



RSPAMD



15-Year Odyssey

From hobby to a large open source project

Vsevolod Stakhov, February 2024

The start

vstakhov * Add skeleton		70c7828 · 16 years ago	🕒 1 Commits
📁 compat	* Add skeleton	16 years ago	
📄 Makefile.in	* Add skeleton	16 years ago	
📄 cfg_file.h	* Add skeleton	16 years ago	
📄 cfg_file.l	* Add skeleton	16 years ago	
📄 cfg_file.y	* Add skeleton	16 years ago	
📄 cfg_utils.c	* Add skeleton	16 years ago	
📄 configure	* Add skeleton	16 years ago	
📄 main.c	* Add skeleton	16 years ago	
📄 main.h	* Add skeleton	16 years ago	
📄 memcached.c	* Add skeleton	16 years ago	
📄 memcached.h	* Add skeleton	16 years ago	
📄 upstream.c	* Add skeleton	16 years ago	
📄 upstream.h	* Add skeleton	16 years ago	
📄 util.c	* Add skeleton	16 years ago	
📄 util.h	* Add skeleton	16 years ago	
📄 worker.c	* Add skeleton	16 years ago	

Main goals

Still a pet project

Main goals

Still a pet project

- Fast emails processing: we struggled with load on our scanners 🚀

Main goals

Still a pet project

- Fast emails processing: we struggled with load on our scanners 🚀
- Minimal support of the required features:

Main goals

Still a pet project

- Fast emails processing: we struggled with load on our scanners 🚀
- Minimal support of the required features:
 - Regexp

Main goals

Still a pet project

- Fast emails processing: we struggled with load on our scanners 🚀
- Minimal support of the required features:
 - Regexps
 - URLs parsing

Main goals

Still a pet project

- Fast emails processing: we struggled with load on our scanners 🚀
- Minimal support of the required features:
 - Regexps
 - URLs parsing
 - UTF8 and international emails!

Main goals

Still a pet project

- Fast emails processing: we struggled with load on our scanners 🚀
- Minimal support of the required features:
 - Regexp
 - URLs parsing
 - UTF8 and international emails!

- Target system: FreeBSD



Pet project

Advantages and disadvantages

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready
- However:

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready
- However:
 - You have **limited amount of time**: it's die or thrive game

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready
- However:
 - You have **limited amount of time**: it's die or thrive game
 - Bad decisions can **hurt** for a long time (e.g. XML in config files)

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready
- However:
 - You have **limited amount of time**: it's die or thrive game
 - Bad decisions can **hurt** for a long time (e.g. XML in config files)
- Even though you still have time to learn from errors and make something cool:

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready
- However:
 - You have **limited amount of time**: it's die or thrive game
 - Bad decisions can **hurt** for a long time (e.g. XML in config files)
- Even though you still have time to learn from errors and make something cool:
 - UCL

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready
- However:
 - You have **limited amount of time**: it's die or thrive game
 - Bad decisions can **hurt** for a long time (e.g. XML in config files)
- Even though you still have time to learn from errors and make something cool:
 - UCL
 - HTTPCrypt

Pet project

Advantages and disadvantages

- You can **experiment** a lot: use different technologies, break compatibility, try something new and cool - that's positive 👍
- You are concentrated on **development**, not support
- Nobody expects it to be perfect or even production ready
- However:
 - You have **limited amount of time**: it's die or thrive game
 - Bad decisions can **hurt** for a long time (e.g. XML in config files)
- Even though you still have time to learn from errors and make something cool:
 - UCL
 - HTTPCrypt
- It's still your pet 🐱

Open Source?

Benefits

Open Source?

Benefits

- Rules and intelligence are individual and not revealed in general

Open Source?

Benefits

- Rules and intelligence are individual and not revealed in general
- 3-rd party users have found many issues and suggested a lot of improvements

Open Source?

Benefits

- Rules and intelligence are individual and not revealed in general
- 3-rd party users have found many issues and suggested a lot of improvements
- Github has proven to be a great collaboration platform

Open Source?

Benefits

- Rules and intelligence are individual and not revealed in general
- 3-rd party users have found many issues and suggested a lot of improvements
- Github has proven to be a great collaboration platform
- I have found some contributors who have helped me with coding and documenting of Rspamd (in particular, Andrew Lewis and Alexander Moisseev)

Open Source?

Unexpected problems

Open Source?

Unexpected problems

- Scaling:

Open Source?

Unexpected problems

- Scaling:
 - Different systems, different use cases, different hardware, different rules etc

Open Source?

Unexpected problems

- Scaling:
 - Different systems, different use cases, different hardware, different rules etc
 - Each OS has it's own requirements

Open Source?

Unexpected problems

- Scaling:
 - Different systems, different use cases, different hardware, different rules etc
 - Each OS has it's own requirements
 - **Public infrastructure is hard to maintain**

Open Source?

Unexpected problems

- Scaling:
 - Different systems, different use cases, different hardware, different rules etc
 - Each OS has it's own requirements
 - Public infrastructure is hard to maintain
- Exploiting:

Open Source?

Unexpected problems

- Scaling:
 - Different systems, different use cases, different hardware, different rules etc
 - Each OS has it's own requirements
 - Public infrastructure is hard to maintain
- Exploiting:
 - Community support is another full time job, but unpaid

Open Source?

Unexpected problems

- Scaling:
 - Different systems, different use cases, different hardware, different rules etc
 - Each OS has it's own requirements
 - Public infrastructure is hard to maintain
- Exploiting:
 - Community support is another full time job, but unpaid
 - ... which is not grateful in general (and very stressful)

Open Source?

Unexpected problems

- Scaling:
 - Different systems, different use cases, different hardware, different rules etc
 - Each OS has it's own requirements
 - Public infrastructure is hard to maintain
- Exploiting:
 - Community support is another full time job, but unpaid
 - ... which is not grateful in general (and very stressful)
 - **Vendors and large companies can eagerly overuse and destroy your infrastructure**

Project growth

Problems

Project growth

Problems

- You have to maintain backward compatibility or provide a clear upgrade path

Project growth

Problems

- You have to maintain backward compatibility or provide a clear upgrade path
- You have to maintain documentation in the actual state

Project growth

Problems

- You have to maintain backward compatibility or provide a clear upgrade path
- You have to maintain documentation in the actual state
- Each upgrade can cause unexpected side effects (different hardware, configuration etc)

Project growth

Problems

- You have to maintain backward compatibility or provide a clear upgrade path
- You have to maintain documentation in the actual state
- Each upgrade can cause unexpected side effects (different hardware, configuration etc)
- Nobody want to test experimental packages

Project growth

Problems

- You have to maintain backward compatibility or provide a clear upgrade path
- You have to maintain documentation in the actual state
- Each upgrade can cause unexpected side effects (different hardware, configuration etc)
- Nobody want to test experimental packages
- You see the same questions again and again... each day. Adding those to documentation/FAQ does not help in general

Project growth

Problems

- You have to maintain backward compatibility or provide a clear upgrade path
- You have to maintain documentation in the actual state
- Each upgrade can cause unexpected side effects (different hardware, configuration etc)
- Nobody want to test experimental packages
- You see the same questions again and again... each day. Adding those to documentation/FAQ does not help in general
- **Adding new features is hard, close to painful**

Project growth

Problems

- You have to maintain backward compatibility or provide a clear upgrade path
- You have to maintain documentation in the actual state
- Each upgrade can cause unexpected side effects (different hardware, configuration etc)
- Nobody want to test experimental packages
- You see the same questions again and again... each day. Adding those to documentation/FAQ does not help in general
- Adding new features is hard, close to painful
- **Migration to some modern technologies is close to impossible**

Some lessons learned

Hard way

Some lessons learned

Hard way

- Test are important. Boring but important, especially in the growth path to prevent regressions

Some lessons learned

Hard way

- Tests are important. Boring but important, especially in the growth path to prevent regressions
- Documentation is the same game: the proper habit is to write code -> tests and documentation simultaneously

Some lessons learned

Hard way

- Tests are important. Boring but important, especially in the growth path to prevent regressions
- Documentation is the same game: the proper habit is to write code -> tests and documentation simultaneously
- You can never satisfy all OS vendors, so just choose your own path

Some lessons learned

Hard way

- Tests are important. Boring but important, especially in the growth path to prevent regressions
- Documentation is the same game: the proper habit is to write code -> tests and documentation simultaneously
- You can never satisfy all OS vendors, so just choose your own path
- Do not blow the size of the core - concentrate on plugins/services

Some lessons learned

Hard way

- Tests are important. Boring but important, especially in the growth path to prevent regressions
- Documentation is the same game: the proper habit is to write code -> tests and documentation simultaneously
- You can never satisfy all OS vendors, so just choose your own path
- Do not blow the size of the core - concentrate on plugins/services
- Study and use the workflow of the collaboration platform (e.g. Github)

Some lessons learned

Hard way

- Tests are important. Boring but important, especially in the growth path to prevent regressions
- Documentation is the same game: the proper habit is to write code -> tests and documentation simultaneously
- You can never satisfy all OS vendors, so just choose your own path
- Do not blow the size of the core - concentrate on plugins/services
- Study and use the workflow of the collaboration platform (e.g. Github)
- **Have a clear and straight migration plan for both external and internal architecture**

The current state

What we do to keep with the time

The current state

What we do to keep with the time

- CI and tests culture

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:
 - Keep the stable branch when a head obtains new features

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:
 - Keep the stable branch when a head obtains new features
 - Don't backport any features and never break compatibility

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:
 - Keep the stable branch when a head obtains new features
 - Don't backport any features and never break compatibility
- Providing Docker images and assist OS maintainers

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:
 - Keep the stable branch when a head obtains new features
 - Don't backport any features and never break compatibility
- Providing Docker images and assist OS maintainers
- Supplying ASAN packages for easier debugging of the issues (helped a lot in the past)

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:
 - Keep the stable branch when a head obtains new features
 - Don't backport any features and never break compatibility
- Providing Docker images and assist OS maintainers
- Supplying ASAN packages for easier debugging of the issues (helped a lot in the past)
- Slowly migrating core of the Rspamd to the modern C++ (C++20 so far)


The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:
 - Keep the stable branch when a head obtains new features
 - Don't backport any features and never break compatibility
- Providing Docker images and assist OS maintainers
- Supplying ASAN packages for easier debugging of the issues (helped a lot in the past)
- Slowly migrating core of the Rspamd to the modern C++ (C++20 so far)
- **Use external services for specific tasks**

The current state

What we do to keep with the time

- CI and tests culture
- Using of the Github workflow (at least for releases processing)
- Following semver strategy:
 - Keep the stable branch when a head obtains new features
 - Don't backport any features and never break compatibility
- Providing Docker images and assist OS maintainers
- Supplying ASAN packages for easier debugging of the issues (helped a lot in the past)
- Slowly migrating core of the Rspamd to the modern C++ (C++20 so far)
- Use external services for specific tasks
- It's still my pet... 

Questions?

Vsevolod Stakhov - vsevolod@rspamd.com