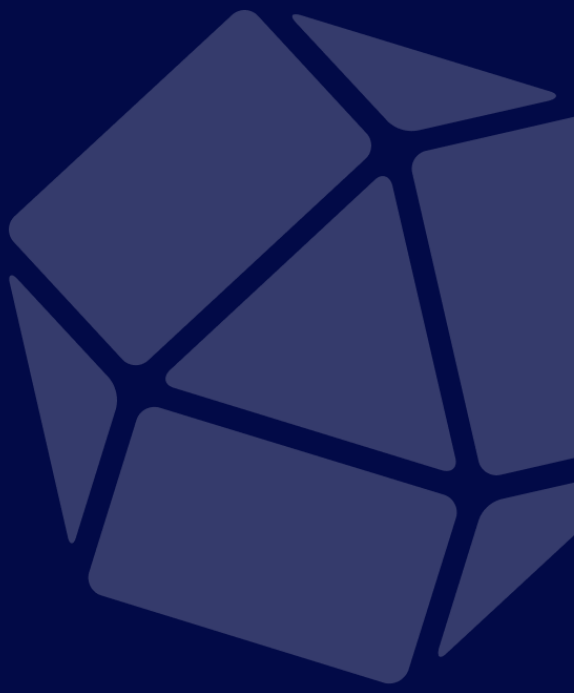
influxdata®

AN INFLUXDATA CASE STUDY

How Swiss Re Went Agentless with InfluxDB

John Hall
IT Artchitect, Swiss Re



OCTOBER 2018

Company in brief

Swiss Reinsurance Company Ltd, generally known as Swiss Re, is a reinsurance company based in Zurich, Switzerland. It is the world's second-largest reinsurer with offices in more than 25 countries. Swiss Re was ranked 118th in Forbes 2000 Global leading companies 2016.

The Swiss Re group is a leading wholesale provider of reinsurance, insurance and other insurance-based forms of risk transfer. Dealing directly and working through brokers, its global client base consists of insurance companies, mid-to-large-sized corporations and public sector clients.

From standard products to tailor-made coverage across all lines of business, Swiss Re deploys its capital strength, expertise and innovation power to enable the risk-taking upon which growth and progress depends.

Case overview

Swiss Re was looking for a proactive monitoring and trending solution that could work both on cloud and on-premise as well as Linux and Windows. They evaluated several solutions while building out the requirements for determining what kind of data to collect and how long they should keep the data.

In the end, they realized that they needed to collect a mixture of business process metrics (BPM's) to ensure they had a solution that could do proactive monitoring and trending. Swiss Re leveraged InfluxDB's flexible retention policies to meet the distinct data storage needs of their various teams and meet various monitoring objectives.

> *"When you have a platform as a service in the cloud, since it was agent-based, it was an impossible solution for us to maintain because you can't get the end cloud vendor to install something for you on a service that effectively they're maintaining."*
>
> ***John Hall**, IT architect, Swiss Re*

The business problem

Swiss Re's existing monitoring solution was problematic on several fronts:

- It was super expensive (in the six-figure range).
- It was slow to adapt (due to vendor-dependent releases) which is a problem with the leasing market that things do change on a regular basis.
- It was not cloud-ready and only worked with installed software, and was not really designed for a PaaS offering. It became impossible to maintain their solution, which was an agent-based PaaS in the cloud, because they can't get the end cloud vendor to install something for them on a service maintained by the vendor. While dashboards and reporting information can be provided by such vendors, it doesn't integrate into one place easily. Taking reports from different places and integrating them would require tremendous work.

Swiss Re's challenges with their existing solution led them to search for a new one, and they determined what the new solution had to do:

- Cost less and be easy to use
- Work with Linux and Windows
- Store audit and performance data

The technical problem

From an IT perspective, Swiss Re also determined what they wanted to do with the solution so that they could identify their technical requirements:

- Use the skills they already have (rather than be required to learn a new application or altering the way they operate)
- Must work on both public and private cloud
- Be as close to real time as possible

While an on-premise environment allows complete control, migration across to a cloud environment presents a very different set of rules because it's an environment that someone else owns and controls. In public cloud, for example with databases, Swiss Re couldn't grab CPU utilization in a lot of cases. But they could grab other useful information such as memory utilization and processor time that was spent on a task, without needing access to the platform.

