



# Содержание

<b>Вступление</b> .....	12
<b>Об авторах</b> .....	15
<b>Колофон</b> .....	16
<b>Часть I. ВВЕДЕНИЕ</b> .....	17
<b>Глава 1. Что такое Prometheus?</b> .....	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
<b>Глава 2. Настройка и запуск Prometheus</b> .....	34
Запуск Prometheus.....	34
Использование браузера выражений.....	38
Запуск экспортера узла.....	42
Оповещение.....	46
<b>Часть II. МОНИТОРИНГ ПРИЛОЖЕНИЙ</b> .....	53
<b>Глава 3. Инструментирование</b> .....	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
<b>Глава 4. Экспортирование метрик .....</b>	<b>76</b>
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
<b>Глава 5. Метки .....</b>	<b>97</b>
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
<b>Глава 6. Создание дашбордов с Grafana .....</b>	<b>112</b>
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 временной шкалы состояний .....	125
Переменные шаблона.....	126
<b>Часть III. МОНИТОРИНГ ИНФРАСТРУКТУРЫ.....</b>	<b>130</b>
<b>Глава 7. Node Exporter.....</b>	<b>131</b>
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
<b>Глава 8. Обнаружение служб.....</b>	<b>144</b>
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

<b>Глава 9. Контейнеры и Kubernetes</b> .....	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
<b>Глава 10. Популярные экспортеры</b> .....	187
Consul .....	187
MySQLd .....	189
Grok Exporter .....	190
Blackbox .....	193
ICMP .....	194
TCP .....	199
HTTP .....	201
DNS .....	203
Настройка Prometheus .....	205
<b>Глава 11. Работа с другими системами мониторинга</b> .....	209
Другие системы мониторинга .....	209
InfluxDB .....	211
StatsD .....	212
<b>Глава 12. Разработка экспортеров</b> .....	215
Consul Telemetry .....	215
Пользовательские сборщики .....	219
Метки .....	223
Методические рекомендации .....	224
<b>Часть IV. PROMQL</b> .....	226
<b>Глава 13. Введение в PromQL</b> .....	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
<b>Глава 14. Операторы агрегирования.....</b>	<b>246</b>
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
<b>Глава 15. Двухместные операторы.....</b>	<b>258</b>
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
<b>Глава 16. Функции.....</b>	<b>273</b>
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
<b>Глава 17. Правила записи.....</b>	<b>294</b>
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
Именованное правило записи.....	302
<b>Часть V. ОПОВЕЩЕНИЕ.....</b>	<b>306</b>
<b>Глава 18. Оповещение.....</b>	<b>307</b>
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

<b>Глава 19. Alertmanager</b> .....	321
Конвейер уведомлений .....	321
Конфигурационный файл .....	322
Дерево маршрутизации .....	323
Приемники .....	330
Подавление .....	340
Веб-интерфейс Alertmanager.....	341
<b>Часть VI. РАЗВЕРТЫВАНИЕ</b> .....	344
<b>Глава 20. Безопасность на стороне сервера</b> .....	345
Функции безопасности в Prometheus .....	345
Включение TLS .....	345
Дополнительные параметры TLS .....	347
Включение базовой аутентификации.....	347
<b>Глава 21. Собираем все вместе</b> .....	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
<b>Предметный указатель</b> .....	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](http://www.dmkpress.com) или [www.дмк.рф](http://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](mailto: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](http://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](mailto: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/](http://dmkpress.com/authors/publish_book/) или напишите в издательство по адресу [dmkpress@gmail.com](mailto: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](mailto: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 относятся 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](mailto: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), Бартомею 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.

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<sup>1</sup> 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<sup>2</sup> упрощает экспортирование метрик Prometheus. Другие системы мониторинга, как с открытым исходным кодом, так

---

<sup>1</sup> Первым стал фреймворк Kubernetes.

<sup>2</sup> 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<sup>1</sup> 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.

