Содержание

Вступление ... 12

Об авторах ... 15

Колофон ... 16

Часть I. ВВЕДЕНИЕ ... 17

Глава 1. Что такое Prometheus? ... 18

Что такое мониторинг? ... 19
- Краткая и неполная история мониторинга ... 21
- Категории мониторинга ... 22

Архитектура Prometheus ... 27
- Клиентские библиотеки ... 27
- Экспортеры ... 28
- Обнаружение служб ... 29
- Извлечение данных ... 30
- Хранилище ... 30
- Дашборды ... 30
- Правила записи и оповещения ... 31

Управление уведомлениями ... 32
- Долгосрочное хранение ... 32

Чем Prometheus не является ... 32

Глава 2. Настройка и запуск Prometheus ... 34

Запуск Prometheus ... 34

Использование браузера выражений ... 38

Запуск экспортера узла ... 42

Оповещение ... 46

Часть II. МОНИТОРИНГ ПРИЛОЖЕНИЙ ... 53

Глава 3. Инструментирование ... 54

Простая программа ... 54

Счетчики ... 56
- Подсчет исключений ... 58
- Подсчет размера ... 59

Датчики 60
Использование датчиков 61
Обратные вызовы 62
Сводные метрики 63
Гистограммы 65
Сегменты 66
Модульное тестирование метрик 68
Подход к инструментированию 69
Что инструментировать? 70
Как широко использовать инструментирование? 72
Какие имена выбирать для метрик? 72

Глава 4. Экспортирование метрик 76

Python 77
WSGI 77
Twisted 78
Многозадачность с Gunicorn 79
Go 82
Java 83
HTTPServer 83
Сервлеты 85
Pushgateway 86
Мосты 90
Парсеры 90
Текстовый формат экспорта 91
Типы метрик 91
Метки 92
Экранирование 93
Отметки времени 93
Проверка метрик 94
OpenMetrics 94
Типы метрик 95
Метки 96
Отметки времени 96

Глава 5. Метки 97

Что такое метки? 97
Инструментированные и целевые метки 98
Инструментирование 99
Метрики 100
Несколько меток 101
Дочерние элементы 101
Агрегирование 103
Шаблоны использования меток 104
Метрики-перечисления 104
Информационные метрики 106

Когда использовать метки 108
Кардинальность 109

Глава 6. Создание дашбордов с Grafana 112

Установка 113
Источники данных 115
Дашборды и панели 116
Как избежать «стены графиков» 117
Панель временных рядов 117
Элементы управления временем 119
Панель статистики 121
Панель таблицы 123
Панель временно́й шкалы состояний 125
Переменные шаблона 126

Часть III. МОНИТОРИНГ ИНФРАСТРУКТУРЫ 130

Глава 7. Node Exporter 131

Сборщик информации о процессоре 132
Сборщик информации о файловой системе 133
Сборщик дисковой статистики 134
Сборщик информации о сетевых устройствах 135
Сборщик информации о потреблении памяти 136
Сборщик информации об аппаратной части 136
Сборщик статистики 137
Сборщик информации об имени узла 138
Сборщик информации об операционной системе 138
Сборщик информации о средней нагрузке 139
Сборщик информации о давлении 139
Сборщик текстовых файлов 140
Использование сборщика текстовых файлов 141
Отметки времени 143

Глава 8. Обнаружение служб 144

Механизмы обнаружения служб 145
Статическое обнаружение служб 146
Обнаружение служб на основе файла 147
Обнаружение служб через HTTP 150
Обнаружение служб с помощью Consul 151
Обнаружение служб с помощью EC2 152
Изменение меток 154
Выбор целей для мониторинга 154
Метки целей 157
Как извлекать метрики 165
metric_relabel_configs 167
Конфликты меток и honor_labels 169

Глава 9. Контейнеры и Kubernetes 170
cAdvisor 170
Метрики потребления процессора 171
Метрики потребления памяти 172
Метки 172
Kubernetes 173
Запуск в Kubernetes 173
Обнаружение служб 175
kube-state-metrics 184
Альтернативные развертывания 185

