DevOps – Lecture 09

Monitoring

1 Nov 2022

Chinmaya Dehury

Chinmaya.Dehury@ut.ee
Where are we now?

1. SDLC + Why DevOps
2. DevOps Phases + Cloud Computing – Basics
3. VCS – GitLab
4. Containerization – Kubernetes
5. Microservices
6. Automation – Ansible
7. CI/CD – GitLab
8. Continuous Testing

9. Monitoring

10. Application Deployment Modeling
11. DataOps
12. DevOps + Serverless
13. Challenges + Future Scope
OUTLINE

• Monitoring in DevOps
• Monitoring Goals and Importance
• What to monitor?
  • Infrastructure, Application, Network, User behavior
• Synthetic monitoring
• Monitoring tool features
• Push vs Pull mechanism
• Prometheus monitoring tool
  • features, installation
  • Architecture
  • Exporter
  • Data Model, Metric type, PromQL
• Service Discovery
• Alerting rules and Alertmanager
Monitor

**monitor**

*verb [T]*

UK  /ˈmɒnɪtər/  US  /ˈmaɪ.nər/  

to watch and check a situation carefully for a period of time in order to discover something about it:

- The new findings suggest that women ought to monitor their cholesterol levels.
- The CIA were monitoring (= secretly listening to) his phone calls.

Src: [https://dictionary.cambridge.org/dictionary/english/monitoring](https://dictionary.cambridge.org/dictionary/english/monitoring)
Estonian Rail

Estonian Air space

Src: https://elron.ee/en
Src: https://www.flightradar24.com/58.37,26.75/6
Monitoring in Computer

• A software tool, that
  • observes
  • tracks the operations and activities of users, applications and network services

Monitoring Software for Airport

Monitoring Software for city road

Img: https://emerging-europe.com/business/the-estonian-start-up-taking-cctv-beyond-surveillance/
Img: https://www.infortrend.com/cz/solutions/surveillance
Monitoring in Computer

• A software tool, that
  • observes
  • tracks the operations and activities of users, applications and network services
Inbuilt Monitoring tools in linux-based system

- **top, atop, htop, ktop**

Other tools
- Vmstat
- Lsof
- Tcpdump
- Iotop
- Iostat
- ...<Google for more>
Monitoring in DevOps

• The last phase of DevOps Loop
• Monitoring provides feedback from production.
• Monitoring delivers information about an application's usage patterns.
• Is monitoring all about?
  • Monitoring the *live* application
• How about, monitoring
  • Planning, Development, Integration and Testing, Deployment,
• Continuous Monitoring
  • a.k.a. ConMon
Monitoring Goals

• Achieve high availability
  • Reduce *time to detect* (TTD)
  • Reduce *time to mitigate* (TTM)
  • Reduce *time to remediate* (TTR)

• Real-time feedback
  • on the overall health of IT infrastructure
  • Application performance issues

• Visibility and transparency of network activity

• Monitor user behavior and feedback
  • Experience with new updates
    • E.g. experience with Windows 11
Importance of DevOps Monitoring

• Visibility
  • E.g. Team members need to see all events when a failure occur

• Automated Collaboration
  • All team members should get the notification to events (merge request, commits, merge, etc.) automatically
  • E.g. Gitlab Slack or Webex notification
    • Push
    • Issues
    • Merge Requests
    • Comments
    • Pipeline
    • Deployment
    • Code quality, Test report, Code coverage, etc...

• Mitigate the risk of cyber attacks
  • With real-time alert system
Importance of DevOps Monitoring contd...

• Change management
  • Quickly see the changes that cause application outage

• Monitoring dependencies
  • You are using *latest* version of a external tools
  • There is an update in the third party tools and
  • Now you need to monitor if that third party tool itself is not vulnerable

• Monitoring external tools/services
  • E.g. AWS S3 buckets, Lambda service, Azure function service, external DBs
What to monitor?

