**1.First Steps with Prometheus:**

Prometheus collects metrics from monitored targets by scraping metrics HTTP endpoints on these targets. We can install, configure and monitor our first resource with Prometheus. We can also download and install an exporter, tools that expose time series data on hosts and services. First exporter will be Prometheus itself, which provides a wide variety of host-level metrics about memory usage, garbage collection.

The Prometheus download comes with a sample configuration in a file called Prometheus.yml. We can define and control all the Prometheus configuration related to alerting and scraping metric data from different instances.

For Linux: Node\_exporter: Install the node exporter on the machine which you want to monitor and add the details of the node\_exporter service in Prometheus.yml file on the Prometheus server. Node exporter listens on 9100 by default.

For Windows: We can use WMI Exporter

For Network Devices: SNMP Exporter

For Batch Jobs: Push Gateway

For JVM Applications: JMX Exporter

\*NB: An endpoint you can scrape is called an instance. A collection of instances with the same purpose, a process replicated for scalability or reliability for example, is called a job.

Recording/Defining a New Rule:

Create a file with the recording rule and save it as e.g.  prometheus.rules.yml.

Example: To record the time series resulting from this expression into a new metric called job\_service:rpc\_durations\_seconds\_count:avg\_rate5m

Groups:

- name: example

rules:

- record: job\_service:rpc\_durations\_seconds\_count:avg\_rate5m

expr: avg(rate(rpc\_durations\_second\_count[5m])) by (job,service)

To make Prometheus pick up this new rule, add a rule\_files statement in your Prometheus.yml. Restart Prometheus with the new configuration and verify that a new time series with the metric name job\_service:rpc\_durations\_seconds\_count:avg\_rate5m is now available by querying it through the expression browser or graphing it.

**2. CONFIGURATION**:

Prometheus is configured via command-line flags and a configuration file. While the command-line flags configure immutable system parameters (such as storage locations, amount of data to keep on disk and in memory, etc.), the configuration file defines everything related to scraping jobs and their instances, as well as which rule files to load. Prometheus can reload its configuration at runtime. A configuration reload is triggered by sending a SIGHUP to the Prometheus process or sending a HTTP POST request to the /-/reload endpoint (when the --web.enable-lifecycle flag is enabled). This will also reload any configured rule files.

**Configuring rules:**

**Recording Rule**: Recording rules allow you to precompute frequently needed or computationally expensive expressions and save their result as a new set of time series

**Alerting Rule**:

To include rules in Prometheus, create a file containing the necessary rule statements and have Prometheus load the file via the rule\_files field in the Prometheus configuration(Prometheus.yml).

To check whether a rule file is syntactically correct without starting a Prometheus server, install and run Prometheus's promtool command-line utility tool and use the below command to check the syntax of the rule file:

Promtool check rules <rulefilepath>

E.g. of rule file:

groups:  
 - name: alert.rules  
rules:  
 - alert: InstanceDown  
 expr: up == 0  
 for: 1m  
labels:  
 severity: "critical"  
annotations:  
summary: "Endpoint {{ $labels.instance }} down"  
description: "{{ $labels.instance }} of job {{ $labels.job }}

\*NB: this rule will create an alert if the instance is down of more than 1 minute. PFB, some more examples of alert rules that can be configured using PromQL and Prometheus alerting rules.



**3.Alert Manager**:

**The Alert Manager is part of the Prometheus stack**, but it is run as a standalone server aside from Prometheus. The basic components of alertmanager are: -

**a**mtool: the amtool is an executable that allows you to view or to modify the current state of the Alert Manager. In other words, the amtool can silence alerts, expire silences, as well as import silences or query them. It can be seen as a utility to customize the AlertManager without directly modifying the configuration of your current alerts.

alertmanager: the executable for the alertmanager. This is the executable that you will run in order to start an AlertManager server on your instance.

alertmanager.yml: This is the configuration file for the AlertManager.