Глава 10. Популярные экспортеры 187
Consul 187
MySQLd 189
Grok Exporter 190
Blackbox 193
ICMP 194
TCP 199
HTTP 201
DNS 203
Настройка Prometheus 205

Глава 11. Работа с другими системами мониторинга 209
Другие системы мониторинга 209
InfluxDB 211
StatsD 212

Глава 12. Разработка экспортеров 215
Consul Telemetry 215
Пользовательские сборщики 219
Метки 223
Методические рекомендации 224

Часть IV. PROMQL 226

Глава 13. Введение в PromQL 227
Основы агрегирования 227
Датчики 227
Счетчики 229
Сводные метрики 230
Гистограммы 231
Селекторы 233
Сопоставители 234
Мгновенный вектор 235
Вектор диапазона 236

Подзапросы....238
Смещение....239
Модификатор @....240
HTTP API....240
query....240
query_range....243

Глава 14. Операторы агрегирования....246

Группировка....246
without....247
by....248
Операторы....249
sum....249
count....250
avg....251
group....252
stddev и stdvar....252
min и max....253
topk и bottomk....254
quantile....254
count_values....255

Глава 15. Двухместные операторы....258

Работа со скалярами....258
Арифметические операторы....259
Тригонометрический оператор....260
Операторы сравнения....260
Сопоставление векторов....262
Один к одному....263
Многие к одному и group_left....265
Многие ко многим и логические операторы....268
Приоритет операторов....272

Глава 16. Функции....273

Изменение типа....273
vector....273
scalar....274
Математические функции....275
abs....275
ln, log2 и log10....276
exp....276
sqrt....276
ceil и floor....277
round....277
clamp, clamp_max и clamp_min....278
sgn....278

Тригонометрические функции ... 278
Время и дата ... 279
time ... 279
Функции minute, hour, day_of_week, day_of_month, day_of_year, days_in_month, month и year ... 280
timestamp ... 281
Метки ... 282
label_replace ... 282
label_join ... 283
Отсутствующие серии, absent и absent_over_time ... 283
Сортировка с помощью sort и sort_desc ... 284
Гистограммы с histogram_quantile ... 285
Счетчики ... 285
rate ... 286
increase ... 287
irate ... 288
resets ... 289
Изменение датчиков ... 289
changes ... 289
deriv ... 290
predict_linear ... 290
delta ... 291
idelta ... 291
holt_winters ... 291
Агрегирование по времени ... 292

Глава 17. Правила записи ... 294

Использование правил записи ... 294
Когда использовать правила записи ... 297
Уменьшение кардинальности ... 297
Составление функций для обработки векторов диапазонов ... 299
Правила для API ... 300
Как не следует использовать правила ... 300
Именование правил записи ... 302

Часть V. ОПОВЕЩЕНИЕ ... 306

Глава 18. Оповещение ... 307

Правила оповещения ... 308
for ... 310
Метки оповещений ... 312
Аннотации и шаблоны ... 315
Что такое хорошее оповещение? ... 317
Настройка диспетчеров уведомлений в Prometheus ... 318
Внешние метки ... 319

Глава 19. Alertmanager 321
Конвейер уведомлений 321
Конфигурационный файл 322
Дерево маршрутизации 323
Приемники 330
Подавление 340
Веб-интерфейс Alertmanager 341

Часть VI. РАЗВЕРТЫВАНИЕ 344

Глава 20. Безопасность на стороне сервера 345
Функции безопасности в Prometheus 345
Включение TLS 345
Дополнительные параметры TLS 347
Включение базовой аутентификации 347

Глава 21. Собираем все вместе 350
Планирование развертывания 350
Расширение Prometheus 351
Выход на глобальный уровень с помощью механизма федерации 353
Долгосрочное хранение 356
Эксплуатация Prometheus 358
Аппаратное обеспечение 358
Управление конфигурацией 360
Сети и аутентификация 362
Планирование сбоев 363
Кластеризация диспетчеров уведомлений 366
Мета- и перекрестный мониторинг 367
Управление производительностью 368
Обнаружение проблем 368
Поиск дорогостоящих метрик и целей 369
Снижение нагрузки 370
Горизонтальное сегментирование 371
Управление изменениями 373
Получение помощи 373

Предметный указатель 375

Вступление

В этой книге подробно описываются приемы работы с системой мониторинга Prometheus, позволяющей наблюдать, строить графики, характеризующие особенности работы приложений и инфраструктуры, и рассылать оповещения в случае сбоев. Она предназначена для разработчиков приложений, системных администраторов и всех, кто находится между ними.

