**Часть I. Что такое Prometheus?**

**Prometheus** — это база данных временных рядов. Был создан для того, что бы мониторить апплики, которые крутятся на контейнерах. Но может мониторить и аплики, которые крутятся на самих серверах.

Собирает метрики, по тому что происходит в контейнерах (закончилась память на серваке, и по этому он выключил контейнер, который отвечал за синкование двух баз данных в кластере. Эти базы данных отвечали за аутентификацию пользователей, из за этого она отвалилась). Prometheus позволяет установить alerts на то, что бы чекать когда произойдет crash какого то компонента.

Но Prometheus — не просто база данных временных рядов.

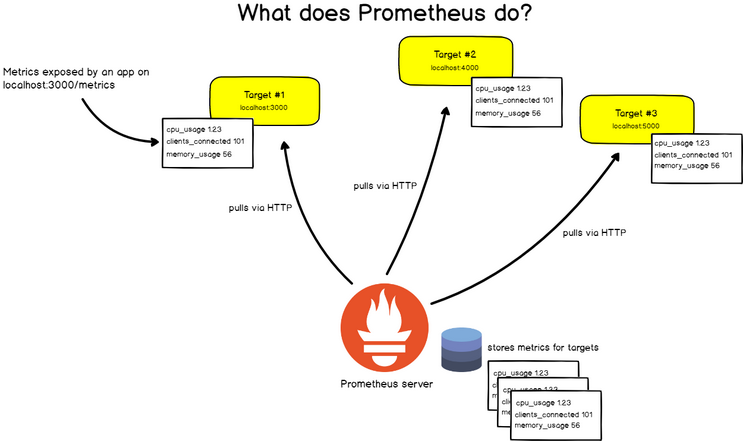
К нему можно присоединить целую экосистему инструментов, чтобы расширить функционал.

Prometheus мониторит самые разные системы: серверы, базы данных, отдельные виртуальные машины, да почти что угодно.

Для этого Prometheus периодически скрейпит свои целевые объекты.

**Что такое скрейпинг?**

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



Возьмем, например, веб-приложение, расположенное по адресу http://localhost:3000. Приложение передает метрики в текстовом формате на некоторый URL. Допустим, http://localhost:3000/metrics.

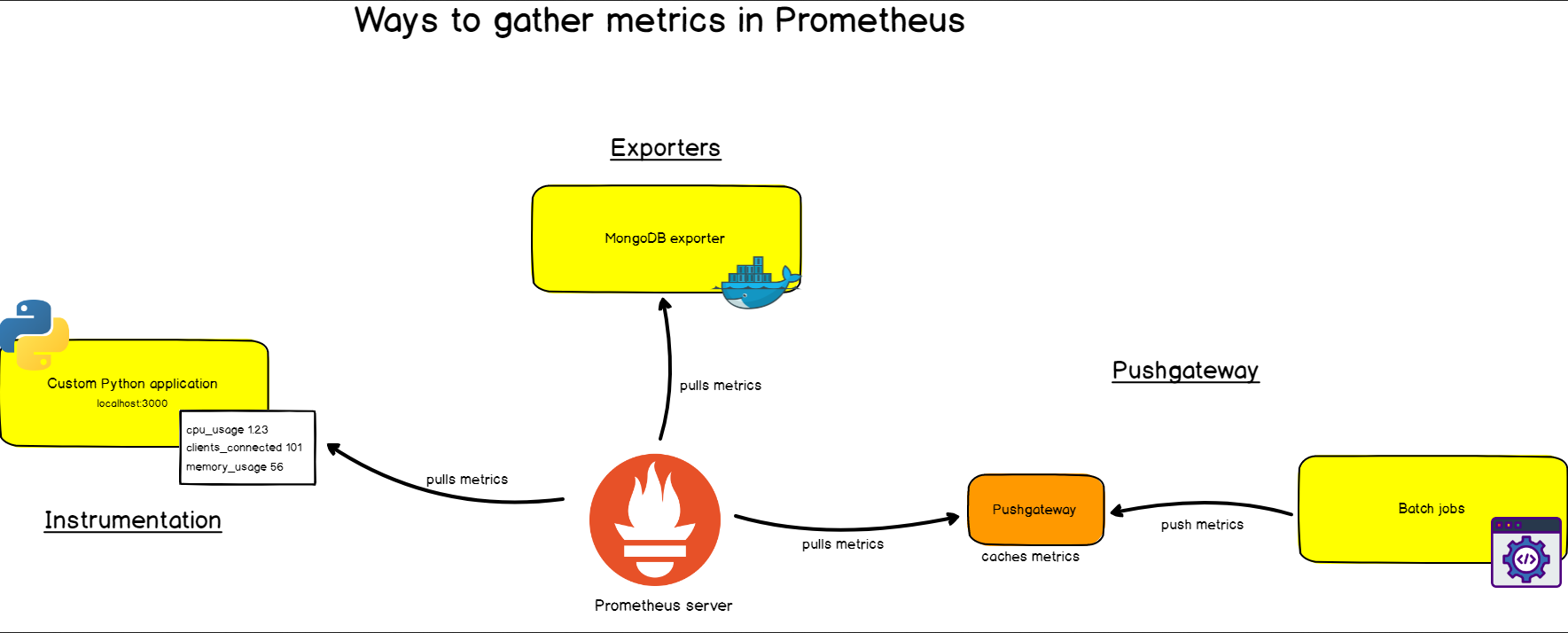
По этому адресу Prometheus с определенными интервалами извлекает данные из целевого объекта.

