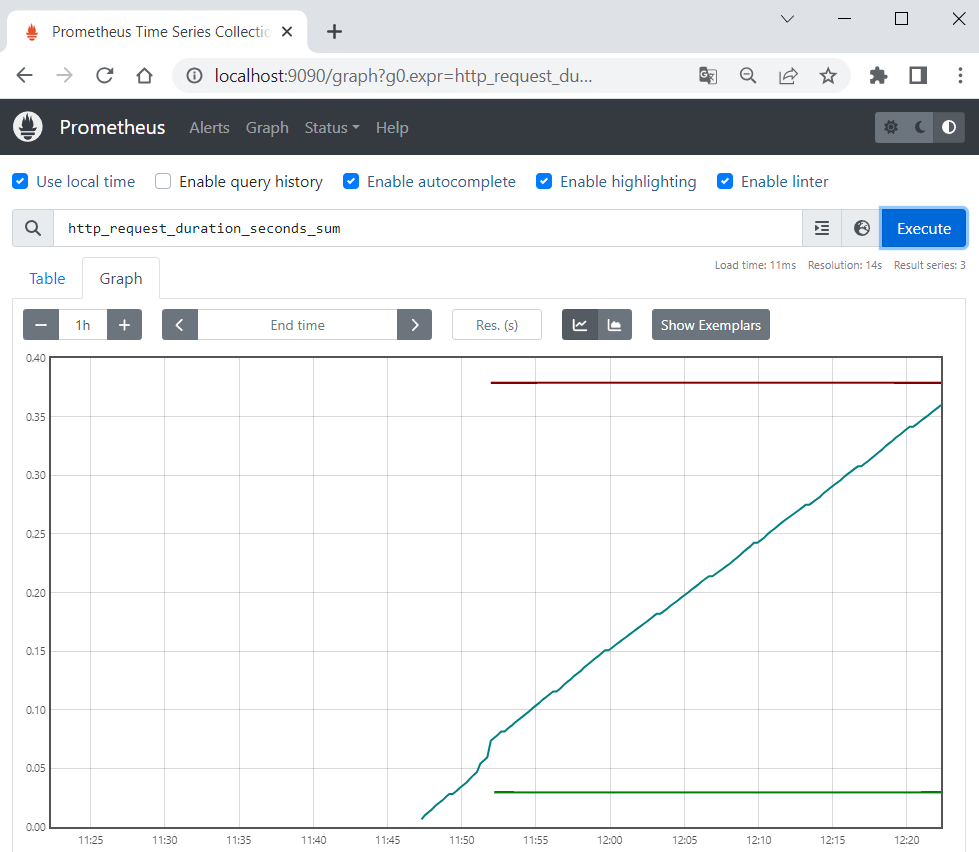
# Exercise: App Monitoring

Exercises for the "Containers and Clouds" course @ SoftUni

### Monitor the "Contact Book" Node.js App with Prometheus

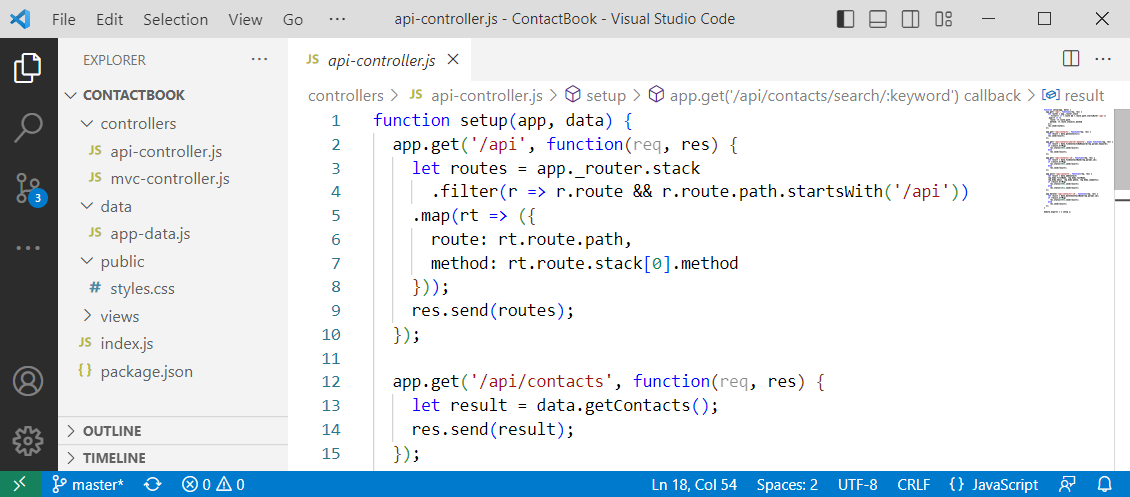
We have the Node.js "Contact Book" **app** in the **resources**. We aim to **monitor it using Prometheus**, so we need its **metrics**. In this case, we will **instrument the app** to **expose the metrics** we want. And then we will **configure Prometheus** to **display these metrics**.



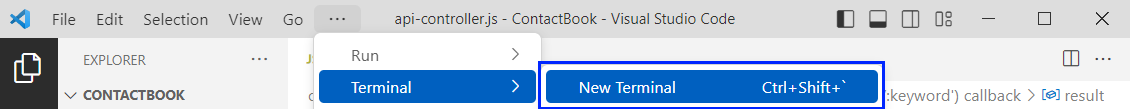
#### Step 1: Examine the App

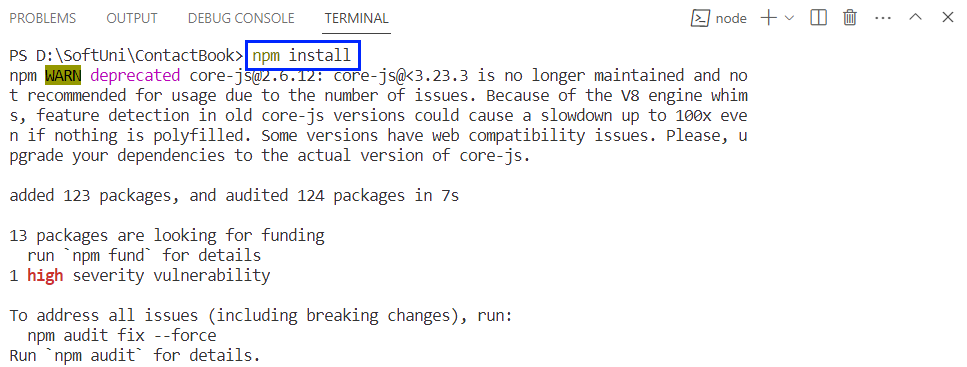
We have the "Contact Book" **Node.js app**, which holds a **searchable list of contacts**. You have pages to **list all contacts** (/contacts), **view a single contact** (/contacts/:id), **search for a contact** (/contacts/search/:keyword) and **add a new contact** (/contacts/create).

