

Blog / Engineering

Step-by-step guide to setting up Prometheus Alertmanager with Slack, PagerDuty, and Gmail



Ivana Huckova • February 25, 2020 • 8 min

In my previous blog post, "[How to explore Prometheus with easy 'Hello World' projects](#)," I described three projects that I used to get a better sense of what Prometheus can do. In this post, I'd like to share how I got more familiar with [Prometheus Alertmanager](#) and how I set up alert notifications for Slack, PagerDuty, and Gmail.

(I'm going to reference my previous blog post quite a bit, so I recommend reading it before continuing on.)

The basics

You can get started with [Prometheus](#) in minutes with [Grafana Cloud](#). We have free and paid Grafana Cloud plans to suit every use case — sign up for free now.

Setting up alerts with Prometheus is a two-step process:

To start, you need to create your alerting rules in Prometheus, and specify under what conditions you want to be alerted (such as when an instance is down).

Second, you need to set up [Alertmanager](#), which receives the alerts specified in Prometheus. Alertmanager will then be able to do a variety of things, including:

- grouping alerts of similar nature into a single notification
- silencing alerts for a specific time
- muting notifications for certain alerts if other specified alerts are already firing
- picking which receivers receive a particular alert

Step 1: Create alerting rules in Prometheus

We are starting with four subfolders that we've previously set up for each project: **server**, **node_exporter**, **github_exporter**, and **prom_middleware**. The process is explained in my blog post on how to explore Prometheus with easy projects [here](#).

We move to **server** subfolder and open the content in the code editor, then create a new rules file. In the **rules.yml**, you will specify the conditions when you would like to be alerted.

```
cd Prometheus/server  
touch rules.yml
```

I'm sure everyone agrees that knowing when any of your instances are down is *very* important. Therefore, I'm going to use this as our condition by using **up** metric. By evaluating this metric in the Prometheus user interface (<http://localhost:9090>), you will see that all running instances have value of 1, while all instances that are currently not running have value of 0 (we currently run only our Prometheus instance).

The screenshot shows the Prometheus web interface. At the top, there's a navigation bar with links for Prometheus, Alerts, Graph, Status, and Help. Below the navigation bar, there's a search bar with the text "up" and a "Execute" button. To the right of the search bar, it says "Try experimental React UI". Further right, it shows performance metrics: Load time: 23ms, Resolution: 14s, and Total time series: 3. Below the search bar, there are two tabs: "Graph" (which is selected) and "Console". Under the "Graph" tab, there's a timestamp selector showing "Moment". The main area displays a table of metrics:

Element	Value
up{instance="localhost:9090",job="prometheus"}	1
up{instance="localhost:9091",job="prom_middleware"}	0
up{instance="localhost:9100",job="node_exporter"}	0

After you've decided on your alerting condition, you need to specify them in **rules.yml**. Its content is going to be the following:

```
groups:  
- name: AllInstances
```

```

rules:
  - alert: InstanceDown
    # Condition for alerting
    expr: up == 0
    for: 1m
    # Annotation - additional informational labels to store more information
    annotations:
      title: 'Instance {{ $labels.instance }} down'
      description: '{{ $labels.instance }} of job {{ $labels.job }} has been down for more
    # Labels - additional labels to be attached to the alert
    labels:
      severity: 'critical'

```

To summarize, it says that if any of the instances are going to be down (`up == 0`) for one minute, then the alert will be firing. I have also included annotations and labels, which store additional information about the alerts. For those, you can use [templated variables](#) such as `{{ $labels.instance }}` which are then interpolated into specific instances (such as `localhost:9100`).

(You can read more about Prometheus alerting rules [here](#).)

Once you have `rules.yml` ready, you need to link the file to `prometheus.yml` and add alerting configuration. Your `prometheus.yml` is going to look like this:

```

global:
  # How frequently to scrape targets
  scrape_interval: 10s
  # How frequently to evaluate rules
  evaluation_interval: 10s

  # Rules and alerts are read from the specified file(s)
rule_files:
  - rules.yml

  # Alerting specifies settings related to the Alertmanager
alerting:

```

```

alertmanagers:
  - static_configs:
    - targets:
      # Alertmanager's default port is 9093
      - localhost:9093

# A list of scrape configurations that specifies a set of
# targets and parameters describing how to scrape them.

scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 5s
    static_configs:
      - targets:
        - localhost:9090
  - job_name: 'node_exporter'
    scrape_interval: 5s
    static_configs:
      - targets:
        - localhost:9100
  - job_name: 'prom_middleware'
    scrape_interval: 5s
    static_configs:
      - targets:
        - localhost:9091

```

If you've started Prometheus with `--web.enable-lifecycle` flag, you can reload configuration by sending POST request to `/-/reload` endpoint `curl -X POST http://localhost:9090/-/reload`. After that, start `prom_middleware` app (`node index.js` in `prom_middleware` folder) and `node_exporter` (`./node_exporter` in `node_exporter` folder).