Yet even access to the platform still limits what can be done. For example, they were able to log on to the machine and to look at counters but couldn't install anything there because it is maintained externally. Once they determined what they could and couldn't do, they reverse engineered that by finding out whether they can get, when logged in, the performance counters, and if so, whether they can do that agentlessly, either remotely or by running a script. Historically, they were very single-minded in simply installing agents, because they had full admin on the machine. But when you can't install software on the machine, you're forced to examine whether you can query it remotely, or just run a batch script that will output that information. They had to approach the problem differently, which compelled a new mentality.

They also wanted the solution to live in a container-based environment because they wanted to work with the principle of infrastructure as code (IaC). To do that, they needed to go for the container base, the ability to spin up repeatable instances without massive work underneath. This would also improve their speed of deployment and scalability. It also meant that they would be able to move between cloud vendors relatively easily if needed.

During the development phase, they faced several challenges including convincing management to transition their solution to containers. Many team members didn't understand the benefit, at the start, of changing service over to containers from VM's. One of the advantages of a container-based environment is that it was easy and repeatable. Yet there was some pushback as Git scripts—and checking them in and out of Git — was a learning experience for the IT admins (who are not developers), as these are traditionally developer tools. This required a slight change in the mentality of the IT department, which was easy to overcome.

Swiss Re evaluated several solutions including their existing solution, the ELK (Elasticsearch, Logstash, and Kibana) stack, and InfluxData's TICK Stack, and settled on the latter.

The solution

> *"So why InfluxDB? Well, a couple of simple reasons. And I don't work for the marketing department, just in case you ask. It was fast, it was scalable, it was stable, and it was easy to use."*

Why InfluxDB?

Swiss Re selected InfluxDB for its speed, scalability, stability and ease of use. They also found InfluxDB to be very flexible and well-documented, thereby reducing the learning curve. Their team had existing knowledge of InfluxDB, which allowed them to evaluate it quickly compared with existing vendor solutions.

For data visualization, Grafana was their first choice since it provided the level of detail needed through complex dashboards and since they already had the skills to use it.

The problem for Swiss Re was never getting the data but rather deciding how to use it constructively and agreeing what data to use. They realized that you need to take time in order to have time. So they sat down with a whiteboard and said, "What do we need, and what do we like to have? What will help us going forward, and what will make it easier?" A very relaxed process resulted from getting out of the firefighting mode. For example, if it took a few extra minutes to write an extra line of code to collect some more useful data, then they found taking that extra time worth it. This thoughtful approach made it easier to determine what else to collect, how often, whether to store it in the same database, and whether to establish different retention policies based on data collected.

They had a long debate on what data to collect and why. An example of this is a DBA collection:

- If you have values already set with an "OK" or "not OK" status within the script, you can identify a warning threshold and get an immediate alert on the dashboard.
- Having these metrics available in a different database allows comparing value changes across time and understanding their cause.

For example, they looked at the webserver and changed some of the website code to include output to the log that would tell them how long that thread took to load:

- They set an arbitrary value, and if the thread duration was higher than that value, the system would indicate the existence of a problem.
- That allowed their developers to know where in their application (and what the user was doing) that was causing the thread slowdown, without having to pile through tons of logs.

For Swiss Re's new monitoring solution, total implementation time and rollout — apart from the meetings of agreeing on what to monitor — was a little over a month.

Specifying data to collect and retention policy durations

- The first problem was reducing data to numbers. They realized that the first step in collecting data is determining what data they needed (logs, performance, BPM, JMX data), and how long they needed it for. The need to answer that question became the driving force for how they collected data because different teams had their own set of data to collect depending on their role and operational focus.
- The second problem was establishing hard retention times for that data. Some teams have hard fixed audit requirements (retention of several years) while others only needed data for a week (for example to track performance during a recent deployment). Identifying retention times drove them to deploy multiple databases with different retention schedules for each application, or at least a group of applications that had similar requirements.

