DOI: 10.58168/PHBSc-IT2025_230-236
УДК 658.51

РАЗРАБОТКА АРХИТЕКТУРЫ API ДЛЯ СТАТИЧЕСКОГО АНАЛИЗА БЕЗОПАСНОСТИ ПРИЛОЖЕНИЯ

DEVELOPMENT OF AN API ARCHITECTURE FOR STATIC APPLICATION SECURITY ANALYSIS

Горбунов В.А., доктор физико-математических наук, профессор, профессор, ФГБОУ ВО «Вологодский государственный университет», Вологда, Россия

Gorbunov V.F., DrSc in Physics and Mathematics, professor, professor, Vologda State University, Vologda, Russia

Кочкин Д.В., кандидат технических наук, доцент, доцент, ФГБОУ ВО «Вологодский государственный университет», Вологда, Россия

Kochkin D.V., PhD in technical science, Docent, Associate Professor, Vologda State University, Vologda, Russia

Сорокин Д.О., студент 4-го курса Института Математики, Естественных и Компьютерных Наук, ФГБОУ ВО «Вологодский государственный университет», Вологда, Россия

Sorokin D. O., 4th year student of the Institute of Mathematics, Natural and Computer Sciences, FSBEI HE «Vologda State University», Vologda, Russia

Аннотация. В статье рассматривается проблема проектирования архитектуры API для SAST-системы. Были использованы современные подходы и технологии, такие как микросервисная модель, Docker, Kafka, REST API, MVC, Spring Boot Framework, Nginx, SpotBugs для создания масштабируемого и отказоустойчивого решения. В статье также рассмотрены преимущества и недостатки выбранных технологий, а также их роль в обеспечении высокой производительности и надёжности SAST-системы. Результаты работы могут быть применены для разработки аналогичных систем, требующих высокой степени автоматизации и масштабируемости.

Ключевые слова: SAST, микросервисы, Kafka, REST API, Docker, балансировка нагрузки, кибербезопасность, CVE.

Abstract. The article considers the problem of designing an API architecture for a SAST system. Modern approaches and technologies such as the microservice model, Docker, Kafka, REST API, MVC, Spring Boot Framework, Nginx, SpotBugs were used to create a scalable and fault-tolerant solution. The article also discusses the advantages and disadvantages of the selected technologies, as well as their role in ensuring high performance and reliability of the SAST system. The results of the work can be applied to the development of similar systems that require a high degree of automation and scalability.

Keywords: SAST, microservices, Kafka, REST API, Docker, load balancing, cybersecurity, CVE.

Введение

В современном мире, где киберугрозы становятся всё более изощрёнными, обеспечение безопасности программного обеспечения является критически важным [1, 2]. SAST (Static Application Security Testing — статическое тестирование безопасности приложений) – это метод анализа исходного кода на наличие уязвимостей без выполнения программы. Статический анализ безопасности приложений позволяет выявлять уязвимости на этапе разработки, что значительно снижает риски эксплуатации небезопасного кода. SAST применяется в различных областях, включая разработку веб-приложений, мобильных приложений и embedded-систем.

Актуальность данной темы обусловлена растущим спросом на инструменты, которые могут интегрироваться в процессы непрерывной интеграции и доставки (CI/CD). Перспективы развития SAST связаны с автоматизацией анализа, улучшением точности обнаружения уязвимостей и снижением числа ложных срабатываний [3].

Постановка задачи

Основной задачей является разработка архитектуры API для SAST, которая обеспечит масштабируемость, гибкость и высокую производительность системы. Архитектура должна поддерживать анализ больших объёмов кода и быть легко интегрируемой в существующие процессы разработки. Для достижения этих целей необходимо решить следующие задачи: разделить систему на независимые компоненты, которые могут работать параллельно; обеспечить взаимодействие между компонентами через REST API и асинхронные сообщения; реализовать возможность горизонтального масштабирования системы за счёт использования Docker и балансировки нагрузки, чтобы обеспечить большое количество пользователей, одновременно работающих с системой; обеспечить возможность простой непрерывной поддержки системы; обеспечить интерфейс для взаимодействия с системой через веб-интерфейс.