---

<sup>1</sup> 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<sup>1</sup>. 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-

---

<sup>1</sup> 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<sup>1</sup> – это спецификация и набор компонентов для организации встроенной телеметрии в проектах. В отношении метрик он совместим с Prometheus при использовании дополнительного сборщика OpenTelemetry<sup>2</sup>, 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.

<sup>1</sup> OTel – неофициальное название OpenTelemetry.

<sup>2</sup> 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 Tcpcdump. 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<sup>1</sup>. Как следствие, для

<sup>1</sup> 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 большую 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 интересных 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 кеша. 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

пропускания сети 100 Мбит и 84 Гбайт пространства в хранилище в день только для журналирования.

Большое преимущество журналирования – отсутствие (обычно) затрат ресурсов на выборку событий, поэтому даже при ограниченном количестве полей журналирование помогает определить влияние медленных запросов на одного конкретного пользователя, обращающегося к одной конкретной конечной точке API.

Подобно мониторингу, журналирование тоже по-разному понимается разными людьми, что может вызвать путаницу. Различные виды журналов имеют разные области применения, долговечность и требования к хранилищу. Всего можно выделить четыре общие категории.

#### *Журналы транзакций*

Важные записи, которые должны храниться в безопасности, и, возможно, вечно. К этой категории относится все, что касается денег или важных функций, ориентированных на пользователя.

#### *Журналы запросов*

Информация о каждом HTTP-запросе или каждом вызове базы данных помещается в журнал запросов. Информация из этих журналов может использоваться для реализации функций, ориентированных на пользователя, или только для внутренней оптимизации. Обычно нежелательно терять эти журналы, но и потеря некоторых из них не означает конца света.

#### *Журналы приложений*

Не вся журналируемая информация относится к запросам; какая-то ее часть характеризует работу самого процесса. Типичными ее представителями являются сообщения, выводимые приложением при запуске, фоновыми задачами и другими механизмами на уровне процесса. Эти журналы часто читаются непосредственно человеком, поэтому старайтесь избегать создания более чем нескольких записей в минуту в обычном режиме работы.

#### *Журналы отладки*

Журналы отладки, как правило, очень подробные, поэтому для их поддержки и хранения требуются большие затраты. Обычно они используются только в очень ограниченных по времени периодах отладки и по своим характеристикам близки к профилированию из-за большого объема данных. Требования к надежности и хранению таких журналов, как правило, низкие, и они могут даже не покидать машину, на которой созданы.

Одинаковая интерпретация различных типов журналов может поставить вас в неприятное положение, когда большой объем данных в журналах отладки сочетается с экстремальными требованиями к надежности журналов транзакций. Поэтому обязательно постарайтесь спланировать отделение журналов отладки, чтобы их можно было обрабатывать отдельно.

Примерами систем журналирования могут служить стек ELK, OpenSearch, Grafana Loki и Graylog.

## Метрики

Метрики редко бывают связаны с контекстом и часто просто представляют совокупность различных типов событий во времени. Чтобы обеспечить потребление ресурсов в разумных пределах, количество отслеживаемых метрик должно быть ограничено: 10 000 на процесс – разумная верхняя граница, которую вы должны иметь в виду.

Примерами метрик могут служить: количество полученных HTTP-запросов; время, потраченное на обработку запросов; и количество запросов, обрабатываемых в настоящий момент. Исключая любую информацию о контексте, можно ограничить в разумных пределах объем данных и время на их обработку.

Однако это не означает, что контекст всегда игнорируется. В отношении HTTP-запросов, например, можно создать отдельную метрику для каждого URL. Но не забывайте о рекомендации в 10 000 метрик, поскольку каждый отдельный URL теперь считается метрикой. Использовать такой контекст, как адрес электронной почты пользователя, было бы неразумно, потому что такие сведения имеют неограниченную кардинальность<sup>1</sup>.