Example of data collection and retention policy decision-making

The Swiss Re example below involved a conversation around a web admin scenario where the admin initially thought all he needed was certain logs and knowing whether he gets a 404 or a 500 response. Then, realizing he might want to measure website performance speed, the team looked back at how he was currently monitoring it. While he had performance data for the server, he didn't have any performance data for the website itself.

- As the web farm is scaled out, testing off the load balancer will only provide a generic response, which doesn't indicate which host it's necessarily coming from.
- The team decided to test this locally and see what the responses are, so that they could then send those responses on.
- They tagged the environment, and based on those tags, they know whether it's a development or a production instance and what application it's running.

This evolved from their initial example into getting status codes and web server response times. In the example, they were retrieving both the time in seconds that the app took to load and the status code. But equally, they could look at the BPM aspect and determine a response time to set a threshold. This is where a time series database such as InfluxDB is useful because they could constantly take those measurements. If the measurement goes above a value, they can find out when and examine why.

Initial example

```
$ts = New-TimeSpan -Days 0 -Hours 0 -Minutes -1
# information to look for
$pattern = '404','500'
while($true)
{
$past = (get-date) + $ts
$now = get-date
get-content /var/log/1.log | where { [datetime] $_.split()[0] -le $now -and [datetime
$_.split()[0] -ge $past | Select-String -Pattern $pattern
sleep 60
}
```

Evolved into

```
while($true){
   $starttime=Get-Date -DisplayHint time
   $status = (curl https://www.facebook.com).statuscode
   $stoptime=Get-Date -DisplayHint time
   $duration=(stoptime-$starttime)

$metrics =@{
   HTTP_Response_Time = $duration.TotalSeconds
   HTTP_Status = $status
   }
   # send to Influx
   Write-Influx -Measure Web -Tags @{App_Prod=$env:COMPUTERNAME} -Metrics $Metrics -Database BPM_DB
-Server $influxDB -Verbose
   Start-Sleep -Seconds 5
}
```

Monitoring performance metrics

Deciding how to monitor performance metrics for regular service took on a similar path. Most team members initially stated that they want to collect performance data on their usage of memory, CPU, etc., and the desire for these metrics was driven primarily by the tools already in place. So they were looking to do a one-to-one replacement. That was the starting point for them to pursue more granularity and determine which data could translate into more team productivity. What they ended up with is a performance metric that started with just two values. It evolved into the below metric (shown in part below) and into figuring out how much data they can get and what is relevant.

Performance

```
while($true)
{
$valuesList = Get-Counter -Counter "\Memory\Available Bytes","\Processor(*)\% Processor Time"
$MemoryActive = [math]::Round($valuesList.CounterSamples[0].CookedValue,3) / 1024 /1024
$MemoryFreeMB = [math]::Round($MemoryActive)
$processorActive = [math]::Round($valuesList.CounterSamples[1].CookedValue,2)

$Metrics = @{
 "cpu active" = $processorActive
 "Free RAM MB" = $MemoryFreeMB
}
```

Evolved into

```
$Metrics = @{
 "cpu active" = $processorActive
 "io pagespersec" = $pagesPerSec
 "pagesinputpersec" = $pagesInputPerSec
 "pagesoutputpersec" = $pagesOutputPerSec
 "pagereadspersec" = $pageReadsPerSec
 "pagewritespersec" = $pageWritesPerSec
 "pagefaultspersec" = $pageFaultsPerSec
 "hardpagefaultspersec" = $hardPageFaultsPerSec
 "softpagefaultspersec" = $softPageFaultsPerSec
 "memorytotal" = $memoryTotal
 "memoryused" = $memoryUsed
 "memoryfree" = $memoryFree
 "memorystolen" = $memoryStolen
 "memorypressure" = $memoryPressure
 "memorylock" = $memoryLock
 "memorylockblocks" = $memoryLockBlocks
 "memorylockspercent" = $memoryLockBlocksPercent
 "buffercachehitratio" = $bufferCacheHitRatio
 "bufferpagelifeexpectancy" = $bufferPageLifeExpectancy
 "bufferlazywritespersec" = $bufferLazyWritesPerSec
 "bufferfreeliststallspersec" = $bufferFreeListStallsPerSec
 "memorygrantedworkspace" =$memoryGrantedWorkspace
 "memorymaximumworkspace" =$memoryMaximumWorkspace
 "batchrequestspersec" = $batchRequestsPerSec
}
Write-Influx -Measure Server -Tags @{Server=$env:COMPUTERNAME} -Metrics $Metrics -Database telegraf -Server http://localhost:8086 -Verbose
Start-Sleep -Seconds 5 }
```

The above screenshots are examples of monitoring performance metrics at Swiss Re. Monitoring performance data is only kept for a day or two because they're interested in evaluating the current state while other data is kept for several months, weeks, and even years.

Technical architecture

> *"This is where a time series database comes in useful because we can constantly take those measurements. If the measurement goes above a value, we can find out when and examine why."*

Swiss Re has multiple versions of InfluxDB with various retention policies:

- They needed to put in a proxy layer to ensure that they could consistently get it updated in case they were doing any maintenance work on InfluxDB. They have multiple services from that point of view with the same databases.
- Equally, they split out some of the development and production tiers as well, not for load purposes or anything else, but just so it was easier, so that they had the ability to update InfluxDB in a development tier first before doing production.
- For some cloud providers, there was a latency delay depending on the location. Swiss Re has Azure setups in the US, and since it is Swiss-based company, this caused slight latency across the ocean. Consequently, it was easier to establish local setups.

Determining system compliance

To determine if the system is compliant, they have a set of internal policies that the system must meet. Once they build up the compliance checklist, they push that into a value to determine whether the system is compliant. They give that value a numerical status code, or sometimes, multiple status codes.

For example, the value may be compliant for the first part of the policy but not for the second part of the policy. Yet being able to create internal code numbers enables knowing which part of the policy failed, which allows very quickly and easily pushing it into a time series database and reporting effectively on your environment if it's consistent with what your policy is, or is not, every time it runs.

Handling data evolution and retention policies

These monitoring practices resulted in an evolution of data over a period of time. Swiss Re started off with a very simple, minimal principle of creating two databases with different models of retention schedules. It quickly became very clear that they needed more retention schedules, resulting in

establishing some 12 different retention policies, because the data is processed per department or per business need. They could also have as many databases as they need.

They recommend, rather than trying to force everything into one database, thinking about the retention duration and about whether there are multiple groups that will fit in the same retention policy, or if it makes more sense to split them out and create as many as needed.

Leveraging business process metrics (BPM)

In Swiss Re's IT environment, uptime is not a big issue since they have a reasonable layer of redundancy. Yet much of their activity involves a high volume of document ingestion in various document formats, and sometimes it's system to system (in JSON or XML formats). Swiss Re's systems then need to process that information. Keeping track of how many documents, or how much data per second, they are handling becomes their business uptime criteria. Slowdowns cause a backlog, so to ensure the system is performing adequately, they set performance expectation values (such as volume of document ingestion per hour) to understand the causes underlying performance problems.

Both IT teams and business process managers get access to dashboards that show relevant system performance metrics. They started to develop BPM probes (which indicate workflow status) for each application to understand if it's running, and if it's running smoothly. Sometimes they just count the number of transactions (number of documents for example) in the last period, output that, and then see near-real-time numbers on how it's progressing.

Being able to identify which process or dashboard is running slow (while the rest of the system is running smoothly) reduces escalation level and allows focusing on the cause of the slowdown. Such visibility enables performance tuning (whereby, for example, comparing performance speed between two consecutive weeks shows improved performance speed in the second week).

From reactive to proactive monitoring

Swiss Re set up thresholds for environment restarts and collected restart data in InfluxDB. Their environment restart was set up as follows:

- If the data collection fails for any reason, the server would attempt to restart, and there would be a count loop which was put into the local environment.
- If the restart fails after several attempts, the server would shut down, and restarting it would require someone to look at it.

Swiss Re even used the collection statements to force a compliance on their environment as well as collect that information also into InfluxDB:

- They established a threshold for environment restarts whereby if environment restarts exceeded it, the data could be pushed to InfluxDB, resulting in a server restart count on their dashboard.
- A higher-than-normal count during a normal week would indicate unexpected behavior and prompt a drilldown.

