Vault/Consul Monitoring Guide

> ***Author:*** *Todd Radel*
> ***Version:*** *0.1*
> ***Date:*** *10 May 2018*

Table of Contents

Introduction

If you're going to rely on any enterprise application in production, there are certain things you expect: the ability to run backups, to fail over to a standby system in case the primary fails ... and the ability to monitor the application to make sure it's healthy, or alert you when it isn't.

This guide will show you how to monitor Vault and Consul. The main topics include:

- How to set up a monitoring stack (in this case: Telegraf, InfluxDB, and Grafana).
- How to configure Vault and Consul to send telemetry to a monitoring agent.
- Which metrics are important to monitor, and why.

Approaches to application monitoring

There are three main ways to monitor the health of an application:

1. **Time-series telemetry data.** This involves capturing metrics from the application, storing them in a special database designed for that purpose, and analyzing trends in the data over time.

   Examples: Grafana, CloudWatch, DataDog, Circonus.
2. **Log analytics.** This means capturing log files from the system and the application, extracting useful signals from the text, and then analyzing that data.

   Examples: Splunk, ELK, SumoLogic.
3. **Active health checks.** This involves active methods of connecting to the application and interacting with it to ensure it is responding properly.

   Examples: Nagios, Sensu, Keynote.

All of these methods have their place, but this guide focuses on the capture and analysis of time-series telemetry. Future versions of this guide might extend to the other areas described above.

Survey of available time-series monitoring solutions

Vault and Consul use the go-metrics library to export telemetry. Currently they support the following options:

- Circonus
- DataDog's DogStatsd
- Statsite
- Statsd

Note that DataDog's agent and Statsite are implementations of statsd, so the last 3 options are nearly the same thing.

Once the metrics reach your statsd-compatible agent, they need to be forwarded somewhere so they can be stored and displayed. There are many options. This page discusses a few of the most popular.

Grafana/Graphite

Graphite is an open-source tool for storing and graphing time-series data. It doesn't support dashboards or alerts, but Grafana can be layered on top of it.

TICK Stack

TICK is an acronym for Telegraf, InfluxDB, Chronograf, and Kapacitor. Together they provide a full solution for storing, displaying, and alerting on time-series data. It is available in both open-source and commercial versions from InfluxData.

- Telegraf provides a statsd-compatible host agent.
- InfluxDB is the time-series database.
- Chronograf is a dashboarding engine roughly similar to Grafana.
- Kapacitor provides alerting.

Amazon Cloudwatch

CloudWatch is Amazon's solution for monitoring AWS cloud resources. It handles both time-series data and log files. If you are running Vault and Consul in AWS, it can be an easy choice to make.

One limitation of CloudWatch is that time-series data is only available at a 1-minute granularity and only for 15 days. After that, the data is rolled up into 5-minute and one-hour buckets. For more details, see the CloudWatch FAQ.

Prometheus

Prometheus is a modern alternative to statsd-compatible daemons, using lightweight HTTP servers called "exporters" which are then scraped by a Prometheus server. Prometheus is increasingly popular in the containerized world.

Rather than the UDP-based "fire and forget" mechanism used by statsd, Prometheus relies on lightweight HTTP servers called "exporters" which are then scraped by a Prometheus server. There is a statsd exporter that you could use in place of Telegraf later in this guide.

DataDog

DataDog is a commercial SaaS solution. They provide a customized statsd agent, DogStatsd, that includes several vendor-specific extensions, mostly tagging and service check results.

If you use DataDog, you would use their DogStatsd in place of Telegraf later in this guide.

Other options

There are many other commercial and open-source choices. Configuring those is beyond the scope of this document.

Now that you know what options are available, we can reveal the Architecture chosen for this guide.

Monitoring solution used in this guide

I've chosen to use Telegraf, InfluxDB, and Grafana to demonstrate how to monitor Vault and Consul.

