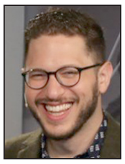
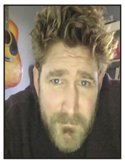
iVoyeur
Sensu Rising: An Interview with Matt Broberg

DAVE JOSEPHSEN


Matt Broberg is VP of Community for Sensu, Inc., focused on the incredible community around Sensu, the open source monitoring framework. Matt is on the board of the Influence Marketing Council, co-maintains the Evangelist Collective, contributes to the Go Community Outreach Working Group, occasionally blogs on Medium.com, and shares code on GitHub. He's also a fan of tattoos, rock climbing, and cats, though remains unsure of Schrödinger's.


Dave Josephsen is a book author, code developer, and monitoring expert who works for Sparkpost. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

It's hard to believe that Sensu, the open-source, distributed monitoring framework, is over seven years old. Its scalable, ultra-flexible design and practitioner-focused development model still make it the most forward-thinking centralized poller in existence. The project is one of the very few that still "feels" fresh to me, and yet it retains that aura of bullet-proof resiliency that only comes with time in the trenches.

In recent months, the project founders have incorporated to form Sensu, Inc., hiring on a dream-team of people I personally adore, including engineers Greg Poirier (Opsee) and Jason Dixon of Graphite fame, as well as ace community architect extraordinaire, Matt Broberg. The fledgling corporation is busy funding and managing Sensu community development, providing enterprise Sensu support, and laying the groundwork for the future in the form of Sensu-Go, a ground-up rewrite of Sensu in the Go programming language.

Given all the exciting change happening in the Sensusphere, I thought it'd be fun to interview Matt and get a feel for what's going on from within.

Dave Josephsen: Describe Sensu in your own words.

Matt Broberg: Sensu provides total visibility for your business, from the server closet to the cloud. Said simply, Sensu connects the dots between every tool in your monitoring solution, providing a single way to manage service checks, telemetry, alerting, and remediation, and it gives you the right primitives to build custom monitoring that scales.

DJ: Who is using Sensu today? How many teams? How are they distributed with respect to industry?

MB: It's helpful to recall that Sensu has been around for seven years, while Sensu Enterprise has nearly three years (full story at [1]). With over 13,000 downloads a day of Sensu Core packages, we know there are more users than the team behind Sensu, Inc. has gotten to know, and we look forward to discovering them. Shameless request: if you're a current user, I'd love to hear from you: community@sensu.io.

Talks from last year's Sensu Summit are a great cross-section of our user base. We have companies large and small, off- and on-premises, running in every environment from bare metal to Kubernetes to AWS. You have folks like GoDaddy scaling a self-service Sensu environment with 40,000 clients spread throughout their globally distributed datacenters. Spin up a box, get base knowledge about your environment out-of-the-box, and then help your product team customize it all.

Then you have an architect at T-Mobile talking about Sensu monitoring their Cloud Foundry environment. Nagios migration is a very common use case for us. Sometimes it seems like everyone at the summit has a story about migrating from Nagios at some point due to scaling or customization challenges. My personal favorite comes from David Schroeder who goes into the detail of the how and why he needed to move on [2].

Like any healthy open core model, we have a majority of users successfully running on their own with the MIT-licensed open source version. A healthy majority of users are open source, using our large library of plugins filled with service checks and telemetry collectors, or running their pre-Sensu plugins for Nagios, or pushing data using the StatsD extension. Some significant OSS shops, like Yelp or TripAdvisor, have open sourced tools that make Sensu even easier to run. One of my favorites is Sens8, which extends Sensu functionality to fit smoothly into Kubernetes. Schuberg Philis [3], for example, has a great blog about the custom code they're running to monitor 20 Kubernetes clusters with Sensu and Sens8.

We have a growing number of Sensu Enterprise customers as well, who get the benefit of enterprise-y integrations (ServiceNow and Jira are popular) along with support and training. It's a perfect choice for those who want all the pieces of Sensu put together for them so they can focus on introducing monitoring to their teams. These companies run the gamut of company size, from those as large as GE, who use Sensu to monitor Predix infrastructure, to smaller organizations like David's I mentioned above. And what I personally love is that many of them are contributors to the community, answering questions for new users or by sharing plugins.

Sensu 1.0

DJ: Compared to other monitoring tools you generally compete with, what are Sensu's particular strengths?

MB: Sensu's strength is how well it meets the challenge of monitoring dynamic infrastructure. Whether you run on bare metal, hypervisors, container orchestrators, or clouds, no matter how short-lived or ephemeral, it all works with Sensu. Our client runs on all operating systems, everything from Windows and Linux to Mac OS, and we can ingest and emit monitoring data with any purpose-specific monitoring system out there.