• Infrastructure Monitoring
• Application Monitoring
• Network Monitoring
• User behavior
What to monitor - Infrastructure

- Infrastructure Monitoring
  - Containers, VMs, Physical machines/servers
  - Resource utilization
  - Storage servers
  - Cluster performance

- Application Monitoring
- Network Monitoring
- User behavior
What to monitor - Application

• Infrastructure Monitoring

• Application Monitoring
  • Uptime
  • Transaction time and volume
  • System responses, throughputs, error rate
  • API responses
  • Front-end and back-end
  • Page load time
  • # of HTTP requests, request rate, HTTP request execution time, etc.

• Network Monitoring

• User behavior
What to monitor - Network

• Infrastructure Monitoring

• Application Monitoring

• Network Monitoring
  • Network traffic, bandwidth, packet flows
  • Firewalls, Gateways, VPNs, security groups
  • Network interfaces, IPs
  • Endpoints,
  • Load balancers, latency
  • SSH, pings, connections, etc...

• User behavior
What to monitor – User behavior

• Infrastructure Monitoring
• Application Monitoring
• Network Monitoring

• User behavior
  • * Usually tradition monitoring are not that useful for this category
  • How users are using your application?
  • overall customer experience
  • Usage of application
  • Application specific data
    • Youtube, CuriosityStream, vimeo, etc.
    • Financial applications
    • Education application
    • Ecommerce application
    • Social networking applications
Synthetic monitoring

• aka Proactive monitoring
• Monitor in simulated environment
• Monitor Application, API, web services and all related components
• Simulate the monitoring environment with
  • Synthetic users
  • Behaviors
  • Transactions
• This is *not* monitoring the production environment
• You can not measure the experience using synthetic monitoring
# Synthetic monitoring vs Real Monitoring

## When to use Synthetic vs Real Application Monitoring:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Synthetic Monitoring</th>
<th>Real Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test a new feature before deploying</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Ensure application is up and running 24X7 (detect outages)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Test a new market/geography</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor web third party/ APIs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Determine the impact of a slowdown or failure on all users</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Resolve support issues</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Determine root cause of issues in application</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Monitor all traffic/real users</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SLA’s</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Transaction performance</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>API performance</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

![Image](https://static1.smartbear.co/smartbear/media/images/products/alertsite/synth-vs-rum.png)
# Synthetic monitoring vs Real Monitoring

<table>
<thead>
<tr>
<th></th>
<th>Synthetic monitoring</th>
<th>Real Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>Consistent set of transactions</td>
<td>Wide variety of transactions</td>
</tr>
<tr>
<td>Performance</td>
<td>Predictable after sometime</td>
<td>Not always Predictable</td>
</tr>
<tr>
<td>Performance Measurements</td>
<td>Repeated after sometime</td>
<td>Not repeated</td>
</tr>
<tr>
<td>User’s profile</td>
<td>Manually created</td>
<td></td>
</tr>
<tr>
<td>User device</td>
<td>Manually created</td>
<td></td>
</tr>
</tbody>
</table>
Monitoring tool features

• Real-time streaming and alert
  • On critical components of application and services
  • On hardware, application, operation
  • Configurable alert

• Query health status

• Automatic reporting

• Automated log aggregation

• Data analytics

• Historical replay

• Visualizations

• Use of REST APIs
Push vs Pull mechanism

Metrics are periodically sent by each monitored system to a central collector

Central collector periodically requests metrics from each monitored system
Push and Pull Production

• Push production:
  • Manufacturer decides what to produce and sent to retailer

• Pull production
  • Retailer decides what to produce and get the products from manufacturer

• Walmart works in a (Pull production) pull mechanism, where as most of the others work in a (push production) push mechanism.

Push vs Pull mechanism

Features

- Discovery:
  - **Push**: No need to configure collector, agent just need to be ready to push the metrics
  - **Pull**: you need to configure and reload the collector periodically

- Scalability:
  - **Push**: Fully distributed
  - **Pull**: More centralized

- Security
  - Against remote attack
Push vs Pull mechanism

Features

• Latency
  • **Push**: what happen when 1000 systems push metrics to single collector?
  • **Pull**: single collector need to make 1000 requests to get the updated information

• Complexity
  • Complexity for Agent
    • What happen when there large number of agents?
  • Complexity of Collector
    • Collector needs to be updated when a new system is added

• Flexibility
  • **Push**: pre-determined metrics are pushed
  • **Pull**: can get any metric at any time from any monitored system
Some tools for monitoring