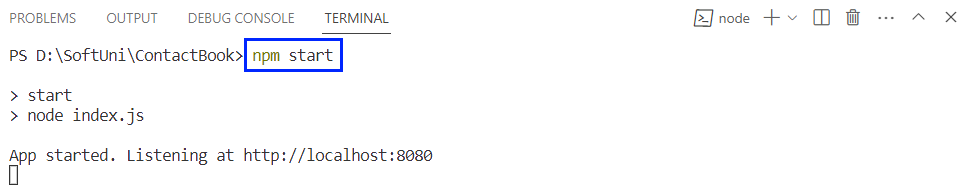
Open the **project in Visual Studio Code** to **examine its files**:



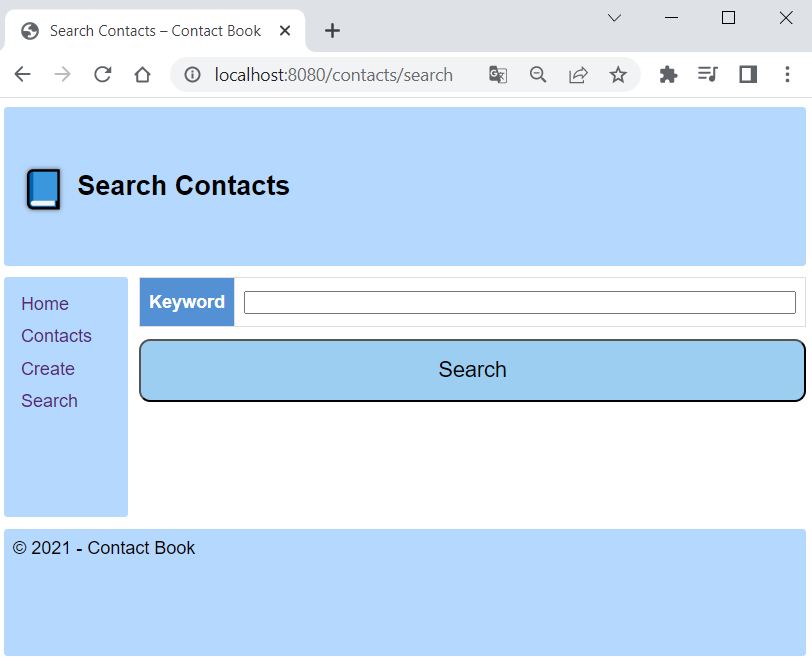
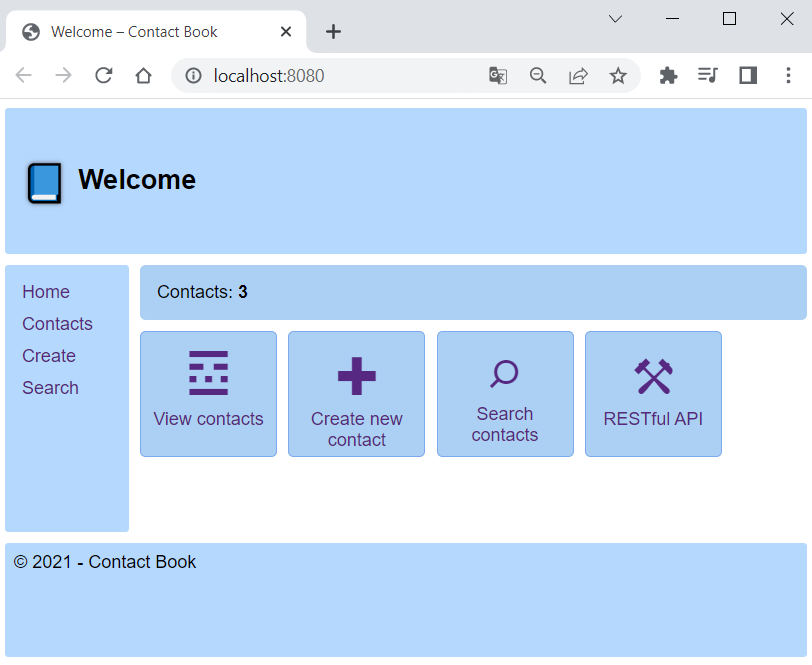
Open a **terminal** and **execute** the "npm install" and "npm start" **commands** to **run the app**:

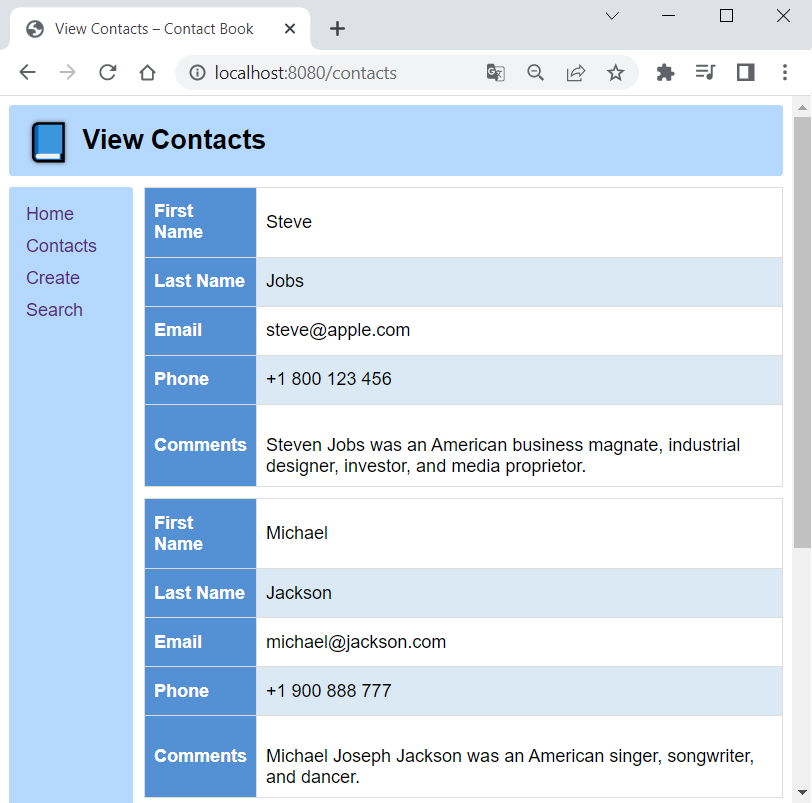
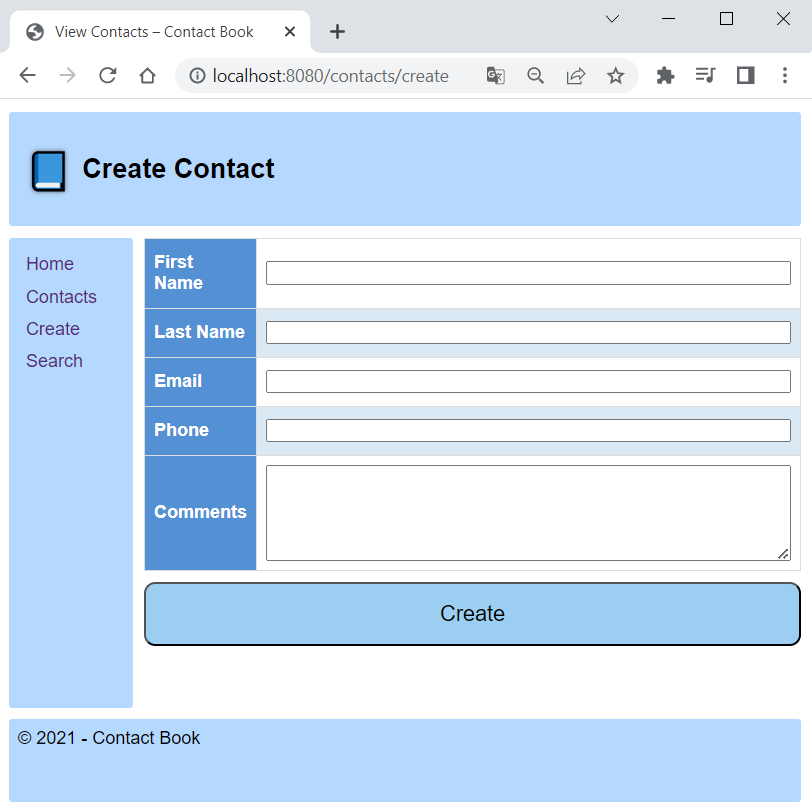
🡪





