Prometheus

Архитектурный документ

Список авторов

Насыхова Анастасия Артемовна БПИ201

Раздел регистрации изменений

Версия документа	Дата	Описание изменения	Автор изменения
1	23.02.2023	Добавление раздела №1.	Насыхова А. А.
2	24.02.2023	Добавление раздела №2.	Насыхова А. А.
3	25.02.2023	Добавление раздела №3.	Насыхова А. А.
4	26.02.2023	Добавление раздела №4.	Насыхова А. А.
5	27.02.2023	Редактирование раздела №4, уточнение пунктов №4.5, 4.9.	Насыхова А. А.
6	28.02.2023	Добавление раздела №5.	Насыхова А. А.
7	01.03.2023	Добавление раздела №6; Итоговое оформление документа.	Насыхова А. А.

1. Введение

1.1. Название проекта

Prometheus

1.2. Задействованные архитектурные представления

Структура архитектурного документа

1.3. Контекст задачи и среда функционирования системы

Prometheus – инструмент для мониторинга. Он записывает показатели в базу данных временных рядов с гибкими запросами и оповещением в реальном времени.

Prometheus предназначен, в основном, для сферы DevOps, но также может применяться и в сфере здравоохранения, и в финансовых услугах.

1.4. Рамки и цели проекта

Prometheus был разработан в SoundCloud, когда компания обнаружила, что ее существующие метрики и решения для мониторинга (с использованием StatsD и Graphite) недостаточны для их потребностей.

Цель разработки: создать инструмент, удовлетворяющий требованиям компании (многомерная модель данных, масштабируемый сбор данных, мощный язык запросов, простота работы и все в одном инструменте).

2. Архитектурные факторы (цели и ограничения)

2.1. Ключевые заинтересованные лица

При анализе области применения системы были выявлены следующие заинтересованные лица:

Действующее лицо	**Заинтересованность в системе**
Разработчик проекта	Простота поддержки и введения новой функциональности.
Пользователь	Легкая совместимость с инструментом для визуализации собранных метрик; Простота подключения инструмента; Быстрые запросы в инструменте; Удобная система уведомления о собранных метриках.

2.2. Ключевые требования к системе

При анализе ключевых заинтересованных лиц были выделены следующие ключевые требования к системе:

- Гибкий язык запросов для выборки и агрегации данных;
- Удобная система уведомлений;
- Многомерная модель данных, с быстрым доступом к значениям по ключу;
- Отсутствие зависимости от распределенных баз данных;
- Поддержка нескольких режимов построения графиков и панелей мониторинга;
- Встроенные экспортеры и возможность добавить собственные экспортеры.

2.3. Ключевые ограничения

При анализе документации были выделены следующие ключевые ограничения системы:

- Инструментирование (добавление клиентских библиотек в приложение, чтобы они предоставляли метрики Prometheus) доступно для Python, Java, Ruby, Go, Node, C#;
- Сервер должен знать API адреса различных экспортеров.
- Использование экспортеров для взаимной совместимости с другими базами данных и сборщиками метрик.

3. Общее архитектурное решение

Ядром архитектуры является сервер Prometheus, который обрабатывает данные и хранит их. Он очищает объекты для получения информации, необходимой для метрик.

Сбор метрик реализован с помощью механизмов pull и push. Во втором случае используется специальный компонент pushgateway, который необходим для экспорта метрик из защищенных систем или если процесс подключения должен проходить в короткие сроки.

Prometheus предоставляет готовую к использованию коллекцию экспортеров для различных технологий.

PromQL позволяет выбирать необходимую информацию из полученного набора данных и выполнять анализ и визуализацию с этими данными с помощью панелей мониторинга Grafana.

Alertmanager генерирует уведомления из этих данных и отправляет их пользователям, используя электронную почту, социальные сети и другие сервисы.

