2	4236a45f-bd9c-d870-a4e1-d83ad0ea0f6a	filebeat	grip	filestream-sr-with-processors-48gb	2 hours	29,519	177,600,000	9.2.0	4dc01bfe25cabff0fb1795e87ee6ccae215e3d9
019a0ca0-8116-415b-a15e- 6cd0164591e2	e785d6b-0b17-4812-b807-5985cf99ed4d	filebeat	plain	filestream-sr-with-processors-48gb	2 hours	29,545	177,600,000	9.2.0	4dc01bfe25cabff0fb1795e87ee6ccae215e3d9

4150 × 10MB

EPS - Relative Performance							
Top 100 values of benchmark_name							Relative Performance Difference
filestream-sr-with-processors-4150-10mb							1.31%

environment_id	raco_id	benchmark_type	suite_name	benchmark_name	duration	EPS	total_events	version	Commit
019a07f1-4c71-45a4-863c- cfa0ca75fd8d	164cf8d9-090f-4aae-8bd9-9540f96cd376	filebeat	plain	filestream-sr-with-processors-4150-10mb	1 hours	46,543	176,375,000	9.2.0	c2ae034e5018456d854fc0dec6dfe271a5
019a07f1-4c71-45a4-863c- cfa0ca75fd8d	b611667-3422-4cd1-9591-6111abcf00	filebeat	grip	filestream-sr-with-processors-4150-10mb	1 hours	47,151	176,375,000	9.2.0	c2ae034e5018456d854fc0dec6dfe271a5



Understanding Filebeat

In a nutshell: How Filebeat works?

Filebeat, reads the data (the logs), process it, then ships it away

Input: filestream

Monitor the file system, find log files, read the files, keep track of current offset for each file, transform each log into filebeat's internal model

Pipeline / Queue

Apply configurable processors which can do all sorts of transformations such as add, remove fields, JSON parsing and field expansion and so on.

Output

Send out the events (the logs). Usually Elasticsearch, but also Logstash, Kafka, Redis and so on.

In a nutshell: How filestream works?

How logs are ingested



File watcher

The file watcher scans the file system every 10s (configurable) to detect:

- new files
- data appended to a file
- a file is truncated
- a file is deleted

For each, an event is sent

Prospector

For each event, it decides if a harvester should:

- start
- stop
- continue
- restart (reset the offset to 0 and start from the beginning)

Harvester

Actually reads the files.

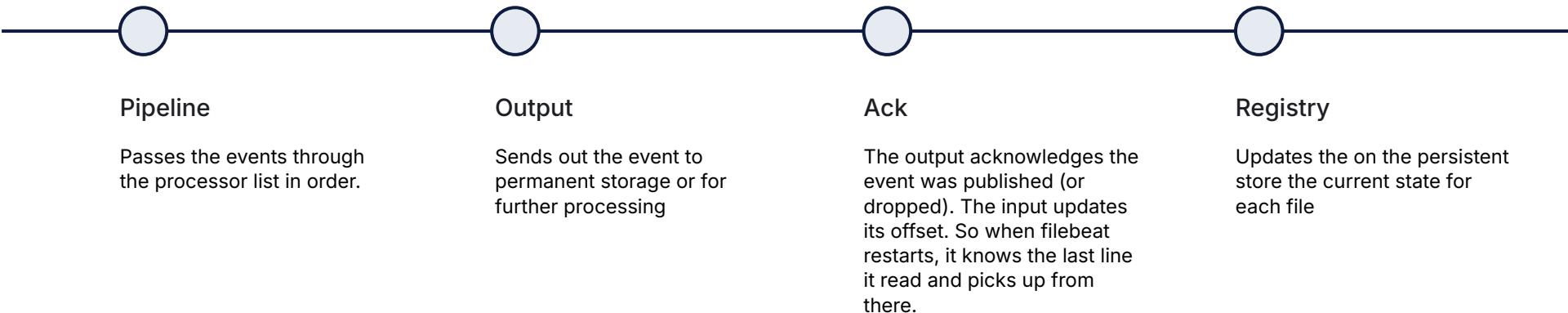
Reads one line at a time and can produce a single event (log) from multiple lines

Publisher

Sends the event to the Pipeline for further processing

In a nutshell: How filestream works?