Look at the **app pages** on [http://localhost:8080](http://localhost:8080/):



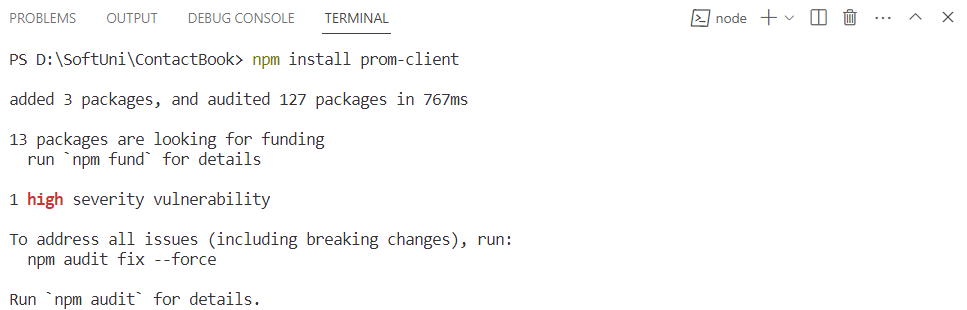
Let's now see how to **modify the app code** to **export app metrics** for **Prometheus**.

#### Step 2: Export Node.js App Metrics

To **make app metrics readable for Prometheus**, we should **install an additional client library** **for Node.js** and then **modify the code** to **define and export the metrics** we want.

##### Install Prom-Client

**Stop the app** with [Ctrl] + [C] in the **terminal**. Then, we should **install the** prom-client **package**, which is the **Prometheus client** **for Node.js** that supports histogram, summaries, gauges and counters. Do it with the following **command**:



You can see that the prom-client **package** is added to the package.json **file** with **project dependencies**:

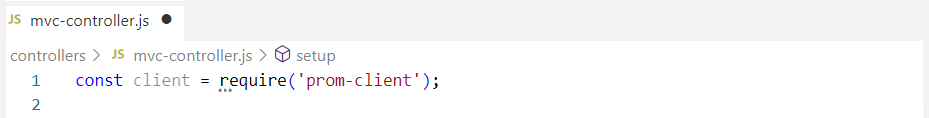


Because of this, you **won't need to install the package** separately from the others next time.

##### Export Default Metrics

Now we will **modify our code** to **collect the default app metrics** together with **some custom** **ones** and **expose them** on the /metrics **endpoint**.

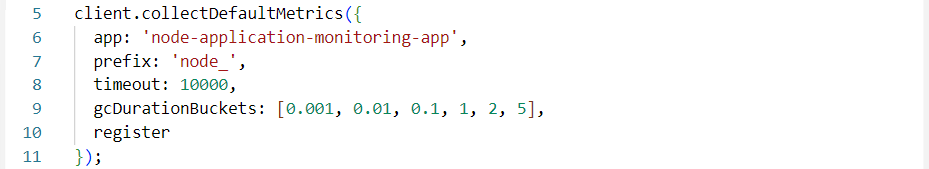
To do this, navigate to mvc-controller.js **file** where the main app routing is and **include the** prom-client **module**, as we will need it:



Then, **create a registry** to register the metrics:

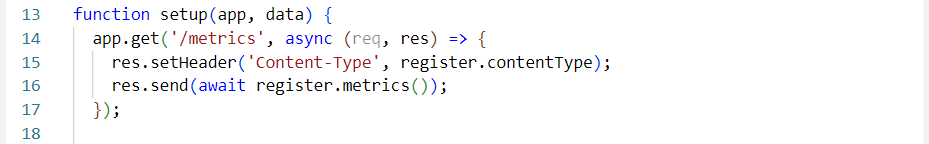


Use the collectDefaultMetrics() **function** from the **imported module** to **collect and register default metrics** for **monitoring the Node.js application**, for example CPU usage, memory usage, event loop latency, and garbage collection duration:

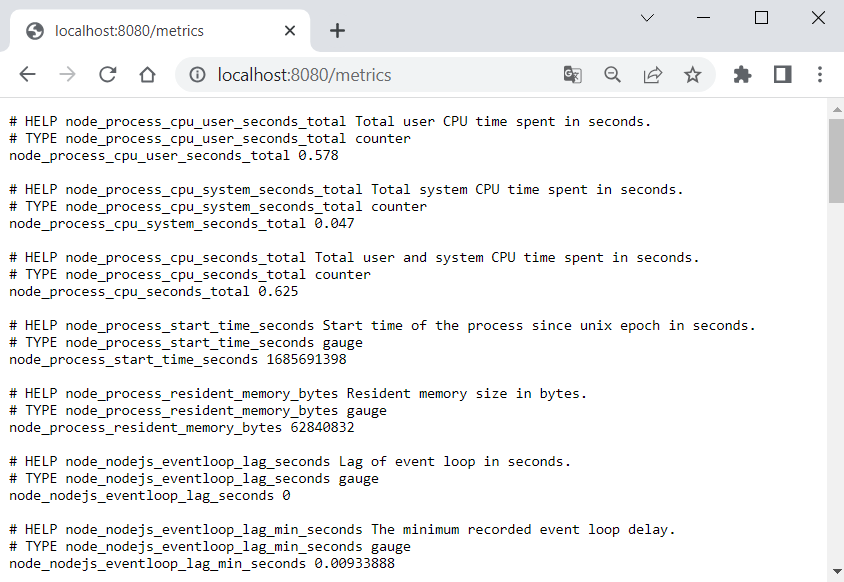


This configuration sets **default metric names** to start with the "**\_node**" prefix, the **timeout** to 10000ms, the **buckets** **for the default metric that measures garbage collection** (GC) **durations** (values represent the upper bounds of each bucket) and the **registry** that we created to be used.

These are the **default metrics** we will export. Now, in the setup() **function**, set up an **HTTP GET route** for the /metrics **endpoint**, which should **return the collected app metrics** **as response**:



Before we add some **custom metrics**, let's see how **default metrics are showed**. Save the changes and **start the app** again. Then, **navigate to** <http://localhost:8080/metrics> in the browser:

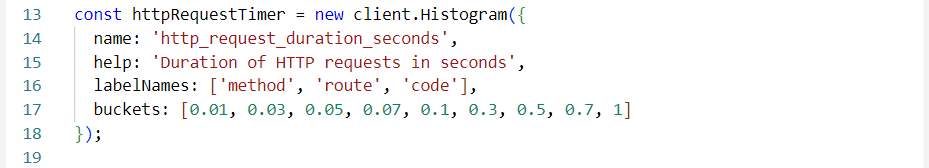


Now let's **add some more metrics**.

##### Export Custom Metrics

The **custom metrics** we shall export are about the **duration of HTTP requests** **to different endpoints in seconds**. They will be saved in a **histogram** with **buckets from 0.01 to 1 seconds** and will keep **request method**, **route** and **status code**.