**Как работает Prometheus?**

Как мы уже сказали, Prometheus состоит из самых разных компонентов.

Во-первых, вам нужно, чтобы он извлекал метрики из ваших систем. Тут есть разные способы

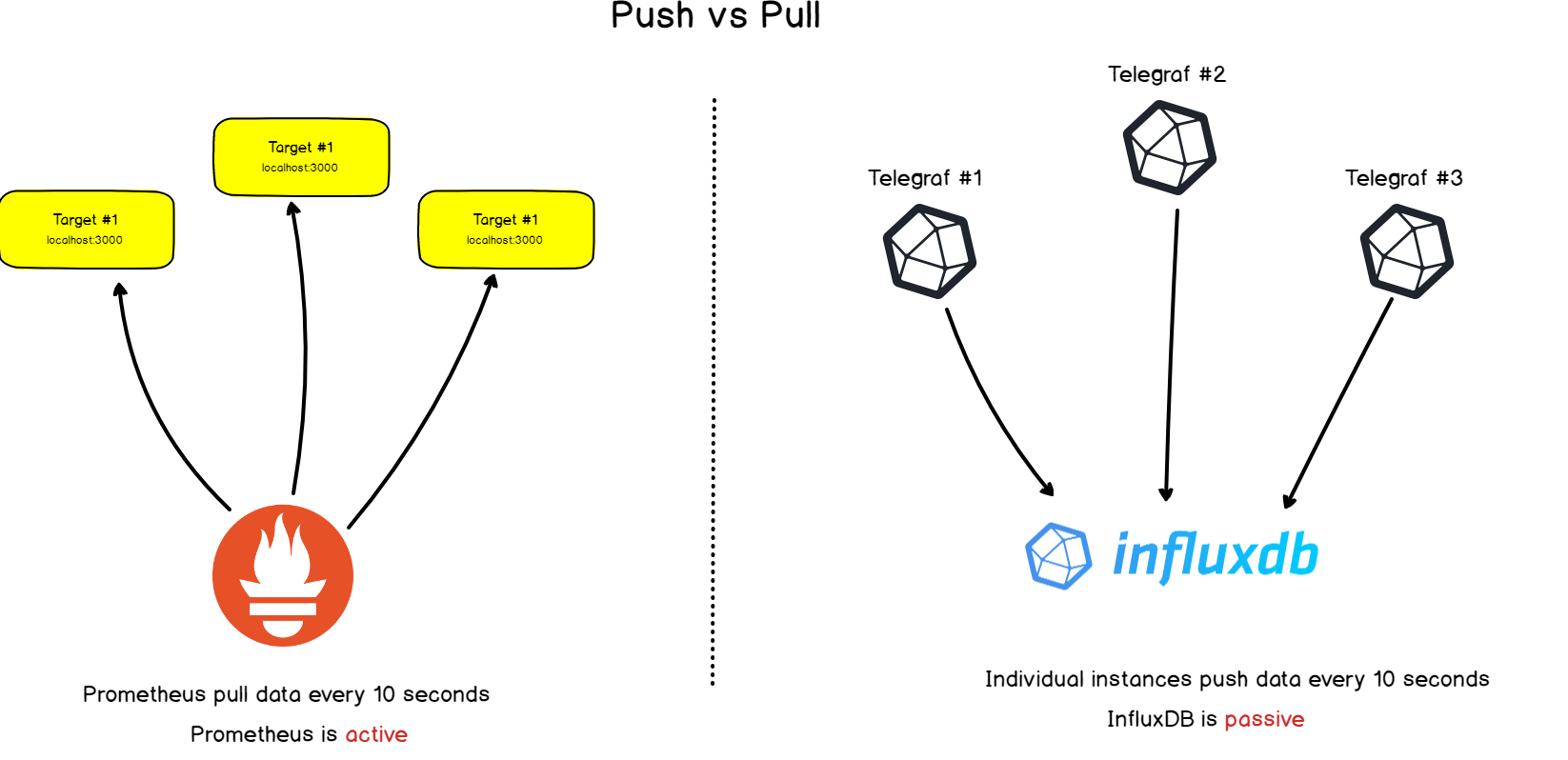
* **Инструментирование** приложения, то есть ваше приложение будет предоставлять совместимые с Prometheus метрики по заданному URL. Prometheus определит его как целевой объект и будет скрейпить с указанным интервалом.
* Использование **готовых экспортеров**. В Prometheus есть целая коллекция экспортеров для существующих технологий. Например, готовые экспортеры для мониторинга машин Linux (Node Exporter), для распространенных баз данных (SQL Exporter или MongoDB Exporter) и даже для балансировщиков нагрузки HTTP (например, HAProxy Exporter).
* Использование **Pushgateway**. Иногда приложения или задания не предоставляют метрики напрямую. Они могут быть не предназначены для этого (например, пакетные задания) или вы сами решили не предоставлять метрики напрямую через приложение.