How logs are ingested



Output: shipping data over the network

Sending data over the network adds overhead for sure. Also the Elastic Agent and beats support different output which each one will have its own performance and limitations.

We have a discard output, which, immediately discards any data it receives, allowing us to minimise the impact of sending data and develop without any extra setup.

Benchbuilder does not supports it, but local tests showed it did not affect the performance of ingesting *plain* or *gzip* files.

The screenshot shows a detailed configuration guide for the Elasticsearch output in Filebeat. It includes sections for basic authentication, API key authentication, and PKI certificate authentication, each with code examples. A sidebar on the left provides navigation through various Elasticsearch components like Config, Inputs, Modules, and more.

Configure the Elasticsearch output

The Elasticsearch output sends events directly to Elasticsearch using the Elasticsearch HTTP API.

Example configuration:

```
output.elasticsearch:
  hosts: ["https://myEHost:9200"]
```

To enable SSL, add https to all URLs defined under hosts.

When sending data to a secured cluster through the `elasticsearch` output, Filebeat can use any of the following authentication methods:

- Basic authentication credentials (username and password).
- Token-based (API key) authentication.
- Public Key Infrastructure (PKI) certificates.

Basic authentication:

```
output.elasticsearch:
  hosts: ["https://myEHost:9200"]
  username: "filebeat_writer"
  password: "YOUR_PASSWORD"
```

API key authentication:

```
output.elasticsearch:
  hosts: ["https://myEHost:9200"]
  api_key: "zCVVnW8gnXOT19tQeKtR0yE41RrSowb0kQ0Hw0A"
```

PKI certificate authentication:

```
output.elasticsearch:
  hosts: ["https://myEHost:9200"]
  ssl.certificate: "/etc/pki/client/cert.pem"
  ssl.key: "/etc/pki/client/cert.key"
```

See [Secure communication with Elasticsearch](#) for details on each authentication method.

Compatibility

Current version (9.0+) ▾

- View as Markdown
- Report an issue
- Edit this page
- Learn how to contribute

On this page

- Compatibility
- Configuration options
 - enabled
 - hosts
 - compression_level
 - escape_html
 - worker or workers
 - loadbalance
 - api_key
 - username
 - password
 - parameters
 - protocol
 - path
 - headers
 - proxy_disable
 - proxy_url
 - proxy_headers
 - index
 - indices
 - ilm
 - pipeline
 - pipelines
 - max_retries
 - bulk_max_size
 - backoff_init
 - backoff_max
 - idle_connection_timeout
 - timeout
 - allow_older_versions
 - ssl
 - kerberos
 - queue
 - non_indexable_policy
 - Elasticsearch APIs

elastic

Processors: impact on overall performance

Logs, or events as we call them, go one by one from the input to the pipeline. Passing by each processor in sequence.

One guarantee we give is if a processor fails, we can still deliver the original event this processor received. All further processors are ignored.

