HashiCorp Vault

Telemetry & Monitoring

Monitoring Patterns

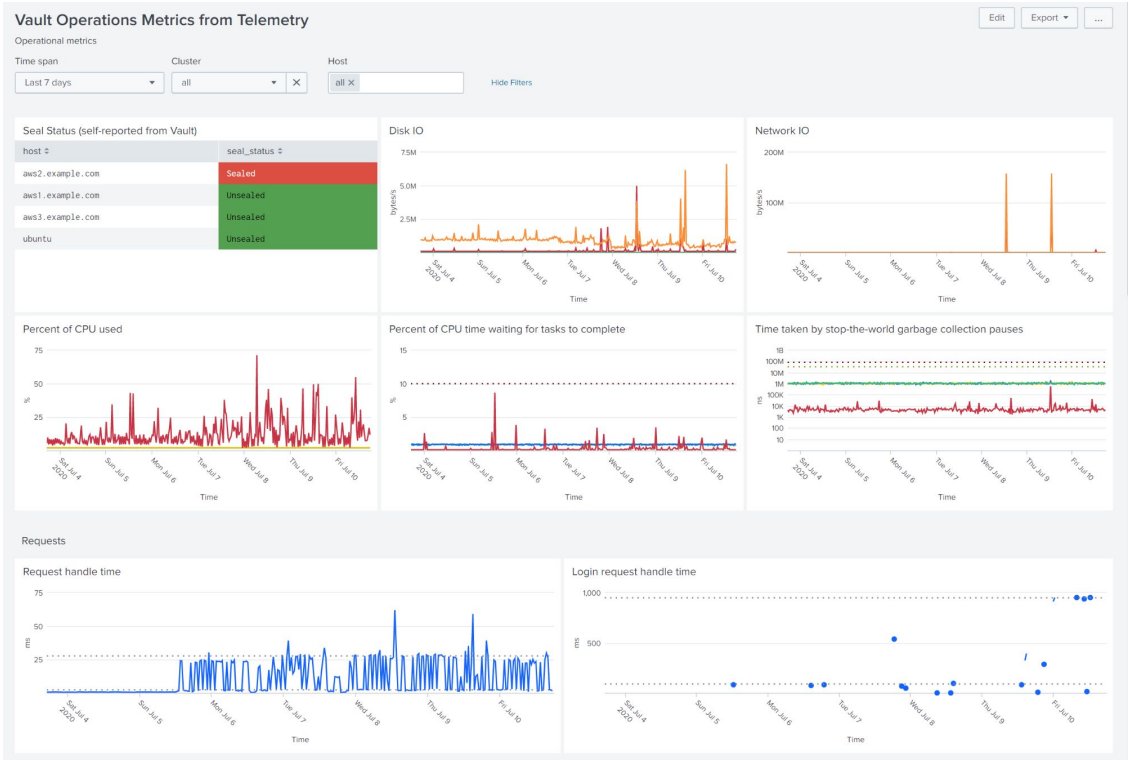
- Vault is typically classified as a Tier 0 application based upon critical applications having it as a upstream dependency
- Monitoring the health of Vault clusters in Production Environments should include all three of the following patterns:
    1. Time-series Telemetry Data
    2. Log Analytics
    3. Active Health Checks

Vault Telemetry

- Vault uses the Golang go-metrics package to export Telemetry Metrics to an upstream service, specified in the telemetry stanza
- Supported sinks include:
  - Circonus
  - DogStatsd
  - Prometheus
  - Statsd
  - Statsite

Vault Metrics

- Run Time Metrics are aggregated across a 10 second interval and retained in memory for 1 minute
- Telemetry from Vault is best stored in a metrics aggregation platform to collect durable metrics and visualize trends



Metric Types

[C] Counter

Cumulative metrics that increment when an event occurs and are reset at the end of the reporting interval

[G] Gauge

Provides measurements of current values

[S] Summary

Provide sample observations of values. Commonly used to measure timing duration of discrete events in the reporting interval.

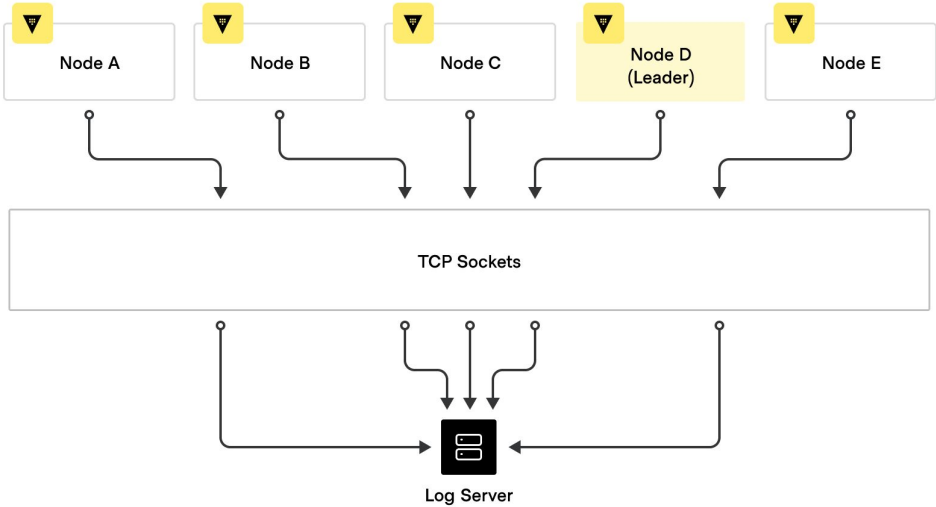
Vault Logging

- Vault generates two types of logs
  - Vault **server logs** contain operational data, plugin errors, debug dumps, and critical security events
  - JSON **audit logs** keep a detailed log of *every authenticated* interaction with Vault, including errors
- Multiple audits devices can (and should) be enabled and Vault will send audit logs to all of them
  - Redundant log copies can be used to check for data tampering in log files
  - Vault considers a request successful if it can log to at least one configured audit device
- If you have only one audit device enabled and it is blocked (network issue, full disk, etc) then Vault will seal itself and cease operations ***by design***