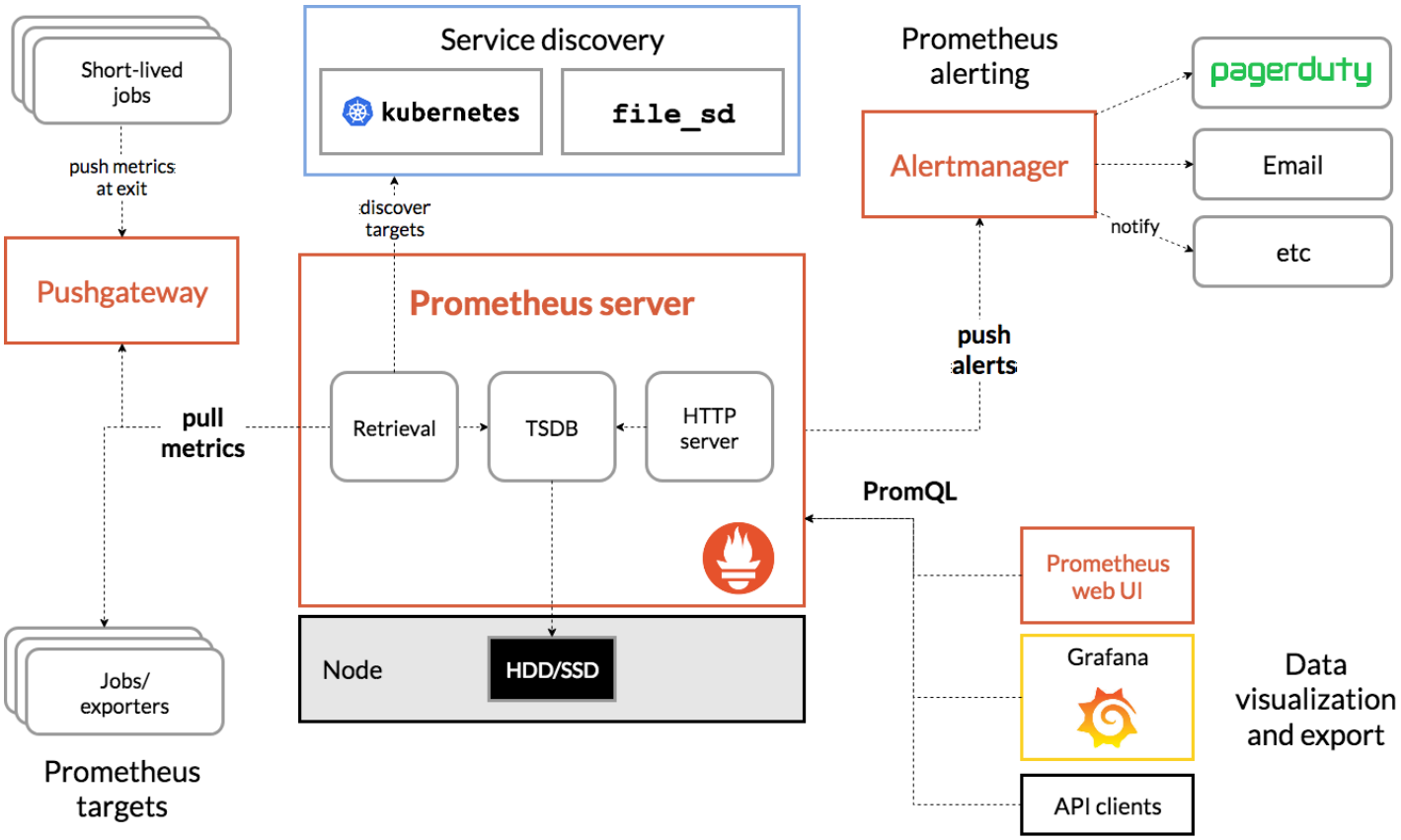

Рис. 1: Архитектура Prometheus и его экосистемных компонентов [1]

3.1. Принципы проектирования

Prometheus разработан в соответствии с микросервисным стилем, когда каждый сервис отвечает за определенную часть функциональности. Например, компонент Alertmanager добавляет возможности удобного оповещения пользователя в реальном времени, а Dashboard предоставляет широкий функционал визуализации с помощью Grafana или других инструментов. При разработке Prometheus особое внимание уделялось принципам надежности и производительности.

4. Архитектурные представления

4.1. Представление прецедентов

Главным и единственным актером является пользователь.

Основные прецеденты: сбор данных, перемещение данных в другое хранилище, визуализация данных, настройка оповещений, генерация новых временных рядов на основе имеющихся.

4.2. Логическое представление

Программа состоит из следующих пакетов:

- Model – пакет с шаблонами экспортеров.
- Promql – пакет для языка запросов.
- TSBD – пакет для базы данных.
- Util – вспомогательные классы.
- Web – пакет для работы web-сервиса. Внутри логика отделена от ui.

4.3. Представление архитектуры процессов

4.4. Физическое представление архитектуры

- Windows 7, 8, 8.1, 10
- macOS 10.10.5 or later
- Ubuntu 18.04+, Debian 10+, openSUSE 15.2+, or Fedora Linux 32+

4.5. Представление развертывания

Существуют различные способы установки Prometheus.

- Предварительно скомпилированные бинарные файлы;
- Образы Docker (образы Docker доступны на Quay.io или Docker Hub);
- Сборка из исходного кода.

