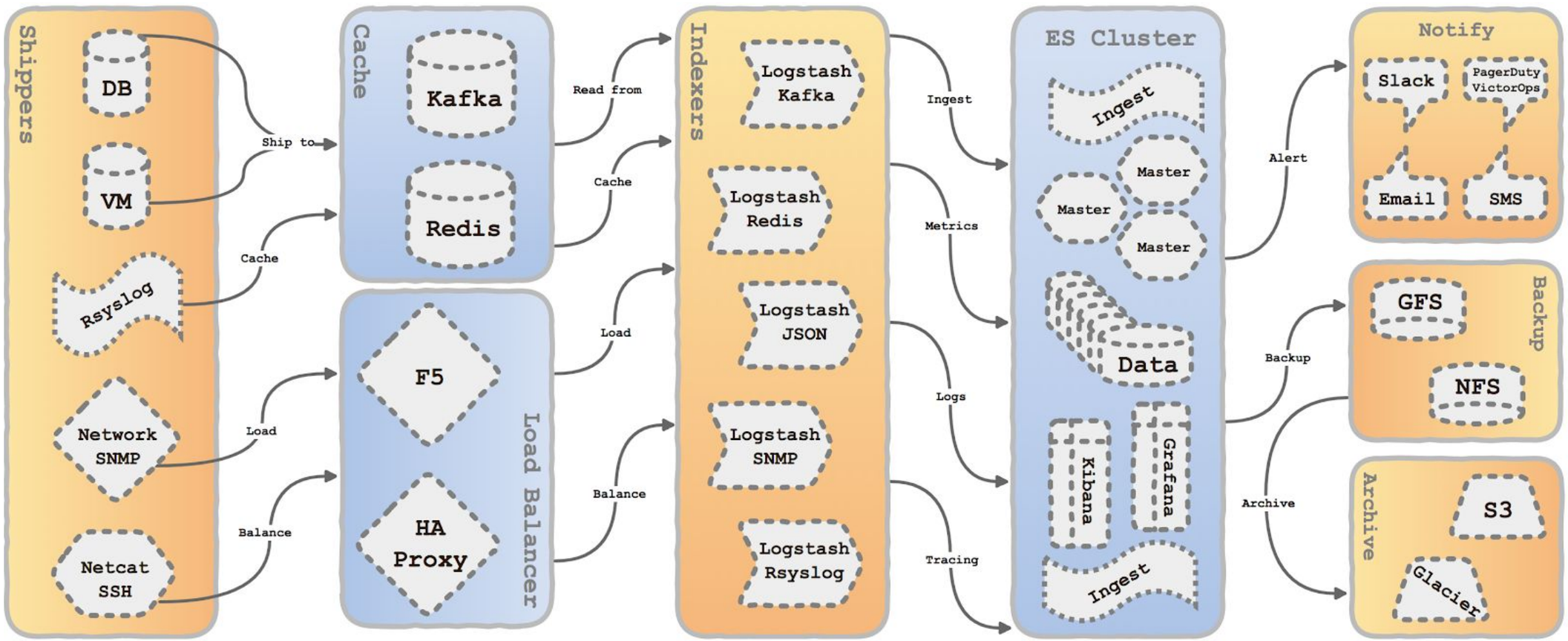
Application Monitoring Architecture



Overview

We need to re architect our application monitoring system because let's say we want to get a more holistic view. The system should be fault tolerant, scalable and maintainable. This monitoring data should also be available and usable by all engineers during outages, troubleshooting issues, capacity planning and improving performance.

Objectives

Metrics: Collect basic system metrics such as Memory, Disk, I/O, Network, Load, Procs, SSH, Ports and Services. Collect advance metrics such as Databases and Application runtime. On the application side we should monitor the HTTP status, load times, error rates and build status.

Logging: System logs and Application logs give further insight into the specifics of the the metrics seen above. These logs should be regularly shipped out to a central store and searchable by Engineers.

