



appgate

# Appgate SDP

## Monitoring the SDP system (using Grafana)

**Type:** Technical guide

**Date:** September 2022

**Applies to:** Appgate SDP v6.0.1 and newer



# Table of Contents

<b>INTRODUCTION</b>	<b>2</b>
<b>MONITORING SDP</b>	<b>2</b>
<b>MONITORING TOOLS</b>	<b>3</b>
<b>SNMP AND PROMETHEUS</b>	<b>4</b>
SNMP	4
PROMETHEUS	4
<b>MONITORING GOALS</b>	<b>5</b>
<b>REAL LIFE MONITORING</b>	<b>6</b>
ENABLING PROMETHEUS	6
AVAILABLE METRICS	6
THE PROMETHEUS/GRAFANA SET-UP	7
GRAFANA DASHBOARDS	8
<b>CONCLUSION</b>	<b>19</b>
<b>MORE INFORMATION</b>	<b>19</b>

## Introduction

This paper takes you through the features that come with an Appgate SDP system that are used for monitoring appliances. From v6.0.1 onwards there is a rich set of metrics provided and this paper explains which ones matter most and the best way to use them to monitor the wellbeing of your Collective.

It uses real-life examples from Prometheus and Grafana to explain how Appgate has implemented this on a test Collective that is in use every day. The examples explored refer to Controllers, Gateways and Portals.

## Monitoring SDP

Let us start by defining some goals for what we should be monitoring. We need to ensure that.....

- the access **infrastructure is operating as intended.**
- access related appliance **functions are performing to the required specification.**
- normal **operation does not impinge on any physical limits.**

These three goals should help inform us about which metrics matter, as each-and-every Collective will be slightly different in the way it has been deployed.





Appgate has supported countless Customers with many different deployment models, and each time we are asked to help with a problem, we ask the question; how could that have been anticipated/avoided? Often the answer lies in metrics - “if only the customer had been monitoring xxx then that could have been avoided”. So, in this paper we hope to share some real insights into some of the things that really matter.

Customers are unlikely to have a detailed understanding of the inner working of the Appgate SDP system, without this insight the prospect of trying to find a metric that is worth monitoring is quite a daunting undertaking. The problem of finding the right metric was exacerbated by the fact that in v5.5 and earlier the metrics were somewhat disorganized, there were more metrics than were needed for day-to-day monitoring and even then, there were one or two useful metrics missing. Many customers will therefore have pushed monitoring into the ‘done it’ pile (monitoring RAM and disk is good enough) or the ‘too hard’ pile (don’t really know where to start).

Monitoring has been a current area of focus for Appgate and in v6.0.1 they have been reimaged to make them fit for enterprise usage. In this paper, we hope to take the customer insight we have gathered, the knowledge of the inner working of Appgate SDP and combine these with the new metrics which are now available; and present this for customers in a way which will help move monitoring off the ‘too hard’ pile and on to the ‘must do’ pile – so please read on.

## Monitoring tools

It is worth explaining what is available in Appgate SDP today. There is some health-check monitoring provided by the Appliance Health Status display:

Home > Appliances Health Status									
Appliances Health Status					Total Appliances Health Status: 25 Items per page: 30				
Name	Site	Status	Functions	Sessions	CPU	Memory	Network out/in	Disk	Version
got - purple1	Gothenburg	warning	gw	14	1%	27%	0.38 Mbps/0.38 Mbps	49%	6.0.1-30125-release (c)
aws - purple40	AWS EU West 2	warning	ctr gw	21	1%	40%	82.3 Kbps/23.6 Kbps	24%	6.0.1-30125-release
lhr - purple80	London	warning	gw	13	0%	27%	9.32 Kbps/19.5 Kbps	21%	6.0.1-30125-release
aws - purple60	AWS SA East 1	warning	gw	2	0%	40%	1.22 Kbps/0.34 Kbps	62%	6.0.1-30125-release (c)

This provides some very basic information which only really serves the requirement *normal operation does not impinge on any physical limits*. Even then you should be aware that the values shown are just a snapshot in time – so the memory may show 51% now, but what was it 1 second earlier? This one small example of why time-based metrics provide much more insight.

The status column indicates if there is a problem and there is some additional information when you click on an appliance. This just about touches on the requirement *functions are performing to the required specification*, but in reality, is only showing the tip of the iceberg - so should not be considered as adequate for production.

Many organizations chose to ‘monitor’ the audit logs, and because they have set some alerts, think they are monitoring the Appgate SDP system adequately. Audit logs are just that – a trail of what has happened (in the past). Sure, you can make assumptions that because logs are appearing then the appliance must be awake and doing work; but we have seen examples such as a pair of HA Controllers, both of which are streaming logs but only one of which is performing any user sign-in operations!

The audit logs are designed to meet compliance requirements providing an accurate and complete record of all the access actions performed by the Appgate SDP system. From these it would be quite possible to identify a user who is being given access to a resource wrongly. Audit logs are designed to gather a great deal of data about one thing, they are not intended to measure functionality, well-being, resource consumption.

SNMP and Prometheus are designed to gather a little bit of data about very many different things to help you understand the state and trajectory of an appliance. It is important to implement one of these tools as well as monitoring the audit logs as part of the roll-out of an Appgate SDP Collective. Having said that, you will find the audit logs do appear in SNMP and Prometheus in the form of *audit\_event*; these are not the audit logs themselves but counters of audit log events.

Status for: got - purple1

Appliance

warning  
CA certificate is expiring. You must renew it before 2022-10-28.  
cz-logs: Connected to:  
wss://35.157.235.196:443  
/fw/TCP/9320

Gateway

healthy

Upgrade Status

idle

CPU

1%

Memory

27%

Disk

49%

Network

RX speed: 0.38 Mbps  
TX speed: 0.38 Mbps  
Busiest NIC: eth0  
Dropped inbound packets: 1  
Dropped outbound packets: 0  
IP addresses on eth0: 172.17.20.30,  
fd00:ffff:b:20::1

Customization

ip6\_off

Close

The proper way to monitor the Appgate SDP Collective – is by integrating with external monitoring tools such as SNMP or Prometheus. When you configure a new appliance, the Miscellaneous tab provides the option to configure an SNMP server or a Prometheus exporter, either of which can be used to feed metrics to external systems which can then provide detailed monitoring services.

The Appgate SDP system generates metrics for SNMP and Prometheus in a unified way internally. Effectively when a metric is collected it is programmatically formatted for both types of output, so even though we will be focused on Prometheus in this paper, there will (almost) always be an exactly matching metric available in SNMP.

## SNMP and Prometheus

### *SNMP*

Simple Network Management Protocol [SNMP] is an application layer protocol which is part of the Transmission Control Protocol / Internet Protocol (TCP/IP) protocol suite originating in the 1980s. It was designed specifically to manage and monitor network elements.