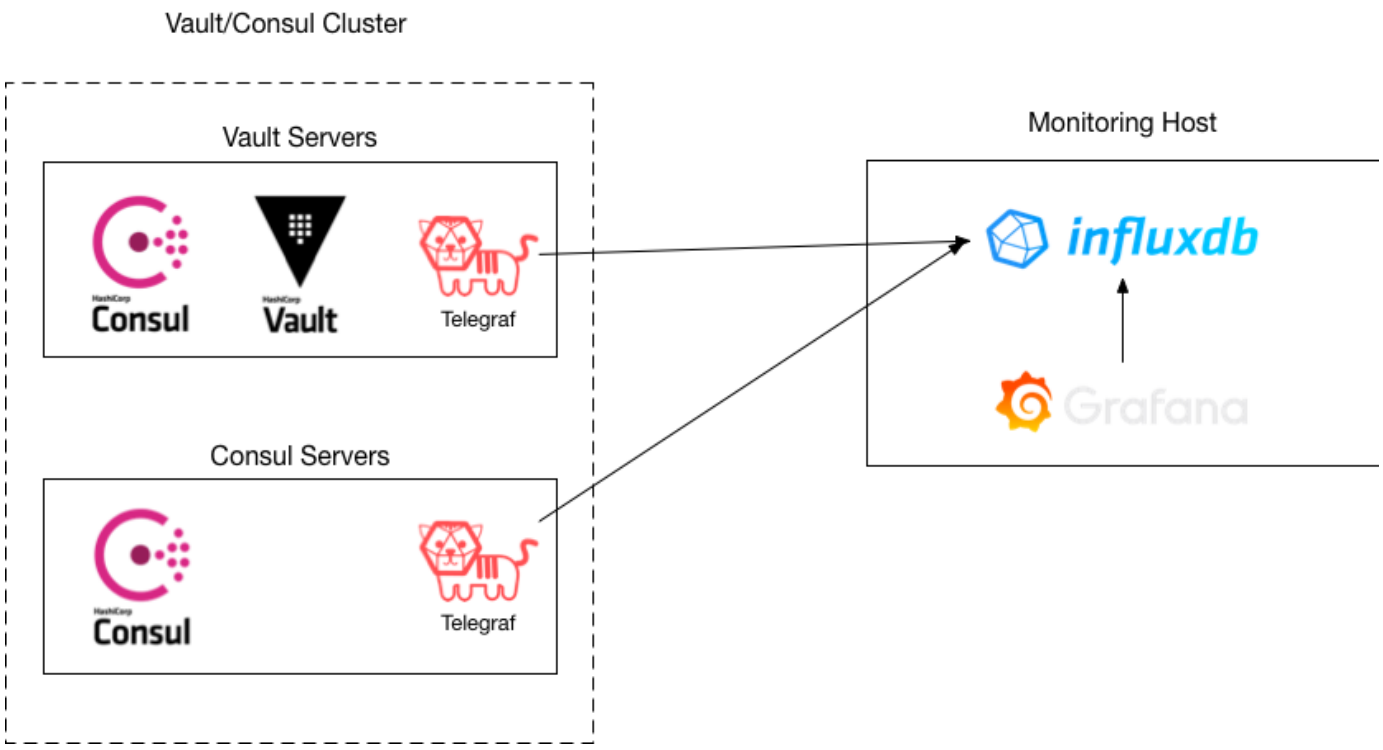
- Telegraf is a lightweight but powerful monitoring agent that is compatible with statsd.
- InfluxDB is a highly performant time-series database.
- Grafana takes the collected data and builds pretty dashboards (and does some basic alerting).

Why those tools in particular?

- They're open-source, popular, and well understood.
- They're compatible with the most popular flavors of Linux, and can even be run in containers if desired.
- They're easy to set up.

Since I'm using Telegraf and InfluxDB already, why not build the whole TICK stack and use Chronograf instead of Grafana? As of current writing, Chronograf's dashboards aren't quite as configurable or pretty as Grafana's, but I plan to keep an eye on the project as it evolves. Feel free to try it out yourself and see what you think.

The next few sections of this guide will explain how to set up a server with InfluxDB and Grafana, and how to install Telegraf agents on each of your Vault and Consul hosts. When you're done, it will look like this:



The Telegraf agent on each host listens for statsd packets on port 8125/udp, converts the data into JSON format, and sends it to InfluxDB. The dashboards we create in Grafana then query that data from InfluxDB and produce colorful charts that are easy to understand.

Sample project accompanying the guide

Accompanying this guide is a Github repository that sets up a simple stack for you to test out the concepts described in this guide. The stack includes:

- Three Vault nodes (`vault0`, `vault1`, and `vault2`).
- Three Consul nodes (`consul0`, `consul1`, and `consul2`).
- A Grafana/InfluxDB server (`statsbox`).

The project uses HashiCorp Vagrant to set up the entire stack on your laptop, using VirtualBox to host the VM's. You should have at least 16GB of RAM to run the stack.

Once the stack is running, you can unseal Vault, then log in to Grafana and import the sample dashboards. If you want to see how the dashboards look when Vault is under heavy load, I suggest using the Vault load test sister project to generate some traffic to Vault.

