# Monitoring Cheatsheet

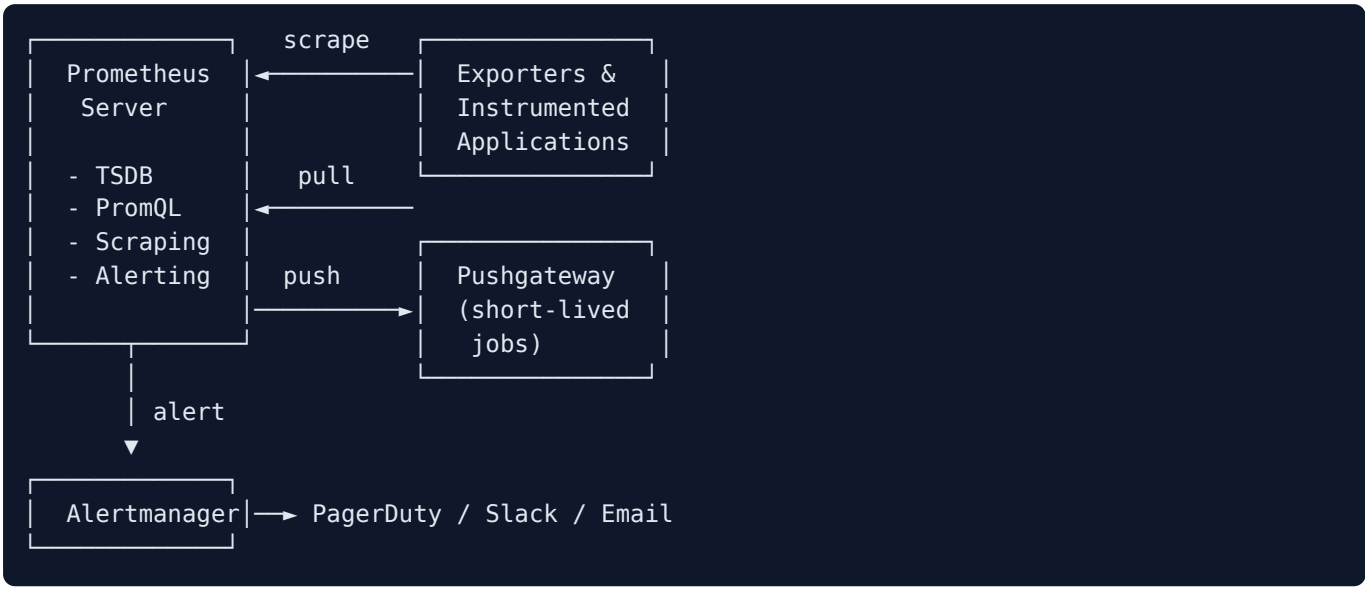
## Monitoring Cheatsheet

A practical quick-reference guide for Prometheus and Grafana observability. Covers configuration, PromQL, alerting rules, exporters, dashboard creation, and production-ready patterns.

## Prometheus Architecture

Prometheus is a pull-based monitoring system with a time-series database, a query language (PromQL), and built-in alerting.

### Core Components




### Key Concepts

| Concept | Description |
|---|---|
| **Instance** | A single machine or pod being scraped |
| **Job** | A group of instances performing the same function |
| **Metric** | A time-series identified by name + labels |
| **Labels** | Key-value pairs that dimension metrics |

| Concept | Description |
|---|---|
| **Scrape Interval** | How often Prometheus pulls metrics (default 15s) |
| **Retention** | How long data is kept (default 15d) |

## prometheus.yml Configuration

The main Prometheus configuration file defines scrape targets, alerting, and storage.

### Minimal Config

```
global:
  scrape_interval: 15s          # Default scrape interval
  evaluation_interval: 15s      # Evaluate rules every 15s
  scrape_timeout: 10s           # Timeout per scrape

scrape_configs:
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9090"]

  - job_name: "node"
    static_configs:
      - targets: ["localhost:9100"]
```

### Production Config

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  scrape_timeout: 10s
  external_labels:
    cluster: "production"
    region: "us-east-1"

# Alertmanager connection
alerting:
  alertmanagers:
    - static_configs:
        - targets: ["localhost:9093"]

# Load rule files
rule_files:
  - "alerts/*.yml"
  - "records/*.yml"

scrape_configs:
  # Prometheus self-monitoring
  - job_name: "prometheus"
    static_configs:
      - targets: ["localhost:9090"]

  # Node metrics via node_exporter
  - job_name: "node"
```

```
    static_configs:
      - targets:
          - "node1:9100"
          - "node2:9100"
          - "node3:9100"

  # Kubernetes API server
  - job_name: "kubernetes-apiservers"
    kubernetes_sd_configs:
      - role: endpoints
    scheme: https
    tls_config:
      ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
    relabel_configs:
      - source_labels:
          [__meta_kubernetes_namespace, __meta_kubernetes_service_name, __meta_kubernetes_end
        action: keep
        regex: default;kubernetes;https

