УДК [004.42+004.272.3]-047.36

# МОНИТОРИНГ ПРИЛОЖЕНИЙ, РАЗРАБОТАННЫХ С ИСПОЛЬЗОВАНИЕМ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ

*Мишота В.Г., Жук Н.Е*

*Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь*

*Научный руководитель: Василькова А.Н. – ассистент кафедры ИПиЭ*

**Аннотация.** Рассмотрены особенности мониторинга приложений, разработанных на базе микросервисной архитектуры. Приведены основные метрики, используемые для мониторинга микросервисных приложений. Перечислены инструменты для сбора, визуализации и анализа данных по метрикам.

**Ключевые слова:** микросервисная архитектура, мониторинг, микросервис, метрика, распределенные системы

***Введение.*** Мониторинг остается важной частью управления любой IT-системой, а проблемы, связанные с мониторингом микросервисов, особенно уникальны. Например, традиционные монолитные системы, развернутые как единый исполняемый файл или библиотека, имеют точки отказа и зависимости, отличные от систем, развернутых с архитектурой микросервисов.

***Основная часть.*** Приложения, разработанные с использованием микросервисной архитектуры, нуждаются в мониторинге по тем же причинам, что и любые распределенные системы: элементы системы время от времени выходят из строя.

Вероятность сбоя – очевидный показатель важности мониторинга, но не единственный. Сложные системы, даже монолитные, могут работать с перебоями, что влияет на производительность и в перспективе может привести к серьезным последствиям. Мониторинг поведения систем предоставляет операторам данные о перебоях до возникновения полноценного сбоя.

Системы мониторинга со временем генерируют ценные данные, которые можно использовать для повышения производительности сервиса. Данные об отказах и производительности системы могут быть проанализированы для поиска закономерностей в системных сбоях, которые могут быть связаны с событиями. Например, рассмотрим случай, когда данные указывают на то, что 25 процентов общих системных сбоев происходят в течение часа после нового развертывания. Таким образом, это будет убедительным показателем того, что процессы развертывания требуют доработки.

Функции оповещения инструментов управления производительностью приложений (*Application Performance Management*, *APM*) позволяют просматривать данные о сбоях и производительности в режиме реального времени, отфильтровывая все, кроме наиболее важных событий. Эти события могут быть получены в виде оповещений через информационные панели, электронную почту, Slack и мобильные приложения [1].

Микросервисные приложения развертываются как семейство независимых сервисов. Каждый сервис выполняет определенную функцию и должен взаимодействовать с другими сервисами для ее выполнения. В микросервисной архитектуре сложные рабочие процессы организуются с помощью ряда микросервисов. Каждый сервис может иметь свои внешние зависимости. Это может быть диск, база данных или другие сервисы. Каждое взаимодействие между сервисом и зависимым ресурсом является потенциальной точкой отказа. Отказ зависимости или даже просто снижение производительности приведет к неизбежному влиянию на производительность системы в целом. Важно обнаруживать эти проблемы заранее, чтобы предотвратить системную деградацию или сбой.

Микросервисная архитектура учитывает эти факты как в контексте разработки, так и в контексте эксплуатации. Команды разработчиков могут внедрять инструменты для мониторинга в системы во время разработки. Необходимо создавать и поддерживать инфраструктуру для сбора данных о приложениях и сторонних сервисах. Данные, собираемые этими сервисами, используются для краткосрочных аварийных сценариев, также называемых оповещениями. Долгосрочное использование включает интеллектуальный анализ данных и аналитику как способ поиска закономерностей. Анализ этих закономерностей поможет в поиске распространенных случаев отказа и принятии превентивных мер.

Мониторинг – это процесс создания отчетов, сбора и хранения данных. При этом стоит вопрос о том, какие данные подлежат сбору, а какие – нет. Большие распределенные системы постоянно обрабатывают огромные объемы данных, потенциально генерируя гигабайты новых данных об их поведении за короткое время. Как найти действительно важный сигнал среди шума? У каждого приложения будут уникальные особенности, связанные с мониторингом. Есть несколько метрик, которые стоит отслеживать. Они включают в себя метрики приложения, метрики платформы и системные события [2].

Метрики приложения относятся к конкретному приложению. Если приложение принимает регистрации пользователей, стандартным показателем может быть количество успешно завершенных регистраций за последний час. Система подписки на *email*-рассылку может записывать контекстно-зависимые события, такие как проверка правильности заполнения формы. Эти данные полезны для команд разработчиков и организаций, чтобы понять функциональное поведение системы. Если отправка формы обычно выполняется 1000 раз в час, и внезапно падает до 500 за последние два часа, эта аномалия может указывать на проблему.