Monitoring Vault and Consul with Telegraf, InfluxDB, and Grafana

Setting up a server with InfluxDB and Grafana

Provisioning a new server

> ***Warning***
> *Do not install InfluxDB and Grafana on the same servers that are already running Vault or Consul. Set them up on a new server instead.*

Recommended hardware specs for running InfluxDB and Grafana depend on whether you have a single Vault/Consul cluster, or multiple clusters that will be sending metrics to InfluxDB:

	Single Cluster	Multiple Clusters
CPU cores	2 to 4	4 to 8
RAM	2 GB to 4 GB	8 GB
IOPS	300	1000
AWS instance type	t2.medium	t2.large or m4.large
GCP instance type	n1-standard-1	n1-standard-2

InfluxDB and Grafana can run on most flavors of Linux. Windows is not recommended.

Installing InfluxDB

Instructions can be found on the InfluxDB web site. Brief instructions for Ubuntu Server are reproduced here.

```
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/lsb-release
echo "deb https://repos.influxdata.com/${DISTRIB_ID,,} ${DISTRIB_CODENAME} stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list

sudo apt-get update && sudo apt-get install influxdb
sudo systemctl enable influxdb
sudo systemctl start influxdb
```

Installing Grafana

Grafana also has pretty comprehensive install guides available, but again we present simple instructions for installing on Ubuntu servers:

```
curl -sL https://packagecloud.io/gpg.key | sudo apt-key add -
echo "deb https://packagecloud.io/grafana/stable/debian/ jessie main" | sudo tee
/etc/apt/sources.list.d/grafana.list
apt-get update && apt-get -y install grafana

sudo apt-get update && sudo apt-get install grafana
systemctl enable grafana-server
systemctl restart grafana-server
```

Securing the installation

> By default, InfluxDB listens on port 8086 with no authentication. Once you have monitoring up and running as described in this guide, you will probably want to take steps to secure your InfluxDB installation:
>
> - Restricting access via firewall or AWS security groups
> - Configuring InfluxDB for HTTPS
> - Enabling authentication
>
> Don't forget to update `telegraf.conf` with the appropriate options too.

Installing and configuring Telegraf agents

Installing Telegraf

Installing Telegraf is straightforward on most Linux distributions with step-by-step instructions. On Ubuntu, for example:

```
# add the influxdata signing key
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -

# configure a package repo
source /etc/lsb-release
echo "deb https://repos.influxdata.com/${DISTRIB_ID,,} ${DISTRIB_CODENAME} stable" | sudo tee
/etc/apt/sources.list.d/influxdb.list

# install Telegraf and start the daemon
sudo apt-get update && sudo apt-get install telegraf
sudo systemctl enable telegraf
sudo systemctl start telegraf
```

Configuring Telegraf

Besides acting as a statsd agent, Telegraf can collect additional metrics of its own. Telegraf itself ships with a wide range of input plugins to collect data from lots of sources. We're going to enable some of the most common ones to monitor CPU, memory, disk I/O, networking, and process status.

The `telegraf.conf` file starts with global options:

```
[agent]
  interval = "10s"
  flush_interval = "10s"
  omit_hostname = false
```

We set the default collection interval to 10 seconds and ask Telegraf to include a `host` tag in each metric.

As mentioned above, Telegraf also allows you to set additional tags on the metrics that pass through it. In this case, we are adding tags for the server role and datacenter. We can then use these tags in Grafana to filter queries (for example, to create a dashboard showing only servers with the `consul-server` role, or only servers in the `us-east-1` datacenter).

```
[global_tags]
  role = "consul-server"
  datacenter = "us-east-1"
```

Next, we set up a statsd listener on UDP port 8125, with instructions to calculate percentile metrics and to parse DogStatsd-compatible tags, when they're sent:

```
[[inputs.statsd]]
  protocol = "udp"
  service_address = ":8125"
  delete_gauges = true
  delete_counters = true
  delete_sets = true
  delete_timings = true
  percentiles = [90]
  metric_separator = "_"
  parse_data_dog_tags = true
  allowed_pending_messages = 10000
  percentile_limit = 1000
```

The full reference to all the available statsd-related options in Telegraf is here.

Now we can configure inputs for things like CPU, memory, network I/O, and disk I/O. Most of them don't require any configuration, but make sure the `interfaces` list in `inputs.net` matches the interface names you see in `ifconfig`.

```
[[inputs.cpu]]
  percpu = true
  totalcpu = true
  collect_cpu_time = false

[[inputs.disk]]
  # mount_points = ["/"]
  # ignore_fs = ["tmpfs", "devtmpfs"]

[[inputs.diskio]]
  # devices = ["sda", "sdb"]
  # skip_serial_number = false

[[inputs.kernel]]
  # no configuration

[[inputs.linux_sysctl_fs]]
  # no configuration

[[inputs.mem]]
  # no configuration

[[inputs.net]]
  interfaces = ["enp0s*"]

[[inputs.netstat]]
  # no configuration

[[inputs.processes]]
  # no configuration

[[inputs.procstat]]
  pattern = "(consul|vault)"

[[inputs.swap]]
  # no configuration

[[inputs.system]]
  # no configuration
```

Another useful plugin is the procstat plugin, which reports metrics for processes you select:

```
[[inputs.procstat]]
  pattern = "(consul|vault)"
```

Finally, Telegraf even includes a plugin to monitor Consul agents.

```
[[inputs.consul]]
  address = "localhost:8500"
  scheme = "http"
```

Putting it all together, you can see complete `telegraf.conf` examples for Consul and Vaulthosts. While you're at it, you may as well set up monitoring on the InfluxDB/Grafana server too. Here's an example of how you could do that.