**Add the following code** to **create the histogram metric** (before the setup() function):



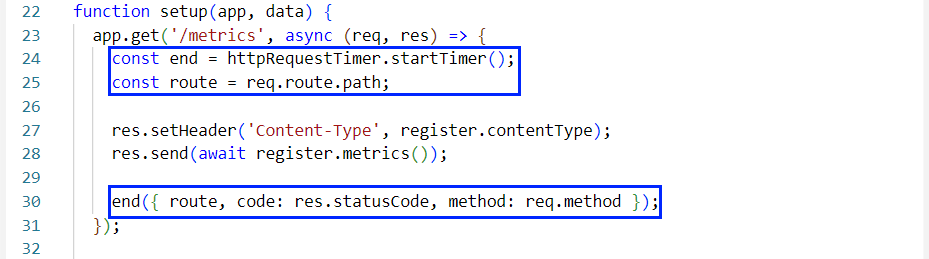
Then we should **register the metric**:



Now, for **each of the routes**, we should:

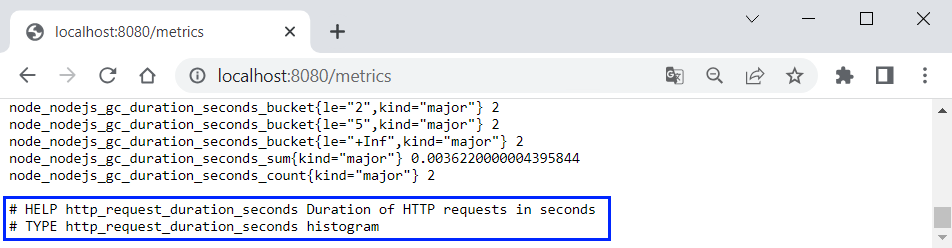
* Start an **HTTP request timer**, saving a **reference** to the returned method
* Save **reference** **to the path** so we can record it when ending the timer
* And finally **end the timer** and **add labels**

In this way, the **HTTP request data and duration** will be recorded. Do it for the /metrics **endpoint** like this:

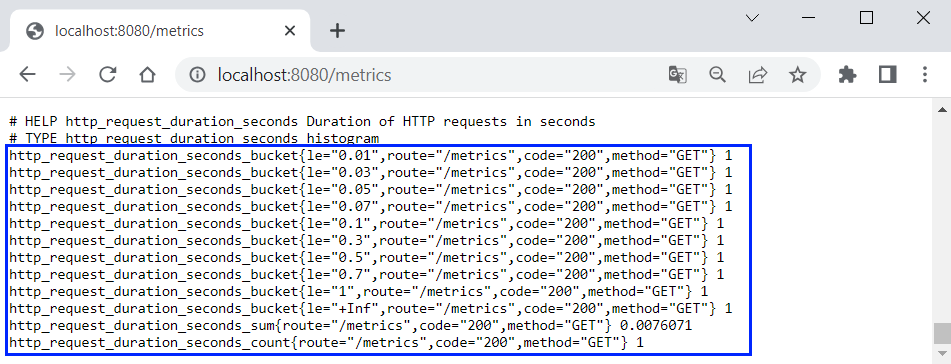


Do it for the **rest of the endpoint methods** in the same way by **adding the above three lines**. When ready, **run the app** again.

When you first **access** /metrics, you will see the **new metrics** **at the bottom**:

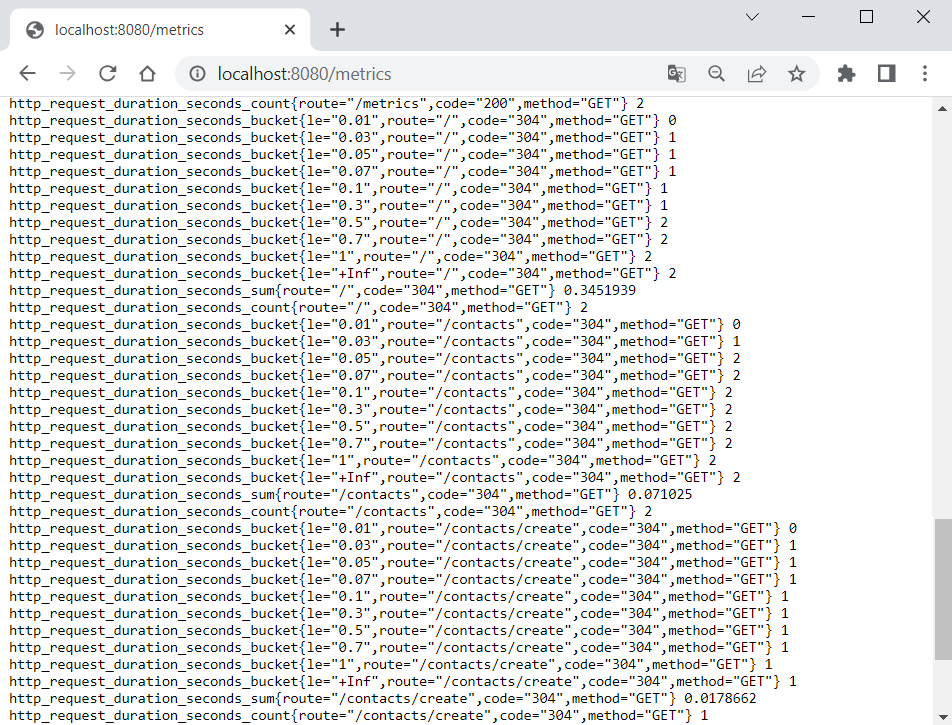


However, you still have **no metric values**. You should **refresh the page**, so that the metrics for the **previous HTTP request** to /metrics are displayed:



As you can see, the **first HTTP request** **to** /metrics took about **0.0076 seconds**, which is **less than 0.01** and that's why it falls into **each of the buckets**.

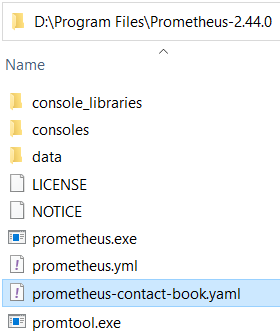
If you **access the other app endpoints**, you will get even **more metric data**:



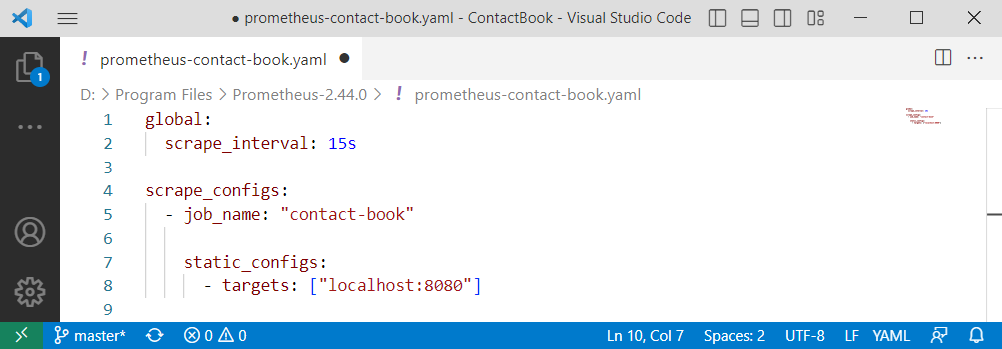
You can use the **browser inspector** in the browser to **compare the HTTP request times** shown here and there – they should be pretty close as values.