Разработка архитектуры.

Для того чтобы определиться с архитектурой, необходимо понять, кто и как будет взаимодействовать с системой [4]. Для решения этой задачи на рисунке 1 приведена Use Case диаграмма. Используя Use Case диаграмму, можно чётко обосновать необходимость микросервисной архитектуры для SAST проекта. Она позволяет разделить систему на независимые компоненты, каждый из которых отвечает за определённую функциональность. Это особенно важно для SAST, где требуется высокая масштабируемость, гибкость и отказоустойчивость.

Приведём некоторые достоинства микросервисной модели [6]: возможна непрерывная доставка и развертывание крупных, сложных приложений; сервисы развертываются независимо друг от друга; сервисы масштабируются независимо друг от друга. Отсюда выходит, что микросервисная архитектура хорошо подходит для решения наших задач [7, 8].

После того, как мы установили, что архитектура системы SAST строится на основе микросервисной модели, которая обеспечивает гибкость и масштабируемость, перечислим основные компоненты системы:

1. UI (Пользовательский интерфейс): Веб-интерфейс для взаимодействия с системой. Пользователь может загружать проект, настраивать параметры анализа и просматривать результаты.

2. Менеджер: Основной компонент, который управляет процессом анализа. Менеджер принимает запросы от UI. Управляет базой данных приложения, отправляет запросы на анализ проектов.

3. Мастер: Координатор, который управляет работой агентов. Мастер отслеживает состояние агентов, распределяет задачи и обеспечивает балансировку нагрузки.

4. Агенты: Компоненты, которые выполняют непосредственно анализ кода. Каждый агент работает независимо и может быть запущен в отдельном Docker-контейнере.