  # Kubernetes pods with annotations-based scraping
  - job_name: "kubernetes-pods"
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
        action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
        target_label: __address__
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name
```

### Storage & Retention

```
# Command-line flags (not in YAML config)
# Set via startup command:
prometheus \
  --storage.tsdb.retention.time=30d \
  --storage.tsdb.retention.size=100GB \
  --storage.tsdb.wal-compression \
  --config.file=/etc/prometheus/prometheus.yml \
  --web.enable-lifecycle
```

### Remote Write & Read

```
# Send metrics to a remote endpoint (e.g., Thanos, Cortex, Mimir)
remote_write:
  - url: "https://thanos-receive.example.com/api/v1/receive"
    queue_config:
      max_samples_per_send: 10000
      max_shards: 10
    write_relabel_configs:
      - source_labels: [__name__]
        regex: "go_.*"
        action: drop

# Query from a remote endpoint
remote_read:
  - url: "https://thanos-query.example.com/api/v1/query"
    read_recent: true
```

## PromQL — Prometheus Query Language

PromQL is the query language for selecting and aggregating time-series data.

### Instant vs Range Vectors

```
# Instant vector — latest value for each series
up

# Range vector — data points over the last 5 minutes
up[5m]

# Offset — look back in time
up offset 1w

# Subquery — expression over a time range (useful with rate)
rate(http_requests_total[5m])[30m:]
```

### Rate & Counter Functions

```
# Per-second average rate of increase over 5 minutes
rate(http_requests_total[5m])

# Rate ignoring minor counter resets (preferred for counters)
rate(http_requests_total[5m])

# Increase over time window (total delta, not per-second)
increase(http_requests_total[1h])

# Rate with 95th percentile (use irate for volatile data)
irate(http_requests_total[1m])

# Count of time-series with non-zero values
count(http_requests_total)
```

```
# Bytes per second for network traffic
rate(node_network_receive_bytes_total{device="eth0"}[5m])
```

### Histogram & Summary

```
# Histogram bucket — count of observations ≤ threshold
http_request_duration_seconds_bucket{le="0.5"}

# 95th percentile from histogram buckets (over last 5m)
histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m]))

# 99th percentile
histogram_quantile(0.99, sum(rate(http_request_duration_seconds_bucket[5m])) by (le))

# Average request duration from histogram
rate(http_request_duration_seconds_sum[5m])
  /
rate(http_request_duration_seconds_count[5m])

# Summary — 50th, 90th, 95th, 99th percentiles (pre-calculated)
http_request_duration_seconds{quantile="0.95"}
```

### Aggregation Operators

```
# Sum across all instances
sum(rate(http_requests_total[5m]))

# Sum by service label
sum(rate(http_requests_total[5m])) by (service)

# Average CPU usage across nodes
avg(rate(node_cpu_seconds_total{mode="user"}[5m])) by (instance)

# Maximum memory usage per pod
max(container_memory_usage_bytes{namespace="production"}) by (pod)

# Count of up instances per job
count(up{job="node"}) by (job)

# Top 5 busiest endpoints
topk(5, sum(rate(http_requests_total[5m])) by (endpoint))

# Bottom 3 instances with least disk space
bottomk(3, node_filesystem_avail_bytes{mountpoint="/"})

# Standard deviation of response times
stddev(rate(http_request_duration_seconds_sum[5m])
  / rate(http_request_duration_seconds_count[5m])) by (endpoint)

# Count distinct values
count(count(up) by (instance))
```

### Binary Operators

```
# Comparison — return 1 when CPU > 80%
node_cpu_seconds_total{mode="user"} > 0.8

# Arithmetic — free memory in GB
(node_memory_MemAvailable_bytes / 1024 / 1024 / 1024)

# Logical AND — up AND high CPU
up == 1 and rate(node_cpu_seconds_total{mode="idle"}[5m]) < 0.1

# Unless — instances NOT in maintenance
up unless on(instance) node_maintenance_mode == 1

# Set operations
http_requests_total{method="GET"} or http_requests_total{method="POST"}
```

### String Functions & Label Manipulation

```
# Relabel on the fly — rename a label
label_replace(up, "host", "$1", "instance", "(.*):.*")

# Drop a label
label_drop(metric{foo="bar"}, "foo")

# Keep only specific labels
label_keep(metric, "job", "instance")
```

### Useful Functions

```
# Clamp value between min and max
clamp(rate(cpu_usage[5m]), 0, 1)