Вы можете использовать метрики для определения задержек и объемов данных, обрабатываемых каждой подсистемой в ваших приложениях, чтобы упростить определение причин замедления. Записи в журналах не могут иметь столько полей, но, узнав, какая подсистема виновна в замедлении приложения, вы сможете задействовать журналирование, чтобы выяснить, какие именно запросы пользователей вносят наибольший вклад в задержку.

Именно здесь компромисс между журналами и метриками становится наиболее очевидным. Метрики позволяют собирать информацию о событиях со всего процесса, но, как правило, не более чем с одним или двумя контекстными полями с ограниченной кардинальностью. Журналы позволяют собирать информацию обо всех событиях одного типа, но могут фиксировать только сотню полей контекста с неограниченной кардинальностью. Важно понимать, что такое кардинальность и какие ограничения она накладывает на метрики, о чем мы и поговорим в последующих главах.

Prometheus, как система мониторинга на основе метрик, предназначена для отслеживания общего состояния системы, ее поведения и производительности, а не отдельных событий. Другими словами, Prometheus может сообщить, что за последнюю минуту было получено 15 запросов, на обработку которых ушло 4 с, что привело к 40 обращениям к базе данных, 17 попаданиям в кеш и двум покупкам клиентов. Стоимость и пути выполнения в коде для обработки отдельных вызовов – это сфера профилирования или журналирования.

Теперь, получив представление о том, какое место Prometheus занимает в общем пространстве мониторинга, давайте рассмотрим различные компоненты этой системы.

---

<sup>1</sup> Адреса электронной почты также часто относятся к категории персональной информации, что влечет за собой проблему соблюдения требований и конфиденциальности, которой лучше избегать при мониторинге.

# Архитектура Prometheus

На рис. 1.1 показана общая архитектура Prometheus. Prometheus обнаруживает цели для извлечения информации, используя механизм обнаружения служб. Этими целями могут быть ваши собственные инструментированные приложения или сторонние приложения, сведения о которых вы можете получать с помощью экспортеров. Собранные данные сохраняются и могут отображаться на дашбордах с помощью PromQL или использоваться для рассылки оповещений диспетчером уведомлений Alertmanager, который преобразует данные в страницы, электронные письма и другие формы уведомлений.

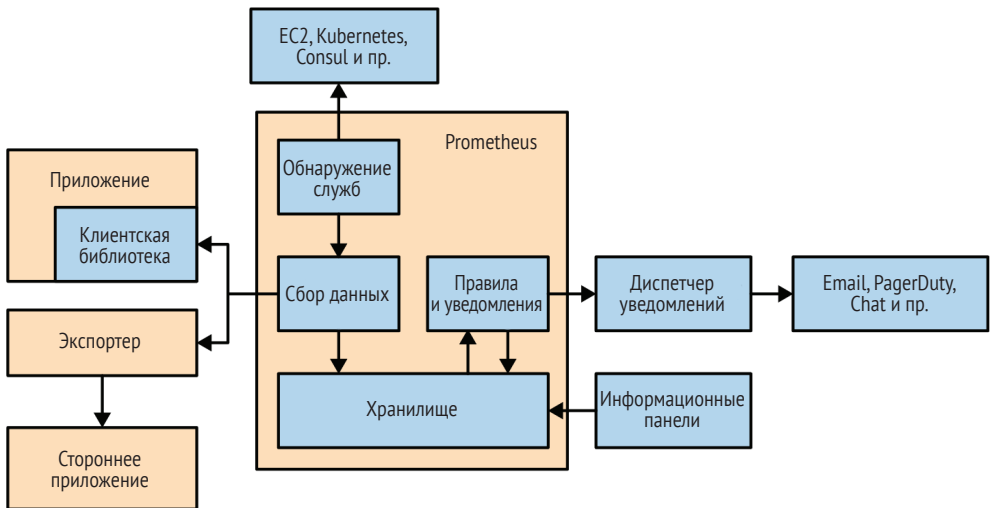


Рис. 1.1 ❖ Архитектура Prometheus

## Клиентские библиотеки

Метрики обычно не возникают в приложениях как по волшебству; кто-то должен добавить инструменты для их создания. И для этой цели используются клиентские библиотеки. Добавив всего пару строк кода, вы можете определить метрику и внедрить в приложение нужные инструменты. Это называется *прямым инструментированием*.

Клиентские библиотеки доступны для всех основных языков программирования и операционных систем. Проект Prometheus предоставляет официальные клиентские библиотеки на Go, Python, Java/JVM, Ruby и Rust. Существует также множество сторонних клиентских библиотек, например для C#/Net, Node.js, Haskell и Erlang.

## Официальные и неофициальные библиотеки

Не надо бояться использовать неофициальные или сторонние клиентские библиотеки. Команда проекта Prometheus просто не в состоянии реализовать и поддерживать различные библиотеки интеграции с сотнями приложений и систем, с которыми вам, возможно, придется иметь дело. Поэтому подавляющее большинство таких библиотек в экосистеме являются сторонними. А чтобы обеспечить согласованность, доступны рекомендации по написанию этих библиотек.

Клиентские библиотеки берут на себя заботу обо всех деталях, таких как безопасность потоков, выполнение рутинных задач и создание сообщений в формате Prometheus и/или OpenMetrics в ответ на HTTP-запросы. Как уже отмечалось, мониторинг на основе метрик не связан с отслеживанием отдельных событий, поэтому потребление памяти клиентской библиотекой не увеличивается с увеличением количества событий. Скорее, объем потребляемой памяти зависит от количества имеющихся метрик.

Если одна из библиотек, включенных в ваше приложение, уже инструментирована средствами Prometheus, то они будут задействоваться автоматически. То есть, инструментировав ключевую библиотеку, такую как RPC-клиент, вы сможете получать использовать это преимущество во всех своих приложениях.

Некоторые метрики, такие как потребление процессора и статистика сборки мусора, обычно предоставляются клиентскими библиотеками по умолчанию.

Клиентские библиотеки могут отправлять метрики не только в текстовых форматах Prometheus и OpenMetrics. Prometheus – это открытая экосистема, и те же самые функции, что используются для создания метрик в текстовых форматах, могут применяться для создания метрик в других форматах или для передачи данных другим системам инструментирования. Точно так же есть возможность получать метрики из других систем инструментирования и передавать их клиентской библиотеке Prometheus, если вы еще не заменили их средствами Prometheus.

## Экспортеры

Не всякий код, который вы запускаете, доступен вам для изменения, и прямое его инструментирование просто невозможно. Например, едва ли можно ожидать, что ядро операционной системы начнет выводить метрики в формате Prometheus через HTTP.

Однако такое ПО часто имеет некоторый интерфейс, через который можно получить доступ к метрикам. Это может быть специальный формат, требующий специальной обработки, что характерно для многих метрик Linux, или хорошо зарекомендовавший себя стандарт, такой как SNMP.

*Экспортер* (exporter) – это программный компонент, развертываемый рядом с приложением, из которого требуется получить метрики. Он при-



нимает запросы от Prometheus, извлекает необходимые данные из приложения, преобразует их в правильный формат и возвращает их в Prometheus. Экспортер можно представить себе как маленький прокси-сервер, преобразующий данные из формата метрик приложения в формат представления Prometheus.

В отличие от прямого инструментирования, которое можно применить к своему коду, экспортеры используют другой способ инструментирования, известный как *настраиваемые сборщики* (custom collectors), или *ConstMetrics*<sup>1</sup>.

Самое замечательное, что, учитывая многочисленность сообщества Prometheus, нужный вам экспортер почти наверняка уже существует и может использоваться без особых усилий с вашей стороны. Если у экспортера отсутствует интересующая вас метрика, то вы всегда сможете послать разработчикам запрос на добавление новой особенности и тем самым поспособствовать совершенствованию экспортера.

## Обнаружение служб

После инструментирования приложения настройки экспортеров Prometheus должен узнать, где они находятся, чтобы знать, что отслеживать и вовремя определять сбои. В динамических средах нельзя просто составить список приложений и экспортеров один раз, так как он быстро устареет. И в таких случаях вам поможет механизм обнаружения служб.