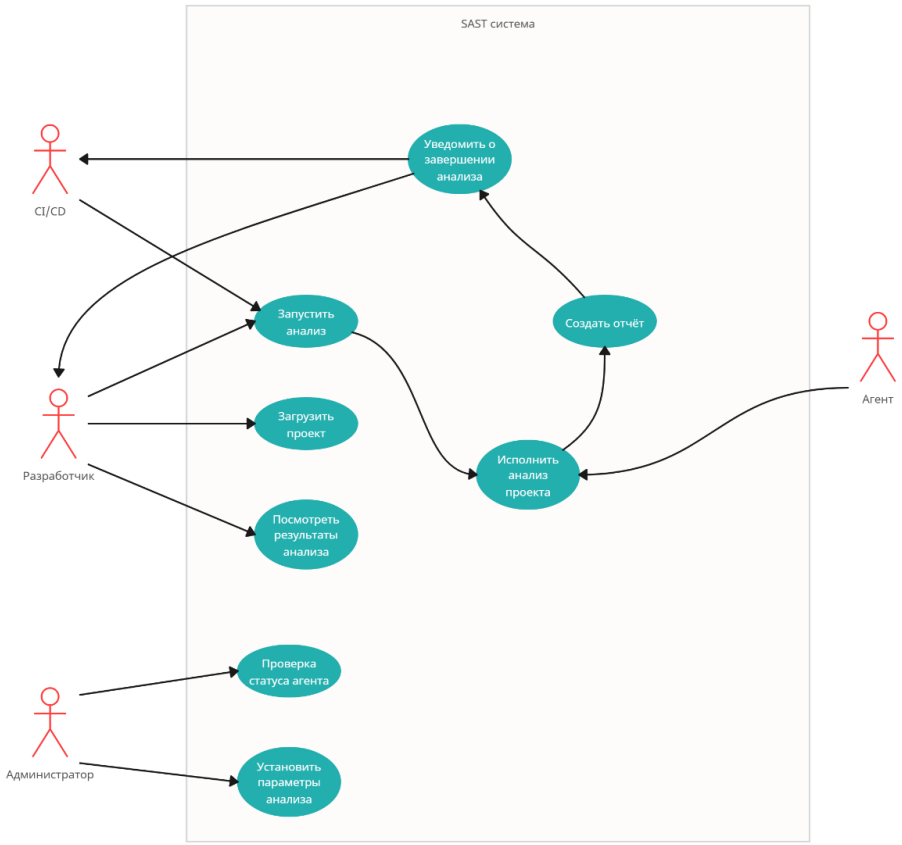

Рисунок 1 – Use Case диаграмма системы

На рисунке 2 представлена общая схема архитектуры приложения.

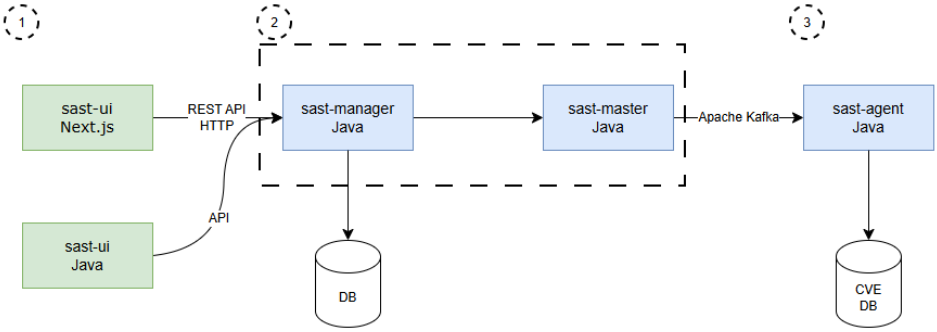

Рисунок 2 – Архитектура приложения

Пользовательский интерфейс (1) состоит из двух частей: 1) UI для пользователей сервиса; 2) UI для администрирования сервиса. UI общается с менеджером посредством протокола HTTP с использованием архитектурного стиля REST API.

На схеме мастер и менеджер (2) разделяются, но на практике решено было их объединить по следующим причинам: 1) упрощение архитектуры с точки зрения разработки; 2) снижение накладных расходов на общение между модулями; 3) упрощение балансировки нагрузки; 4) экономия ресурсов на содержание двух отдельных сервисов.

Мастер общается с агентом посредством распределённого программного брокера асинхронных сообщений Apache Kafka. Поскольку агент в данной системе является потенциально узким местом из-за долгого процесса анализа, то асинхронность Kafka пойдёт только на пользу [9].

Горизонтальное масштабирование легко осуществить с помощью docker, создавая множество экземпляров и установив балансировщик нагрузки nginx. Такой подход позволяет повысить отказоустойчивость системы с минимальными усилиями и потерями в производительности. На рис. 3 изображена схема горизонтального масштабирования системы.