Увеличение осведомленности

Сопровождая системы, важно знать, что они запущены и работают, но истинная ценность заключается не в этом. Гораздо важнее иметь представление об особенностях работы систем.

Под особенностями работы мы подразумеваем не только производительность – время отклика и уровень нагрузки на центральный процессор (ЦП) для каждого запроса, – но и многие другие показатели. Сколько запросов к базе данных требуется для обработки каждого заказа? Не пора ли приобрести сетевое оборудование с более высокой пропускной способностью? Сколько промахов кеша наблюдается? Достаточно ли много пользователей взаимодействует со сложной функцией, чтобы оправдать ее дальнейшее существование?

Это лишь некоторые из вопросов, на которые система мониторинга, основанная на метриках, может дать ответ и помочь вам понять, почему она дала именно такой ответ. Мы рассматриваем мониторинг как средство получения информации обо всей системе, от обзоров высокого уровня до перечисления мельчайших деталей, полезных для отладки. Полный набор средств мониторинга для отладки и анализа включает не только метрики, но также журналы, трассировки и результаты профилирования, но во главе угла стоят все-таки метрики, и они – первое, к чему вы должны обращаться, ощутив потребность в ответах на вопросы системного уровня.

Prometheus поощряет широкое внедрение своих инструментов в системы, от приложений до «голого железа». Они помогут вам увидеть, как взаимодействуют ваши подсистемы и компоненты, и преобразовать неизвестные в известные.

Эволюция Prometheus

Prometheus перешагнул 10-летний рубеж, и в этом втором издании мы постарались отразить все новые достижения. Prometheus продолжает развиваться и расширяться, с каждым годом предлагая еще больше возможностей для извлечения и хранения данных. Этот прогресс является результатом упорного труда сообщества пользователей и разработчиков, использующих Prometheus в самых разных отраслях для мониторинга самых разных систем и приложений.

Во втором издании этой книги рассказывается о многих новых функциях PromQL, средствах обнаружения служб и приемниках Alertmanager, добавленных со времени выхода первого издания книги.

В новой специальной главе рассматриваются функции обеспечения безопасности на стороне сервера, такие как TLS, добавленные в Prometheus и некоторые экспортеры.

Обозначения и соглашения, принятые в этой книге

В книге действуют следующие типографские соглашения.

Курсив
: Используется для обозначения новых терминов, имен файлов и их расширений.

`Моноширинный шрифт`
: Используется для оформления листингов программ, а также в обычном тексте для обозначения элементов программы, таких как имена переменных или функций, баз данных, типов данных, переменных окружения, инструкций и ключевых слов.

`Моноширинный полужирный шрифт`
: Используется для выделения команд и другого текста, который должен быть набран самим пользователем.

`Моноширинный курсив`
: Используется для выделения текста, который нужно заменить данными пользователя или значениями, определяемыми контекстом.

Так обозначаются советы.

Так обозначаются примечания общего характера.

Так обозначаются предупреждения и предостережения.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Мы высоко ценим, хотя и не требуем ссылки на наши издания. В ссылке обычно указывается имя автора, название книги, издательство и ISBN, например: «Пивотто Ж., Бразил Б. Запускаем Prometheus. М.: ДМК Пресс, 2023. Copyright © 2023 O'Reilly Media, Inc., 978-1-098-13114-2 (англ.), 978-601-81034-1-4 (казах.)».

Если вы полагаете, что планируемое использование кода выходит за рамки изложенной выше лицензии, пожалуйста, обратитесь к нам по адресу dmkpress@gmail.com.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Благодарности

Эта книга едва ли увидела бы свет без работы всей команды Prometheus и сотен участников сообщества. Мы хотели бы особо поблагодарить Джулиуса Фольца (Julius Volz), Ричарда Хартманна (Richard Hartmann), Карла Бергквиста (Carl Bergquist), Эндрю Макмиллана (Andrew McMillan) и Грега Старка (Greg Stark) за отзывы к рукописи первого издания этой книги. Спасибо также Брайану Бразилу (Brian Brazil), Бартломею Плотке (Bartłomiej Płotka), Карлу Бергквисту (Carl Bergquist), Т. Дж. Хоплоку (TJ Hoplock) и Ричарду Хартманну (Richard Hartmann) за отзывы ко второму изданию.

