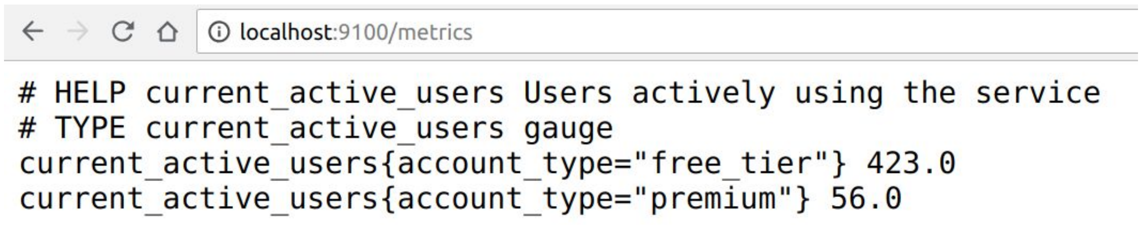
Monitoring with Prometheus Stack

What is Monitoring

- Monitoring applications & application servers is an important part of the today's DevOps culture & process.
- You want to continuously monitor your applications and servers for application exceptions, server CPU & memory usage, or storage spikes.
- You also want to get some type of notification if CPU or memory usage goes up for a certain period of time or a service of your application stops responding so you can perform appropriate actions against those failures or exceptions.
- Metrics are the primary way to represent both the overall health of your system and any other specific information you consider important for monitoring and alerting or observability

Monitoring with Prometheus

- Prometheus is an open source monitoring and alerting system that directly scrapes metrics from agents running on the target hosts and stores the collected samples centrally on its server.
- Multi-dimensional data model: The model is based on key-value pairs, similar to how Kubernetes itself organizes infrastructure metadata using labels. It allows for flexible and accurate time series data, powering its Prometheus query language.
- Accessible format and protocols: Exposing prometheus metrics is a pretty straightforward task. Metrics are human readable, are in a self-explanatory format, and are published using a standard HTTP transport. You can check that the metrics are correctly exposed just using your web browser:



```
# HELP current_active_users Users actively using the service
# TYPE current_active_users gauge
current_active_users{account_type="free_tier"} 423.0
current_active_users{account_type="premium"} 56.0
```

- Service discovery: The Prometheus server is in charge of periodically scraping the targets, so that applications and services don't need to worry about emitting data (metrics are pulled, not pushed). These Prometheus servers have several methods to auto-discover scrape targets, some of them can be configured to filter and match container metadata, making it an excellent fit for ephemeral Kubernetes workloads.
- Modular and highly available components: Metric collection, alerting, graphical visualization, etc, are performed by different composable services. All these services are designed to support redundancy and sharding.