After you edit the `telegraf.conf`, don't forget to restart the Telegraf agent:

```
systemctl restart telegraf
```

Configuring Vault and Consul to send telemetry data to Telegraf

Configuring Consul agents

Asking Consul to send telemetry to Telegraf is as simple as adding a `telemetry` section to your agent configuration:

```
{
  "telemetry": {
    "dogstatsd_addr": "localhost:8125",
    "disable_hostname": true
  }
}
```

As you can see, we only need to specify two options. The `dogstatsd_addr` specifies the hostname and port of the statsd daemon.

Note that we specify DogStatsd format instead of plain statsd, which tells Consul to send tagswith each metric. Tags can be used by Grafana to filter data on your dashboards (for example, displaying only the data for which `role=consul-server`. Telegraf is compatible with the DogStatsd format and allows us to add our own tags too, as you'll see later.

The second option tells Consul not to insert the hostname in the names of the metrics it sends to statsd, since the hostnames will be sent as tags. Without this option, the single metric `consul.raft.apply` would become multiple metrics:

```
consul.server1.raft.apply
    consul.server2.raft.apply
    consul.server3.raft.apply
```

If you are using a different agent (e.g. Circonus, Statsite, or plain statsd), you can find the configuration reference here.

Configuring Vault

Similar to Consul, configuring Vault to send us telemetry is painless. Just add one stanza to your Vault config:

```
telemetry {
  dogstatsd_addr   = "localhost:8125"
  disable_hostname = true
}
```

The options are the same as they were for Consul. The full reference can be found here.

Setting up the database connection in Grafana

If everything is working, you should now be able to pull up the Grafana UI, configure the connection to InfluxDB, and start exploring. If you are using the sample project, the UI is forwarded from the Grafana server to http://localhost:3000. Log in as "admin" with password "admin".

Configuring the data source

Before we can create dashboards, we need to tell Grafana where to get data from. From the Grafana home screen, click "Create your first data source", or go to the sidebar menu and choose **Configuration** > **Data Sources** and then click "Add a data source".

Fill in the settings as follows:

- **Name**: any name you like.
- **Default**: checked.
- **Type**: InfluxDB.
- **HTTP**:
    - **URL**: http://localhost:8086
    - **Access**: direct
- **Auth**: leave all options unchecked.
- **Advanced HTTP Settings**: leave options at defaults.
- **InfluxDB Details**:
    - **Database**: telegraf
    - **User**: telegraf
    - **Password**: telegraf
- **Min time interval**: 10s (matches the Telegraf agent `interval` setting)

