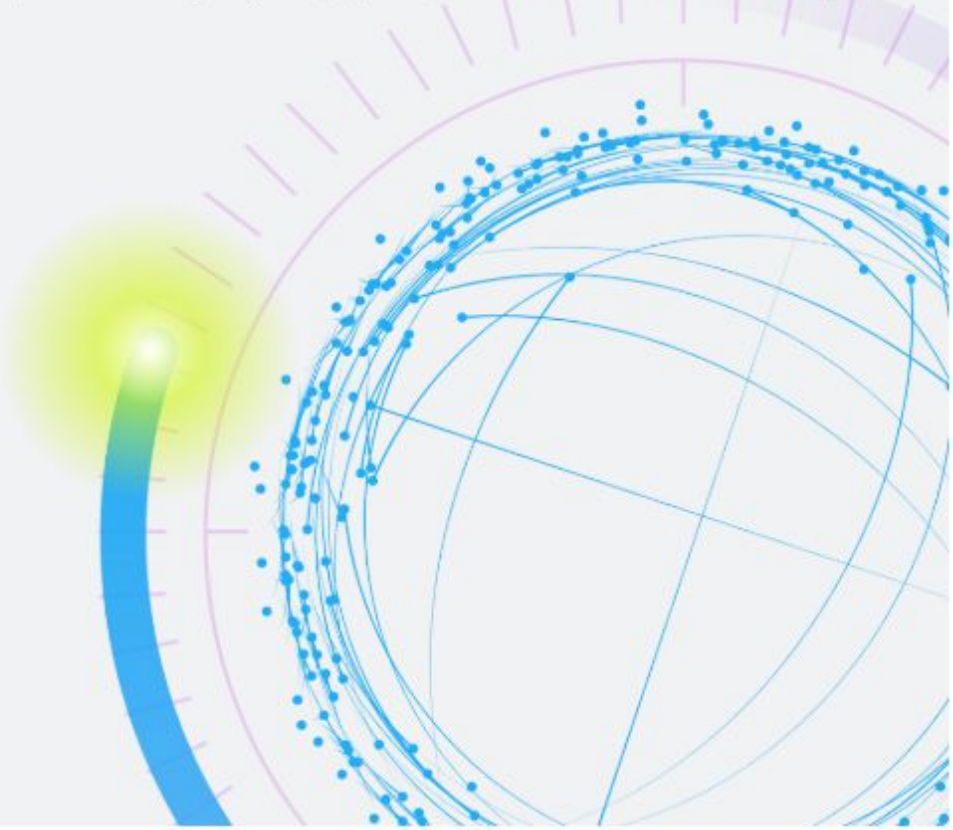
influxdata

Reducing Snowflakes with Automatic Deployments via Lighthouse, Telegraf and InfluxDB

AN INFLUXDATA CASE STUDY

Matthew Iverson, Site Reliability Engineer, Optum, a Division of UnitedHealth Group



March 2019

Company in brief

Optum is a health services and innovation company on a mission to help people live healthier lives and to help make the health system work better for everyone. Optum, part of the UnitedHealth Group family of businesses, is powering modern health care by connecting and serving the whole health system across 150 countries. Optum combines cutting-edge technology, the world's largest health care database and vast expertise to improve health care delivery, quality and efficiency.

Optum is revolutionizing health care that serves more than 100,000 physicians, practices and other health care facilities, as well as 127 million individual consumers. It powers modern health through data and analytics, health care delivery, health care operations, pharmacy care services, population health management and advisory services. Optum connects payers, providers and employers — to share knowledge, resources and insights — which leads to more holistic, innovative care.

Case overview

Optum needed a dynamic configuration automation solution that would scale with its massive data center infrastructure, flex with the diverse needs of its various teams and minimize manual work to provide a self-support model, thereby freeing its teams to focus on application optimization rather than infrastructure. Optum developed an attribute repository in-house, called "Lighthouse", to control what configurations, versions and plugins go out to each individual server deployed with any of the UnitedHealth Group data centers. As with any large enterprise, scale is a critical piece to consider with any item. Optum's automation journey involved deploying Telegraf, InfluxDB and Lighthouse at scale. Lighthouse gives Optum the ability to dynamically change items and have configurations rolled out within 30 minutes without ever touching a server.

