A Beginner's Guide to collectd

splunk>
turn data into doing™

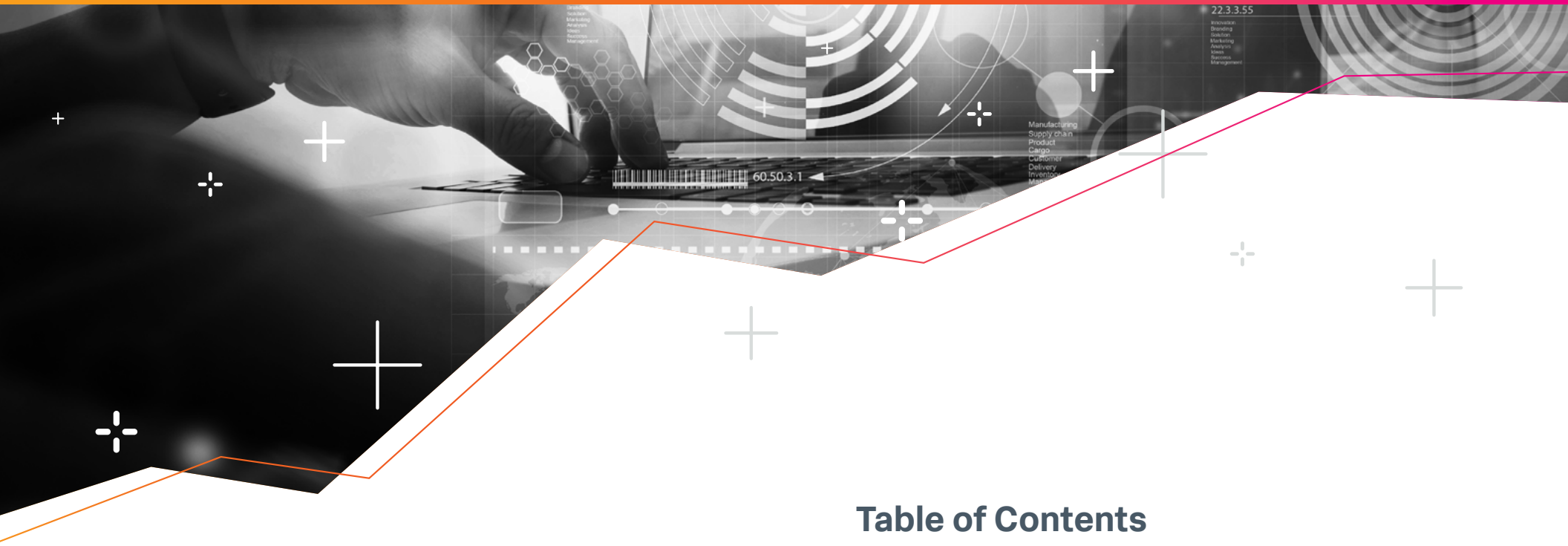


Table of Contents

collectd in a Nutshell

What is collectd?

collectd is a daemon — a process that runs in the system's background — that collects system and application performance metrics.

Metrics are reports on how a specific aspect of a system is performing at a given moment. They are snapshots of how a certain function of the system — like storage usage, CPU temperature or network status — delivered at regular intervals, and consist of:

- A timestamp
- A metric name
- A measurement (a data point)
- Dimensions (that often describe the host, kind of instance or other attributes to filter or sort metrics on)

Metrics are particularly useful for monitoring. Like a heart monitor that regularly checks a patient's pulse, metrics provide insight into trends or problems that affect the performance and availability of infrastructure and applications. But a heart monitor won't tell you why a patient has a sudden issue with their heart rate — you need other means to quickly identify the cause of the problem and stabilize the patient.

Metrics are typically generated by a daemon (or process) that runs on a server (OS), container or application. Each data measurement is delivered over the network to a server that indexes and analyzes that information. It's important to note that collectd is just performing the collection and transmission of the metrics. It does not analyze or present the metrics on its own. You need a tool, such as Splunk®, or open source options like Graphite to collect and visualize the metrics.

collectd involves an agent running on a server that's configured to measure specific attributes and transmit that information to a defined destination. collectd is an extensible measurement engine, so you can collect a wide range of data. Currently, collectd is most often used for core infrastructure monitoring insights, such as getting insight on the workload, memory usage, I/O and storage of servers and other infrastructure components. You can learn more about collectd by visiting **http://collectd.org**.

Getting collectd
There are many places to get collectd, and its license is free.

