Appgate SDP

Monitoring the SDP system

Type: Technical guide
Date: February 2021
Applies to: Appgate SDP v5.5.1 and newer

TABLE OF CONTENTS

INTRODUCTION

As with most other access devices, Appgate SDP is likely to form a critical part of a company's access infrastructure, therefore it is vital 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 (such as disk space).

It is therefore important to make plans to monitor the Appgate SDP system continuously and where appropriate trigger alerts and take any corrective actions before user access is impacted. The monitoring does not need to be there from the outset as the dashboard includes some basic health check information. However as you scale up the number of users it is important to have monitoring in place to get a clear understanding of how the system is performing.

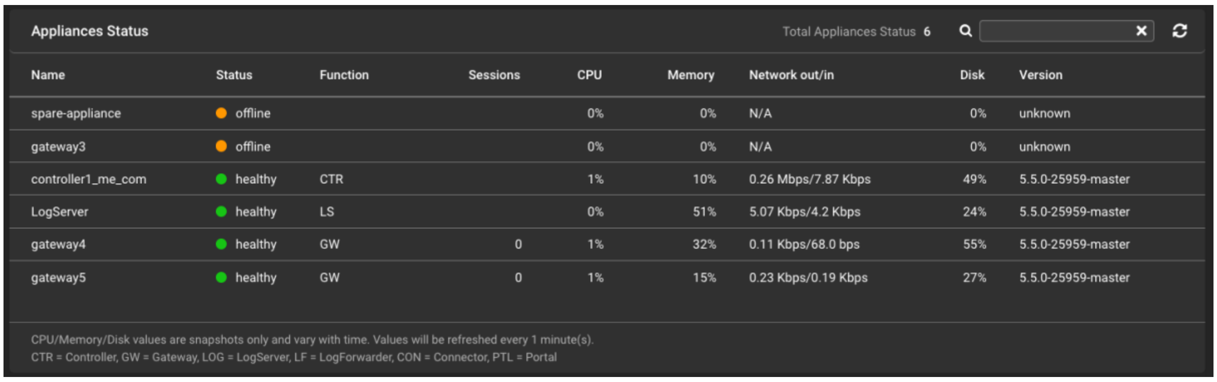
This paper takes you through the features that come with the Appgate SDP system which can be used for monitoring appliances. The paper also tries to explain which metrics matter and the best way to monitor those.

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 and Gateways – but we will also briefly touch on some of the other appliance functions as well.

MONITORING TOOLS

First, it is worth explaining what is available in Appgate SDP today.

There is some health-check monitoring provided by the Appliance Status display in the dashboard:



Appliances Status

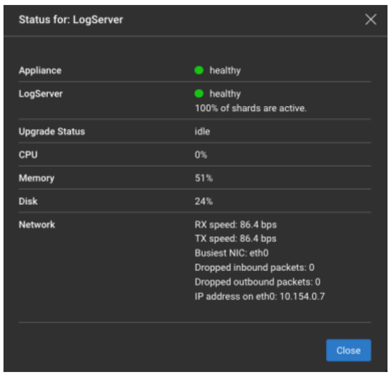
Total Appliances Status 6

Name	Status	Function	Sessions	CPU	Memory	Network out/in	Disk	Version
spare-appliance	● offline			0%	0%	N/A	0%	unknown
gateway3	● offline			0%	0%	N/A	0%	unknown
controller1_me_com	● healthy	CTR		1%	10%	0.26 Mbps/7.87 Kbps	49%	5.5.0-25959-master
LogServer	● healthy	LS		0%	51%	5.07 Kbps/4.2 Kbps	24%	5.5.0-25959-master
gateway4	● healthy	GW	0	1%	32%	0.11 Kbps/68.0 bps	55%	5.5.0-25959-master
gateway5	● healthy	GW	0	1%	15%	0.23 Kbps/0.19 Kbps	27%	5.5.0-25959-master

CPU/Memory/Disk values are snapshots only and vary with time. Values will be refreshed every 1 minute(s).
CTR = Controller, GW = Gateway, LOG = LogServer, LF = LogForwarder, CON = Connector, PTL = Portal

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 provides some additional information when you click on an appliance's status. 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.



Status for: LogServer

Appliance	● healthy
LogServer	● healthy 100% of shards are active.
Upgrade Status	idle
CPU	0%
Memory	51%
Disk	24%
Network	RX speed: 86.4 bps TX speed: 86.4 bps Busiest NIC: eth0 Dropped inbound packets: 0 Dropped outbound packets: 0 IP address on eth0: 10.154.0.7

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.

The proper way to monitor the Appgate SDP Collective – is by integrating with external monitoring tools such 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.

WHAT TO MONITOR

In the introduction, we have already set out some goals for what we should be monitoring:

- 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 what should matter as each and every Collective will be slightly different in the way it works.

Appgate has supported countless Customers with many different types of problems and each time we see another, 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 what actually matters.

Customers are unlikely to have a detailed understanding of the inner working of the Appgate SDP system, even with this insight, the prospect of trying to find a metric that is worth monitoring is quite a daunting undertaking. Many customers will therefore have pushed monitoring into the ‘done it’ pile (monitoring RAM and disk is good enough) or the ‘too hard’ pile.

This problem of finding the right metric is exacerbated by the fact that in v5.5 there are far more metrics exported than you are likely to need for day-to-day monitoring; most of these metrics are somewhat disorganized; and there are currently one or two useful metrics that are not available via SNMP/Prometheus. Monitoring is a current area of focus for Appgate, so expect to see additions/changes in the upcoming releases. If there is something we are missing in the current versions, please let support know and we will try to add it in an upcoming release.

In this paper, we hope to take the insight we have gathered, combine it with knowledge of the inner working of Appgate SDP 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 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 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 always use a minimum of 25% of the RAM available.

