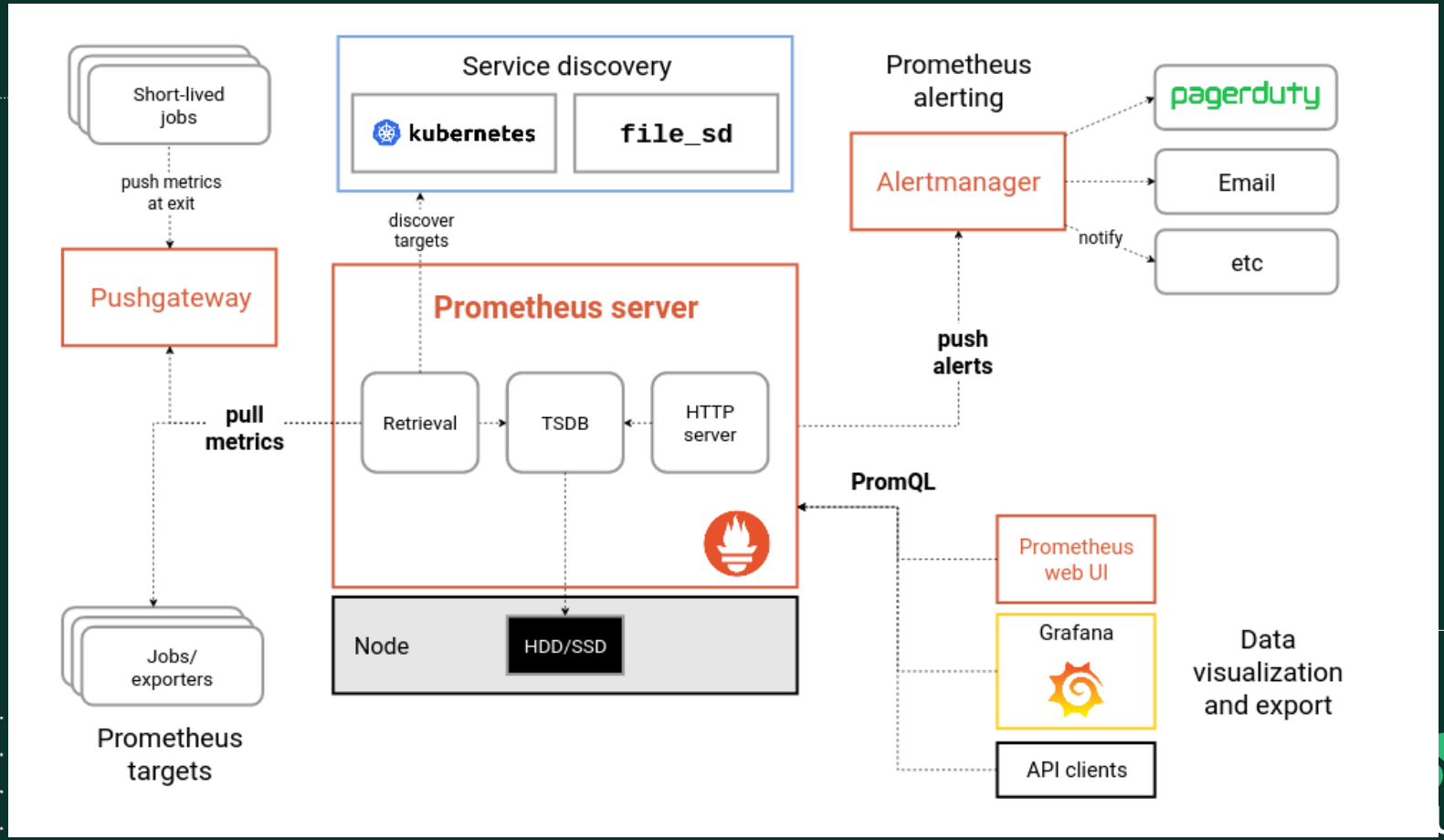
Self-healing with prometheus alerts

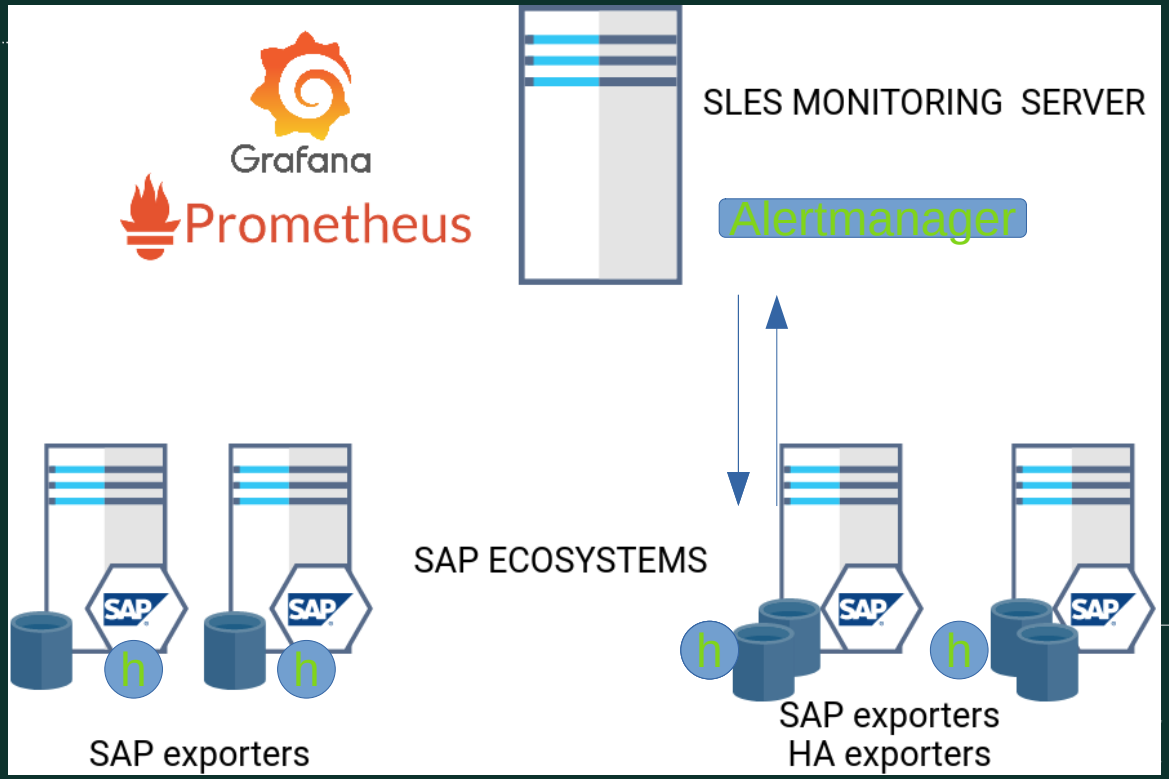
Dario Maiocchi <dmaiocchi@suse.com>

Warning:

the implementation here is for POC purposes.