Once you do this, anytime you want to create an alert, you can just stop the `node_exporter` or `prom_middleware` app.

Step 2: Set up Alertmanager

Create an **alert_manager** subfolder in the Prometheus folder, `mkdir alert_manager`. To this folder, you'll then download and extract Alertmanager from the [Prometheus website](#), and without any modifications to the **alertmanager.yml**, you'll run `./alertmanager --config.file=alertmanager.yml` and open `localhost:9093`.

Depending on whether or not you have any active alerts, Alertmanager should be properly set up and look something like the image below. To see the annotations that you added in the step above, you'd click on the **+Info** button.

The screenshot shows the Alertmanager interface at `localhost:9093/#/alerts`. At the top, there's a navigation bar with links for Alertmanager, Alerts, Silences, Status, and Help, and a "New Silence" button. Below the navigation is a search bar and filter options for "Filter" and "Group". On the right, there are buttons for "Receiver: All", "Silenced", and "Inhibited". A "Custom matcher, e.g. `env="production"`" input field is present. The main area displays two alerts under the group "alername='InstanceDown'". The first alert is from "14:06:39, 2020-01-28 (UTC)" and the second is from "14:07:39, 2020-01-28 (UTC)". Each alert has "Info", "Source", and "Silence" buttons. Below each alert, there are dropdown menus for "instance" and "job".

With all of that completed, we can now look at the different ways of utilizing Alertmanager and sending out alert notifications.

How to set up Slack alerts

If you want to receive notifications via Slack, you should be part of a Slack workspace. If you are currently not a part of any Slack workspace, or you want to test this out in separate workspace, you can quickly create one [here](#).

To set up alerting in your Slack workspace, you're going to need a **Slack API URL**. Go to **Slack → Administration → Manage apps**.

The screenshot shows the Slack interface. On the left is the sidebar with user information (Ivana), workspace details (IP, iprometheusal-srm9503.slack.com), and administration options (Manage members, Workspace settings, Manage apps). The main area displays a banner for "Bring your team into Slack" with three people interacting around a screen. Below the banner is a message: "Slack is better with teammates – invite them to start collaborating." A green "Add People" button is visible.

In the Manage apps directory, search for **Incoming WebHooks** and add it to your Slack workspace.

The screenshot shows the Slack App Directory. A search bar at the top contains the text "hooks". Below the search bar, a list of apps is displayed:

- Incoming WebHooks** - Send data into Slack in real-time.
- Outgoing WebHooks** - Get data out of Slack in real-time.
- Status Hook** - Create web hooks to control your status.
- Cronhooks** - Cronhooks enables you to schedule on ti...
- Drafted - The Referral Network** - Drafted supercharges employee referrals...

A green "Add to Slack" button is located at the bottom of the search results.

Next, specify in which channel you'd like to receive notifications from Alertmanager. (I've created #monitoring-infrastructure channel.) After you confirm and add Incoming WebHooks integration, webhook URL (which is your Slack API URL) is displayed. Copy it.

to our APIs.

close

Setup Instructions

We'll guide you through the steps necessary to configure an Incoming Webhook so you can start sending data to Slack.

Webhook URL

<https://hooks.slack.com/services/TSUJTM1HQ/BT7JT5RFS/5eZMpbDkk8wk2VUFQB6Rhuz>

Sending Messages

You have two options for sending data to the Webhook URL above:

- Send a JSON string as the `payload` parameter in a POST request

Then you need to modify the `alertmanager.yml` file. First, open subfolder `alert_manager` in your code editor and fill out your `alertmanager.yml` based on the template below. Use the url that you have just copied as `slack_api_url`.

```
global:  
  resolve_timeout: 1m  
  slack_api_url: 'https://hooks.slack.com/services/TSUJTM1HQ/BT7JT5RFS/5eZMpbDkk8wk2VUFQB6Rhuz'  
  
route:  
  receiver: 'slack-notifications'  
  
receivers:  
  - name: 'slack-notifications'  
    slack_configs:  
      - channel: '#monitoring-instances'  
        send_resolved: true
```

Reload configuration by sending POST request to `/-/reload` endpoint `curl -X POST http://localhost:9093/-/reload`. In a couple of minutes (after you stop at least one of your instances), you should be receiving your alert notifications through Slack, like this:

#monitoring-instances

You created this channel today. This is the very beginning of the #monitoring-instances channel.