> *"Our goal is to make this so easy that people can increase observability within their applications without having to do a ton of work to accomplish that."*
>
> ***Matthew Iverson**, SRE team lead, Optum*

The business problem

From a site reliability perspective, the goal of Optum's SRE team is to make systems and software as reliable and resilient as possible. Yet system and software performance requires observability. Developers and companies want to increase observability within their applications. Whether for delivering healthcare (to know how healthy someone is) or at an e-commerce site (to know how many transactions you have), observability is needed into dynamic data systems to ensure that they are working reliably and consistently. This observability journey might be reactive (trying to fix issues after they are reported) or proactive (installing a log forwarder or metrics collection agent such as Telegraf) to increase observability. Observability sheds light on performance and mitigates risk, just like a backup camera on a car enables observing what's behind us,

Optum began their observability journey by asking themselves: "What are we missing? What are we not seeing?" From there, they sought to figure out solutions. The aim of an observability journey is often to figure out how to increase knowledge or make decisions with automation. In Optum's case, the observability journey evolved hand in hand with their growing scalability and configuration automation needs. Optum had to tackle the issue of having a vast number of different tools and solutions while being very much a self-service company, geared at enabling their data center teams to focus on application improvement without having to worry about underlying configurations and infrastructure. Achieving that was difficult because they lacked lockdown control. They set out to help these teams make their applications more reliable by enabling automation and increasing observability.

The technical problem

To achieve configuration automation, Optum needed to reduce snowflakes (servers that require special configuration beyond that covered by automated deployment scripts). They started monitoring their systems by installing Telegraf on their services and on a few dozen of their Virtual Machines (VMs). Their teams, encouraged by the increased data collection frequency enabled by Telegraf (every 10 seconds rather than every minute or 5 minutes), started utilizing Telegraf on their VMs. They used it for data collection for various purposes, from setting alerts through Kapacitor to creating dashboards with Grafana or Chronograf.

Meeting the challenge to scale system monitoring

Once Optum started to scale to thousands of VM's, they had to rethink their architecture to make system monitoring work for tens of thousands of servers. They set up Ansible (to automate tasks via SSH connections in order to install configurations from a broad perspective). They could list 20 servers and run a script, and Ansible would install the scripts they would program it to install.

Yet the teams would do the install but not the maintenance that Optum needed. Every time Optum had to change a setting within the main Telegraf config file, they couldn't ask dozens or hundreds of business groups to go into a server (which in many cases was not their server) and make that change. This setup using Ansible didn't work or scale for Optum's purposes.

Automating configuration to reduce snowflakes

Optum switched to using Chef (a configuration management tool for writing system configuration "recipes") which runs as a service, in Optum's case every 30 minutes. Chef ensures that all config files are set the same way and that services are running as intended.

With Chef, the user writes "recipes" that describe how Chef manages server applications and utilities and how they are to be configured. These recipes (which can be grouped together as a "cookbook" for easier management) describe a series of resources that should be in a particular state: packages that should be installed, services that should be running, or files that should be written.

These various resources can be configured to specific versions of software to run and can ensure that software is installed in the correct order based on dependencies. Chef makes sure each resource is properly configured and corrects any resources that are not in the desired state.

Chef worked well for Optum. They decided to create a Chef cookbook, so their teams can apply it to their servers and thereby have what they need. Then they thought that if they needed to change a config file, they would update it in Chef, and that gets applied to all of the servers using their cookbook.

The problem is that teams started requesting settings and actions, from a Telegraf perspective, that Optum's current setup couldn't support. For example, some users had a specific collection rate, needed a different config, and different zones existed between the DMZ and core network of various data centers (depending on where the server lives, different configuration files should be used).