#### **How SNMP works**

The SNMP server in Appgate SDP exposes metrics in the form of variables organized in a management information base [MIB], which describes the metrics relevant to the SDP system. These metrics can then be remotely queried by monitoring applications. The MIB can be downloaded from the Settings>Utilities page in the admin UI.

### *Prometheus*

Prometheus is an open-source project managed by the Cloud Native Computing Foundation (CNCF). It was designed to collect metrics about your application and infrastructure.

#### **How Prometheus works**

A Prometheus library has been built into the Appgate SDP appliance. This exposes an HTTP endpoint from where Prometheus can scrape metrics. These scraped metrics are then stored, and rules applied to aggregate or generate time series from the data. Grafana is the default means of visualizing this data.



## Monitoring Goals

Before we dive in, let's take a moment to understand each of the three goals in the context of a deployed Appgate SDP Collective.

### Infrastructure is operating as intended

The Appgate SDP Collective can be a complex beast spread across many locations and networks. It can include multiple redundant instances – which because of the core design principal can operate with a high degree of autonomy. A Site with only three operational Gateways out of five will look fine until maybe you run out of available capacity!

Key here is to ensure that the whole system is operational in the way you expected. If you have multiple Controllers, are they all active? If you have a primary and a backup system – are they both available? If you are exporting audit logs – are they actually sending or just filling the disk?

It may not be possible to answer all your infrastructure questions from Appgate SDP metrics alone, but there is a surprising amount you can do. For the rest you may need to gather additional information from other related systems to get a complete picture of the infrastructure.

### Functions are performing to the required specification

Functions refer to the 6 different functions that you can enable on an Appliance: Controller, Gateway, Connector, Portal, LogForwarder and LogServer.

It is important to show some metrics that confirm the function is doing what is intended. If something is meant to be forwarding logs – then how many is it forwarding? You might have designed for a maximum of 1000 log records per minute – but what is the commissioned system actually doing?

As well as getting a sense of well-being from seeing the function doing its job, the metrics can inform us about how well the job is being done. You might have 2 Controllers and see the Ax-H3 appliance can sign-in user in an average of 200mS, whereas the reserve machine (which is a VM) takes 600mS for the same thing. This sort of (historical) information can be very useful for planning capacity or making hardware investments.

Given most systems will be operating in a HA configuration then clear graphical metrics which show appliances are in good balance with each other is another important consideration.

When thinking about a function's 'ability to perform' think about things which might prevent this happening. DoS is an obvious one to consider – which metrics that can be used to suggest a DoS attack is beginning on a specific appliance?

### Operation does not impinge on any physical limits

As we said earlier, without a clear understanding of the inner working of the Appgate SDP system it can be hard to know what to monitor. Many customers have opted for the obvious metrics they always use on all their systems – such as CPU, RAM and Disk. It does no harm measuring this low hanging fruit, but the most probable result will be to reveal; either design features such as the use of Java which allocates (a lot of) heap space but does not usually give it back because heap re-sizing is expensive operation; or design faults such as a file which is not being cleaned up correctly consuming an increasing amount of disk space.

Controllers for instance should not normally consume any significant RAM to perform their function. If it is, then there is likely to be some backlog forming such as long running scripts. It can operate at near 100% CPU but will get quite unhappy when it runs short of disk space!

Portal supports only a stated maximum number of users (Client sessions), so monitoring the number of active users is very important; but measuring RAM might be misleading as the product is designed to work within the available RAM and will always use a minimum of 25% of the RAM.

Do not forget about limits imposed by others. The system relies on things like DNS, The DNS servers or any firewalls between the Appgate SDP appliances and the DNS server might have some sort of per second limit. Exceeding this might result in requests being dropped and a user's access being denied. So, monitoring DNS requests/sec might be very relevant in some deployments.

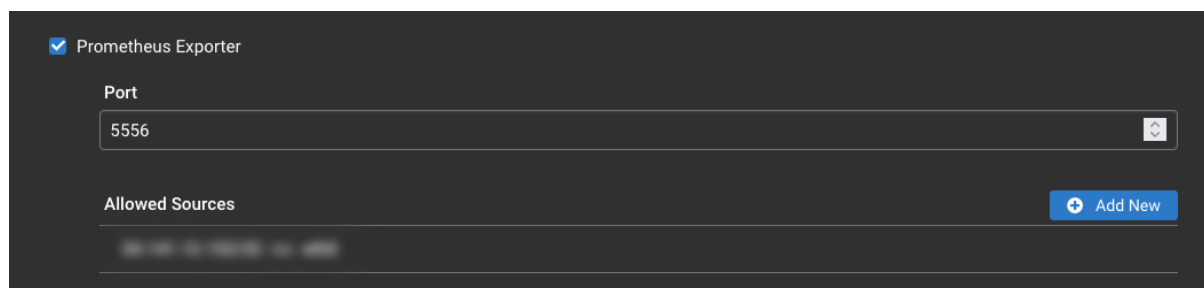


## Real life Monitoring

For everything that follows, we are looking at a real-life Collective with 25 appliances. A number of these appliances have been set up to use the Prometheus exporter. From this we drive a Grafana dashboard designed to monitor the health of the Collective.

### Enabling Prometheus

This is as easy as checking the box, and then filling in the IP address of your Prometheus server as an 'Allowed Sources'. This restricts access to only these known IP addresses.

A screenshot of a configuration interface for the Prometheus Exporter. At the top, there is a checkbox labeled 'Prometheus Exporter' which is checked. Below this, there is a 'Port' field with a text input containing '5556' and a dropdown arrow on the right. Underneath the port field is an 'Allowed Sources' section with a text input area and a blue button labeled '+ Add New' on the right.