[Set a description](#) [+ Add an app](#) [Add people to this channel](#)

Today

 **Ivana** 5:51 PM
added an integration to this channel: [incoming-webhook](#)

 **AlertManager** APP 6:01 PM
[FIRING:2] (InstanceDown critical)

If you would like to improve your notifications and make them look nicer, you can use the template below, or use [this tool](#) and create your own.

```
global:  
  
  resolve_timeout: 1m  
  slack_api_url: 'https://hooks.slack.com/services/TSUJTM1HQ/BT7JT5RFS/5eZMpbDkK8wk2VUFQB6  
  
route:  
  receiver: 'slack-notifications'  
  
receivers:  
- name: 'slack-notifications'  
  slack_configs:  
    - channel: '#monitoring-instances'  
      send_resolved: true  
      icon_url: https://avatars3.githubusercontent.com/u/3380462  
      title: |-  
        {{ .Status | toUpper }}{{ if eq .Status "firing" }}:{{ .Alerts.Firing | len }}{{ end }}  
        {{- if gt (len .CommonLabels) (len .GroupLabels) -}}}  
        {{" "}}(  
        {{- with .CommonLabels.Remove .GroupLabels.Names }}  
        {{- range $index, $label := .SortedPairs -}}}  
          {{ if $index }}, {{ end }}  
          {{- $label.Name }}={{ $label.Value -}}"  
        {{- end }}  
        {{- end -}}  
    )
```

```

{{- end }}

text: >-

{{ range .Alerts -}>

*Alert:* {{ .Annotations.title }}{{ if .Labels.severity }} - `{{ .Labels.severity }}`

*Description:* {{ .Annotations.description }}

*Details:*

{{ range .Labels.SortedPairs }} • *{{ .Name }}:* `{{ .Value }}`{{ end }}{{ end }}

```

And this is the final result:

AlertManager APP 6:06 PM

[RESOLVED] InstanceDown for (severity="critical")

Alert: Instance localhost:9100 down - **critical**

Description: localhost:9100 of job node_exporter has been down for more than 1 minute.

Details:

- **alertname:** `InstanceDown`
- **instance:** `localhost:9100`
- **job:** `node_exporter`
- **severity:** `critical`

Alert: Instance localhost:9091 down - **critical**

Description: localhost:9091 of job prom_middleware has been down for more than 1 minute.

Details:

- **alertname:** `InstanceDown`
- **instance:** `localhost:9091`
- **job:** `prom_middleware`
- **severity:** `critical`

[Show less](#)

How to set up PagerDuty alerts

PagerDuty is one of the most well-known incident response platforms for IT departments. To set up alerting through PagerDuty, you need to create an account there. (PagerDuty is a paid service, but you can always do a 14-day free trial.) Once you're logged in, go to **Configuration → Services → + New Service**.

Your PagerDuty trial expires in 14 days. [Upgrade Now »](#)

Incidents on All Team

Your open incidents
0 triggered
0 acknowledged

[Acknowledge](#) [Reassign](#)

Open	Triggered	Acknowledged
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Status	Urgency	Time

Schedules Services Event Rules Escalation Policies Response Plays Business Services Impact Metrics Users Teams API Access Extensions Incident Priorities Account Settings

All open incidents
0 triggered
0 acknowledged

All Incidents Assigned to me All

Created Service Assigned To

Next Steps: Your Team
You are all alone being on-call all the time.
Get some backup, invite your team:
[Add your colleagues](#)

Then share on-call responsibilities:
[Create an on-call schedule](#)

Next Steps: Your Tools
Add all your tools to begin monitoring your systems today:
[Add new services](#)

Choose Prometheus from the Integration types list and give the service a name. I decided to call mine Prometheus Alertmanager. (You can also customize the incident settings, but I went with the default setup.) Then click save.

Add a Service

A service may represent an application, component or team you wish to open incidents against.

General Settings

Name	Prometheus Alertmanager
Description	Add a description for this service (optional)

Integration Settings

Integrations can open and resolve incidents. Once a service is created, it can have multiple integrations.