# Absolute value
abs(node_memory_swap_cached_bytes - node_memory_swap_free_bytes)

# Ceil and floor
ceil(rate(http_requests_total[5m]) / 60)
floor(3.7)  # → 3

# Timestamp of last sample
timestamp(up)

# Day of week (0=Sunday, 6=Saturday)
day_of_week()

# Hour of day (0-23)
hour()

# Sort by value (descending)
sort_desc(sum(rate(http_requests_total[5m])) by (service))

# Vector with constant value
vector(1)

# Predict value 1 hour from now based on last 4 hours
predict_linear(node_filesystem_avail_bytes[4h], 3600)
```

### Common Query Patterns

```
# CPU utilization percentage per instance
100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)

# Memory utilization percentage
(1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) * 100

# Disk usage percentage per mount
(1 - (node_filesystem_avail_bytes{fstype!~"tmpfs|overlay"}
  / node_filesystem_size_bytes{fstype!~"tmpfs|overlay"})) * 100

# HTTP error rate (5xx) as percentage
sum(rate(http_requests_total{status=~"5.."}[5m])) by (service)
  /
sum(rate(http_requests_total[5m])) by (service) * 100

# Request rate per second
sum(rate(http_requests_total[5m])) by (method, path)

# Pod restart count
sum(rate(kube_pod_container_status_restarts_total[1h])) by (namespace, pod)

# Certificate expiry in days
(time() - node_cert_expiry_timestamp_seconds) / 86400

# Predict disk full in 4 hours
predict_linear(node_filesystem_avail_bytes{mountpoint="/"}[1h], 4 * 3600) < 0
```

## Recording Rules

Recording rules pre-compute frequently used or expensive queries. They improve dashboard performance and enable subquery expressions in alerts.

### Recording Rules File

```
# records/kubernetes.yml
groups:
  - name: kubernetes.cpu
    interval: 30s
    rules:
      # Average CPU utilization per namespace
      - record: namespace:cpu_utilization:ratio
        expr: |
          sum(rate(container_cpu_usage_seconds_total{container!=""}[5m])) by (namespace)
          /
          sum(kube_node_status_allocatable{resource="cpu"}) by (namespace)

      # Per-pod CPU usage
      - record: pod:cpu_usage:rate5m
        expr: |
          sum(rate(container_cpu_usage_seconds_total{container!=""}[5m])) by (namespace, pod)

  - name: kubernetes.memory
    interval: 30s
```

```
    rules:
      # Memory utilization per namespace
      - record: namespace:memory_utilization:ratio
        expr: |
          sum(container_memory_working_set_bytes{container!=""}) by (namespace)
          /
          sum(kube_node_status_allocatable{resource="memory"}) by (namespace)

  - name: http.rules
    interval: 15s
    rules:
      # Request rate
      - record: job:http_requests:rate5m
        expr: sum(rate(http_requests_total[5m])) by (job, method, path)

      # Error rate
      - record: job:http_errors:rate5m
        expr: sum(rate(http_requests_total{status=~"5.."}[5m])) by (job, method, path)

      # 95th percentile latency
      - record: job:http_latency:p95
        expr: histogram_quantile(0.95, sum(rate(http_request_duration_seconds_bucket[5m])) by
```

## Alerting Rules

Alerting rules define conditions that trigger notifications through Alertmanager.

### Alert Rules File

```
# alerts/node.yml
groups:
  - name: node.alerts
    rules:
      # Instance is down
      - alert: InstanceDown
        expr: up == 0
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Instance {{ $labels.instance }} is down"
          description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for mor

      # High CPU usage
      - alert: HighCPUUsage
        expr: 100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "High CPU usage on {{ $labels.instance }}"
          description: "CPU usage is {{ $value }}% on {{ $labels.instance }} for over 10 minu

      # Disk space running low
      - alert: DiskSpaceLow
        expr: (1 - (node_filesystem_avail_bytes{fstype!~"tmpfs|overlay"} / node_filesystem_si
```

```
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "Disk space low on {{ $labels.instance }}"
          description: "Disk {{ $labels.device }} on {{ $labels.instance }} is {{ $value | pr

      # Memory pressure
      - alert: MemoryPressure
        expr: (1 - (node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes)) > 0.9
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "Memory pressure on {{ $labels.instance }}"
          description: "{{ $labels.instance }} has only {{ $value | printf \"%.1f\" }}% avail

      # Predicted disk full
      - alert: DiskWillFillIn4Hours
        expr: predict_linear(node_filesystem_avail_bytes{fstype!~"tmpfs|overlay"}[1h], 4 * 36
        for: 1h
        labels:
          severity: critical
        annotations:
          summary: "Disk will fill in 4 hours on {{ $labels.instance }}"
          description: "Based on current rate, {{ $labels.device }} on {{ $labels.instance }}

  - name: http.alerts
    rules:
      # High error rate
      - alert: HighErrorRate
        expr: |
          sum(rate(http_requests_total{status=~"5.."}[5m])) by (service)
          /
          sum(rate(http_requests_total[5m])) by (service) > 0.05
        for: 5m
        labels:
          severity: warning
        annotations:
          summary: "High error rate on {{ $labels.service }}"
          description: "{{ $labels.service }} has a {{ $value | printf \"%.2f\" }}% error rat