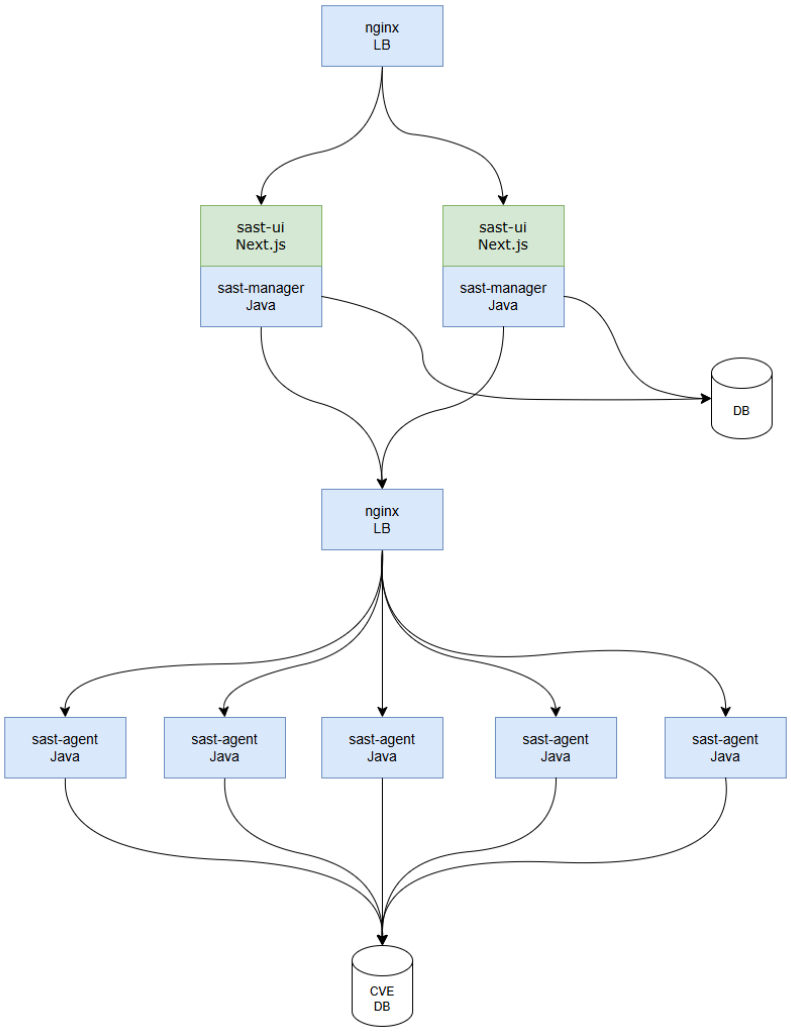

Рисунок 3 – Схема горизонтального масштабирования

Для обеспечения гибкости, масштабируемости и простой поддержки в процессе эксплуатации хорошо подходит шаблон проектирования MVC (Model – View – Controller), поскольку: 1) каждый компонент может разрабатываться и масштабироваться независимо. Например, можно изменить способ генерации отчётов (View), не затрагивая бизнес-логику (Model); 2) бизнес-логика (Model) может тестироваться отдельно от представления (View) и контроллеров (Controller). Общая UML-диаграмма для шаблона приведена на рисунке 4.