Integration Type	<input checked="" type="radio"/> Prometheus
We integrate with dozens of monitoring systems. This may involve configuration steps in your monitoring tool.	
<input type="radio"/> Integrate via email	
If your monitoring tool can send email, it can integrate with PagerDuty using a custom email address.	
<input type="radio"/> Use our API directly	
If you're writing your own integration, use our Events API. More information is in our developer documentation.	
Events API v2	
<input type="radio"/> Don't use an integration	
If you only want incidents to be manually created. You can always add additional integrations later.	
Integration Name	Prometheus



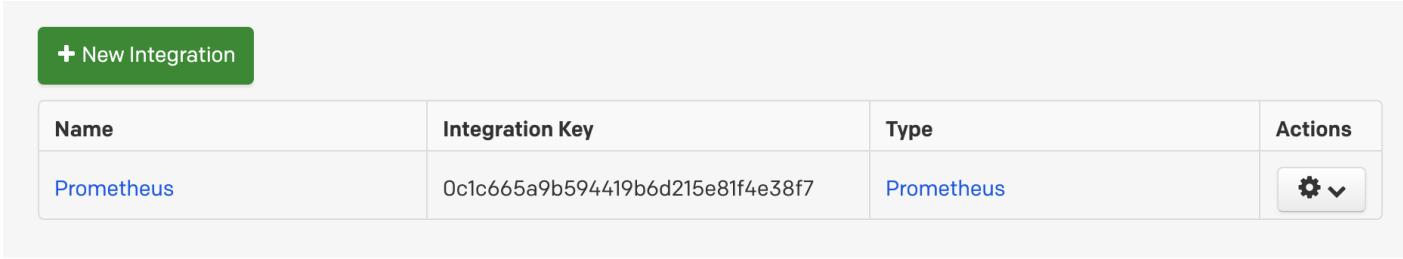
Prometheus is an open-source systems monitoring and alerting toolkit. It features a multi-dimensional data model, a flexible query language to leverage this dimensionality, has no reliance on distributed storage, time series collection happens via a pull model over HTTP, pushing time series is supported via an intermediary gateway, targets are discovered via service discovery or static configuration, and has multiple modes of graphing and dashboarding support.

Incident Settings

Escalation Policy	Default
-------------------	---------

The policy specifies who will be **assigned responsibility for resolution** when this service is triggered.

The Integration Key will be displayed. Copy the key.



Alertmanager Integrations			
Name	Integration Key	Type	Actions
Prometheus	0c1c665a9b594419b6d215e81f4e38f7	Prometheus	

You'll need to update the content of your `alertmanager.yml`. It should look like the example below, but use your own service_key (integration key from PagerDuty). `Pagerduty_url` should stay the same and should be set to <https://events.pagerduty.com/v2/enqueue>. Save and restart the Alertmanager.

```
global:  
  resolve_timeout: 1m  
  pagerduty_url: 'https://events.pagerduty.com/v2/enqueue'  
  
route:  
  receiver: 'pagerduty-notifications'  
  
receivers:  
  - name: 'pagerduty-notifications'  
    pagerduty_configs:  
      - service_key: 0c1cc665a594419b6d215e81f4e38f7  
        send_resolved: true
```

Stop one of your instances. After a couple of minutes, alert notifications should be displayed in PagerDuty.

The screenshot shows a list of incidents in PagerDuty. At the top, there are filters: 'Open' (selected), 'Triggered', 'Acknowledged', 'Resolved', and 'Any Status'. To the right are dropdowns for 'All Incidents' and 'Assigned to me' (selected). Below the filters is a table header row with columns: 'Status', 'Urgency ▾', 'Title', 'Created', 'Service', and 'Assigned To'. A single incident is listed: 'Triggered' (Status), 'High' (Urgency), '[FIRING-2] (InstanceDown critical)' (Title), 'at 1:26 PM' (Created), 'Prometheus Alertmanager' (Service), and 'Monitoring Instances' (Assigned To). At the bottom right are pagination controls: 'Per Page: 25', '1-1', and navigation arrows.

In PagerDuty user settings, you can decide how you'd like to be notified: email and/or phone call. I chose both and they each worked successfully.

How to set up Gmail alerts

If you prefer to have your notifications come directly through an email service, the setup is even easier. Alertmanager can simply pass along messages to your provider — in this case, I used Gmail — which then sends them on your behalf.

It isn't recommended that you use your personal password for this, so you should create an [App Password](#). To do that, go to **Account Settings → Security → Signing in to Google → App password** (if you don't see App password as an option, you probably haven't set up 2-Step Verification and will need to do that first). Copy the newly created password.