Vault Logging

Socket-to-remote Log Server

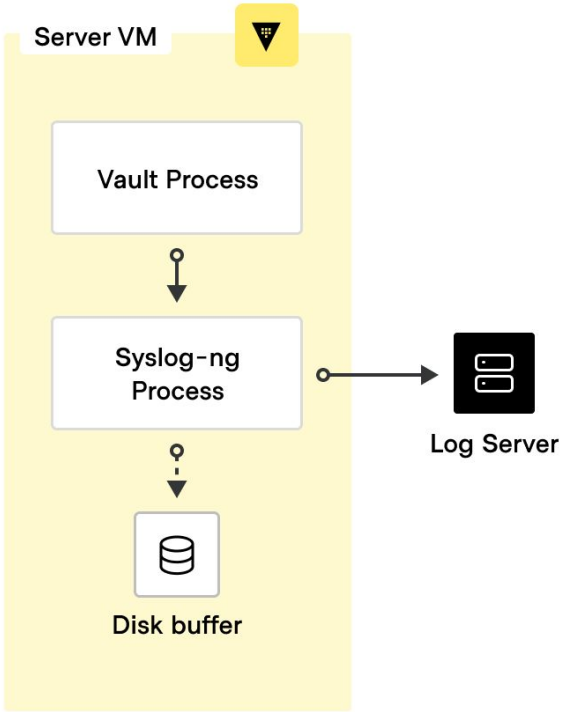
- Vault's socket audit device writes to a TCP, UDP, or UNIX socket
- Using a network log server retains logs in JSON format for easy parsing, and eliminates the potential to fill a local filesystem
- Use with a second audit method or with multiple instances to prevent downtime caused by log server failure



Vault Logging

Syslog-ng daemon

- Utilize syslog-ng daemon to forward audit logs directly to a centralized log server
- Any analysis or resource intensive tasks are performed on the remote log server while keeping Vault simple and continuing to service requests
- Very reliable solution because of its simplicity



Enable Telemetry

[Telemetry - Configuration]

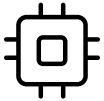
- Specify upstream monitoring service in telemetry stanza in Vault Server configuration

```
telemetry {
  statsd_address = "statsd.company.local:8125"
}
```

Contributing Factors in Performance

- Know the expected workload
- Vault System Resources (CPU, MEM, Disk)
- Complexity of the Vault Policies
- Audit Logging
- Network for all the things

Contributing Factors in Performance



Vault CPU

Select the host VMs to handle the concurrent workload

Audit Device

Be sure that the write location won't become the bottleneck

Policies

Never run load test using root token

Network

Know all the systems that are involved and connectivity between them

Storage Backend

- Determine appropriate TTLs for tokens and leases
- Leverage batch tokens and Vault Agent Caching

Key System Metrics

Metric	Description	What to look for?
cpu.user_cpu	Percentage of CPU being used by user processes	Look for high Vault CPU consumption
cpu.iowait_cpu	Percentage of CPU time spent waiting for I/O tasks to complete	Look for `cpu.iowait_cpu` greater than 10%
net.bytes_recv	Bytes received on each network interface	Look for sudden large changes to the net metrics (greater than 50% deviation from baseline)
net.bytes_sent	Bytes transmitted on each network interface	
linux_sysctl_fs.file-nr	Number of file handles being used across all processes on the host	If the `file-nr` reaches 80% of the `file-max` then you should alert and investigate
linux_sysctl_fs.file-max	Total number of available file handles	

Key Integrated Storage Metrics

Metric	Description	What to look for?
vault.runtime.total_gc_pause_ns	Number of nanoseconds consumed by stop-the-world garbage collection (GC) pauses since Vault started	This would be considered a _warning_ if `total_gc_pause_ns` exceeds 2 seconds/minute and _critical_ if it exceeds 5 seconds/minute
vault.raft.leader.lastContact`	Time to retrieve a value for a path	Look for candidate > 0, or leader > 0, or `lastContact` greater than 200ms which indicates that consensus is unhealthy
vault.raft.state.candidate	Time to insert a log entry to the persist path	
vault.raft.state.leader	This increments whenever a raft server becomes a leader	
diskio.read_bytes	Bytes read from each block device	You will want to monitor for large changes to the diskio metrics for greater than 50% deviation from baseline, or more than 3 deviations from your standard baseline. Then you will want to monitor for over 80% utilization on block device mount points on which Vault data are persisted.
diskio.write_bytes	Bytes written to each block device	
disk.used_percent	Per-mount-point block device utilization	

Key Usage Metrics

Metric	Description
vault.token.creation	A new service or batch token was created
vault.token.count	Number of service tokens available for use.
vault.token.count.by_auth	Number of existing tokens broken down by the auth method used to create them.
vault.token.count.by_policy	Number of existing tokens, counted in each policy assigned.
vault.token.count.by_ttl	Number of existing tokens, aggregated by their TTL at creation.
vault.secret.kv.count	Count of secrets in key-value stores.
vault.secret.lease.creation	Count of leases created by a secret engine (excluding leases created internally for token expiration.)

Thank you

academy-onboarding@hashicorp.com

www.hashicorp.com