#### Step 3: Condifure and Run Prometheus

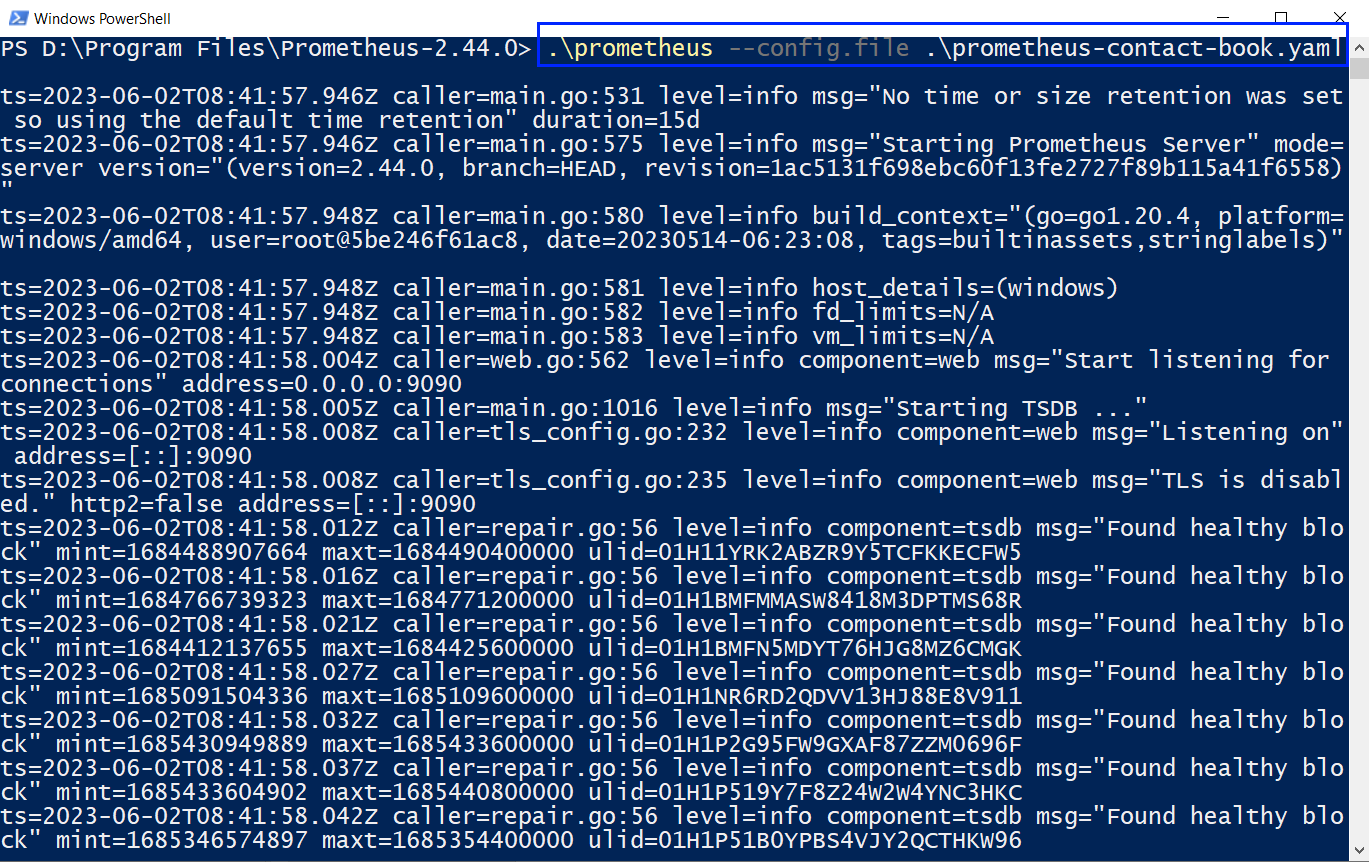
Go to the **Prometheus installation directory** where our binary files are and **create a YAML file** where we will **write the configuration** for **monitoring the** "ContactBook" **app**:



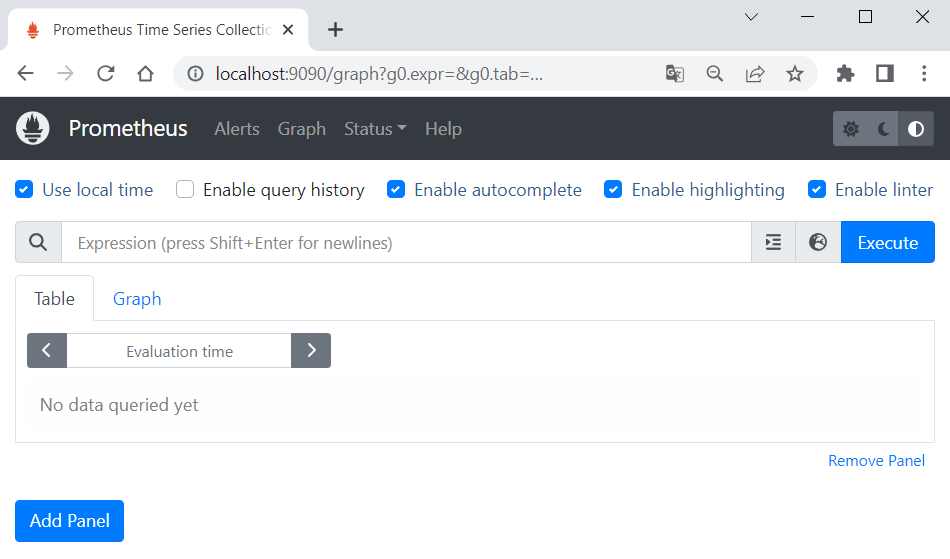
In the **Prometheus configuration file**, we should **define а single job** to **monitor our app on** <http://localhost:8080> and **scrape target metrics** on **every 15 seconds**:



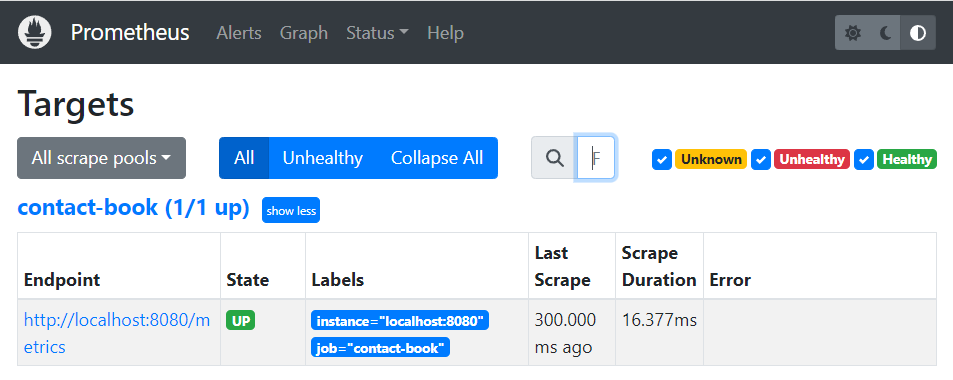
**Save the file** and **open a terminal**. Navigate to the **Prometheus installation directory** and **run Prometheus** with this **configuration file**:



**Prometheus server** should be **available on** <http://localhost:9090> by default:

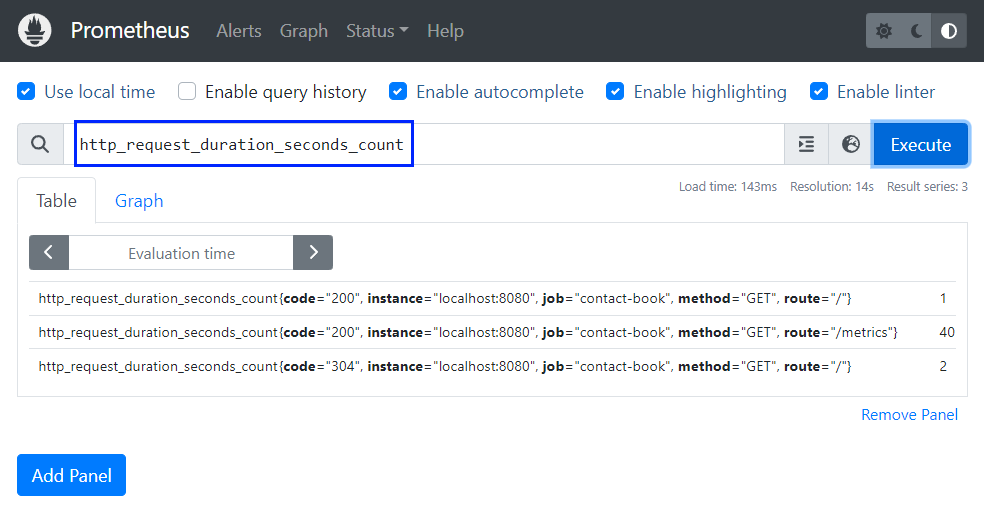


You can navigate to [Status] 🡪 [Targets] to see that **Prometheus connects** to the **configured target app** successfully:

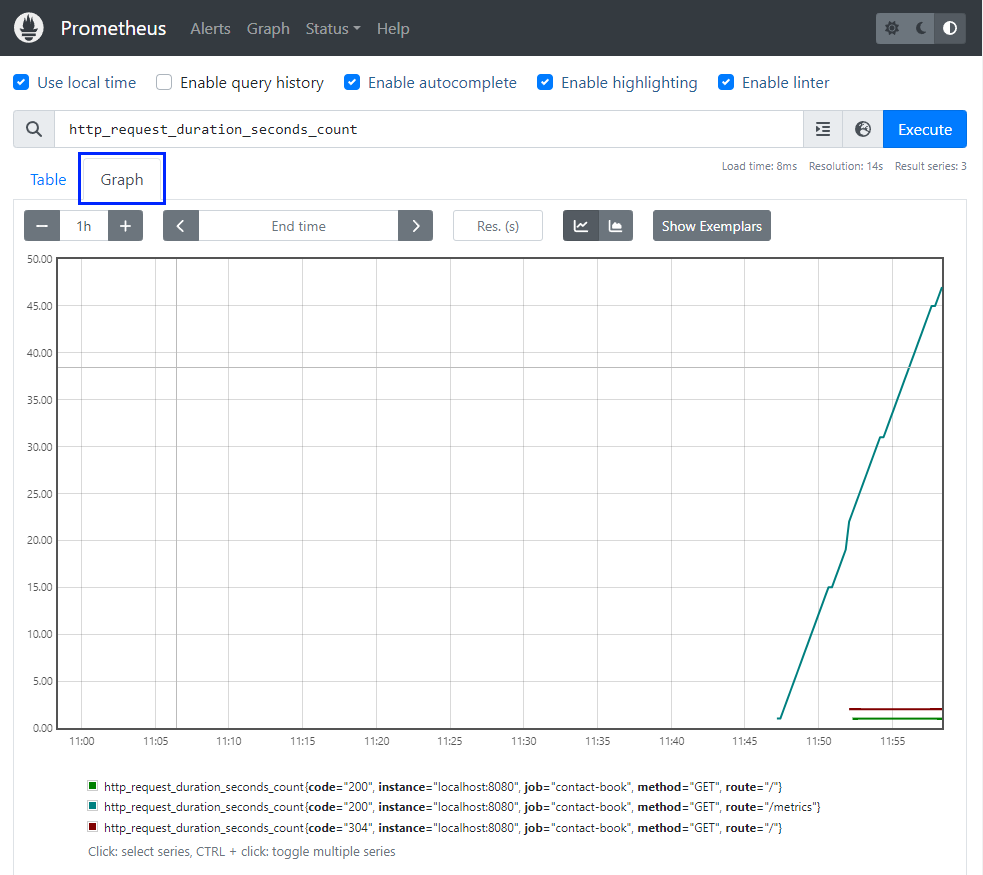


**NOTE:** the "Contact Book" **app should be running** to expose metrics.

Now you can go back to the [Graph] **page** and **display some of the metric values**, using an **expression**. For example, let's see the **count of all different HTTP requests**:



Switch to [Graph] to **look at a graph for the metric**:



You can **examine more metrics** you want. When **metrics change**, click on [Execute] to **load the changed** **graph**.

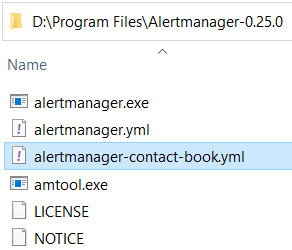
As we know how to **work with Prometheus**, let's see how to **add** Alertmanager to **manage alerts** and **send notifications**.

### Manage "Contact Book" App Alerts with Alertmanager

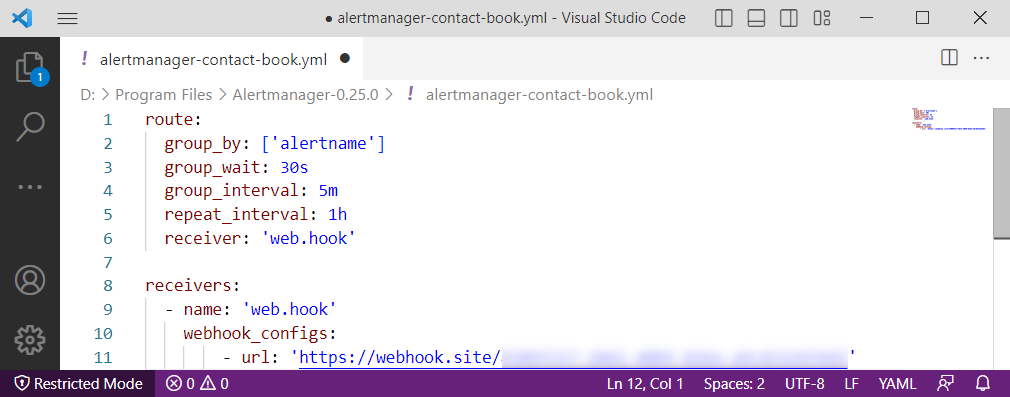
In this task, we will manage Prometheus alerts with Alertmanager and send them to Webhook.site to keep them. Our aim is to fire alerts when any page has been accessed more than 3 times during a 5-minute period.

#### Step 1: Configure and Run Alertmanager

Let's first see how to write a configuration file for Alertmanager to handle alerts. First, create a YAML file in the Alertmanager installation directory:



Open the file in an editor and write the sample configuration below:

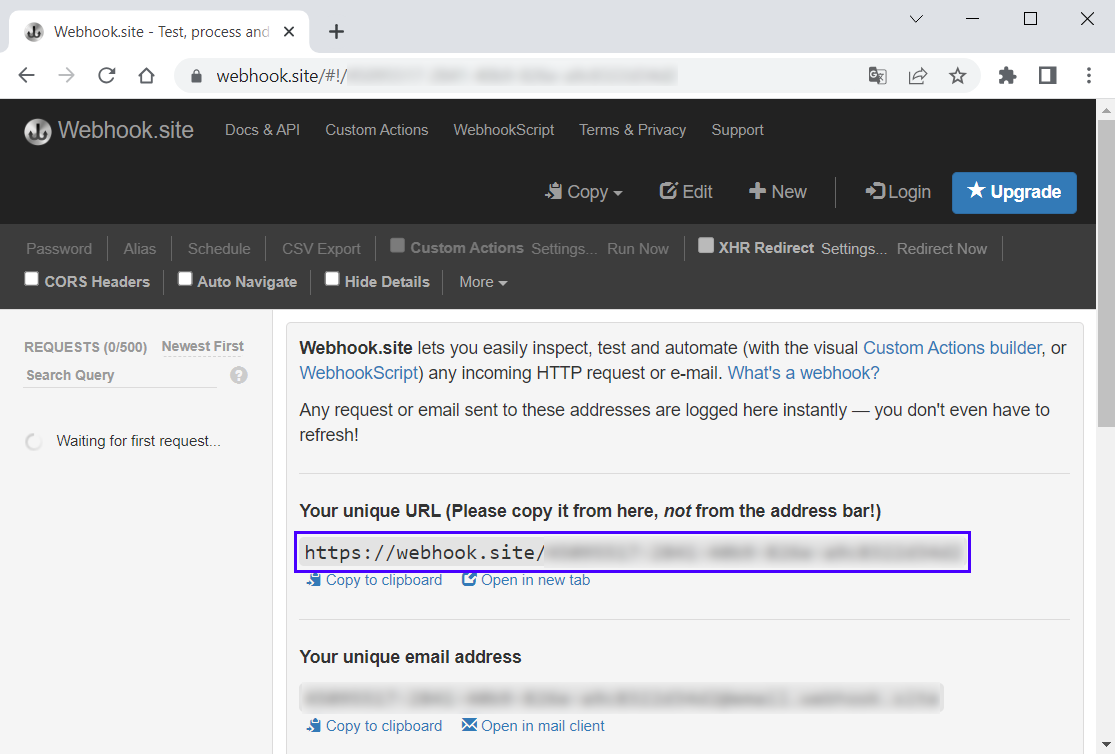


This configuration:

* Groups alerts by their name
* Sets alerts to be grouped together for a period of 30 seconds before being sent
* Sets notifications for unresolved alert groups to be sent every 5 minutes
* Sets notifications for unresolved alerts to be repeated every 1 hour
* Defines notification receiver to be "web.hook"
* Configures receiver

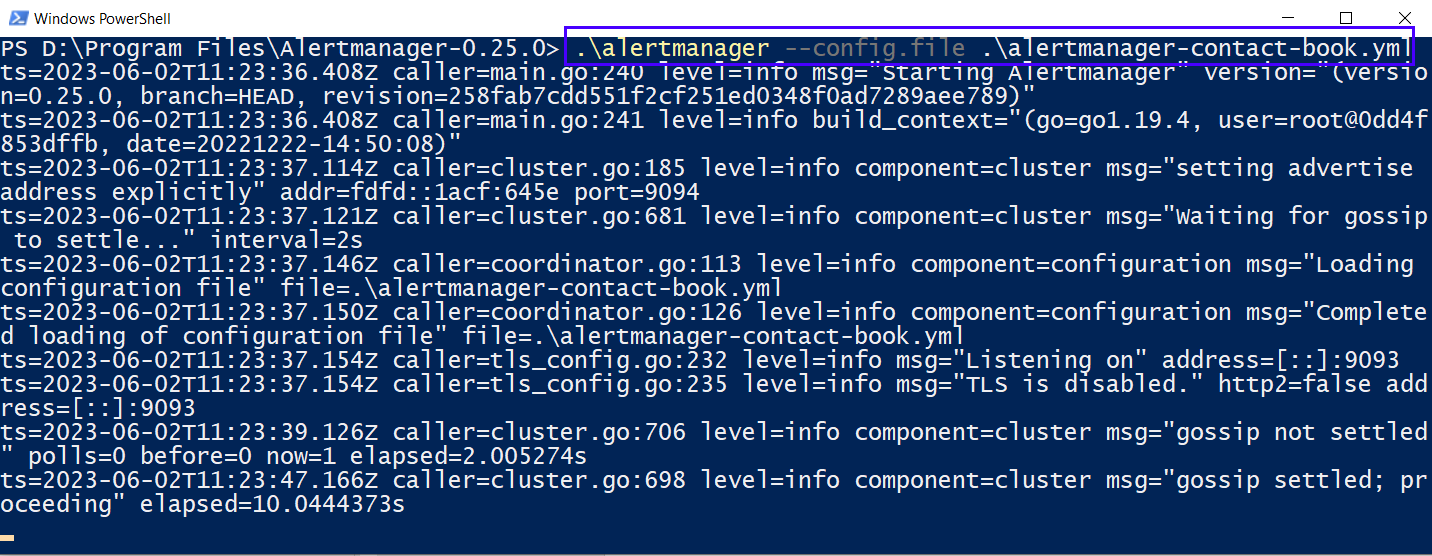
As you can see, the configuration contains a Webhook.site URL, which is unique. Webhook.site allows us to create temporary endpoints (webhooks) and capture the incoming requests sent to those endpoints, e.g., our Prometheus notifications.

To get your URL, navigate to Webhook.site and copy the provided URL, without closing the browser tab after this:

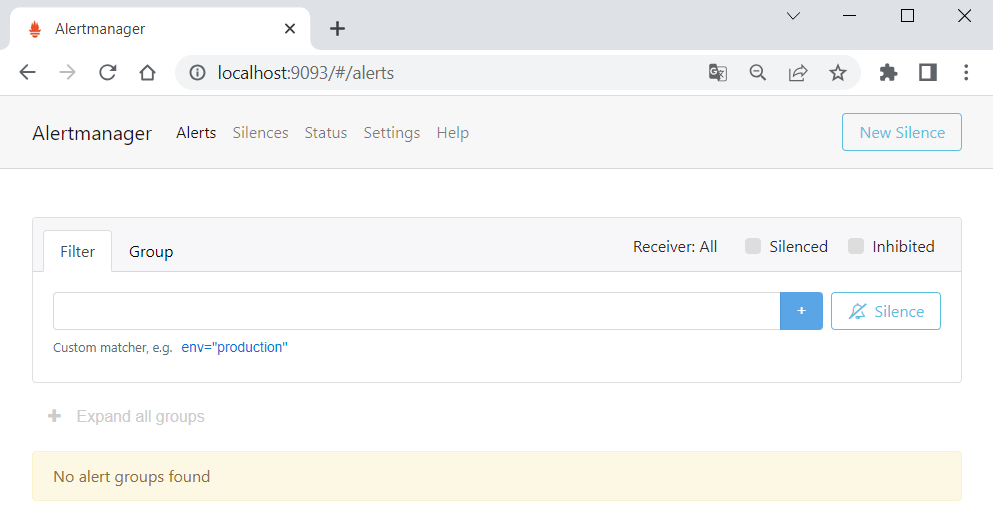


Add the URL to your configuration file and save it.

Next, run Alertmanager with the configuration file:



Go to <http://localhost:9093> in the browser and you should see that Alertmanager is up and working:

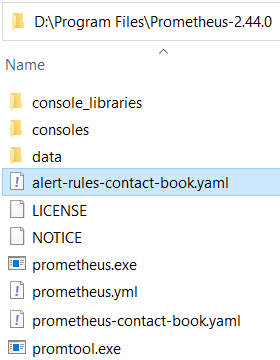


Now let's configure Prometheus to work with Alertmanager.

#### Step 2: Configure and Run Prometheus

We should do 2 things to make Prometheus send alerts to Alertmanager – first, create a YAML file with rules for firing an alert and, second, modify the Prometheus configuration file to use the rules and send alerts to Alertmanager.