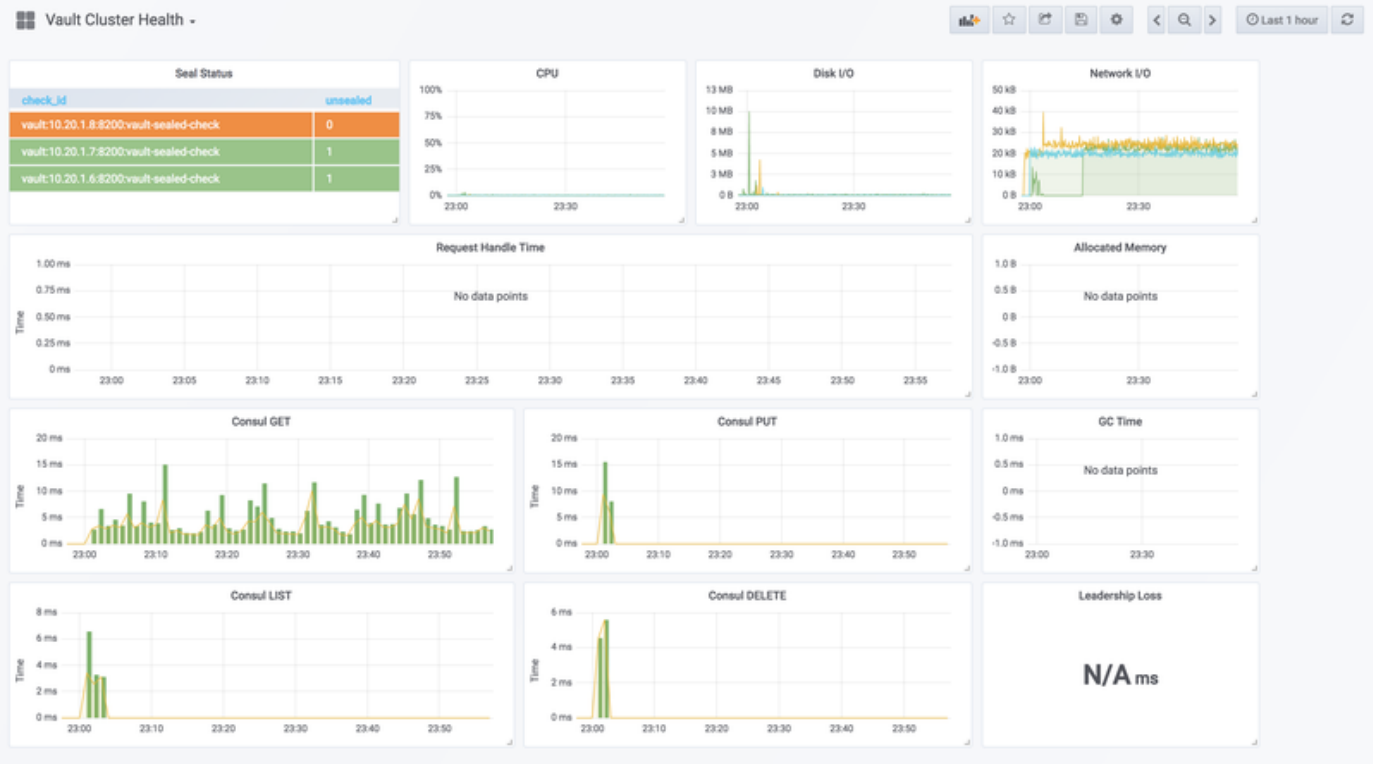
Make sure your screen looks like this screenshot, then click "Save & Test". You should see messages indicating that "Data source is working" and "Data source saved". If not, double check your entries.