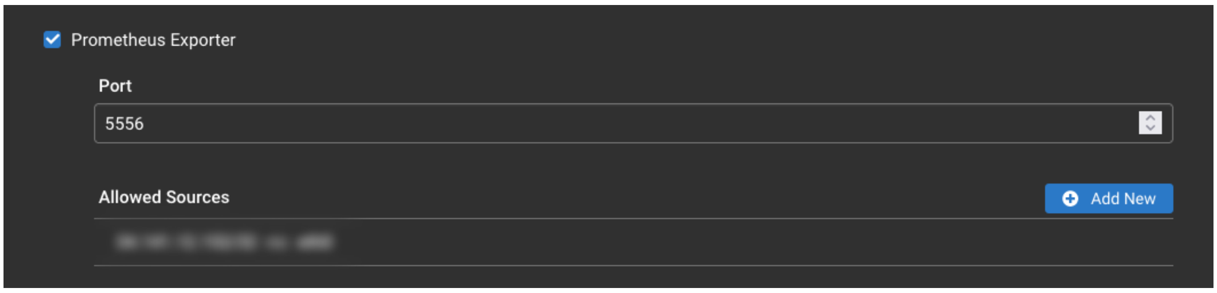
Do not forget about limits imposed by others. The system relies on things like DNS, The DNS servers or any firewalls between the Appgate SDP appliances and the DNS server might have some sort of per second limit. Exceeding this might result in requests being dropped and a user's access being denied. 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 Source’. This restricts access to only these known IP addresss.


☑ Prometheus Exporter

Port

5556

Allowed Sources

[redacted]

Available Metrics

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

```
ctrlLicenseEntitledUsers OBJECT-TYPE
   SYNTAX Gauge32
   MAX-ACCESS read-only
   STATUS current
   DESCRIPTION "Total number of entitled users in license"
   DEFVAL { 0 }
   ::= { ctrlLicenseEntitled 1 }
```

The equivalent metric in Prometheus would look like this:

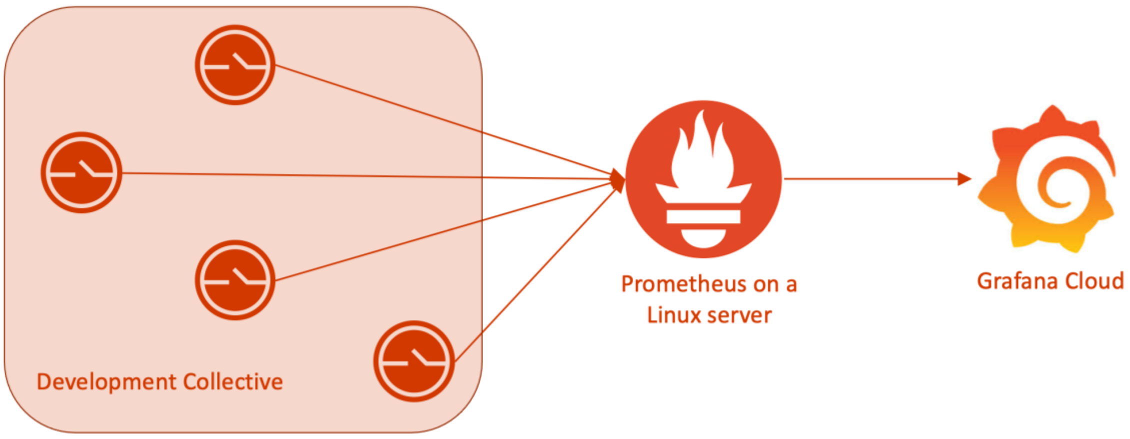
```
ctrl_license_entitled_users
```

The description is carried across to Prometheus unchanged.

You can usually browse the available metrics in Prometheus (or Grafana) so downloading the SNMP MIB is not strictly necessary.