Optum realized that while it's possible to map out configurations in a file that gets uploaded to Chef, this practice would not be maintainable. So they decided to figure out how to dynamically assign server attributes and variables for these Chef runs without having to upload a multi-thousand line JSON file mapping certain servers to configs. They created a proof of concept, which then turned into the attribute repository that they call Lighthouse.

The solution

> *"Our people don't want to worry about how the data gets from A to B. They just want it to be where it's going to go and be able to query it in real time. This is why we designed Lighthouse — to gain the ability to dynamically change these attributes."*

Why InfluxDB and Telegraf?

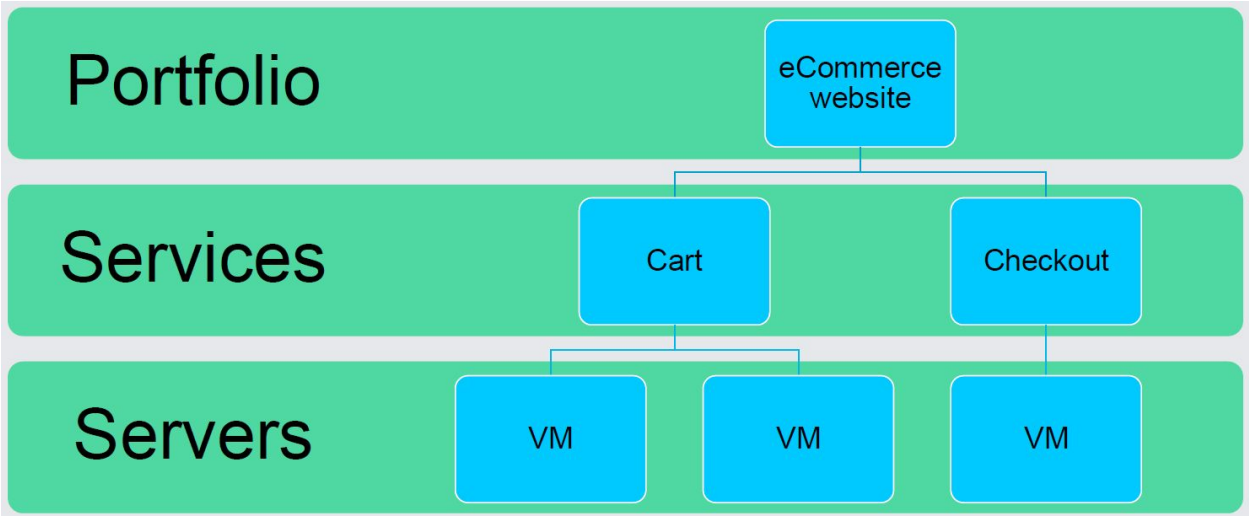
Optum uses the InfluxDB time series database to ingest monitoring data (which is time series data) given InfluxDB's purpose-built design, high write throughput and scalability.

With Telegraf, Optum wanted to collect data in a way that's so easy that they would have no reason to use a different metrics collection tool. They wanted to set Telegraf to collect all the data their teams needed, enabling them to act on it as they needed (such as for alerting or sending it to a different platform). This required the ability to assign attributes dynamically, using Lighthouse as their attribute repository.

The purpose of Lighthouse is to store different pieces of information for different groups of servers. When scaling from tens to thousands and hundreds of thousands of servers, Lighthouse gives Optum the capability to store attributes for groups of servers and to dynamically add attributes.

Yet if Optum needed to change a config, such as an item for a server, they would have to update it in a lot of records. Each server was a record in the database as a row. They realized that looping through all the records doesn't make sense when all the servers will use the same attribute. So they created a nested hierarchy: portfolio, services and servers.

Lighthouse Attribute Structure



There are multiple portfolios within an organization (ecommerce website, HR website, etc.) and multiple services within a portfolio. For example, an e-commerce website requires a checkout and cart, and has dedicated VMs that tie to these different pieces. In order to scale, instead of trying to assign attributes to just the server level, Optum decided to assign attributes to the server, service, and portfolio levels. That way, from a nested perspective, they could have the attributes that they assign get automatically distributed across all of the tiers beneath them within the hierarchical structure.