Об авторах

Жюльен Пивотто (**Julien Pivotto**) – ведущий разработчик сервера Prometheus и экосистемы CNCF с 2017 года. В настоящее время работает главным архитектором программного обеспечения в компании O11y и одновременно является ее соучредителем. Специализируется на поддержке различных инструментов мониторинга, в том числе Prometheus, Cortex, Loki и Jaeger. Обладая многолетним опытом, помогает организациям в развертывании и обслуживании этих инструментов, а также предоставляет индивидуальные решения для разработки.

Брайан Бразил (**Brian Brazil**) – основатель компании Robust Perception и разработчик ядра Prometheus. Занимается решением проблем с мониторингом для самых разных компаний, начиная от стартапов и заканчивая корпорациями из списка Fortune 500. Хорошо известен в сообществе Prometheus, провел бесчисленное количество презентаций на конференциях и освещает многие аспекты Prometheus и мониторинга в своем блоге на веб-сайте Robust Perception.

Колофон

На обложке «Запускаем Prometheus» изображен каменный орел (*Aquila rapax*) – хищная птица, обитающая в Африке, на Ближнем Востоке и в Индии. Имея длину 1,5–2 м и такой же размах крыльев, каменный орел немного меньше других представителей рода *Aquila*. Имеет коричневый окрас с желтоватым оттенком, наиболее преобладающим в верхней части тела, уступая место более темным перьям в хвосте.

Каменные орлы, как правило, строят свои гнезда на высоких деревьях, где моногамные пары откладывают от одного до трех яиц в год. Предпочитают засушливые области, такие как пустыни, степи и саванны, где питаются падалью, рептилиями и мелкими млекопитающими.

Считается, что из-за широкого ареала обитания каменные орлы не находятся под угрозой вымирания. Однако популяция каменного орла в Западной Африке сокращается из-за расширения обрабатываемых угодий, вторгающихся в их среду обитания.

Многие животные на обложках O'Reilly находятся под угрозой исчезновения; все они важны для мира.

Рисунок для обложки выполнила Карен Монтгомери, взяв за основу старинную гравюру из энциклопедии «British Birds». Текст на обложке набран шрифтами Gilroy Semibold и Guardian Sans. Текст книги набран шрифтом Adobe Minion Pro; текст заголовков – шрифтом Adobe Myriad Condensed; а фрагменты программного кода – шрифтом Ubuntu Mono, созданным Далтоном Магом (Dalton Maag).

Часть I

ВВЕДЕНИЕ

Этот раздел познакомит вас с мониторингом в целом и системой Prometheus в частности.

В главе 1 вы узнаете о множестве различных вариантов мониторинга и подходов к нему; о метриках, которые использует Prometheus, и об архитектуре Prometheus.

В главе 2 вы запустите Prometheus в простейшей конфигурации и увидите, как система собирает машинные метрики, оценивает запросы и отправляет оповещения.

Глава 1

Что такое Prometheus?

Prometheus – это система мониторинга с открытым исходным кодом, основанная на метриках. Конечно, Prometheus далеко не единственная такая система, но что делает ее примечательной?

Prometheus выполняет только мониторинг – и делает это хорошо. Она имеет простую, но мощную модель данных и язык запросов, позволяющий анализировать производительность приложений и инфраструктуры. Она не пытается решать проблемы, выходящие за рамки метрик, оставляя решение этой задачи другим, более подходящим инструментам.

С той поры (2012 год), когда в SoundCloud работало всего несколько разработчиков, вокруг Prometheus выросло обширное сообщество. Ядро Prometheus написано на Go и распространяется под лицензией Apache 2.0. Однако в проект внесли свой вклад сотни людей, и он не контролируется какой-либо одной компанией. Всегда сложно сказать, сколько пользователей имеет проект с открытым исходным кодом, но, по нашим оценкам, в 2022 году Prometheus использовали сотни тысяч организаций. В 2016 году проект Prometheus стал вторым членом[1] Cloud Native Computing Foundation (CNCF).