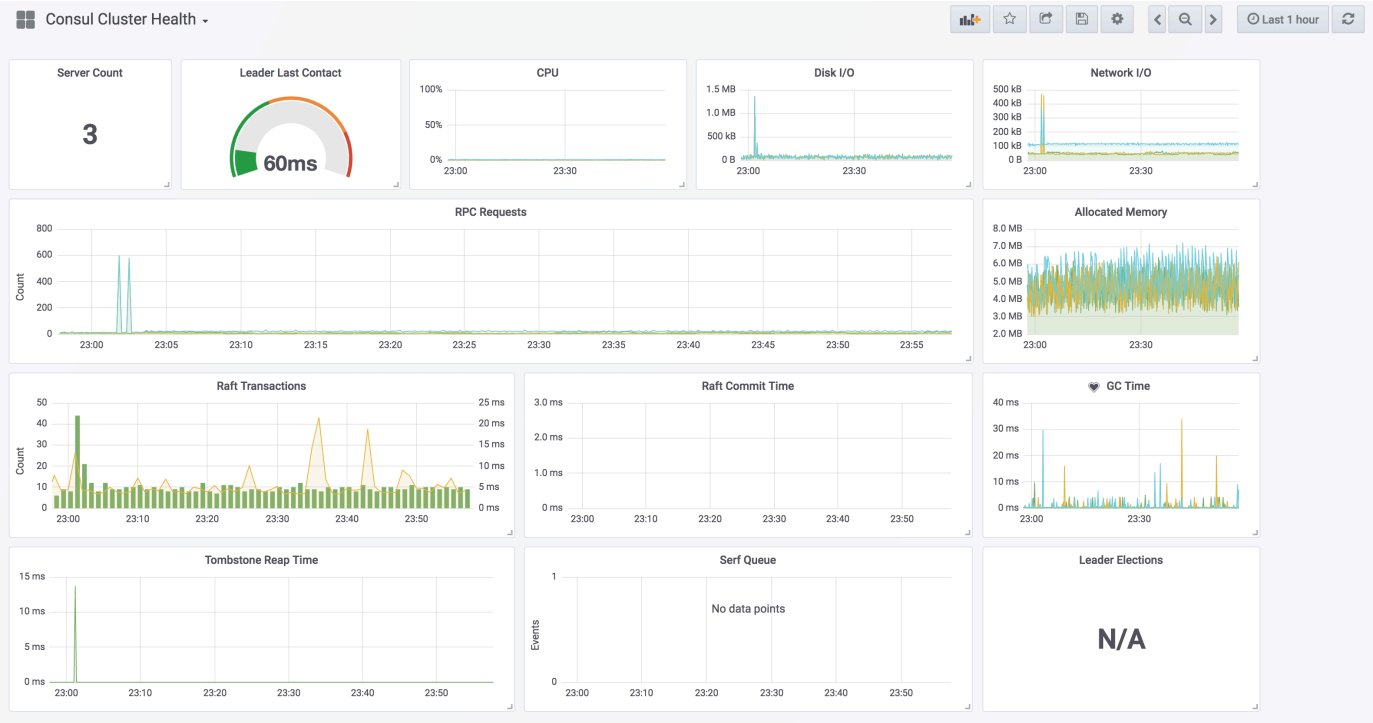
Importing the sample dashboards

We have provided sample dashboards for Vault and Consul here. To import them into your own Grafana server:

1. Download the dashboards to your machine.
2. Open the Grafana web interface. In the sample project, this would be at http://localhost:3000.
3. Click the "plus" icon to open the Create menu, and select Import (screenshot).
4. Click the green "Upload JSON File" button.
5. Browse to the first dashboard file and upload it.
6. From the dropdown list, select the data source you created earlier (screenshot).
7. Click "Import".

Now you should be able to see beautiful dashboards full of Vault and Consul metrics. If there are errors, you might need to customize each dashboard slightly. Depending on the versions of Vault and Consul you have, some metrics may not be available or may have been renamed.





Guide to important metrics

Vault and Consul provide reference documentation for all of the metrics they export in their telemetry (Vault hereand Consul here). Those pages list and describe every metric that is available, but they don't do a good job of explaining which ones are most important to monitor, why they're important, or what thresholds you might want to set for alerts.

In this section, we try to demystify the Vault and Consul telemetry a bit, and tell you what's most important to keep an eye on.

Consul server metrics

Transaction timing

Metric Name	Description
`consul.kvs.apply`	This measures the time it takes to complete an update to the KV store.
`consul.txn.apply`	This measures the time spent applying a transaction operation.
`consul.raft.apply`	This counts the number of Raft transactions occurring over the interval.
`consul.raft.commitTime`	This measures the time it takes to commit a new entry to the Raft log on the leader.

Why they're important: Taken together, these metrics indicate how long it takes to complete write operations in various parts of the Consul cluster. Generally these should all be fairly consistent and no more than a few milliseconds. Sudden changes in any of the timing values could be due to unexpected load on the Consul servers, or due to problems on the servers themselves.

What to look for: Deviations (in any of these metrics) of more than 50% from baseline over the previous hour.

Leadership changes

Metric Name	Description

`consul.raft.leader.lastContact`	Measures the time since the leader was last able to contact the follower nodes when checking its leader lease.
`consul.raft.state.candidate`	This increments whenever a Consul server starts an election.
`consul.raft.state.leader`	This increments whenever a Consul server becomes a leader.

Why they're important: Normally, your Consul cluster should have a stable leader. If there are frequent elections or leadership changes, it would likely indicate network issues between the Consul servers, or that the Consul servers themselves are unable to keep up with the load.

What to look for: If `candidate` > 0, or `leader` > 0, or `lastContact` greater than 200ms.

Autopilot

Metric Name	Description
`consul.autopilot.healthy`	This tracks the overall health of the local server cluster. If all servers are considered healthy by Autopilot, this will be set to 1. If any are unhealthy, this will be 0.

Why it's important: Obviously, you want your cluster to be healthy.

What to look for: Alert if `healthy` is 0.

Memory usage

Metric Name	Description
`consul.runtime.alloc_bytes`	This measures the number of bytes allocated by the Consul process.
`consul.runtime.sys_bytes`	This is the total number of bytes of memory obtained from the OS.
`mem.total`	Total amount of physical memory (RAM) available on the server.
`mem.used_percent`	Percentage of physical memory in use.
`swap.used_percent`	Percentage of swap space in use.

Why they're important: Consul keeps all of its data in memory. If Consul consumes all available memory, it will crash. You should also monitor total available RAM to make sure some RAM is available for other processes, and swap usage should remain at 0% for best performance.

What to look for: If `sys_bytes` exceeds 90% of `total_bytes`, if `mem.used_percent` is over 90%, or if `swap.used_percent` is greater than 0.

Garbage collection

Metric Name	Description
`consul.runtime.total_gc_pause_ns`	Number of nanoseconds consumed by stop-the-world garbage collection (GC) pauses since Consul started.

Why it's important: As mentioned above, GC pause is a "stop-the-world" event, meaning that all runtime threads are blocked until GC completes. Normally these pauses last only a few nanoseconds. But if memory usage is high, the Go runtime may GC so frequently that it starts to slow down Consul.