**Сбор vs. Отправка**

У Prometheus есть заметное отличие от других баз данных временных рядов: **он активно сканирует целевые объекты, чтобы получить у них метрики**.

InfluxDB, например, работает иначе: вы сами напрямую **отправляете ему данные**.



Оба подхода имеют свои плюсы и минусы. На основе доступной документации мы составили список причин, по которым создатели Prometheus выбрали такую архитектуру:

* **Централизованный контроль**. Если Prometheus отправляет запросы целевым объектам, всю настройку мы выполняем на стороне Prometheus, а не отдельных систем.

Prometheus сам решает, **где** и **как часто** проводить скрейпинг.

Если объекты сами отправляют данные, есть риск, что таких данных будет слишком много, и на сервере произойдет сбой. Когда система собирает данные, можно **контролировать частоту сбора** и создавать несколько конфигураций скрейпинга, чтобы выбирать **разную частоту для разных объектов**.

* Prometheus хранит **агрегированные метрики**.

Это дополнение к первой части, где мы обсуждали роль Prometheus.

Prometheus не основан на событиях и этим сильно отличается от других баз данных временных рядов. Он не перехватывает отдельные события с привязкой ко времени (например, перебои с сервисом), **а собирает предварительно агрегированные метрики о ваших сервисах**.

Если конкретно, веб-сервис не отправляет сообщение об ошибке 404 и сообщение с причиной ошибки. Отправляется сообщение о факте, что сервис получил сообщение об ошибке 404 за последние пять минут.

Это главное различие между базами данных временных рядов, которые собирают агрегированные метрики, и теми, что собирают необработанные метрики.

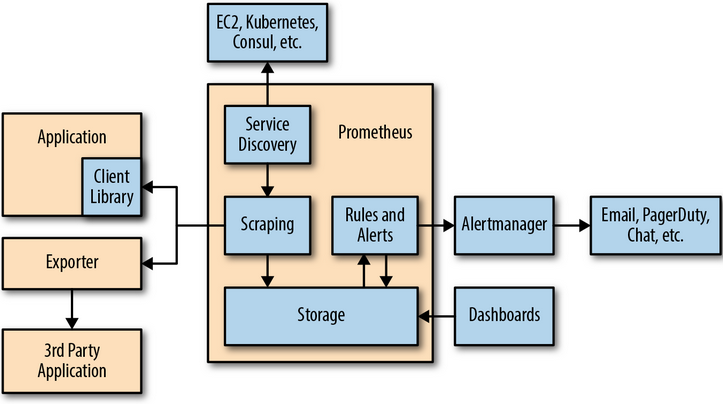
**Развитая экосистема Prometheus**

По сути Prometheus — база данных временных рядов

Но при работе с такими базами данных часто нужно **визуализировать** данные, **анализировать** их и настраивать по ним **оповещения**.