The main options: -

config.file : we need to set this variable to the correct configuration file of the “etc/alertmanager” folder.

storage.path : again, we defined a custom folder for data which is “/data/alertmanager”

web.external-url: if you followed the Prometheus setup entirely, your Prometheus instance is running behind a reverse proxy. In this case, we are going to set the URL for the AlertManager to be externally reachable.

Alertmanager can reload its configuration at runtime. If the new configuration is not well-formed, the changes will not be applied and an error is logged. A configuration reload is triggered by sending a SIGHUP to the process or sending a HTTP POST request to the /-/reload endpoint

We can define grouping of alerts, http configurations, inhibit rules, TLS configuration, routes, child routes, receivers, etc. using the configuration file for alert manager.

A http\_configs allows configuring the HTTP client that the receiver uses to communicate with HTTP-based API services.

We can user Pager Duty, Slack, Hip Chat, Push Over, Opsgenie, Webhooks, Victorops, Wechat, to integrate custom notification.

The main steps to setting up alerting and notifications are:

Setup and configure the Alertmanager

[Configure Prometheus](https://prometheus.io/docs/prometheus/latest/configuration/configuration/#alertmanager_config) to talk to the Alertmanager

Create [alerting rules](https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/) in Prometheus

E.g. of a alertmanager.yml file which fires an alert to slack webhook url defined below, where the alert name matches the name “InstanceDown”

global:  
 slack\_api\_url: <https://hooks.slack.com/services/SlackAPI>

route:  
 group\_by: ['instance', 'severity']  
 group\_wait: 30s  
 group\_interval: 5m  
 repeat\_interval: 3h  
 routes:  
 - match:  
 alertname: InstanceDown  
 receiver: 'alert-team'receivers:  
- name: 'alert-team'  
 slack\_configs:  
 - channel: "#webhook-channel"  
 text: "summary: {{ .CommonAnnotations.summary }}\ndescription: {{ .CommonAnnotations.description }}"

**4. SENDING ALERTS**:

The Alertmanager has two APIs, v1 and v2, both listening for alerts. Clients are expected to continuously re-send alerts as long as they are still active (usually on the order of 30 seconds to 3 minutes). Clients can push a list of alerts to Alertmanager via a POST request. The labels of each alert are used to identify identical instances of an alert and to perform deduplication. The annotations are always set to those received most recently and are not identifying an alert.

The details of the v1 and v2 APIs can be found at <https://prometheus.io/docs/alerting/clients/>

Alert Manager also supports templating, though it is different from templating in Prometheus. Refer <https://prometheus.io/docs/alerting/notifications/>

Alert Manager Management API –

Health Check: GET /-/healthy

Readiness check: GET /-/ready

Reload: POST /-/reload

**5.STEPS TO CONFIGURE PROMETHEUS:**

1. Download and install Prometheus on any machine. This machine will act as Prometheus Server.
2. Download and install node\_exporter on the same machine(Prometheus Server) for monitoring the server itself.
3. Download and install node\_exporter on each machine you want to monitor and add it as a target in the Prometheus.yml file present on the Prometheus server.



4. Download and install Alertmanager on your machine and bind alertmanager in Prometheus.

E.g.

alerting

-alertmanagers:

-staticonfigs:

-targets:

-localhost:9300

\*NB: AlertManager runs on port 9300 by default.

5. Define the alerting rules, grouping, inhibit\_rules, http\_configs, tls\_config, receiver, email\_config, any one of the notification integrations in the Alert Manager’s configuration file.

6. We have configured the SMTP email config. to get the alerts on email.

\*Required email config. details:

- smtp\_smarthost: FQDN or IP of the smtp server.

-smtp\_port: port on which the smtp server is running. (default is 25)

-Username: <username to login to smtp server>

-password: <password for smtp server>

-smtp\_require\_tls: <false>

7. Add the IMAP server on IIMSS, create a new rule, and then create policy using the rule to create alerts on IIMSS.