      # High latency
      - alert: HighLatencyP99
        expr: histogram_quantile(0.99, sum(rate(http_request_duration_seconds_bucket[5m])) by
        for: 10m
        labels:
          severity: warning
        annotations:
          summary: "High P99 latency on {{ $labels.service }}"
          description: "P99 latency is {{ $value }}s on {{ $labels.service }}."
```

### Alertmanager Configuration

```
# alertmanager.yml
global:
  resolve_timeout: 5m
  smtp_smarthost: "smtp.example.com:587"
```

```
  smtp_from: "alerts@example.com"
  smtp_auth_username: "alerts@example.com"
  smtp_auth_password: "password"

# Inhibit — suppress lower-severity alerts when a higher one fires
inhibit_rules:
  - source_match:
      severity: critical
    target_match_re:
      severity: warning|info
    equal: ["alertname", "instance"]

# Routing tree
route:
  receiver: "default"
  group_by: ["alertname", "cluster"]
  group_wait: 30s        # Wait before sending first notification
  group_interval: 5m     # Wait before sending next group
  repeat_interval: 4h    # Re-send notification if alert is still firing
  routes:
    - match:
        severity: critical
      receiver: "pagerduty-critical"
      repeat_interval: 1h
    - match:
        severity: warning
      receiver: "slack-warnings"
      repeat_interval: 4h
    - match:
        alertname: DiskSpaceLow
      receiver: "disk-alerts"
      group_by: ["alertname", "instance"]

receivers:
  - name: "default"
    email_configs:
      - to: "ops@example.com"
        send_resolved: true

  - name: "pagerduty-critical"
    pagerduty_configs:
      - service_key: "YOUR_PAGERDUTY_SERVICE_KEY"
        severity: "{{ .GroupLabels.severity }}"

  - name: "slack-warnings"
    slack_configs:
      - api_url: "https://hooks.slack.com/services/T00/B00/xxx"
        channel: "#monitoring-warnings"
        title: "{{ .GroupLabels.alertname }}"
        text: >-
          {{ range .Alerts }}
          *Alert:* {{ .Annotations.summary }}
          *Instance:* {{ .Labels.instance }}
          *Value:* {{ .Value }}
          {{ end }}
        send_resolved: true

  - name: "disk-alerts"
    email_configs:
```

```
      - to: "storage-team@example.com"
        send_resolved: true
```

## Common Exporters

Exporters expose metrics from third-party systems in Prometheus format.

### node_exporter — Host Metrics

```
# Docker run
# docker run -d --net=host --pid=host --name node_exporter \
#   quay.io/prometheus/node-exporter:latest \
#   --path.rootfs=/host

# Key metrics exposed on :9100

# CPU utilization
100 - (avg by(instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)

# Memory available percentage
node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes * 100

# Disk I/O read/write bytes per second
rate(node_disk_read_bytes_total[5m])
rate(node_disk_written_bytes_total[5m])

# Network receive/transmit bytes per second
rate(node_network_receive_bytes_total{device="eth0"}[5m])
rate(node_network_transmit_bytes_total{device="eth0"}[5m])

# Filesystem usage
(1 - node_filesystem_avail_bytes{fstype!~"tmpfs|overlay"} / node_filesystem_size_bytes{fstype

# System load average
node_load1
node_load5
node_load15
```

### blackbox_exporter — Probing & Uptime

```
# blackbox.yml — probe configuration
modules:
  http_2xx:
    prober: http
    timeout: 5s
    http:
      preferred_ip_protocol: ip4
      valid_status_codes: [200, 301, 302]
      tls_config:
        insecure_skip_verify: false

  http_post_2xx:
    prober: http
    http:
```

```
      method: POST
      body: '{"health": "check"}'
      headers:
        Content-Type: application/json

  tcp_connect:
    prober: tcp
    timeout: 5s

  icmp:
    prober: icmp
    timeout: 5s

  dns_check:
    prober: dns
    timeout: 5s
    dns:
      query_name: "example.com"
      query_type: "A"
```

```
# Prometheus scrape config for blackbox
scrape_configs:
  - job_name: "blackbox-http"
    metrics_path: /probe
    params:
      module: [http_2xx]
    static_configs:
      - targets:
          - https://api.example.com/health
          - https://grafana.example.com
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: localhost:9115  # blackbox_exporter address