Developing Lighthouse turned out to be an evolving process during which Optum received requests and underwent trial-and-error periods. Once built, Lighthouse allowed Optum to change configurations very quickly and dynamically since their organization has thousands of VMs spun up and spun down every day. They set up Lighthouse to be queried API-first. That way, users can query it, log in and view configurations through the front-end that Optum built.

Technical architecture

> *"How do you make this so easy? That was our goal – we want this to install Telegraf and use InfluxDB. We want this to be so simple that you don't have to do any work other than requesting that the Chef cookbook get applied to your server."*

Lighthouse is written in Ruby, and the attribute data is stored in a MySQL database because they wanted it to be able to be changed quickly and dynamically when needed. The data flow encompassing Chef and Lighthouse is as follows:

- Every 30 minutes, Chef runs on a given server.
- Chef is an open API, so at the beginning of its run, it calls Lighthouse and obtains a list of attributes that Lighthouse has (by merging in the attributes it gets for the given server, service, and portfolio).
- Then Lighthouse grabs all these attributes through the API and merges them in with the Chef attributes that are in the cookbook by default.
- With a Chef cookbook, you set numerous attributes, but Optum didn't want to have to upload a new cookbook every time they wanted to change a variable. Their system calls Lighthouse and overrides its default values by merging in the differences between the servers, services, and portfolios. This gives Optum a highly dynamic environment.

For example, they might want a specific server to have Telegraf version 1.9.0, the service to have 1.9.1, and the portfolio to have another version. If they have a service and know that they can't run the newest version of Telegraf, they can set that at that service level — that will allow running those versions without having to manually manage any of these configurations. From a scale perspective, this setup worked well.

Using Lighthouse to manage Telegraf configurations

Optum considered how to apply and scale Lighthouse functionality to other system components, such as Telegraf, so that they could set attributes for it. But now they had other teams looking at log collection. To set log locations for tens of thousands of different servers, the same premise applied: setting that the server, service, or portfolio level. For instance, if an ecommerce website's shopping cart has many servers, these servers are likely to have their logs in the same location. Rather than set the log location at the server level, you can go up a tier and set it at the services level.

This tiered assignment of attributes gives Optum flexibility in how to manage and maintain Telegraf because they want their system to scale for the entire enterprise and to work for everybody, not just a small subset of people, especially when they considered the benefits of being able to increase observability by collecting metrics every 10 seconds. Some of their teams have tools in place that collect metrics, some don't, and various teams have various engineering talents.

The Lighthouse attribute payloads, written in JSON script, are shown below.

Lighthouse Attribute Payloads

```
{"telegraf":{"version":"1.9.0",  "package":"installed"}}
{"telegraf":{"version":"1.9.2",  "service":"stopped"}}
{"telegraf":{"package":"purged"}}
```

```
{“telegraf”:{“interval”:“60s”}}
```

These payloads allow Optum to specify any single variable entered into a config and dynamically change settings at the service, server or portfolio level. There's also a default category which gets applied to everything. They can specify whether the package is installed, whether the service is running or stopped, or delete all their configs.

One reason why they built Lighthouse to encompass the four above-mentioned tiers is that if they had an application that's not performing correctly and they're worried about decreasing CPU on that box at any cost, they can update the JSON payload inside Lighthouse from service “running” to service “stopped”. Then, when Chef checks in (every 30 minutes), it will automatically stop that service. When ready to turn Telegraf back on, all they have to do is flip the setting back to “running”. This gives them the ability to dynamically change settings in a quick fashion.

A queueing system to handle back pressure

When they want to change how they connect with their environment (encompassing thousands of servers), they use RabbitMQ message queuing system to handle data back pressure, rather than doing the writes directly to InfluxDB. RabbitMQ gets data across to InfluxDB and handles the queue in case of connectivity issues or an InfluxDB outage.