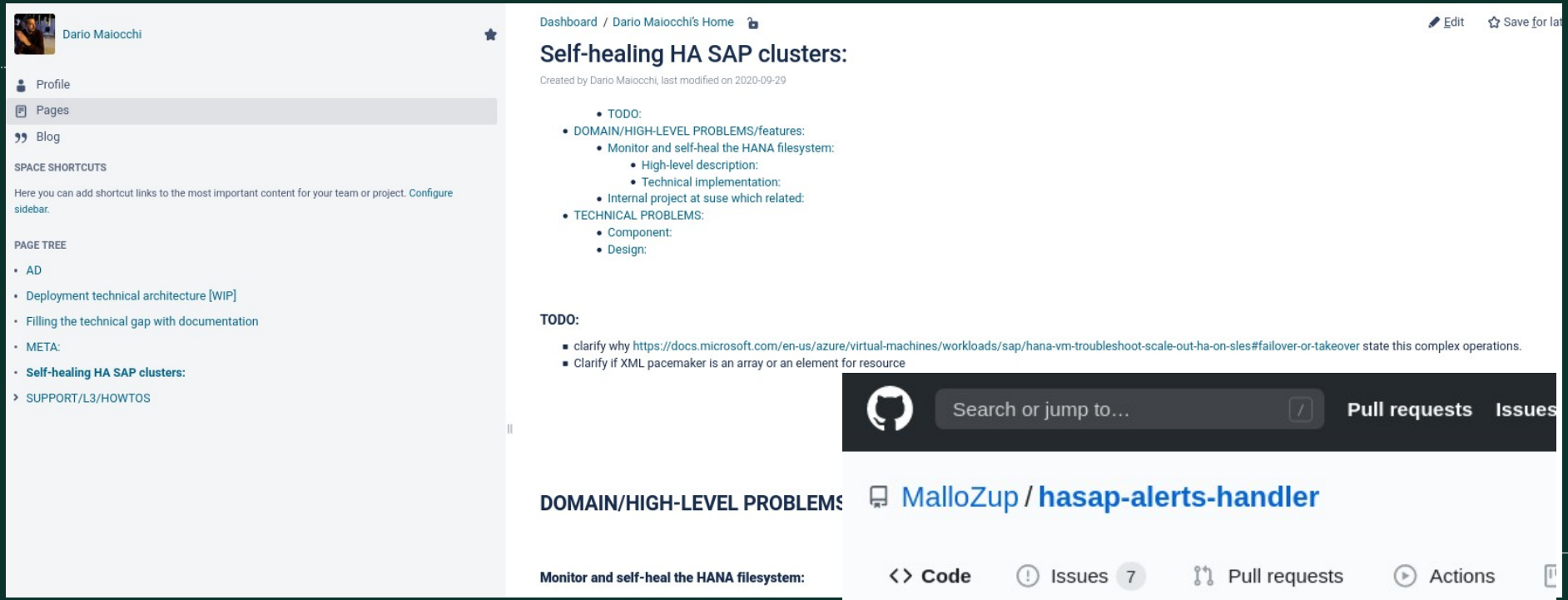

Short-lived jobs
push metrics at exit
Pushgateway
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
pull metrics
Node
HDD/SSD
Jobs/ exporters
Prometheus targets
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

This is a possible architecture. (simplest)



h= handler. Each server has a new binary, the handler which will handle the alerts

For technical details, infos check my confluence page



RFC will come later. (didn't had time)

Features:

- Secure (https) possible.

- Self-heal on specific alerts, based on declarative API

- Raise alerts back to alertmanager in case selfheal fail
  (can be used for other purposes, like monitor config file etc)

Declarative api

```
- name: hana filesystem monitoring
  rules:
# all critical severity will be handled as self-healing
  - alert: HanaFileSystemFull
    expr: ((node_filesystem_size_bytes{mountpoint="/hana"} - node_filesyste
    labels:
      severity: critical
      selfhealing: true
      component: hana
    annotations:
      summary: Hana file systems is full more then %95 percent
```

SUSE

Declarative api

```
- name: hana filesystem monitoring
  rules:
# all critical severity will be handled as self-healing
  - alert: HanaFileSystemFull
    expr: ((node_filesystem_size_bytes{mountpoint="/hana"} - node_filesyste
    labels:
      severity: critical
      selfhealing: true
      component: hana
    annotations:
      summary: Hana file systems is full more then %95 percent
```

labels play a role on the handler

this will be documented clearly

SUSE

```
- name: hana filesystem monitoring
  rules:
# all critical severity will be handled as self-healing
  - alert: HanaFileSystemFull
    expr: ((node_filesystem_size_bytes{mountpoint="/hana"} - node_filesyste
    labels:
      severity: critical
      selfhealing: true
      component: hana
    annotations:
      summary: Hana file systems is full more then %95 percent
```

severity:

define how severe is an alert.

SUSE

```
- name: hana filesystem monitoring
  rules:
# all critical severity will be handled as self-healing
  - alert: HanaFileSystemFull
    expr: ((node_filesystem_size_bytes{mountpoint="/hana"} - node_filesyste
    labels:
      severity: critical
      selfhealing: true
      component: hana
    annotations:
      summary: Hana file systems is full more then %95 percent
```

selfhealing:

Disable selfhealing or enable it.
If omitted is disabled on that alert.

SUSE

```
- name: hana filesystem monitoring
  rules:
# all critical severity will be handled as self-healing
  - alert: HanaFileSystemFull
    expr: ((node_filesystem_size_bytes{mountpoint="/hana"} - node_filesyste
    labels:
      severity: critical
      selfhealing: true
      component: hana
    annotations:
      summary: Hana file systems is full more then %95 percent
```

component:

Infuence routing of alerts

Alertermanager

Route and filter alert, send them to various handlers

```
route:
  receiver: 'default-receiver'
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 10s
  group_by: [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 component netweaver or hana are dispatched to sap-hook.
  - receiver: 'hana-hook-receiver'
    # overwrite default root value
    # group_wait: 10s
    match_re:
      component: hana
  # All alerts with the component label match this sub-route.
  - receiver: 'ha-hook-receiver'
    # group_by: [product, environment]
    match:
      component: sbd
~
```