Для инструментирования своего кода вы можете использовать клиентские библиотеки, написанные на всех популярных языках, включая Go, Java/JVM, C#/.Net, Python, Ruby, Node.js, Haskell, Erlang и Rust. Многие популярные приложения уже оснащены клиентскими библиотеками Prometheus, и среди них Kubernetes, Docker, Envoy и Vault. Для стороннего программного обеспечения, предоставляющего метрики в формате, отличном от Prometheus, доступны сотни библиотек интеграции. Они называются экспортерами и обеспечивают поддержку таких продуктов, как HAProxy, MySQL, PostgreSQL, Redis, JMX, SNMP, Consul и Kafka. Друг Брайана даже добавил поддержку мониторинга своих серверов Minecraft, стремясь обеспечить приемлемую частоту обновления экрана.

Простой текстовый формат[2] упрощает экспортирование метрик Prometheus. Другие системы мониторинга, как с открытым исходным кодом, так

---

[1] Первым стал фреймворк Kubernetes.

[2] Наряду с простым текстовым форматом был реализован текстовый формат Prometheus – OpenMetrics, более стандартизированная версия, имеющая некоторые отличия от других форматов.

и коммерческие, добавили поддержку этого формата. Это позволяет всем таким системам мониторинга сосредоточиться на основных функциях, а не тратить время на дублирование поддержки каждой отдельной части программного обеспечения, которую пользователь может пожелать отслеживать.

Модель данных идентифицирует каждый временной ряд не только по имени, но и по неупорядоченному набору пар ключ–значение, называемых метками. Язык запросов PromQL допускает агрегирование по любой из этих меток, что позволяет анализировать данные мониторинга не только по отдельным процессам, но также по центрам обработки данных, службам и любым другим меткам, которые вы определите. Эти данные можно отобразить в виде графиков в системах поддержки дашбордов (информационных панелей), таких как Grafana (https://oreil.ly/f5uMZ) и Perses (https://oreil.ly/YF-xW).

Оповещения можно определять с использованием того же языка запросов PromQL, что используется для построения графиков. Если параметр можно изобразить на графике, то при его изменении можно послать оповещение. Метки упрощают обслуживание оповещений, позволяя создать одно предупреждение, охватывающее все возможные значения меток. В некоторых других системах мониторинга для этого придется создавать оповещения отдельно для каждой машины/приложения. Кроме того, механизм обнаружения служб может автоматически определять, какие приложения и машины необходимо извлечь из тех или иных источников, таких как Kubernetes, Consul, Amazon Elastic Compute Cloud (EC2), Azure, Google Compute Engine (GCE) и OpenStack.

При всех этих возможностях и преимуществах Prometheus остается простым в использовании. Сервер Prometheus способен обрабатывать миллионы выборок в секунду. Это один статически связанный двоичный файл с файлом конфигурации. Все компоненты Prometheus можно запускать в контейнерах, и они не делают ничего необычного, что могло бы помешать инструментам управления конфигурацией. Он реализован с учетом возможности интеграции в уже имеющуюся инфраструктуру как ее часть, а не как платформа управления.

Теперь, когда вы получили общее представление о Prometheus, вернемся немного назад и поговорим о том, что подразумевается под термином «мониторинг», чтобы заложить основу для дальнейшего обсуждения. После этого мы перечислим основные компоненты Prometheus и поговорим о том, чем Prometheus не является.

Что такое мониторинг?

В средней школе один из учителей сказал Брайану, что если спросить 10 экономистов, что означает слово «экономика», то вы получите 11 ответов. То же верно и для мониторинга. В настоящее время нет единого устоявшегося определения этого термина. Когда Брайан рассказывает другим, чем занимается, то люди думают, что его работа включает в себя все: от наблюдения за

температурой на фабриках до выявления сотрудников, общающихся в Facebook в рабочее время, и даже обнаружения злоумышленников в сетях.

Система мониторинга Prometheus не позволяет ничего из перечисленного. Она не предназначена для этого[1] и создавалась с целью помочь разработчикам программного обеспечения и администраторам в работе с промышленными производственными компьютерными системами, такими как приложения, инструменты, базы данных и сети, поддерживающие популярные веб-сайты.

Так что же такое мониторинг в нашем понимании? Давайте сузим этот вид оперативного мониторинга компьютерных систем до четырех вариантов использования.

Оповещение
: Рассылка уведомлений, когда что-то идет не так, – как правило, самое важное, для чего предназначен мониторинг. Всегда желательно иметь возможность автоматически известить ответственного сотрудника о сложившейся ситуации.

Отладка
: Получив уведомление, сотрудник должен оценить состояние системы, определить причину или причины и, наконец, решить проблему.

Анализ тенденций
: Оповещения и отладка обычно характеризуются некоторой протяженностью во времени, от минуты до нескольких часов, и иногда желательно иметь возможность видеть, как системы используются и меняются с течением времени. Тенденции (или тренды) могут учитываться в проектных решениях и процессах, таких как планирование дальнейшего наращивания.

Решение практических задач
: Когда у вас в руках оказывается молоток, то все предметы вокруг начинают казаться гвоздями. В конце концов, все системы мониторинга организованы как конвейеры обработки данных. Иногда удобнее выделить часть системы мониторинга для решения некоторой практической задачи, не связанной с мониторингом, чем создавать отдельное решение. Это не совсем мониторинг, но на практике такой вариант применения встречается довольно часто, поэтому мы включили его в список.

В зависимости от специализации и опыта человека, с которым вы разговариваете, он может счесть, что только эти варианты составляют мониторинг. Такое недопонимание приводит к многочисленным дискуссиям на тему мониторинга, которые могут заканчиваться еще большим недопониманием. Чтобы помочь вам лучше понять суть, рассмотрим небольшую часть истории мониторинга.

---

[1] Мониторинг температуры внутри машин и центров обработки данных на самом деле не редкость. Более того, есть даже пользователи, которые ради забавы используют Prometheus для мониторинга погоды.

Краткая и неполная история мониторинга

За последние несколько лет в мониторинге произошел сдвиг в сторону использования специализированных инструментов, включая Prometheus. Долгое время доминирующим решением была некая комбинация Nagios и Graphite или их вариантов.

Когда мы говорим Nagios, то подразумеваем большое семейство программного обеспечения, такое как Icinga, Zmon и Sensu. Работа такого ПО основана на регулярном выполнении сценариев, называемых *проверками*. Если проверка завершается неудачно, возвращая ненулевой код завершения, то генерируется предупреждение. Первая версия Nagios была создана Итаном Галстадом (Ethan Galstad) в 1996 году как приложение для MS-DOS. Затем в 1999 году оно было выпущено под названием NetSaint и в 2002-м переименовано в Nagios.

История Graphite началась в далеком 1994 году, когда Тобиас Этикер (Tobias Oetiker) написал сценарий на Perl, который в 1995 году превратился в Multi Router Traffic Grapher, или MRTG 1.0. Как следует из названия, он в основном использовался для мониторинга сети через простой протокол управления сетью (Simple Network Management Protocol, SNMP). MRTG также мог получать метрики, запуская сторонние сценарии[1]. В 1997 году произошли существенные изменения, связанные с переносом части кода на C и созданием базы данных Round Robin Database (RRD), использовавшейся для хранения метрик. Это привело к заметному улучшению производительности, а RRD стала основой для некоторых других инструментов, включая Smokeping и Graphite.

В проекте Graphite, основанном в 2006 году, для хранения метрик стала использоваться база данных Whisper с архитектурой, похожей на RRD. Приложение Graphite не собирает данные само, а отправляет их с помощью специализированных инструментов, таких как collectd и StatsD, созданных в 2005 и 2010 годах соответственно.

Отсюда можно сделать вывод, что когда-то построение графиков и оповещение были совершенно разными задачами, выполняемыми разными инструментами. Вы можете написать сценарий для оценки запроса в Graphite и генерировать оповещения на этой основе, но большинство проверок, как правило, оказывались в неожиданных состояниях, например, когда процесс не запущен.