This approach eliminated much of the reactive nature and pushed them into a proactive state because they know where and what's going on in their environment, even though they have no agents deployed. Swiss Re were using a few scripts that are nested into their Git repository and are simply copied and run on the machine the moment they fire up a new instance.

What's next for Swiss Re?

Swiss Re has plans for using other components of the TICK Stack and has a dedicated monitoring team who is looking at expanding stack implementation in the company. They are gradually phasing out and replacing their original tooling. They currently deploy several different tools, including Datadog, for some operations, and are looking at how much they can switch over to the TICK Stack to perform those operations given its ease of installation and operation.

Results

> *"What we have ended up is a very flexible solution whereby every time we decided that we want to onboard a new metric—and this is one of the things that I do love about InfluxDB—we just point it to the DB and say, 'This is what I collect'."*

Using InfluxDB, Swiss Re's new monitoring solution enabled them to meet and exceed their monitoring goals:

Saving on storage space

The data they were collecting, with the previous application sets that they were using, was substantial (200-300 GB of data per month). Since there was only one retention policy they could set (it was global for the tool), that was problematic. Even when they did need the data, but only for a short term, they

couldn't expire it. They were consuming storage constantly for data that they used only for a week or two. So being able to separate out those data collections, and to set multiple retention policies using InfluxDB, saved significantly in storage.

Building dashboards and gaining granularity

Using InfluxDB and Grafana, Swiss Re now create dashboards that they can group together to see their monitoring "big picture" and perform drilldowns:

- They capture a tag — which could be a server name, environment, or application — and post it through. What they found works best over time, was to use an environment and a machine name. For a given application, they can easily identify the prod environment, and the host server in the prod environment.
- They have gained the granularity of being able to drill down to the individual machine in the dashboard and to collect any metric they want. In some cases, they found that they were collecting 30 or 40 metrics, depending on the application.

Accelerating new release issuing

Using InfluxDB in a container environment made it easier and faster to issue new releases:

- New releases involved having a patching cycle. When a production machine gets replaced (spin up of new containers), they would start collecting that new information. In your traditional environment (VMs or others), you need to manage every one of those agents to collect the needed data, and deploy templates where applicable. Comparatively, refreshing containers provides a lot of flexibility.
- They were even able to determine if a container is up to date (based on its build number) and accordingly push the value into the environment variable.
- They could also see how many containers in the environment were on which build number. As they refreshed containers, they could see the numbers decline. So even if they missed one, they would know that it was there.

Faster reaction time and improved customer trust

Prior to using InfluxDB, Swiss Re faced the challenge of slow reaction time. Before the data would make its way through the various integrations and generate a ticket for IT to inspect, roughly 15 minutes would have passed, by which time a user has found a problem, phoned in, and already started complaining. Now Swiss Re are down to a 10-15 second response time:

- Swiss Re started integrating this proactive monitoring process into the help desk system.
- The help desk team would get notified when the dashboard goes red as well, are aware that the IT team has already started working on it, and could thereby respond with prior knowledge of

the problem when a customer calls in to report a problem and provide that customer with the ticket number.
- Laying down the groundwork for proactive monitoring also changes the customer's perspective (who now realizes that they're proactively addressing the problem) and builds customer trust.

Because everything is set up internally today, the Swiss Re team know how their IT architecture works and can use the setup done for one department as a model for evaluating the needs of other departments.

About InfluxData

InfluxData is the creator of InfluxDB, the leading time series platform. We empower developers and organizations, such as Cisco, IBM, Lego, Siemens, and Tesla, to build transformative IoT, analytics and monitoring applications. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by sensors, applications and computer infrastructure. Easy to start and scale, InfluxDB gives developers time to focus on the features and functionalities that give their apps a competitive edge. InfluxData is headquartered in San Francisco, with a workforce distributed throughout the U.S. and across Europe. For more information, visit influxdata.com and follow us @InfluxDB.

548 Market St. PMB 77953, San Francisco, CA 94104