← → C myaccount.google.com/u/2/security

Google Account Search Google Account ?

Security

Home Personal info Data & personalization Security People & sharing Payments & subscriptions

Settings and recommendations to help you keep your account secure

Security issues found

Protect your account now by resolving these issues

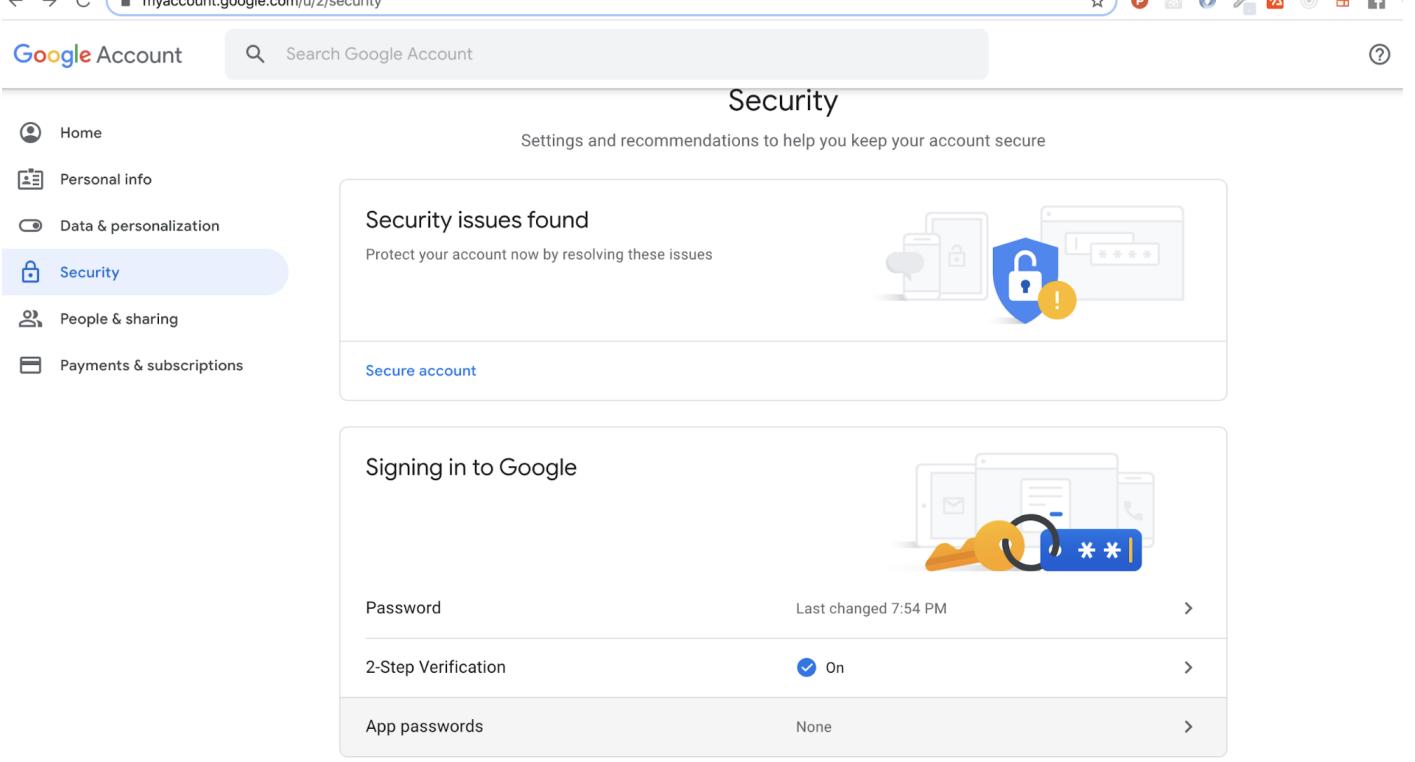
Secure account

Signing in to Google

Password Last changed 7:54 PM >

2-Step Verification On >

App passwords None >



You'll need to update the content of your **alertmanager.yml** again. The content should look similar to the example below. Don't forget to replace the email address with your own email address, and the password with your new app password.

```
global:  
  resolve_timeout: 1m  
  
route:  
  receiver: 'gmail-notifications'  
  
receivers:  
  - name: 'gmail-notifications'  
    email_configs:  
      - to: monitoringinstances@gmail.com  
        from: monitoringinstances@gmail.com  
        smarthost: smtp.gmail.com:587  
        auth_username: monitoringinstances@gmail.com  
        auth_identity: monitoringinstances@gmail.com  
        auth_password: password  
        send_resolved: true
```

Once again, after a couple of minutes (after you stop at least one of your instances), alert notifications should be sent to your Gmail.