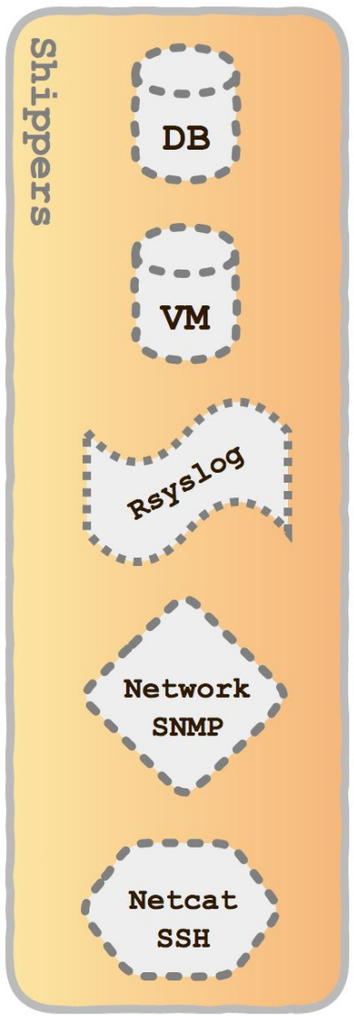
Tracing: Application Performance Monitoring (APM) is helpful in debugging, monitoring the performance and general profile of the code itself. It could help us identify things such as long running calls and slow queries.

Note: The above are just a few examples, the system will be flexible enough to add any kind of monitoring.

Fault Tolerant and Scalable: It should scale as more systems and services are added and monitored. If a portion of the system goes down it should still be able to handle the load until the issue is fixed. Doing maintenance such as config changes and upgrades should be simple and routine.

Holistic view: Everyone in the Engineering team should be able to use and integrate with little to no effort. Engineers should be encouraged to add as many data points as possible while designing the App or feature. It should have a dashboard to view the metrics and logs which will help see historical data and aide in capacity planning.

Data Retention: These metrics and logs should be available to be viewable and searchable for at least 1 year



Shippers are agents that send data payload from the app systems to the backend.

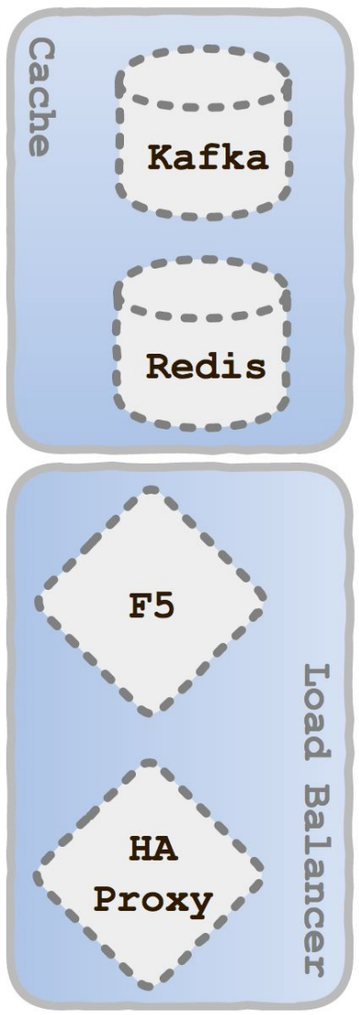
Common OS tools include

- Rsyslog
- SNMP
- NetCat (nc)
- SSH
- Built-in App Stream

Common agents include

- Filebeat
- Metricbeat
- Packetbeat
- Icinga Agent
- StatsD

I realize 3rd party tools are not allowed but I strongly believe we should evaluate on a case by case basis. Reason being some application are better designed to work with the agent than the common tools available from OS.

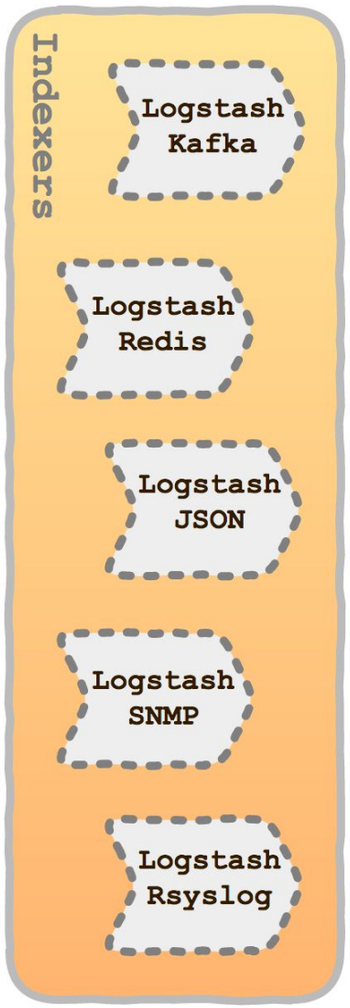


Cache layer is needed if the data needs to be held somewhere for a moment if the systems behind it are overloaded or multiple systems need to read the same payload. It also helps with the following:

- Load Balancing
- Buffering data
- Infra Outages
- Performing Maintenance

Load Balancer is used when a shipper does not support one of the above caching layers, common systems include:

- Routers
- Switches
- IoT
- Network gear
- Security gear



Indexers read the metric or log messages from the cache and process them. Once the payload is received the data is transformed to be put into a datastore.

