



Sandpiper (/spaces/127/sandpiper) ▸ Discussions (...)

Private Space · Technology Standards (/spaces/9/technology-standards/technologyhome)

(/spaces/127/sandpiper)

Activity Stream (/spaces/127/sandpiper/?act=1)

People (/spaces/127/sandpiper/people)

Content ▼

Posted in: Development (/spaces/127/sandpiper/forums/5366/development)

Technology Stack

✉ Unsubscribe from this discussion

📡 Subscribe to RSS (../..../..../spaces-cf/forums/rss-space-posts.ashx?

spaceID=127&topicID=5368&forumID=5366&key=rrer%2B1xDo%2BlxpC7jBRbM5w%3D%3D)



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



12/4/2019

Introduction

There are several popular technology stacks available today for web development. Due to some fairly unique requirements, however, some choices may be a better fit for Sandpiper than others. This is not your normal web-site application.

Regardless of the selections made, it's important to remember that we're creating a Sandpiper "reference implementation". Anyone is free to use the published API documents to implement Sandpiper on their own.

Overarching Goals

1. **Open Source.** There should be no requirements for proprietary tooling or commercial licensing.
2. **Mainstream Technology.** Nothing is forever, but we should make choices that have a good track record and a better than average expectation of continued support (in the open-source community).
3. **Minimum Requirements.** We are not re-creating Twitter. There's no reason to create a micro-services architecture with load-balancing, memory-caching, service discovery, reverse-proxy and the like. We don't even need a separate web-server (e.g. IIS or Apache). Less is more.

4. **Cross Platform.** Some IT departments may not want to support a new operating system in their environment. For this reason, we should allow deployment on both Windows and Linux systems. This requirement may be less important now, however, with the advent of Docker and cloud platform-as-a-service (PaaS) offerings.

5. **Simple Deployment.** The system should include automated deployment as much as possible. We envision being able to "clone" from GitHub, run a script, change some configurations and launch.

6. **Simple Upgrades.** We expect equally simple version upgrades. There should be no reason not to stay on the latest release. We should avoid any 'breaking changes' within major versions (using "semantic versioning_(<https://semver.org/>)").

7. **Security.** We should protect against known attack vectors (e.g. man-in-the-middle, cross-site scripting, sql injection, etc.). Sandpiper does not store credit card or personal data, but there is intellectual property that needs protection (e.g. customer pricing and OE interchanges). It could be that DDoS attacks are more of a risk, but protecting against network attacks is out of scope and better handled by DNS services (e.g. Cloudflare).

8. **Test Coverage.** We expect to include automated testing (with mocking) for as many functions as possible on the server.

Application Layers

Since each software layer has its own requirements (and so technology choices), we will discuss each one separately.

Server-side

The primary responsibility for the server process is to sync data-objects between trading partners. Everything else is configuration. This means we should focus our attention on creating a bullet-proof, high-speed data sync process. Secondly, we need to define an Admin API for organizing and managing these data-objects (via subscribers, subscriptions & slices) and reporting/tracking

Data Persistence

While there are more choices than ever for storing data, we will focus on traditional relational database systems (RDBMS). We want to enforce referential integrity at the database level, especially since Sandpiper may not be the only app using it (although we expect external systems to use the exposed API). The schema is fairly simple, and there are no multi-tenant or multi-schema requirements that we see at this point.

Client-side

There is an Admin dashboard requirement for managing the publish-subscribe model. Although not a requirement, it would also be useful to display usage statistics in a graphical way.

Selections

We will discuss technology stack choices (and justifications) in the comment section.

technology (/spaces/127/sandpiper/searchresults?keyword=%22technology%22&searchtags=1)

👍 Like

Reply (/forums/post?tid=5368&ReplyPostID=5369&SpaceID=127)



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



12/4/2019

Server-Side Development

We considered several languages and frameworks for the server-side development. These included:

- Ruby (grape- (<http://www.ruby-grape.org/>) ap (<http://www.ruby-grape.org/>))
- Ruby on Rails (api mode (https://guides.rubyonrails.org/api_app.html))
- PHP (Laravel (<https://laravel.com/>))
- Node.js (Feathers (<https://feathersjs.com/>))
- Golang (Go (<https://golang.org/>))

In the end, we selected **Go** for the following reasons:

1. Open source, with strong ecosystem (<https://github.com/avelino/awesome-go/blob/master/README.md>).
2. Mainstream (developed at Google). Mature and stable (<https://golang.org/doc/go1compat>). (10 years old (<https://blog.golang.org/10years>)).
3. Strongly typed (<https://blog.ankuranand.com/2018/11/29/a-closer-look-at-go-golang-type-system/>). (errors caught at compile time)

4. Zero dependencies. Runs natively on Windows and Linux without a runtime.
5. Very performant (<https://eng.uber.com/go-geofence/>). (with true concurrency. (<https://golangbot.com/goroutines/>))
6. Native http (<https://golang.org/pkg/net/http/>) handling (no need for a webserver)
7. Built-in testing (<https://golang.org/pkg/testing/>) framework
8. Great support for websockets and RPC (<https://www.gorillatoolkit.org/>).

It really came down between Feathers and Go, but in the end, we didn't want to use typescript (<https://www.typescriptlang.org/>) on top of javascript just to get type safety (why would you not want a compiler to catch errors for you?).

Also, we're wary of the latest-shiny-thing. Not that Node is going anywhere, but Feathers could easily fall out of favor in years to come (as can happen when a non-paid maintainer loses interest).
"WARNING: This repository is no longer maintained".

And, finally, the dev team had good experience with Go on other projects.

Spoiler alert... we like javascript for the front-end.

golang (/spaces/127/sandpiper/searchresults?keyword=%22golang%22&searchtags=1)

server-side (/spaces/127/sandpiper/searchresults?keyword=%22server-side%22&searchtags=1)

👍 Like

Reply (/forums/post?tid=5368&ReplyPostID=5370&SpaceID=127)

Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)



12/5/2019

Doug, your thoughtful analysis is greatly appreciated! Preserving your thought process and rationale here for posterity will pay dividends down the road for sure.

On the subject of "client" vs "server" (in the http sense): In your mind have we completely settled the question of roles? Are you ready to say *"The Content author/authority is always represented by a Sandpiper server and the content receiver is always represented by a Sandpiper client"*

👍 Like

Reply (/forums/post?tid=5368&ReplyPostID=5376&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



12/5/2019

"The Content author/authority is always represented by a Sandpiper server and the content receiver is always represented by a Sandpiper client"

I feel this is the cleanest design. That's not to say the setup and Admin screens can't hide this difference, though. Maybe there's a configuration setting like:

```
sandpiper-role: "sender", "receiver" or "both"
```

One thing to remember is that the "client" is not a browser (as in most web apps). It's just another service running somewhere, happily chugging along allowing machine-to-machine data syncs.

```
sandpiper-role (/spaces/127/sandpiper/searchresults?keyword=%22sandpiper-role%22&searchtags=1)
```

👍 Like

Reply (/forums/post?tid=5368&ReplyPostID=5377&SpaceID=127)

Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)



12/9/2019

I like Go as an implementation better than Node. Node in particular has a very heavy implementation, is not self-hosting, doesn't easily allow stand-alone builds without pulling in tons of ancillary functionality, is reliant on a working stack on each platform, and npm has had some recent high-profile security issues around the pool of packages.

The downside to Go is Google, but as you said, it's been going (hah!) for long enough that it has some life on its own, and was invented by legitimate geniuses who are invested in it. And the ecosystem has matured, including a GCC port.

👍 Like

Reply (/forums/post?tid=5368&ReplyPostID=5386&SpaceID=127)

Answer



Adrian Parker (<https://autocare.communifire.com/people/aparker>) 

8/12/2020



On 12/9/2019 10:05 AM, Krister Kittelson said:

Node in particular has a very heavy implementation, is not self-hosting


You want a light implementation, but you also want self-hosting ... :)

 Like

Reply (/forums/post?tid=5368&ReplyPostID=6111&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>) 

8/12/2020



The reference implementation is currently 18MB with no external dependencies, dynamic libraries or virtual machines, and includes a web server (I guess that's what is meant here by "self-hosting").

 Like

Reply (/forums/post?tid=5368&ReplyPostID=6112&SpaceID=127)

Answer



Adrian Parker (<https://autocare.communifire.com/people/aparker>) 

8/12/2020



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)

The contradiction made me smile is all.

 Like

Reply (/forums/post?tid=5368&ReplyPostID=6113&SpaceID=127)

Answer