Вероятно, у вас уже есть база данных с информацией о ваших машинах, приложениях и том, что они делают. Эта информация может храниться в базе данных Chef, в файле инвентаризации Ansible, в тегах ваших экземпляров EC2, в метках и аннотациях Kubernetes или, может быть, просто в вашей вики-документации.

Prometheus способна интегрироваться со многими распространенными механизмами обнаружения сервисов, такими как Kubernetes, EC2 и Consul. Существует также универсальная интеграция для тех, чьи окружения немного отличаются от общепринятых (см. разделы «Файл» и «HTTP» в главе 8).

Тем не менее остается еще одна проблема. Тот факт, что Prometheus имеет список машин и служб, не означает, что мы знаем, как они вписываются в вашу архитектуру. Например, вы можете использовать тег Name в EC2<sup>2</sup>, чтобы указать, какое приложение работает на компьютере, тогда как другие могут использовать тег app.

Поскольку каждая организация делает это немного по-своему, Prometheus позволяет настроить сопоставление метаданных из службы обнаружения с целями мониторинга и их метками, предлагая возможность *изменения меток*.

<sup>1</sup> ConstMetric – это разговорный термин и происходит от имени функции MustNewConstMetric в клиентской библиотеке Go, используемой для создания метрик экспортерами, написанными на Go.

<sup>2</sup> Тег Name в EC2 – это имя экземпляра EC2, отображаемое в веб-консоли EC2.

## Извлечение данных

Обнаружение служб и поддержка изменения меток позволяют получить список целей для мониторинга. Имея такой список, Prometheus должен как-то получить метрики. Для этого она отправляет HTTP-запрос на *извлечение*, анализирует ответ и сохраняет полученные метрики в хранилище. Дополнительно Prometheus добавляет несколько полезных метрик, например признак удачного получения данных и сколько времени это заняло. Операции извлечения выполняются регулярно, обычно каждые 10-60 с для каждой цели.

### Получение по запросу или пассивный прием

Prometheus – это система, основанная на получении метрик по запросу (pull). Она сама решает, когда и что извлекать, основываясь на своей конфигурации. Существуют также системы, предполагающие пассивный прием (push), когда цели мониторинга сами решают, когда и как часто они будут передавать свои метрики.

В интернете ведутся бурные споры об этих двух подходах, которые во многом напоминают дебаты – что лучше, Vim или EMACS. Достаточно сказать, что оба подхода имеют свои плюсы и минусы, и в целом выбор того или иного не имеет большого значения.

Как пользователь Prometheus, вы должны понимать, что получение по запросу укоренилось в ядре Prometheus, и пытаться заставить эту систему принимать данные, посылаемые по инициативе целей мониторинга, в лучшем случае неразумно.

## Хранилище

Prometheus хранит данные локально в собственной базе данных. Распределенные системы сложно сделать надежными, поэтому Prometheus не пытается использовать какую-либо кластеризацию. Кроме надежности этот аспект также упрощает запуск Prometheus.

За прошедшие годы технологии хранения претерпели ряд изменений, и система хранения в Prometheus 2.0 стала третьей итерацией. Она может обрабатывать миллионы выборок в секунду, что позволяет контролировать тысячи машин с помощью одного сервера Prometheus. Используемый алгоритм сжатия может достигать 1.3 байта на выборку для реальных данных. В качестве основы рекомендуется использовать твердотельные накопители, такие как SSD, но это не является обязательным условием.

## Дашборды

Prometheus предлагает ряд HTTP API, позволяющих получать исходные данные и обрабатывать запросы PromQL. Их можно использовать для создания

графиков и дашбордов. По умолчанию Prometheus предоставляет *браузер выражений* (expression browser). Он использует эти API и подходит для обработки специальных запросов и исследования данных, но это не типичная система дашбордов.