There are a number of listeners supported:

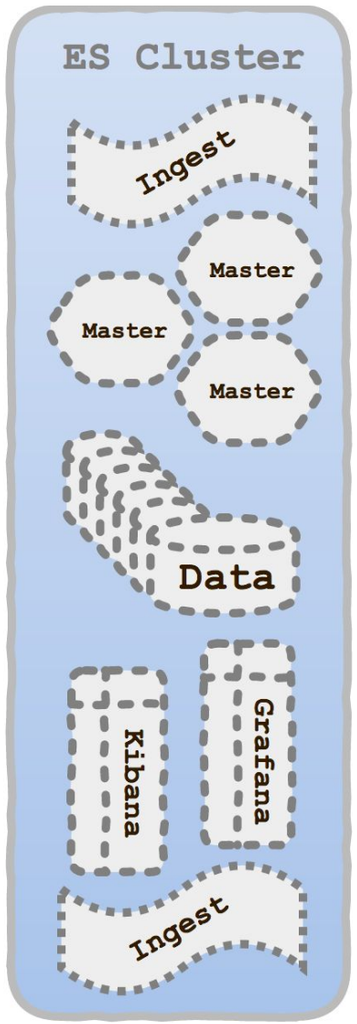
- TCP
- SNMP
- Rsyslog
- Beats

To normalize and grok the data you can use:

- JSON
- RegEx
- Transform

Data can be put into one of the following backends:

- InfluxDB
- OpenTSDB
- ElasticSearch



ElasticSearch cluster is the backend we chose for a number of reasons.

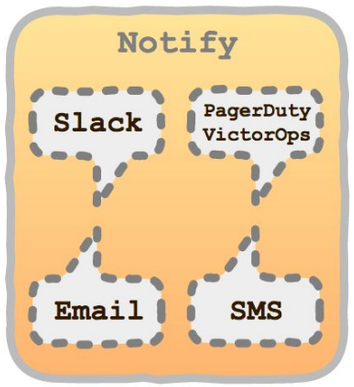
- Can scale well without having issues.
- Has good community support and used in other parts of the infra.
- Compatible with rest of the ecosystem.
- HA features are free.

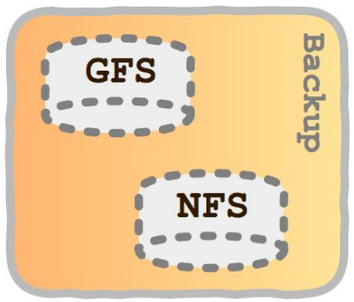
Every component in the ES cluster is designed with fault tolerance in mind. There are multiple ingest, master and data nodes. Even Kibana and Grafana are taken into consideration by deploying 2 instances of each.

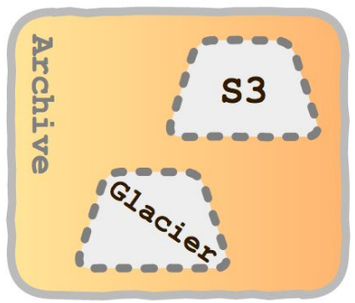
Each component in the cluster will be configured to do a specific task:

- Master: Keeps state of the cluster.
- Ingest: Listens for payload.
- Data: Indexes and Stores the payload.
- Coordinator: Kibana and Grafana UIs.

Moreover, each component can be scaled discretely from each other.







Notify when systems go down or application error. Alert via multiple mediums:

- Slack
- PagerDuty / VictorOps
- Email
- SMS

Backup the data because HA != Backup. Data can get corrupted and humans have known to make snafus.

- GlusterFS
- NFS
- SAN

Archive the backup offsite in a different region because a meteor might take out the datacenter at any moment.

- S3
- Glacier

Frequently Asked Questions

- **Redis or Kafka for caching?**
  - Determine if caching is required in the first place. Logstash can do some rudimentary caching and buffering, but if the data volume is huge, evaluate Redis and then if need be go with Kafka. Please note if you do not have a caching layer you must implement some form of load balancing to spread out the traffic.

- **Were any other Indexers/Transformers evaluated?**
  - Besides Logstash, Fluentd is another comparable transformer which supports ES. It does lack in some tight integration that Logstash has with ES since both of them are made by the same group.

- **Were any other Backends evaluated?**
  - OpenTSDB and InfluxDB were evaluated but were eventually discarded as they did not meet all the objectives listed above. For instance, OpenTSDB has the overhead of Hadoop cluster. InfluxDB has some issues when scaling with huge amounts of metrics because of the storage engine they use, also, the HA features are not available in community edition.