For example, since they had users utilizing Telegraf, Optum started enabling additional input plugins because they want to create a self-support model. They had a team that enabled the MySQL input plugin (available as version 1 and version 2, which are incompatible with each other). If the data is already in InfluxDB and the same measurement name is used, this creates conflicts. The team had enabled the MySQL input plugin without first checking for potential conflicts. Optum saw write errors when writing to InfluxDB. To troubleshoot, all they had to do was see where the hosts were coming from and flip the flag to “stopped”, thereby shutting off these services. Telegraf was shut off on these servers while they figured it out.

Setting the queuing system’s credentials inside the JSON payload enabled Optum to rotate the credentials as often as they like. That way, their teams don't have to worry about the credentials or about which RabbitMQ server to connect to. They just focus on collecting the data they want and on using their preferred plugin.

Nested attributes enabled Optum to set different version numbers, for any attribute, at different levels. Additionally, they could do broad changes at the default or portfolio level to a different version. Within 30 minutes, all their Telegraf installations will be running that version. In the same manner, they could also start, stop or change the collection jitter or any other adjustable settings within the config files.

Nested Attributes

```
{
   “default”:  {   “version”: “1.9.0” },
   “portfolio”: {  “version”: “1.9.1” },
   “service”:   {  “version”: “1.9.2” },
   “server”:    {  “version”: “1.9.3” },
}
```

Only attributes that need to be overridden are set, and not necessarily all attributes. For the thousands of servers in Lighthouse, “version” is not set on any of them, and if it is, it would be for temporary purposes. If an attribute needs to be reversed, the server gets precedence over a service, which gets precedence over a portfolio.

In Optum’s automated configuration deployment, a given server knows that it belongs to a given service and knows the attributes it's supposed to have. Optum know what servers are going to be onboarded from a Telegraf and a Lighthouse perspective, so they add them and add the relationship before they get onboarded. Optum’s long-term plan is to allow anybody to request server onboarding at any point. They will do integrations with their IT service management (ITSM) tools to figure out that a given server belongs to a given Continuous Integration (CI) and ensure that the hierarchy is automatically generated.

What’s next for Optum?

Optum has default plugins that get enabled. From a Lighthouse perspective, they want to try to automatically enable more monitoring. Telegraf has over 200 input plugins, enough input plugins for every service Optum’s teams are running. When they run Chef, they plan to look at the different services and figure out what's running.

For example, if they have HAProxy running on a given server, they realized they don’t have to do anything manually since they can report that back to Lighthouse and since Telegraf has an HAProxy Input Plugin. Optum thought of notifying the relevant team that one of their unmonitored services could be monitored via Telegraf. Then they realized that there is no need to even ask the team if they want to use the plugin. If they know they can connect to and monitor it, they could simply the HAProxy Input Plugin automatically.

Since the HAProxy example could apply to all available Telegraf plugins (MySQL, Elasticsearch, HAProxy, Apache and many others), they set out to create a list of enabled plugins and store it within

Lighthouse so that their teams can automatically enable more monitoring and easily increase their services' observability.

Optum also wanted to allow people to get access to their data that is stored in InfluxDB, without having to manually create credentials, but rather by enabling them to subscribe to their data. They can do that in Lighthouse because they're managing everything with Chef.

For example. Optum has a team seeking to fork telemetry data and send it to one of Optum's logging clusters. Optum's perspective is that if that team has a use for such data in a logging cluster, they want to enable them to do that. Part of the platform's purpose is to serve Optum's self-support model through automation so that their teams can utilize it for monitoring. Optum also want to benefit, for their platform, from a development in progress at InfluxData. InfluxData is in the process of providing a way to tie one's Telegraf installation in with a URL. That way, the Telegraf config can be managed within InfluxDB and through the UI tier.