### Available Metrics

The metrics available are the same as you can find in the SNMP MIB which we publish in Settings>Utilities. An SNMP MIB entry might look like this:

```
apnImageSize OBJECT-TYPE
    SYNTAX CounterBasedGauge64
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "The size of this appliance image partition in bytes"
    DEFVAL { 0 }
    ::= { sdpApn 3 }
```

The equivalent metric in Prometheus would look like this:

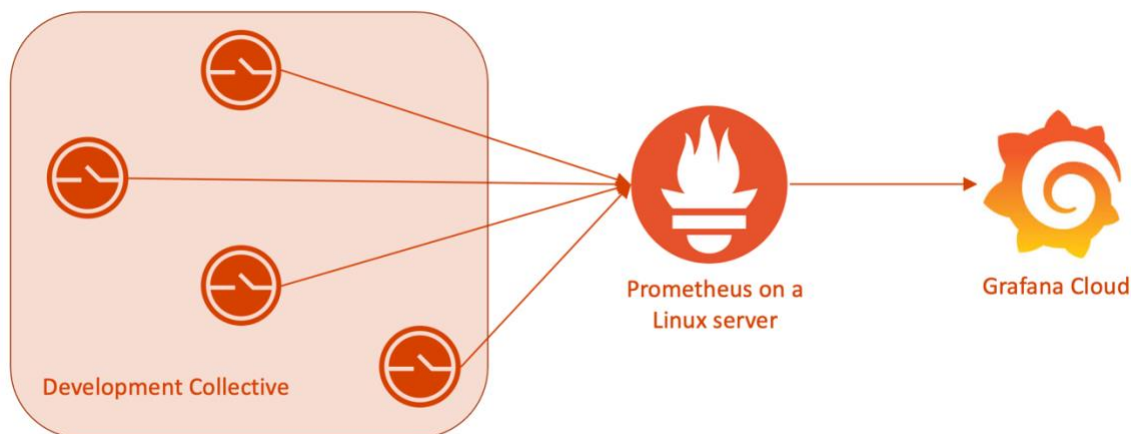
```
apn_image_size
```

The description field is carried across to Prometheus unchanged. You can usually browse the available metrics in Prometheus (or Grafana) and the description will be shown so downloading the SNMP MIB is not required when using Prometheus.



## The Prometheus/Grafana set-up

In our case we have a simple set up which looks like this:



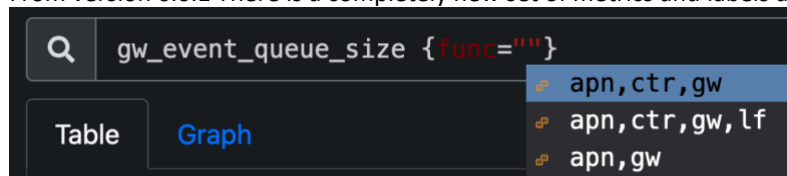
The way Prometheus works is it will scrape all available metrics from the defined targets every 15 seconds. These are all stored in a time-series database. The defined targets are set by a YAML file which the Prometheus server reads.

The individual entries in the YAML file look like this:

```
- job_name: 'purple80'
  static_configs:
    - targets: ['194.237.252.18:9090']
      labels:
        gateway: 'y'
```

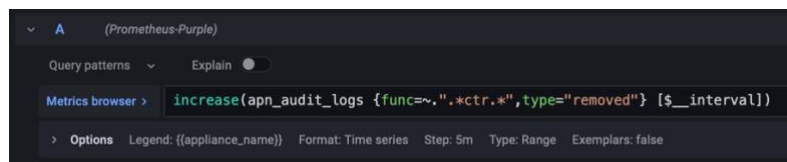
Previous versions of Appgate SDP (v5.5 and earlier) have a working but somewhat unstructured implementation of Prometheus. The metrics themselves were hard to understand and did not use labels. In the above example you will see we had to add an extra label – *gateway* on the Prometheus server to make the metrics easier to use once they hit Grafana.

From version 6.0.1 There is a completely new set of metrics and labels are now included automatically.

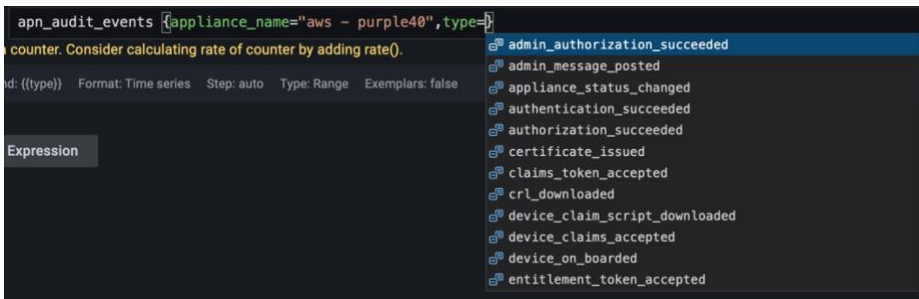


There is now, for instance, a *func* label which includes a comma separated list of the functions enabled on each appliance. This is very valuable when designing your panels in Grafana because they allow easy selection of data sources.

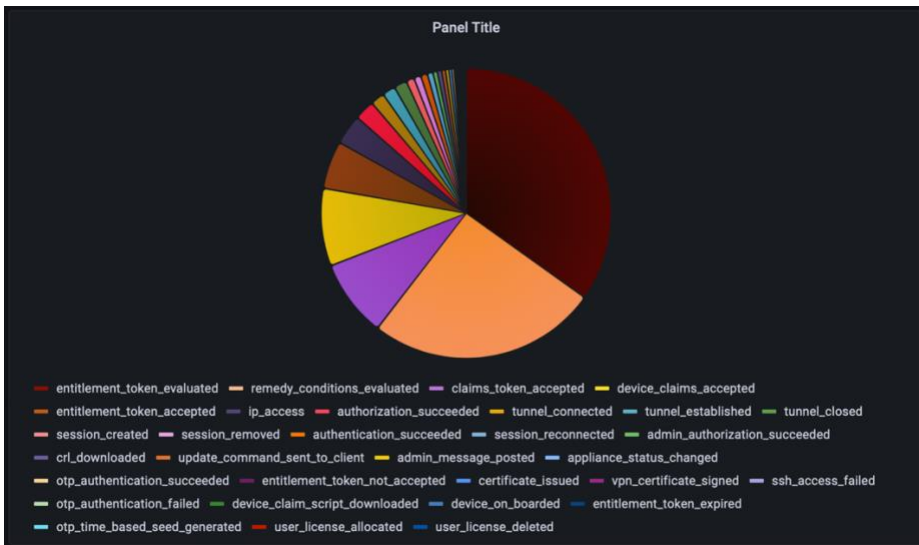
When doing say a Controller dashboard, the metrics can be filtered with a regular expression to get only the memory metrics for Controllers. In this example the appliance's audit logs for just Controllers.



Labels also simplify the display of data. Once you have filtered the timeseries using all but one of the labels then the remaining vector will now be used in the panel automatically.



If in this example if we do not specify the *type* then a pie chart will display all the *types* broken down by number.



This is not a very useful metric in itself but is a good example of how the use of labels simplify the ability to configure panels with minimal input.

## Grafana dashboards

In this paper we explore 3 example dashboards – one each for Controllers, Gateways and Portal. For each, we describe the panel/metrics we have used. We will go through the rationale for their inclusion, the specific panel type used and the metric(s) query we used.

You might want to use less, more or different panels but hopefully, these examples provide sufficient ideas for you to adapt them to meet your own needs.

Many of the metrics are counters (value always increases) so we quite often have to derive increases or rates in order to display meaningful data.