Scalability is a big win for Sensu as well. Our clients participate in a pub/sub relationship with a scalable transport layer which, in turn, communicates with a scalable server layer. If you have more infrastructure to monitor, spinning up more Sensu servers linearly scales your processing of checks. Dynamic client registration has historically been a bit of a thorny problem in the monitoring world, but it's been a solved problem for Sensu from its inception. The Sensu client's dynamic self-registration (and deregistration) hits home for those who have felt the pain of getting alerts for servers that were deprovisioned long ago.

The last major category of users that I know will love Sensu are used to monitoring tools that only work when you use them in the exact way they were intended. Configuration of those tools becomes unmanageable as soon as you leave that happy path, and our users love customization. Sensu has sane defaults, but will also never give you that feeling of being limited. Its API exposes the right primitives to let you build the monitoring you need with all the event handling, filtering, and automation of bits you can imagine.

DJ: We understand Sensu was architected from the ground up with a scalable distributed architecture model. Can you give us a rough idea of what the architecture looks like? For example, what are the primary components of a Sensu install, and how do they work together to achieve visibility?

MB: The Sensu client is our heavy lifter; it collects measurements. Typically, it runs on the instance you want to monitor, but it can also interrogate remote entities like switches or act as a process-local endpoint to receive, for example, thread-level metrics from a locally running app. Clients can also cooperate with other clients to achieve summarization or route messages, commonly around things like firewalls.

Clients self-register with the Sensu transport layer, which, by default, is a RabbitMQ Queue. Clients use the transport layer to publish their check results to the Sensu server and consume new check requests from Sensu server or events from other clients.

The Sensu server orchestrates service checks by publishing check requests to, and collecting service check results from, various clients via the transport layer. Nagios style (OK,WARN,CRIT,UNKNOWN) checks, as well as metrics and telemetry collection, happen by the same means, via messages passed through the transport layer. The Sensu server stores its state in Redis, performing roughly a single write operation per check result. Every Sensu server in the installation uses the same Redis state DB, ensuring that each individual Sensu server is, itself, stateless.

Most users persist data beyond Sensu, which fits perfectly with the design. It was a design goal for Sensu to easily and cheaply route telemetry or check results to external time-series databases like Graphite, Librato, and InfluxDB or store output to logging platforms like ElasticSearch.

DJ: Sensu ships with a very nice RESTful API. What sorts of operations are available from the API, and how do your customers use it?

MB: One of Sensu's greatest strengths is its RESTful API, making all of the data captured by Sensu accessible via HTTP. This API-first approach is a huge win for those living in dashboards; users can query for everything from current events (i.e., incidents) to registered Sensu client information. The fact that everything—from the Uchiwa dashboard and CLI to third-party dashboards like Grafana—uses the same API to communicate provides a single authority to keep results consistent.

Beyond the dashboard, you have a ton of options when you think about the API. Customers are a quick curl loop away from silencing alerts or getting a snapshot of the client health. There are endless ways users can combine API calls to flesh out runbooks, then add links to them to your check results with a custom attribute. The world is your oyster.

DJ: Many monitoring systems are protective with the monitoring data they collect and inflexible with respect to exporting that data to other systems. Can you talk a little bit about Sensu's philosophy on interoperability?

MB: What personally attracted me to working at Sensu was the story of how we intentionally fit into a best-of-class set of monitoring software instead of trying to be everything to everyone. Sensu wants to help you build the monitoring pipeline you need. In most cases we can natively ingest check results from your existing plugins in foreign data formats (Nagios, StatsD, and now Prometheus through extensions) and output to an ever-growing litany of other monitoring systems (Graphite, OpenTSDB, Metrics 2.0, JSON, and more).

Through our pluggable architecture, you choose where your data lives: your favorite TSDB, SaaS, S3 buckets, or anywhere else. Same goes for on-call or escalation management through OpsGenie or VictorOps or otherwise. Sensu makes sure it gets there. You get to decide where it goes.

DJ: Can you give us a rough idea of how hard it is to migrate to Sensu from an existing monitoring system (like Nagios)?

MB: It's as easy as running an existing Nagios check in a Sensu check config file. If you have an existing plugin for Nagios, maybe through `yum install`, and if you want to run it in Sensu, you deploy Sensu—ideally through your favorite configuration management tooling—and then wrap the command into a check definition under the 'command' attribute. Ta-da, you're done.

Sensu 2.0

DJ: We understand Sensu2 is available via GitHub at https://github.com/sensu/sensu-go. What is the release date, and what is the current status of the release candidate?

MB: Sensu Core 2.0 is in an Alpha state as of today, making it the perfect time to dive in to make sure to get your feedback in to guide the user experience. I recommend spinning it up on a non-production environment and seeing how it goes. When we hit Beta, we'll have a fully documented API and some larger-scale test cases to point to for performance expectations. Our official release target is for later this year, but we are committed to production readiness being a gate to GA, not a date.

DJ: We understand Sensu 2 is a from-scratch rewrite in Golang. Can you talk about what prompted the rewrite and share any top-level goals you set out to accomplish?