Wanting to benefit from growing container adoption as well, Optum considered how to apply Lighthouse to containers, but realized Chef doesn't necessarily run inside of a container. So the next logical step was to mock-up what the InfluxDB API will do from giving that configs, and they could then generate those configs within Lighthouse. So when a container starts, Optum can give the configs a unique URL. So when the container starts their sidecar, they'll call Lighthouse and get all the configs that they need for their application. This allows them to:

- Do the tagging (shown above) from a Grafana perspective
- Figure out how to expand Lighthouse to be the self-service tool for enabling monitoring in telemetry data

As for rolling back configs, the current version of Lighthouse released in production does not include that functionality, but Optum has plans for creating an audit log. This is particularly useful for large enterprises, which have security and compliance teams that require documentation and controls. Audit logs would enable Optum to know what attribute changed on what server, portfolio, or server — and *when* that change took place, which then allows switching back multiple versions.

Optum is planning to open-source Lighthouse so the community can use it to manage their infrastructure. From a configuration management perspective, this could change how infrastructure monitoring operates given the mindset difference between having a recipe (or anything static that does a run) versus changing it to be dynamic.

Results

> *"Lighthouse gives us this powerful flexibility to control the environment but still allow people to be enabled so that we can increase observability from an organizational standpoint."*

Building Lighthouse, coupled with using InfluxDB and Telegraf, has allowed Optum to enable its people and to increase observability from an organizational standpoint. The below Grafana dashboard shows how this architecture helps Optum in practice.

- At the top, the dashboard shows the variables they have set (such as environment, portfolio, service, etc.). This gives teams the capability of creating a standard dashboard (without having to say for a specific team); selecting their relevant portfolio and service; and seeing their metrics without having to manage another Grafana dashboard. If teams want to fork the data and use it in their desired way, they absolutely can. Optum wants to enable its teams to use this platform but need it to be simple enough to give teams the option of not having to manage dashboards, especially since dealing with multiple data centers in multiple zones requires the ability to drill in quickly to understand what is going on.
- Taking the data from a Telegraf perspective was critical because Optum's hostnames didn't provide info on where the server is and what it does. When Lighthouse is called, it generates the tags inserted into the telegraf.com file to indicate server role or server data center and zone. For example, if half of the organization's servers are low on CPU and half of them are high, their dashboard can show which ones are which (see the "DataCenter" dropdown in the

above graph). This gives teams the flexibility to gain observability without having to manage Telegraf or run their own InfluxDB instance.

- Optum can also select whether to see production or staging, and the option of Grafana dashboards for various environments. All this gives them that flexibility to change items without managing them.

Optum has met its goal of making Lighthouse a self-service solution, All their teams have to do is add a Chef recipe, add a Chef cookbook to their server and within 30 minutes they get metrics in Grafana and in InfluxDB for them to look at. From there, they can decide what to do, such as creating alerts or creating a Grafana dashboard. Through their observability journey and by continually updating their automation architecture, Optum enabled their teams and made it easier for them to collect data, resulting in further benefits:

- The easier it is to collect data, the higher the chance that they'll collect it.
- The more data that they collect, the higher the chance that they can increase observability, and with that, the reliability of a system.
- System reliability can increase the satisfaction of customers (who are not necessarily purchasing anything in Optum's case but sometimes use their service by default, such as a public utility, healthcare, or ecommerce site). To create such reliable systems that achieve user satisfaction, observability into the full infrastructure is necessary.

By building Lighthouse and using InfluxDB and Telegraf, and from an infrastructure performance perspective, Optum supported the company's broader mission of knowing how to improve healthcare by actually being the "how".

About InfluxData

InfluxData is the creator of InfluxDB, the open source time series database. Our technology is purpose-built to handle the massive volumes of time-stamped data produced by IoT devices, applications, networks, containers and computers. We are on a mission to help developers and organizations, such as Cisco, IBM, PayPal, and Tesla, store and analyze real-time data, empowering them to build transformative monitoring, analytics, and IoT applications quicker and to scale. InfluxData is headquartered in San Francisco with a workforce distributed throughout the U.S. and across Europe.

Learn more.

InfluxDB documentation, downloads & guides

Download InfluxDB
Get documentation
Additional case studies
Join the InfluxDB community

799 Market Street
San Francisco, CA 94103
(415) 295-1901
www.InfluxData.com
Twitter: @InfluxDB
Facebook: @InfluxDB