- Prometheus
- Solarwinds
- Tenable
- Sensu
- Nagios
- Sematext
- Datadog
- ELK
- New Relic
- SysDig Monitor
Some tools for monitoring

- **Prometheus**
  - Solarwinds
  - Tenable
  - Sensu
  - Nagios
  - Sematext
  - Datadog
  - ELK
  - New Relic
  - SysDig Monitor
Prometheus monitoring tool

- [https://prometheus.io/](https://prometheus.io/)
- Monitoring and alerting toolkit
- Originally built at [SoundCloud](https://soundcloud.com)
- Open source and community-driven
- Written in Go Language
- Gather fine grained data (metrics) at a regular time interval
- Follows pull mechanism
  - over HTTP
- Store the data in memory time-series format
Prometheus monitoring tool

- Uses PromQL, a **flexible query language**
- Targets are discovered via service discovery or static configuration

In layperson terms:

- metrics are numeric measurements
  - E.g.
    - number of active connections
    - number of active queries
    - # of requests
- time series means that changes are recorded over time
Prometheus features

⚠️ Dimensional data
Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.

🔍 Powerful queries
PromQL allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.

👉 Great visualization
Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.

襲 Efficient storage
Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and federation.

💡 Simple operation
Each server is independent for reliability, relying only on local storage. Written in Go, all binaries are statically linked and easy to deploy.

⚠️ Precise alerting
Alerts are defined based on Prometheus's flexible PromQL and maintain dimensional information. An alertmanager handles notifications and silencing.

<![CDATA[</>]]>
Many client libraries
Client libraries allow easy instrumentation of services. Over ten languages are supported already and custom libraries are easy to implement.

✈️ Many integrations
Existing exporters allow bridging of third-party data into Prometheus. Examples: system statistics, as well as Docker, HAProxy, StatsD, and JMX metrics.

Src: https://prometheus.io/
Prometheus Installation

• **The most simple way is to use Docker**
  - `docker run -p 9090:9090 prom/prometheus`

• From binaries

  **2.31.1 / 2021-11-05**  [Release notes](#)

<table>
<thead>
<tr>
<th>File name</th>
<th>OS</th>
<th>Arch</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>prometheus-2.31.1.darwin-amd64.tar.gz</td>
<td>darwin</td>
<td>amd64</td>
<td>69.72 MiB</td>
</tr>
<tr>
<td>prometheus-2.31.1.linux-amd64.tar.gz</td>
<td>linux</td>
<td>amd64</td>
<td>69.69 MiB</td>
</tr>
<tr>
<td>prometheus-2.31.1.windows-amd64.zip</td>
<td>windows</td>
<td>amd64</td>
<td>70.99 MiB</td>
</tr>
</tbody>
</table>

• [https://prometheus.io/download/](https://prometheus.io/download/)
• [https://prometheus.io/docs/prometheus/latest/installation/](https://prometheus.io/docs/prometheus/latest/installation/)
Prometheus Web Interface

• The most simple way is to use Docker
  • docker run -p 9090:9090 prom/prometheus
Prometheus Web Interface

Some http related metrics:
Prometheus – Architecture

Service discovery
- kubernetes
- file_sd

Prometheus server
- Retrieval
- TSDB
- HTTP server

Prometheus alerting
- pagerduty
- Email
- etc

Alertmanager

PromQL

Prometheus web UI

Grafana

API clients

Data visualization and export

Short-lived jobs
- push metrics at exit

Pushgateway
- pull metrics

Jobs/exporters

Prometheus targets

Node
- HDD/SSD

LTAT.06.015 : Lec 09 : Monitoring 34
Prometheus – How it works – An Example?