Для создания дашбордов лучше использовать Grafana. Она имеет широкий спектр функций, включая официальную поддержку Prometheus как источника данных, может создавать самые разнообразные дашборды, такие как на рис. 1.2. Grafana поддерживает возможность взаимодействия с несколькими серверами Prometheus даже в рамках одного дашборда.



Рис. 1.2 ❖ Дашборд Grafana

## Правила записи и оповещения

PromQL и поддержка хранилища – мощные и эффективные механизмы, но оперативное агрегирование метрик с тысяч машин в процессе визуализации графиков может немного отставать. Правила записи позволяют вычислять выражения PromQL через заданные интервалы времени и загружать их результаты в механизм хранения.

Правила оповещения – еще одна форма правил записи. Они также регулярно вычисляют выражения PromQL, и любые результаты этих выражений становятся уведомлениями. Уведомления отправляются в диспетчер уведомлений Alertmanager.

## Управление уведомлениями

Диспетчер уведомлений Alertmanager получает сообщения от серверов Prometheus и превращает их в уведомления. Уведомления могут включать адрес электронной почты, приложения чата, такие как Slack, и службы, такие как PagerDuty.

Alertmanager не просто слепо превращает сообщения в уведомления по принципу «один к одному». Связанные уведомления могут объединяться в одно уведомление для предотвращения шторма уведомлений<sup>1</sup>, и для каждой из команд сотрудников можно настроить разные маршруты рассылки уведомлений. Также есть возможность отключить уведомления, например чтобы отложить проблему, решение которой запланировано в ближайший период технического обслуживания.

На отправке уведомлений роль Alertmanager заканчивается; для управления реакцией людей на инциденты следует использовать такие службы, как PagerDuty и системы регистрации инцидентов.



Уведомления и соответствующие пороговые значения настраиваются в Prometheus, а не в Alertmanager.

## Долгосрочное хранение

Поскольку Prometheus хранит данные только на локальной машине, вы ограничены объемом дискового пространства на этой машине<sup>2</sup>. Обычно интерес представляют только самые свежие данные, полученные, например, за последние сутки или чуть больше, но для перспективного планирования развития желателен более длительный срок хранения.

Prometheus не предлагает кластерных решений для хранения данных на нескольких машинах, но имеет API для чтения и записи в удаленные хранилища, что позволяет перекладывать роль хранилищ на другие системы и прозрачно выполнять запросы PromQL как к локальным, так и к удаленным данным.

## Чем Prometheus не является

Теперь, получив представление о том, какое место Prometheus занимает в более широком ландшафте мониторинга и его основных компонентах,

<sup>1</sup> Уведомление сообщает дежурному инженеру о необходимости незамедлительно изучить проблему и устранить ее. Уведомления могут рассылаться на традиционные радиопейджеры, но в наши дни, скорее всего, рассылка будет осуществляться на мобильные телефоны в виде SMS, push-уведомлений или телефонных звонков. Шторм уведомлений возникает, когда рассылается большое количество уведомлений за короткий промежуток времени.

<sup>2</sup> Современные машины могут хранить огромные объемы данных локально, поэтому вам может не понадобиться отдельная кластерная система хранения.

рассмотрим некоторые случаи, когда Prometheus оказывается не лучшим выбором.

Как система, основанная на метриках, Prometheus не подходит для хранения журналов событий или отдельных событий. Она также не лучший выбор для данных с высокой кардинальностью, таких как адреса электронной почты или имена пользователей.

Prometheus предназначена для оперативного мониторинга, когда небольшие неточности и состояния гонки из-за таких факторов, как планирование выполнения процессов в ядре и неудачные попытки сбора информации, являются обычным делом. Prometheus идет на компромисс и предпочитает предоставлять данные, которые на 99.9 % верны, вместо остановки мониторинга в ожидании идеальных данных. Таким образом, в приложениях, связанных с финансами или выставлением счетов, Prometheus следует использовать с большой осторожностью.

В следующей главе мы покажем вам, как запустить Prometheus и настроить простой мониторинг.