Prometheus поддерживает следующие инструменты, расширяющие его функционал:

* **Alertmanager**. Prometheus отправляет оповещения в Alertmanager на основе кастомных правил, определенных в файлах конфигурации. Оттуда их можно экспортировать в разные конечные точки (например, Pagerduty или Slack).
* **Визуализация данных**. Как и в Grafana, вы можете визуализировать временные ряды прямо в пользовательском веб-интерфейсе Prometheus. Вы можете фильтровать данные и составлять конкретные обзоры происходящего в разных целевых объектах.
* **Обнаружение сервисов**. Prometheus динамически обнаруживает целевые объекты и автоматически скрейпит новые цели по запросу. Это особенно удобно, если вы работаете с контейнерами, которые динамически меняют адреса в зависимости от спроса



**Концепции Prometheus**

**Модель данных «ключ-значение»**

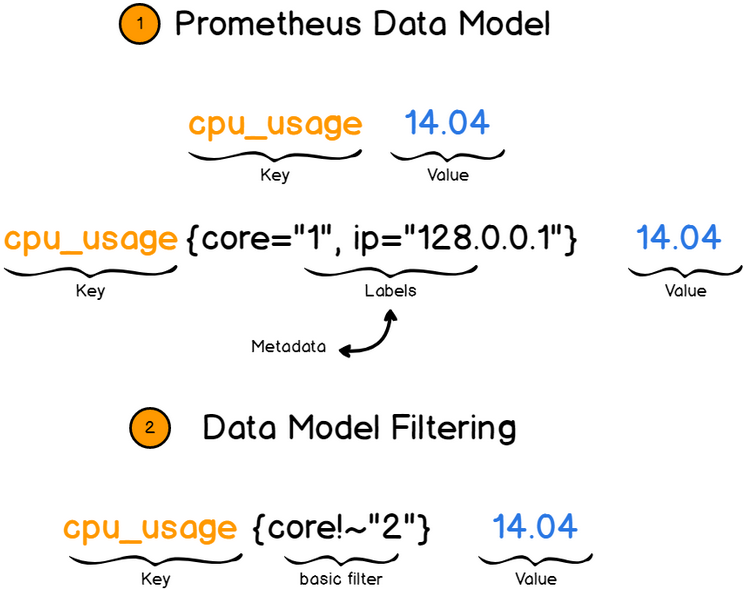
Prometheus работает с **парами «ключ-значение»**. Ключ описывает, что мы измеряем, а значение хранит фактическую величину в виде числа.

* **Помните**: Prometheus не создан для хранения необработанной информации, вроде обычного текста. Он хранит метрики, агрегированные за период времени.

Ключом в данном случае называется **метрика**. Это, например, скорость процессора или занятый объем памяти

Но что если нужно больше деталей о метрике?Например, у процессора 4 ядра, и нам нужно 4 отдельных метрики?

И здесь на помощь приходят **ярлыки**. **Ярлыки** дают больше сведений о метриках, добавляя дополнительные поля. Например, вы описываете не просто скорость процессора, а скорость одного ядра по определенному IP



Потом вы сможете фильтровать метрики по ярлыкам и просматривать только нужную информацию.

**Типы метрик**

При мониторинге с Prometheus метрики можно описать четырьмя способами

**Счетчик**

Это, наверное, самый простой тип метрик. Счетчик, как понятно из названия, **считает элементы за период времени**.

Если вы хотите посчитать, например, ошибки HTTP на серверах или посещения веб-сайта, используйте **счетчик**.

И по логике, разумеется, **счетчик может только увеличивать или обнулять число**, поэтому не подходит для значений, которые могут уменьшаться, или для отрицательных значений.