Central Prometheus Server

1. http GET query
2. SQL queries with dedicated user
3. SQL queries results
4. Plain text response
5. Store with additional metadata

Monitored System

MySQL Server Exporter (mysqld_exporter)
Prometheus Exporter:

- Just a piece of software
- Expose metrics with an HTTP API
  - Starts a web server that exposes a `/metrics` URL
- Do not save data
- Turn the local statistics into Prometheus metrics
- There’s over 160 exporters
  - Databases
  - Storage
  - HTTP
  - APIs
  - Logging, etc.
  - [https://prometheus.io/docs/instrumenting/exporters/](https://prometheus.io/docs/instrumenting/exporters/)
Prometheus – some other exporters

Prometheus Exporter:
- Monitoring Docker container metrics using **cAdvisor**
  - short for **container Advisor**
  - `docker run -d -p 8080:8080 gcr.io/cadvisor/cadvisor:latest`
  - [https://github.com/google/cadvisor/blob/master/docs/storage/prometheus.md](https://github.com/google/cadvisor/blob/master/docs/storage/prometheus.md)
- Monitoring Linux host metrics with the **Node Exporter**
  - `node_exporter`
  - hardware and OS related metrics
Prometheus Basics – Data Model

• Stores all data as *time series*
• timestamp : a millisecond-precisioned
• *Value* : float64 type value
• Every time series is uniquely identified by its *metric name* and optional key-value pairs called *labels*.
  
  • *Format*: `<metric name>{<label name>=<label value>, ...}
  
  • *E.g.*:
    • `container_cpu_usage_seconds_total` - Cumulative cpu time consumed in seconds.
    • `container_cpu_usage_seconds_total{cpu="cpu03"} 66.475582802`
    
    • `node_network_receive_bytes_total{device="eth0"} 341159`
Prometheus Basics – Metric type

• Counter:
  • A counter is a cumulative metric that represents a single monotonically increasing counter
  • value can only increase
  • Can be reset to zero on restart
  • E.g.:
    • number of requests
    • Number of containers running

• Gauge:
  • a metric that represents a single numerical value that can arbitrarily go up and down.
  • E.g.
    • Memory or CPU usage
    • Number of concurrent requests
Prometheus Basics – Metric type

• **Histogram:**
  • measures the frequency of value observations that fall into specific predefined buckets.
  • **Scenario:**
    • To take many measurements of a value, to later calculate averages or percentiles
  • **E.g.**:
    • Response size
    • Duration of a request

If there are four requests that took 1.2 sec, 3 sec, 8 sec, and 9 sec

Example:

```yaml
# HELP request_duration Time for HTTP request.
# TYPE request_duration histogram
request_duration_bucket{le="0.005",} 0.0
request_duration_bucket{le="0.05",} 0.0
request_duration_bucket{le="0.075",} 0.0
request_duration_bucket{le="0.1",} 0.0
request_duration_bucket{le="0.5",} 0.0
request_duration_bucket{le="1.0",} 0.0
request_duration_bucket{le="2.5",} 1.0
request_duration_bucket{le="5.0",} 2.0
request_duration_bucket{le="7.5",} 2.0
request_duration_bucket{le="10.0",} 4.0
request_duration_bucket{le="+Inf",} 4.0
request_duration_count 4.0
request_duration_sum 21.2
```
Prometheus Basics – Metric type

Summary:

• Similar to Histogram
• Histograms:
  • quantiles are calculated on the Prometheus server.
• Summaries:
  • quantiles over a sliding time window
  • quantiles are calculated on the client side.

Example:

```plaintext
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.3811e-05
  go_gc_duration_seconds{quantile="0.25"} 4.5901e-05
  go_gc_duration_seconds{quantile="0.5"} 7.2422e-05
  go_gc_duration_seconds{quantile="0.75"} 0.000109434
  go_gc_duration_seconds{quantile="1"} 0.001273562
  go_gc_duration_seconds_sum 0.036197814
  go_gc_duration_seconds_count 40
```

Src: [https://prometheus.io/docs/practices/histograms/](https://prometheus.io/docs/practices/histograms/)
Get the list of metrics