Consider the time intervals that are used for each panel to make the information meaningful.

Also think about what you want to see from what you are measuring. Metrics should be designed to either show what matters now, or should be designed to look very different over time (trends). An auto-scaled graph might look the same every day as the scale keeps adapting, masking the fact the values are creeping up towards some physical limit!





## Controller Health dashboard

Below are panel descriptions that relate to the SDP Controller example dashboard we have made available in GitHub.

This example dashboard covers all three Controllers deployed in the 'Purple' Collective. The exact panels you select for your dashboard are likely to be very similar as a Controller's function is quite universal.

Distribution of authorizations is important as it checks that all the Controllers are performing their function. Having set up DNS round-robin the 3 Controllers should be sharing the load fairly equally. The pie chart makes this very visible but does not show the actual load the Controllers are under.

PIE CHART uses:

```
ctr_client_authorization
```



The same health status as you see in the admin UI is available in Prometheus, so why not include this as the first element in the panel? These are triggered by very many different checks that are performed by either come from the specific function selected for the appliance (upper line) or that come from the hosting appliance itself (lower line).

STAT uses:

```
apn_function_status {name="controller"}
```

```
apn_status
```

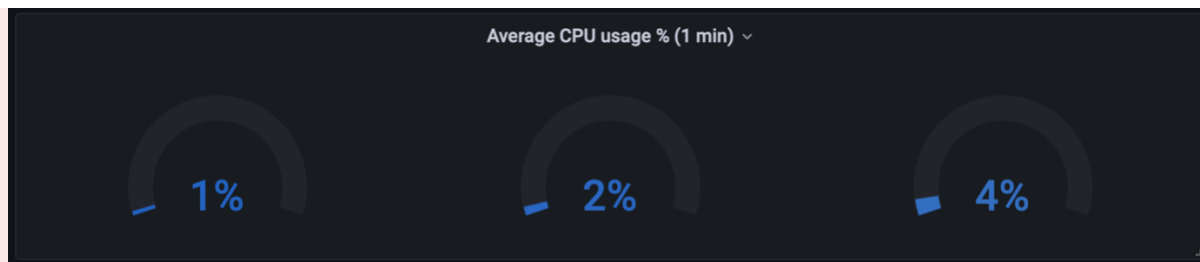


*Logs being successfully exported* benefits from the use of RELP which is a guaranteed delivery protocol. Logs are added to the export queue and removed only when an ack is received back from the receiving appliance. This metric is there to ensure a good implementation with working two-way communications, and that there is no bottleneck between the appliance and the log destination.

Additionally, this metric also shows that the appliances are working correctly (generating an expected amount of logs).

STAT uses:

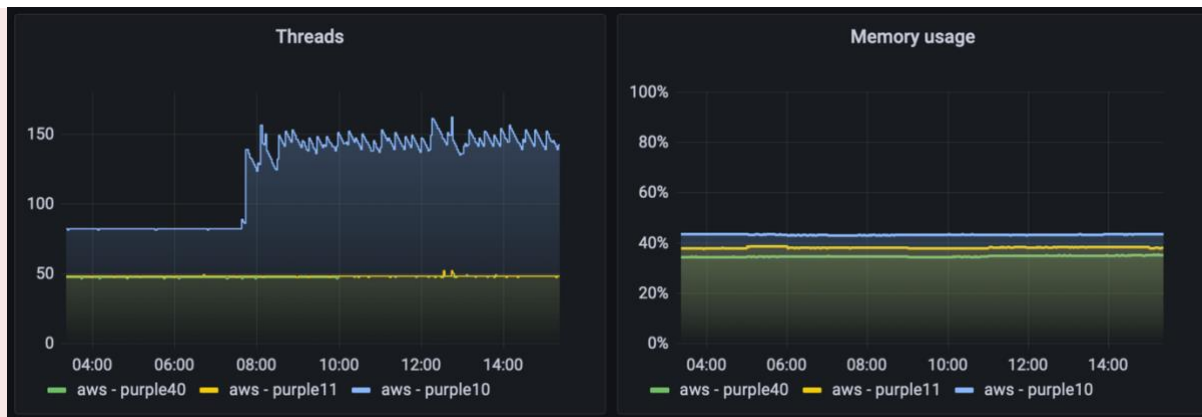
```
apn_audit_logs {type="removed"}
```



*Average CPU usage* for most Controllers is unlikely to be an issue. Only when there is some form of unexpected event might there be a temporary issue. An example could be if someone deleted the Policy for 100s of headless Clients. They would all then be trying to sign in repeatedly - which would effectively look like a DoS attack on the Controllers.

GAUGE uses:

```
apn_cpu_usage_percent
```

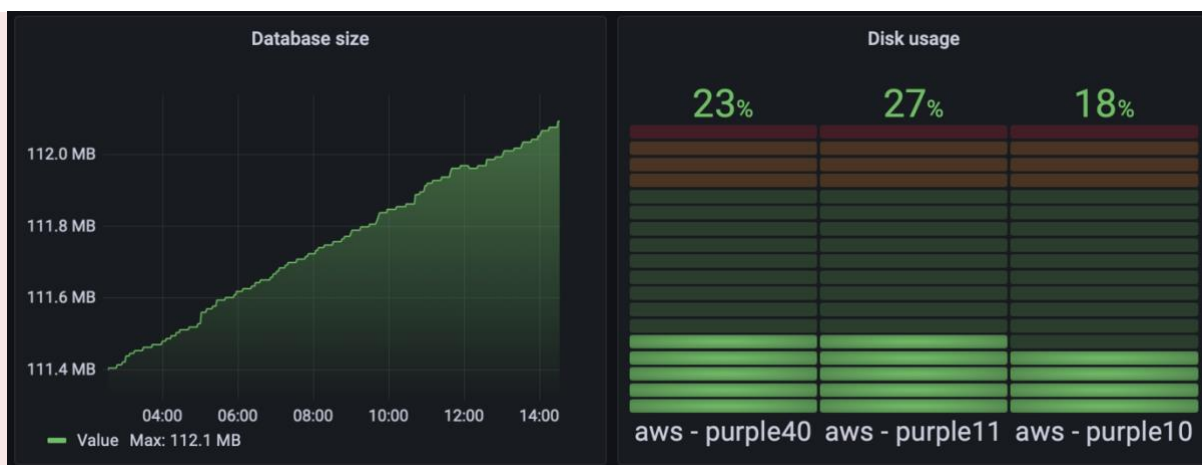


*Threads* reflect the fact that the Controller uses Java and creates a thread for each task it is asked to perform. All threads consume some resources such as RAM. Threads can get 'stuck' for instance while waiting for a reply from an external call. The thread count should not be climbing forever!