The most direct place to get collectd is to download the collectd package, which you can find at **https://github.com/collectd/collectd/**.

```
git clone git://github.com/collectd/collectd.git
```

From there, you'll need to compile collectd by using the build.sh script that's part of the distribution.

There are some compiled packages, which will save you a step. These are distribution dependent. For a comprehensive list, go to **https://collectd.org/download.shtml** to see what distributions are available that you can install with `pkg_add, apt-get`, etc.

Once compiled, you can add **plugins** (more on plugins shortly), which perform specific measurements. You'll also modify `collectd.conf,` which defines plugins, the frequency to take measurements, and how to transmit the collected data.

You can also acquire collectd as binaries (if you're ready to use them), or you can get the source code, modify it if needed and then compile the source code.

What collectd measures
collectd can measure a wide range of metrics.

By default, it enables the CPU, interface, load and memory plugins — perfect for getting fundamental metrics from your servers.

There are also approximately 100 plugins that you can add to collectd. Many of the plugins focus on networking topics (network protocols from netlink to SNMP), while others relate to popular devices from UPS to sensor). Popular application stack software components are included. You'll find MySQL, Redis and many other popular plugins as well. A complete list of plugins can be found at: **https://collectd.org/wiki/index.php/Table_of_Plugins.**

Benefits of collectd

It's Free!
You don't get charged per agent, and you can push collectd to as many systems as you see fit. You can also use whatever plugins you like.

It's lightweight.
collectd has a small footprint from a memory and disk standpoint. Its modular architecture allows the agent to be the minimal size required to do the job.

You're less dependent on software vendors.
collectd only collects and transmits metrics. You can direct that information to any number of tools to consume that data.

It provides flexibility in what you collect.
collectd enables you to specifically declare what metrics you want to capture, and at what frequency you want to capture them. This means you can scale back your metrics collection to the data that's right for your observability needs. Or you may only want to collect data every five minutes on systems that don't support mission-critical SLAs, but accelerate the polling interval to once per minute for systems where you have "4 or 5 9's" availability requirements.

There are some challenges to working with collectd, however. collectd does require you to actively manage how you distribute agents (although a CI/CD tool like **Puppet** can be of great help).

How Do Splunk and collectd Work Together?

Splunk is focused on indexing and analyzing both metrics and machine data.

While collecting data is a necessary step in effective indexing and analysis of data, Splunk's focus is on providing analytics around infrastructure data. Meanwhile, collectd is focused on making collecting and transmitting data to an analytics tool easy, lightweight and free.

Using collectd and the Splunk platform together enables you to analyze large amounts of infrastructure data — easily and securely.

Today, "analyzing" data means more than creating dashboards, alerting on thresholds and creating visualizations on demand. As your environment and observability needs grow, Splunk software helps you identify trends, issues and relationships you may not be able to observe with just the naked eye.

Even more important, metrics, like those generated by collectd, only tell half the story. They're useful for monitoring, where we discover the answers to "What's going on?" They don't help us rapidly troubleshoot problems and answer "Why is this problem happening?" By correlating logs and metrics in time-series sequence, you can add the context that uniquely comes from logs

Is collectd cloud-native? It's complicated.

According to the **Cloud Native Computing Foundation (CNCF)**, cloud native systems have the following properties:

1. **They're packaged in containers** — **hub.docker.com** lists well over **20 hub repos that contains multiple flavor or another of collectd**. With more than five million pulls across several repos, this likely passes the litmus test that collectd cannot only be a containerized package but it is widely adopted by the container community.

2. **They're dynamically managed** — Dynamically writing a config file may be the best and only way possible (as far as we can tell) to manage the state and behavior of collectd. Some open source projects make use of an environment variable as a method to dynamically configure the state of collectd to provide runtime configurations. But the lack of interfaces can make it challenging to configure and manage collectd's state. Maintenance and operational costs when running it in a containerized environment can certainly be improved.

3. **They're microservices oriented** — There's no doubt that collectd is a very loosely coupled collection daemon. Its plugin architecture allows any combination of plugins to run independently of each other.

It might be a challenge for collectd to meet all the criteria identified by the CNCF, but there's always room for improvement, and of course, more open source contributions to this great project: **https://github.com/collectd/collectd**. Who's up for a challenge?

Extensibility
It's all about performance, reliability and flexibility.

collectd is one of the most battle-tested system monitoring agents out there, and has an official catalog of more than 130 plugins.