Еще один пережиток той эпохи – ручной подход к администрированию компьютерных служб. Службы развертывались на отдельных машинах, и их поддержкой заботливо занимались системные администраторы. Преданные своему делу инженеры оперативно реагировали на предупреждения, которые могли указывать на проблему. С ростом популярности облачных технологий, таких как EC2, Docker и Kubernetes, инженеры пришли к осознанию, что обращение с отдельными машинами и службами как с домашними питомцами, когда каждому уделяется отдельное внимание, плохо масшта-

[1] У Брайана остались приятные воспоминания о настройке MRTG в начале 2000-х, о написании сценариев для отчетов о температуре и использовании сети на моих домашних компьютерах.

бируется. Вместо этого предпочтительнее относиться к ним как к крупному рогатому скоту и управлять ими как группами. Точно так же, как индустрия перешла от ручного управления к использованию таких инструментов, как Chef и Ansible, и постепенно начала использовать такие технологии, как Kubernetes, в мониторинге тоже появилась необходимость совершить подобный переход. Это означает переход от проверок отдельных процессов на отдельных машинах к мониторингу работоспособности службы в целом.

Позднее из двух проектов с открытым исходным кодом, OpenCensus и OpenTracing, появился проект OpenTelemetry. OTel[1] – это спецификация и набор компонентов для организации встроенной телеметрии в проектах. В отношении метрик он совместим с Prometheus при использовании дополнительного сборщика OpenTelemetry[2], отвечающего за сбор и передачу метрик серверу Prometheus.

Вы могли заметить, что мы не упомянули журналирование, трассировку и профилирование. Исторически журналы использовались как ручной инструмент, используемый вместе с `tail`, `grep` и `awk`. Возможно, вам приходилось встречаться с инструментом анализа, таким как AWStats, позволяющим создавать почасовые или посуточные отчеты. В последние годы журналы также использовались в качестве важного компонента мониторинга, например в стеке Elasticsearch, Logstash и Kibana (ELK) и OpenSearch. Трассировка и профилирование обычно выполняются с помощью собственного программного стека; например, для трассировки широко используются Zipkin и Jaeger, а для непрерывного профилирования – Parca и Pyroscope.

Теперь, вкратце познакомившись с графиками и оповещениями, давайте посмотрим, как метрики и журналы вписываются в общий ландшафт. Есть ли еще категории мониторинга, кроме этих двух?

Категории мониторинга

Большая часть мониторинга сводится к одному и тому же: событиям. События могут быть практически любыми, в том числе это:

- получение HTTP-запроса;
- отправка HTTP-ответа 400;
- вход в функцию;
- достижение ветки `else` в операторе `if`;
- выход из функции;
- вход пользователя;
- запись данных на диск;
- чтение данных из сети;
- запрос дополнительной памяти у ядра.

---

[1] OTel – неофициальное название OpenTelemetry.

[2] В то время, когда мы работали над этой книгой, на саммите разработчиков Prometheus было решено добавить в сервер Prometheus поддержку протокола OTel в будущем, правда, не были установлены четкие временные рамки воплощения этого решения в жизнь.

Все события имеют некоторый контекст. HTTP-запрос будет иметь исходящий и входящий IP-адреса, URL назначения, cookie и идентификатор пользователя, отправившего запрос. В HTTP-ответе будет указана продолжительность обработки запроса, код состояния HTTP и длина тела ответа. События, связанные с функциями, имеют стек вызовов функций, по которому можно отследить путь, приведший к вызову данной функции, и все, что связано с этим вызовом, например HTTP-запрос.

Наличие контекста для всех событий может пригодиться для отладки и изучения особенностей работы системы с технической и с коммерческой точек зрения, но такой объем данных нецелесообразно обрабатывать и хранить. Поэтому на практике используются четыре основных подхода к сокращению этого объема данных до более приемлемого уровня, а именно профилирование, трассировка, журналирование и получение метрик.

Профилирование

В профилировании используется подход, согласно которому для событий сохраняется не весь контекст, а только ограниченная его часть, характеризующая некоторый небольшой интервал времени.