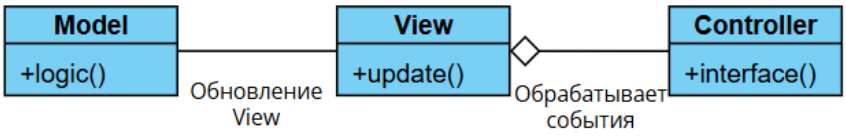

Рисунок 4 – UML-диаграмма MVC

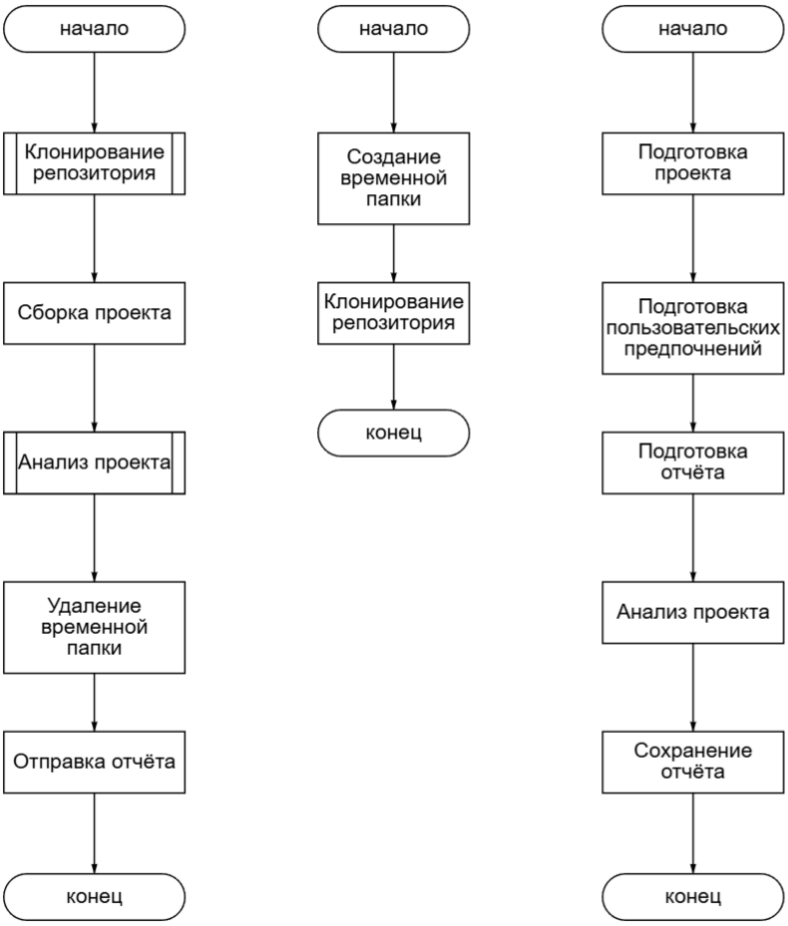

Рисунок 5 – Блок-схема алгоритма анализа проекта

Для реализации шаблона следует использовать Spring Boot Framework. Он имеет “из коробки” богатый инструментарий и позволяет сосредоточиться на написании кода [10] за счёт механизма автоматической конфигурации, ускоряющей процесс разработки.

На рисунке 5 представлена блок-схема алгоритма анализа проекта. В данном алгоритме для анализа проекта используется сторонняя библиотека SpotBugs – инструмент статического

анализа кода для Java, который помогает выявлять потенциальные ошибки и уязвимости в коде. Выбор пал на него, так как у SpotBugs достаточно большое и активное сообщество, хорошее качество анализа, а также возможность создания собственного стиля для отчётов об ошибках. Единственным крупным недостатком этой библиотеки является то, что она рассчитана лишь на Java, поэтому для анализа кода на иных языках программирования придётся использовать иные библиотеки.

Заключение

Разработанная архитектура API для SAST обеспечивает высокую производительность, масштабируемость и удобство интеграции в процессы разработки. Использование микросервисной модели, Docker и Kafka позволяет гибко настраивать систему под конкретные задачи и масштабировать её в зависимости от объёмов анализируемого кода.

Внедрение подобной системы в процессы разработки позволит значительно повысить уровень безопасности программного обеспечения, снизить риски эксплуатации уязвимостей и ускорить процесс анализа кода.

СПИСОК ЛИТЕРАТУРЫ

1. Building self-organizing information and telecommunications systems / A.A. Sukonschikov, A.N. Shvetsov, I.A. Andrianov, D.V. Kochkin // Journal of Physics: Conference Series, Krasnoyarsk, Russian Federation, 25 сентября – 04 октября 2020 года. Vol. 1679. – Krasnoyarsk, Russian Federation: Institute of Physics and IOP Publishing Limited, 2020. – P. 32013. – DOI 10.1088/1742-6596/1679/3/032013.

2. Распределенные интеллектуальные информационные системы и среды / А.Н. Швецов, А.А. Суконщиков, Д.В. Кочкин [и др.] ; Под редакцией А.Н. Швецова и А.А. Суконщикова. – Курск : Закрытое акционерное общество "Университетская книга", 2017. – 197 с.

3. Кочкин, Д.В. Проектирование и конструирование программного обеспечения : учебное пособие / Д.В. Кочкин, А.Н. Швецов. – Вологда : Вологод, 2023. – 127 с.