Метрики платформы показывают состояние основных элементов инфраструктуры, например, общее среднее время выполнения каждого из десяти наиболее часто выполняемых запросов к базе данных; среднее время выполнения самых быстрых 10 процентов; среднее время выполнения самых медленных 10 процентов; количество запросов в минуту, которые получает служба; среднее время отклика для каждой конечной точки службы; соотношение успешных и неудачных сценариев для каждого сервиса. В совокупности эти показатели можно использовать для понимания производительности и поведения системы на низком уровне. В идеале эти показатели будут предупреждать о снижении производительности, которое может повлиять на общую работоспособность или привести к системному сбою. Важно использовать данные низкого уровня таким образом, чтобы прогнозировать и предотвращать более масштабные сбои до их возникновения.

Внешние факторы постоянно воздействуют на системы. Наиболее распространенными и потенциально наиболее разрушительными являются новые развертывания. Существует сильная корреляция между развертыванием нового кода и системными сбоями. Таким образом, стоит учитывать данные о такого рода системных событиях наряду с остальными показателями. События масштабирования, обновления конфигурации и другие операционные изменения также актуальны и должны быть записаны. Запись этих событий также позволит соотнести их с поведением системы.

Технологии и инструменты мониторинга делятся на две большие категории: библиотеки и сторонние сервисы. Библиотеки подключаются в приложение во время его разработки. Наиболее популярные языковые фреймворки, такие как *Java*, *.NET*, *Go* и другие; включают ресурсы для передачи потоков данных по сети. Эти ресурсы можно использовать для регистрации и мониторинга. Примерами сторонних библиотек являются библиотеки с открытым исходным кодом, такие как *AppMetrics* для *.NET* и *SPF4J* для *Java*. Сторонние сервисы же в основном сосредоточены на сборе и анализе данных, полученных от приложений, операционных систем и сетевых платформ, на которых они работают. Вот несколько примеров инструментов, которые могут быть частью платформы мониторинга микросервисов:

*Raygun APM*. Платформа *Raygun APM* – еще один пример комплексной системы, которая предоставляет инструментарий для сбора данных, процессы-сборщики данных, а также панель мониторинга для визуализации данных по метрикам. *Raygun APM* поддерживает *.NET*. Поддержка *Java* и *Ruby* находится в разработке.

*Zipkin* – это система отслеживания с открытым исходным кодом, разработанная специально для отслеживания запросов между микросервисами. Это особенно полезно для анализа проблем с задержкой. *Zipkin* включает в себя как инструментальные библиотеки, так и процессы-сборщики, которые собирают и сохраняют данные трассировки.

*Apache Kafka* – это система обработки потоков. Она использует методологию «публикация/подписка» для чтения и записи данных в логический «поток», который по своей концепции аналогичен системе обмена сообщениями, такой как *RabbitMQ*. *Kafka* можно комбинировать с другими инструментами, такими как *Zipkin*, чтобы обеспечить надежное решение для передачи и хранения данных метрик.

*Grafana*: данные, собранные вышеперечисленными инструментами, не очень полезны, если их нельзя интерпретировать и анализировать. *Grafana* представляет собой веб-инструмент визуализации, который используется для визуализации этих данных.

*Prometheus* – это решение для мониторинга с открытым исходным кодом, первоначально разработанное *SoundCloud*. Оно широко используется для хранения и запроса «временных рядов», то есть данных, описывающих действия во времени. *Prometheus* часто сочетается с другими инструментами, особенно с *Grafana*, для визуализации данных временных рядов и предоставления визуализации.

***Заключение.*** Требования к мониторингу следует учитывать с самого начала жизненного цикла приложения. Системный мониторинг требует участия как стороны разработки, так и стороны эксплуатации. Это важная часть поддержки любой распределенной системы. Ввиду своей распределенности приложения с микросервисной архитектурой требуют повышенного внимания и активного мониторинга. Собирать актуальные данные так же важно, как и анализировать уже собранные данные. Необходим сбор данных не только из приложений, но и со сторонних сервисов и систем развертывания. Данные, собранные в результате мониторинга имеют большое значение для поддержки отказоустойчивой, надежной и доступной распределенной системы.

### *Список литературы*

1. *Ричардсон, Крис. Microservices Patterns / Крис Ричардсон. – O'Reilly Media, 2018. – 520.*
2. *Ньюман, Сэм. Building Microservices / Сэм Ньюман. – O'Reilly Media, 2015. – 259 с.*

UDC [004.42+004.272.3]-047.36

# MONITORING APPLICATIONS DEVELOPED USING MICROSERVICE ARCHITECTURE

*Mishota U.H., Zhuk M.E.*

*Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus*

*Vasilkova A.N. – assistant of the Department of EPE*

**Annotation**. The article shows the significance of monitoring microservice applications. The main metrics used to monitor microservice applications are given. Tools for collecting, visualizing and analyzing data by metrics were listed.

**Keywords**: microservice architecture, monitoring, metrics, microservice, distributed systems.