С его помощью особенно удобно считать **количество наступлений** определенного события за период времени, т. е. показатель изменения метрики со временем.

*А если нужно измерить, допустим, используемую память за определенный период?  
Эта величина может уменьшаться. Как посчитать ее с Prometheus?*

**Измерители**

Измерители имеют дело со значениями, которые **со временем могут уменьшаться**. Их можно сравнить с термометрами — если посмотреть на термометр, увидим текущую температуру.

*Но если измерители могут увеличиваться и уменьшаться и принимать положительные и отрицательные значения, то выходит, они лучше счетчиков?  
Значит, счетчики — бесполезны?*

Поначалу и я так думал. Раз они могут все, давайте использовать их везде. Логично?

**А вот и нет.**

Измерители идеально подходят для измерения текущего значения метрики, которое со временем может уменьшиться.

Вот тут-то и кроются те самые подводные камни: измеритель не показывает развитие метрики за период времени. Используя измерители, **можно упустить нерегулярные изменения метрики со временем**.

«Измеритель показывает среднее значение дельты счетчика для единицы за период времени.  
  
Счетчик учитывает каждую использованную единицу (если это процессор, то операции, циклы или такты), а потом вы можете выбрать, показатели за какой период вам нужны.  
  
Если вы используете измеритель, частота выборки должна быть точной. Если частота отличается хотя бы на несколько микросекунд, значение будет недостоверным. Это еще более заметно при большой нагрузке, где время между измерениями возрастает в геометрической прогрессии, потому что планировщик системы не успевает уделять внимание приложению мониторинга».

Если система отправляет метрики каждые 5 секунд, а Prometheus скрейпит целевой объект каждые 15, в процессе можно потерять некоторые метрики. Если выполнять дополнительные вычисления с этими метриками, точность результатов окажется еще ниже.

У счетчика каждое значение агрегировано. Когда Prometheus собирает его, он понимает, что значение было отправлено в определенный интервал.

**Гистограмма**

Гистограмма — это более сложный тип метрики. Она предоставляет дополнительную информацию. Например, сумму измерений и их количество.

Значения собираются в области с настраиваемой верхней границей. Поэтому гистограмма может:

* **Рассчитывать средние значения**, то есть сумму значений, поделенную на количество значений.
* **Рассчитывать относительные измерения значений**, и это очень удобно, если нужно узнать, сколько значений в определенной области соответствуют заданным критериям. Особенно это полезно, если нужно отслеживать пропорции или установить индикаторы качества.

В реальном мире я бы хотел получать оповещение, если у 20% моих серверов отклик больше 300 мс или отклик серверов больше 300 мс более 20% времени.

Если вы имеете дело с пропорциями, **вам нужны гистограммы**.

**Сводки**

Сводки — **это расширенные гистограммы**. Они тоже показывают сумму и количество измерений, а еще **квантили за скользящий период**.

Квантили, если что, — это деление плотности вероятности на отрезки равной вероятности.

*Итак: гистограммы или сводки?*

Все зависит от **намерения**.

Гистограммы объединяют значения за период времени, предоставляя сумму и количество, по которым можно отследить развитие определенной метрики.

Сводки, с другой стороны, показывают квантили за скользящий период (т. е. непрерывное развитие во времени).

Это особенно удобно, если вам нужно узнать значение, которое представляет 95% значений, записанных за период.

**Задания и экземпляры**

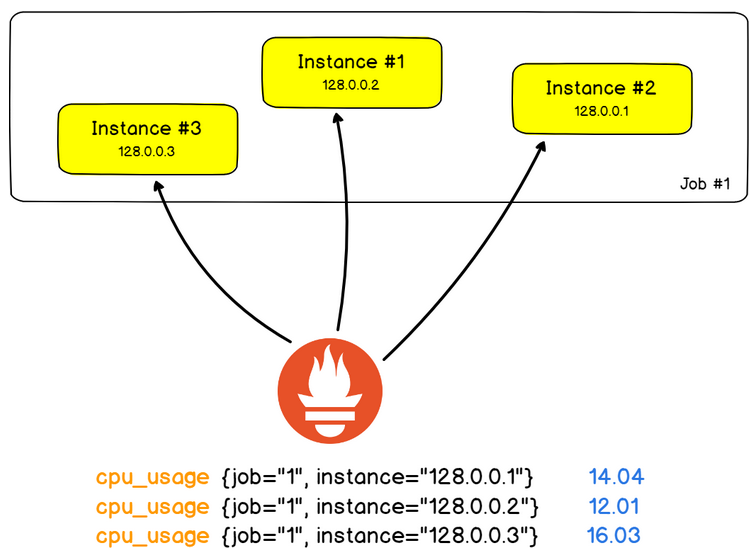
Учитывая последние успехи в распределенных архитектурах и популярность облачных решений, вряд ли вы используете одинокий сервер, работающий сам по себе.