  - job_name: "blackbox-icmp"
    metrics_path: /probe
    params:
      module: [icmp]
    static_configs:
      - targets:
          - 10.0.1.1
          - 10.0.1.2
          - 10.0.1.3
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: localhost:9115
```

### Other Common Exporters

| Exporter | Port | Purpose |
|---|---|---|
| `node_exporter` | 9100 | Host CPU, memory, disk, network |
| `blackbox_exporter` | 9115 | HTTP, TCP, ICMP, DNS probing |
| `mysqld_exporter` | 9104 | MySQL/MariaDB metrics |
| `postgres_exporter` | 9187 | PostgreSQL metrics |
| `redis_exporter` | 9121 | Redis metrics |
| `consul_exporter` | 9107 | Consul service health |
| `haproxy_exporter` | 9101 | HAProxy stats |
| `nginx-exporter` | 9113 | NGINX stub_status |
| `cadvisor` | 8080 | Container metrics |
| `jmx_exporter` | 5556 | Java/JVM metrics |
| `kube-state-metrics` | 8080 | Kubernetes object state |
| `statsd_exporter` | 9102 | StatsD to Prometheus bridge |

## Instrumenting Go Applications

```
package main

import (
    "net/http"
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

var (
    httpRequestsTotal = prometheus.NewCounterVec(
        prometheus.CounterOpts{
            Name: "http_requests_total",
            Help: "Total number of HTTP requests.",
        },
        []string{"method", "path", "status"},
    )

    httpDuration = prometheus.NewHistogramVec(
        prometheus.HistogramOpts{
            Name:    "http_request_duration_seconds",
            Help:    "HTTP request duration in seconds.",
            Buckets: prometheus.DefBuckets, // .005, .01, .025, .05, .1, .25, .5, 1, 2.5, 5,
        },
        []string{"method", "path"},
    )
)

func init() {
    prometheus.MustRegister(httpRequestsTotal)
    prometheus.MustRegister(httpDuration)
}

func main() {
    // Expose metrics at /metrics
```

```
    http.Handle("/metrics", promhttp.Handler())
    http.ListenAndServe(":8080", nil)
}
```

## Service Discovery

Prometheus supports dynamic service discovery to automatically find scrape targets.

### Kubernetes Service Discovery

```
# Discover pods annotated for scraping
- job_name: "k8s-pods"
  kubernetes_sd_configs:
    - role: pod
  relabel_configs:
    # Only scrape pods with prometheus.io/scrape=true
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
      action: keep
      regex: true
    # Override the metrics path from annotation
    - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
      action: replace
      target_label: __metrics_path__
      regex: (.+)
    # Override the port from annotation
    - source_labels: [__address__, __meta_kubernetes_pod_annotation_prometheus_io_port]
      action: replace
      regex: ([^:]+)(?::\d+)?;(\d+)
      replacement: $1:$2
      target_label: __address__

# Discover services
- job_name: "k8s-services"
  kubernetes_sd_configs:
    - role: service
  relabel_configs:
    - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
      action: keep
      regex: true

# Discover endpoints (pods behind services)
- job_name: "k8s-endpoints"
  kubernetes_sd_configs:
    - role: endpoints
  relabel_configs:
    - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
      action: keep
      regex: true
```

### Consul Service Discovery

```
- job_name: "consul-services"
  consul_sd_configs:
    - server: "localhost:8500"
```

```
      services: []  # Empty = all registered services
      tags: ["production"]
  relabel_configs:
    - source_labels: [__meta_consul_service]
      target_label: service
    - source_labels: [__meta_consul_tags]
      regex: ",production,"
      action: keep
```

### EC2 Service Discovery

```
- job_name: "ec2-instances"
  ec2_sd_configs:
    - region: us-east-1
      access_key: "${AWS_ACCESS_KEY}"
      secret_key: "${AWS_SECRET_KEY}"
      port: 9100
  relabel_configs:
    - source_labels: [__meta_ec2_tag_Name]
      target_label: instance
    - source_labels: [__meta_ec2_private_ip]
      target_label: __address__
      regex: "(.*):.*"
      replacement: "$1:9100"
```

### DNS Service Discovery

```
- job_name: "dns-discovery"
  dns_sd_configs:
    - names:
        - "_prometheus._tcp.monitoring.svc.cluster.local"
      type: "SRV"
      port: 9100
      refresh_interval: 30s
```

## Grafana Dashboard Setup

### Configuration (grafana.ini)

```
[server]
protocol = http
http_port = 3000
domain = grafana.example.com
root_url = %(protocol)s://%(domain)s/

[security]
admin_user = admin
admin_password = strong_password_here

[auth.anonymous]
enabled = false

[users]
```

```
allow_sign_up = false

[alerting]
enabled = true

[unified_alerting]
enabled = true

[paths]
data = /var/lib/grafana
logs = /var/log/grafana
plugins = /var/lib/grafana/plugins
provisioning = /etc/grafana/provisioning
```

### Data Source Provisioning

```
# /etc/grafana/provisioning/datasources/prometheus.yml
apiVersion: 1
datasources:
  - name: Prometheus
    type: prometheus
    access: proxy
    url: http://prometheus:9090
    isDefault: true
    editable: false
    jsonData:
      httpMethod: POST
      timeInterval: "15s"
```

### Dashboard Provisioning

```
# /etc/grafana/provisioning/dashboards/dashboards.yml
apiVersion: 1
providers:
  - name: "default"
    orgId: 1
    folder: ""
    type: file
    disableDeletion: false
    updateIntervalSeconds: 30
    options:
      path: /etc/grafana/provisioning/dashboards/json
      foldersFromFilesStructure: false
```

## Grafana Panel Types

### Time Series (default)

The standard panel for time-series data with multiple display options.

```
{
  "type": "timeseries",
  "title": "Request Rate",
```

```
  "datasource": {"type": "prometheus", "uid": "prometheus"},
  "targets": [
    {
      "expr": "sum(rate(http_requests_total[5m])) by (method, status)",
      "legendFormat": "{{method}} {{status}}"
    }
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "reqps",
      "color": {"mode": "palette-classic"},
      "custom": {
        "drawStyle": "line",
        "lineWidth": 2,
        "fillOpacity": 10,
        "pointSize": 5,
        "showPoints": "auto",
        "spanNulls": true
      }
    }
  }
}
```

### Stat Panel

Single-value display for key metrics.

```
{
  "type": "stat",
  "title": "Uptime",
  "targets": [
    {
      "expr": "count(up == 1) / count(up) * 100",
      "legendFormat": "Uptime"
    }
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "percent",
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {"color": "red", "value": null},
          {"color": "yellow", "value": 90},
          {"color": "green", "value": 99}
        ]
      },
      "mappings": []
    },
    "overrides": []
  },
  "options": {
    "reduceOptions": {
      "calcs": ["lastNotNull"],
      "fields": "",
      "values": false
    },
    "orientation": "auto",
```

```
    "textMode": "auto",
    "colorMode": "background",
    "graphMode": "area"
  }
}
```

### Table Panel

Tabular display of current values.

```
{
  "type": "table",
  "title": "Node Status",
  "targets": [
    {
      "expr": "node_uname_info",
      "format": "table",
      "instant": true
    },
    {
      "expr": "100 - (avg by(instance) (rate(node_cpu_seconds_total{mode=\"idle\"}[5m])) * 10
      "format": "table",
      "instant": true
    },
    {
      "expr": "(1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100",
      "format": "table",
      "instant": true
    }
  ],
  "transformations": [
    {"id": "merge"},
    {"id": "organize", "options": {
      "excludeByName": {"Time": true, "__name__": true, "job": true},
      "indexByName": {"instance": 0, "nodename": 1, "Value #A": 2, "Value #B": 3},
      "renameByName": {
        "instance": "Instance",
        "nodename": "Hostname",
        "Value #A": "CPU %",
        "Value #B": "Memory %"
      }
    }}
  ]
}
```

### Gauge, Bar Gauge, and Pie Charts

```
{
  "type": "gauge",
  "title": "CPU Utilization",
  "targets": [
    {"expr": "100 - (avg by(instance) (rate(node_cpu_seconds_total{mode=\"idle\"}[5m])) * 100
  ],
  "fieldConfig": {
    "defaults": {
      "unit": "percent",
```

```
      "min": 0,
      "max": 100,
      "thresholds": {
        "mode": "absolute",
        "steps": [
          {"color": "green", "value": null},
          {"color": "yellow", "value": 70},
          {"color": "red", "value": 90}
        ]
      }
    }
  },
  "options": {
    "reduceOptions": {"calcs": ["lastNotNull"], "fields": "", "values": false},
    "orientation": "auto",
    "showThresholdLabels": false,
    "showThresholdMarkers": true
  }
}
```

### Log Panel

View logs correlated with metrics (requires Loki).

```
{
  "type": "logs",
  "title": "Application Logs",
  "datasource": {"type": "loki", "uid": "loki"},
  "targets": [
    {
      "expr": "{app=\"myapp\", namespace=\"production\"} |= \"error\" | json"
    }
  ],
  "options": {
    "showTime": true,
    "showLabels": true,
    "wrapLogMessage": true,
    "sortOrder": "Descending"
  }
}
```

## Grafana Variables & Templates

Variables make dashboards dynamic and reusable.

### Variable Definitions

```
# dashboard JSON — variables section
templating:
  list:
    # Query variable — values from Prometheus
    - name: datasource
      type: datasource
      query: "prometheus"
      current: { selected: true, text: "Prometheus", value: "Prometheus" }
```

```
    - name: namespace
      type: query
      datasource: "$datasource"
      query: "label_values(kube_pod_info, namespace)"
      refresh: 2  # On dashboard load
      multi: true
      includeAll: true
      allValue: ".*"

    - name: pod
      type: query
      datasource: "$datasource"
      query: "label_values(kube_pod_info{namespace=~\"$namespace\"}, pod)"
      refresh: 2
      multi: true
      includeAll: true
      allValue: ".*"

    - name: instance
      type: query
      datasource: "$datasource"
      query: "label_values(up, instance)"
      refresh: 1  # Manual refresh only
      multi: false

    # Custom values
    - name: quantile
      type: custom
      current: { selected: true, text: "0.95", value: "0.95" }
      options:
        - { text: "P50", value: "0.5" }
        - { text: "P90", value: "0.9" }
        - { text: "P95", value: "0.95" }
        - { text: "P99", value: "0.99" }

    # Interval variable (for rate[] windows)
    - name: interval
      type: interval
      current: { selected: true, text: "5m", value: "5m" }
      options:
        - { text: "1m", value: "1m", selected: false }
        - { text: "5m", value: "5m", selected: true }
        - { text: "15m", value: "15m", selected: false }
        - { text: "30m", value: "30m", selected: false }
        - { text: "1h", value: "1h", selected: false }

    # Text box — user input
    - name: search_query
      type: textbox
      current: { selected: true, text: "", value: "" }
      placeholder: "Filter by label..."
```

### Using Variables in Queries

```
# Use variable in label matchers
rate(http_requests_total{namespace="$namespace", pod=~"$pod"}[$interval])
```

```
# Use quantile variable
histogram_quantile($quantile, sum(rate(http_request_duration_seconds_bucket[$interval])) by (

# Regex with variable
up{job=~"$job"} == 0

# Multi-select with regex
sum(rate(http_requests_total{namespace=~"$namespace"}[5m])) by (pod)
```

## Alert Channels & Notifications

### Grafana Alerting (Unified Alerting)

```
# /etc/grafana/provisioning/alerting/contact-points.yml
apiVersion: 1
contactPoints:
  - name: Slack Channel
    orgId: 1
    receivers:
      - uid: slack_receiver
        type: slack
        settings:
          endpointUrl: "https://hooks.slack.com/services/T00/B00/xxx"
          channel: "#alerts"
          title: "{{ .CommonLabels.alertname }}"
          text: >-
            {{ range .Alerts }}
            *Alert:* {{ .Labels.alertname }}
            *Severity:* {{ .Labels.severity }}
            *Instance:* {{ .Labels.instance }}
            *Value:* {{ .ValueString }}
            {{ end }}
```

### Grafana Alert Rules

```
# /etc/grafana/provisioning/alerting/alert-rules.yml
apiVersion: 1
groups:
  - name: application.alerts
    folder: Application
    interval: 1m
    rules:
      - uid: high_error_rate
        title: High Error Rate
        condition: C
        data:
          - refId: A
            datasourceUid: prometheus
            model:
              expr: sum(rate(http_requests_total{status=~"5.."}[5m])) by (service)
                / sum(rate(http_requests_total[5m])) by (service)
              instant: true
              intervalMs: 1000
              legendFormat: "{{service}}"
          - refId: B
```

```
            datasourceUid: "__expr__"
            model:
              type: threshold
              expression: "A"
              conditions:
                - evaluator:
                    type: gt
                    params: [0.05]
                  operator:
                    type: and
                  reducer:
                    type: last
                    params: []
                  query:
                    params: [A]
          - refId: C
            datasourceUid: "__expr__"
            model:
              type: math
              expression: "$B"
        # NoData state
        noDataState: NoData
        # ExecErrState
        execErrState: Error
        for: 5m
        annotations:
          description: "Error rate is {{ $values.A }} for service {{ $labels.service }}"
          summary: "High error rate detected"
        labels:
          severity: warning
```

### Prometheus vs Grafana Alerting

| Feature | Prometheus Alertmanager | Grafana Unified Alerting |
|---|---|---|
| **Evaluated by** | Prometheus server | Grafana server |
| **Multi-datasource** | Prometheus only | Prometheus, Loki, Tempo, etc. |
| **Routing** | Complex tree-based routing | Contact point tags |
| **Grouping** | Built-in group_by + timings | Per alert rule |
| **Silencing** | Built-in silences API | Built-in silence & mute timings |
| **Templates** | Go templates | Go templates |
| **Best for** | Infrastructure & Kubernetes alerts | Application & cross-datasource alerts |

## Retention & Storage

### Local Storage Tuning

```
# Start Prometheus with custom retention
prometheus \
  --storage.tsdb.retention.time=60d \
  --storage.tsdb.retention.size=500GB \
  --storage.tsdb.wal-compression \
```

```
  --storage.tsdb.wal-segment-size=64MB

# Compaction and chunk management
# Prometheus automatically compacts TSDB blocks
# Check block status:
promtool tsdb status /var/lib/prometheus/data