• Example:

```bash
# HELP gc_gc_duration_seconds A summary of the pause duration of garbage collection cycles:
# TYPE gc_gc_duration_seconds summary
  go_gc_duration_seconds{quantile="0.05"} 2.5702e-05
  go_gc_duration_seconds{quantile="0.25"} 1.6861e-05
  go_gc_duration_seconds{quantile="0.5"} 7.2422e-05
  go_gc_duration_seconds{quantile="0.75"} 0.000102252
  go_gc_duration_seconds{quantile="1"} 0.001273562
  go_gc_duration_seconds_sum 0.041599197
  go_gc_duration_seconds_count 473
# HELP gc_goroutines Number of goroutines that currently exist.
# TYPE gc_goroutines gauge
  gc_goroutines 0
# HELP gc_info Information about the Go environment.
# TYPE gc_info gauge
  gc_info{version="go1.16.7"} 1
# HELP gc_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE gc_memstats_alloc_bytes gauge
  gc_memstats_alloc_bytes 4.991904e+06
# HELP gc_memstats_alloc_bytes_SI Total number of bytes allocated, even if freed.
# TYPE gc_memstats_alloc_bytes_SI gauge
  gc_memstats_alloc_bytes_SI 9.33226304e+09
# HELP gc_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE gc_memstats_buck_hash_sys_bytes gauge
  gc_memstats_buck_hash_sys_bytes 1.552226e+06
# HELP gc_memstats_frees_total Total number of frees.
# TYPE gc_memstats_frees_total gauge
```
PromQL

• A query language for Prometheus monitoring system
• lets the user select and aggregate time series data in real time.
• Result of PromQL query execution
  • Shown as a graph
  • viewed as tabular data
  • consumed by external systems (e.g. Grafana) via the HTTP API.
PromQL - Examples

Filtering example

```
node_network_receive_bytes_total(device="eth0")
```

```
node_network_receive_bytes_total(device="eth0");
instance="172.17.91.13:9100", job="monitor-node-exporter"
```

Output:

```
351180
```
Filter: Get the list of container with running state

```
container_tasks_state{state="running"}
```
Get historical data

```
node_network_receive_bytes_total{device="eth0", instance="172.17.91.139:9100", job="monitor-node-exporter"}                   58476
node_network_receive_bytes_total{device="lo", instance="172.17.91.139:9100", job="monitor-node-exporter"}                  0
```
PromQL - Examples

Count the number of container last seen by the exporter

```
count(container_last_seen{image=""})
```

Load time: 30ms  Resolution: 1s  Result series: 1

Table  | Graph
---|---
Evaluation time

0 | 4
Prometheus – Service Discovery

• a standard method of finding endpoints to scrape for metrics.

• Configuration file /etc/prometheus/prometheus.yml

• What would you do when a new VM added or an existing VM deleted?
Prometheus – Service Discovery

• a standard method of finding endpoints to scrape for metrics.

• Configuration file
/etc/prometheus/prometheus.yml

An Example

```yaml
# HELP gc_gc_duration_seconds A summary of the pause
duration of garbage collection cycles.
# TYPE gc_gc_duration_seconds summary
gc_gc_duration_seconds{quantile="0.01"} 2.7793e-05
gc_gc_duration_seconds{quantile="0.25"} 5.6971e-05
gc_gc_duration_seconds{quantile="0.5"} 7.6402e-05
gc_gc_duration_seconds{quantile="0.75"} 9.000104782
gc_gc_duration_seconds{quantile="1.0"} 1.0015323973
# HELP gc_gc_duration_seconds_sum Total number of garbage collection
# HELP gc_gc_duration_seconds_count Total number of garbage collection
# TYPE gc_gc_duration_seconds gauge
gc_gc_duration_seconds 0.00000000
```

localhost:9090/metrics
Prometheus – Service Discovery

- a standard method of finding endpoints to scrape for metrics.
- Configuration file /etc/prometheus/prometheus.yml

An Example

targets.json

Another way to provide targets information.
Prometheus – Service Discovery

File-based service discovery (**doc**) 

• provides a more generic way to configure static targets

1. Install Node exporter  
   • IP: 120.125.123.12  
   • running on 9100 port

2. Open `prometheus.yml` configuration file

3. Add following job

   ```yaml
   - job_name: node-exporter
     file_sd_configs:
     - files:
       - 'targets.json'
   ```

4. Create `targets.json` in the same dir. and add following

   ```json
   [
   {
   "labels": {
   "job": "node-exporter"
   },
   "targets": [
   "120.125.123.12:9100"
   ]
   }
   ```
File-based service discovery (doc)

• provides a more generic way to configure static targets
• When you save the changes, Prometheus will automatically be notified of the new list of targets.
  • Inotify makes this approach event-based
• Does not require Prometheus instance to restart.
Some more service discovery options


\texttt{<http\_sd\_config>}

- discovery of targets over the HTTP protocol
- provides a more generic way to configure static targets and serves as an interface to plug in custom service discovery mechanisms.
- fetches targets from an HTTP endpoint containing a list of zero or more \texttt{<static\_config>}s.