**Серверы реплицируются и распределяются по всему миру.**

Чтобы это проиллюстрировать, давайте рассмотрим классическую архитектуру из двух серверов HAProxy, которые перераспределяют нагрузку по девяти бэкенд-веб-серверам

В этом примере из реальной жизни мы отследим **число ошибок HTTP, возвращенных веб-серверами**

На языке Prometheus один веб-сервер называется **экземпляром**. **Заданием** будет тот факт, что вы измеряете число ошибок HTTP на всех экземплярах.



Прелесть в том, что задания и экземпляры — это поля в ярлыках, и вы можете фильтровать результаты по определенному экземпляру или заданию

**PromQL**

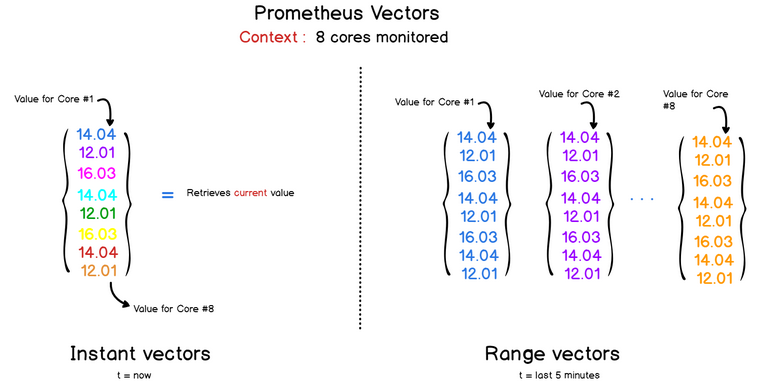
У Prometheus есть свой язык для запросов и извлечения данных с серверов: **PromQL**.

Как мы уже знаем, данные представлены в виде пар «ключ-значение». PromQL использует тот же синтаксис и возвращает результаты в виде векторов.

Что за векторы?

В Prometheus и PromQL есть два вида векторов:

* **Моментальные векторы**, которые представляют все метрики по последней метке времени.
* **Векторы с диапазоном времени**: если вам нужно посмотреть развитие метрики со временем, вы можете указать диапазон времени в запросе к Prometheus. В итоге получите вектор, объединяющий все значения, записанные за выбранный период.



PromQL API предоставляет набор функций для операций с данными в запросах.

Вы можете сортировать значения, применять к ним математические функции (например, рассчитывать производные или экспоненты) и даже строить прогнозы (например, по модели Хольта-Уинтерса).

**Инструментирование**

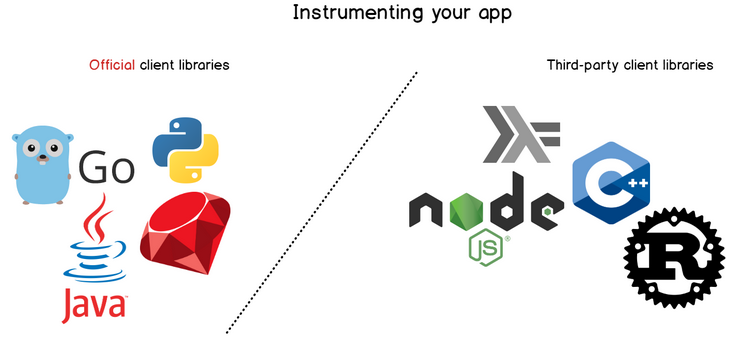
**Инструментирование** — это еще одна важная часть Prometheus. Вы инструментируете приложения, прежде чем извлекать из них данные.