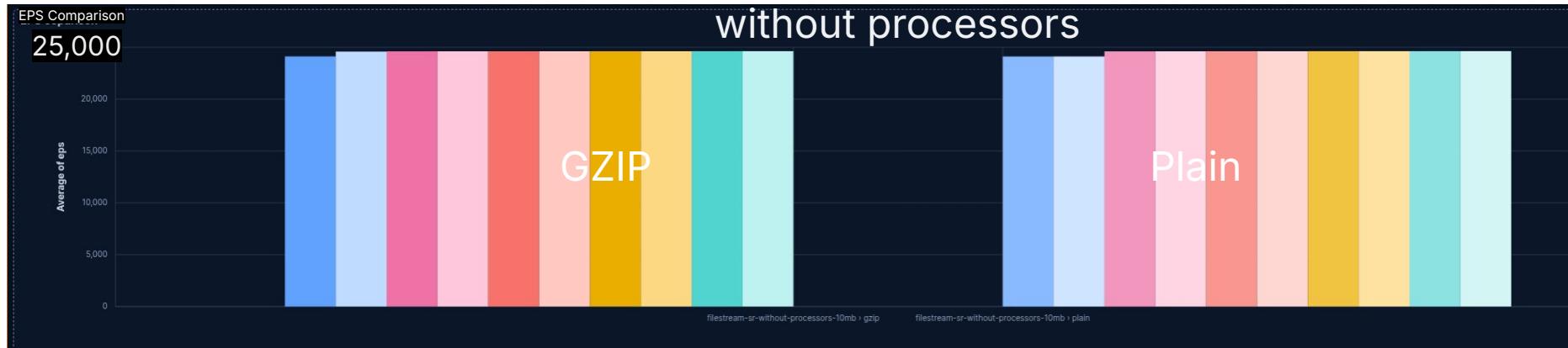
Processors which perform destructive actions must work on a copy of the event, allowing to “rollback” on error.

The screenshot shows a section of the Elastic documentation for Filebeat Processors. The left sidebar lists various processor types: Define processors, add_cloud_metadata, add_cloud_file_metadata, add_docker_metadata, add_fields, add_host_metadata, add_id, add_kubernetes_metadata, add_labels, add_locale, add_network_direction, add_nomad_metadata, add_observer_metadata, add_process_metadata, add_tags, append, cache, community_id, convert, copy_fields, decode_base64_field, decode_cef, decode_csv_fields, decode_duration, decode_json_fields, decode_xml_fields, decode_xml_wineventlog. The main content area has a title "Filter and enhance data with processors". It includes a "Stack" section with a note about filtering and enhancing data, and a "Processors" section with a list of benefits: reducing the number of exported fields, enhancing events with additional metadata, performing additional processing and decoding. Below this is a note about processors executing in order. A "Drop event example" section shows a configuration snippet:

```
processors:
  - drop.event:
```

Remove the processors

EPS comparison



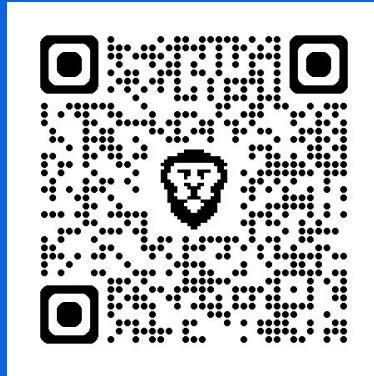
Remove the processors

Top 100 values of benchmark_name	Top 100 values of suite_name	Median of eps
filestream-sr-without-processors-100mb	plain	35,448
filestream-sr-without-processors-100mb	gzip	35,240
filestream-sr-without-processors-10mb	plain	24,618.5
filestream-sr-without-processors-10mb	gzip	24,615.5
filestream-sr-with-processors-10mb	plain	19,679
filestream-sr-with-processors-10mb	gzip	19,657.5
filestream-sr-with-processors-100mb	plain	19,438
filestream-sr-with-processors-100mb	gzip	19,414

EPS - Relative Performance	
Top 100 values of benchmark_name	Relative Performance Difference
filestream-sr-with-processors-100mb	0.03%
filestream-sr-with-processors-10mb	-0.39%
filestream-sr-without-processors-100mb	-0.43%
filestream-sr-without-processors-10mb	0.19%

Conclusion

- Reading GZIP files is slower than reading plain files
- However, it does not affect Filebeat/Elastic Agent performance
- The bottleneck is the queue/pipeline
- There is a ~100KB memory increase per file read
- No CPU increase in the overall performance



github.com/AndersonQ/talks
Email: me@andersonq.me

Questions?

Thank you!



LinkedIn: **Anderson Queiroz**
GitHub: **@AndersonQ**