These plugins do everything from pulling vital system metrics (like CPU, memory and disk usage) to technology-specific metrics for popular technologies like NetApp, MySQL, Apache Webserver and many more. Some of the plugins specifically extend collectd's data forwarding and collection abilities, allowing it to serve as a central collection point for multiple collectd agents running in an environment.

The strength of collectd, as with any open source agent, is its extensibility. For example, all of the plugins in the official collectd documentation are written in C, which enables collectd to run with minimal impact to the host system and provides extensibility without the overhead of additional dependencies. This isn't always the case. The **collectd plugin for Python**, for example, requires specific versions of Python to be available on the host system. For many of the plugins, however, a dependent set of libraries isn't required.

C, however, has its own issues. For many developers, it's a language that isn't widely used or discussed — outside of college or tangential conversations about an approach to a specific architectural problem. Fortunately, there are other paths to extending the core capabilities of collectd. The **collectd GO plugin** enables developers to write powerful extensions of collectd in GO, a language similar to C in that is compiled and can be used without any external dependencies. For many system admins and SREs attempting to ship software and services on optimized infrastructure, having to add Python to something like CoreOS is a non-starter. Extending collectd with C and GO is a perfect fit for those looking for the golden prize of infrastructure optimization.

If you're looking for speed and ease of use, collectd can easily be extended with the Python or Perl plugins. Just be aware of what you're getting into. Both Perl and Python:

- Compile at runtime, which means they don't get packaged up neatly without dependencies
- May not perform with the same reliability as a plugin written in C or GO
- Can add additional overhead to your systems

This might not be a problem in development, but production engineers and admins rarely like to add additional overhead if it can be avoided. That being said, Python and Perl are fantastic for rapid development and for their ease-of-use and straightforward syntax. Both also have massive volumes of community contributed libraries, which make them excellent options when speed and flexibility outweigh overall resource utilization for your systems.

Analyzing collectd Data

As with any monitoring solution, the tool is only as good as the data, and the data is only as good as the use case.

Starting with a "collect it all and figure it out later" strategy is a great approach to classic big data problems where data spans multiple environments, sources and formats.

The first step is to know what you are responsible for. If you're our beloved system admin for example, questions you would want to consider include:

- Are my hosts performing as expected?
- Is the CPU over- or under-utilized?
- Is there network traffic going to servers as expected?
- Are my hard drives filling up?

collectd has both out-of-the-box plugins for these data sources and many ways to ensure you can provide the data that's relevant to your environment. In this section you'll learn how to configure collectd to answer the questions above and discover what to look for in a visualization tool.

Example collectd.conf file global settings:
Here's how to set up global variables and important plugins for system monitoring in collectd.

```
##############################################################
# Global settings for collectd                               #
##############################################################


Hostname "my.host"

Interval 60
```

About these settings:

The most basic collectd.configuration requires no global settings. In this example, we're using *Hostname* to set the *Hostname* that will be sent with each measurement from the collectd daemon. Note that when *Hostname* isn't set, collectd will determine the *Hostname* using *gethostname(2)* system call.

The *Interval* setting will set the interval in seconds at which the collectd plugins are called. This sets the granularity of the measurements. In the example above, measurements are collected every 60 seconds. Note that while the *Interval* setting doesn't need to be set, it defaults to once every 24 hours.

A note on Interval:
This value shouldn't change often, as the interval of measurement collection should line up with your technical and service-level requirements. In other words, if you don't need to know the CPU utilization every 60 seconds, then there's no need to measure it every 60 seconds. Keep in mind that collectd plugins all run on the same interval.

In the past, three to five minutes was a perfectly acceptable interval for monitoring. But as workloads and deployments become more automated and complex (and expectations of service delivery increase), it's not uncommon to see collection intervals at 60 seconds, with some organizations collecting data as often as every second. Sixty seconds is a good standard approach to collection intervals, and any tools you use to collect and use metric data should be able to provide this interval of collection and analysis.

Loading collectd plugins:
The syntax for loading plugins is straightforward: to keep the conf files easy to read and update, keep a separate section for loading the plugins and for plugin settings.

```
##############################################################
# LoadPlugin section                                         #
##############################################################


LoadPlugin syslog
LoadPlugin logfile
LoadPlugin cpu
LoadPlugin interface
```

These lines load the necessary plugins for:

- Setting the logging level and location (Syslog, Logfile)
- Collecting CPU metrics (CPU)
- Collecting network metrics (Interface)