На языке Prometheus инструментирование означает добавление клиентских библиотек в приложение, чтобы они предоставляли метрики Prometheus.

Инструментирование доступно для большинства распространенных языков программирования: например, **Python, Java, Ruby, Go и даже Node или C#**.

По сути, вы создаете объекты памяти (например, измерители или счетчики), которые будут динамически увеличивать или уменьшать значение.

Потом вы выбираете, где предоставлять метрики. Prometheus заберет их оттуда и сохранит в свою базу данных временных рядов.



**Экспортеры**

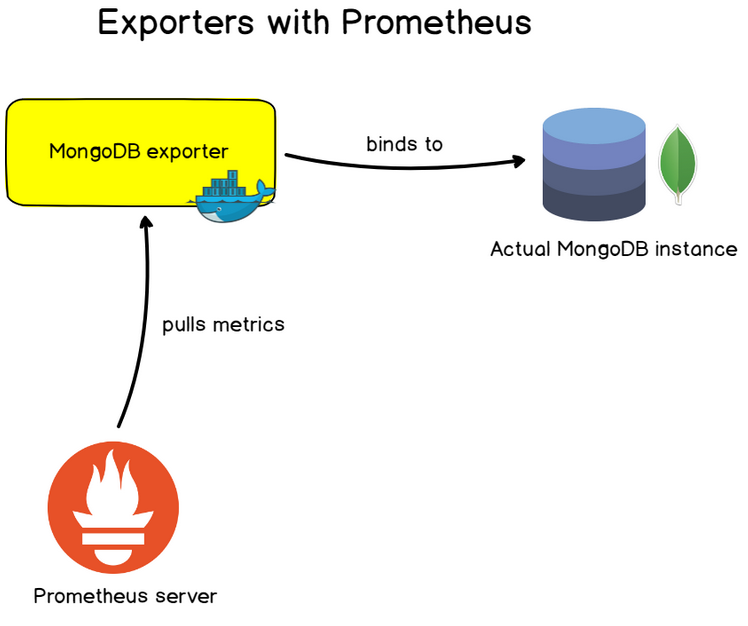
В написанных вами приложениях очень удобно настраивать предоставляемые метрики и их изменение со временем с помощью инструментирования

Для известных приложений, серверов и баз данных **Prometheus предлагает экспортеры, с помощью которых можно мониторить целевые объекты**.

Эти экспортеры обычно представлены в виде образов Docker и легко настраиваются. Они предоставляют готовый набор метрик и часто готовые панели мониторинга, с которыми можно настроить мониторинг за считанные минуты.

Примеры экспортеров:

* Экспортеры баз данных: для баз данных MongoDB, серверов SQL и MySQL.
* Экспортеры HTTP: для серверов HAProxy, Apache или NGINX.
* Экспортеры Unix: производительность системы можно отслеживать с помощью встроенных экспортеров узлов, которые предоставляют все системные метрики без дополнительной настройки.



**Пара слов о взаимной совместимости**

Большинство баз данных временных рядов поддерживают взаимную совместимость для своих систем.

Prometheus не единственная система мониторинга со своими требованиями к предоставлению метрик. Например, у [InfluxDB](https://www.influxdata.com/) (через Telegraf), [CollectD](https://collectd.org/), [StatsD](https://github.com/statsd/statsd) и [Nagios](https://www.nagios.org/) тоже есть свои стандарты.

Поэтому для взаимодействия разных систем создаются экспортеры. Даже если Telegraf отправляет метрики не в том формате, который принимает Prometheus, Telegraf может послать эти метрики в экспортер InfluxDB, откуда их потом заберет Prometheus.

**Оповещения**

При работе с базами данных временных рядов вам нужна обратная связь от данных, и за это отвечают менеджеры оповещений.

В Grafana оповещения обычное дело, но они доступны и в Prometheus через менеджер оповещений.

**Менеджер оповещений — это отдельный инструмент, который присоединяется к Prometheus и запускает кастомные оповещатели**.

Оповещения определяются в **файле конфигурации** и задают набор правил для метрик. Если во временных рядах возникает соответствие правилу, оповещение инициируется и отправляется заданным получателям.

Как и в Grafana, в качестве получателя можно указать электронный адрес, **вебхук Slack, PagerDuty и кастомные HTTP-объекты**.