What to look for: Warning if `total_gc_pause_ns` exceeds 2 seconds/minute, critical if it exceeds 5 seconds/minute.

NOTE: `total_gc_pause_ns` is a cumulative counter, so in order to calculate rates (such as GC/minute), you will need to apply a function such as non_negative_difference.

File descriptors

Metric Name	Description
`linux_sysctl_fs.file-nr`	Number of file handles being used across all processes on the host.

`linux_sysctl_fs.file-max`	Total number of available file handles.

Why it's important: Practically anything Consul does -- receiving a connection from another host, sending data between servers, writing snapshots to disk -- requires a file descriptor handle. If Consul runs out of handles, it will stop accepting connections. See the Consul FAQ for more details.

By default, process and kernel limits are fairly conservative. You will want to increase these beyond the defaults.

What to look for: If `file-nr` exceeds 80% of `file-max`.

CPU usage

Metric Name	Description
`cpu.user_cpu`	Percentage of CPU being used by user processes (such as Vault or Consul).
`cpu.iowait_cpu`	Percentage of CPU time spent waiting for I/O tasks to complete.

Why they're important: Consul is not particularly demanding of CPU time, but a spike in CPU usage might indicate too many operations taking place at once, and `iowait_cpu` is critical -- it means Consul is waiting for data to be written to disk, a sign that Raft might be writing snapshots to disk too often.

What to look for: if `cpu.iowait_cpu` greater than 10%.

Network activity

Metric Name	Description
`net.bytes_recv`	Bytes received on each network interface.
`net.bytes_sent`	Bytes transmitted on each network interface.

Why they're important: A sudden spike in network traffic to Consul might be the result of a misconfigured Vault client causing too many requests.

What to look for: Sudden large changes to the `net` metrics (greater than 50% deviation from baseline).

NOTE: The `net` metrics are counters, so in order to calculate rates (such as bytes/second), you will need to apply a function such as non_negative_difference.

Disk activity

Metric Name	Description
`diskio.read_bytes`	Bytes read from each block device.
`diskio.write_bytes`	Bytes written to each block device.

Why they're important: Since Consul keeps everything in memory, there normally isn't much disk activity. If the Consul host is writing a lot of data to disk, it probably means that Consul is under heavy write load, and consequently is checkpointing Raft snapshots to disk frequently. It could also mean that debug/trace logging has accidentally been enabled in production, which can impact performance. Too much disk I/O can cause the rest of the system to slow down or become unavailable, as the kernel spends all its time waiting for I/O to complete.

What to look for: Sudden large changes to the `diskio` metrics (greater than 50% deviation from baseline, or more than 3 standard deviations from baseline).

NOTE: The `diskio` metrics are counters, so in order to calculate rates (such as bytes/second), you will need to apply a function such as non_negative_difference.

Vault server metrics

Request processing

Metric Name	Description
`vault.core.handle_request`	Duration of requests handled by the Vault core.

Why it's important: This is the key measure of Vault's response time.

What to look for: Changes to the `count` or `mean` fields that exceed 50% of baseline values, or more than 3 standard deviations above baseline.

Consul response time

Metric Name	Description
`vault.consul.get`	Count and duration of `GET` operations against the Consul storage backend.
`vault.consul.put`	Count and duration of `PUT` operations against the Consul storage backend.
`vault.consul.list`	Count and duration of `LIST` operations against the Consul storage backend.
`vault.consul.delete`	Count and duration of `DELETE` operations against the Consul storage backend.

Why they're important: These metrics indicate how long it takes for Consul to handle requests from Vault.

What to look for: Large deltas in the `count`, `upper`, or `90_percentile` fields.

Write-ahead log processing

Metric Name	Description
`vault.wal.persistWALs`	Amount of time required to persist the Vault write-ahead logs (WAL) to the Consul backend.
`vault.wal.flushReady`	Amount of time required to flush the Vault write-ahead logs (WAL) to the persist queue.

Why they're important: The Vault write-ahead logs (WALs) are used to replicate Vault between clusters. Surprisingly, the WAL's are kept even if replication is not currently enabled. The WAL is purged every few seconds by a garbage collector. But if Vault is under heavy load, the WAL may start to grow, putting pressure on Consul.

What to look for: If `flushReady` is over 500ms, or if `persistWALs` is over 1000ms.

Leadership changes

Metric Name	Description
`vault.core.leadership_lost`	Total duration of cluster leadership losses in a highly-available cluster.

Why it's important: There should not be a leadership change unless the leader crashes or becomes otherwise unavailable. While the other servers elect a leader, Vault is unable to process any requests.

What to monitor: Any value greater than 0 should cause an alert condition.