Configuring Settings for Plugins

This section goes into detail about how to set up the plugins we loaded in the previous section. Each plugin has their own unique settings, when setting up collectd for the first time we like to set the logging to info to make sure these settings are correct. A syntax error could cause the collectd daemon to fail. Once you are confident with your settings the log level can be changed to something less verbose like warn.

```
##############################################################

# Plugin configuration                                       #

##############################################################

<Plugin logfile>
          LogLevel info
          File "/etc/collectd/collectd.log"
          Timestamp true
          PrintSeverity true
</Plugin>

<Plugin syslog>
          LogLevel info
</Plugin>
```

This part of the conf file is just there to make sure you're getting vital information about collectd and catching any potential errors in our log files. You can either use these logs in the command line to troubleshoot agent deployment problems or you can use a log analysis tool, like **the Splunk platform**, to do more advanced analytics. (We have a recommended tool for that, but you choose what's best for you.)

Logfile: Here you're setting the log level and adjusting some key settings like location (File), whether to add a timestamp and whether to add the severity. You can change the Level to debug for a more verbose log or a WARN or ERROR for critical errors only.

Syslog: This tells collectd what level to write to our systemlog (if you are sending data to a collectd.log, this may be unnecessary). It's important to complete this step if using Syslog for troubleshooting and investigation a host.

CPU: This plugin will collect measurement by CPU (disable this setting to collect aggregate CPU measures across all CPU cores). ReportByState breaks down CPU metrics into the actual CPU state — system, user, idle, etc. If this isn't set, the metrics will be for Idle and an aggregated Active state only.

```
<Plugin cpu>
          ReportByState true
          ValuesPercentage true
</Plugin>
```

You may not be looking at per CPU values, but this can be very important if your goal is to ensure that all logical CPUs are being used by workloads passed to the system. For most monitoring use cases, getting utilization in aggregate is enough.

The chart below (Figure 1) from an example build server shows that this machine's CPU remains largely idle. This means that you have likely overprovisioned this instance and that you can do almost the same workload with significantly less hardware.

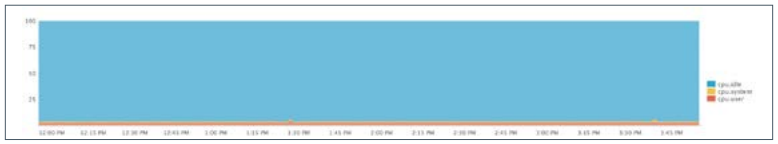


Figure 1: The utilization for inactive and active CPU states for our host.

Both images show that this system is usually idle. In fact it's only using approximately 1 percent of its total CPU capacity. But this is a mission-critical server, so it's unlikely it will be replaced by something with smaller hardware. Once you go top shelf, it's hard to go back.

Interface: In this instance, there aren't any specific settings for interface. This means you can collect network data from all interfaces.

```
<Plugin interface>
</Plugin>
```

Network traffic is one of the first things to check whenever there's a drop in overall resource utilization within a server. Losing connectivity in a local network is a common issue in a datacenter. If there is near 100 percent idle CPU usage, it's possible that this server isn't receiving traffic from any source. The interface plugin helps figure that out.

This configuration uses the most basic setup to capture network data for all interfaces. The **collectd interface plugin** allows you to explicitly ignore certain interfaces when needed (although not necessary in this scenario).

Note: There are many reasons to omit an interface. Loopback interfaces don't have associated physical hardware, so measuring their performance isn't valuable. You can eliminate monitored interfaces statically by explicitly listing the interfaces to ignore or eliminate them dynamically using regular expression.

When analyzing this data, you should aggregate all the interfaces together since you're looking for basic activity across all interfaces (we have also done a little math to make sure we're looking at the traffic in megabytes per second). The collectd interface plugin collects Net I/O as Octets. Octets, as those who are musically inclined know, are groups of 8 musicians, a small ensemble. Octets in the world of computing are roughly equivalent to 8 bits, or a byte.

Note that many interface metrics come out as counters, that is, the values do not decrease unless there is a reset to 0. The formula we're using to put this data in megabytes per second is interface.octets.rx*(0.000001/60)

In the figure below, y ou can see there's steady network traffic to this server.

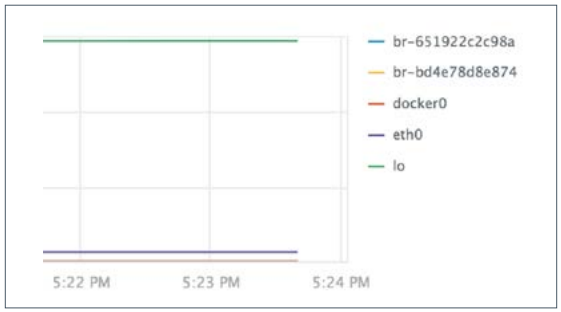


Figure 2: This search shows Inbound network data for this host. We've converted the octets into megabytes.