*Memory usage* of the Controller should be pretty constant as they do not really consume very much memory while doing its job unless the thread count is out of control.

TIME SERIES uses:

ctr\_threads  
apn\_memory



*Database size* can grow to several GB these days as we have some quite long tables in newer versions (with larger image sizes too). The sawtooth is the vacuum process PostgreSQL uses.

*Disk usage* is important for Controllers – especially when performing upgrades which can cause downtime when something goes wrong.

This is more of a check for something that is behaving in an unexpected way as opposed to monitoring some trajectory of a correctly performing system.

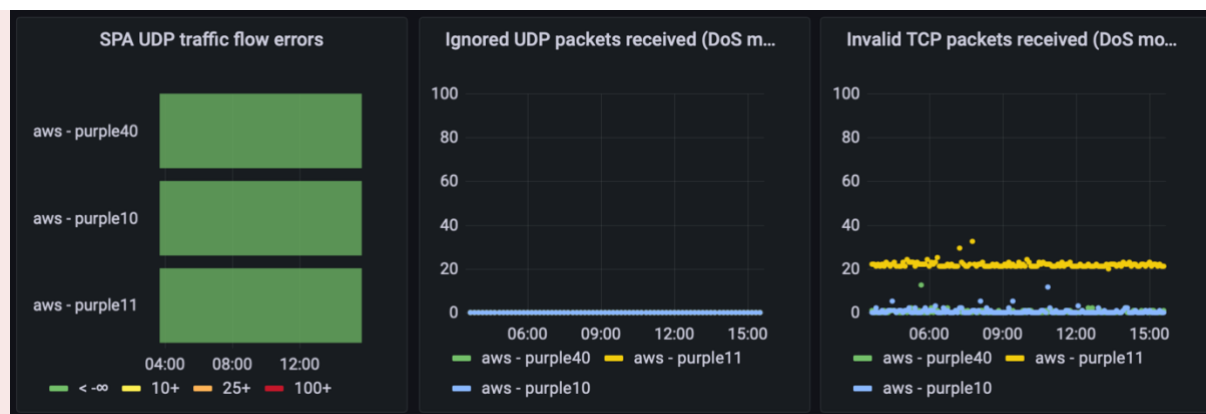
TIME SERIES uses:

ctr\_database\_size

BAR GAUGE uses:

apn\_disk





These 3 panels are concerned with networking – including that the Controllers are able to accept user connections.

*SPA UDP traffic flow errors* is monitoring the arrival of the UDP packets from Clients/Appliances. Connections always start by sending 2 different UDP SPA packets. Only one needs to arrive but for reliable operation we need to be sure that both are getting through. Here we are looking for the difference between the two (there should always be the same number). This quickly shows when one of the paths is blocked.

*Ignored UDP packets received* relate to invalid DTLS and DNS (non SPA) packets; *Invalid TCP packets received* relate to invalid TLS (non SPA) packets; so here we are basically reporting the on non SDP traffic hitting port 443 (& 53).

These might for instance indicate a DoS attack. This is more relevant on Controllers where they are likely to have a public DNS record. We need to look for an increase in the metric reported to give us a value per time. This could also be done for IPv6.

STATE TIMELINE uses:

```
abs( increase(spa_dtls_authorized) - increase(spa_dns_authorized)
```

TIME SERIES uses:

```
increase(apn_spa_packets {type="ignored"})
```

```
increase(apn_spa_packets {proto="tcp", type="invalid" }
```



*Average sign-in time* measures one of the primary functions of a Controller which is to authenticate and authorize users/devices when they connect to the Appgate SDP Collective. The key metric here is how long sign-in it takes – you don't want users hanging around for many seconds because of some system misconfiguration! This will quickly expose slow external calls or scripts that might be taking longer than you expect.

This metric uses metrics which don't suffer from user induced delays such as when performing some sort of MFA input.

TIME SERIES uses:

```
ctr_client_authentication {measure="average_time"} + ctr_client_authorization {measure="average_time" }
```

This is effectively the time to generate the claims token + the time to generate the entitlement token. We have set a warning area in the chart at 1second.

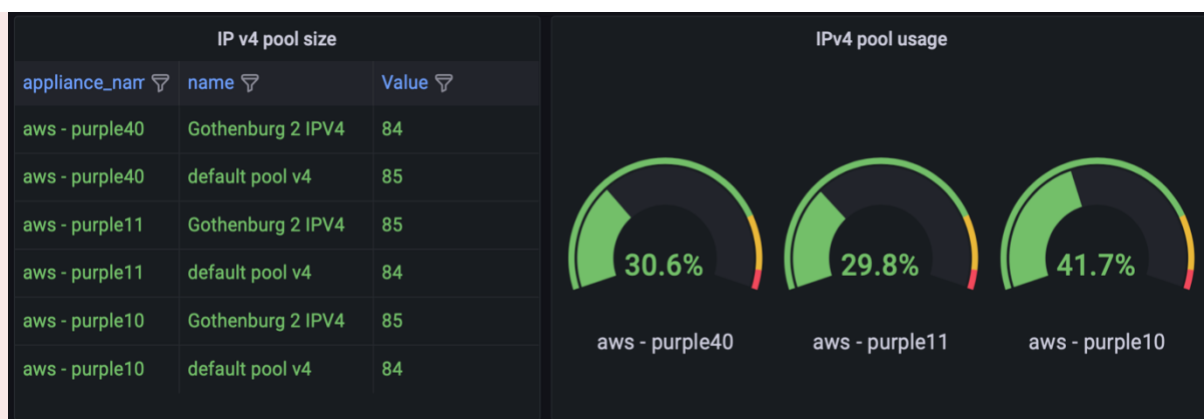


*Failed Authentications per hour* is a measure of user behavior. Remember because of SPA this is unlikely to be BOTs guessing their way in and is more likely to be users with real problems.

*Authorizations per hour* is the best representation of the full work being done by the Controllers as it is measuring the output of the process.

The TIME SERIES uses:

```
rate(ctr_client_authentication {measure="error"})
rate(ctr_client_authorization {measure="success"}[$__interval])*60*60
```



*IPv4 pool size* shows you that in fact the pool is split (not shared) – in this case the default pool is split 3 ways. When one or more Controllers are down (during an upgrade) the availability of new IP addresses is massively reduced! It is important to measure each Controller separately.

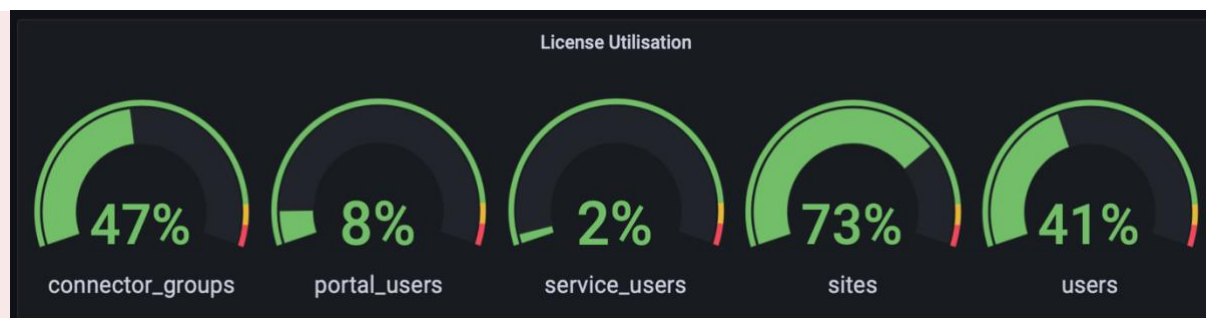
*IPv4 pool usage* monitors one of the things that can stop a Controller performing its job (which should always get priority when it comes to monitoring). When a user signs-in they are assigned an IP address from the pool that the Controllers use. If there are none available, then the user won't be allowed to sign-in.

TABLE uses:

```
ctr_ip_pool {usage="total"}
```

GAUGE uses:

```
(ctr_ip_pool{usage="current"}+ignoring (usage) ctr_ip_pool{usage="reserved"})/ignoring (usage)
ctr_ip_pool{usage="total"}*100
```



*License utilization* is the other thing that can stop a Controller performing its job is running out of Licenses. The ones that matter (in real time) are the user and portal licenses. For completeness we also include the other types of Licenses that can be installed in the Collective.

GAUGE uses:

```
ctr_license {measure="used"} / ignoring (measure) ctr_license {measure="entitled"} *100
```

You only need to query a single Controller because the license pool is shared across all Controllers.



## Gateway Health Dashboard

You might find some of the panel descriptions missing for Gateway. We have omitted any panels/descriptions that are shared with the Controller unless it is important to emphasize specific differences.

This example dashboard covers two of the Sites in a Collective: Heathrow and Gothenburg. The exact panels you select for your dashboard should reflect the Site's use case. A default gateway Site might need to reflect the fact that they have potentially very high network traffic throughputs while using an almost static set of access rules.

Heathrow

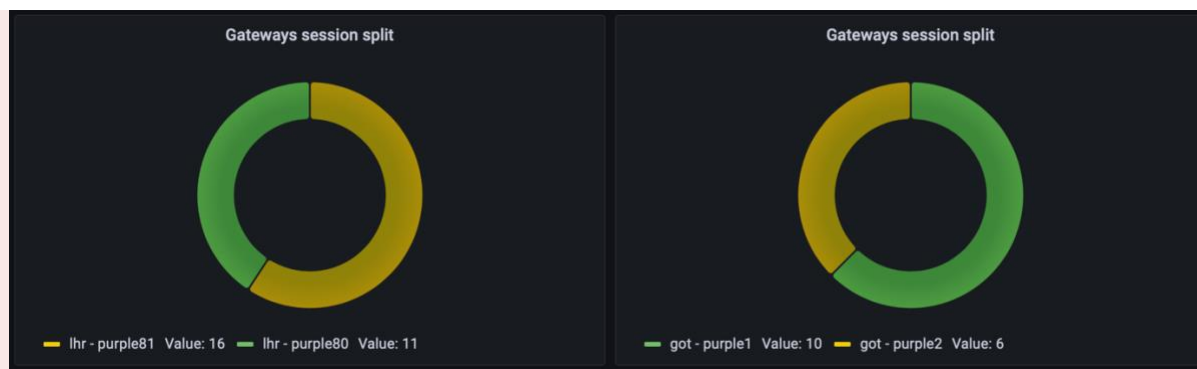
Gothenburg

Each Site (Heathrow and Gothenburg) has two Gateways (Purple 80/81 and Purple 1/2).

TEXT uses:

`<h2><center>Heathrow</center> and`

`<h2><center>Gothenburg</center>`



*Gateways session split* reflects the current load balance between the Gateways in the Site. This is dictated by the weighting factor that was set when the Gateway was configured. If the (two) weightings are the same, then you would expect to see a 50/50 split in a steady state system. Clearly if there is a networking problem this would quickly show up. But the split can also change for valid reasons – maybe you recently upgraded the Collective in which most users will now be on the first Gateway that was upgraded.

Client Tunneling - Load Balance Weighting Factor

100

PIE CHART uses:

`gw_vpn_sessions`



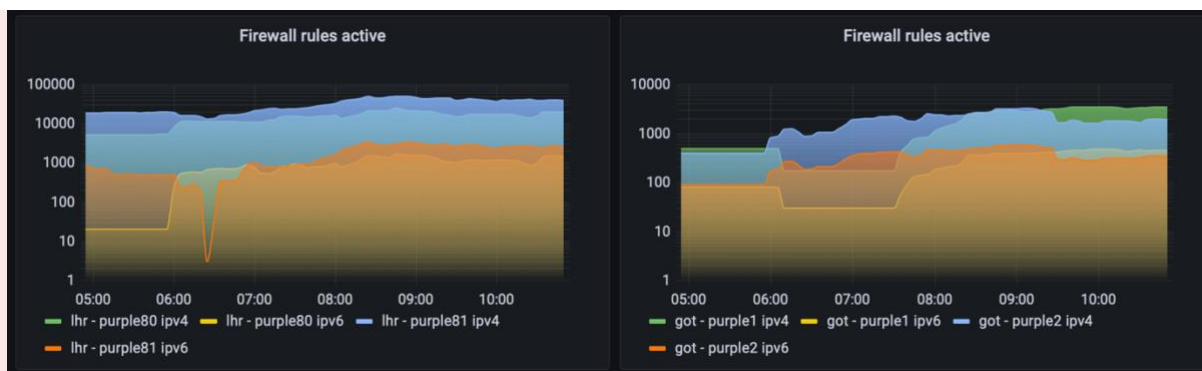
*NIC throughputs* are another key metric to monitor as Gateways are primarily about passing user/device traffic. Appliances/instances can have very different NIC specifications so it might be good to set a warning band at their limit. Reaching the limit is not a major problem as the available bandwidth will be shared across all the users/devices. The specific interfaces are not referenced as this is just an indicator of any potential problem. The dashboard will show this detail if required.

TIME SERIES uses:

`avg_over_time(apn_network_interface_rx_speed)/1000000`

`avg_over_time(apn_network_interface_tx_speed)/1000000`



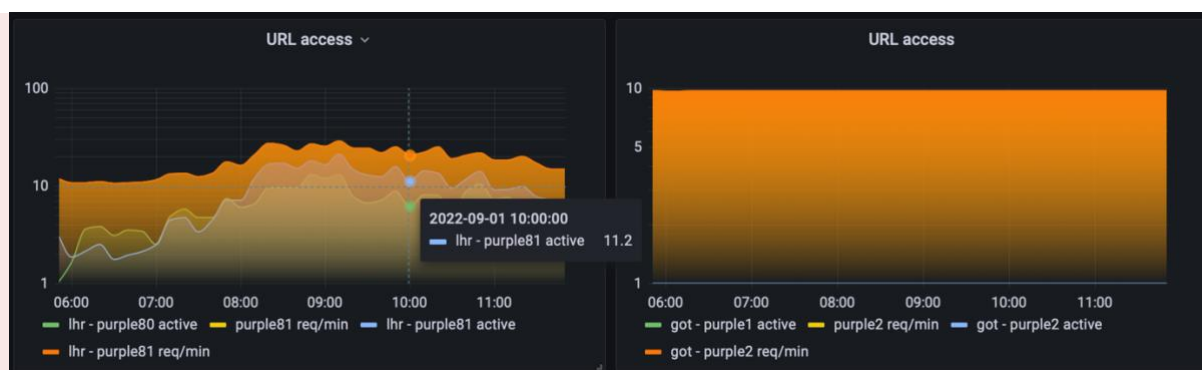


*Firewall rules active* is fun to see but can have a serious purpose as well. It is fun because Appgate SDP generates firewall rules as users sign-in, so at night there can be very few and during the day, there may be 10M or more active rules! The serious side is more to do with checking that this is the behavior you expect to see from a security perspective – should there be new rules appearing at 3am?

This metric also tracks the behavior of failover events when for instance system upgrades are performed.

TIME SERIES uses:

`avg_over_time(gw_vpn_rules [5m])`



*URL access* is used when you choose the HTTP up action type for an Entitlement (the firewall rules engine is not used). A web proxy is used to filter the traffic based on the URL.

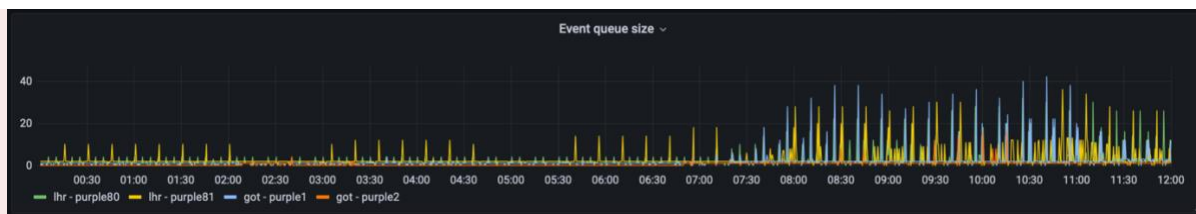
**URL Access (i.e. containers)**  
HTTP up

*Active* (connections) provides an equivalent metric to the one above for this specific type of action.

Req/m metric is useful as the number of https requests that can be handled is resource constrained.

TIME SERIES uses:

`avg_over_time(url_access_active_connections [5m])`



*Event queue size* is recording the peak value of the current event queue in a rolling time window.

Appgate Gateways are very different from traditional firewalls. Everything is managed on a per-user basis and access rules can be quite dynamic. So, there can be lots of individual events to handle in a system.

A finite number of events can be processed in parallel (depending on the RAM available in the appliance). Beyond this number, events are queued for processing later. This metric is quite important for ensuring the queue size does not grow uncontrollably in say a Gateway failover; a situation has been seen a number of times. The queue recovery time can be seen from this metric also. CPU and/or RAM can then be adjusted to suit the required recovery time.

Of the nine types of events, re-evaluations are possibly the most common of these events and are mostly done at 5, 15 and 60 minute intervals.

TIME SERIES uses:

`gw_event_queue_period_peak`

```
domainResolveUpdate
handleRemedy
handleTunnelDisconnected
login
nameResolveUpdate
reEvaluateConditions
reGenerateExpressConnectorRules
updateTokens
validateTokens
```



*Time to sign-in* and *time to re-evaluate Conditions* are typical of the types of events in the system at any one time. It is therefore very important to monitor the event execution time. If times grow longer, eventually the system will run out of time to clear the event queue at which point things will start going awry.

The *type* label allows you to select the types of events which are of interest to you. Ones that rely on outside influences (via scripts) are likely to be more critical than the others, so here we show two: re-evaluate Conditions and sign-in.

Do remember to monitor all the Gateways. In this case all 4 are combined into the one heatmap; where you can see there is a high frequency of events taking between 3 and 4 seconds!

HEATMAP uses:

`gw_session_event_timing {measure="average",type="login"}` and  
`gw_session_event_timing {measure="average",type="reEvaluateConditions"}`



*Java heap* relates to the fact that the session daemon is written in Java which has its own strange ways of working. It grabs chunks of memory based on what it is doing and only gives them back occasionally when it performs garbage collection.

Even though two Java metrics are provided - It is probably unnecessary to monitor either *java thread count* (if the events are being monitored adequately) or *Java heap* (as its maximum size is based on a proportion of what RAM is available). However there may be some value in knowing roughly what Java is using when making a final judgement as to the amount of RAM to configure in your Gateways.

TIME SERIES uses:

```
gw_sessiond_heap {measure="used"}
```



*VPN daemon memory usage* is unlike the session daemon usage. It will just use more and more based on what the Gateway is being asked to do. The amount per user will only be in the KB range; even though this appears small, when multiplied by several thousand users can amount to many GB.

Again, it is probably unnecessary to monitor this if memory usage (overall) is measured. However there may be some value in knowing roughly what is being used when making a final judgement as to the amount of RAM to configure in your Gateways.

TIME SERIES uses:

```
avg_over_time(gw_vpn_memory_usage [5m])
```



*Memory usage* in Gateways can have very dynamic use of memory (unlike Controllers). Because we do not use disk caching then users may be dropped when a Gateway runs out of memory. So, this is a key metric to monitor. If you have complex Conditions and use scripting then this can be a very important metric as these use VPN and session daemon resources which can get quite memory hungry.

In Linux there are several different ideas for how to report memory usage, a conservative metric has been used (it will show the worst case).