Примером инструмента профилирования может служить Tcpdump. Он фиксирует сетевой трафик на основе заданного фильтра. Это важный инструмент отладки, но его нельзя запустить в постоянную работу, потому что он быстро исчерпает место на диске.

Другой пример – отладочные сборки двоичных файлов, фиксирующие данные профилирования. Они предоставляют множество полезной информации, но процедуры сбора всей этой информации, такой как фиксация времени вызова каждой функции, сильно влияют на производительность, а это означает, что профилирование редко бывает целесообразно запускать в промышленном окружении на постоянной основе.

Механизм расширенных фильтров пакетов Беркли (enhanced Berkeley Packet Filters, eBPF) в ядре Linux позволяет детально исследовать события, возникающие в ядре, от операций с файловой системой до сетевых действий. Этот механизм обеспечивает такой уровень проникновения вглубь происходящего, какой ранее был недоступен. eBPF обладает и другими преимуществами, такими как переносимость и безопасность. Мы настоятельно рекомендуем прочитать работы Брендана Грегга (Brendan Gregg; https://oreil.ly/n15mM) на эту тему.

Профилирование, по сути, является тактическим приемом отладки. Для осуществления профилирования на протяжении долгого времени необходимо предусмотреть сокращение объема данных или перейти на использование методов *непрерывного профилирования*, позволяющих осуществлять сбор и хранение накапливаемых данных.

Главное отличие непрерывного профилирования – снижение частоты сбора метрик для сокращения объема данных и уменьшения накладных расходов. Один из новых инструментов непрерывного профилирования, Parca Agent на основе eBPF (https://parca.dev/), использует частоту 19 Гц[1]. Как следствие, для

[1] Сравните с частотой 100 Гц в среде выполнения Go или даже 10 000 Гц в Chromium.

накопления статистически значимых объемов данных требуются минуты, а не секунды, но при этом он способен собрать данные, необходимые для понимания, какие участки кода потребляют бóльшую часть процессорного времени, и помочь повысить оптимизировать именно эти участки кода.

Трассировка

Обычно при выполнении трассировки рассматриваются не все события, а только некоторая их доля, например одно из ста, проходящих через исследуемые функции. В ходе трассировки отмечаются функции, находящиеся в стеке вызовов, а также часто время выполнения каждой из этих функций. Из этой информации можно получить представление о том, где программа тратит больше всего времени и какие пути выполнения в коде вызывают наибольшие задержки.

Вместо создания моментальных снимков трассировки стека в интересующих точках некоторые системы трассировки определяют и записывают время работы каждой функции, находящейся в стеке ниже интересующей функции. Например, система может выбирать один из сотни HTTP-запросов и для каждого выбранного запроса фиксировать время, затраченное на взаимодействие с серверными программными компонентами, такими как базы данных и кеши. Эта информация позволяет увидеть, как различаются времена в зависимости от таких факторов, как попадание в кеш и промахи кеша.

Распределенная трассировка делает еще один шаг вперед, позволяя выполнять трассировку между процессами, присваивая уникальные идентификаторы запросам, которые передаются между процессами при вызове удаленных процедур (Remote Procedure Call, RPC), и определяя, является ли этот запрос тем, который следует отслеживать. Затем трассировки различных процессов и машин могут объединяться по идентификатору запроса. Это очень важный инструмент для отладки архитектур распределенных микросервисов. В качестве примеров технологий в этой области можно назвать OpenZipkin и Jaeger.

Выполняя трассировку, важно иметь в виду, что частота выборки и объемы данных могут оказывать существенное влияние на производительность инструментов.

Журналирование

Механизм журналирования получает ограниченный набор событий и фиксирует некоторый контекст для каждого из них. Например, он может получать все входящие HTTP-запросы или все исходящие запросы к базе данных. Чтобы не потреблять слишком много ресурсов, каждая запись в журнале обычно ограничивается примерно сотней полей. Помимо этого, необходимо учитывать пропускную способность и объем хранилища.

Например, для сервера, обрабатывающего 1000 запросов в секунду, когда на каждый запрос создается запись со 100 полями по 10 байт каждый, размер журнала будет увеличиваться на 1 Мбайт в секунду. Это потребует полосы