Knowing that there is a pretty standard stream of network data to this server, you can set an alert for if the inbound network data ever falls below this line for a sustained period of time. That could provide just enough warning that you're about to start seeing critical failures on this server and getting complaints from my developers.

DF: The **DF plugin** provides vital stats about your filesystem, including the total free space, the amount used and the amount reserved. You can filter measurement by MountPoint, FSType or Devicename. As with the interface plugin, you can statically set this filter or use regex for dynamic filtering. The example setup below shows root partitions only getting the percentages for the measurements. You can also see the absolute values, but for this use case we are interested in knowing when the local filesystem is about to be full. A full disk (or near full disk) is often the root cause for application and server failure.

```
<Plugin df>
          MountPoint "/"
          ReportByDevice true
          ValuesAbsolute false
          ValuesPercentage true
          IgnoreSelected false
</Plugin>
```

As with CPU, this example uses our analysis tool to combine all the metrics into a single graph:

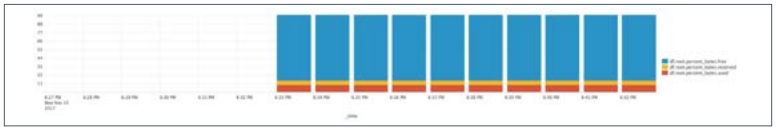


Figure 3: This chart shows the amount of data free, used and reserved over time for our root file system.

As with the other metrics analyzed so far, this server is in a solid state. In this scenario, a few additional alerts would be useful. The first would be for when disk free reaches less than or equal to 50 percent, the next would be a warning when this server has less than 25 percent free, and the last would be a critical warning when it reaches less than 20 percent free. These **alerts** will help to prevent a disk outage.

As an example, when working with one group of system admins, we had a critical failure with a mission-critical application. After several hours and many rounds of blame, someone found that one of the primary batch processes that moved data from local DBs to a central DB had failed because the VM it was hosted on had run out of space. There were no alerts for VM disk space, and as a result our central reporting had fallen behind by almost a day.

This is just a basic introduction to critical system metrics that every infrastructure team should look out for. Even in the world of containers and microservices, underneath there's a vast infrastructure of CPUs, network interfaces and hard disks. Understanding the vital signs of your servers is the first step in a lasting and healthy infrastructure.

Understanding collectd's naming schema and serialization

Collectd uses identifiers to categorize metrics into a **naming schema.** The identifier consists of five fields, two of which are optional:

- host
- plugin
- plugin instance (optional)
- type
- type instance (optional)

In addition, **data sources** are used to categorize metrics:

- dsnames
- dstypes

Splunk metrics combines these fields into a metric name that follows a dotted string notation. This makes it easy to discern a hierarchy of metrics. User interfaces in Splunk can make use of the dotted string notation to group metrics that belong together and enable users to browse this hierarchy.

These are example values from a metrics payload that the **write_http plugin** sends to Splunk:

- **host:** server-028
- **plugin:** protocols
- **plugin instance:** IpExt
- **type:** protocol_counter
- **type instance:** InOctets
- **dsnames:** ["value"]

Splunk uses the following pattern to create a metric name:

```
plugin.type.type_instance.dsnames[X]
```

For the payload above, this would result in the following metric name:

```
protocols.protocol_counter.InOctets.value
```

Dimensions

In the payload above, there are two fields that aren't used to string together a metric name from the collectd identifiers — the host and plugin_instance fields. If the Splunk platform is setup correctly (see the "**Using Splunk With collectd**" section), the content of the host field in collectd will be used as the value for the internal host field in Splunk as well.

This field can then be used to filter for hosts or compare a metric across hosts using a *groupby* operation when querying a metric. Likewise, *plugin_instance* will be ingested as a custom dimension, which allows for the same filter and *groupby* operations. Examples for *plugin_instance* are IDs of CPU cores ("0", "15", etc.) or the identifiers of network interfaces ("IpExt", etc.).

Using Splunk With collectd

Getting collectd data into Splunk

collectd supports a wide range of write plugins that could be used to get metrics into Splunk. Some of them are specific to a product or service (such as TSDB, Kafka, MongoDB), while others support more generic technologies. The supported way to ingest collectd metrics in Splunk is to use the write_http plugin, which sends metrics in a standardized JSON payload to any HTTP endpoint via a POST request. On the receiving end, the HTTP Event Collector (HEC) endpoint in Splunk can be easily configured to receive and parse these payloads, so they can be ingested into a metrics index.

HTTP Event Collector (HEC):

Prior to setting up the write_http plugin, the HEC needs to be enabled to receive data. Configure this data input before setting up collectd because you'll need to use data input details for the collectd configuration.

1. In Splunk Web, click **Settings > Data Inputs**.
2. Under **Local Inputs**, click **HTTP Event Collector**.
3. Verify that the HEC is enabled.
   a. Click **Global Settings**.
   b. For all tokens, click **Enabled** if this button is not already selected.
   c. Note the value for **HTTP Port Number**, which you'll need to configure collectd.
   d. Click **Save**.
4. Configure an HEC token for sending data by clicking **New Token**.
5. On the **Select Source** page, for **Name**, enter a token name, for example "collectd token."
6. Leave the other options blank or unselected.
7. Click **Next**.
8. On the **Input Settings** page, for **Source** type, click **Select**.
9. Click **Select Source Type**, then select **Metrics > collectd_http**.
10. Next to **Default Index**, select your metrics index, or click **Create** a new index to create one.
11. If you choose to create an index, in the **New Index** dialog box.
    a. Enter an **Index Name**. User-defined index names must consist of only numbers, lowercase letters, underscores, and hyphens. Index names cannot begin with an underscore or hyphen.
    b. For **Index Data Type**, click **Metrics**.
    c. Configure additional index properties as needed.
12. Click **Save**.Click **Review**, and then click **Submit**.
13. Copy the **Token Value** that is displayed, which you'll need to configure collectd.

To test your data input, you can send collectd events directly to your metrics index using the /collector/raw REST API endpoint, which accepts data in the collectd JSON format. Your metrics index is assigned to an HEC data input that has its unique HEC token, and "collectd_http" as its source type.

The following example shows a curl command that sends a collectd event to the index associated with your HEC token:

```
curl -k
https://localhost:8088/services/collector/raw?
sourcetype=collectd_http
\
-H "Authorization: Splunk <HEC_token>"
\
-d
'[{"values":[164.9196798931339196],"dstypes":
["derive"],"dsnames":["value"],"time":
1505356687.894,"interval":10.000,"host":"
collectd","plugin":"protocols","
plugin_instance":"IpExt","type":"
protocol_counter","type_instance":"InOctets"}]'
```

You can verify the HEC data input is working by running a search using mcatalog to list all metric names, with the time range set to "All Time", for example:

```
| mcatalog values(metric_name) WHERE index=
<your_metrics_index> AND metric_name=protocols.
protocol_counter.InOctets.value
```

The write_http plugin:

After setting up the HEC, the **write_http plugin** needs to be enabled and configured in the collectd configuration file (collectd.conf). It requires the following fields from your HEC data input:

Field Name	Description	Syntax	Example
URL	URL to which the values are submitted. This URL includes your Splunk host machine (IP address, host name or load balancer name) and the HTTP port number.	URL "https://<Splunk _host>:<HTTP_port>/ services/collector/ raw"	URL "https://10.66.104. 127:8088/services collector/raw"
Header	An HTTP header to add to the request.	Header "Authorization: Splunk <HEC_token>"	Header "Authorization: Splunk b0221cd8- c4b4-465a-9a3c- 273e3a75aa29"
Format	The format of the data.	Format "JSON"	Format "JSON"

Cloud and collectd

Why collectd in the Cloud?

Cloud adoption is growing at a staggering rate year over year. Server refresh projects are driving organizations to consider moving to the cloud to lower infrastructure and human capital costs for adapting to the growing demand from the business.

With a hybrid or multicloud approach, how do you create a unified view of your servers across on-prem and cloud vendors? Adopting a standard cloud native approach to collecting system statistics can help you monitor and troubleshoot performance and risk. And collecting and correlating these metrics with logs from these servers in one solution will reduce your time to identification and help you focus on what matters most.

AWS

Getting Started: Install and Configure collectd on EC2 server

1. Launch a new Amazon Linux AMI (any other AMIs/OS's can be used, but note that the install instructions may be different).
2. SSH to the AMI to install and configure collectd:
   ssh -i <ssh-key> ec2-user@<aws-externalip>
3. Install collectd: **http://docs.splunk.com/Documentation/Splunk/7.0.0/Metrics/GetMetricsInCollectd**
   a. On an Amazon Linux AMI, the following command to install AWS CloudWatch's version is recommended.

```
sudo yum -y install collectd
```

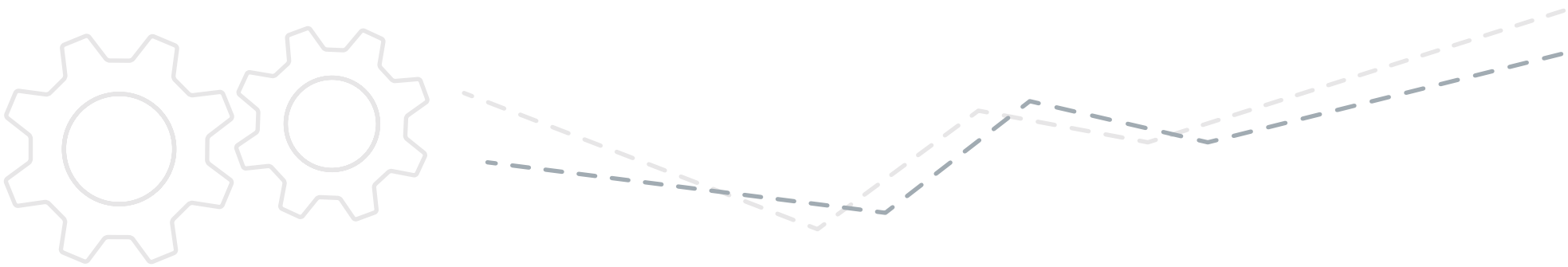
   b. Running the following command will confirm that you have successfully installed collectd and that you are running a version 5.6 and above required to send metrics to Splunk: *collectd -h*

4. Install write_http and disk plugins:

```
yum -y install collectd collectd-write_http.x86_64
yum -y install collectd collectd-disk.x86_64
```

5. Configure collectd: adding system level plugins to test integration
   a. Create a new configuration file.
   b. Here's a sample collectd file to get started: https://s3.amazonaws.com/splunk-collectd-beginnerguide/collectd.beginnerguide.conf. (For more information about available plugins and their configuration, refer to **https://collectd.org/wiki/index.php/Table_of_Plugins**.)
   c. Configure the <splunk_host> , <hec_port> and <hec_token> in the configurations provided with your own environment values.
   d. Save your changes.
   e. Overwrite local collectd.conf file with the new configured version.

```
curl -sL -o collectd.beginnerguide.conf https://
s3.amazonaws.com/splunk-collectd-beginnerguide/
collectd.beginnerguide.conf
vi collectd.beginnerguide.conf
```



f. Using vi, update the content of the file (e.g., configure splunk_host, hec_port and hec_token) and save your updates.

g. Overwrite the local collectd.conf with the new updates:

```
sudo cp ./collectd.beginnerguide.conf /etc/collectd.conf
```

6. Restart collectd: sudo /etc/init.d/collectd restart

For more detailed configurations please refer to: **http://docs.splunk.com/Documentation/Splunk/7.0.0/Metrics/GetMetricsInCollectd**.

Analyze Your Metrics

1. Open **Search and Reporting** in the Splunk Web UI.

2. Query for the list of metrics by name being reported: http://localhost:8000/en-US/app/search/search?q=%7C%20mcatalog%20values(metric_name)&display.page.search.mode=smart&dispatch.sample_ratio=1&earliest=0&latest=&sid=1509734051.54&display.general.type=statistics

a. List metrics names collected: | mcatalog values(metric_name)

3. Use mstats to create aggregations as needed (For more information about mstats refer to **http://docs.splunk.com/Documentation/Splunk/7.0.0/SearchReference/Mstats**.)

a. Average CPU idle over specified time window: | mstats avg (_value) WHERE metric_name=cpu.percent.idle.value

b. Average CPU idle over specified time window with a specified span: | mstats avg(_value) WHERE metric_name=cpu.percent.idle.value span=5mins

c. Average CPU idle over specified time window with a specified span split by host: | mstats avg(_value) WHERE metric_name=cpu.percent.idle.value span=5mins by host

d. Same as above displayed in timechart by host: | mstats avg(_value) prestats=true WHERE metric_name=”cpu.percent.idle.value” span=5mins BY “host” | timechart avg(_value) as “Avg” agg=max limit=20 useother=false span=5mins BY “host”

Troubleshooting:

collectd won’t start? Look at the errors logs (e.g., less /var/log/collectd.log). You likely need to install a missing plugin.