Seal status

Metric Name	Description
`consul_health_checks[check_name="Vault Sealed Status"].passing`	Value of 1 indicates Vault is unsealed; 0 means sealed.

Why they're important: By default, Vault is sealed on startup, so if this value changes to 0 during the day, Vault has restarted for some reason. And until it's unsealed, it won't answer requests from clients.

What to look for: A value of 0 being reported by any host.

NOTE: This metric is actually reported by the Consul plugin to Telegraf.

Memory usage

Metric Name	Description
`vault.runtime.alloc_bytes`	This measures the number of bytes allocated by the Vault process.
`vault.runtime.sys_bytes`	This is the total number of bytes of memory obtained from the OS.
`mem.total_bytes`	Total amount of physical memory (RAM) available on the server.
`mem.used_percent`	Percentage of physical memory in use.
`swap.used_percent`	Percentage of swap space in use.

Why they're important: Vault doesn't need as much memory as Consul, but if it does run out, it too will crash. You should also monitor total available RAM to make sure some RAM is available for other processes, and swap usage should remain at 0% for best performance.

What to look for: If `sys_bytes` exceeds 90% of `total_bytes`, if `mem.used_percent` is over 90%, or if `swap.used_percent` is greater than 0.

Garbage collection

Metric Name	Description
`vault.runtime.total_gc_pause_ns`	Number of nanoseconds consumed by stop-the-world garbage collection (GC) pauses since Vault started.

Why it's important: As mentioned above, GC pause is a "stop-the-world" event, meaning that all runtime threads are blocked until GC completes. Normally these pauses last only a few nanoseconds. But if memory usage is high, the Go runtime may GC so frequently that it starts to slow down Vault.

What to look for: Warning if `total_gc_pause_ns` exceeds 2 seconds/minute, critical if it exceeds 5 seconds/minute.

Additional notes: `total_gc_pause_ns` is a cumulative counter, so in order to calculate rates (such as GC/minute), you will need to apply a function such as non_negative_difference.

File descriptors

Metric Name	Description
`linux_sysctl_fs.file-nr`	Number of file handles being used across all processes on the host.
`linux_sysctl_fs.file-max`	Total number of available file handles.

Why it's important: Practically anything Vault does -- receiving a connection from another host, sending data between servers, writing snapshots to disk -- requires a file descriptor handle. If Vault runs out of handles, it will stop accepting connections.

By default, process and kernel limits are fairly conservative. You will want to increase these beyond the defaults.

What to look for: If `file-nr` exceeds 80% of `file-max`.

CPU usage

Metric Name	Description
`cpu.user_cpu`	Percentage of CPU being used by user processes (such as Vault or Consul).
`cpu.iowait_cpu`	Percentage of CPU time spent waiting for I/O tasks to complete.

Why they're important: Encryption can place a heavy demand on CPU. If the CPU is too busy, Vault may have trouble keeping up with the incoming request load. You may also want to monitor each CPU individually to make sure requests are evenly balanced across all CPUs.

What to look for: if `cpu.iowait_cpu` greater than 10%.

Network activity

Metric Name	Description

`net.bytes_recv`	Bytes received on each network interface.
`net.bytes_sent`	Bytes transmitted on each network interface.

Why they're important: A sudden spike in network traffic to Vault might be the result of a misconfigured client causing too many requests, or additional load you didn't plan for.

What to look for: Sudden large changes to the `net` metrics (greater than 50% deviation from baseline).

NOTE: The `net` metrics are counters, so in order to calculate rates (such as bytes/second), you will need to apply a function such as non_negative_difference.

Disk activity

Metric Name	Description
`diskio.read_bytes`	Bytes read from each block device.
`diskio.write_bytes`	Bytes written to each block device.

Why they're important: Vault generally doesn't require too much disk I/O, so a sudden change in disk activity could mean that debug/trace logging has accidentally been enabled in production, which can impact performance. Too much disk I/O can cause the rest of the system to slow down or become unavailable, as the kernel spends all its time waiting for I/O to complete.

What to look for: Sudden large changes to the `diskio` metrics (greater than 50% deviation from baseline, or more than 3 standard deviations from baseline).

NOTE: The `diskio` metrics are counters, so in order to calculate rates (such as bytes/second), you will need to apply a function such as non_negative_difference.

Future topics

This guide will continue to be updated periodically. Future topics to be added may include:

- Monitoring with Prometheus, CloudWatch, or TICK
- Log monitoring with Splunk, CloudWatch, or the ELK Stack

Contributing to the guide

If you have topics you'd like to see added, please create an issue in the GitHub repo. If you have material to contribute to the guide, please send us a pull request. And thank you in advance for helping make this guide even better!

Todd Radel
May 2018