SUSE

Templates are possible

```
global:
  slack_api_url: '<slack_webhook_url>'

route:
- receiver: 'slack-notifications'
  group_by: [alertname, datacenter, app]

receivers:
- name: 'slack-notifications'
  slack_configs:
  - channel: '#alerts'
    text: '{{ template "slack.myorg.text" . }}'

templates:
- '/etc/alertmanager/templates/myorg.tmpl'
```

https://prometheus.io/blog/2016/03/03/custom-alertmanager-templates/

```
global:
receivers:
      - name: default-receiver
        webhook_configs:
          - url: "http://10.162.29.223:9999/hooks-default"
          - url: "http://10.162.32.38:9999/hooks-default"
          # this is for debugging
          - url: "http://10.163.16.126:9999/hooks-default"

      - name: ha-hook-receiver
        webhook_configs:
          - url: "http://10.162.29.223:9999/hooks-ha"
          # this is for debugging
          - url: "http://10.163.16.126:9999/hooks-default"

      - name: hana-hook-receiver
        webhook_configs:
          - url: "http://10.162.29.223:9999/hooks-sap"
          # this is for debugging
          - url: "http://10.163.16.126:9999/hooks-sap"


route:
  receiver: 'default-receiver'
  group_wait: 10s
  group_interval: 10s
  repeat_interval: 10s
  group_by: [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 component netweaver or hana are dispatched to sap-hook.
  - receiver: 'hana-hook-receiver'
    # overwrite default root value
    # group_wait: 10s
    match_re:
      component: hana
  # All alerts with the component label match this sub-route.
  - receiver: 'ha-hook-receiver'
    # group_by: [product, environment]
    match:
      component: sbd
~
```

SUSE

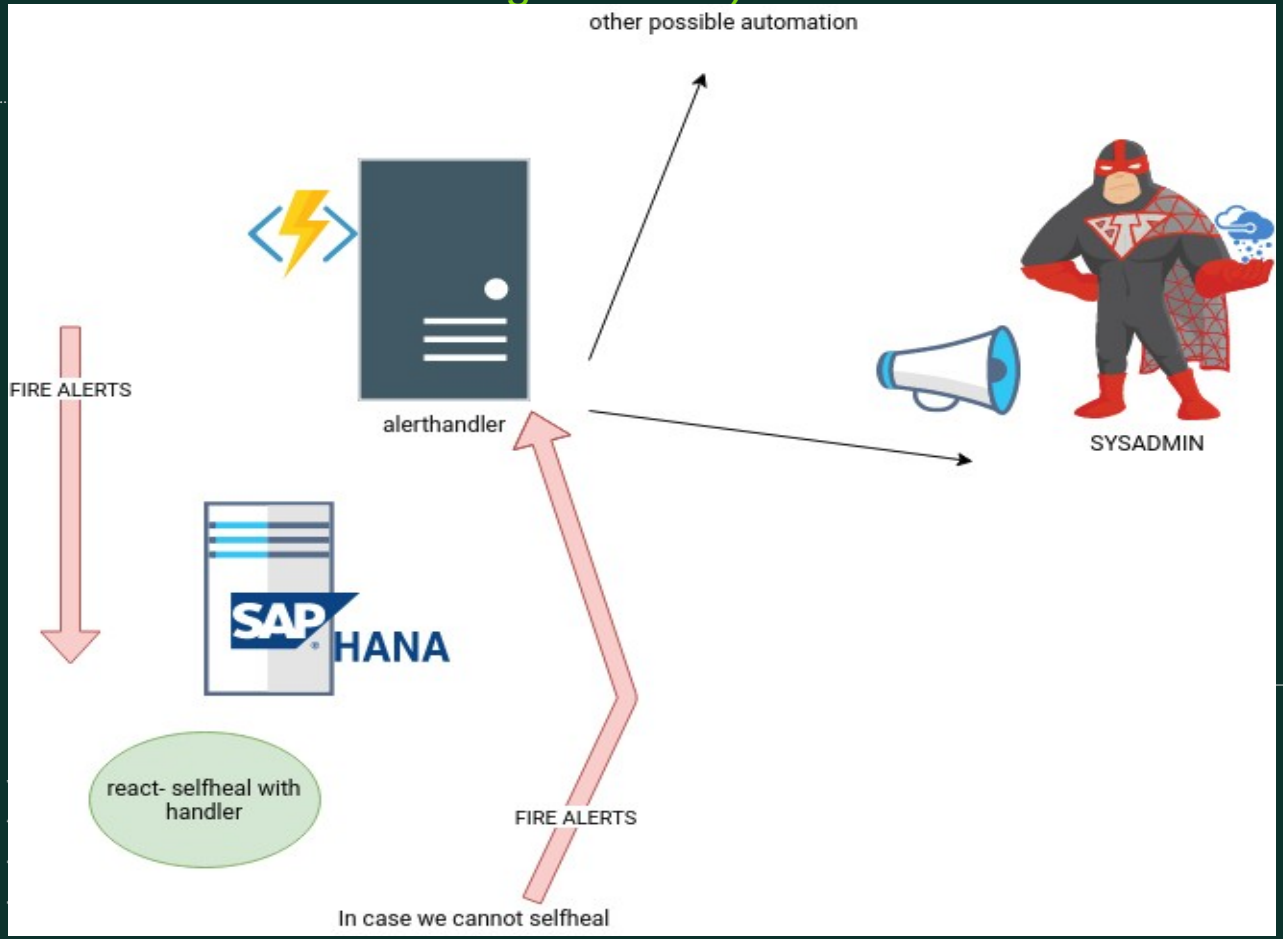
From an application codebase we can fire alerts.
This is needed for the handlers when they cannot selfheal