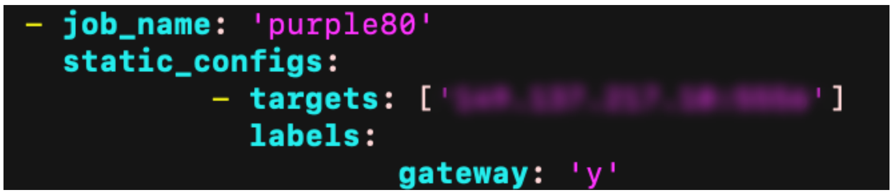
The Prometheus set-up

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



The way Prometheus works is it will scape 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: ['[redacted]']
            labels:
                    gateway: 'y'
```

The current version of Appgate SDP (v5.5) has a working but somewhat unstructured implementation of Prometheus. In the above example you will see we are adding an extra label – *gateway*. We should be including labels automatically but at this time they are not in use in the product.

It would for instance make sense to have labels like *function={"ctr* | *gw* | *log* | *lf* | *con* | *ptl"}* and *site={"site1* | *site2* | *site3"}* included. These are very valuable when designing your panels in Grafana because they allow easy selection of data sources – say when doing a dashboard for the Gateways on a specific Site.

It is worth thinking about the panels you want to display first; this will drive the choice of labels that you might want to use. We suggest you compensate for the temporary dearth of labels by adding some of your own labels at the same time as you are editing your YAML file.

Grafana dashboards

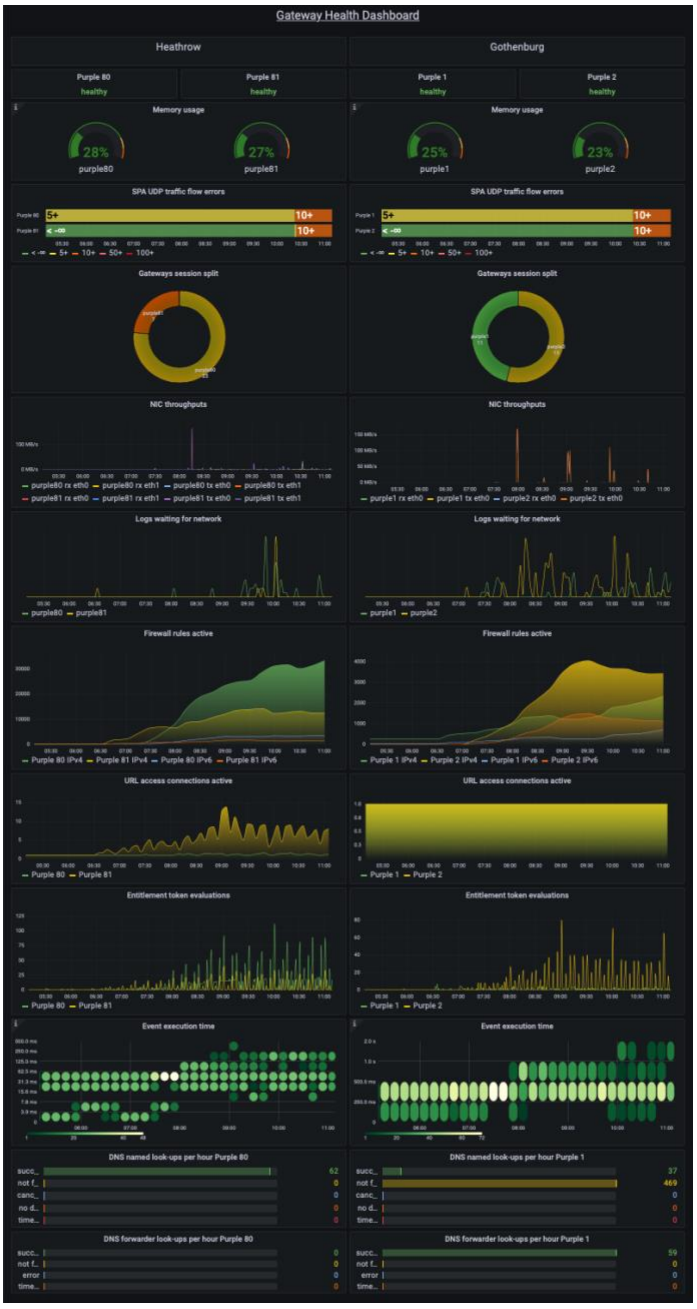
In this paper we explore 2 example dashboards – one for Controllers and the other for Gateways.

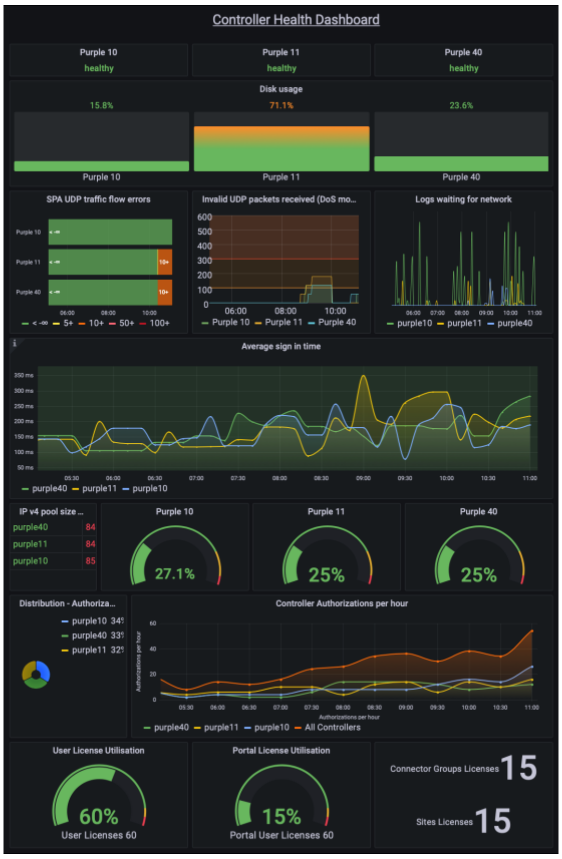
Many of the metrics are counters (value always increases) so we quite often have to derive increases or rates in order to display meaningful data.

Consider the time intervals that are used for each panel in order to make the information easy to consume.

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 look very different over time.

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!

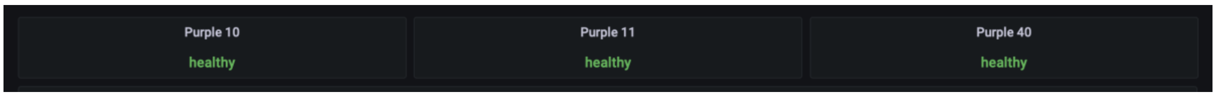




For each of the metrics we have included in these two example dashboards, we will go through the rational for their inclusion. Details of the specific metric(s) we use are also included.

Controller Health dashboard

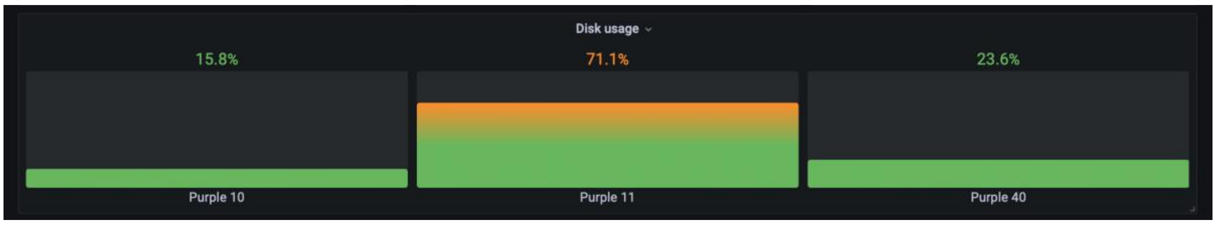
This dashboard covers all three Controllers deployed in the Collective.



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. This is triggered by very many different checks that are performed by the hosting appliance, but is also driven by some health-checks that come from the specific functions that have been enabled in the appliance.

STAT uses:

sdp_appliance_function_status

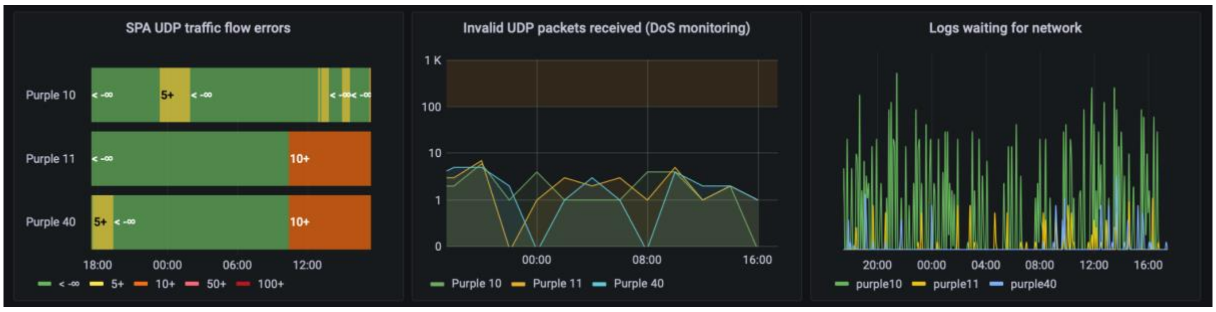


Disk usage is important for Controllers – especially when performing upgrades which can cause downtime when something goes wrong. It will grow by a few GB as the database size grows and may grow with newer versions (with larger image sizes).

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.

BAR GAUGE uses:

sdp_appliance_disk_usage_percent



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

Left

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 though. 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.

The STATE TIMELINE uses:

abs(increase(spa_dtls_authorized [1d]) - increase(spa_dns_authorized [1d]))

Middle

As we already mentioned we are looking for valid DTLS and DNS SPA packet so here we are basically reporting the *Invalid UDP packets received* on port 443. This might for instance indicate a DoS attack. This is more relevant on Controllers where they are likely to have a public DNS record. The metrics used are unfortunately not quite working as intended in v5.5 so we need to look for an increase in the metric reported. This could also be done for IPv6.

The GRAPH uses:

increase(ipv4_invalid_udp_dns_packets [2h]) + increase(ipv4_invalid_udp_dtls_packets [2h])

Right

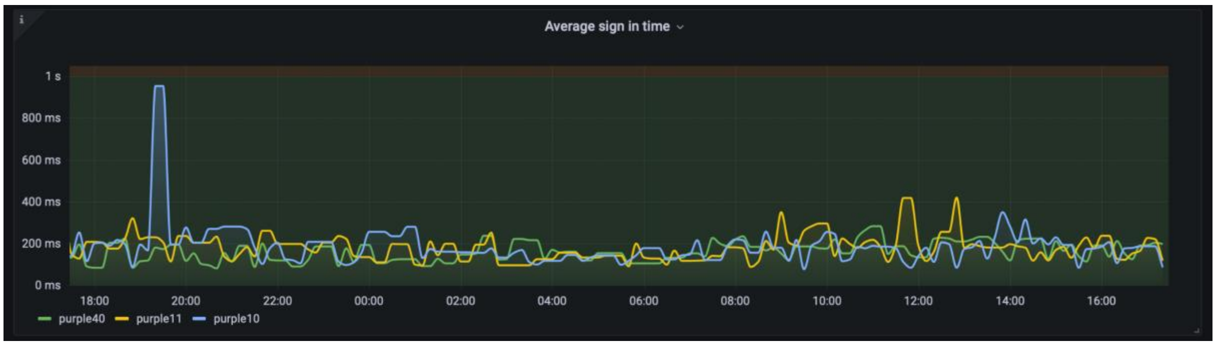
Logs waiting for network is monitoring the export of logs from each appliance. Logs are added to the export queue and removed only when an ack is received back from the receiving appliance. This metric is there to ensure a good implementation with working two-way communications, and that there is no bottleneck between the appliance and the log destination.

There is no scale shown because the size of this queue is not interesting – indeed, the queue will normally be close to empty in a fast well-functioning system. Any spikes should always return to the axis promptly on a lightly loaded system. During an upgrade you might see is a growing backlog which then clears, but if it does not clear that means the network/log server cannot cope with the current load.

The TIME SERIES uses:

avg_over_time (audit_logs_read [3m])-avg_over_time(audit_logs_removed [3m])

appgate



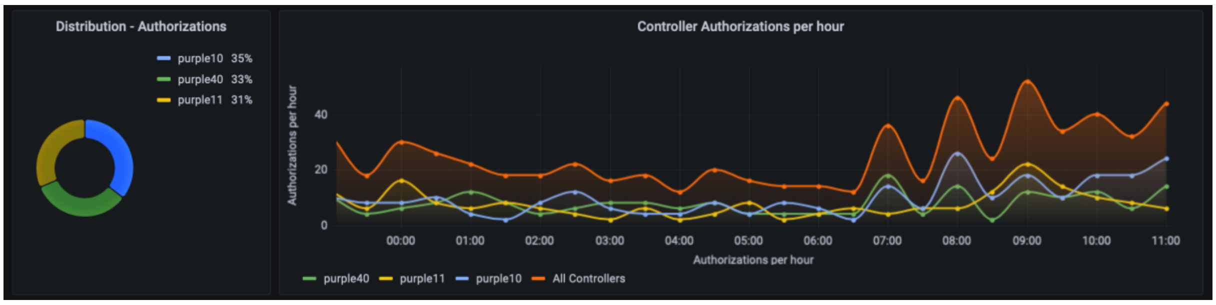
The primary function of a Controller is to authenticate and authorize users/devices when they connect to the Appgate SDP Collective. The key metric here is how long sign-in it takes – you don't want users hanging around for many seconds because of some system misconfiguration! This will quickly expose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

This is tracking aspects of *Average sign-in time* which don't suffer from too much user induced delays such as when performing some sort of MFA input.

This TIME SERIES uses:

ctrl_client_authentication_average_time + ctrl_client_authorization_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.



It is important to check that all the Controllers are performing their roles. Having set up DNS round-robin the 3 Controllers should be sharing the load fairly equally. The pie chart *Distribution – Authorizations* makes this very visible but does not show the actual load the Controllers are under.

For this we have included *Controller Authorizations per hour*. If we had used 'authentications' this would have included failed ones or ones that do not go on to be authorized. 'Authorizations' therefore represents the better; the full work being done by the Controllers.

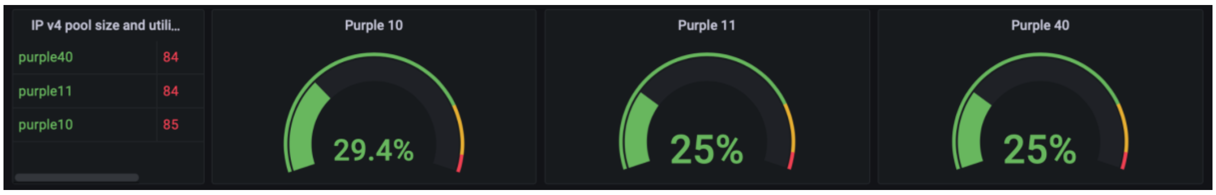
The PIE CHART uses:

ctrl_client_authorization_total

The TIME SERIES uses:

*rate(ctrl_client_authorization_total[30m])*60*60*

*sum(rate(ctrl_client_authorization_total[30m]))*60*60*



purple40	84
purple11	84
purple10	85

Things that can stop a Controller performing its job 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.

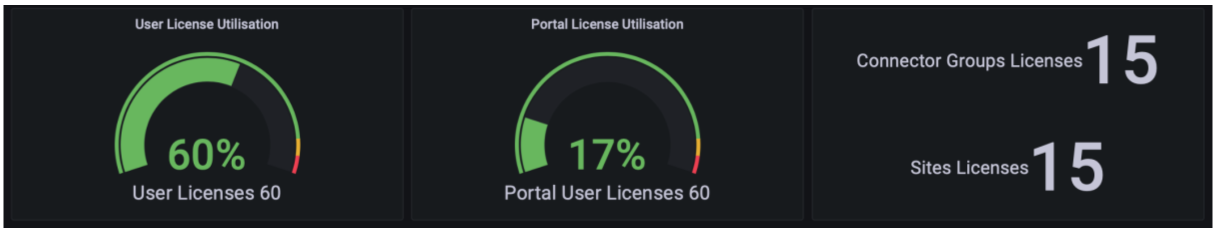
The pool is in fact split (not shared) – in this case the default pool is split 3 ways. When one or more Controllers are down (during an upgrade) the availability of new IP addresses is massively reduced! It is important to measure each Controller separately.

The split is shown using a TABLE which uses:

ctrl_ip_pool_total

The GAUGE for each Controller uses:

*(ctrl_ip_pool_currently_used + ctrl_ip_pool_reserved) / ctrl_ip_pool_total *100*



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. So here we have included a GAUGE for each of these. For completeness we also include a STAT which shows how many Connector Resource Group Licenses and Site Licenses are installed in the Collective.

This GAUGE uses:

*ctrl_license_used_users/ctrl_license_entitled_users*100*

*ctrl_license_used_portal_users/ctrl_license_entitled_portal_users*100*

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

And STAT uses:

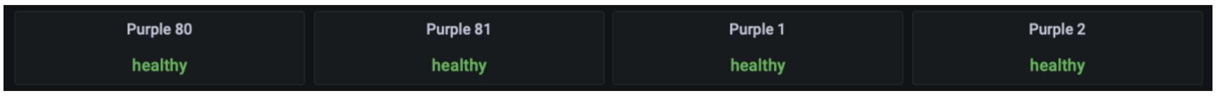
ctrl_license_entitled_connector_groups

ctrl_license_entitled_sites

Gateways

This dashboard covers just of two of the Sites in the 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 almost static access rules.

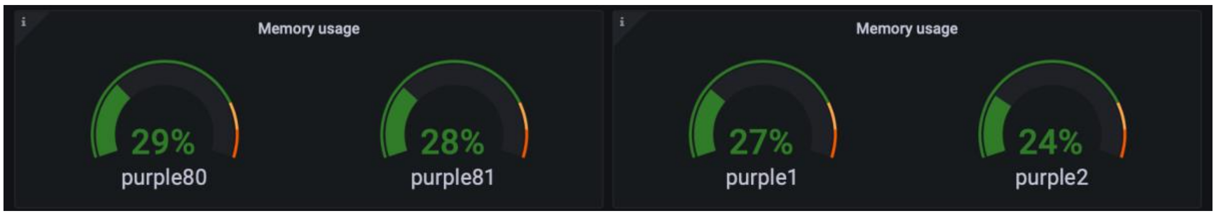
Each Site has two Gateways (Purple 80/81 and Purple 1/2).



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. This is triggered by very many different checks that are performed by the hosting appliance, but is also driven by some health-checks that come from the specific functions that have been enabled in the appliance.

STAT uses:

sdp_appliance_function_status

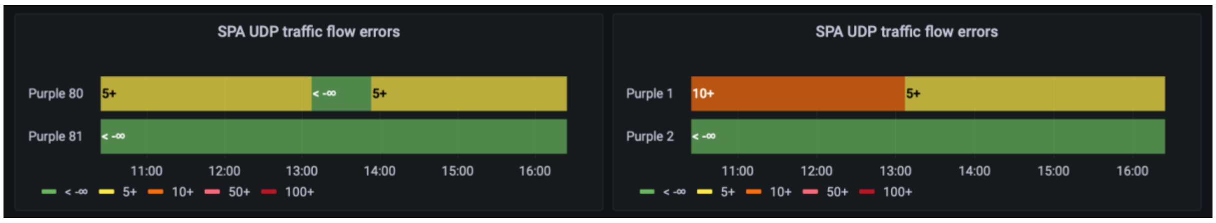


Unlike Controllers, Gateways can have very dynamic use of memory. Because we do not use disk caching then users may be dropped when a Gateway runs out of memory. So, this is a key metric to monitor. If you have complex Conditions and use scripting then this can be a very important metric as these use a Java engine which can get very memory hungry.

In Linux there are several different ideas for how to report *Memory usage*, here we use a conservative model as it will show the worst case.

The GAUGEs use:

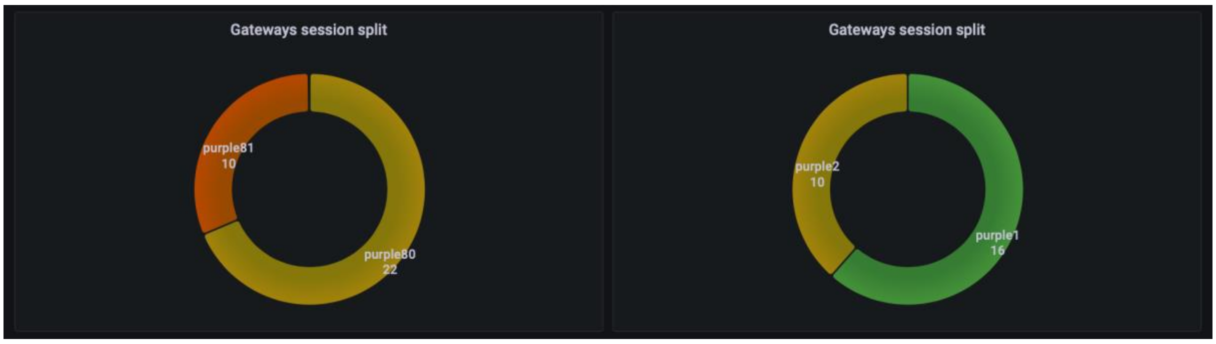
*100-(sdp_appliance_memory_available_bytes/sdp_appliance_memory_total_bytes*100)*



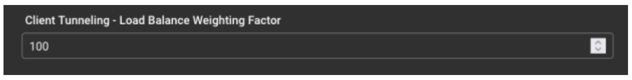
SPA UDP traffic flow errors is monitoring the arrival of the UDP packets from Clients/Appliances. Connections always start by sending 2 different UDP SPA packets. Only one needs to arrive but for reliable operation we need to be sure that both are getting though. Here we are looking for the difference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

The STATE TIMELINE uses:

abs(increase(spa_dtls_authorized [1d]) - increase(spa_dns_authorized [1d]))

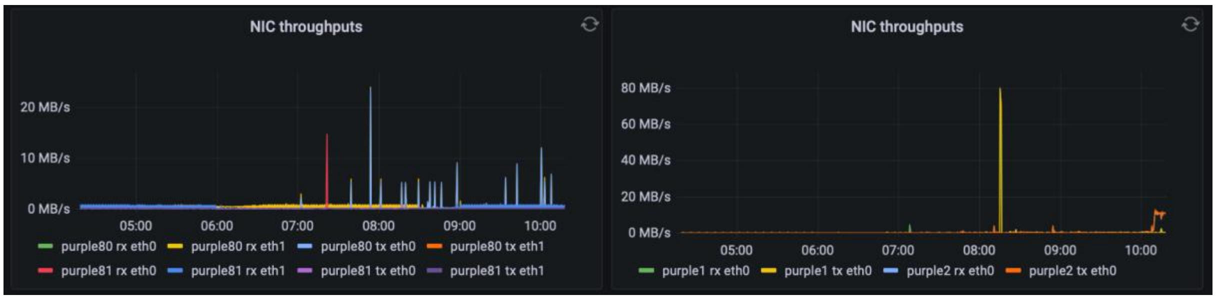


Gateways session split reflects the current load balance between the Gateways in the Site. This is dictated by the weighting factor that was set when the Gateway was configured. If the (two) weightings are the same, then you would expect so see a 50/50 split in a steady state system. Clearly if there 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

The PIE CHART uses:

sdp_appliance_function_number_of_sessions

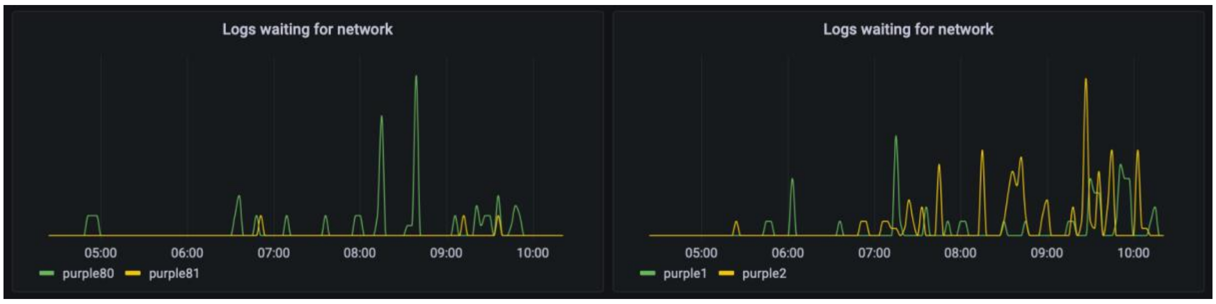


Gateways are primarily about passing user/device traffic, so *NIC throughputs* are another key metric to monitor. Appliances/instances can have very different NIC specifications so it might be good to set a warning band at their limit. Reaching the limit is not a major problem as the available bandwidth will be shared across all the users/devices.

The TIME SERIES uses:

avg_over_time(sdp_appliance_network_interface_rx_speed [30s])/1000000

avg_over_time(sdp_appliance_network_interface_tx_speed [30s])/1000000

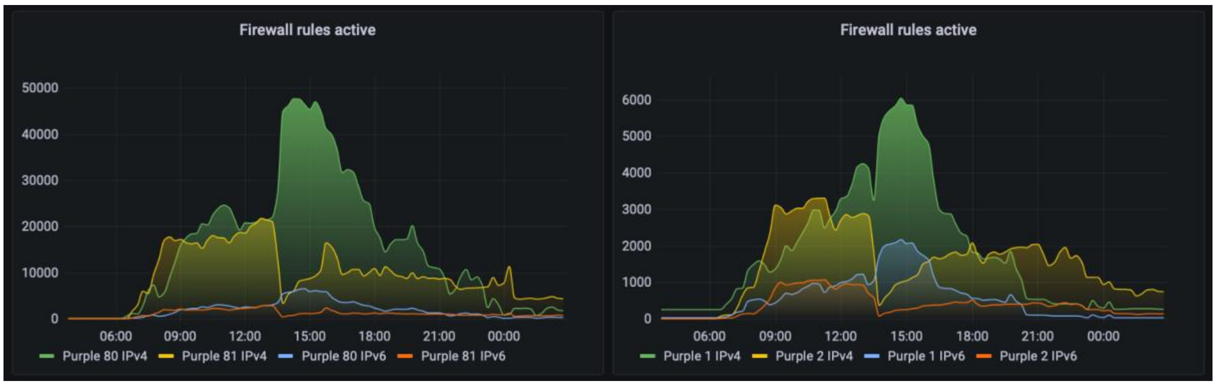


Logs waiting for network is monitoring the export of logs from each appliance. Logs are added to the export queue and removed only when an ack is received back from the receiving appliance. This metric is there to ensure a good implementation with working two-way communications, and that there is no bottleneck between the appliance and the log destination.

There is no scale shown because the size of this queue is not interesting – indeed, the queue will normally be close to empty in a fast well-functioning system. Any spikes should always return to the axis promptly on a lightly loaded system. During an upgrade you might see is a growing backlog which then clears, but if it does not clear that means the network/log server cannot cope with the current load.

The TIME SERIES uses:

avg_over_time (audit_logs_read [3m])-avg_over_time(audit_logs_removed [3m])



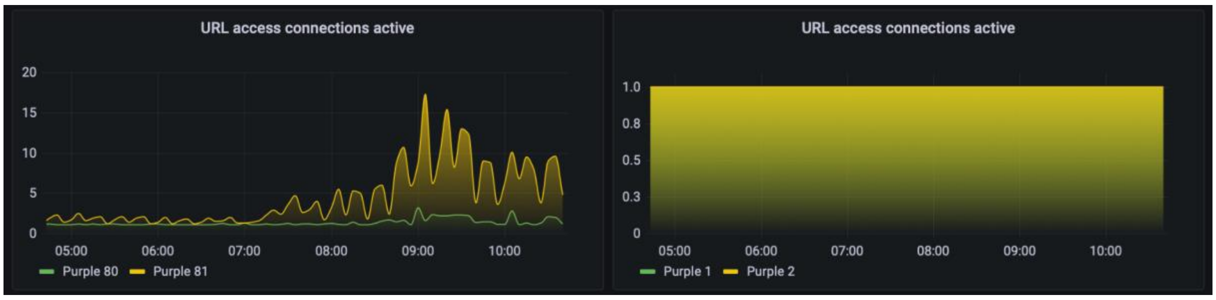
Firewall rules active is fun to see but can have a serious purpose as well. It is fun because Appgate SDP generates firewall rules as users sign-in, so at night there can be none (as you can see from Purple 2) and during the day there maybe10M or more active rules! The serious side is more to do with 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 clearly (as you can see at about 13:30) when for instance system upgrades are performed.

The TIME SERIES uses:

avg_over_time(vpn_total_ipv4_rules [15m])

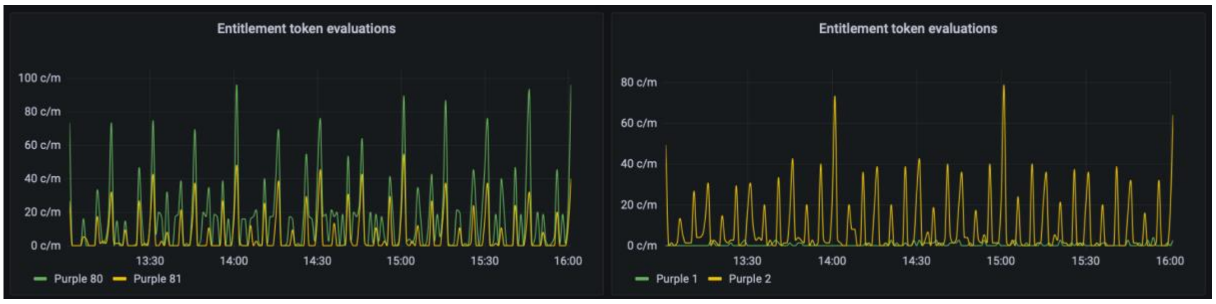
avg_over_time(vpn_total_ipv6_rules [15m])



When you use the HTTP up action type for an Entitlement, the firewall rules engine is not used. Instead, we have a web proxy that we use to filter the traffic based on the URL. *URL access connections active* provides an equivalent metric to the one above for this specific type of action.

The TIME SERIES uses:

avg_over_time(url_access_active_connections [5m])



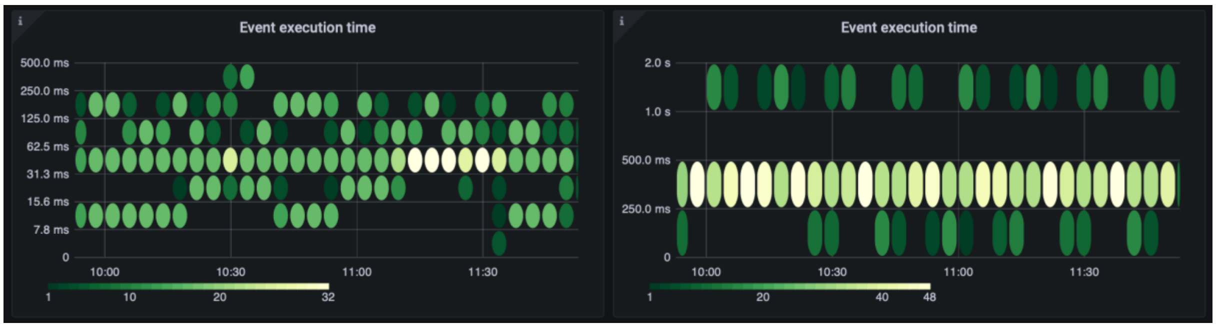
Appgate Gateways are very different from traditional firewalls. Everything is managed on a per-user basis - and access rules can be quite dynamic. So, there can be lots of rule change events to handle in a system.

Entitlement token evaluations are possibly the most common of these rule change events and are mostly done at 5, 15 and 60 minute intervals. This metric certainly shows the hourly peak. If this peak it too large, then events will be queued. This may not be the optimal metric to show how busy the non-traffic side of a Gateway is, but it certainly might hint at a potential problem that might otherwise remain hidden from view.

In this case we are looking at the audit_event metric. This provides counts/minute for many of the different audit log events we generate. In this case we are interested in the entitlement_token_evaluated type. In other situations, device_claims or claims_tokens might be of more interest.

The TIME SERIES uses:

*rate(audit_event {type="entitlement_token_evaluated"} [1m])*60*



As we discussed above there can be lots of rule change events in the system at any one time. It is therefore very important to monitor the *Event execution time*. As the time grows longer, eventually the system will run out of time to clear the event queue at which point things will start going awry. It would of course be better to monitor the current size of the event queue but in v5.5 this metric is not available.

If you remove the name label then all the average times will be shown, however, there are three types of events which are much heavier than the others, so here we show only these three: reevaluate conditions, update tokens and login.

Do remember to monitor all the Gateways in the Site.

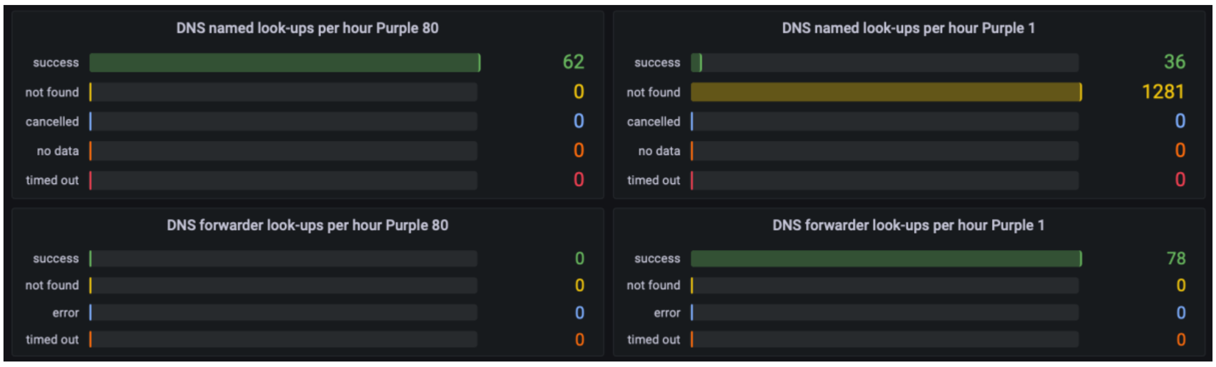
The HEATMAP uses:

sdp_gateway_session_event_average_time {name="reEvaluateConditions"} /1000

sdp_gateway_session_event_average_time {name="updateTokens"} /1000

sdp_gateway_session_event_average_time {name="login"} /1000

appgate



There are very many metrics available in respect of DNS and name resolving in general. Here we include one from the name resolver and one from the DNS forwarder more as an example than the best metric to be using. If you are a heavy AWS user then displaying AWS name resolver metrics will probably have more value that 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.

The BAR GAUGE uses:

*rate(nr_dns_query_a_??? [1h])*3600*

*rate(nr_dns_query_a_??? [1h])*3600*

??? is replaced by the different query response options available for each metric.

CONCLUSION

I hope the information and examples have shown you some 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 more 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 in fact has gone badly wrong!

We have explained two 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 both these to sdp-grafana-dashboards in 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

And the Appgate SDP product documentation is available here:

Admin Guide: https://sdphelp.appgate.com/adminguide

Client User Guide: https://sdphelp.appgate.com/userguide