4.6. Представление архитектуры данных

Prometheus хранит все данные в виде временных рядов: потоки с метками времени, принадлежащие одной метрике. Помимо сохраненных временных рядов, Prometheus в результате запросов может генерировать производные временные ряды.

4.7. Представление архитектуры безопасности

Prometheus поддерживает TLS и базовую аутентификацию для HTTP эндпоинтов. Скрейпинг таргетов происходит через HTTPS вместо HTTP. Метрики собираются с поддержкой HTTPS, аутентификации по клиентским сертификатам и базовой аутентификации.

4.8. Представление реализации и разработки

Prometheus написан на Golang. В качестве средства визуализации используется сервис Grafana.

4.9. Представление производительности

Производительность системы будет обеспечиваться с помощью эффективного хранения данных, а также с помощью гибкого языка запросов PromQL.

4.10. Атрибуты качества системы

4.10.1 Объем данных и производительность системы

Prometheus работает с данными, имеющими высокую размерность (временные ряды).

4.10.2 Гарантии качества работы системы

Prometheus обещает стабильность API в основной версии и стремится избежать критических изменений для ключевых функций.

Вещи, считающиеся стабильными для версии 2.x:

- Язык запросов и модель данных;
- Правила оповещения и записи;
- Формат файла конфигурации;
- Формат файла оповещения;
- Синтаксис и семантика шаблона консоли.

Тестирование является самой главной гарантией качества Prometheus. Тесты должны покрывать все возможные сценарии работы с системой. Если у пользователя возникает ошибка, он может написать на github в раздел issues или на почту разработчиков.

5. Технические описания отдельных ключевых архитектурных решений

5.1. Техническое описание решения №1: Хранение данных в базе данных

5.1.1 Проблема

Как должно быть обеспечено хранение данных в базе данных?

5.1.2 Идея решения

Следует хранить данные в пары ключ-значение.

5.1.3 Факторы

Требования технического задания не позволяют использовать стандартные структуры данных (массивы и списки), так как требуется обеспечение высокой производительности при больших объемах данных (размерности метрик).

5.1.4 Решение

База данных будет состоять из пар ключ-значение. Название метрики будет выступать в качестве ключа, а в качестве значения – собранные Prometheus показатели, представленные в виде временного ряда.

5.1.5 Мотивировка

Такая система хранения удобна с точки зрения быстрого доступа к значениям и простоты интеграции данных в инструменты визуализации.

5.1.6 Неразрешенные вопросы

Нет.

5.1.7 Альтернативы

Альтернативные решения не рассматривались.

5.2. Техническое описание решения №2: Язык программирования

5.2.1 Проблема

Какой язык для программирования использовать?

5.2.2 Идея решения

Следует использовать язык программирования Golang.

5.2.3 Факторы

Требования технического задания позволяют использовать Golang.

5.2.4 Решение

Все компоненты должны быть написаны на Golang.

5.2.5 Мотивировка

Встроенные возможности языка.

5.2.6 Неразрешенные вопросы

Нет.

5.2.7 Альтернативы

Альтернативные решения не рассматривались.

6. Приложения

6.1. Словарь терминов

Alertmanager – создает оповещения, агрегирует их, группирует, удаляет дубликаты, а затем отправляет уведомления на электронную почту, Pagerduty, Slack и т.д.

PromQL — это язык запросов Prometheus, предоставляющий широкий спектр операций, включая агрегацию, нарезку и нарезку кубиками, прогнозирование и соединение.

Pushgateway – компонент, который сохраняет последний push метрик из пакетных заданий. Это позволяет Prometheus соскребать свои метрики после их закрытия.

Клиентская библиотека – библиотека на каком-либо языке (например, Go, Java, Python, Ruby), которая позволяет напрямую инструментировать код, писать пользовательские экспортеры и предоставлять их Prometheus.

Метрика – числовая величина, соответствующая определенному показателю системы, наблюдаемая на протяжении определенного промежутка времени. Например, для веб-сервера это может быть время запроса, количество активных соединений или количество активных запросов и т. д.

Скрейпинг – Prometheus извлекает метрики через HTTP-вызовы к определенным конечным точкам, указанным в конфигурации Prometheus.

Таргет – объект для очистки.

Экспортер — бинарный файл, работающий вместе с приложением, с которого нужно получить метрики. Экспортер предоставляет метрики Prometheus, преобразуя их в формат, поддерживаемый Prometheus.

6.2. Ссылки на используемые документы

[1] Официальная документация Prometheus [Электронный ресурс] / Режим доступа: https://prometheus.io/docs/introduction/overview/, свободный (дата обращения: 23.02.2023 – 01.03.2023)