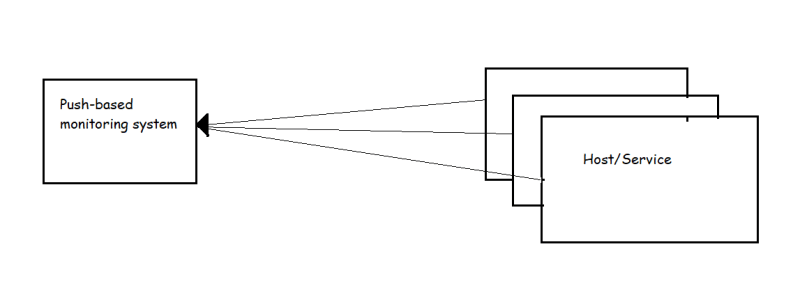
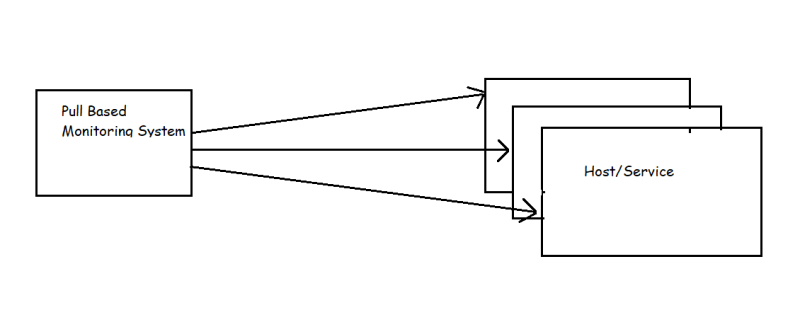
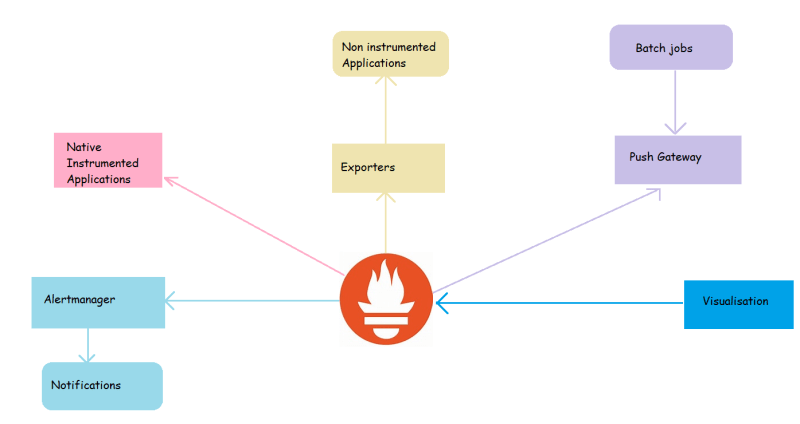
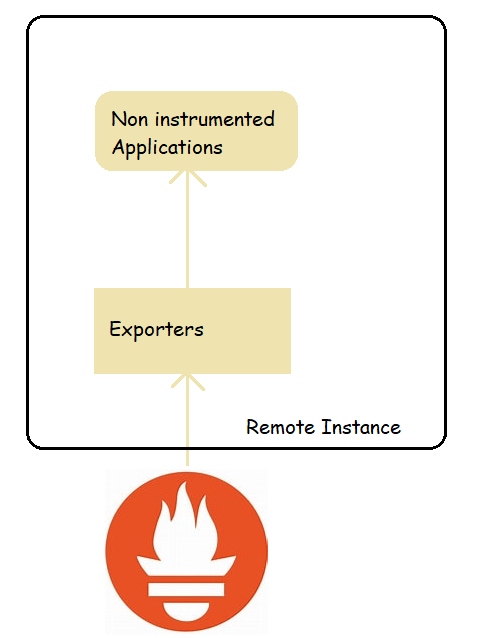
**An Overview of Metric Collection approaches (Push and Pull)**

* In Push Based monitoring systems, emitted metrics or events are sent directly from producing application or from a logical agent to the collecting server 
* Some examples of this approach is used in Elastic Search, LogStash and Kibana (Elastic Stack)
* In contrast, pull based monitoring systems collect metrics directly from applications or from proxy processes that makes those metrics available 
* Prometheus is a pull-based monitoring system & it also provides a way of ingesting pushed metrics by using a gateway that converts from push to pull.

**Prometheus**

* Prometheus is time series based open source monitoring system.
* It collects data by sending HTTP requests to hosts and services on metric endpoints, which it makes available for analysis and alerting using a powerful query language
* <https://prometheus.io/docs/introduction/overview/> for official docs
* Prometheus has joined Cloud Native Computing Foundation (CNCF) in year 2016.
* The Prometheus ecosystem is composed of several components
* High level overview of main components of Prometheus eco system 
* As we can see in the image above,
  + Prometheus Server collects time series data, stores it and makes it available for querying and send alerts based on it
  + The Alert Manager recieves alert triggers from Prometheus and handling routing and dispatching of events
  + The Push gatewayy handles the metrics that have been pushed from short-lived jobs (cron jobs or batch jobs)
  + Applications that support the Prometheus exposition format make internal state available through and HTTP endpoint
  + Community driven exporters expose metrics from applications that do not support prometheus natively
  + First-Party and Third-party dashboarding provide a visualization of collected
* Prometheus when it was originally created at SoundCloud was influenced from Google’s Borgmon.
  + Scraping plain text from metrics endpoints
  + exporters as proxies for metrics collections
  + time series as multi dimensional vectors
  + use of ruleset evaluations

**Exposing Internal State with exporters**

* Not all applications are built with Prometheus compatible instrumentations, Sometimes no metrics are exposed at all, In these case we can rely of exporter. 
* Exporter is nothing more that a piece of software that collects data from service or application and exposes via HTTP in the Prometheus
* Node Exporter is one of most commonly used exporters, which presents number of kernel statistics such as disk I/O, CPU, Memory, network, filesystem usage and much more.
* We have exporters for pretty much everything <https://prometheus.io/docs/instrumenting/exporters/>
* Terminology:
  + Scrape: The HTTP GET request made by the Prometheus server to the observed system for metric collection is called as scrape.
* Guidelines:
  + If you are one writing the service, the best option is to instrument the code directly using a Prometheus client library.
  + There are official libraries for
    - Go
    - Java
    - Python
    - Ruby
  + There are community driven client libraries for almost all the programming languages <https://prometheus.io/docs/instrumenting/clientlibs/>
  + If you want to develop exporters <https://prometheus.io/docs/instrumenting/writing_exporters/>
* Alerting routes: There are multiple out of the box integrations available for most common use case such as
  + email
  + HipChat
  + Slack
  + OpsGenie
  + PagerDuty

**Visualization Data collected**

* Prometheus exposes a well defined AP where PromQL queries can provide raw data for visualizations
* As of now the best external software for visualization used along with Prometheus is Grafana.
* The Prometheus server also ships with two internal visualization components
  + Experssion browser: Here we can run PromQL directly to quickly query and visualize data instantly
  + Consoles: These are web pages that ar built using the Golang templating language and served by Prometheus server itself.

**Local Environment**

* Windows 10:
  + Download Docker
  + Install kubectl
  + Install Helm