# Inspect a block:
promtool tsdb inspect /var/lib/prometheus/data/blocks/<block-id>

# Clean tombstones:
promtool tsdb clean /var/lib/prometheus/data
```

### Long-Term Storage with Thanos

```
# thanos-sidecar (runs alongside Prometheus)
# Exposes StoreAPI and uploads to object storage
thanos:
  sidecar:
    extraArgs:
      tsdb.path: /prometheus
      objstore.config-file: /etc/thanos/objstore.yml
      grpc-address: "0.0.0.0:10901"
      http-address: "0.0.0.0:10902"

# objstore.yml — S3-compatible storage
type: S3
config:
  bucket: "prometheus-long-term"
  endpoint: "s3.amazonaws.com"
  region: "us-east-1"
  access_key: "${AWS_ACCESS_KEY}"
  secret_key: "${AWS_SECRET_KEY}"
```

```
# thanos-compactor — downsamples and compacts blocks
thanos:
  compact:
    extraArgs:
      retention.resolution.raw: 30d
      retention.resolution.5m: 90d
      retention.resolution.1h: 180d
      objstore.config-file: /etc/thanos/objstore.yml
```

### Remote Write Alternatives

| Solution | Description | Best For |
|---|---|---|
| **Thanos** | Sidecar + store gateway + compactor + query | Kubernetes, S3 storage |
| **Cortex** | Horizontally scalable, multi-tenant | Large-scale, multi-tenant |
| **Mimir** | Successor to Cortex, single binary mode | Grafana Cloud, self-hosted |
| **VictoriaMetrics** | High performance, compatible API | Cost-effective long-term storage |
| **TimescaleDB** | PostgreSQL extension with Prometheus support | SQL-based querying needs |

## Production Best Practices

### Relabeling Patterns

```
# Drop noisy metrics
metric_relabel_configs:
  - source_labels: [__name__]
    regex: "go_goroutines|go_memstats_alloc_bytes|process_start_time_seconds"
    action: drop

# Drop specific labels to reduce cardinality
  - source_labels: [__name__]
    regex: "container_.*"
    action: labeldrop
    regex: "container_id|image"

# Keep only specific namespaces
  - source_labels: [namespace]
    regex: "production|staging"
    action: keep

# Rename instance to show just hostname (strip port)
  - source_labels: [instance]
    target_label: instance
    regex: "(.*):\\d+"
    replacement: "$1"
```

### High Availability

```
# Prometheus HA with Thanos sidecar — multiple replicas
# Each Prometheus scrapes the same targets
# Thanos query frontend deduplicates results

# Replica label for deduplication
global:
  external_labels:
    replica: "prometheus-1"  # prometheus-2 on the other replica

# Thanos query configuration
# query.frontend -- deduplicates results from multiple store APIs
```

### Cardinality Management

```
# Avoid high-cardinality labels
# BAD — user_id can have millions of values
http_requests_total{user_id="12345", path="/api"}

# GOOD — use histogram bucket + aggregate
histogram_quantile(0.99, sum(rate(http_request_duration_seconds_bucket[5m])) by (le, endpoint

# Monitor cardinality
# Query: count by (__name__) ({__name__=~".+"})
# Alert when any metric exceeds label count threshold
```

### Performance Tips

```
# Check TSDB head stats
curl -s http://localhost:9090/api/v1/status/tsdb | jq

# Check target health
curl -s http://localhost:9090/api/v1/targets | jq '.data.activeTargets[] | {job, health, last

# Check rule evaluation stats
curl -s http://localhost:9090/api/v1/status/rules | jq

# Check config validation
promtool check config /etc/prometheus/prometheus.yml

# Check rules validation
promtool check rules /etc/prometheus/alerts/*.yml

# Benchmark a query
promtool query benchmark 'sum(rate(http_requests_total[5m])) by (service)'
```

### Quick Reference: Essential PromQL

| Use Case | Query |
|---|---|
| CPU % | `100 - (avg by (instance)(rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)` |
| Memory % | `(1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100` |
| Disk % | `(1 - node_filesystem_avail_bytes / node_filesystem_size_bytes) * 100` |
| Request rate | `sum(rate(http_requests_total[5m])) by (service)` |
| Error rate % | `sum(rate(http_requests_total{status=~"5.."}[5m])) by (service) / sum(rate(http_requests_total[5m])) by (service) * 100` |
| P95 latency | `histogram_quantile(0.95, sum(rate(http_request_duration_seconds_bucket[5m])) by (le, service))` |
| Uptime % | `avg(up) * 100` |
| Pod restarts | `sum(rate(kube_pod_container_status_restarts_total[1h])) by (namespace, pod)` |
| Disk fill prediction | `predict_linear(node_filesystem_avail_bytes[1h], 4 * 3600)` |
| Network I/O | `rate(node_network_receive_bytes_total{device="eth0"}[5m])` |

## Next Steps

- Kubernetes Cheat Sheet — Cluster management commands
- Docker Cheat Sheet — Container operations reference
- Terraform Cheat Sheet — Infrastructure as code reference
- Helm Cheat Sheet — Kubernetes package management