GAUGES use:

```
apn_memory {measure="percent"}
```

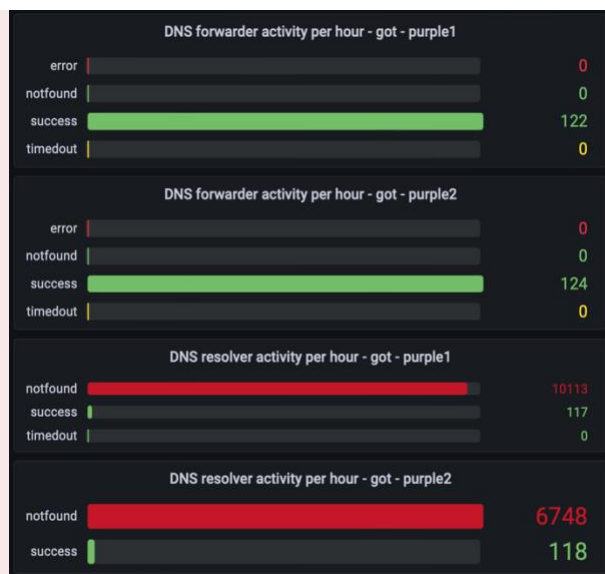


*DNS Forwarder activity per hour* and *DNS resolver activity per hour* use just two of the metrics available in respect of name resolving. Here we include the one from the DNS forwarder and the one from the DNS resolver. These are shown more as an example than the right metric for you to use. If, for example, you are a heavy AWS user then displaying AWS name resolver metrics will probably have more value than the DNS ones.

In both these metrics we are just looking at DNS look-ups and the related results. This gives us two important things; firstly, the rate of lookups (DNS may be rate limited), and secondly, the results - which would immediately show if there was some sort of configuration issue.

BAR GAUGES use:

```
rate(gw_dns_forwarder_query
{proto="ipv4",type="a"} [1h])*3600
rate(gw_dns_resolver_query {type="a"} [1h])*3600
```



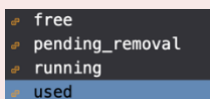
## Portal Health Dashboard

You might find some of the panel descriptions missing for Portal. We have omitted any panels/descriptions that are shared with the Controller/Gateway unless it is important to emphasize specific differences.

This example dashboard covers a single Portal. The exact panels you select for your dashboard are likely to be very similar as a Portal's metrics are quite straight forward.

*Running Client Instances* in the Portal will be based on the installed RAM. Make sure this is adequate for your user-base.

There are several *type* options you can use. Here we are displaying *running* and *used*.



STAT uses:

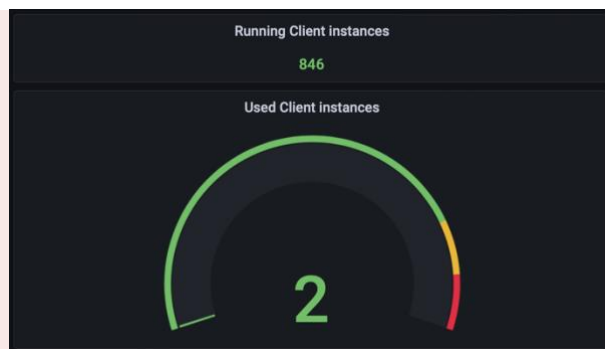
```
ptl_client{type="running"}
```

GAUGE uses (for the value):

```
ptl_client{type="used"}
```

and to set the full-scale value:

```
ptl_client{type="running"}
```



*Top Domains* metric is available because we run a DNS proxy on the Portal. (This ensures that each users' DNS traffic is handled uniquely). It allows us to see which services users are using through the Portal. This can allow the Entitlement assignments to be optimized. Remember the Portal is not SPA protected so limiting access rights should be a priority!

TABLE uses:

```
ptl_dns_proxy_domain
```

Top Domains	
name	Value
got-teamcity.agi.appgate.com.	257
got-testrail.agi.appgate.com.	7
got-youtrack.agi.appgate.com.	311



Hourly DNS Proxy activity is looking at DNS look-ups and their related results. This gives us two important things; firstly, the rate of lookups (DNS may be rate limited), and secondly, the results - which would immediately show-up any sort of configuration issue.

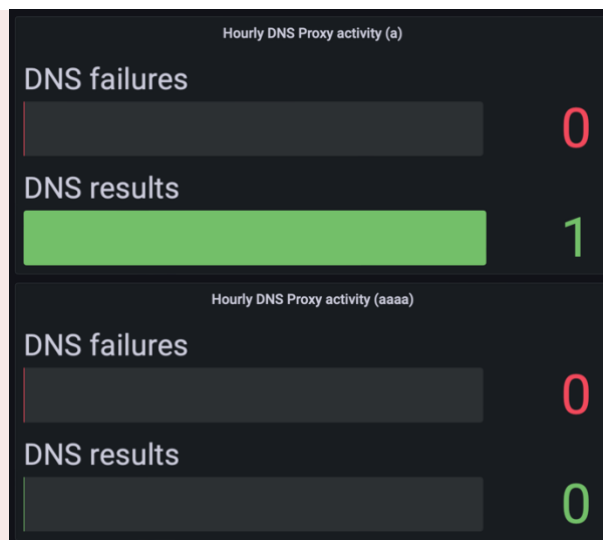
Separate gauges are included for *type a* and *aaaa* lookups.

For the bars we have added the different *result* types to produce just two, one for good and one for bad.

There is also a `{result="cache"}` label which shows how many of the successes came from the internal cache.

BAR GAUGES use:

```
increase(ptl_dns_proxy_query {result="timedout"} [1h]) + ignoring (result) increase
(ptl_dns_proxy_query {result="notfound"} [1h]) +
ignoring (result) increase (ptl_dns_proxy_query
{result="error"} [1h])
increase(ptl_dns_proxy_query {result="success"} [1h])
```



## Conclusion

The information and examples we have shown are likely to cover the key metrics that should be monitored in most Collectives. The approach we have taken is more important than the exact examples we used. It is important to monitor your Appgate SDP system in the way that best suits your deployment. Measuring what it is doing is probably as important than measuring what it is consuming.

The ability to judge if something is coping well or is on a bad trajectory should be easy to see and where appropriate trigger alerts and take any corrective actions before any users are impacted. These types of systems can fail fast (avalanche type failures) when things begin to go wrong so relying on metrics like RAM usage will give no advanced warning and only serve to confirm that something has already gone badly wrong!

We have explained three real-life examples using Prometheus and Grafana and hope these will inspire you to do the same for your own Collectives. To help you on your way we have added these three [sdp-grafana-dashboards](#) to our public Github repository.

## More information

Appgate delivers secure access solutions that thwart complex threats, reduce costs, boost operational efficiency and secure the lives of the people that rely on them. Through a suite of network security and fraud protection solutions - including the world's leading Zero Trust Software-Defined Perimeter architecture.

Appgate is relied on by global enterprises seeking to protect their digital assets.

Start your secure access journey with confidence by visiting <https://www.appgate.com/software-defined-perimeter>

Check out the Appgate SDP admin Guide: <https://sdphelp.appgate.com/adminguide/v6.0/monitoring-and-logs.html>