Prometheus Components

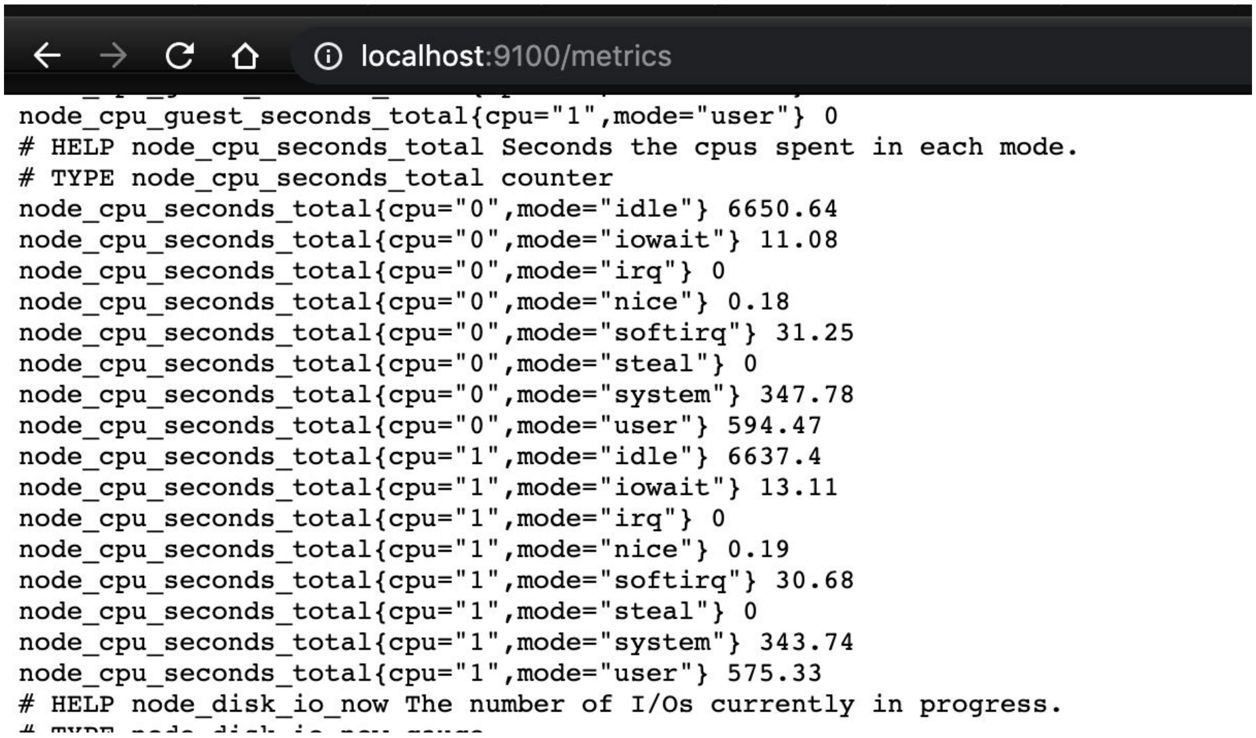
A typical monitoring platform with Prometheus is composed of multiple tools:

- Multiple exporters that typically run on the monitored host to export local metrics.
- Prometheus to centralize and store the metrics.
- Alertmanager to trigger alerts based on those metrics.
- Grafana to produce dashboards.
- PromQL is the query language used to create dashboards and alerts.

Prometheus Exporter

An exporter is a binary running alongside the application you want to obtain metrics from. The exporter exposes Prometheus metrics, commonly by converting metrics that are exposed in a non-Prometheus format into a format that Prometheus supports. Exporters can expose a number of metric types:

- Counters - A counter is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart.
- Gauges - A gauge is a metric that represents a single numerical value that can arbitrarily go up and down.
- Histograms - A histogram samples observations (usually things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.
- Quantiles are convenient when (for example) expressing median (2-quantile) and 95th percentiles.



```
node_cpu_guest_seconds_total{cpu="1",mode="user"} 0
# HELP node_cpu_seconds_total Seconds the cpus spent in each mode.
# TYPE node_cpu_seconds_total counter
node_cpu_seconds_total{cpu="0",mode="idle"} 6650.64
node_cpu_seconds_total{cpu="0",mode="iowait"} 11.08
node_cpu_seconds_total{cpu="0",mode="irq"} 0
node_cpu_seconds_total{cpu="0",mode="nice"} 0.18
node_cpu_seconds_total{cpu="0",mode="softirq"} 31.25
node_cpu_seconds_total{cpu="0",mode="steal"} 0
node_cpu_seconds_total{cpu="0",mode="system"} 347.78
node_cpu_seconds_total{cpu="0",mode="user"} 594.47
node_cpu_seconds_total{cpu="1",mode="idle"} 6637.4
node_cpu_seconds_total{cpu="1",mode="iowait"} 13.11
node_cpu_seconds_total{cpu="1",mode="irq"} 0
node_cpu_seconds_total{cpu="1",mode="nice"} 0.19
node_cpu_seconds_total{cpu="1",mode="softirq"} 30.68
node_cpu_seconds_total{cpu="1",mode="steal"} 0
node_cpu_seconds_total{cpu="1",mode="system"} 343.74
node_cpu_seconds_total{cpu="1",mode="user"} 575.33
# HELP node_disk_io_now The number of I/Os currently in progress.
```

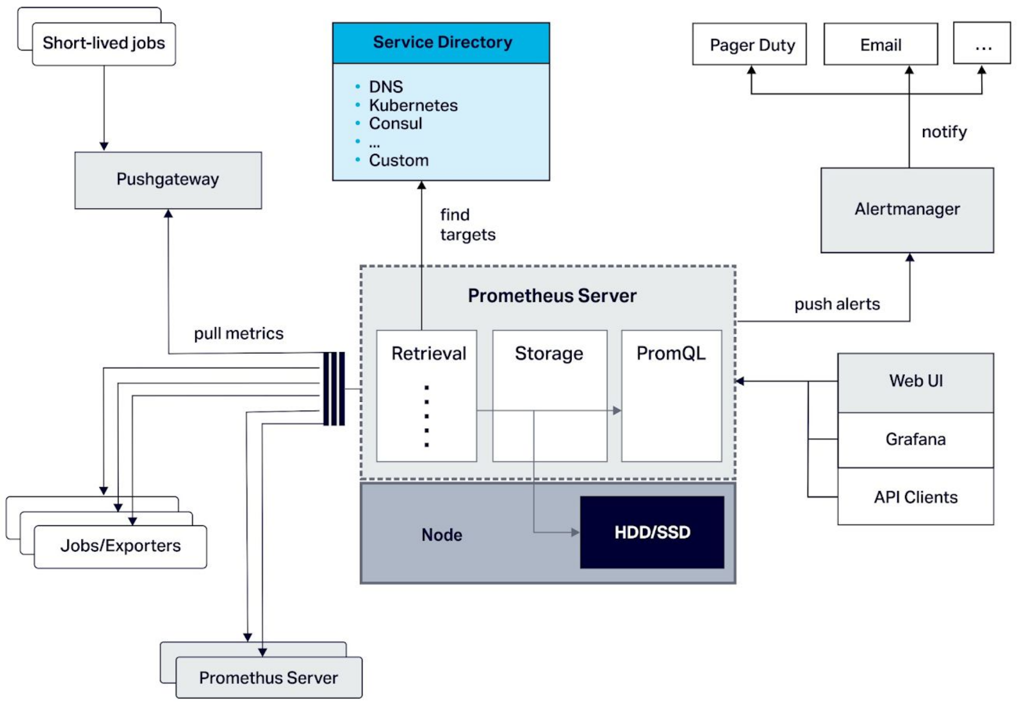
Metric Collection

- After setting up our exporters, we will need to have the ability to collect their content and save it to prometheus server.
- Prometheus works in a way that it is pulling the metrics from the exporters.
- When we want to configure a new exporter we will need to setup a new target.
- A target is the definition of an object to scrape. For example, what labels to apply, any authentication required to connect, or other information that defines how the scrape will occur.
- Once we have our exporters up and running and showing metrics we will need to configure a new job for prometheus server.
- Job can be seen as a collection of targets with the same purpose.

Configuring Prometheus Server

When we configure a prometheus server we will configure the properties:

1. Global - The global configuration specifies parameters that are valid in all other configuration contexts. They also serve as defaults for other configuration sections.
2. Rules - Alerting rules definitions to alert to Alertmanager.
3. Scrape Configs - Where we define our scraping jobs from different targets we defined earlier.
4. Alerting - Alerting specifies settings related to the Alertmanager.



Pushgateway

- The Pushgateway is an intermediary service which allows you to push metrics from jobs which cannot be scraped.
- It is only recommended using the Pushgateway in certain limited cases.
- There are several pitfalls when blindly using the Pushgateway instead of Prometheus's usual pull model for general metrics collection:
- When monitoring multiple instances through a single Pushgateway, the Pushgateway becomes both a single point of failure and a potential bottleneck.
- You lose Prometheus's automatic instance health monitoring via the up metric (generated on every scrape).
- The Pushgateway never forgets series pushed to it and will expose them to Prometheus forever unless those series are manually deleted via the Pushgateway's API.

PromQL

PromQL is the Prometheus Query Language. It allows for a wide range of operations including aggregation, slicing and dicing, prediction and joins.

1. For instance, the following query would return all the time series with name node_network_receive_bytes_total
2. How to select time series matching only device="eth1": node_network_receive_bytes_total{device="eth1"}
3. Regex is supported with =~
4. We can also filter by multiple labels node_network_receive_bytes_total{instance="node42:9100", device=~"eth.+"}
5. PromQL allows querying historical data and combining / comparing it to the current data: node_network_receive_bytes_total offset 7d
6. rate - calculates per-second rate for all the matching time series: rate(node_network_receive_bytes_total[5m])

More PromQL examples can be found here:
https://medium.com/@valyala/promql-tutorial-for-beginners-9ab455142085

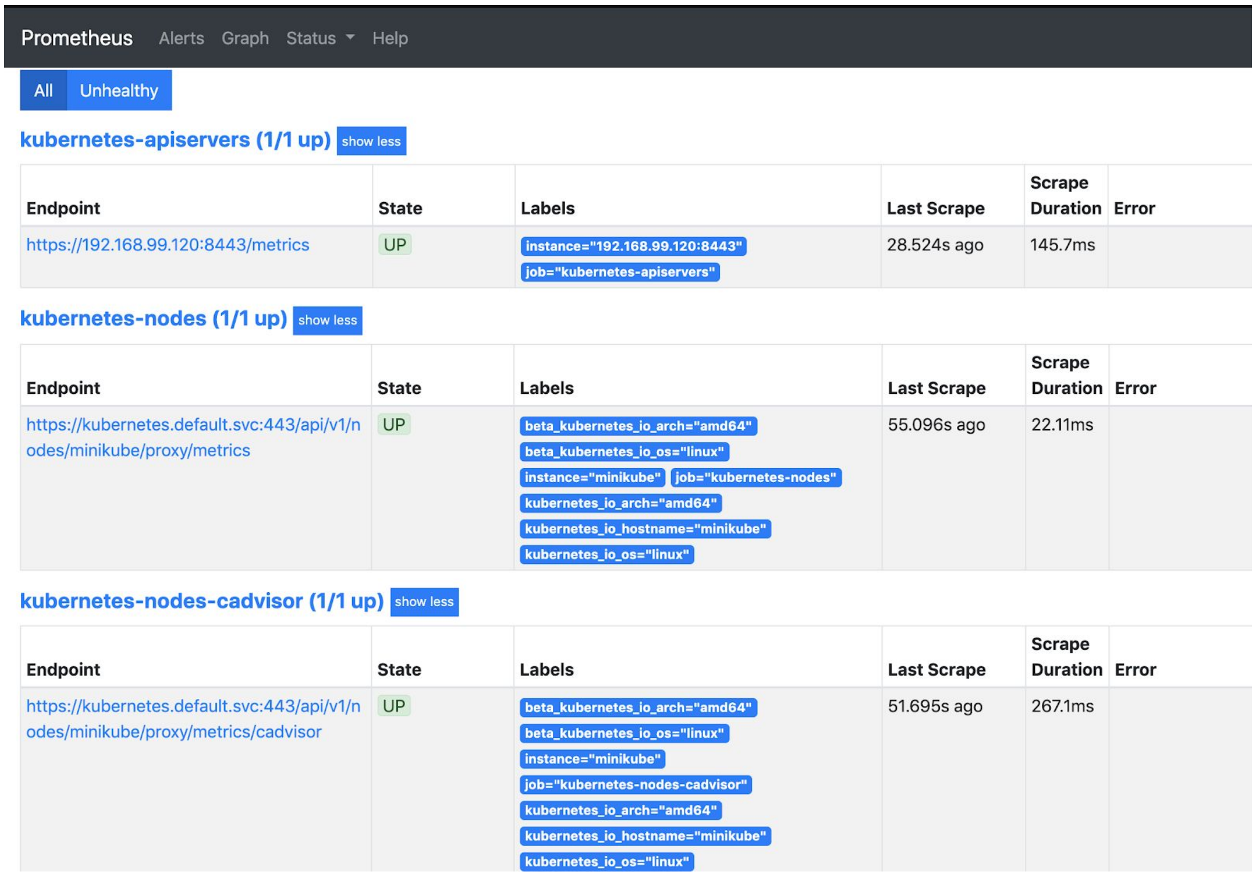
Service Discovery

- Prometheus server enables us to automate the process of updating our targets list
- Prometheus has a the understanding of service discovery
- Service discovery is the ability to dynamically know the list of services serving a specific application
- Prometheus can delegate it's "target discovery" onto a third party service discovery tool
- Kubernetes for example can be defined as a service discovery tool for kubernetes services / nodes / pods and so on

Prometheus Web UI

Prometheus also exposes a web UI for several reasons:

- Allow us to run ad-hoc PromQL queries
- Check the status of our alerts
- Check status of our targets
- Prometheus server configuration
- Service discovery status



kubernetes-apiservers (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://192.168.99.120:8443/metrics	UP	instance="192.168.99.120:8443" job="kubernetes-apiservers"	28.524s ago	145.7ms	

kubernetes-nodes (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://kubernetes.default.svc:443/api/v1/nodes/minikube/proxy/metrics	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" instance="minikube" job="kubernetes-nodes" kubernetes_io_arch="amd64" kubernetes_io_hostname="minikube" kubernetes_io_os="linux"	55.096s ago	22.11ms	

kubernetes-nodes-cadvisor (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
https://kubernetes.default.svc:443/api/v1/nodes/minikube/proxy/metrics/cadvisor	UP	beta_kubernetes_io_arch="amd64" beta_kubernetes_io_os="linux" instance="minikube" job="kubernetes-nodes-cadvisor" kubernetes_io_arch="amd64" kubernetes_io_hostname="minikube" kubernetes_io_os="linux"	51.695s ago	267.1ms	

Alert Manager

- The Alertmanager handles alerts sent by client applications such as the Prometheus server.
- It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty, or OpsGenie.
- Silences are a straightforward way to simply mute alerts for a given time. A silence is configured based on matchers, just like the routing tree. Incoming alerts are checked whether they match all the equality or regular expression matchers of an active silence. If they do, no notifications will be sent out for that alert.
- Grouping categorizes alerts of similar nature into a single notification. This is especially useful during larger outages when many systems fail at once and hundreds to thousands of alerts may be firing simultaneously.
- Inhibition is a concept of suppressing notifications for certain alerts if certain other alerts are already firing.

```
# The root route with all parameters, which are inherited by the child
# routes if they are not overwritten.
route:
  receiver: 'default-receiver'
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  group_by: [cluster, alertname]
  # All alerts that do not match the following child routes
  # will remain at the root node and be dispatched to 'default-receiver'.
  routes:
  # All alerts with service=mysql or service=cassandra
  # are dispatched to the database pager.
  - receiver: 'database-pager'
    group_wait: 10s
    match_re:
      service: mysql|cassandra
  # All alerts with the team=frontend label match this sub-route.
  # They are grouped by product and environment rather than cluster
  # and alertname.
  - receiver: 'frontend-pager'
    group_by: [product, environment]
    match:
      team: frontend
```

Grafana

Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore, and share dashboards with your team and foster a data driven culture:

- Visualize: Fast and flexible client side graphs with a multitude of options. Panel plugins for many different way to visualize metrics and logs.
- Dynamic Dashboards: Create dynamic & reusable dashboards with template variables that appear as dropdowns at the top of the dashboard.
- Explore Metrics: Explore your data through ad-hoc queries and dynamic drilldown. Split view and compare different time ranges, queries and data sources side by side.
- Explore Logs: Experience the magic of switching from metrics to logs with preserved label filters. Quickly search through all your logs or streaming them live.
- Alerting: Visually define alert rules for your most important metrics. Grafana will continuously evaluate and send notifications to systems like Slack, PagerDuty, VictorOps, OpsGenie.
- Mixed Data Sources: Mix different data sources in the same graph! You can specify a data source on a per-query basis. This works for even custom datasources.

Grafana includes built-in support for Prometheus
Adding prometheus data source:

1. Open the side menu by clicking the Grafana icon in the top header.
2. In the side menu under the Dashboards link you should find a link named Data Sources.
3. Click the + Add data source button in the top header.
4. Select Prometheus from the Type dropdown.

Deploy the stack:

Go to monitoring/k8s-setup
follow steps detailed at commands.txt