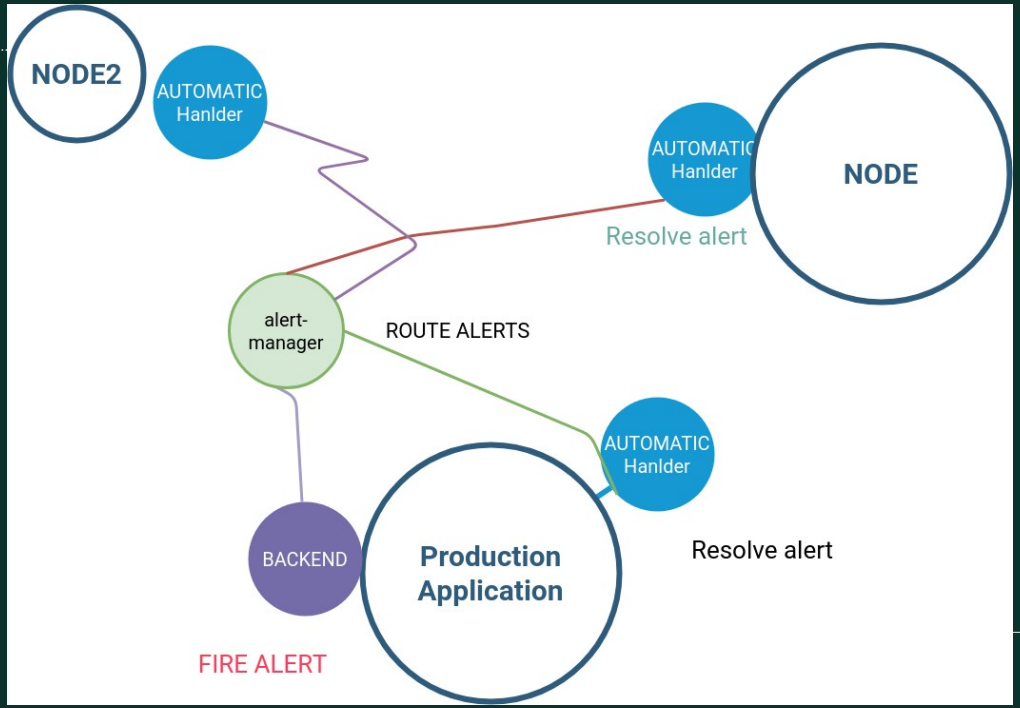
but it can be reused for other purposes, like monitoring configuration files change etc..

1) Fire alert from your `application`

```
if diskFull == True {  // your critical condition
  log.Error("full disk ... etc")
  // ..
   var a *AlertFire
   a = new(AlertFire)
   a.Status = "firing"
   a.Labels.Alertname = "FOO-ALERT"
   a.Labels.Component = "unit-test component"
   a.Labels.Severity = "critical"
   a.Labels.Instance = "test instance"
   a.Annotations.Summary = "just a test"
   a.GeneratorURL = "unit-test"
   // this is your prometheus alertmanager server
   a.sendAlert("http://10.162.31.2:9093/api/v1/alerts")
}
```

Firing alerts from application codebase.
(This is the inverse of reacting on alerts)





For details:

https://mallozup.github.io/posts/selfhealing-systems-part2/

SUSE

DEMO

Thank you.