Google Cloud Platform (GCP)

Getting Started: Install and Configure collectd on GCE VM Instance

1. Create a new VM instance, using default Debian GNU/Linux 9 (stretch) as the Boot disk.
2. SSH to the AMI to install and configure collectd, gcloud compute ssh [INSTANCE_NAME].
3. Run the following commands. (Instructions will differ depending on the OS you've selected. For more instructions, go to: https://collectd.org/wiki/index.php/First_steps.)
   a. sudo apt-get update
   b. apt-get install --force-yes --assume-yes collectd
4. Configure collectd: adding system level plugins to test integration.
   a. Create a new configuration file: vi collectd.conf.
   b. Add the content of this file into collectd, **collectd.beginnerguide.conf** (For more information about available plugins and their configuration, go to: https://collectd.org/wiki/index.php/Table_of_Plugins.)
   c. Configure the <splunk_host>, <hec_port> and <hec_token> in the configurations provided with your own unique values.
   d. Save your changes.
   e. Overwrite local collectd.conf file, cp ./collectd.conf /etc/collectd/collectd.conf.
5. Start collectd: sudo /etc/init.d/collectd restart.

For more detailed configurations, please refer to: http://docs.splunk.com/Documentation/Splunk/7.0.0/Metrics/GetMetricsInCollectd.

Analyzing collectd Data Across Cloud Vendors

Comparing servers from any cloud providers can be quick and easy. **The Analytics for Linux Splunk App** provides a great overview of your system statistics for your servers. You can also use search and reporting or build your your own dashboard to analyze your results.

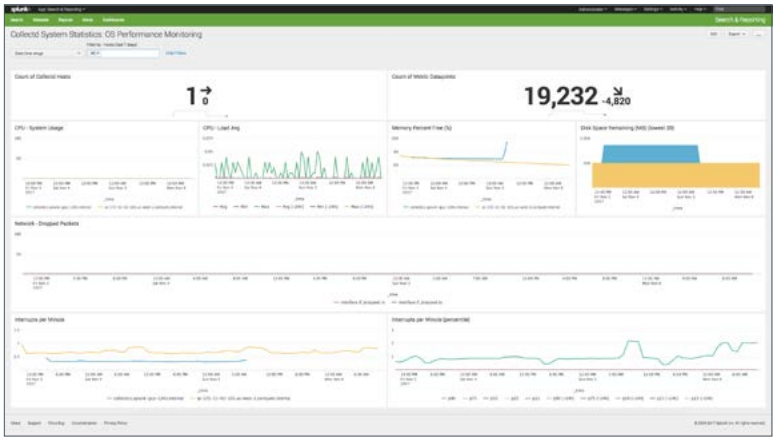


Figure 4: collectd system stats overview.

Use the following collectd dashboard to get started:

1. Open **Search and Reporting** in the Splunk Web UI, http://localhost:8000/en-US/app/search/search
2. Create a new dashboard
   a. Click **Dashboards**.
   b. Click **Create New Dashboard**.
   c. Set the **Title** (e.g., collectd).
   d. Click **Create Dashboard**.
   e. Replace the content of the XML with the following: db_collectd_perfmon-optimized.xml

```
curl -sL -o db_collectd_beginnerguide.xml https://
s3.amazonaws.com/splunk-collectd-beginnerguide/db_
collectd_beginnerguide.xml
```

   f. Save your updates.

Some of the sample queries:

- Count of hosts over time: |mstats count WHERE metric_name=* by host span=auto | timechart count as “Count of hosts” cont=false\
- CPU % System over time: | mstats avg(_value) WHERE metric_name=cpu.percent.system.value span=auto by host | timechart first(avg(_value)) as “avg” by host

More Info

Collectd.org
This site provides a list of resources, from binaries to examples for getting, configuring and using collectd

Splunk Insights for Infrastructure
Want to start collecting and analyzing metrics in your infrastructure in a matter of minutes? Splunk Insights for Infrastructure lets you distribute and collect data from collectd and statsd, and correlate it with metrics from your systems as well. You don't need to be a current Splunk Enterprise user, and you can use it for free!

Metrics for Splunk
Splunk Enterprise provides many features, including the ability to collect and analyze metrics, and correlate it with logs. This resource introduces Splunk users to the new capabilities in Splunk Enterprise and gives you ideas on how to get started.

Splunk Communities
Splunk has a network of thousands of passionate customers and advocates who welcome new ideas and questions. Ask questions and share your knowledge with others!

Getting Started.

splunk.com/insights-for-infrastructure

20-20901-Splunk-A Beginner's Guide to collectd-EB-107

splunk> turn data into doing™