<input type="checkbox"/>	☆	me	[RESOLVED] (InstanceDown localhost:9091 prom_middleware critical) - 1 alert for View in Alert...	Jan 28
<input type="checkbox"/>	☆	me	[FIRING:1] (InstanceDown localhost:9100 node_exporter critical) - 1 alert for View in AlertMan...	Jan 28
<input type="checkbox"/>	☆	me	[FIRING:1] (InstanceDown localhost:9091 prom_middleware critical) - 1 alert for View in Alert...	8:23 PM
<input type="checkbox"/>	☆	me	[FIRING:1] (InstanceDown localhost:9091 prom_middleware critical) - 1 alert for View in Alert...	8:18 PM
<input type="checkbox"/>	☆	me	[FIRING:1] (InstanceDown localhost:9091 prom_middleware critical) - 1 alert for View in Alert...	8:12 PM

And that's it!

Prometheus Alerting

Up next

Grafana Cloud Hosted Prometheus supports long-term storage, receives metrics from multiple clusters, and is designed to support high availability. And it's simple to connect Prometheus-Ksonnet



Connecting Prometheus-Ksonnet to Grafana Cloud

3 min | 24 Feb 2020

JOIN US FOR THE FIRST-EVER

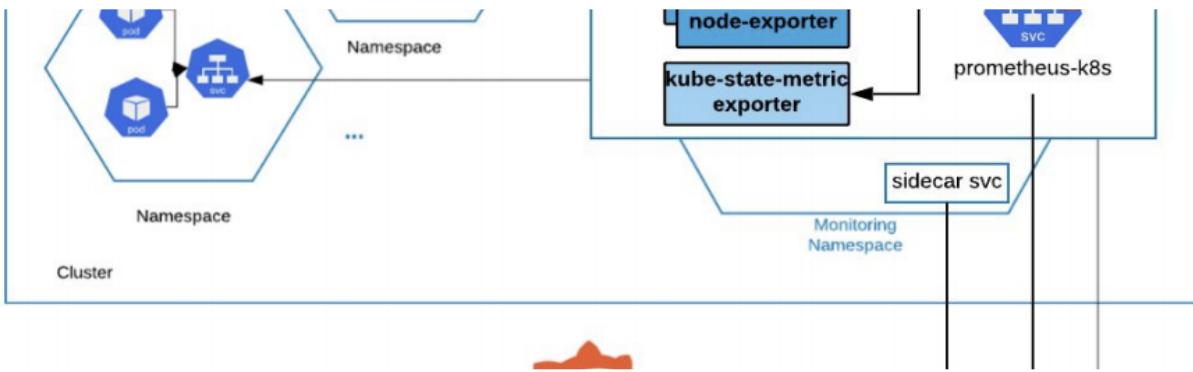


Grafana Labs user group meetup: San Francisco, Feb. 27.



Don't Miss the First Bay Area Grafana Labs User Group Meetup Feb. 27

1 min | 13 Feb 2020



Tinder & Grafana: A Love Story in Metrics and Monitoring

6 min | 26 Mar 2019



Keep up with us.
Product developments and observability innovations.

Email

Subscribe

Grafana

[Overview](#)
[Deployment options](#)
[Plugins](#)
[Dashboards](#)

Company

[Our mission](#)
[The team](#)
[Press](#)
[Careers](#)
[Events](#)
[Partnerships](#)
[Contact](#)

Products

[Grafana Cloud](#)
 [Grafana Cloud Status](#)
[Grafana Enterprise Stack](#)
[Grafana OnCall](#)

Open Source

[Grafana](#)
[Prometheus](#)
[Grafana Loki](#)
[Grafana Mimir](#)
[Grafana OnCall](#)
[Grafana Tempo](#)
[Metricstank](#)
[Graphite](#)

[Grafana k6](#)

[Tanka](#)

[OpenTelemetry](#)

 [GitHub](#)

Learn

[Grafana Labs Blog](#)

[Documentation](#)

[Downloads](#)

[Community](#)

[Grafana ObservabilityCON](#)

[GrafanaCON](#)

[Successes](#)

[Training](#)

[Videos](#)

[Grafana Cloud Status](#) [Sitemap](#) [Legal and Security](#) [Terms of Service](#) [Trademark Policy](#)

Copyright 2022 © Grafana Labs