4. Кочкин, Д.В. Информационные сети и телекоммуникации / Д.В. Кочкин, А.А. Суконщиков. Том часть 1. – Курск : Закрытое акционерное общество "Университетская книга", 2016. – 233 с.

5. Краснов, А.А. Разработка системы тестов для тестирования веб-приложения / А.А. Краснов, Д.В. Кочкин // Вестник Вологодского государственного университета. Серия: Технические науки. – 2024. – № 3(25). – С. 43-48.

6. Ричардсон, К. Микросервисы. Паттерны разработки и рефакторинга / К. Ричардсон. – Санкт-Петербург: Питер, 2019. – 544 с.

7. Модели и методы построения нейро-нечетких интеллектуальных агентов в информационно-телекоммуникационных системах / А.А. Суконщиков, И.А. Андрианов, С.В. Дианов [и др.]. – Курск : Закрытое акционерное общество "Университетская книга", 2021. – 152 с.

8. Интеллектуальные информационно-телекоммуникационные системы / А.Н. Швецов, А.А. Суконщиков, И.А. Андрианов [и др.]. – Вологда : Вологодский государственный университет, 2023. – 127 с.

9. Скотт, Д. Kafka в действии / Д. Скотт, В. Гамов, Д. Клейн. – Москва: ДМК Пресс, 2022. – 310 с.

10. Уоллс, К. Spring в действии / К. Уоллс. – 6-е изд./ пер. с англ.А. Н. Киселева. – Москва: ДМК Пресс, 2022. – 544 с.

REFERENCES

1. Building self-organizing information and telecommunications systems / A.A. Sukonschikov, A.N. Shvetsov, I.A. Andrianov, D.V. Kochkin // Journal of Physics: Conference Series, Krasnoyarsk, Russian Federation, September 25 – October 04, 2020. Vol. 1679. – Krasnoyarsk, Russian Federation: Institute of Physics and IOP Publishing Limited, 2020. – P. 32013. – DOI 10.1088/1742-6596/1679/3/032013.

2. Distributed intelligent information systems and environments / A.N. Shvetsov, A.A. Sukonschikov, D.V. Kochkin [et al.]; Edited by A.N. Shvetsov and A.A. Sukonschikov. – Kursk: Closed Joint-Stock Company "University Book", 2017. – 197 p.

3. Kochkin, D.V. Software Design and Construction : a tutorial / D.V. Kochkin, A.N. Shvetsov. – Vologda : Vologod, 2023. – 127 p.

4. Kochkin, D.V. Information Networks and Telecommunications / D.V. Kochkin, A.A. Sukonshchikov. Volume Part 1. – Kursk : Closed Joint-Stock Company "University Book", 2016. – 233 p.

5. Krasnov, A.A. Development of a Test System for Testing a Web Application / A.A. Krasnov, D.V. Kochkin // Bulletin of the Vologda State University. Series: Technical Sciences. – 2024. – No. 3(25). – P. 43-48.

6. Richardson, K. Microservices. Development and Refactoring Patterns / K. Richardson. – St. Petersburg: Piter, 2019. – 544 p.

7. Models and Methods for Building Neuro-Fuzzy Intelligent Agents in Information and Telecommunication Systems / A.A. Sukonshchikov, I.A. Andrianov, S.V. Dianov [et al.]. – Kursk: Closed Joint-Stock Company "University Book", 2021. – 152 p.

8. Intelligent Information and Telecommunication Systems / A.N. Shvetsov, A.A. Sukonshchikov, I.A. Andrianov [et al.]. – Vologda: Vologda State University, 2023. – 127 p.

9. Scott, D. Kafka in Action / D. Scott, V. Gamow, D. Klein. – Moscow: DMK Press, 2022. – 310 p.

10. Walls, K. Spring in Action / K. Walls. – 6th ed. / trans. from English by A. N. Kiseleva. – Moscow: DMK Press, 2022. – 544 p.