MB: While extremely resilient and powerful, Sensu 1.x's dependencies are numerous. We knew an adjustment was necessary to get to where monitoring needs to go to manage bare metal alongside container and serverless workloads. Moving from the external dependence and runtime requirements to a simple two-binaries-and-you're-done design will have a major impact to ease of deployment.

Go, as a language, offers clear advantages for that future as well. Concurrency is straightforward with goroutines, and many features that are seen as advanced in other languages are baked into Go's suite of tools, like race detection, testing, and performance analysis to name a few of my favorites.

In recent years, Go has established itself as the new language of systems programming. Because of this, our users are increasingly learning Go as part of their development toward an SRE skill set, making it even more essential to ensure community participation. Go's popularity and growing community provide a wealth of shared knowledge and understanding. The language and its documentation are welcoming to this new, growing segment of users coming from higher-level interpreted languages and frameworks in Ruby and Python.

DJ: We were excited to hear that the datastore in Sensu2 will be changed from Redis to etcd. Has your experience with etcd been positive so far?

MB: Etcd has been a powerhouse of a datastore. Sensu backend will have etcd embedded for clustered state and configuration management, replacing the state that Redis managed and the configuration files that used to live on disk. It's exciting to get a highly available Sensu backend that has the thorough testing that etcd gets in order to support large-scale Kubernetes deployments. That's a slam dunk for us.

We've also really enjoyed our interactions with the team working on etcd. We've contributed a few patches and have seen turnaround on bug reports in just a few days. The last fix we worked on with them was in a release only two weeks after the bug was filed and fixed on master. It's wonderful to be part of that community as well.

DJ: To what extent, as a user of Sensu2, will I need to be an etcd adept? Is its presence completely abstracted away to the point where I don't even know it's there? Or will I be expected to perform light maintenance/tuning?

MB: Our goal with 2.x is to abstract project dependencies so users can focus on the goal of monitoring the right services. If we can stick to sane defaults and avoid exposing config we don't need, it will be for the better. Maybe that's idealistic and we'll need to allow users to tune etcd's configuration, but we'll cross that bridge when it's warranted. The core team is open to adapting as we continue to test in larger environments.

DJ: Given that the datastore is changing, is the transport layer, RabbitMQ, also changing or possibly going away entirely?

MB: Yes, it is. RabbitMQ is a great piece of technology, providing pub/sub messaging with queueing and several routing topologies. It also does way more than Sensu ever needed it to do. With Sensu 2.x, we've implemented a built-in messaging transport, greatly simplifying Sensu's architecture, while still having the key capabilities that RabbitMQ provided. Given that we only ever scratched the surface of the power of RabbitMQ, it made sense for us to simplify the architecture and build the little bit of queueing we need. Fans of simpler architectures will be happy to know we have the same pub/sub model without additional services to run.

DJ: What will the upgrade path between Sensu and Sensu2 look like? Can Sensu2 process events from an in-place Sensu client? Is Sensu2 plugin-compatible with Sensu? API-compatible?

MB: The team at Sensu, Inc. and the many community contributors are committed to making the migration path as simple as can be. That said, this major release will be a breaking change in a few ways. New Sensu clients, backend and dashboard, will be deployed during installation. They will no longer need RabbitMQ and Redis services alongside them, so these can be spun down as part of installation as well.

Most importantly, all your existing plugins will run on both versions of Sensu. We will have runtimes available for download so you can pick up an embedded Ruby environment to keep your plugins up-and-running. Client config and plugins will need to be deployed alongside the new binaries. Our main focus will be to release updated Ansible, Puppet, and Chef code to enable the majority of users to painlessly deploy Sensu Core 2.0. For others not living the infrastructure-as-code paradigm, we will have upgrade readiness guides and CLI tooling to ease the transition.

DJ: Where can I hang out with the Sensu community at large and/or perhaps contribute to the development of Sensu2?

MB: We would love to have you get involved. We have a #sensu2 channel to talk through user experience and give live feedback in our Community Slack (and yes, you can sign in using IRC!). For the day-to-day banter of software development, join the #core-dev channel. If—or should I say when—you have a great idea or a new bug to share with us, get involved on GitHub at https://github.com/sensu/sensu-go. Star the repo to let us know you're interested, or watch it to get regular notifications every step of the way. If you prefer the highlights over the details, sign up for our newsletter for regular updates.

References

[1] Caleb Hailey, "From Open Source to Open Core: The Hard Way," The Sensu Blog, May 2, 2016: https://blog.sensuapp.org/from-open-source-to-open-core-bc3007c96236.

[2] Sensu Summit 2017, YouTube: https://bit.ly/2GEcWsR.

[3] Andy Repton, "Our Journey Implementing Sensu to Monitor Kubernetes in Production," The Sensu Blog: https://blog.sensuapp.org/our-journey-implementing-sensu-to-monitor-kubernetes-in-production-5764aff2dd50.