\texttt{Prometheus.yml}

```yaml
scrape_configs:
  - job_name: http-sd
    http_sd_configs:
      - url: http://httpsd.local/prometheus-http-sd-targets
```
## Prometheus – Service Discovery

Some more service discovery options

### `<http_sd_config>` vs `<file_sd_configs>`

<table>
<thead>
<tr>
<th>Item</th>
<th>File SD</th>
<th>HTTP SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Based</td>
<td>Yes, via inotify</td>
<td>No</td>
</tr>
<tr>
<td>Update frequency</td>
<td>Instant, thanks to inotify</td>
<td>Following refresh_interval</td>
</tr>
<tr>
<td>Format</td>
<td>Yaml or JSON</td>
<td>JSON</td>
</tr>
<tr>
<td>Transport</td>
<td>Local file</td>
<td>HTTP/HTTPS</td>
</tr>
<tr>
<td>Security</td>
<td>File-Based security</td>
<td>TLS, Basic auth, Authorization header, OAuth2</td>
</tr>
</tbody>
</table>

[https://prometheus.io/docs/prometheus/latest/http_sd/](https://prometheus.io/docs/prometheus/latest/http_sd/)
Prometheus – Service Discovery

Some more service discovery options

<azure_sd_config>
<consul_sd_config>
<digitalocean_sd_config>
<docker_sd_config>
<dockerswarm_sd_config>
<dns_sd_config>
<ec2_sd_config>
<openstack_sd_config>
<puppetdb_sd_config>
<file_sd_config>
<gce_sd_config>
<hetzner_sd_config>

<http_sd_config>
<kubernetes_sd_config>
<kuma_sd_config>
<lightsail_sd_config>
<linode_sd_config>
<marathon_sd_config>
<nerve_sd_config>
<serverset_sd_config>
<triton_sd_config>
<eureka_sd_config>
<scaleway_sd_config>
<uyuni_sd_config>
Alerting rules

- Define alert conditions based on Prometheus expression language
- send notifications about firing alerts to an external service (e.g. alertmanager)
- Prometheus can be configured to automatically discover available Alertmanager instances through its service discovery integrations.

Example-1

```yaml
rules:
  - alert: InstanceDown
    expr: absent(container_start_time_seconds{name="my_flask_app"})
    for: 10s
```

Example-2

```yaml
rules:
  - alert: HighRequestLatency
    expr: job:request_latency_seconds:mean5m{job="myjob"} > 0.5
    for: 10m
```
Alerting rules - Alertmanager

Alertmanager

• Some situations where we need Alertmanager:
  • Add summary
  • notification rate limiting
  • silencing
  • alert dependencies on top of the simple alert definitions
  • Takes care of duplications, routing

• Prometheus can be configured to automatically discover available Alertmanager instances through its service discovery integrations.

• Receiver integration can be:
  • Email
  • Slack, Wechat
  • PagerDuty, or OpsGenie
Use case for Lab session
Use case for Lab session
References

- https://dictionary.cambridge.org/dictionary/english/monitoring
- https://docs.microsoft.com/en-us/devops/operate/what-is-monitoring
- https://www.devopsonline.co.uk/the-importance-of-monitoring-in-devops/
- https://smartbear.com/learn/topic-area/synthetic-vs-rum/
- https://www.atlassian.com/devops/devops-tools/devops-monitoring
- https://resources.infosecinstitute.com/topic/top-10-tools-for-continuous-monitoring/
- https://prometheus.io/
- https://prometheus.io/docs/prometheus/latest/http_sd/
- https://prometheus.io/docs/alerting/latest/alertmanager/
- https://inuits.eu/blog/prometheus-http-service-discovery/
Any Question?

THANK YOU