Create a new YAML file in the Prometheus installation directory, which will define alert rules:

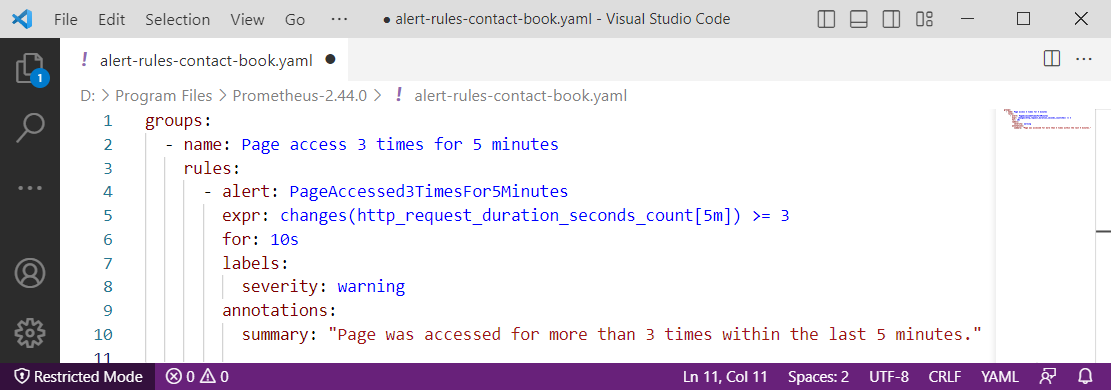


As we said earlier, we will fire an alert when an endpoint is accessed more than 3 times for 5 minutes. We will measure that using the http\_request\_duration\_seconds\_count metric – if its value has changed more than 3 times for the last 5 minutes. We shall have the following expression:

changes(http\_request\_duration\_seconds\_count[5m]) >= 3

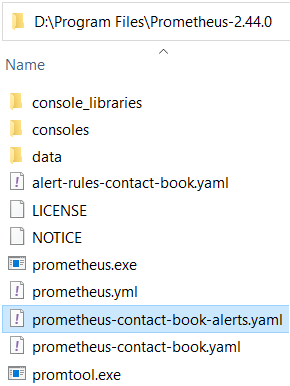
Note that in our case we count how many times the requests count has changed on data scrape (on every 15 seconds), not how many times the count has changed generally.

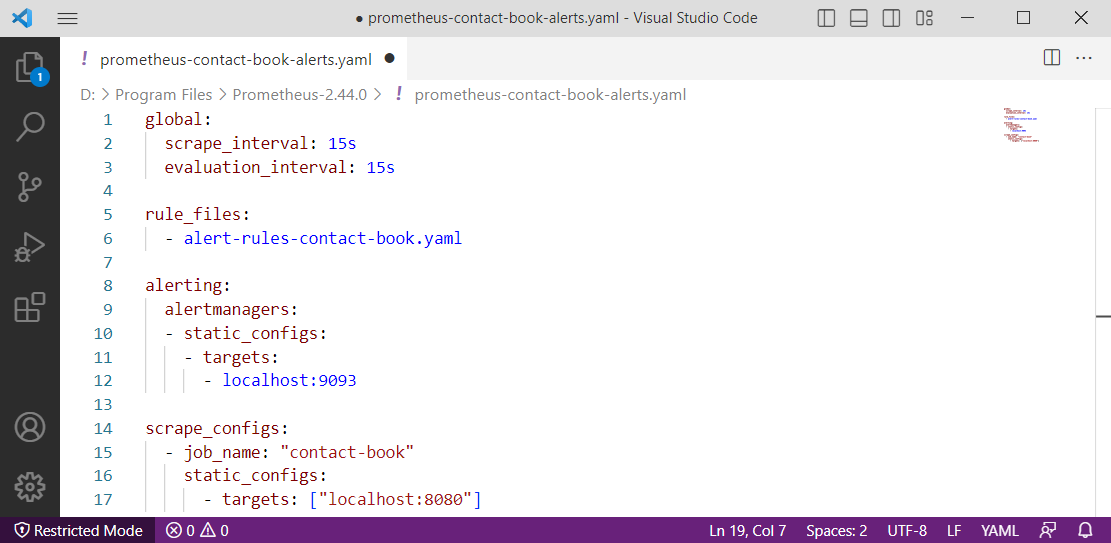
Having this expression, add the following rules configuration to the created YAML file:



Here we have a single rules group and an alert that will be fired if the given expression is true for at least 10 seconds. The alert will have a label and summary.

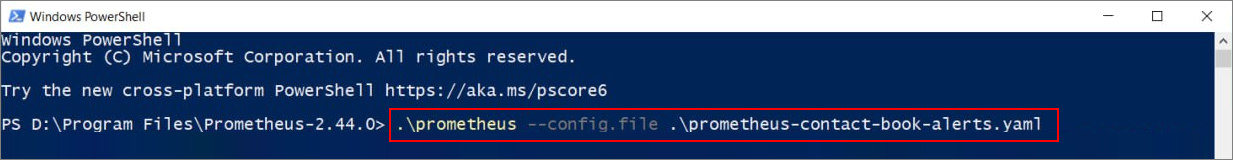
Save the file and let's modify (or create a new separate file) the Prometheus configuration. It should look like this:

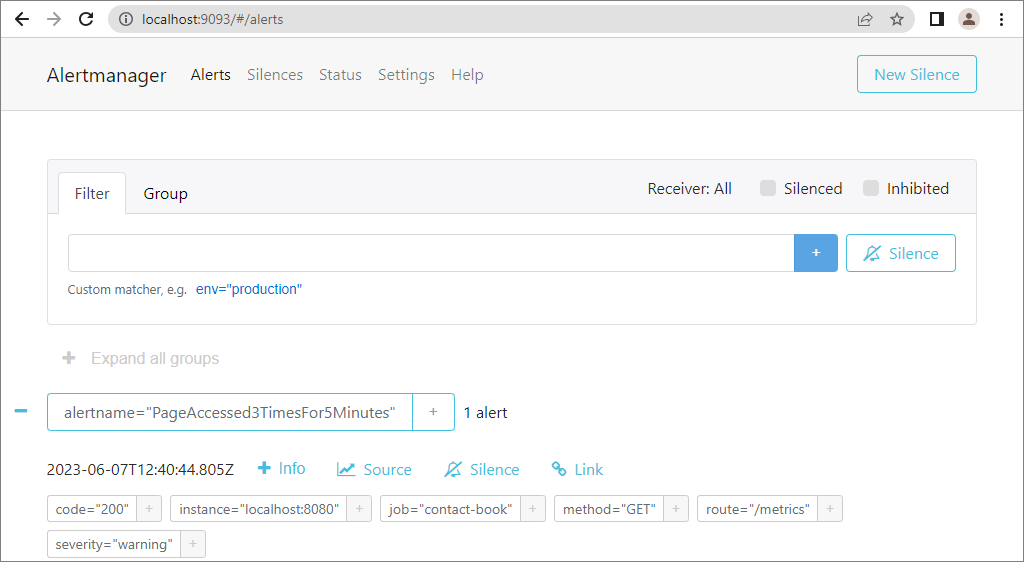
 🡪



As you can see, we have added the name of the rules file and configurations for connection to Alertmanager, which is accessible on <http://localhost:9093> by default. Also, we have set evaluation\_interval, which is the interval based on which Prometheus evaluates the query for alerting.

Now you should run Prometheus with the new / modified configuration file (don't forget to change the name of the configuration file if you have named it in a different way):

Go to the **Contact Book app** on <http://localhost:8080/> and reload the page more than 3 times. Now, access **Alertmanager** on <http://localhost:9093> and you should be able to see the new alert:



Now, visit the opened tab with Webhook.site. You should be able to see the detailed info about the sent incoming requests:

