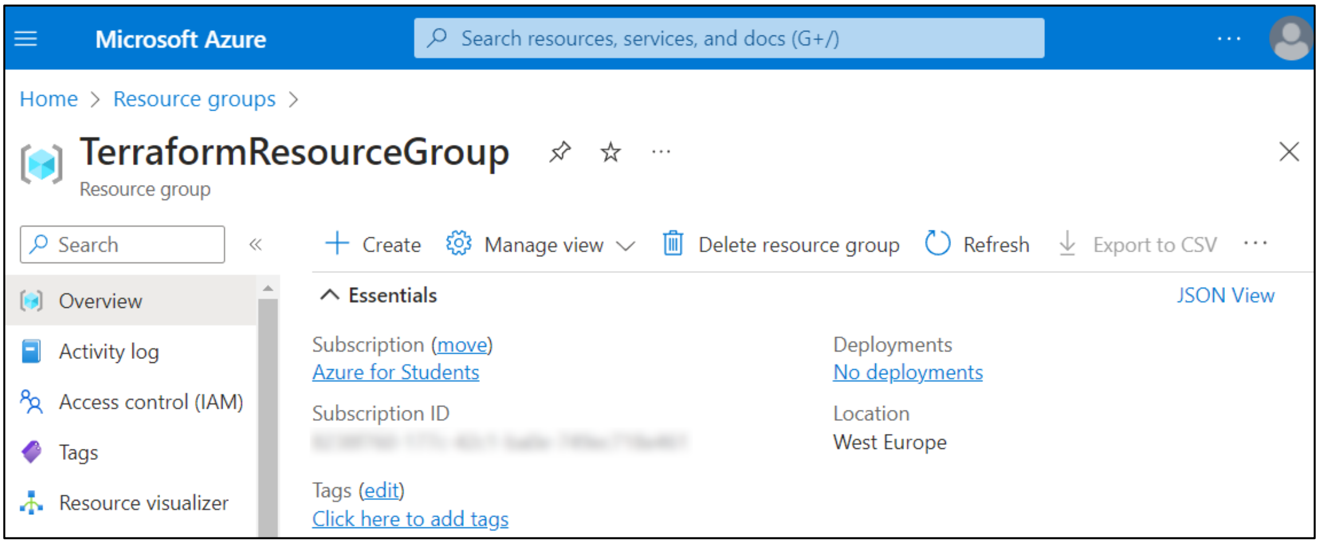
Exercise: IaC and Monitoring

Exercise assignment for the ["Containers and Clouds" course @ SoftUni.](https://softuni.bg)

I. Infrastructure as Code

1. Azure Resource Group

Now you have a task to **create a `Terraform` configuration** to **deploy an `Azure` resource group**.



Subscription (move)
Azure for Students

Subscription ID

Tags (edit)
Click here to add tags

Deployments
No deployments

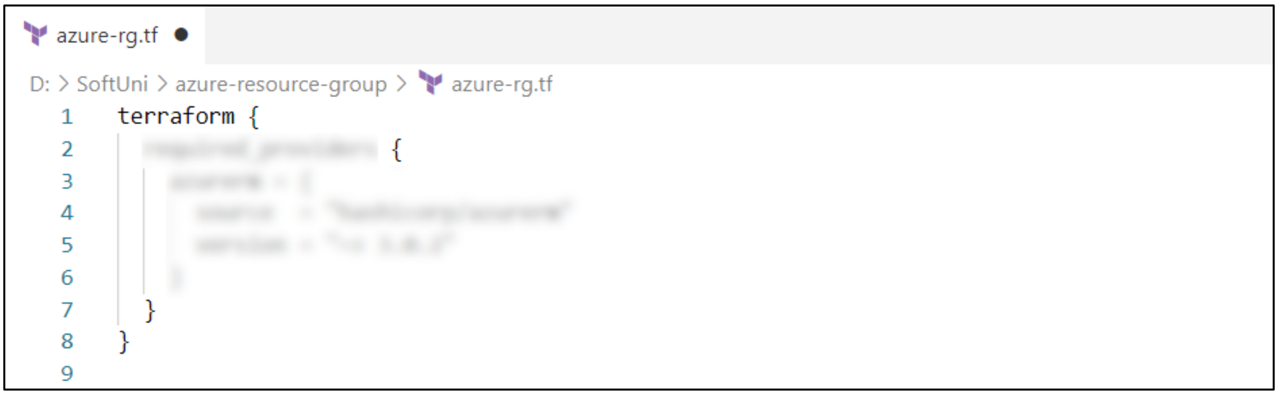
Location
West Europe

Hints

Open a **terminal** (for example PowerShell), **create a Terraform configuration folder** with an **empty configuration file** and **follow the steps below** to fulfill the task:

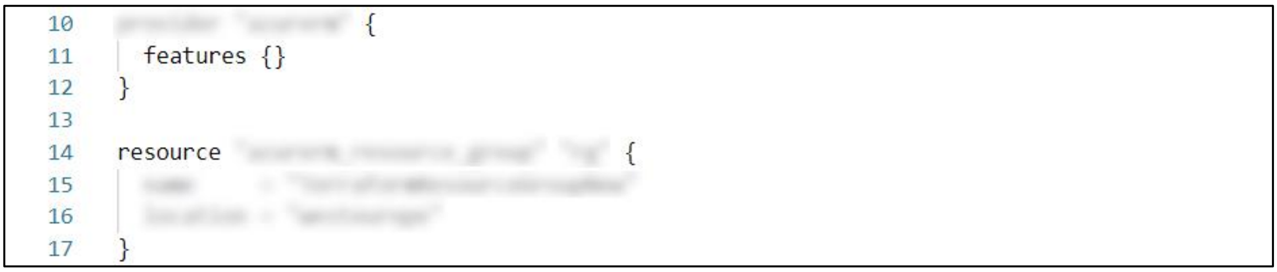
1. **Authenticate** using the **`Azure CLI`**, i.e. **log in to `Azure`**, as **`Terraform` must authenticate** to create infrastructure
2. **Write the configuration** for creating an **`Azure` resource group**
   - You need an **`Azure` provider**, available here: https://registry.terraform.io/providers/hashicorp/azurerm/latest
   - The **`Azure` provider** needs a **`feature {}` block** in the **configuration**
   - At the end, the **resource group** should be created using the "**`azurerm_resource_group`**" **`Terraform` resource**, whose **required arguments** can be seen here: https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs/resources/resource_group

   The **configuration file** looks like shown below. The **resource group name and location** are for you to choose:



D: > SoftUni > azure-resource-group > azure-rg.tf

```
terraform {
  [redacted] {
    [redacted]
  }
}

```



```
[redacted] {
  features {}
}

resource [redacted] {
  [redacted]
}
```

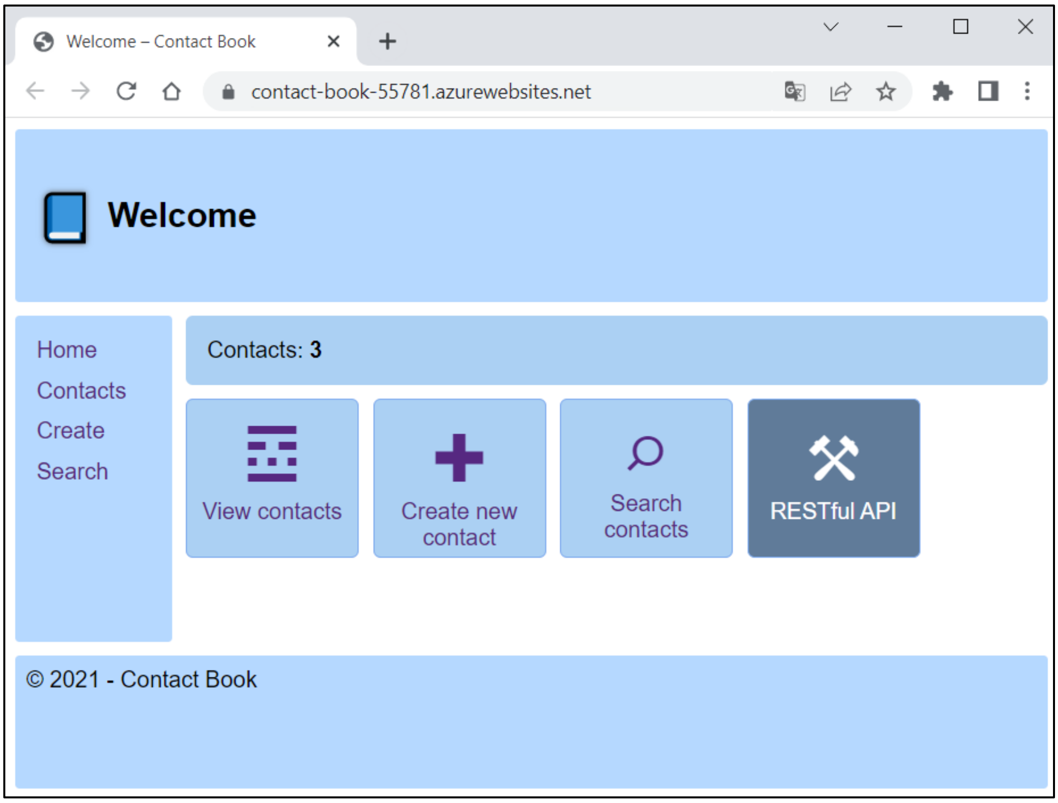
3. **Initialize**, **format**, **validate** and **apply your `Terraform` configuration**
4. **Navigate to `Azure Portal`** in the browser and validate that a **resource group was created**

Later you can **delete the resource group** from **`Azure`** again **using `Terraform`**.

As we know how to **create an `Azure` resource group with `Terraform`**, let's see how this would be **useful for us in the next task**.

2. Azure Web App

You are already **familiar with `Azure` Web Apps** and now you should **use `Terraform`** to **create a resource group**, then **create an `App Service Plan`** and finally **deploy the** "**`Contact Book`**" **app to `Azure`** from a **`GitHub` repo**.



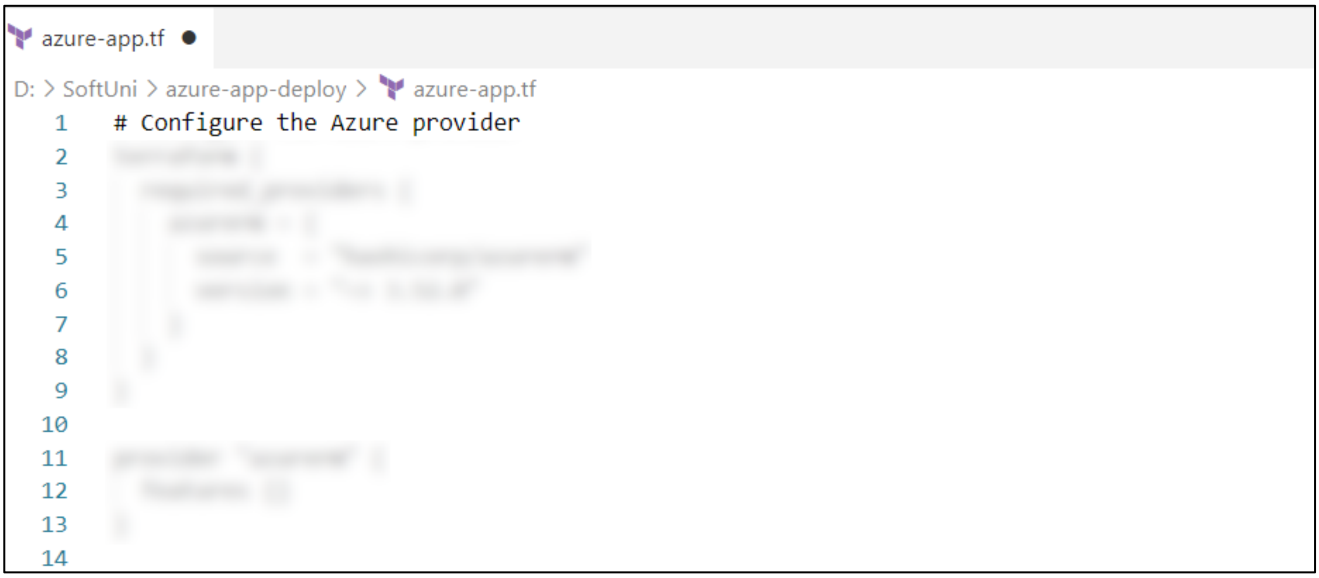
"**`Contact Book`**" is a **Node.js app without a database**, available here: https://github.com/nakov/ContactBook.

Hints

To **fulfill your task**, you need to **create a `Terraform` configuration file**. Find the **`Terraform` resources** you need in the **`Terraform Registry`** and use them: https://registry.terraform.io.

The **configuration** you should write:

- Uses and configures an **Azure provider** (as in the previous exercise)

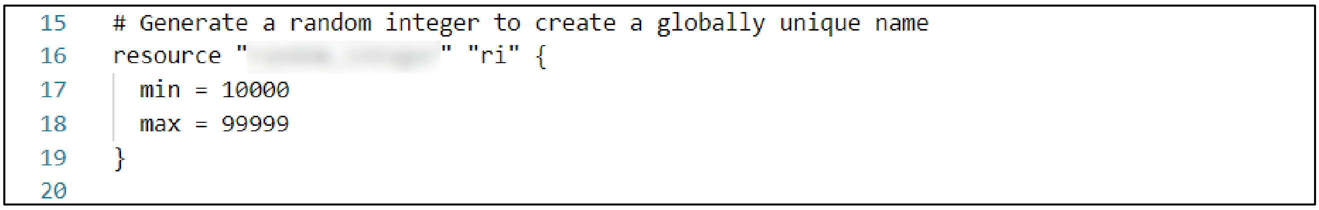


azure-app.tf ●

D: > SoftUni > azure-app-deploy > azure-app.tf

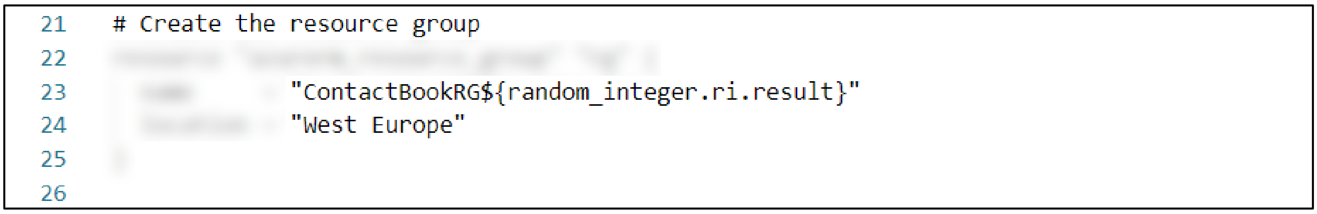
```
# Configure the Azure provider
[illegible]
```

- **Generates a random integer** with **minimum and maximum number range** to be used for **creating unique resource names**



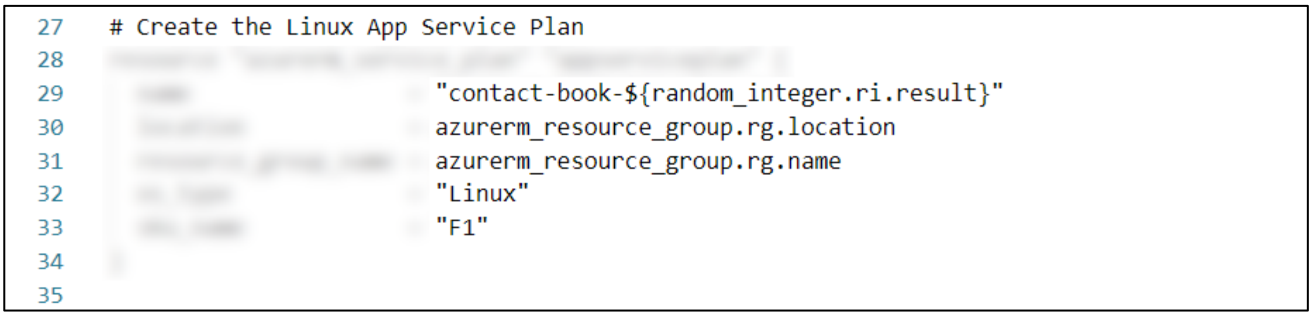
```
# Generate a random integer to create a globally unique name
resource "[illegible]" "ri" {
  min = 10000
  max = 99999
}
```

- Creates a **resource group**, whose name **uses the randomly-generated integer** by a **reference to the above resource**



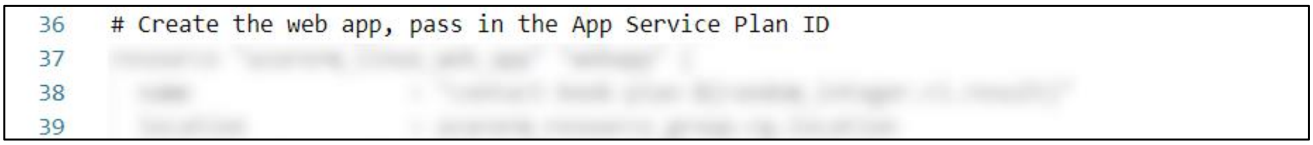
```
# Create the resource group
[illegible]
  [illegible]     "ContactBookRG${random_integer.ri.result}"
  [illegible]     "West Europe"
[illegible]
```

- Creates an **`App Service Plan`** with **name**, **location** (**reference the location** from the **resource group**), **resource group name** (reference the **name of the resource group**), **operating system** (set to "**`Linux`**") and **type of SKU** (set to "**F1**")



```
# Create the Linux App Service Plan
[illegible]
  [illegible]                 "contact-book-${random_integer.ri.result}"
  [illegible]                 azurerm_resource_group.rg.location
  [illegible]                 azurerm_resource_group.rg.name
  [illegible]                 "Linux"
  [illegible]                 "F1"
[illegible]
```

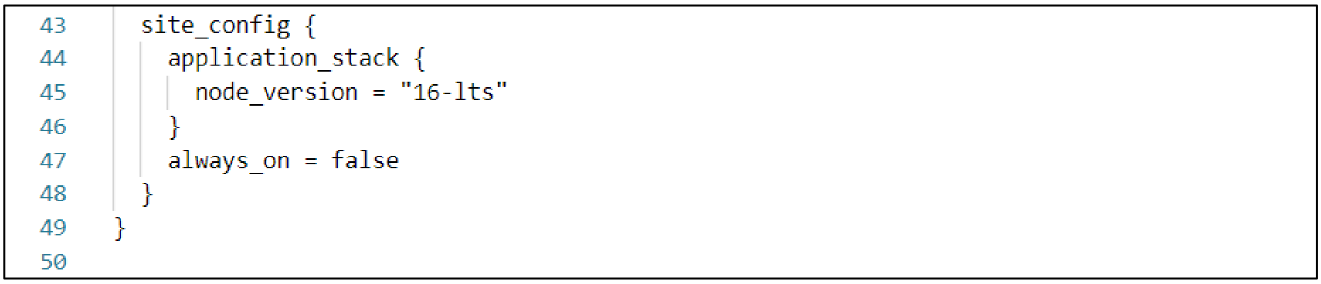
- Creates an **Azure Linux Web app** with **name**, **location**, **resource group name** and the **id of the service plan** (use **references** to the above resources)



```
# Create the web app, pass in the App Service Plan ID
[redacted]
```

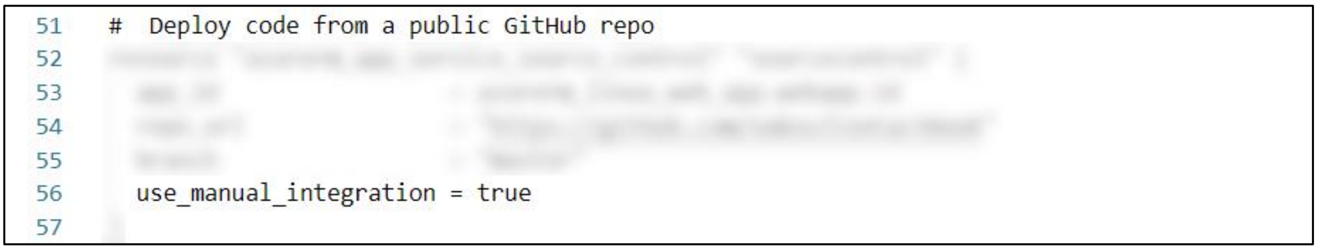
```
[redacted]

```

  - In addition, you should **add site configurations** including the **app's Node.js version** and a restriction for the **app to not always be on** (as we use the **free pricing plan**)



```
  site_config {
    application_stack {
      node_version = "16-lts"
    }
    always_on = false
  }
}

```

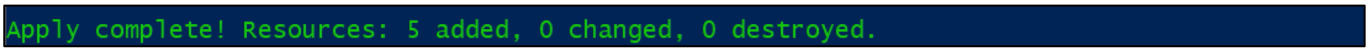
- **Deploys code** from the https://github.com/nakov/ContactBook **repo**, providing the **Web app id**, the **URL of the repo** and the **main branch name**



```
#  Deploy code from a public GitHub repo
[redacted]
  use_manual_integration = true
[redacted]
```

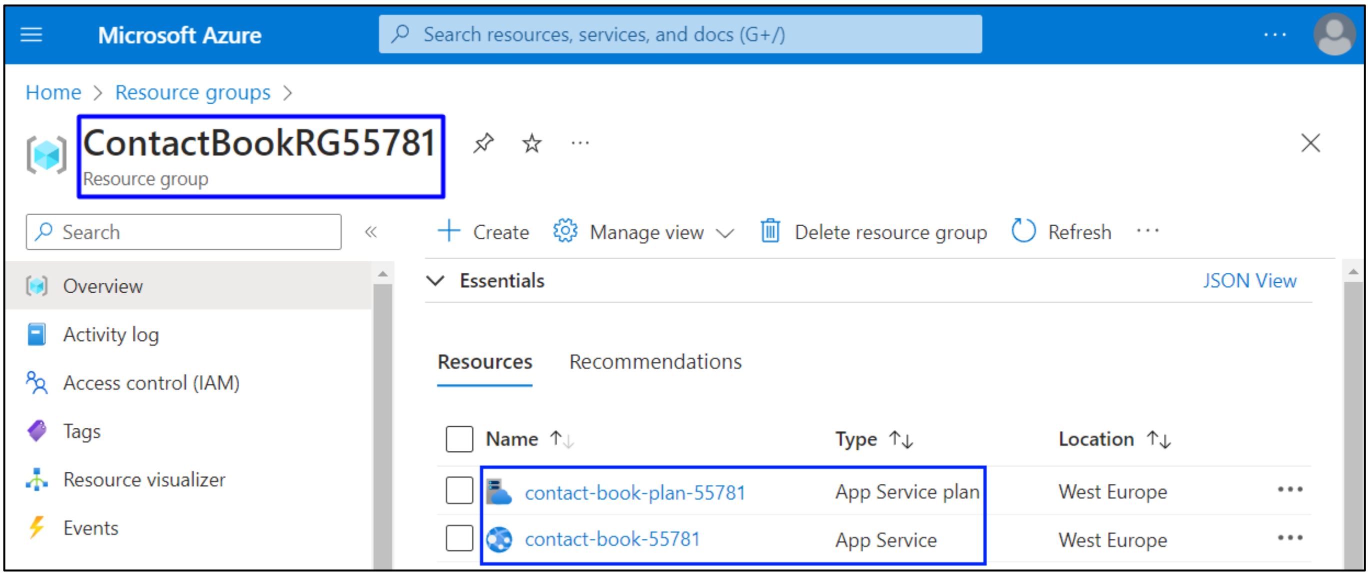
  - Moreover, we should set the **`use_manual_integration` argument** to **`true`**, so that we **agree to deploy the app and its updates manually** when we use an **external Git** (a public GitHub repo, which is not our own and we cannot run CI/CD in GitHub Actions)

When **ready with the configuration file**, **initialize `Terraform`**, **format and validate the configuration** and **provision the resources** from the file. Know that this may **take a while**. It should be **successful** at the end:



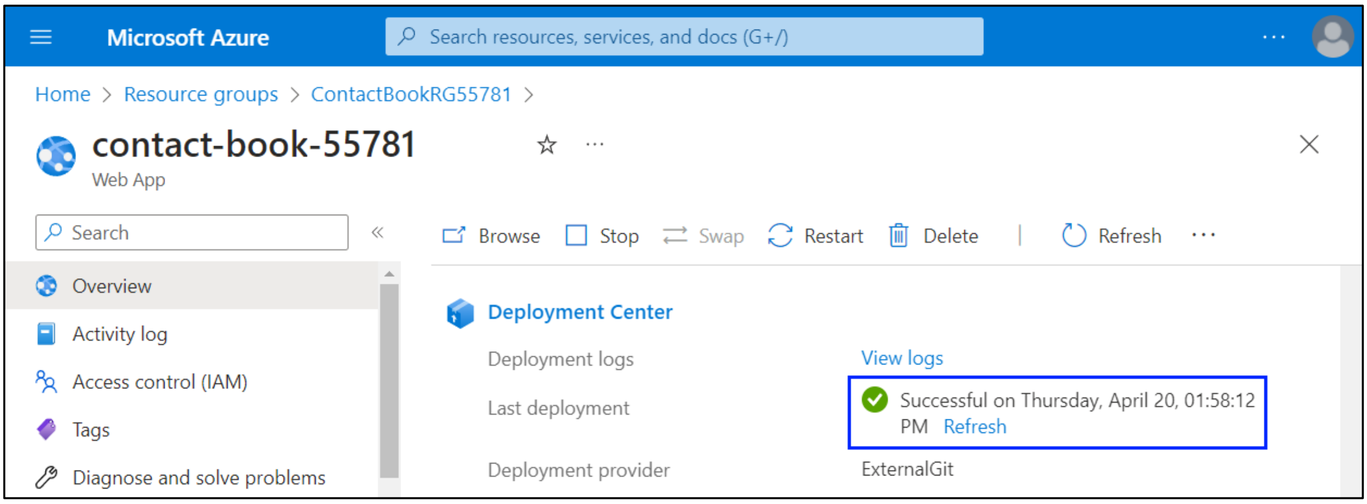
```
Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
```

When **done**, make sure that you **have a resource group**, an **app service plan** and a **Web app** in **`Azure`**:



Name	Type	Location
contact-book-plan-55781	App Service plan	West Europe
contact-book-55781	App Service	West Europe

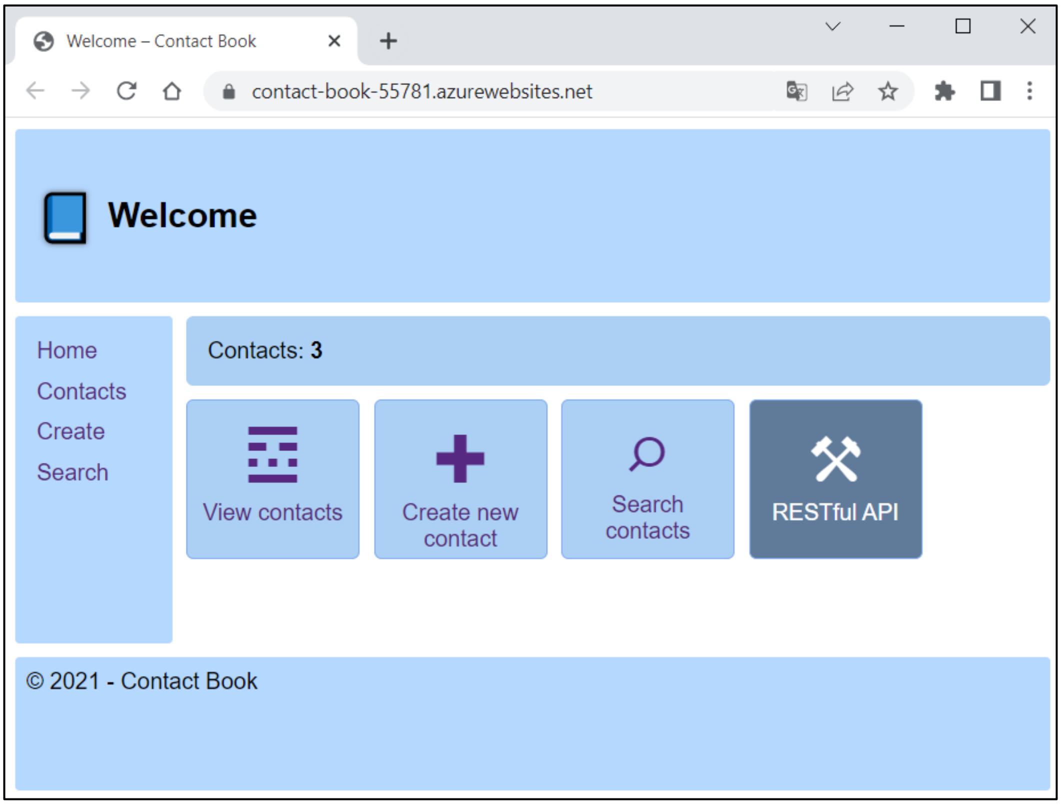
Also, make sure that the "**`Contact` Book**" **app is up and working** on the **provided domain URL** in Azure. First, however, you should **wait a bit** and make sure that the **deployment is successful**:



Deployment logs: View logs

Last deployment: Successful on Thursday, April 20, 01:58:12 PM Refresh

Deployment provider: ExternalGit

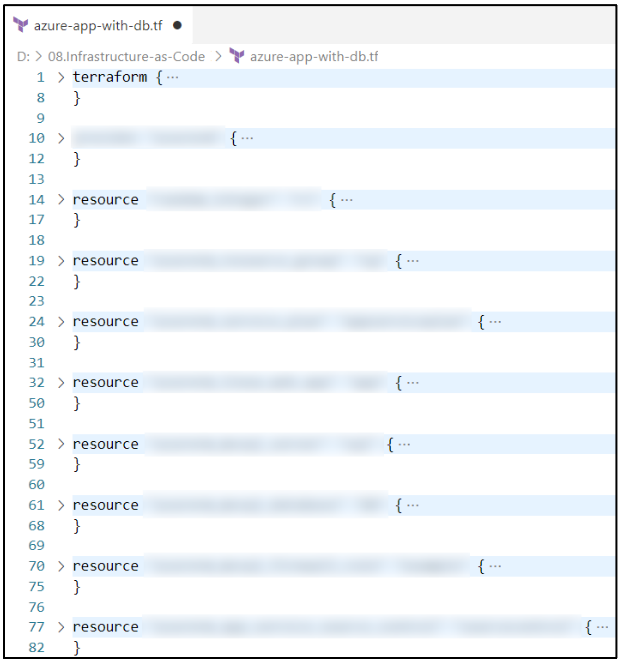


Finally, you can **destroy the created Azure resources** using the **well-known `Terraform` command**.

And this is how you can **deploy an app to `Azure`** with some easy steps, using **`Terraform`**.

3. Azure Web App with Database

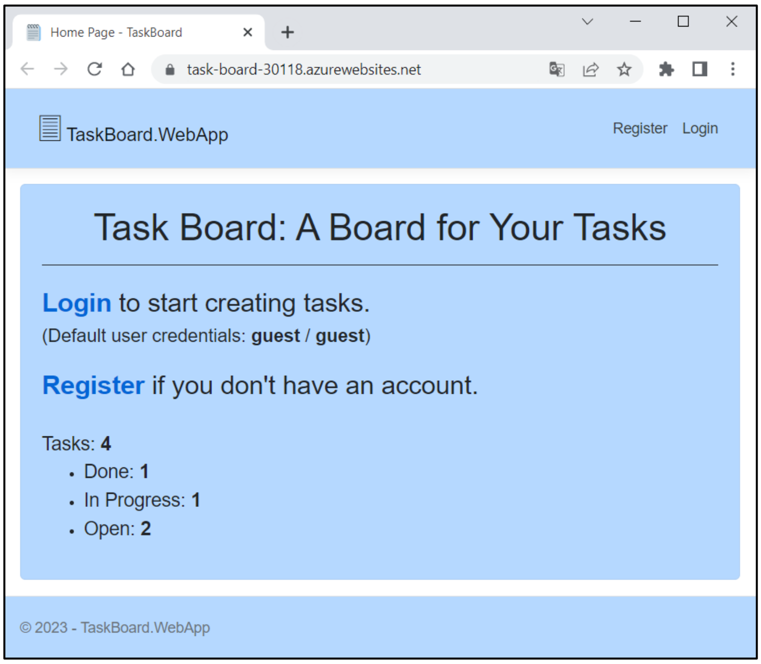
Create a **`Terraform` configuration** to **create and deploy the** "**`TaskBoard`**" **Web app** from the **resources** to **`Azure Web Apps`**. It is an **ASP.NET Core Web app** with a **SQL Server database**, which you should **upload to a GitHub repo** before you start.



azure-app-with-db.tf ●

D: > 08.Infrastructure-as-Code > azure-app-with-db.tf

```
terraform {···
}

[redacted] {···
}

resource [redacted] {···
}

resource [redacted] {···
}

resource [redacted] {···
}

resource [redacted] {···
}

resource [redacted] {···
}

resource [redacted] {···
}

resource [redacted] {···
}

resource [redacted] {···
}
```



Home Page - TaskBoard

task-board-30118.azurewebsites.net

TaskBoard.WebApp

Register Login

Task Board: A Board for Your Tasks

Login to start creating tasks.

(Default user credentials: **guest** / **guest**)

Register if you don't have an account.

Tasks: **4**

- Done: **1**
- In Progress: **1**
- Open: **2**

© 2023 - TaskBoard.WebApp

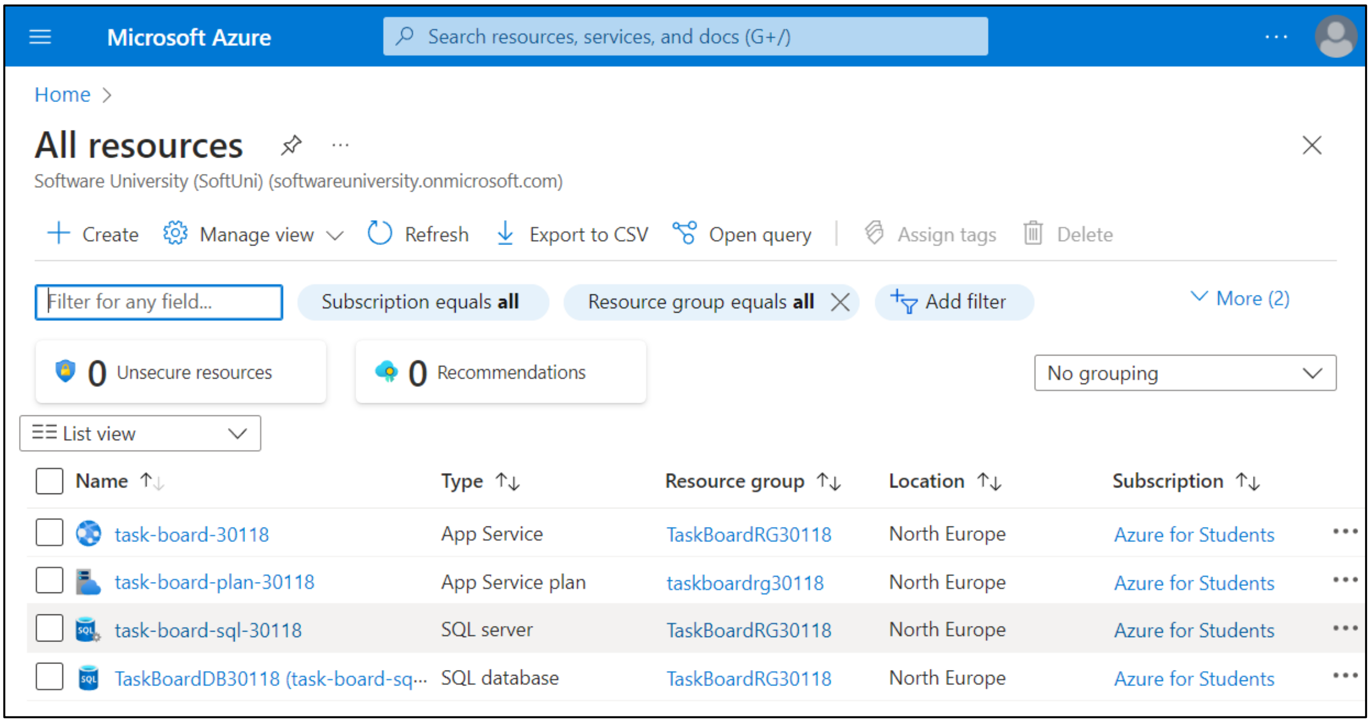
Write and Apply a Terraform Configuration

In this task, you can use the **Terraform configuration from the previous task** but you should make the following **modifications and additions**:

- Create a **server resource in Azure** with **name**, **resource group name**, **location**, **version**, **administrator username** and **administrator password** arguments
- Create a **database resource in Azure** with **name**, **server ID**, **collation**, **license type**, **SKU name** and **zone redundancy** arguments
- Create a **firewall rule for the Azure server**, which has a **name** and **server ID** and sets "**`0.0.0.0`**" as **start and end IP addresses** (this means that it allows other **Azure resources to access the server**)
- **Application stack** should be set to **`dotnet_version = "6.0"`**
- The **Linux Web app** should contain a **`connection_string` block** with:
  - Name: "**`DefaultConnection`**"
  - Type: "**`SQLAzure`**"
  - Value: "**`Data Source=tcp:${`*`fully qualified domain name of the MSSQL server`*`},1433;Initial Catalog=${`*`name of the SQL database`*`};User ID=${`*`username of the MSSQL server administrator`*`};Password=${`*`password of the MSSQL server administrator`*`};Trusted_Connection=False; MultipleActiveResultSets=True;`**"
- The **GitHub repo URL** should be changed to point out a **repo with the source code** of the "**`TaskBoard`**" **app**

Find the **Terraform resources** you need and **how to configure them** by yourself. Also, use the **random integer** you have created as a resource to **generate unique names**, as well as **resource references** where possible.

When your **configuration is written**, use the well-known **`Terraform` commands** to **apply it**. After a while, your **declared resources should be provisioned in `Azure`**:



Name	Type	Resource group	Location	Subscription
task-board-30118	App Service	TaskBoardRG30118	North Europe	Azure for Students
task-board-plan-30118	App Service plan	taskboardrg30118	North Europe	Azure for Students
task-board-sql-30118	SQL server	TaskBoardRG30118	North Europe	Azure for Students
TaskBoardDB30118 (task-board-sq...	SQL database	TaskBoardRG30118	North Europe	Azure for Students

And then, when the **app is deployed from the GitHub repo**, your **app should be up and working**.

Separate Configuration to Multiple Files

What we should do now is **separate our Terraform configuration to multiple files**, as it is **good practice** that allows **configuration modularity**, **reusability**, etc.

When done, we will have the **following files** (not necessary with the same file names):

- **`main.tf`** – the main Terraform configuration file
- **`variables.tf`** – contains variable declarations
- **`values.tfvars`** – contains values for the variables
- **`outputs.tf`** – contains outputs declarations

Let's see how to **separate our configuration**.

Step 1: Define Input Variables

You have the **configuration for provisioning and deploying a Web app with database** but it is all in one **`.tf` file** – including resource names, administrator credentials, etc. There are quite a **few hard coded values** that would make sense to have as **input parameters instead**, as this would allow us to **re-use the same template** to create multiple environments with a slightly different configuration.

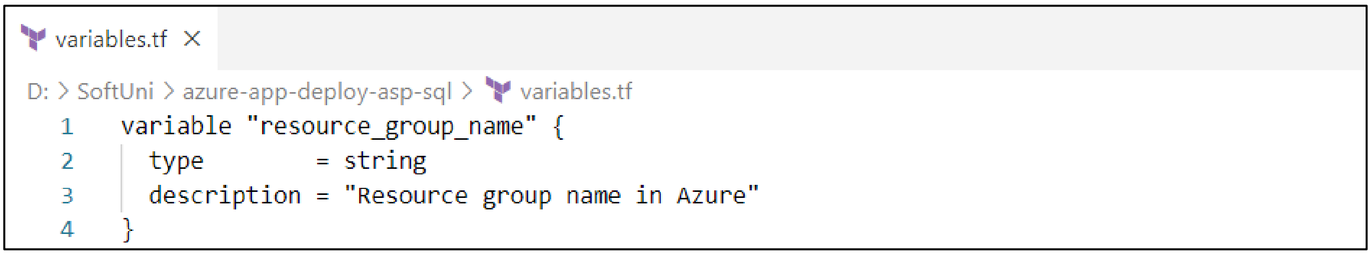
In our **configuration**, we have the following **values** that can be turned into **input parameters**:

- Resource group name
- Resource group location
- App service plan name
- App service name
- SQL server name
- SQL database name
- SQL administrator login username
- SQL administrator password
- Firewall rule name

- GitHub repo URL

Create a **new `.tf` file** in the **Terraform configuration directory** and let's **define the input variables**. Each **variable** will have a **name**, **type** and **description**. In addition, it can have a **default value** that you can add if you want.

Define each variable from the above list in this way:

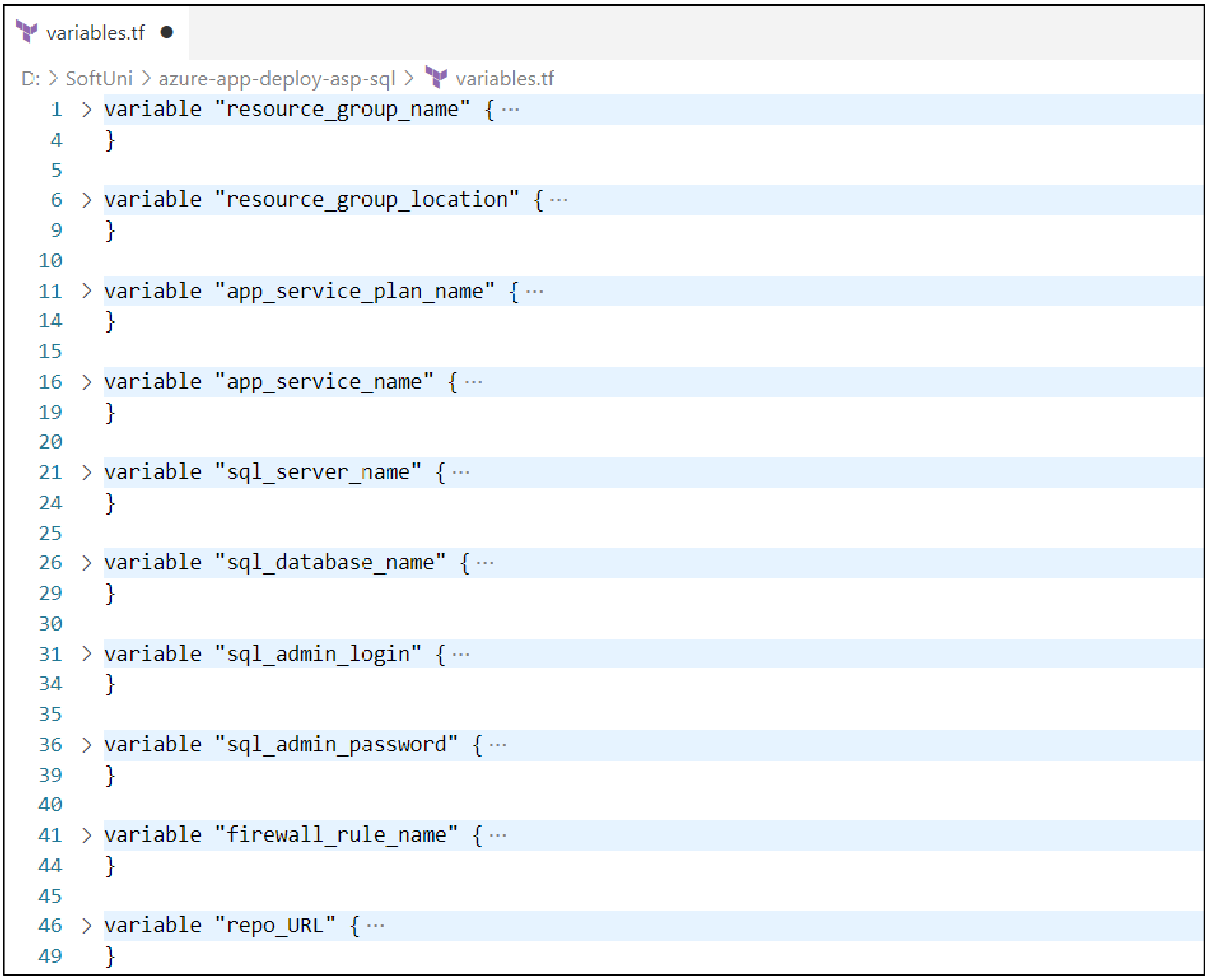


variables.tf ×

D: > SoftUni > azure-app-deploy-asp-sql > variables.tf

```
variable "resource_group_name" {
  type        = string
  description = "Resource group name in Azure"
}
```

You can go on with the **rest of the variables' definition by yourself**, following the **syntax** shown. At the end, you should have **10 variables**:

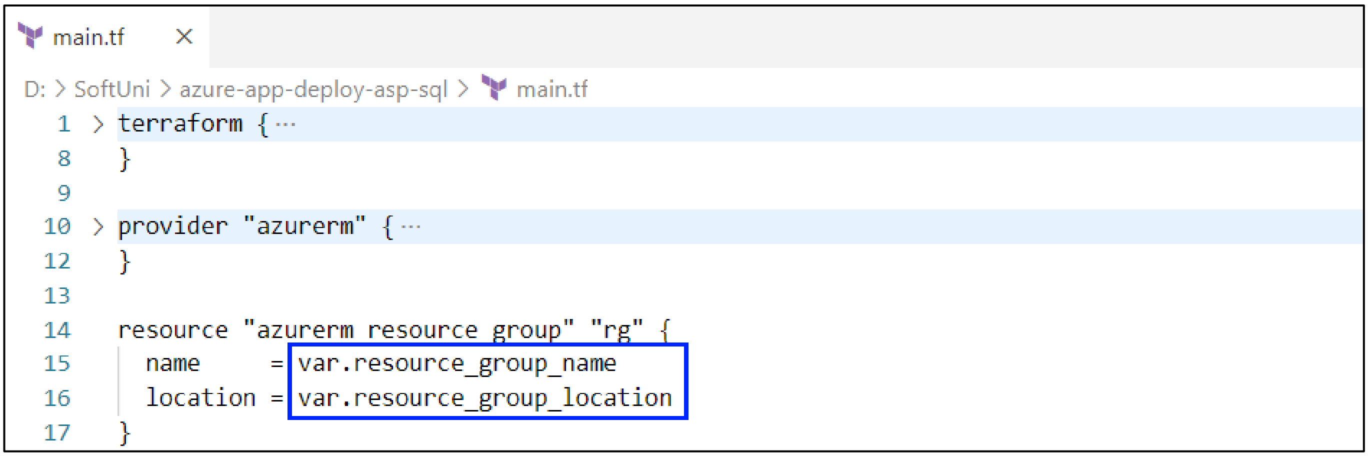


variables.tf ●

D: > SoftUni > azure-app-deploy-asp-sql > variables.tf

```
> variable "resource_group_name" {⋯
}

> variable "resource_group_location" {⋯
}

> variable "app_service_plan_name" {⋯
}

> variable "app_service_name" {⋯
}

> variable "sql_server_name" {⋯
}

> variable "sql_database_name" {⋯
}

> variable "sql_admin_login" {⋯
}

> variable "sql_admin_password" {⋯
}

> variable "firewall_rule_name" {⋯
}

> variable "repo_URL" {⋯
}
```

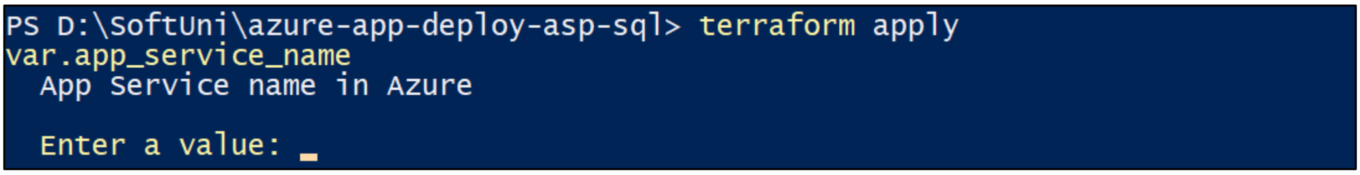
Now let's **use these variables** in the **main Terraform configuration file** we have. To do this, use the following **syntax**: **`var.{`*`variable name`*`}`**. Do it like this for **all input variables** you defined:



```
main.tf
D: > SoftUni > azure-app-deploy-asp-sql > main.tf
> terraform {···
  }

> provider "azurerm" {···
  }

  resource "azurerm_resource_group" "rg" {
    name     = var.resource_group_name
    location = var.resource_group_location
  }
```

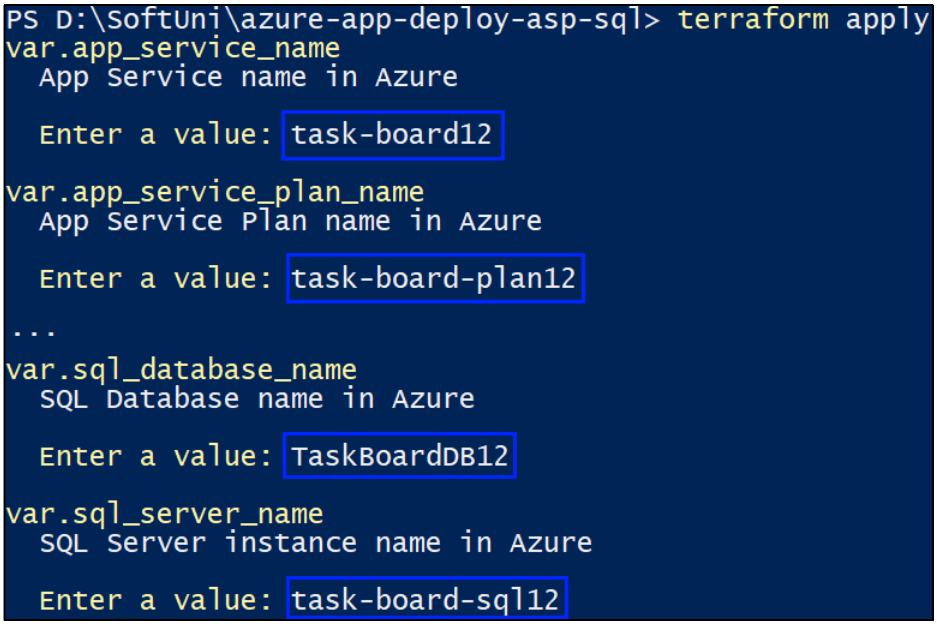
In addition, you can still use the **randomly generated integer value** as **part of the resource names** or you can **remove this resource** if you don't need it. However, make sure that your **resource names are unique enough** or **errors may appear**.

Now let's try to **apply the Terraform configuration** we have and see what will happen:



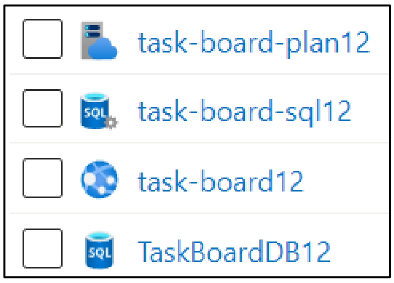
```
PS D:\SoftUni\azure-app-deploy-asp-sql> terraform apply
var.app_service_name
  App Service name in Azure

  Enter a value: _
```

As you can see, you are **prompted to enter an app service name** for the **`app_service_name` input variable**. You should **add values for all variables** and then they will be **used in your configuration**. All of them are **required** as we didn't put default values.



```
PS D:\SoftUni\azure-app-deploy-asp-sql> terraform apply
var.app_service_name
  App Service name in Azure

  Enter a value: task-board12

var.app_service_plan_name
  App Service Plan name in Azure

  Enter a value: task-board-plan12

...
var.sql_database_name
  SQL Database name in Azure

  Enter a value: TaskBoardDB12

var.sql_server_name
  SQL Server instance name in Azure

  Enter a value: task-board-sql12
```

→

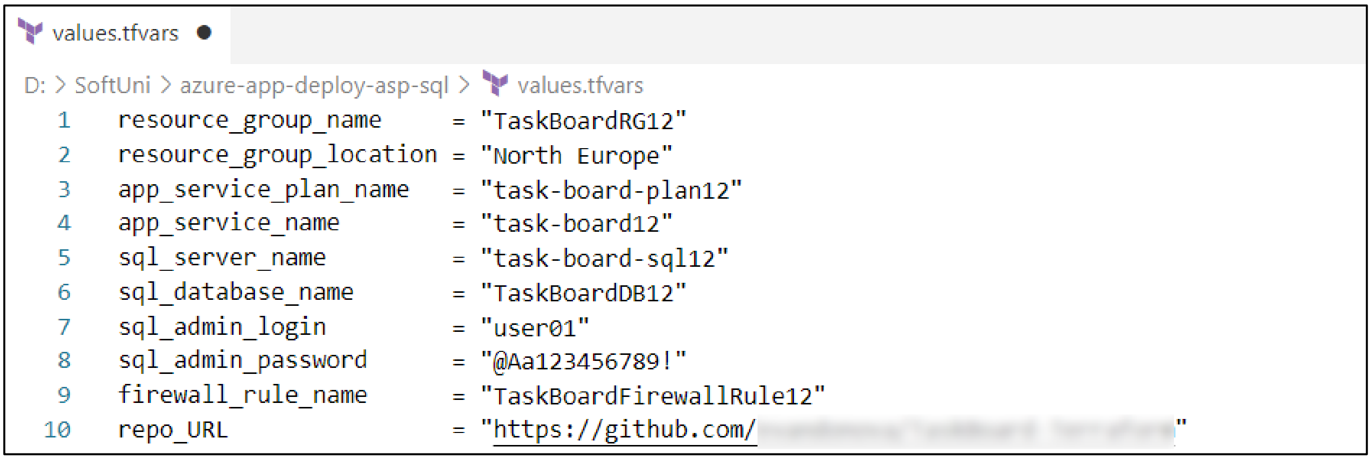


□ task-board-plan12
□ task-board-sql12
□ task-board12
□ TaskBoardDB12

Now we have **input variables for our configuration**, which is nice. However, if we run **`terraform destroy`**, we should **enter the same values again**, which is not pleasant.

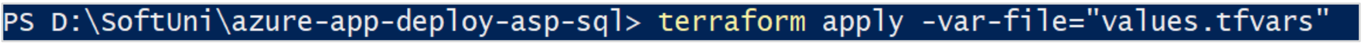
Step 2: Create File with Variable Values

If we **don't want to enter values for the input variables**, we can **create a file** for them. Create a **file** with the **`.tfvars` extension** and **add value for each variable** using this syntax: **`{variable name} = "{variable value}"`**.



```
resource_group_name     = "TaskBoardRG12"
resource_group_location = "North Europe"
app_service_plan_name   = "task-board-plan12"
app_service_name        = "task-board12"
sql_server_name         = "task-board-sql12"
sql_database_name       = "TaskBoardDB12"
sql_admin_login         = "user01"
sql_admin_password      = "@Aa123456789!"
firewall_rule_name      = "TaskBoardFirewallRule12"
repo_URL                = "https://github.com/[redacted]"
```

Now we can **apply our configuration** again, using the **`.tfvars` file** we created:



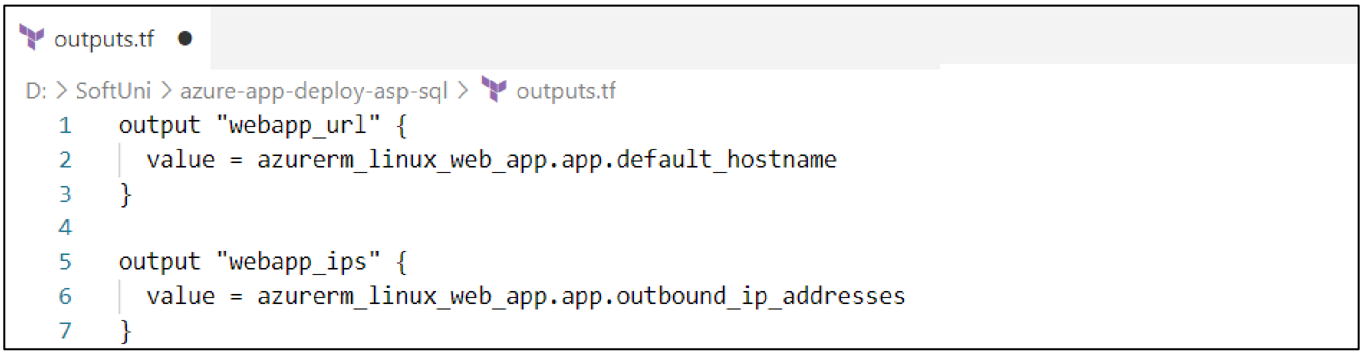
```
PS D:\SoftUni\azure-app-deploy-asp-sql> terraform apply -var-file="values.tfvars"
```

The **file should be found** and **values used** – you should **not be prompted** to add any value manually.

Step 3: Define Outputs

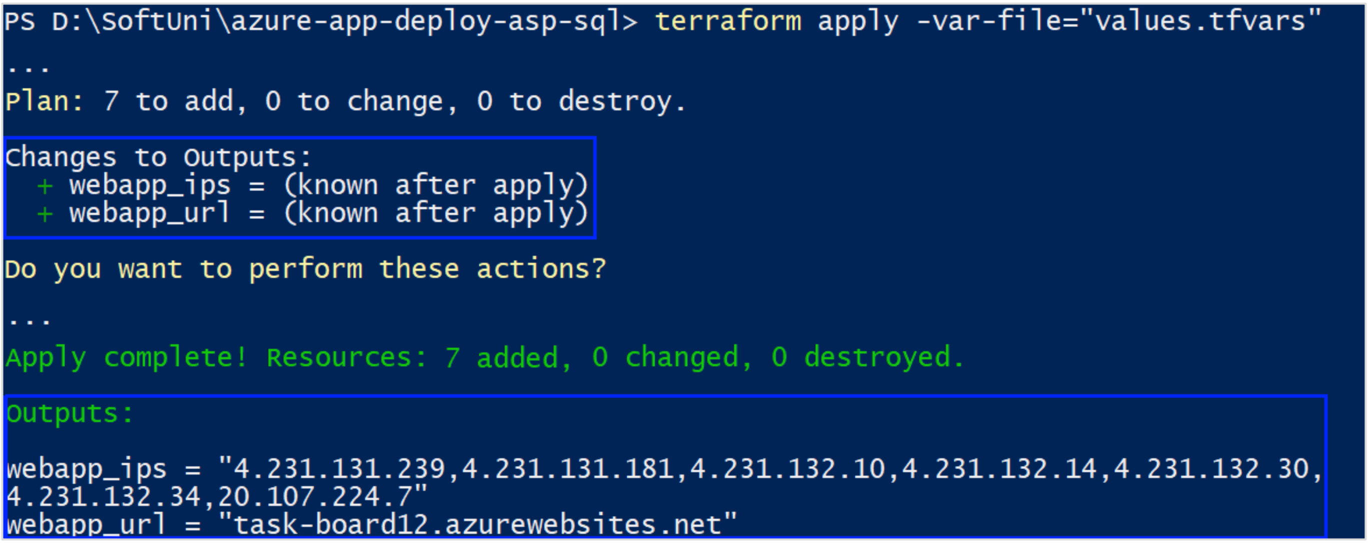
At the end, we can **add outputs** that will **print us the URL of the Azure Web app** that will be created and its **outbound IP addresses**. **Outputs** are basically just pieces **state information** that you want to have available for different purposes.

You should create **a new `.tf` file** and **define the outputs** with **name** and **value** using the following syntax:



```
output "webapp_url" {
  value = azurerm_linux_web_app.app.default_hostname
}

output "webapp_ips" {
  value = azurerm_linux_web_app.app.outbound_ip_addresses
}
```

When you **apply the configuration**, the **values of the outputs** should be **printed in the terminal**:



```
PS D:\SoftUni\azure-app-deploy-asp-sql> terraform apply -var-file="values.tfvars"
...
Plan: 7 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + webapp_ips = (known after apply)
  + webapp_url = (known after apply)

Do you want to perform these actions?
...
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.

Outputs:

webapp_ips = "4.231.131.239,4.231.131.181,4.231.132.10,4.231.132.14,4.231.132.30,
4.231.132.34,20.107.224.7"
webapp_url = "task-board12.azurewebsites.net"
```

After all this separation of the **Terraform configuration to files**, it should still be **working** and **provision the resources in Azure** successfully.

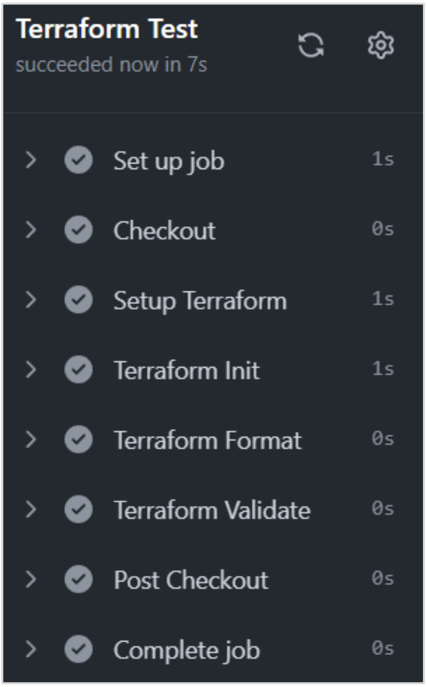
Now your **configuration follows good practices**. However, in the **next task** we will see how to **improve it** even more.

4. Terraform with CI/CD

Now we will **upload the Terraform configuration from** the **previous task** (for provisioning **Azure resources** and **deploying the "`TaskBoard`" Web app** to Azure Web Apps) **to GitHub** and will use **`GitHub Actions` workflows** to **test and run the configuration**.

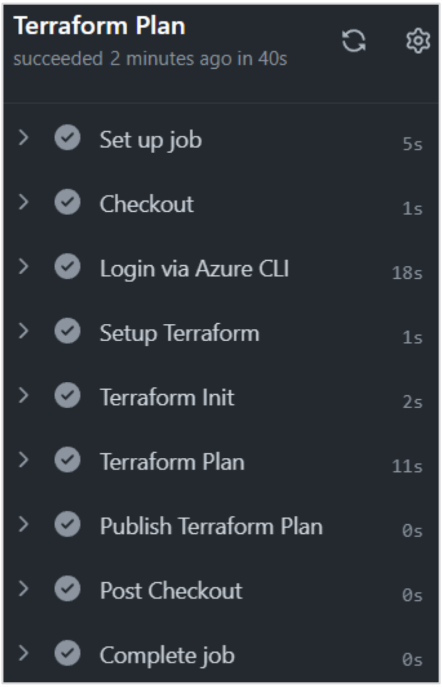
By combining **Terraform with GitHub Actions**, we can **automate the infrastructure provisioning process**, **ensure consistency**, and **integrate it into your CI/CD workflows**, promoting **efficient software delivery** and **reducing manual tasks**. It provides a streamlined and efficient **workflow for managing infrastructure as code**, making it easier to **maintain**, **test**, and **deploy your infrastructure resources**.

We will have **GitHub Actions workflows** that will provision the **Azure resources** we want:



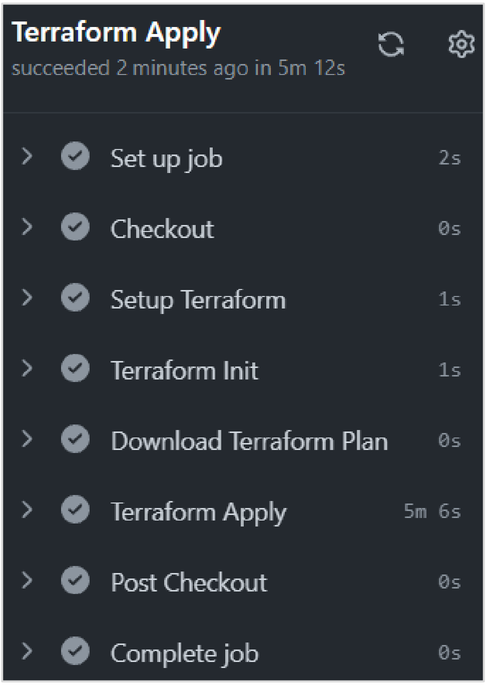
Terraform Test
succeeded now in 7s

Set up job	1s
Checkout	0s
Setup Terraform	1s
Terraform Init	1s
Terraform Format	0s
Terraform Validate	0s
Post Checkout	0s
Complete job	0s



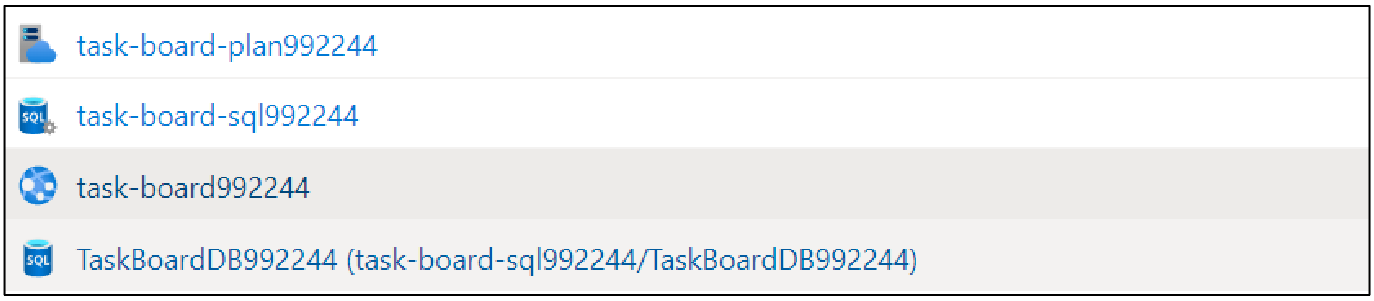
Terraform Plan
succeeded 2 minutes ago in 40s

Set up job	5s
Checkout	1s
Login via Azure CLI	18s
Setup Terraform	1s
Terraform Init	2s
Terraform Plan	11s
Publish Terraform Plan	0s
Post Checkout	0s
Complete job	0s



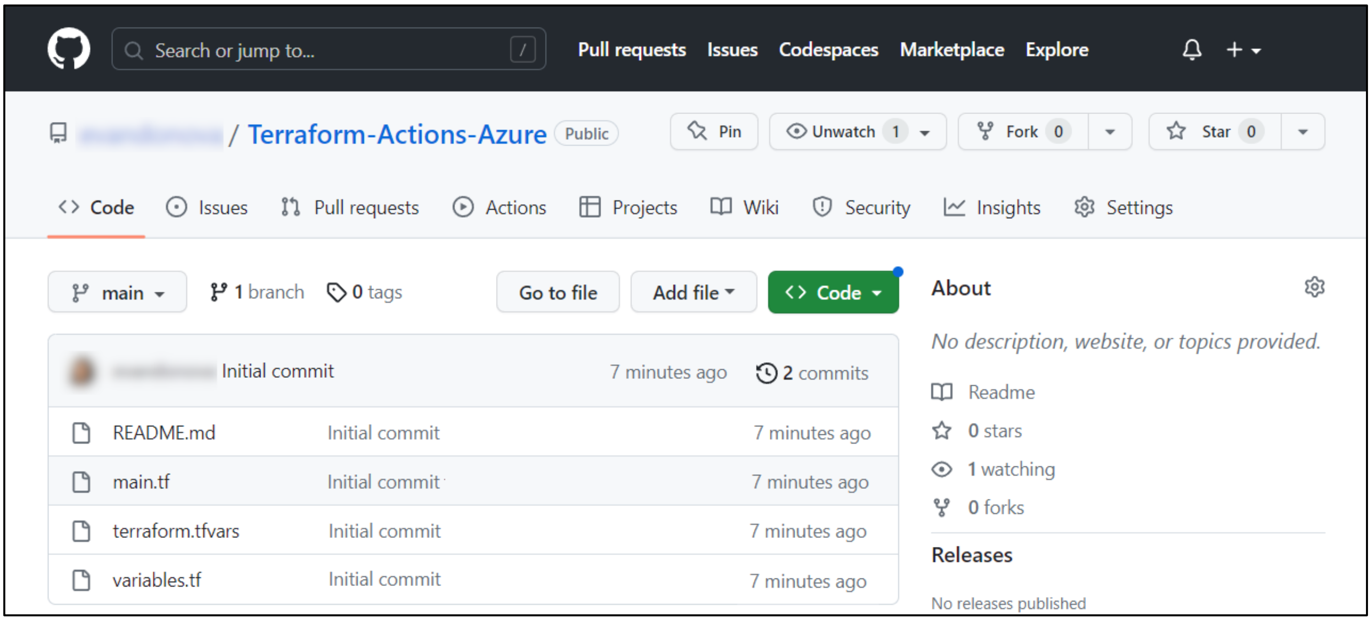
Terraform Apply
succeeded 2 minutes ago in 5m 12s

Set up job	2s
Checkout	0s
Setup Terraform	1s
Terraform Init	1s
Download Terraform Plan	0s
Terraform Apply	5m 6s
Post Checkout	0s
Complete job	0s



task-board-plan992244
task-board-sql992244
task-board992244
TaskBoardDB992244 (task-board-sql992244/TaskBoardDB992244)

Start by creating a **GitHub repository**, which should contain your **`main.tf` Terraform configuration file** and your additional **Terraform files** – **`terraform.tfvars`** and **`variables.tf`**:



[redacted] / Terraform-Actions-Azure Public

1 branch 0 tags

[redacted] Initial commit		7 minutes ago 2 commits
README.md	Initial commit	7 minutes ago
main.tf	Initial commit	7 minutes ago
terraform.tfvars	Initial commit	7 minutes ago
variables.tf	Initial commit	7 minutes ago

About

No description, website, or topics provided.

Readme
0 stars
1 watching
0 forks

Releases

No releases published

Note: when the **`.tfvars` file** with **variable values** is **named** "**`terraform`**", **Terraform finds it on its own** and you **should not point to it specifically** in the **Terraform commands you run**.

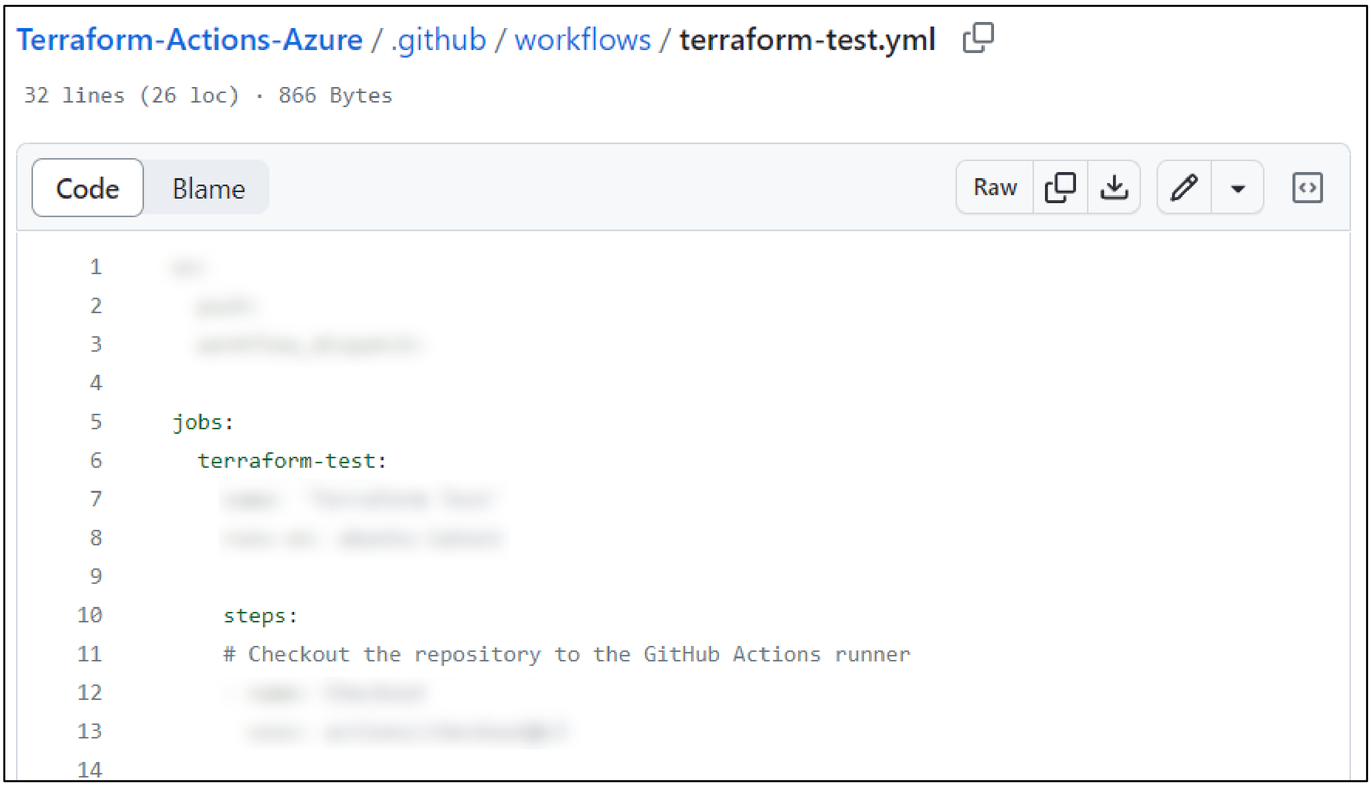
Also, you **don't need the `outputs.tf` file**, as you can use GitHub Actions to show you what you need when a workflow is run.

Now let's see how to write the **GitHub Actions workflows** we need.

Test Workflow

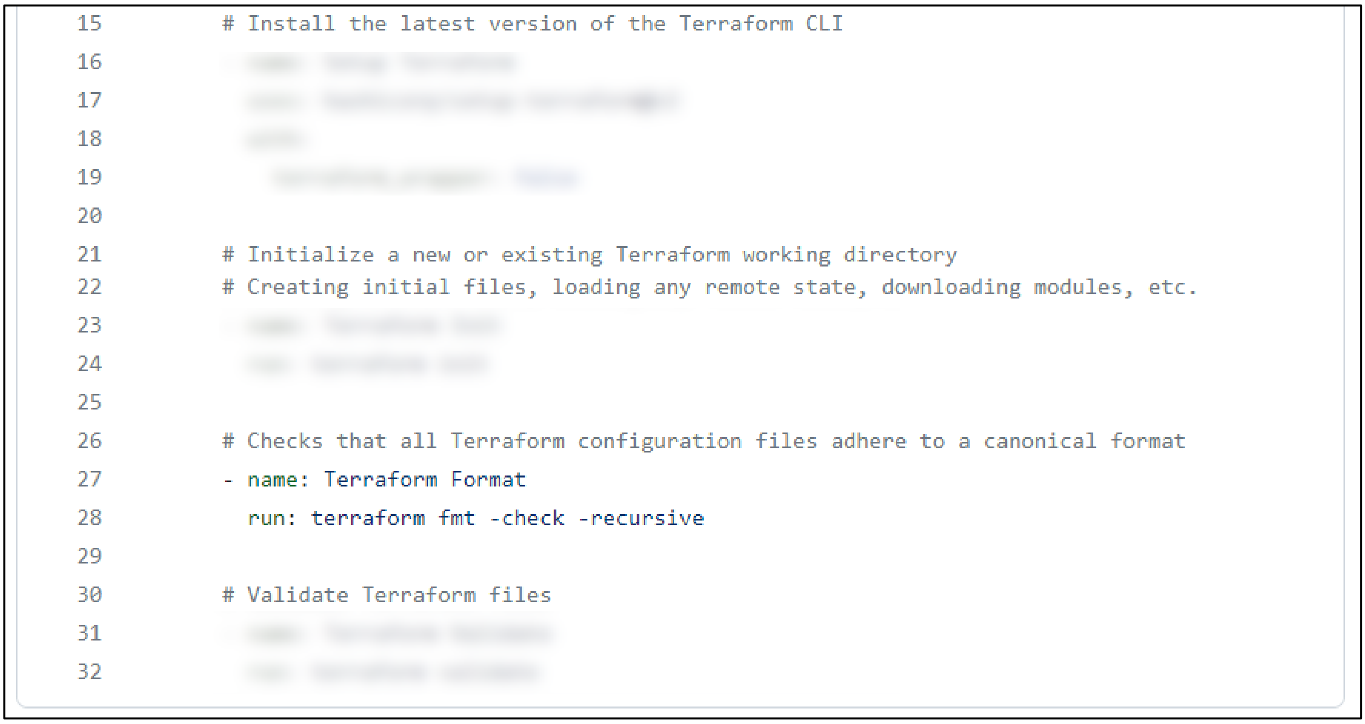
We will first write a **test workflow in GitHub Actions** that will try to **initialize the working directory**, **check if the configuration files are correctly formatted** and **validate the configuration**.

Create a **YAML file** in **GitHub Actions**. The **workflow** should look like this:


Terraform-Actions-Azure / .github / workflows / terraform-test.yml

32 lines (26 loc) · 866 Bytes

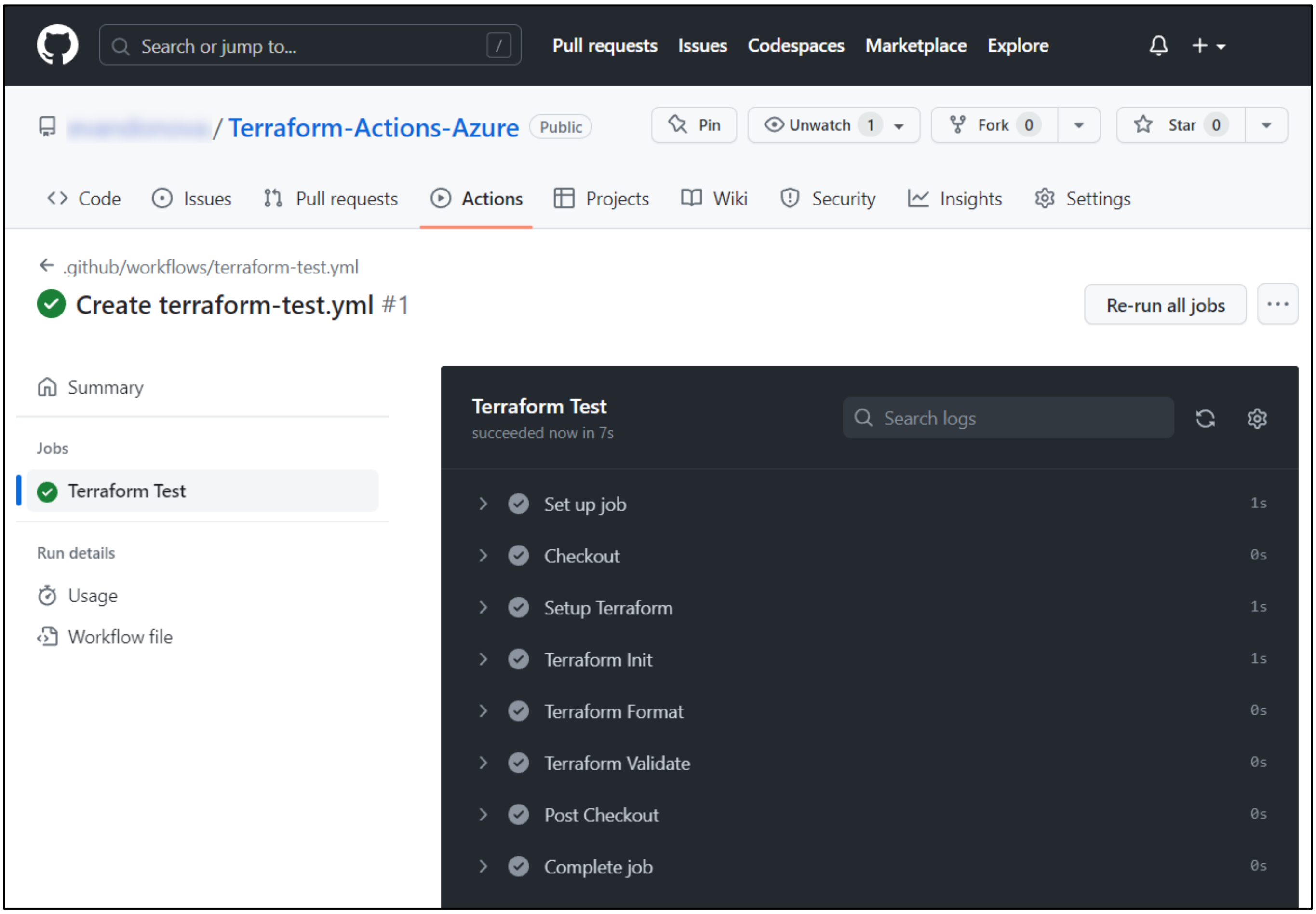
```
[redacted]

jobs:
  terraform-test:
    [redacted]

    steps:
    # Checkout the repository to the GitHub Actions runner
    [redacted]

```



```
      # Install the latest version of the Terraform CLI
      [redacted]

      # Initialize a new or existing Terraform working directory
      # Creating initial files, loading any remote state, downloading modules, etc.
      [redacted]

      # Checks that all Terraform configuration files adhere to a canonical format
      - name: Terraform Format
        run: terraform fmt -check -recursive

      # Validate Terraform files
      [redacted]
```

Look at the **comments in the above workflow** – they **describe the steps** for **testing the Terraform configuration**.

Write the workflow and **run it**. It should be **successful**:



.github/workflows/terraform-test.yml

Create terraform-test.yml #1

Terraform Test

Terraform Test
succeeded now in 7s

```
Set up job            1s
Checkout              0s
Setup Terraform       1s
Terraform Init        1s
Terraform Format      0s
Terraform Validate    0s
Post Checkout         0s
Complete job          0s
```

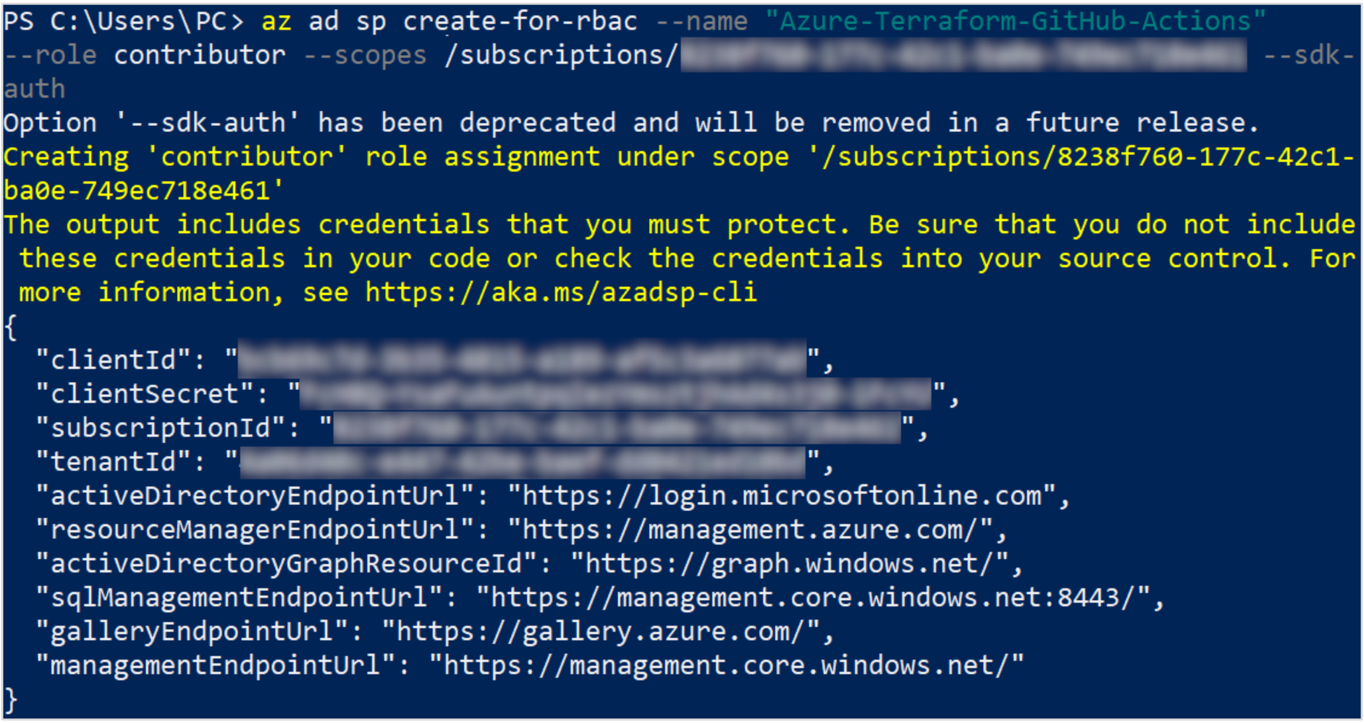
If you **receive any error**, **fix it** – you may have problems with your **Terraform configuration files** or the **workflow file** you have just created.

Apply Configuration Workflow

When we have a **valid configuration** with **working tests** in **GitHub Actions**, let's use a **workflow to provision resources** and **deploy the "TaskBoard" Web app** to **Azure**. You should **authenticate in Azure** using a **service principal** and then **write the workflow**.

Step 1: Create Service Principal

We should **create a service principal** with a "**Contributor**" **role** in **Azure** that we will use to **authenticate GitHub Actions**. Do it with the **following command locally** or **manually through Azure Portal**:

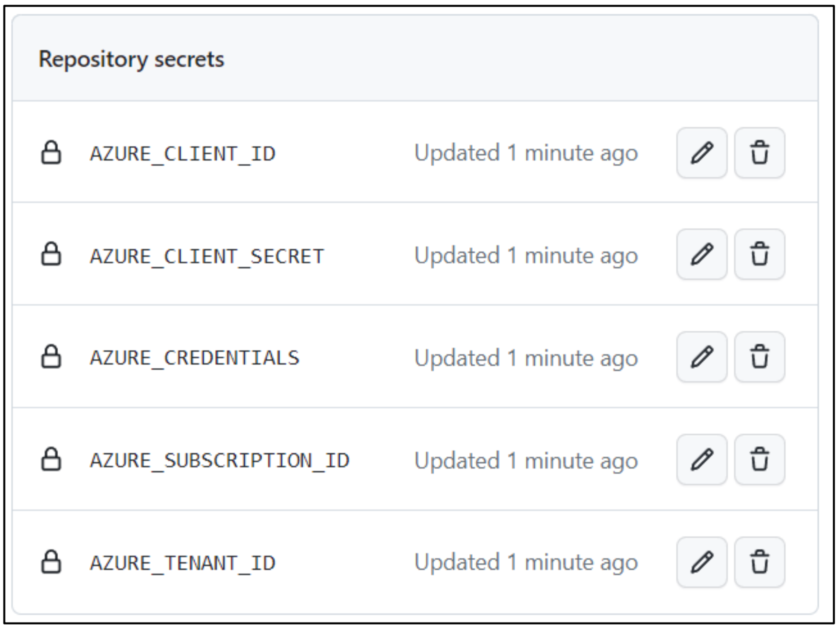


```
PS C:\Users\PC> az ad sp create-for-rbac --name "Azure-Terraform-GitHub-Actions"
--role contributor --scopes /subscriptions/[redacted] --sdk-
auth
Option '--sdk-auth' has been deprecated and will be removed in a future release.
Creating 'contributor' role assignment under scope '/subscriptions/8238f760-177c-42c1-
ba0e-749ec718e461'
The output includes credentials that you must protect. Be sure that you do not include
 these credentials in your code or check the credentials into your source control. For
 more information, see https://aka.ms/azadsp-cli
{
  "clientId": "[redacted]",
  "clientSecret": "[redacted]",
  "subscriptionId": "[redacted]",
  "tenantId": "[redacted]",
  "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/"
}
```

Copy the credentials JSON as you will need it for the next step.

Step 2: Create GitHub Secrets

As you know, it is **good practice** to **store your credentials** as **secrets in GitHub**. You need the following secrets:



Repository secrets

AZURE_CLIENT_ID	Updated 1 minute ago
AZURE_CLIENT_SECRET	Updated 1 minute ago
AZURE_CREDENTIALS	Updated 1 minute ago
AZURE_SUBSCRIPTION_ID	Updated 1 minute ago
AZURE_TENANT_ID	Updated 1 minute ago

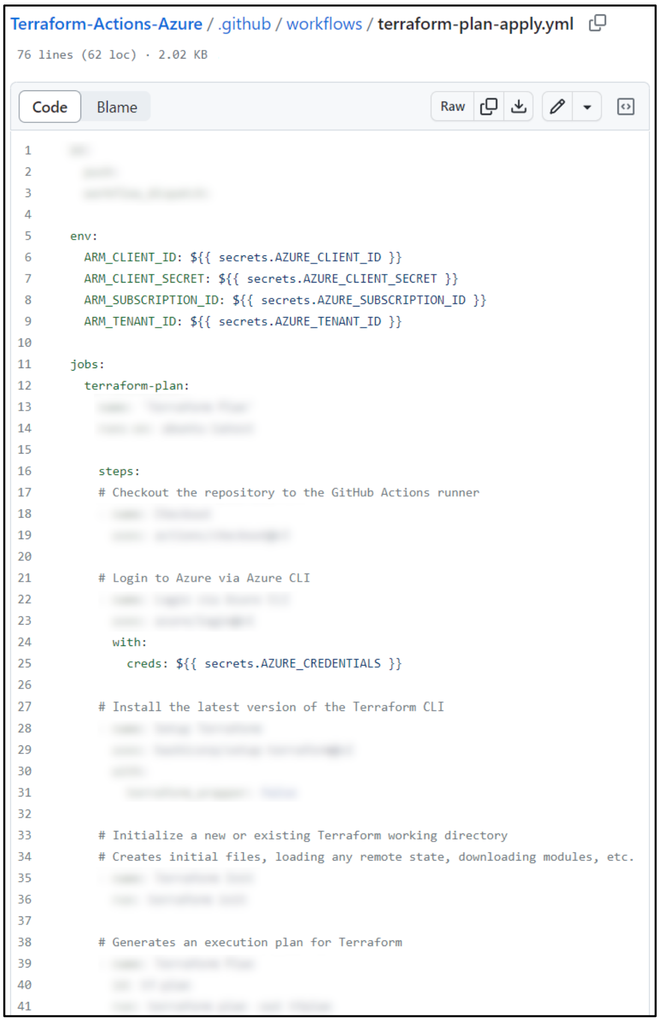
"**AZURE_CREDENTIALS**" should **contain the whole JSON that we copied earlier** and the **rest of the variables** should contain **only the corresponding parts** of it (only the **value**, without quotes "").

Now we are ready to write the **GitHub workflow** that uses these secrets.

Step 3: Write the Workflow

Finally, let's **write the workflow** that will consist of **2 jobs** – the first one will **create the Terraform plan** and the **second one will apply it**.

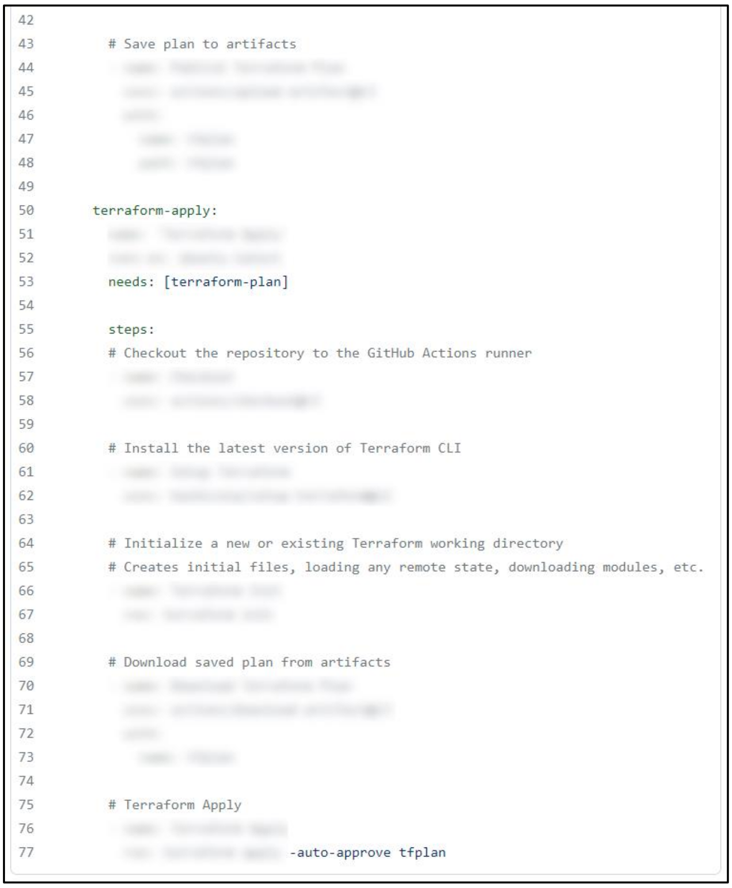
Write the workflow in this way:



Terraform-Actions-Azure / .github / workflows / terraform-plan-apply.yml

76 lines (62 loc) · 2.02 KB

```
env:
  ARM_CLIENT_ID: ${{ secrets.AZURE_CLIENT_ID }}
  ARM_CLIENT_SECRET: ${{ secrets.AZURE_CLIENT_SECRET }}
  ARM_SUBSCRIPTION_ID: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
  ARM_TENANT_ID: ${{ secrets.AZURE_TENANT_ID }}

jobs:
  terraform-plan:

    steps:
    # Checkout the repository to the GitHub Actions runner

    # Login to Azure via Azure CLI
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    # Install the latest version of the Terraform CLI

    # Initialize a new or existing Terraform working directory
    # Creates initial files, loading any remote state, downloading modules, etc.

    # Generates an execution plan for Terraform
```



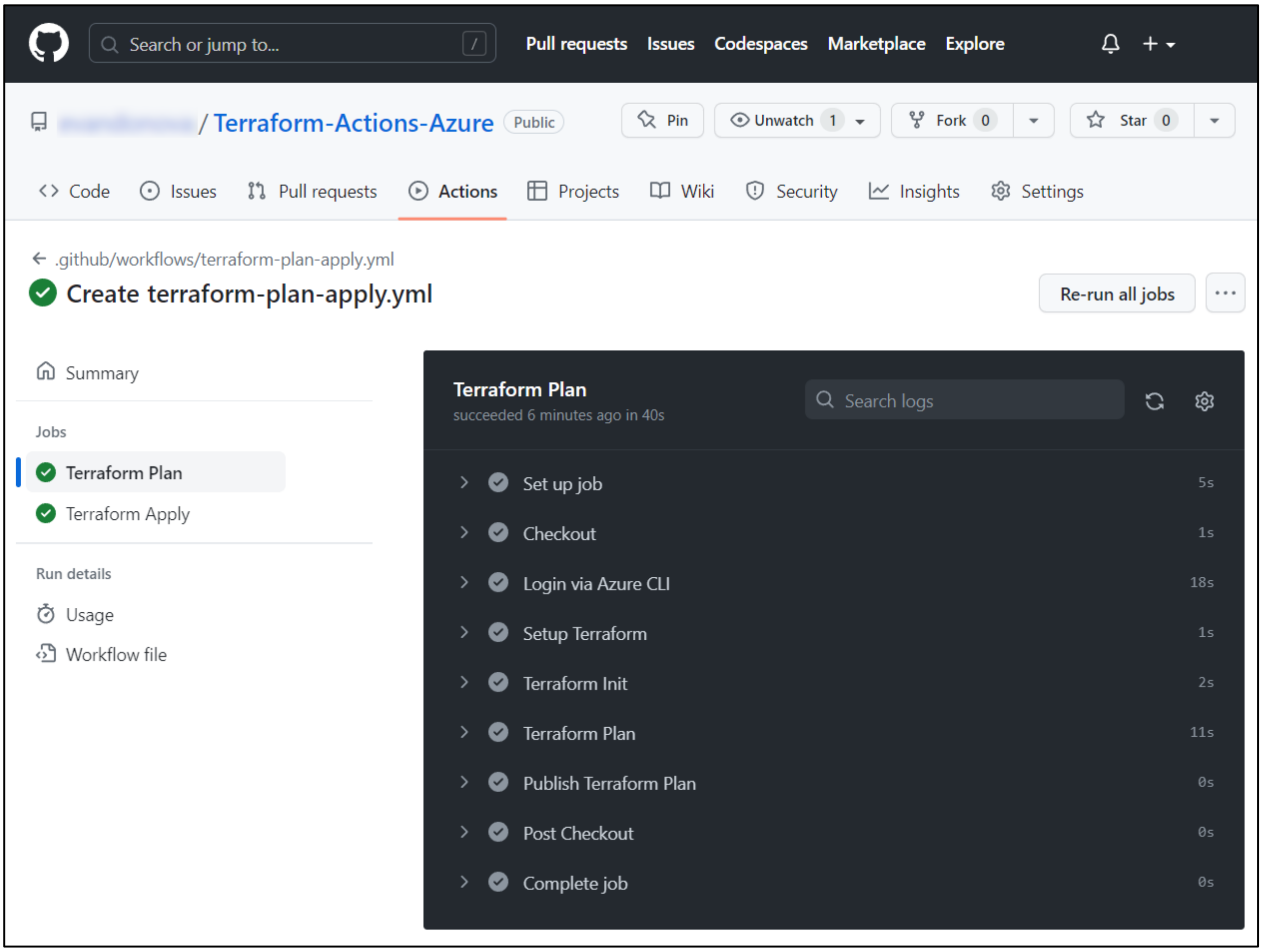
```
    # Save plan to artifacts

  terraform-apply:
    needs: [terraform-plan]

    steps:
    # Checkout the repository to the GitHub Actions runner

    # Install the latest version of Terraform CLI

    # Initialize a new or existing Terraform working directory
    # Creates initial files, loading any remote state, downloading modules, etc.

    # Download saved plan from artifacts

    # Terraform Apply
                                -auto-approve tfplan
```

You can use the **steps from the test workflow** we created earlier as part of **this YAML file**.

Note some **specific things** about this **workflow**:

- You need some **environment variables** so that **Terraform can authenticate in Azure**.
- You should use the "**AZURE_CREDENTIALS**" **GitHub secret** to **authenticate GitHub Actions in Azure**.
- The **second job** should **depend on the execution** of the **first one**.
- You should **add the** "**-auto-approve tfplan**" **flag** to **automatically approve the changes** in the "**tfplan**" without requiring manual confirmation during the workflow run.

The **workflow should run successfully**:



[redacted] / Terraform-Actions-Azure Public

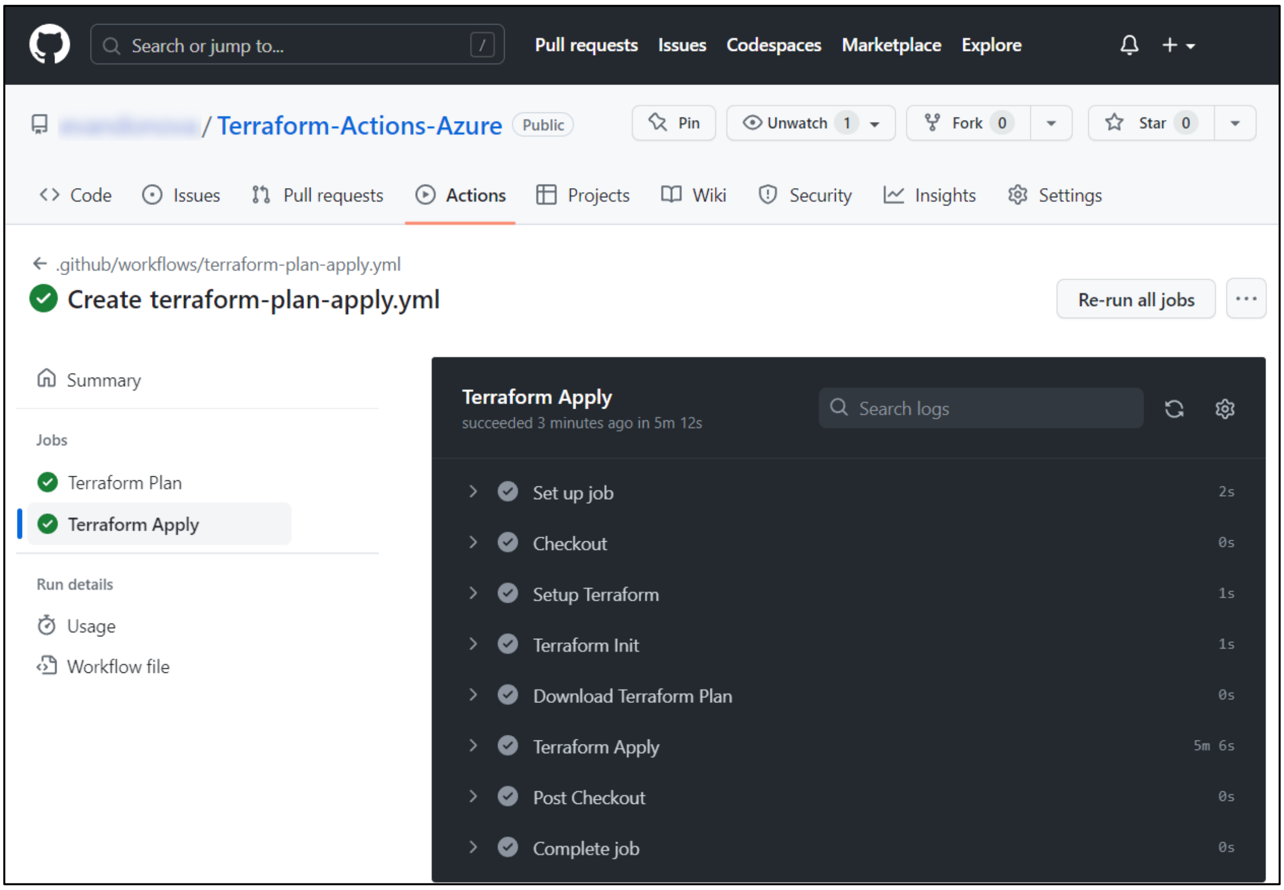
.github/workflows/terraform-plan-apply.yml

Create terraform-plan-apply.yml

Jobs
- Terraform Plan
- Terraform Apply

Terraform Plan
succeeded 6 minutes ago in 40s

```
Set up job               5s
Checkout                 1s
Login via Azure CLI     18s
Setup Terraform          1s
Terraform Init           2s
Terraform Plan          11s
Publish Terraform Plan   0s
Post Checkout            0s
Complete job             0s
```



[redacted] / Terraform-Actions-Azure Public

.github/workflows/terraform-plan-apply.yml

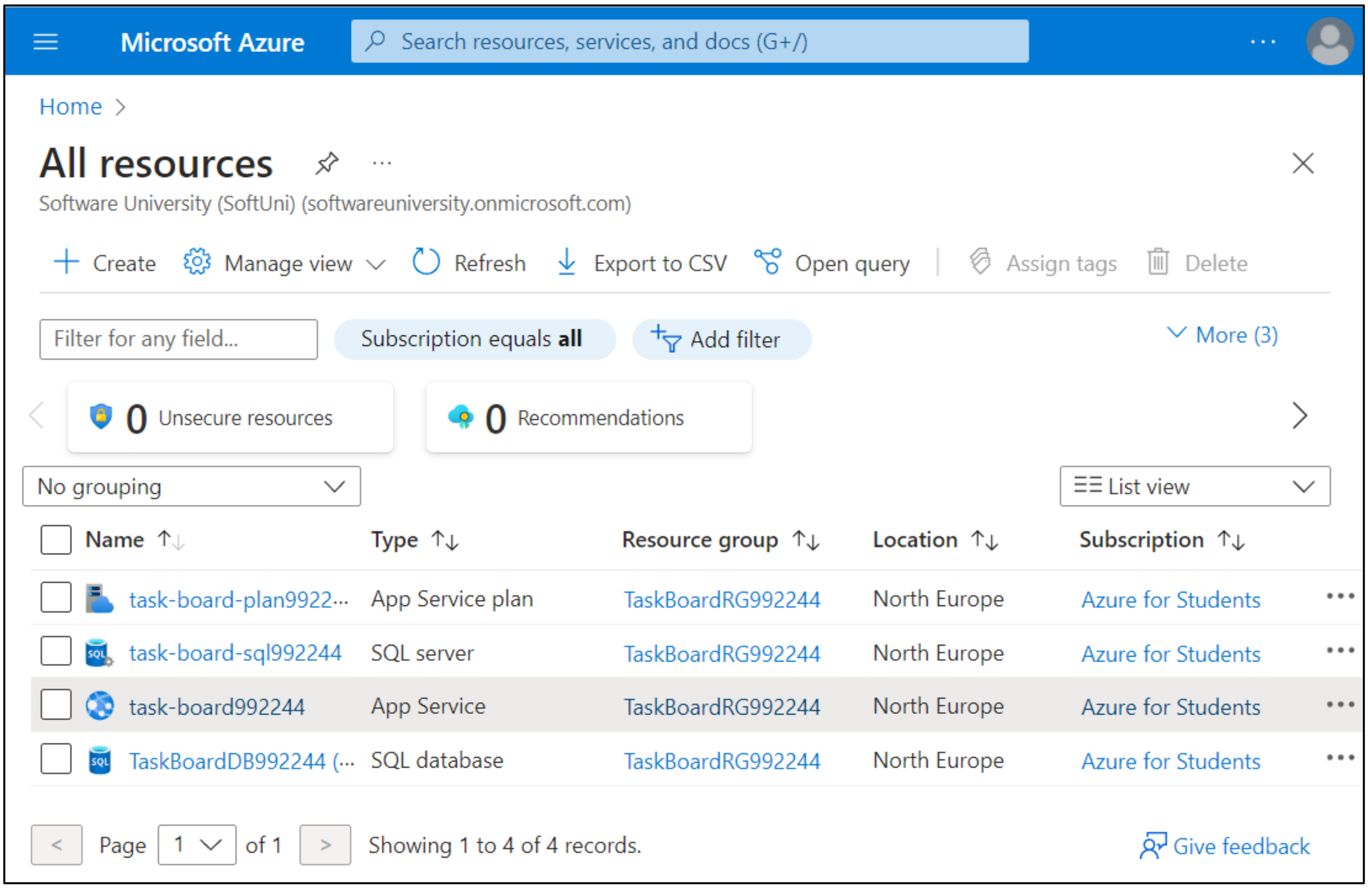
Create terraform-plan-apply.yml

Jobs
- Terraform Plan
- Terraform Apply

Terraform Apply
succeeded 3 minutes ago in 5m 12s

```
Set up job                 2s
Checkout                   0s
Setup Terraform            1s
Terraform Init             1s
Download Terraform Plan    0s
Terraform Apply         5m 6s
Post Checkout              0s
Complete job               0s
```

Also, the **Azure resources** you defined in the **Terraform configuration** should be **provisioned** and the "**TaskBoard**" **app deployed and working**:



All resources

Software University (SoftUni) (softwareuniversity.onmicrosoft.com)

Filter for any field...

Subscription equals **all**

0 Unsecure resources

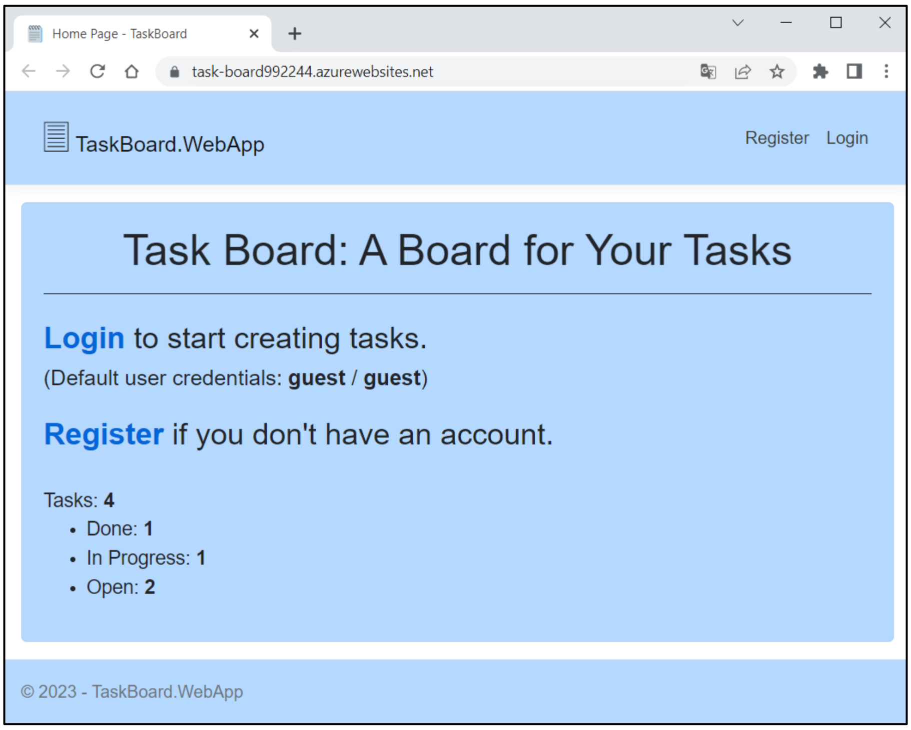
0 Recommendations

No grouping

Name	Type	Resource group	Location	Subscription
task-board-plan9922...	App Service plan	TaskBoardRG992244	North Europe	Azure for Students
task-board-sql992244	SQL server	TaskBoardRG992244	North Europe	Azure for Students
task-board992244	App Service	TaskBoardRG992244	North Europe	Azure for Students
TaskBoardDB992244 (...	SQL database	TaskBoardRG992244	North Europe	Azure for Students

Page 1 of 1 Showing 1 to 4 of 4 records.

Give feedback



Task Board: A Board for Your Tasks

Login to start creating tasks.

(Default user credentials: **guest** / **guest**)

Register if you don't have an account.

Tasks: **4**

- Done: **1**
- In Progress: **1**
- Open: **2**

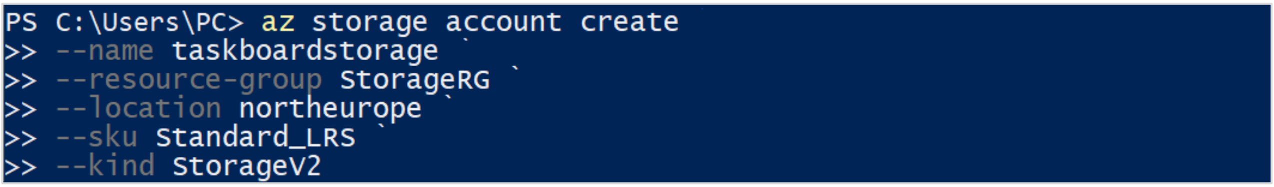
© 2023 - TaskBoard.WebApp

We successfully used **GitHub Actions** to **run a Terraform configuration** that **provisions resources in Azure**. However, if we **change the configuration** and **run the workflow again**, an **error will occur**. This happens because we **don't save the Terraform configuration state file**.

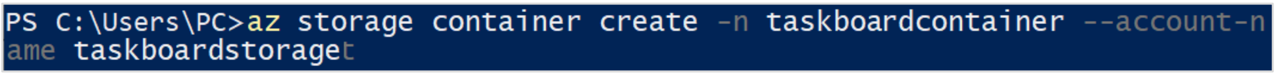
Store State File in Azure Storage Account

Terraform utilizes a **state file** to **store information** about the **current state of your managed infrastructure** and associated configuration. This file will need to be **persisted between different runs of the workflow**.

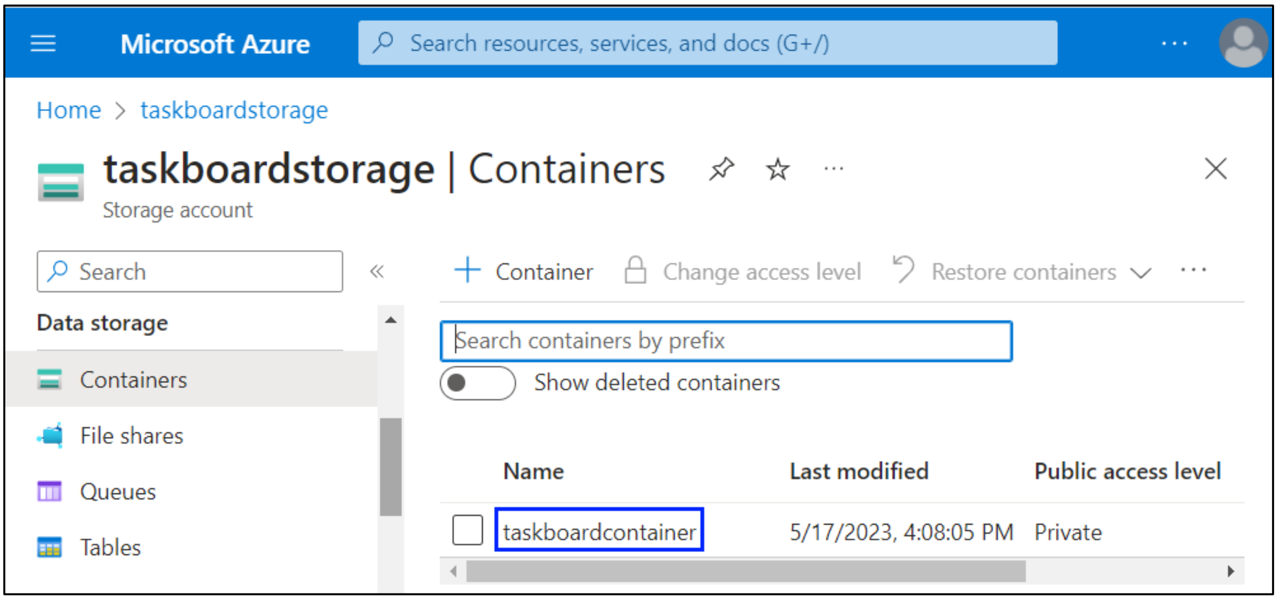
The recommended approach is to **store this file** within an **`Azure Storage Account`** and this is what we will do now. First, you should **create an Azure storage account** with a **container** to **store the state file**:



```
PS C:\Users\PC> az storage account create `
>> --name taskboardstorage `
>> --resource-group StorageRG `
>> --location northeurope `
>> --sku Standard_LRS `
>> --kind StorageV2
```

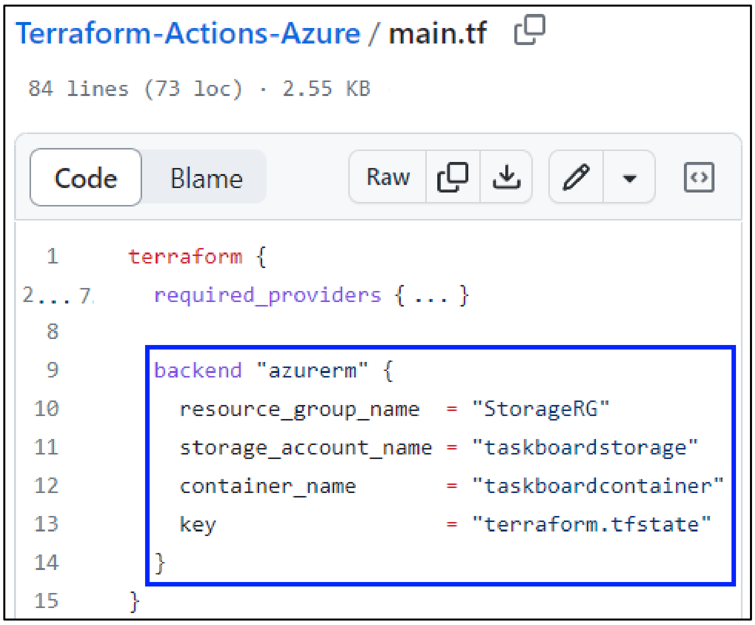


```
PS C:\Users\PC>az storage container create -n taskboardcontainer --account-n
ame taskboardstoraget
```



Name	Last modified	Public access level
□ taskboardcontainer	5/17/2023, 4:08:05 PM	Private

Then, to **use this storage in Terraform**, you should **add a backend block** in the **`main.tf` configuration file**:



Terraform-Actions-Azure / main.tf
84 lines (73 loc) · 2.55 KB
Code
Blame
Raw

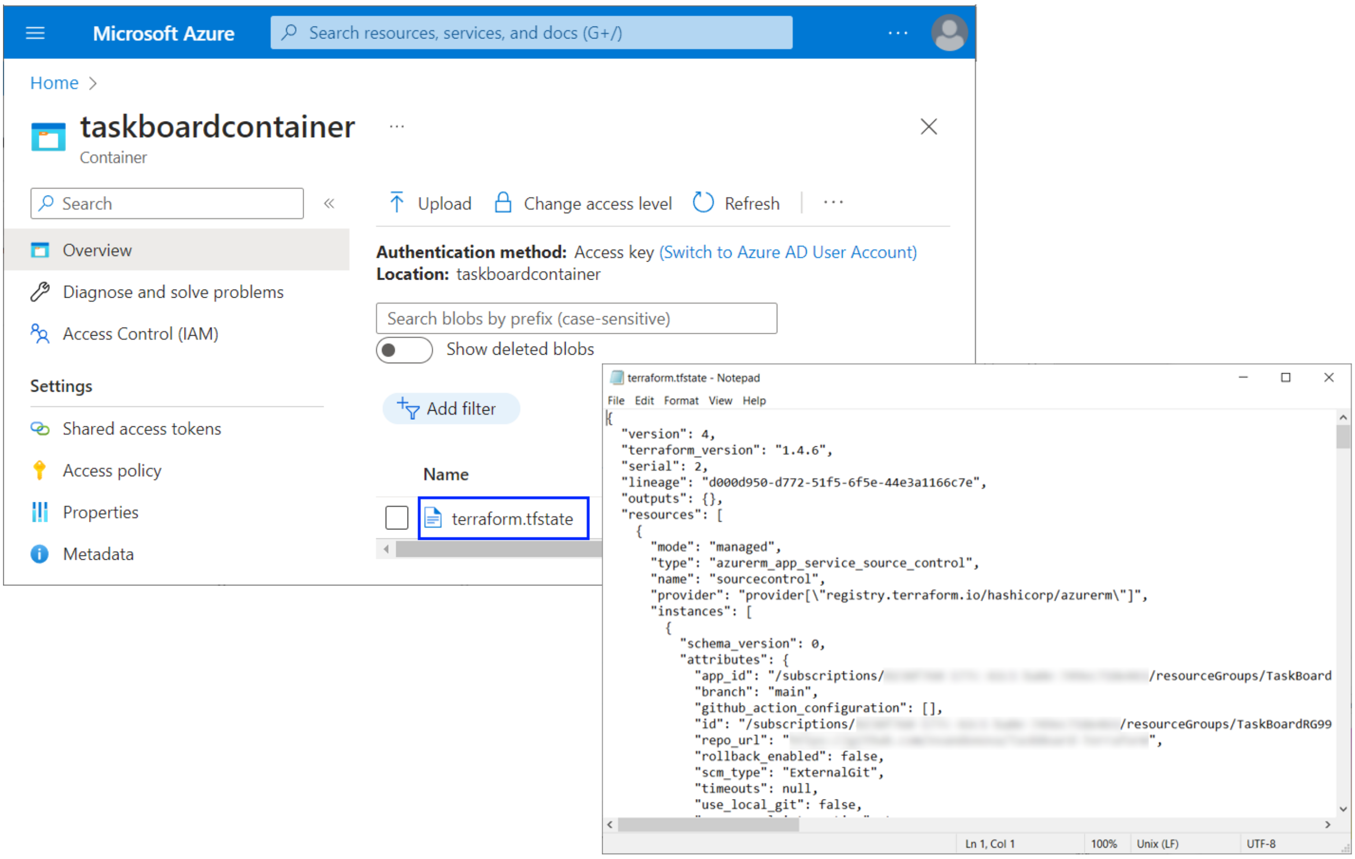
```
terraform {
  required_providers { ... }

  backend "azurerm" {
    resource_group_name  = "StorageRG"
    storage_account_name = "taskboardstorage"
    container_name       = "taskboardcontainer"
    key                  = "terraform.tfstate"
  }
}
```

A **backend block** defines where **Terraform stores its state data files**. You should provide the **names of your resource group**, **storage account** and **container**, as well as to set a **name of the state file** that will be created.

Commit the changed file to GitHub and wait for **GitHub Actions to run the workflow**.

Note: you **should not have your resources in Azure now** or the **GitHub Actions workflow will still give you an error** when **trying to create them**, as they are **not defined in the state file. Delete the resources** you created previously with your Terraform configuration from **Azure**.

The **workflow should be successful** and you should see that a **`terraform.tfstate` file** was created in your **Azure storage container**:



Authentication method: Access key (Switch to Azure AD User Account)
Location: taskboardcontainer

```
{
  "version": 4,
  "terraform_version": "1.4.6",
  "serial": 2,
  "lineage": "d000d950-d772-51f5-6f5e-44e3a1166c7e",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "azurerm_app_service_source_control",
      "name": "sourcecontrol",
      "provider": "provider[\"registry.terraform.io/hashicorp/azurerm\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "app_id": "/subscriptions/[redacted]/resourceGroups/TaskBoard
            "branch": "main",
            "github_action_configuration": [],
            "id": "/subscriptions/[redacted]/resourceGroups/TaskBoardRG99
            "repo_url": "[redacted]",
            "rollback_enabled": false,
            "scm_type": "ExternalGit",
            "timeouts": null,
            "use_local_git": false,
```

Go and **make a change in your Terraform configuration in GitHub** and **run the workflow again** – the **modified resources** should be **updated successfully in Azure**.

Now you have a **fully working `GitHub Actions` + `Terraform` + `Azure` configuration** to create and manage resources.

* More Configuration Improvements

We have a **good Terraform configuration** and **GitHub Actions workflows** created during the previous tasks but here are some **additional challenges for you** to overcome to **improve your Terraform skills** even more:

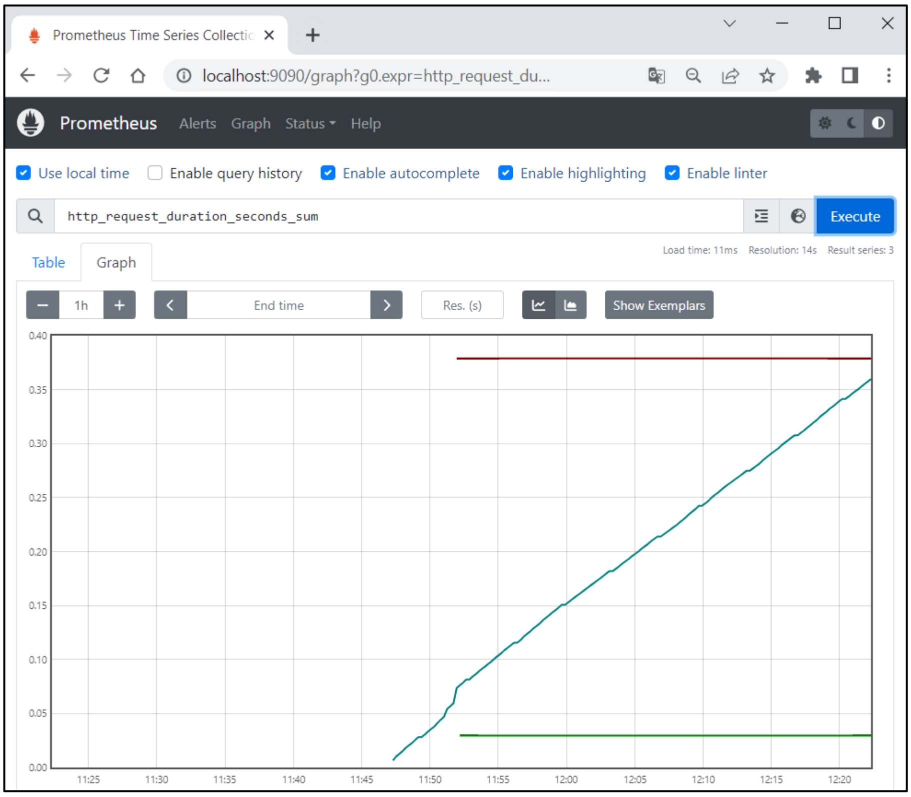
- You can **create a Terraform configuration file** to **provision an Azure storage account and container** for the **Terraform backend**, instead of doing it with commands like we did previously. Then, you can use a **GitHub Actions workflow** to **run that configuration** and **provision the resources in Azure**.
- You can **create a Terraform configuration file** to **create the service principal** and **assign the "`Contributor`" role to it** instead of doing it manually or with commands through Azure CLI. You can again try to **run the configuration in GitHub Actions**, not only locally.

By **fulfilling these additional tasks**, you would have **fully explored** and used the **integration between `Terraform`, `GitHub Actions`** and **`Azure`**.

II. App Monitoring

5. 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**.



☑ Use local time □ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

```
http_request_duration_seconds_sum
```

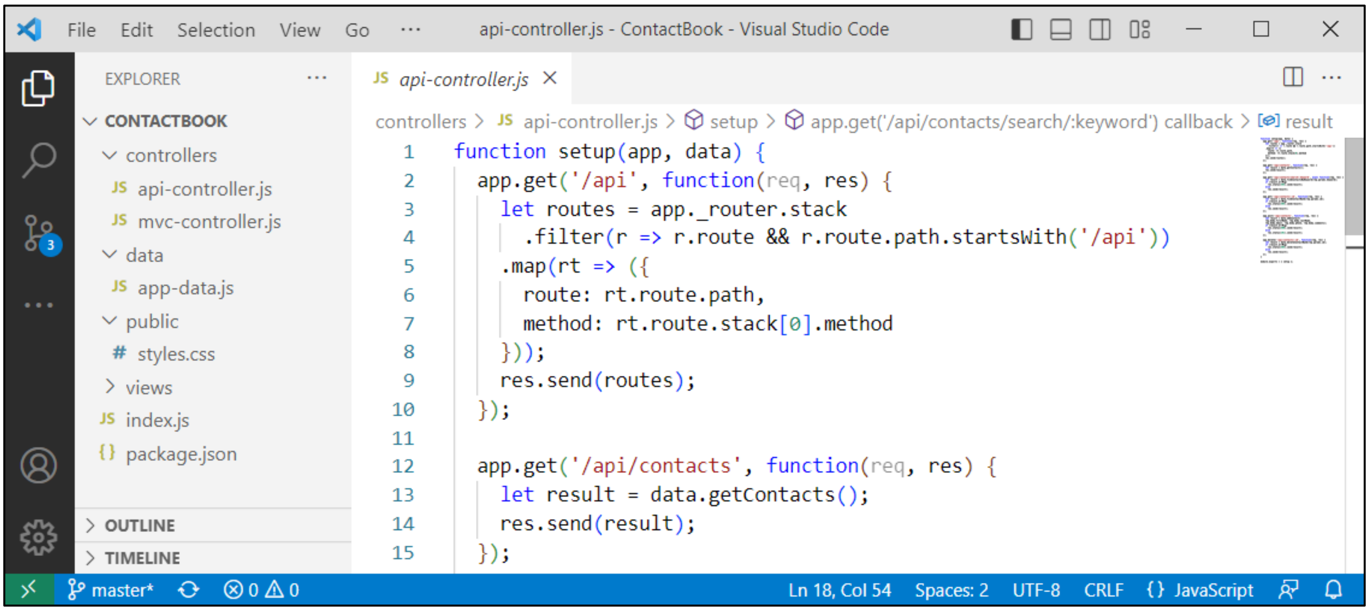
Execute

Load time: 11ms Resolution: 14s Result series: 3

Step 1: Examine the App

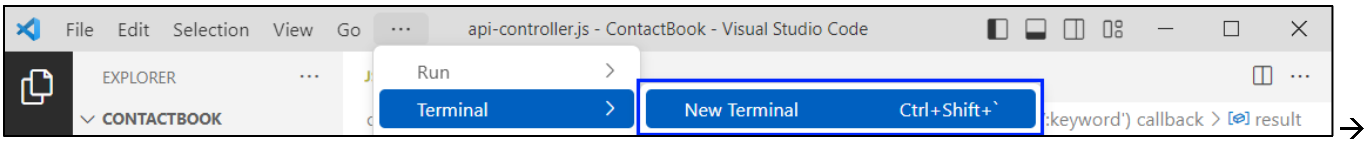
We have the "**Contact Book**" **Node.js app**, which holds a **searchable list of contacts**. You have pages to **list all contacts** (**`/contacts`**), **view a single contact** (**`/contacts/:id`**), **search for a contact** (**`/contacts/search/:keyword`**) and **add a new contact** (**`/contacts/create`**).

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

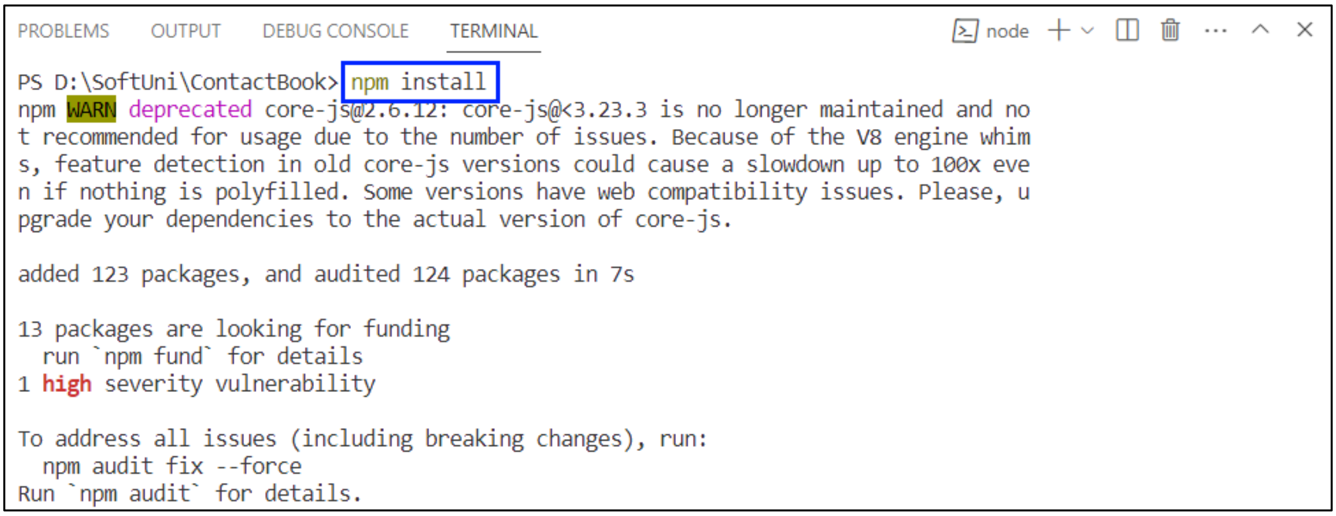


```
function setup(app, data) {
  app.get('/api', function(req, res) {
    let routes = app._router.stack
      .filter(r => r.route && r.route.path.startsWith('/api'))
    .map(rt => ({
      route: rt.route.path,
      method: rt.route.stack[0].method
    }));
    res.send(routes);
  });

  app.get('/api/contacts', function(req, res) {
    let result = data.getContacts();
    res.send(result);
  });
```

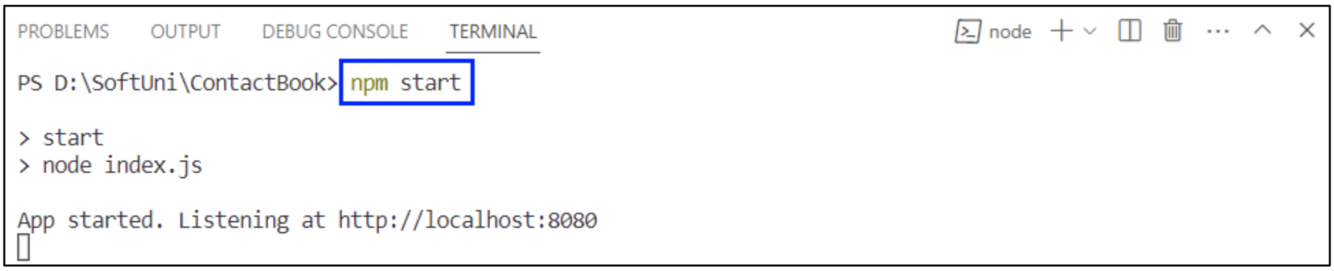
Open a **terminal** and **execute** the "**`npm install`**" and "**`npm start`**" **commands** to **run the app**:



File
Edit
Selection
View
Go
...
api-controller.js - ContactBook - Visual Studio Code
EXPLORER
CONTACTBOOK
Run
Terminal
New Terminal Ctrl+Shift+`
:keyword') callback > result

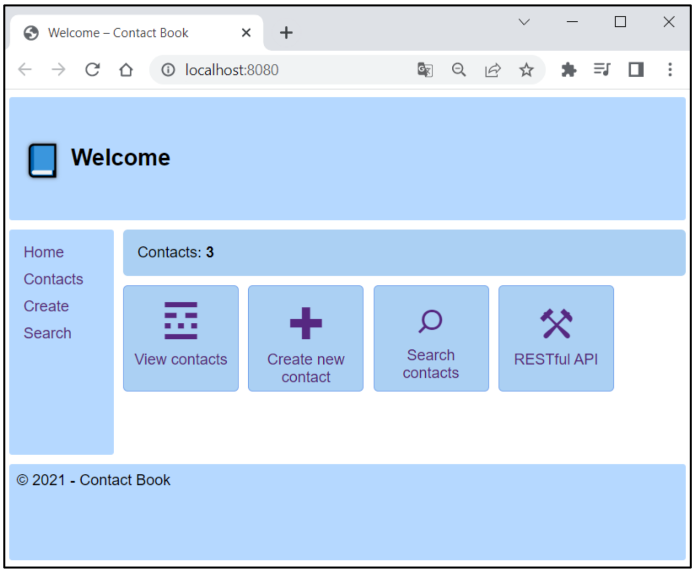


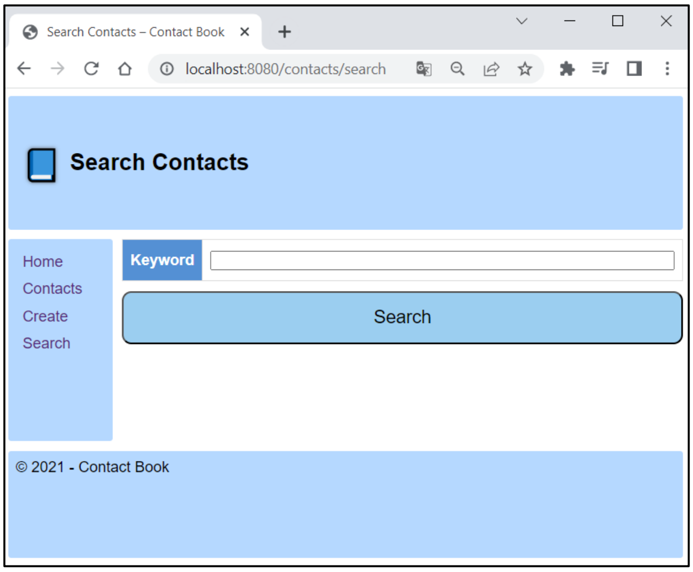
```
PS D:\SoftUni\ContactBook> npm install
npm WARN deprecated core-js@2.6.12: core-js@<3.23.3 is no longer maintained and no
t recommended for usage due to the number of issues. Because of the V8 engine whim
s, feature detection in old core-js versions could cause a slowdown up to 100x eve
n if nothing is polyfilled. Some versions have web compatibility issues. Please, u
pgrade your dependencies to the actual version of core-js.

added 123 packages, and audited 124 packages in 7s

13 packages are looking for funding
  run `npm fund` for details
1 high severity vulnerability

To address all issues (including breaking changes), run:
  npm audit fix --force
Run `npm audit` for details.
```

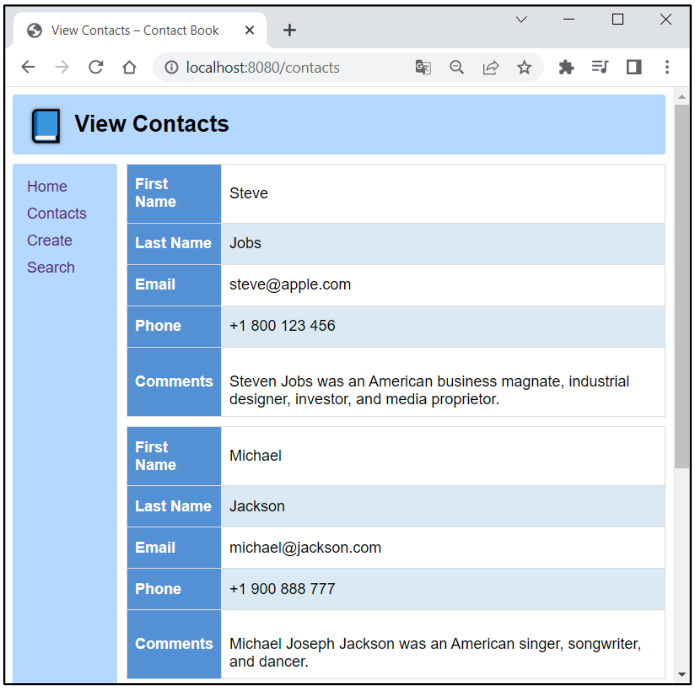


```
PS D:\SoftUni\ContactBook> npm start

> start
> node index.js

App started. Listening at http://localhost:8080
```

Look at the **app pages** on http://localhost:8080:



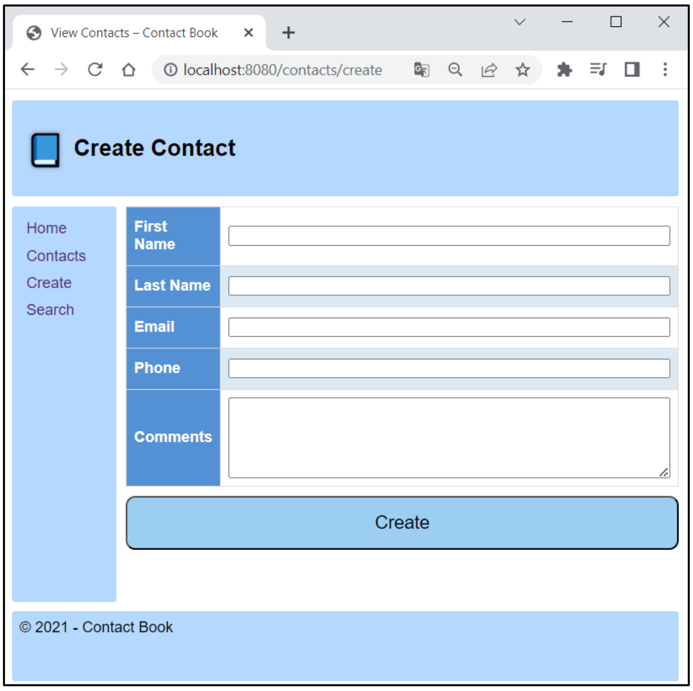




View Contacts

First Name	Steve
Last Name	Jobs
Email	steve@apple.com
Phone	+1 800 123 456
Comments	Steven Jobs was an American business magnate, industrial designer, investor, and media proprietor.

First Name	Michael
Last Name	Jackson
Email	michael@jackson.com
Phone	+1 900 888 777
Comments	Michael Joseph Jackson was an American singer, songwriter, and dancer.



Create Contact

First Name	
Last Name	
Email	
Phone	
Comments	

© 2021 - Contact Book

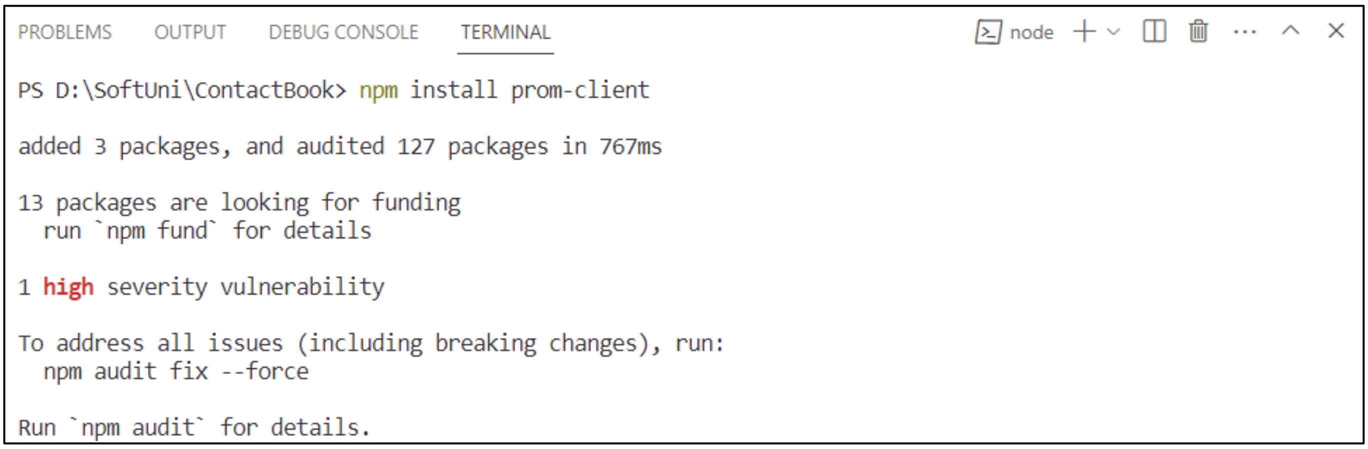
Let's now see how to **modify the app code** to **export app metrics** for **Prometheus**.

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.

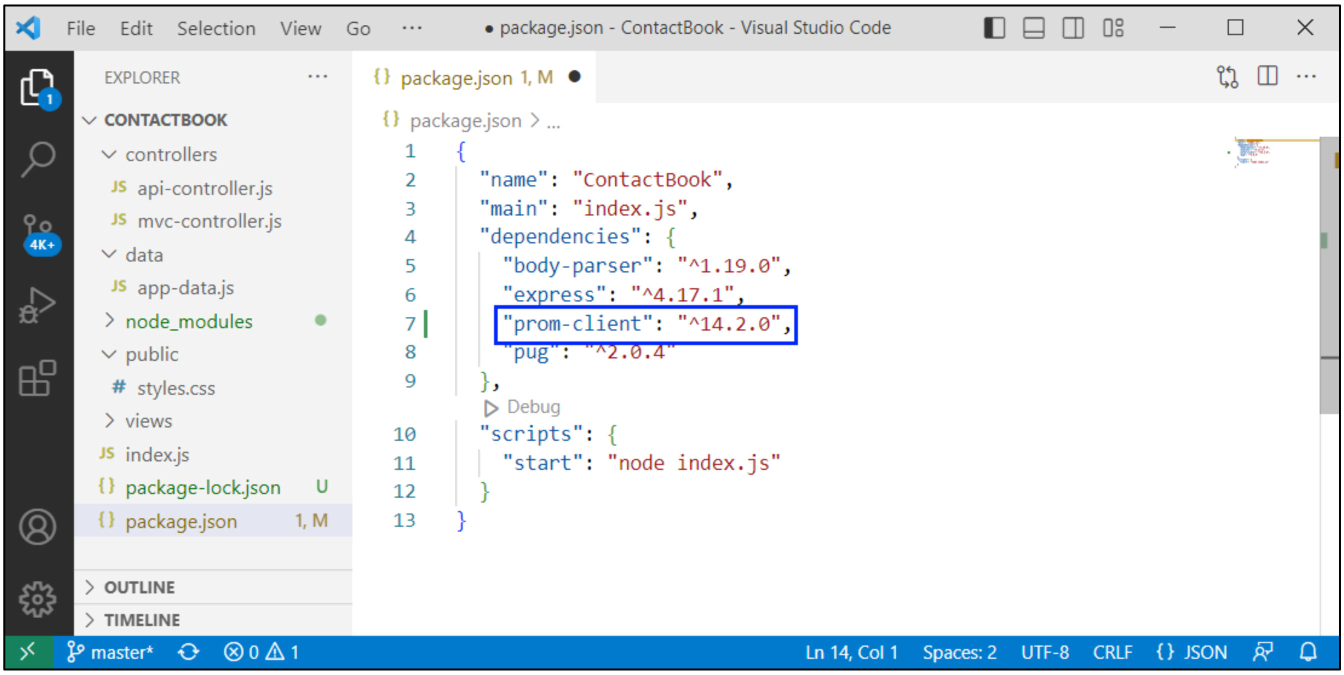
Install Prom-Client

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**:



```
PS D:\SoftUni\ContactBook> npm install prom-client

added 3 packages, and audited 127 packages in 767ms

13 packages are looking for funding
  run `npm fund` for details

1 high severity vulnerability

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

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



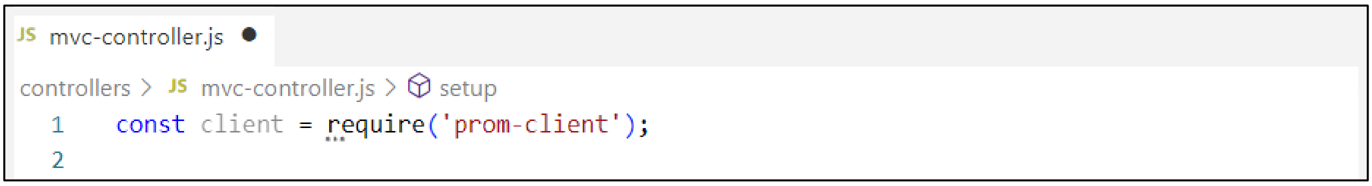
```json
{
  "name": "ContactBook",
  "main": "index.js",
  "dependencies": {
    "body-parser": "^1.19.0",
    "express": "^4.17.1",
    "prom-client": "^14.2.0",
    "pug": "^2.0.4"
  },
  "scripts": {
    "start": "node index.js"
  }
}
```

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

Export Default Metrics

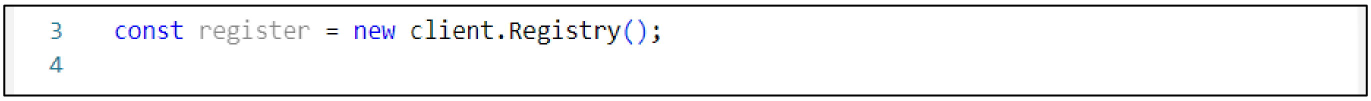
Now we will **modify our code** to **collect the default app metrics** together with **some custom ones** and **expose them** on the **`/metrics` endpoint**.

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:



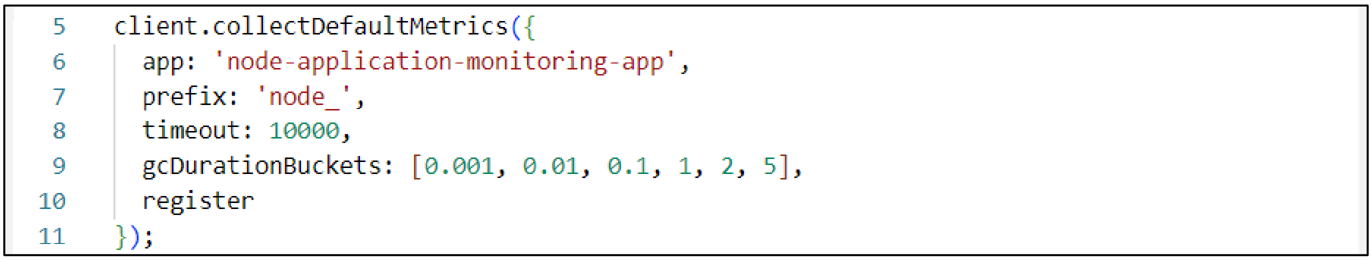
```javascript
const client = require('prom-client');

```

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



```javascript
const register = new client.Registry();

```

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:



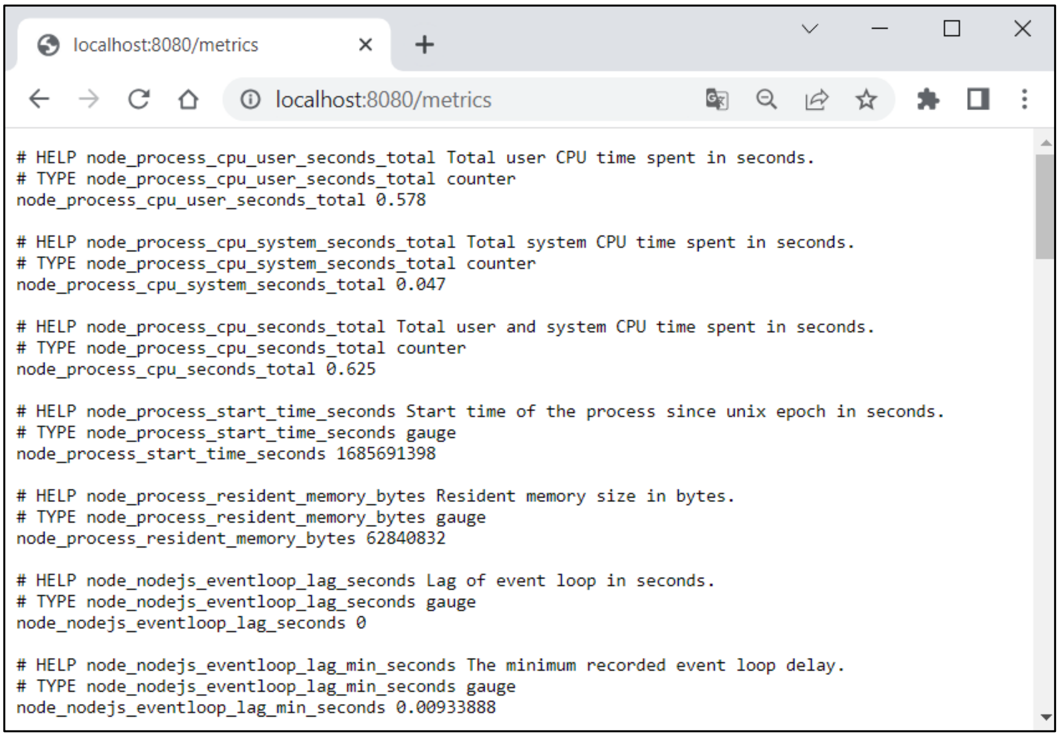
```javascript
client.collectDefaultMetrics({
  app: 'node-application-monitoring-app',
  prefix: 'node_',
  timeout: 10000,
  gcDurationBuckets: [0.001, 0.01, 0.1, 1, 2, 5],
  register
});
```

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**:

```
function setup(app, data) {
  app.get('/metrics', async (req, res) => {
    res.setHeader('Content-Type', register.contentType);
    res.send(await register.metrics());
  });

```

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:



```
# HELP node_process_cpu_user_seconds_total Total user CPU time spent in seconds.
# TYPE node_process_cpu_user_seconds_total counter
node_process_cpu_user_seconds_total 0.578

# HELP node_process_cpu_system_seconds_total Total system CPU time spent in seconds.
# TYPE node_process_cpu_system_seconds_total counter
node_process_cpu_system_seconds_total 0.047

# HELP node_process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE node_process_cpu_seconds_total counter
node_process_cpu_seconds_total 0.625

# HELP node_process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE node_process_start_time_seconds gauge
node_process_start_time_seconds 1685691398

# HELP node_process_resident_memory_bytes Resident memory size in bytes.
# TYPE node_process_resident_memory_bytes gauge
node_process_resident_memory_bytes 62840832

# HELP node_nodejs_eventloop_lag_seconds Lag of event loop in seconds.
# TYPE node_nodejs_eventloop_lag_seconds gauge
node_nodejs_eventloop_lag_seconds 0

# HELP node_nodejs_eventloop_lag_min_seconds The minimum recorded event loop delay.
# TYPE node_nodejs_eventloop_lag_min_seconds gauge
node_nodejs_eventloop_lag_min_seconds 0.00933888
```

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):

```
const httpRequestTimer = new client.Histogram({
  name: 'http_request_duration_seconds',
  help: 'Duration of HTTP requests in seconds',
  labelNames: ['method', 'route', 'code'],
  buckets: [0.01, 0.03, 0.05, 0.07, 0.1, 0.3, 0.5, 0.7, 1]
});

```

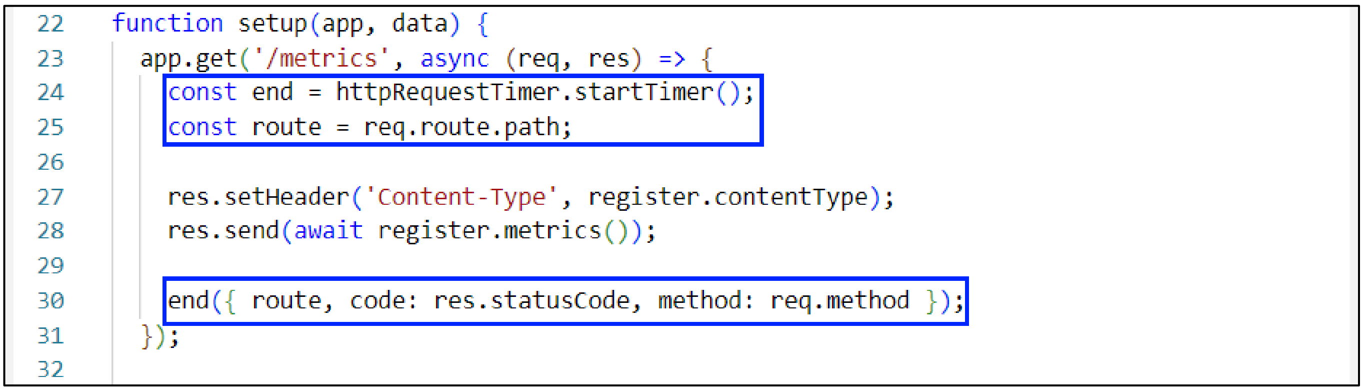
Then we should **register the metric**:

```
register.registerMetric(httpRequestTimer);

```

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**

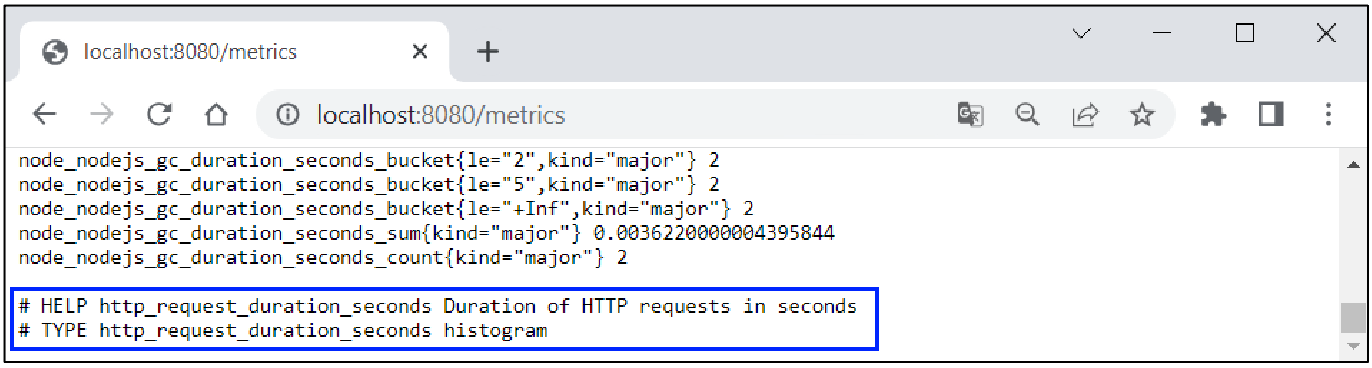
In this way, the **HTTP request data and duration** will be recorded. Do it for the **`/metrics` endpoint** like this:



```
function setup(app, data) {
  app.get('/metrics', async (req, res) => {
    const end = httpRequestTimer.startTimer();
    const route = req.route.path;

    res.setHeader('Content-Type', register.contentType);
    res.send(await register.metrics());

    end({ route, code: res.statusCode, method: req.method });
  });

```

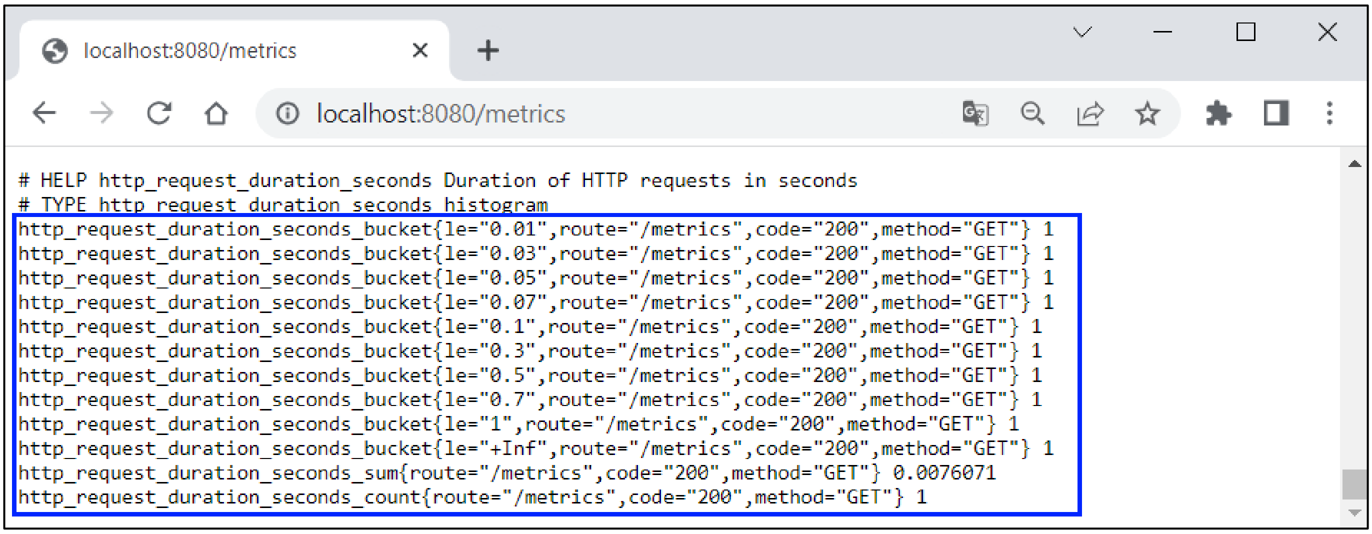
Do it for the **rest of the endpoint methods** in the same way by **adding the above three lines**. When ready, **run the app** again.

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



```
node_nodejs_gc_duration_seconds_bucket{le="2",kind="major"} 2
node_nodejs_gc_duration_seconds_bucket{le="5",kind="major"} 2
node_nodejs_gc_duration_seconds_bucket{le="+Inf",kind="major"} 2
node_nodejs_gc_duration_seconds_sum{kind="major"} 0.0036220000004395844
node_nodejs_gc_duration_seconds_count{kind="major"} 2

# HELP http_request_duration_seconds Duration of HTTP requests in seconds
# TYPE http_request_duration_seconds histogram
```

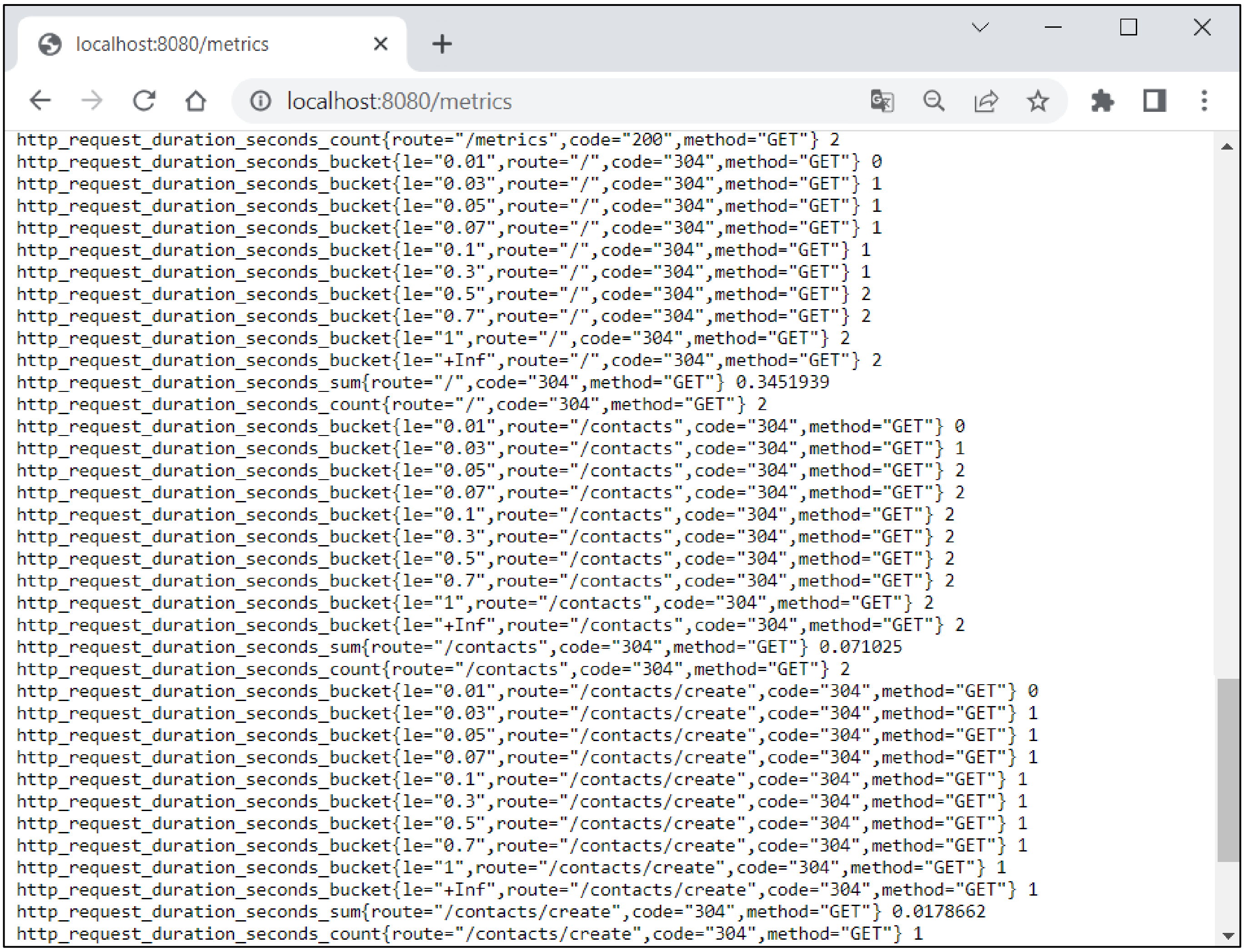
However, you still have **no metric values**. You should **refresh the page**, so that the metrics for the **previous HTTP request** to **`/metrics`** are displayed:



```
# HELP http_request_duration_seconds Duration of HTTP requests in seconds
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{le="0.01",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="0.03",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="0.05",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="0.07",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="0.1",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="0.3",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="0.5",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="0.7",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="1",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_bucket{le="+Inf",route="/metrics",code="200",method="GET"} 1
http_request_duration_seconds_sum{route="/metrics",code="200",method="GET"} 0.0076071
http_request_duration_seconds_count{route="/metrics",code="200",method="GET"} 1
```

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**:

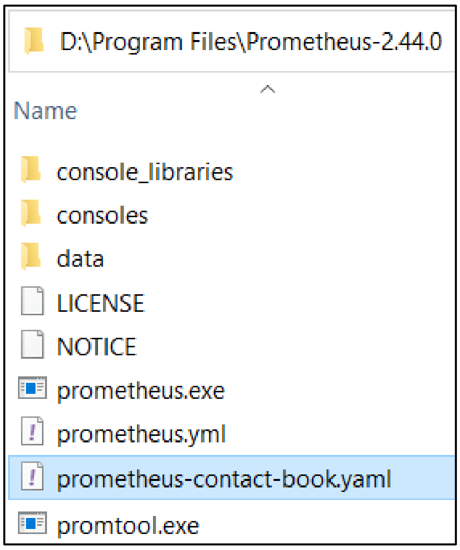


```
http_request_duration_seconds_count{route="/metrics",code="200",method="GET"} 2
http_request_duration_seconds_bucket{le="0.01",route="/",code="304",method="GET"} 0
http_request_duration_seconds_bucket{le="0.03",route="/",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.05",route="/",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.07",route="/",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.1",route="/",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.3",route="/",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.5",route="/",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.7",route="/",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="1",route="/",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="+Inf",route="/",code="304",method="GET"} 2
http_request_duration_seconds_sum{route="/",code="304",method="GET"} 0.3451939
http_request_duration_seconds_count{route="/",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.01",route="/contacts",code="304",method="GET"} 0
http_request_duration_seconds_bucket{le="0.03",route="/contacts",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.05",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.07",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.1",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.3",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.5",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.7",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="1",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="+Inf",route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_sum{route="/contacts",code="304",method="GET"} 0.071025
http_request_duration_seconds_count{route="/contacts",code="304",method="GET"} 2
http_request_duration_seconds_bucket{le="0.01",route="/contacts/create",code="304",method="GET"} 0
http_request_duration_seconds_bucket{le="0.03",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.05",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.07",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.1",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.3",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.5",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="0.7",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="1",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_bucket{le="+Inf",route="/contacts/create",code="304",method="GET"} 1
http_request_duration_seconds_sum{route="/contacts/create",code="304",method="GET"} 0.0178662
http_request_duration_seconds_count{route="/contacts/create",code="304",method="GET"} 1
```

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.

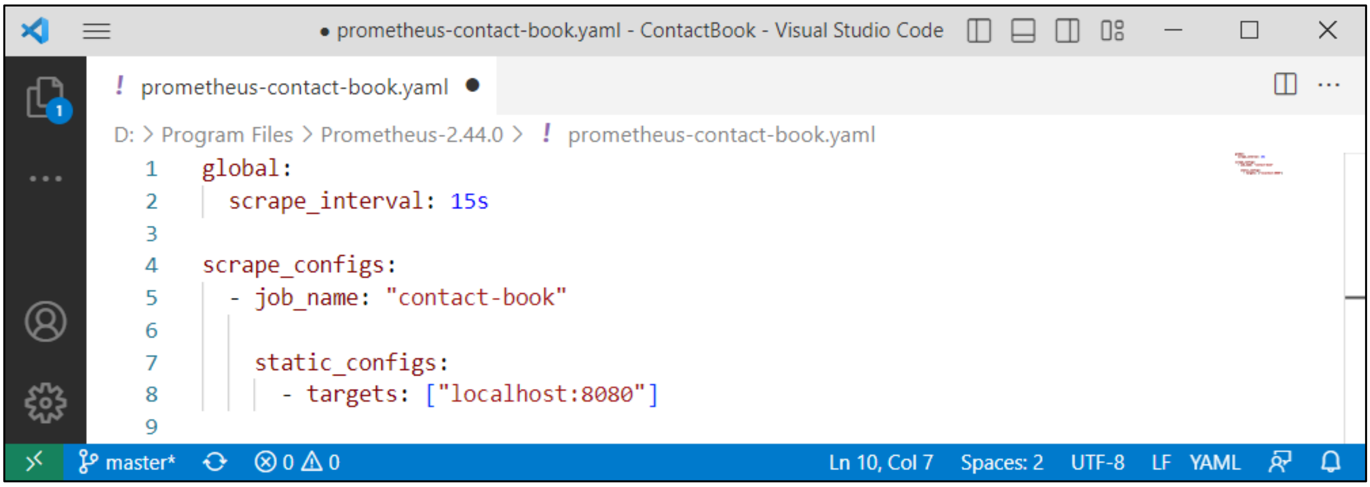
Step 3: Condifure and Run Prometheus

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** "`Contact` **Book**" **app**:



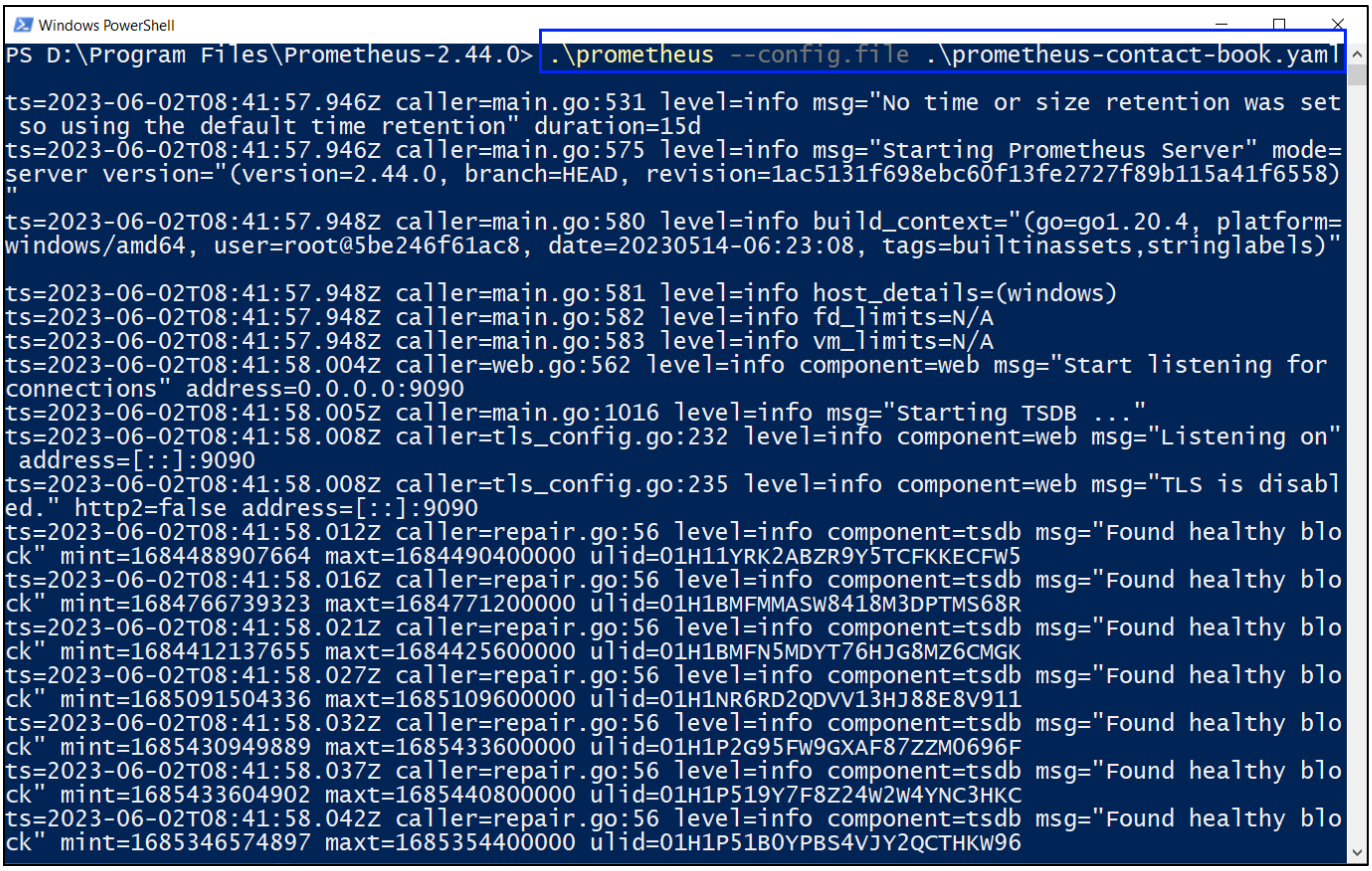
```
console_libraries
consoles
data
LICENSE
NOTICE
prometheus.exe
prometheus.yml
prometheus-contact-book.yaml
promtool.exe
```

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


● prometheus-contact-book.yaml - ContactBook - Visual Studio Code
prometheus-contact-book.yaml
D: > Program Files > Prometheus-2.44.0 > prometheus-contact-book.yaml
master*
⊗ 0 ⚠ 0
Ln 10, Col 7
Spaces: 2
UTF-8
LF
YAML

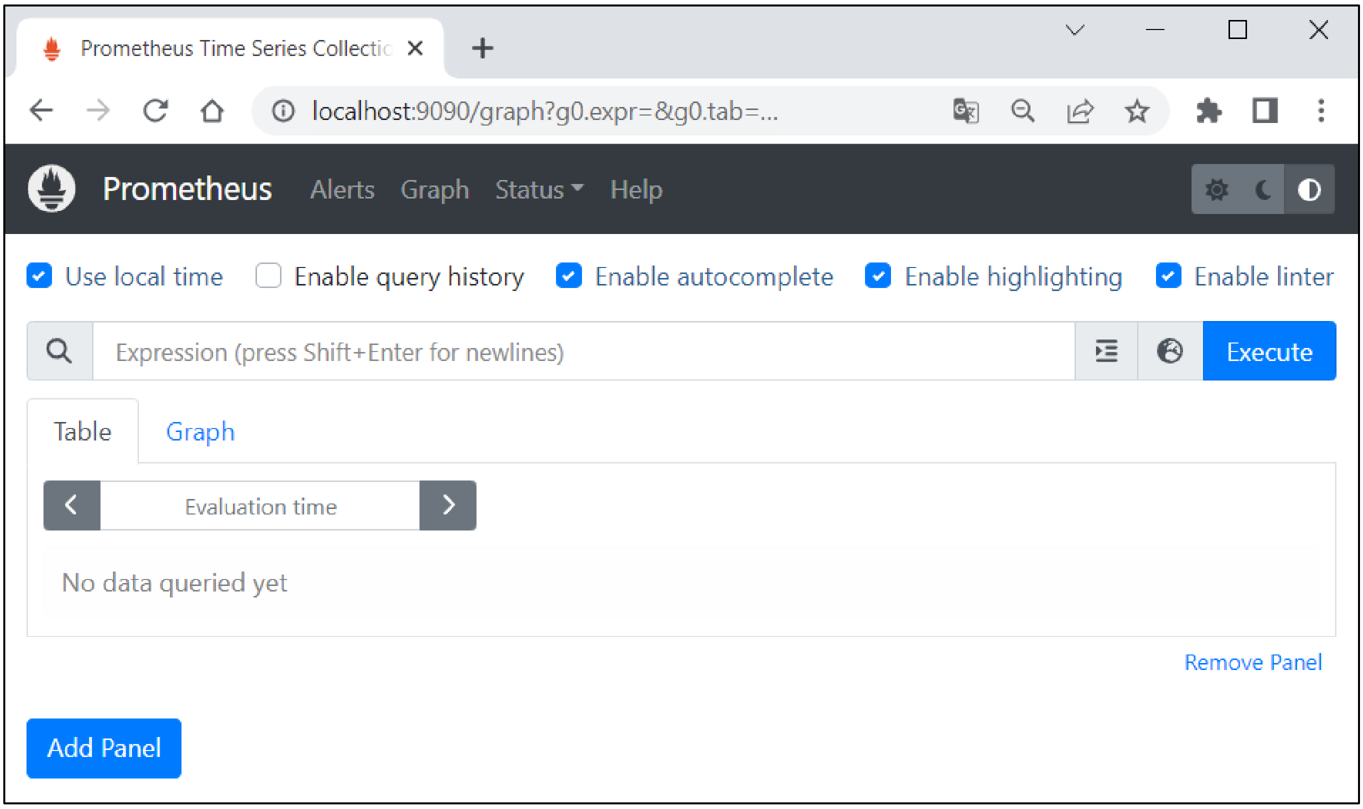
```yaml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: "contact-book"

    static_configs:
      - targets: ["localhost:8080"]

```

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



```
PS D:\Program Files\Prometheus-2.44.0> .\prometheus --config.file .\prometheus-contact-book.yaml

ts=2023-06-02T08:41:57.946Z caller=main.go:531 level=info msg="No time or size retention was set
 so using the default time retention" duration=15d
ts=2023-06-02T08:41:57.946Z caller=main.go:575 level=info msg="Starting Prometheus Server" mode=
server version="(version=2.44.0, branch=HEAD, revision=1ac5131f698ebc60f13fe2727f89b115a41f6558)
"
ts=2023-06-02T08:41:57.948Z caller=main.go:580 level=info build_context="(go=go1.20.4, platform=
windows/amd64, user=root@5be246f61ac8, date=20230514-06:23:08, tags=builtinassets,stringlabels)"

ts=2023-06-02T08:41:57.948Z caller=main.go:581 level=info host_details=(windows)
ts=2023-06-02T08:41:57.948Z caller=main.go:582 level=info fd_limits=N/A
ts=2023-06-02T08:41:57.948Z caller=main.go:583 level=info vm_limits=N/A
ts=2023-06-02T08:41:58.004Z caller=web.go:562 level=info component=web msg="Start listening for
connections" address=0.0.0.0:9090
ts=2023-06-02T08:41:58.005Z caller=main.go:1016 level=info msg="Starting TSDB ..."
ts=2023-06-02T08:41:58.008Z caller=tls_config.go:232 level=info component=web msg="Listening on"
 address=[::]:9090
ts=2023-06-02T08:41:58.008Z caller=tls_config.go:235 level=info component=web msg="TLS is disabl
ed." http2=false address=[::]:9090
ts=2023-06-02T08:41:58.012Z caller=repair.go:56 level=info component=tsdb msg="Found healthy blo
ck" mint=1684488907664 maxt=1684490400000 ulid=01H11YRK2ABZR9Y5TCFKKECFW5
ts=2023-06-02T08:41:58.016Z caller=repair.go:56 level=info component=tsdb msg="Found healthy blo
ck" mint=1684766739323 maxt=1684771200000 ulid=01H1BMFMMASW8418M3DPTMS68R
ts=2023-06-02T08:41:58.021Z caller=repair.go:56 level=info component=tsdb msg="Found healthy blo
ck" mint=1684412137655 maxt=1684425600000 ulid=01H1BMFN5MDYT76HJG8MZ6CMGK
ts=2023-06-02T08:41:58.027Z caller=repair.go:56 level=info component=tsdb msg="Found healthy blo
ck" mint=1685091504336 maxt=1685109600000 ulid=01H1NR6RD2QDVV13HJ88E8V911
ts=2023-06-02T08:41:58.032Z caller=repair.go:56 level=info component=tsdb msg="Found healthy blo
ck" mint=1685430949889 maxt=1685433600000 ulid=01H1P2G95FW9GXAF87ZZM0696F
ts=2023-06-02T08:41:58.037Z caller=repair.go:56 level=info component=tsdb msg="Found healthy blo
ck" mint=1685433604902 maxt=1685440800000 ulid=01H1P519Y7F8Z24W2W4YNC3HKC
ts=2023-06-02T08:41:58.042Z caller=repair.go:56 level=info component=tsdb msg="Found healthy blo
ck" mint=1685346574897 maxt=1685354400000 ulid=01H1P51B0YPBS4VJY2QCTHKW96
```

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


Prometheus Time Series Collectic
localhost:9090/graph?g0.expr=&g0.tab=...
Prometheus
Alerts
Graph
Status
Help
Execute
Table
Graph
Remove Panel
Add Panel

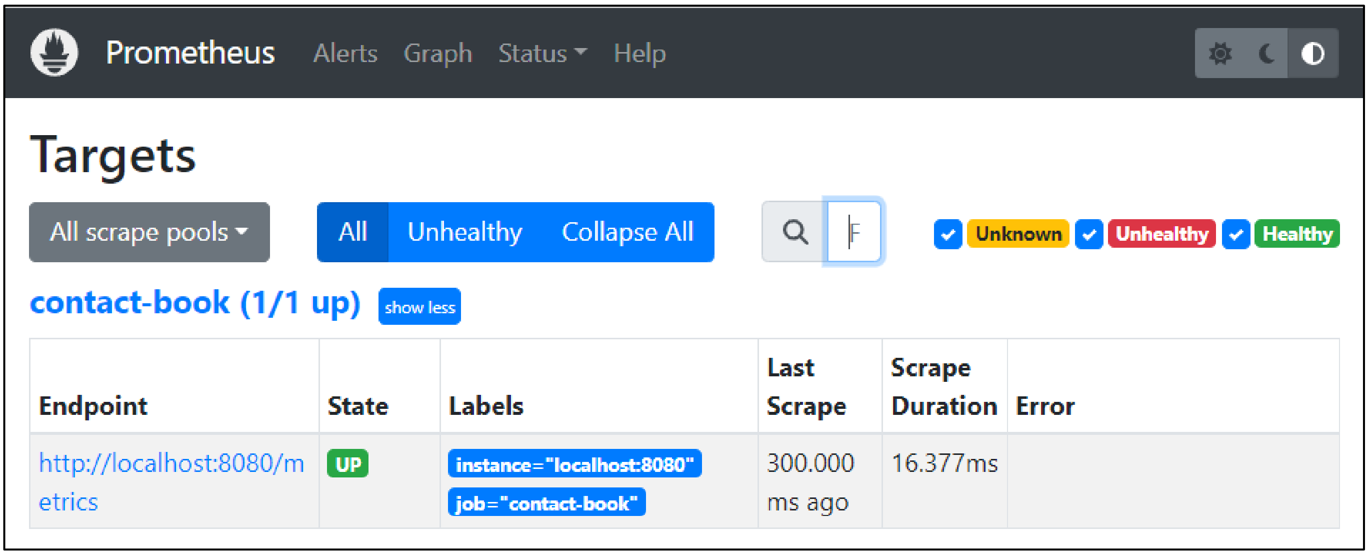
☑ Use local time ☐ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

Expression (press Shift+Enter for newlines)

Evaluation time

No data queried yet

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



Targets

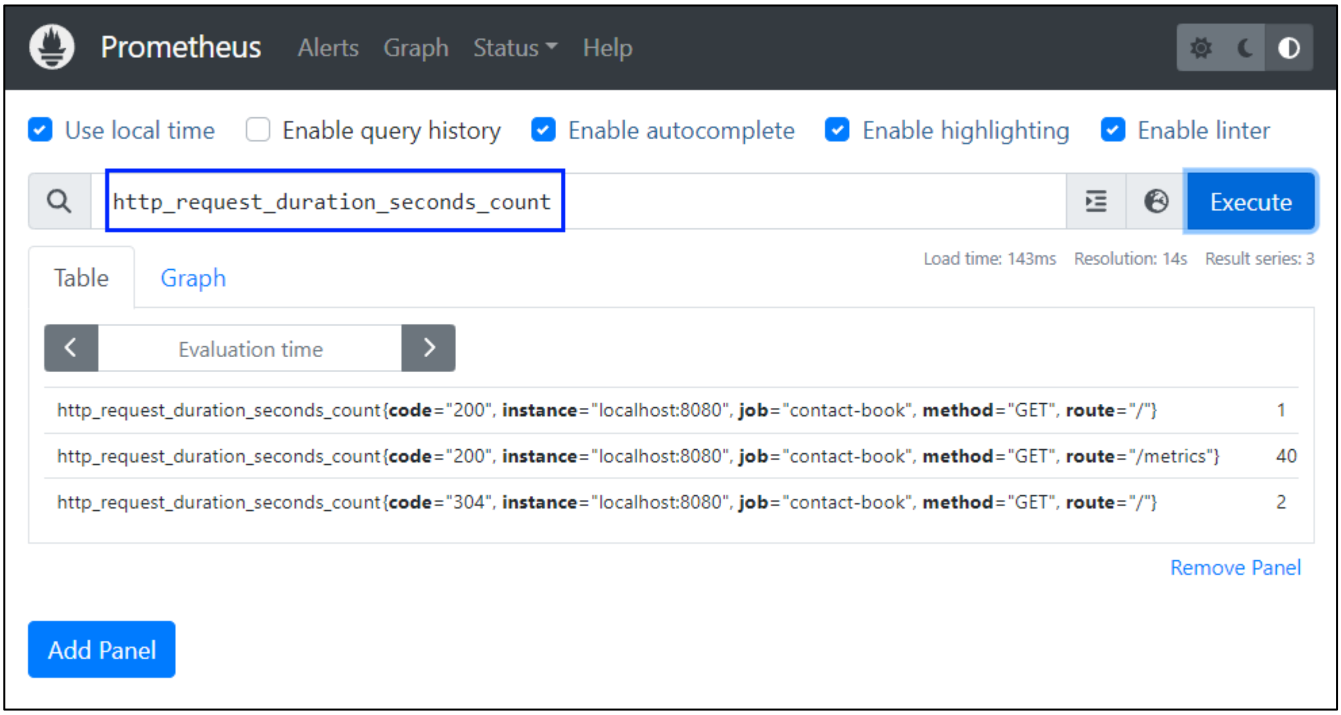
☑ Unknown ☑ Unhealthy ☑ Healthy

contact-book (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:8080/metrics	UP	instance="localhost:8080" job="contact-book"	300.000 ms ago	16.377ms	

<u>**NOTE:**</u> 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**:



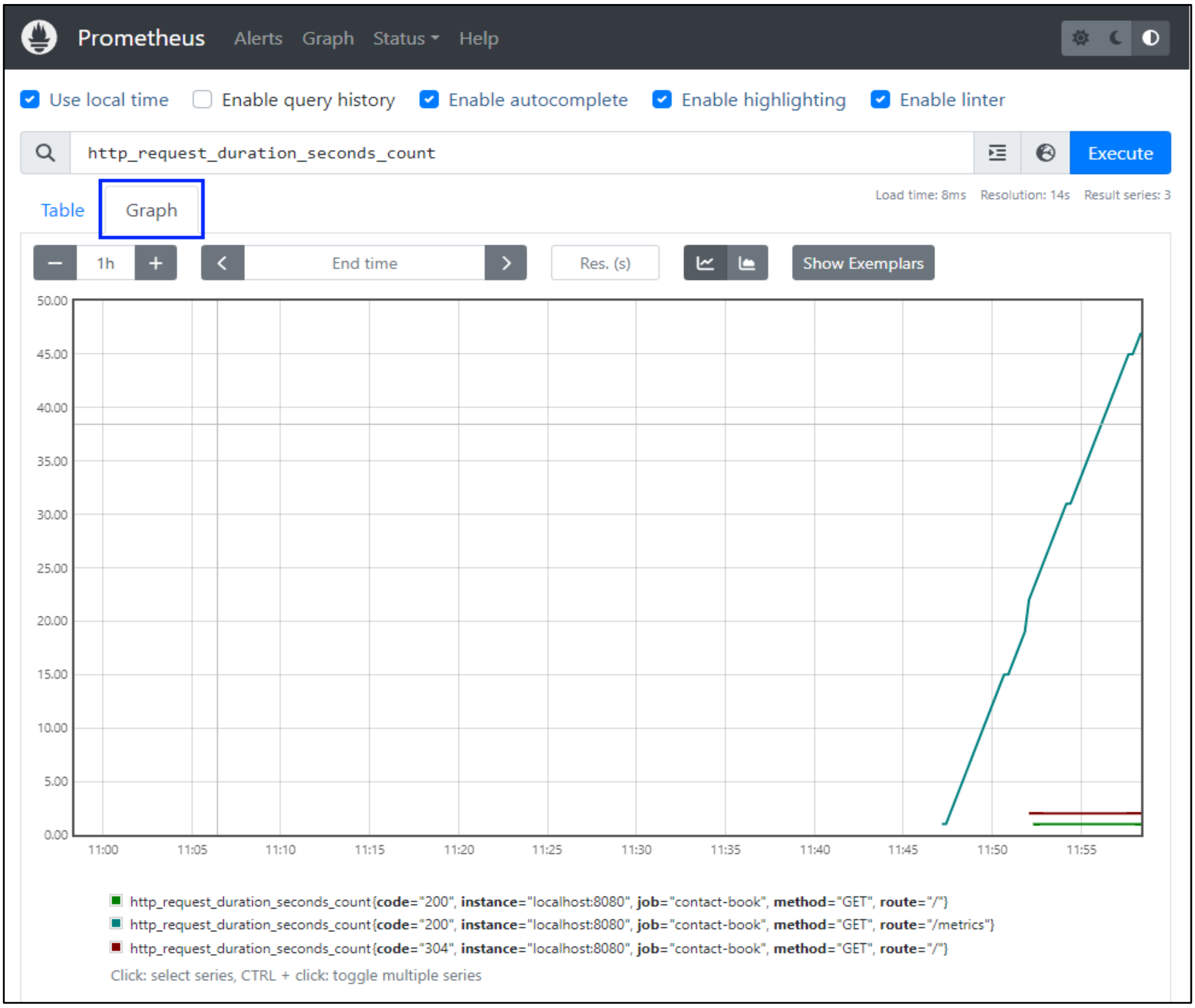
☑ Use local time ☐ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

```
http_request_duration_seconds_count
```

Load time: 143ms Resolution: 14s Result series: 3

```
http_request_duration_seconds_count{code="200", instance="localhost:8080", job="contact-book", method="GET", route="/"}          1
http_request_duration_seconds_count{code="200", instance="localhost:8080", job="contact-book", method="GET", route="/metrics"}   40
http_request_duration_seconds_count{code="304", instance="localhost:8080", job="contact-book", method="GET", route="/"}          2
```

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



☑ Use local time ☐ Enable query history ☑ Enable autocomplete ☑ Enable highlighting ☑ Enable linter

```
http_request_duration_seconds_count
```

Load time: 8ms Resolution: 14s Result series: 3

🟩 http_request_duration_seconds_count{code="200", instance="localhost:8080", job="contact-book", method="GET", route="/"}
🟩 http_request_duration_seconds_count{code="200", instance="localhost:8080", job="contact-book", method="GET", route="/metrics"}
🟥 http_request_duration_seconds_count{code="304", instance="localhost:8080", job="contact-book", method="GET", route="/"}

Click: select series, CTRL + click: toggle multiple series

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

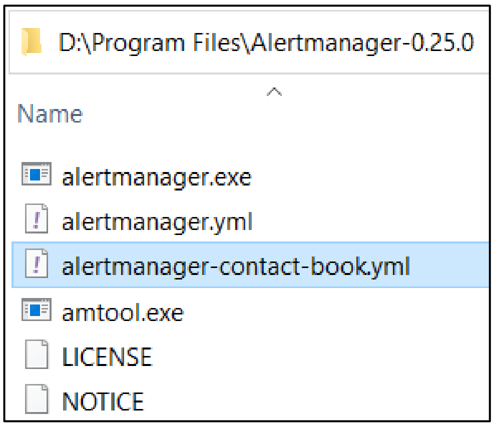
As we know how to **work with Prometheus**, let's see how to **add `Alertmanager`** to **manage alerts** and **send notifications**.

6. 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.

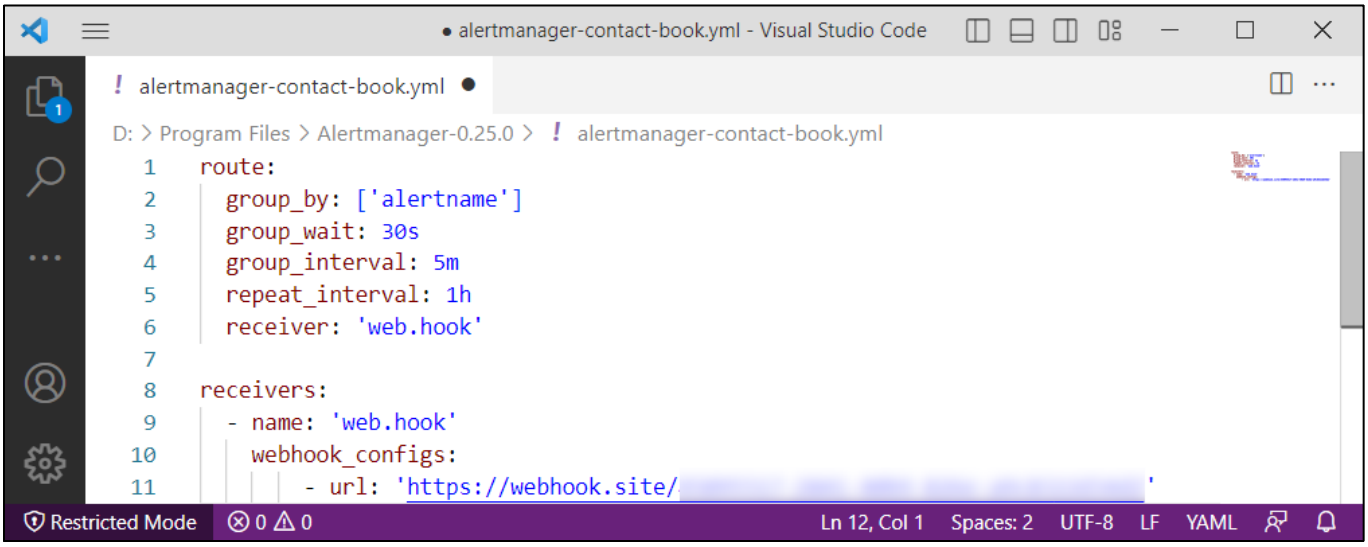
Step 1: Configure and Run Alertmanager

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:



```
D:\Program Files\Alertmanager-0.25.0

Name
alertmanager.exe
alertmanager.yml
alertmanager-contact-book.yml
amtool.exe
LICENSE
NOTICE
```

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



● alertmanager-contact-book.yml - Visual Studio Code
! alertmanager-contact-book.yml ●
D: > Program Files > Alertmanager-0.25.0 > ! alertmanager-contact-book.yml
Restricted Mode
⊗ 0 ⚠ 0
Ln 12, Col 1
Spaces: 2
UTF-8
LF
YAML

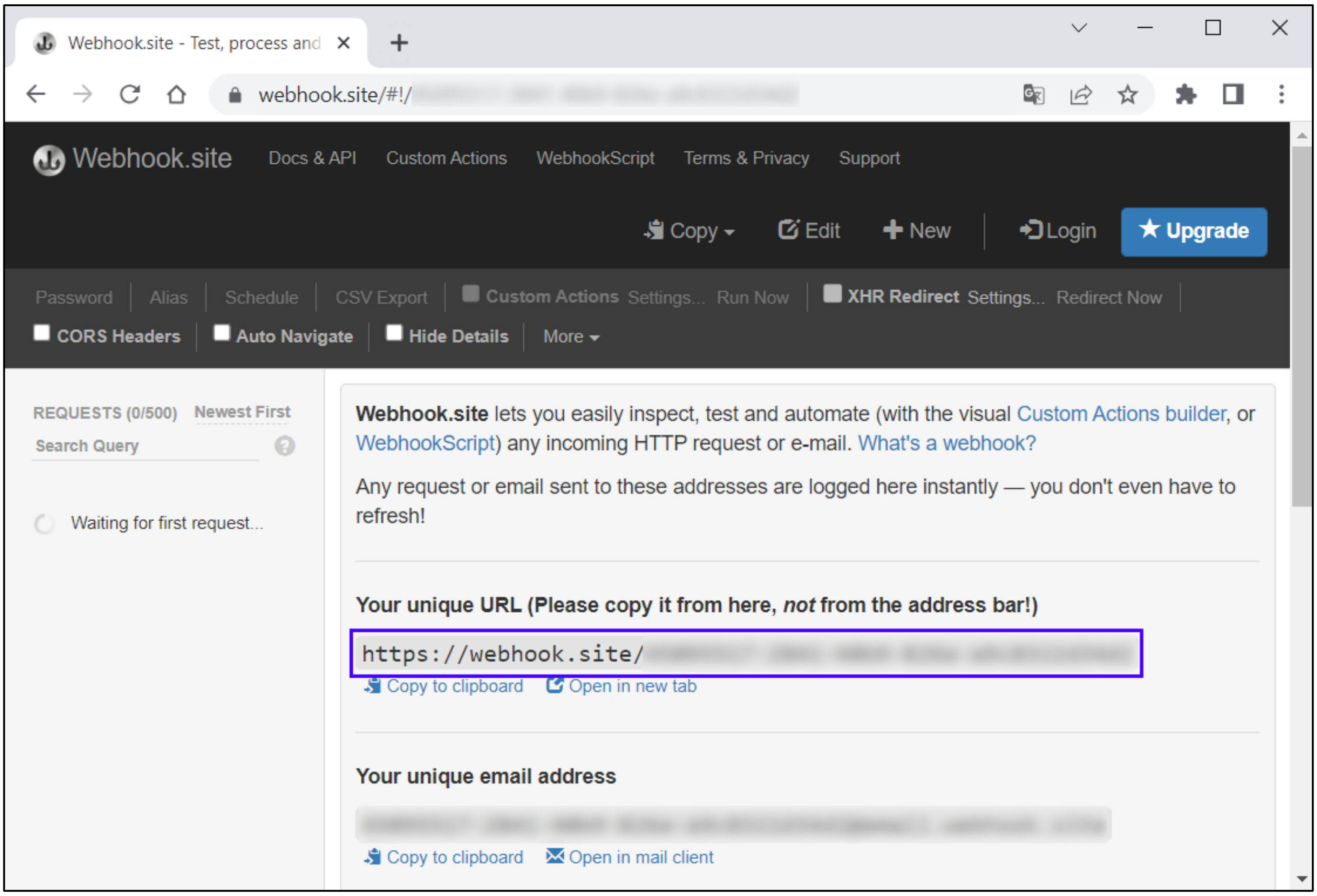
```yaml
route:
  group_by: ['alertname']
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 1h
  receiver: 'web.hook'

receivers:
  - name: 'web.hook'
    webhook_configs:
      - url: 'https://webhook.site/[redacted]'
```

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:



Webhook.site lets you easily inspect, test and automate (with the visual Custom Actions builder, or WebhookScript) any incoming HTTP request or e-mail. What's a webhook?

Any request or email sent to these addresses are logged here instantly — you don't even have to refresh!

Your unique URL (Please copy it from here, *not* from the address bar!)

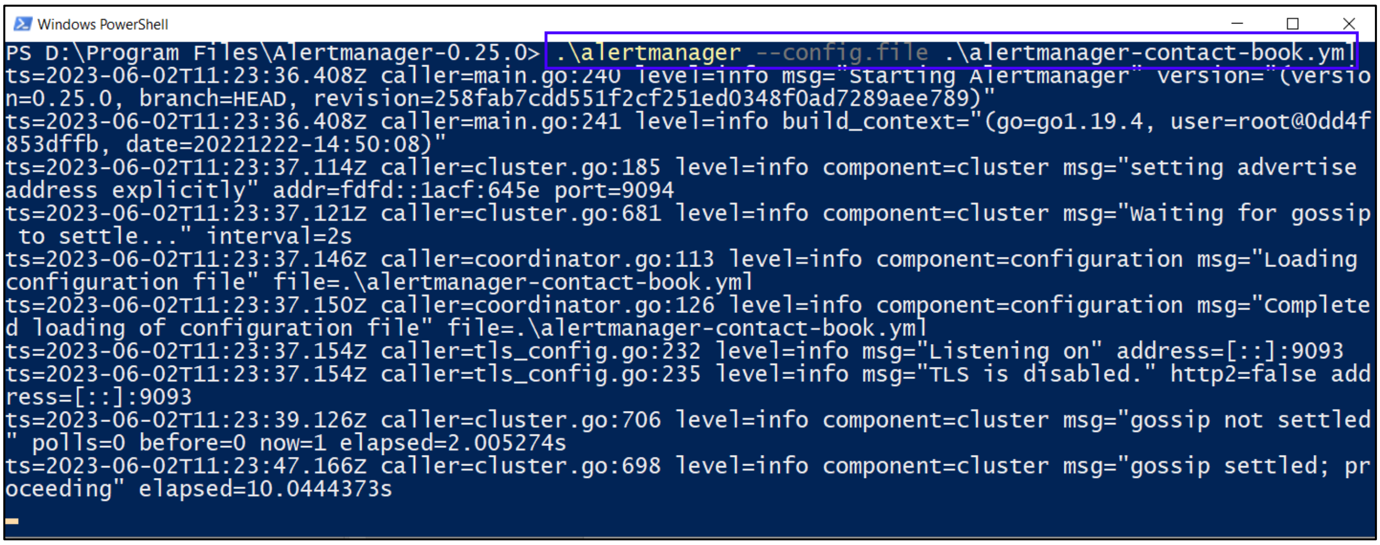
https://webhook.site/[redacted]

Your unique email address

[redacted]

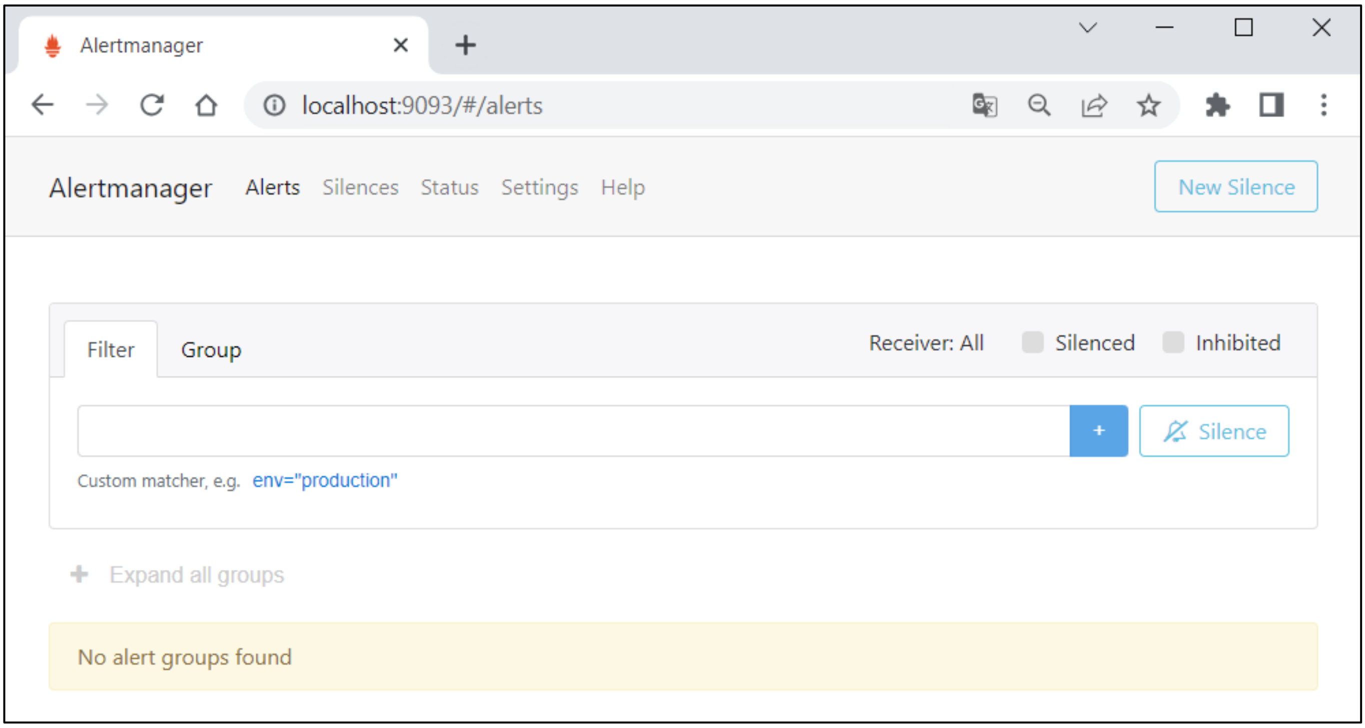
Add the URL to your configuration file and save it.

Next, run Alertmanager with the configuration file:



```
PS D:\Program Files\Alertmanager-0.25.0> .\alertmanager --config.file .\alertmanager-contact-book.yml
ts=2023-06-02T11:23:36.408Z caller=main.go:240 level=info msg="Starting Alertmanager" version="(versio
n=0.25.0, branch=HEAD, revision=258fab7cdd551f2cf251ed0348f0ad7289aee789)"
ts=2023-06-02T11:23:36.408Z caller=main.go:241 level=info build_context="(go=go1.19.4, user=root@0dd4f
853dffb, date=20221222-14:50:08)"
ts=2023-06-02T11:23:37.114Z caller=cluster.go:185 level=info component=cluster msg="setting advertise
address explicitly" addr=fdfd::1acf:645e port=9094
ts=2023-06-02T11:23:37.121Z caller=cluster.go:681 level=info component=cluster msg="Waiting for gossip
 to settle..." interval=2s
ts=2023-06-02T11:23:37.146Z caller=coordinator.go:113 level=info component=configuration msg="Loading
configuration file" file=.\alertmanager-contact-book.yml
ts=2023-06-02T11:23:37.150Z caller=coordinator.go:126 level=info component=configuration msg="Complete
d loading of configuration file" file=.\alertmanager-contact-book.yml
ts=2023-06-02T11:23:37.154Z caller=tls_config.go:232 level=info msg="Listening on" address=[::]:9093
ts=2023-06-02T11:23:37.154Z caller=tls_config.go:235 level=info msg="TLS is disabled." http2=false add
ress=[::]:9093
ts=2023-06-02T11:23:39.126Z caller=cluster.go:706 level=info component=cluster msg="gossip not settled
" polls=0 before=0 now=1 elapsed=2.005274s
ts=2023-06-02T11:23:47.166Z caller=cluster.go:698 level=info component=cluster msg="gossip settled; pr
oceeding" elapsed=10.0444373s
```

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



Custom matcher, e.g. env="production"

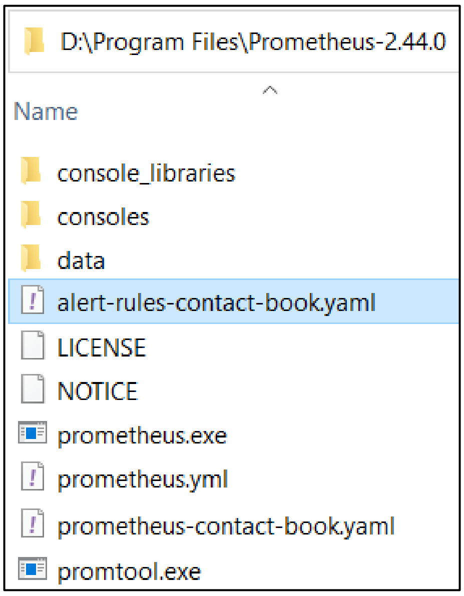
No alert groups found

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:



D:\Program Files\Prometheus-2.44.0

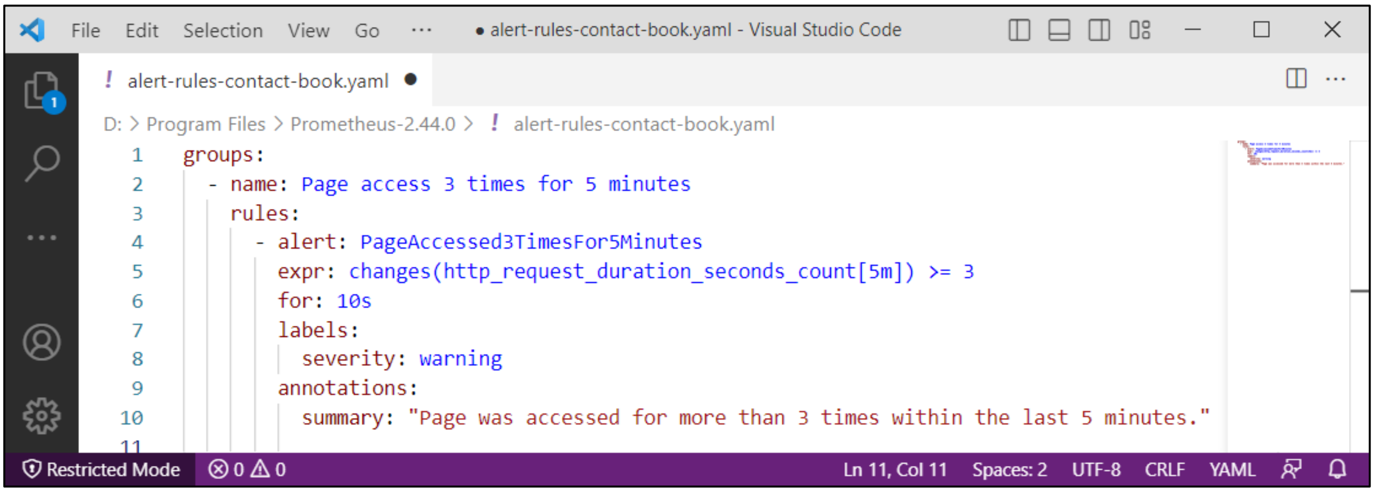
Name
console_libraries
consoles
data
alert-rules-contact-book.yaml
LICENSE
NOTICE
prometheus.exe
prometheus.yml
prometheus-contact-book.yaml
promtool.exe

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.

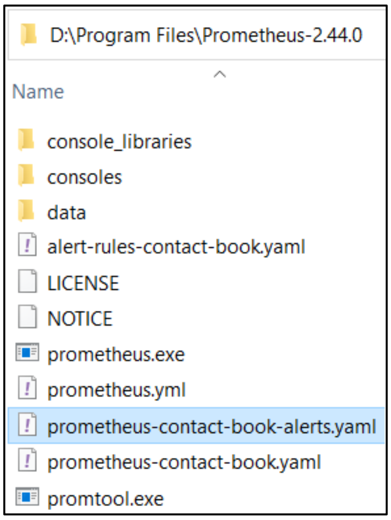
Having this expression, add the following rules configuration to the created YAML file:


File
Edit
Selection
View
Go
• alert-rules-contact-book.yaml - Visual Studio Code
alert-rules-contact-book.yaml
D: > Program Files > Prometheus-2.44.0 > alert-rules-contact-book.yaml
Restricted Mode
0 0
Ln 11, Col 11
Spaces: 2
UTF-8
CRLF
YAML

```yaml
groups:
  - name: Page access 3 times for 5 minutes
    rules:
      - alert: PageAccessed3TimesFor5Minutes
        expr: changes(http_request_duration_seconds_count[5m]) >= 3
        for: 10s
        labels:
          severity: warning
        annotations:
          summary: "Page was accessed for more than 3 times within the last 5 minutes."
```

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:

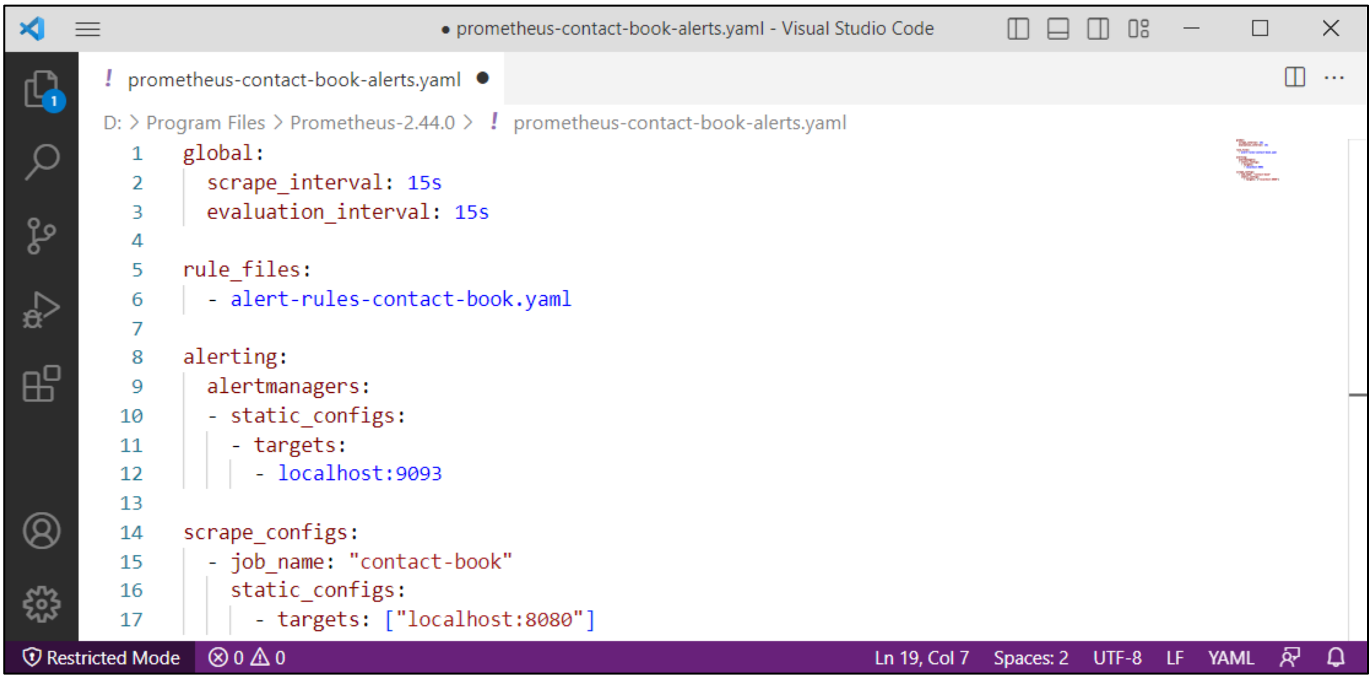


D:\Program Files\Prometheus-2.44.0

Name

- console_libraries
- consoles
- data
- alert-rules-contact-book.yaml
- LICENSE
- NOTICE
- prometheus.exe
- prometheus.yml
- prometheus-contact-book-alerts.yaml
- prometheus-contact-book.yaml
- promtool.exe

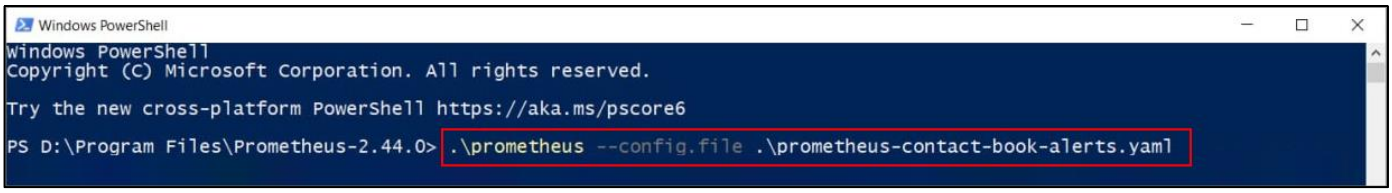
→


• prometheus-contact-book-alerts.yaml - Visual Studio Code
prometheus-contact-book-alerts.yaml
D: > Program Files > Prometheus-2.44.0 > prometheus-contact-book-alerts.yaml
Restricted Mode
0 0
Ln 19, Col 7
Spaces: 2
UTF-8
LF
YAML

```yaml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  - alert-rules-contact-book.yaml

alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - localhost:9093

scrape_configs:
  - job_name: "contact-book"
    static_configs:
      - targets: ["localhost:8080"]
```

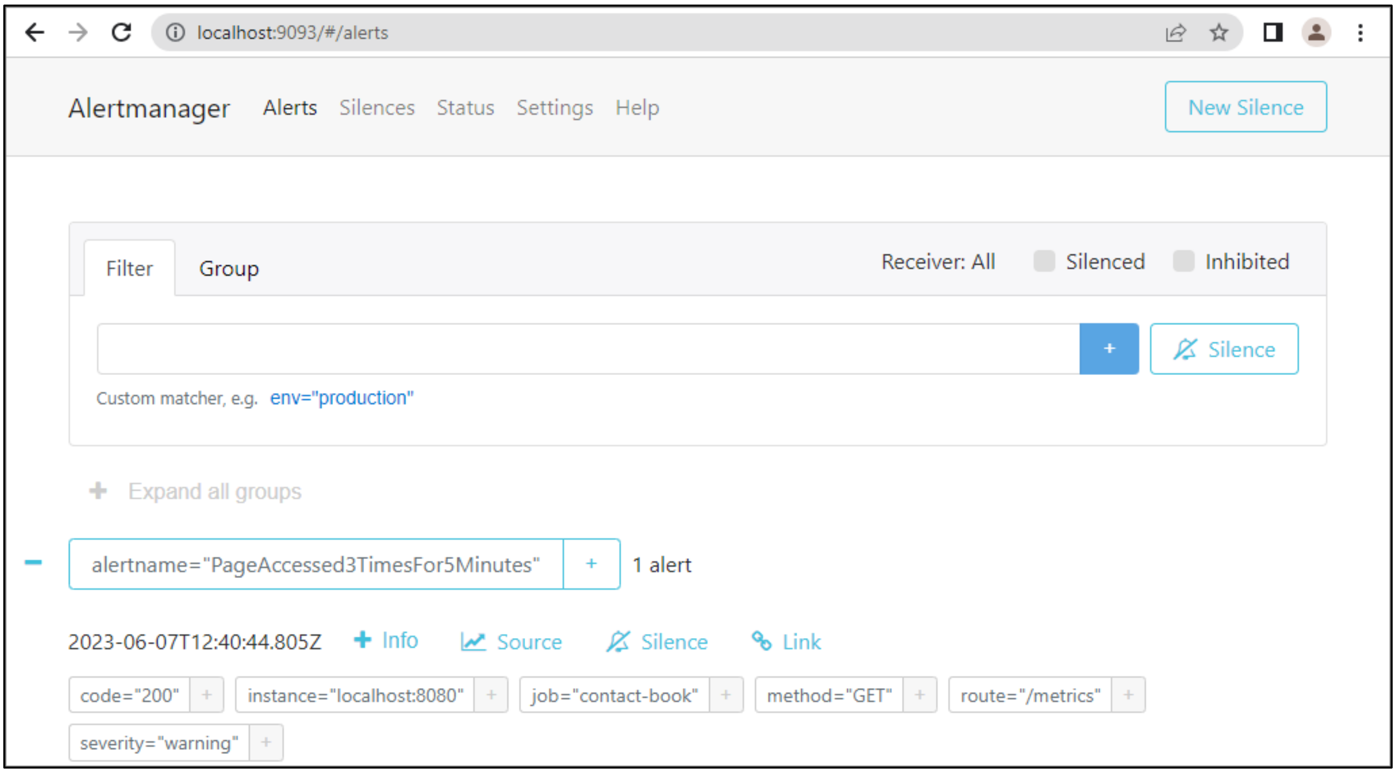
As you can see, we have added the name of the rules file and configurations for connection to Alertmanager, which is accessible on http://localhost:9093 by default. Also, we have set **`evaluation_interval`**, which is the interval based on which Prometheus evaluates the query for alerting.

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):



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\Program Files\Prometheus-2.44.0> .\prometheus --config.file .\prometheus-contact-book-alerts.yaml
```

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:

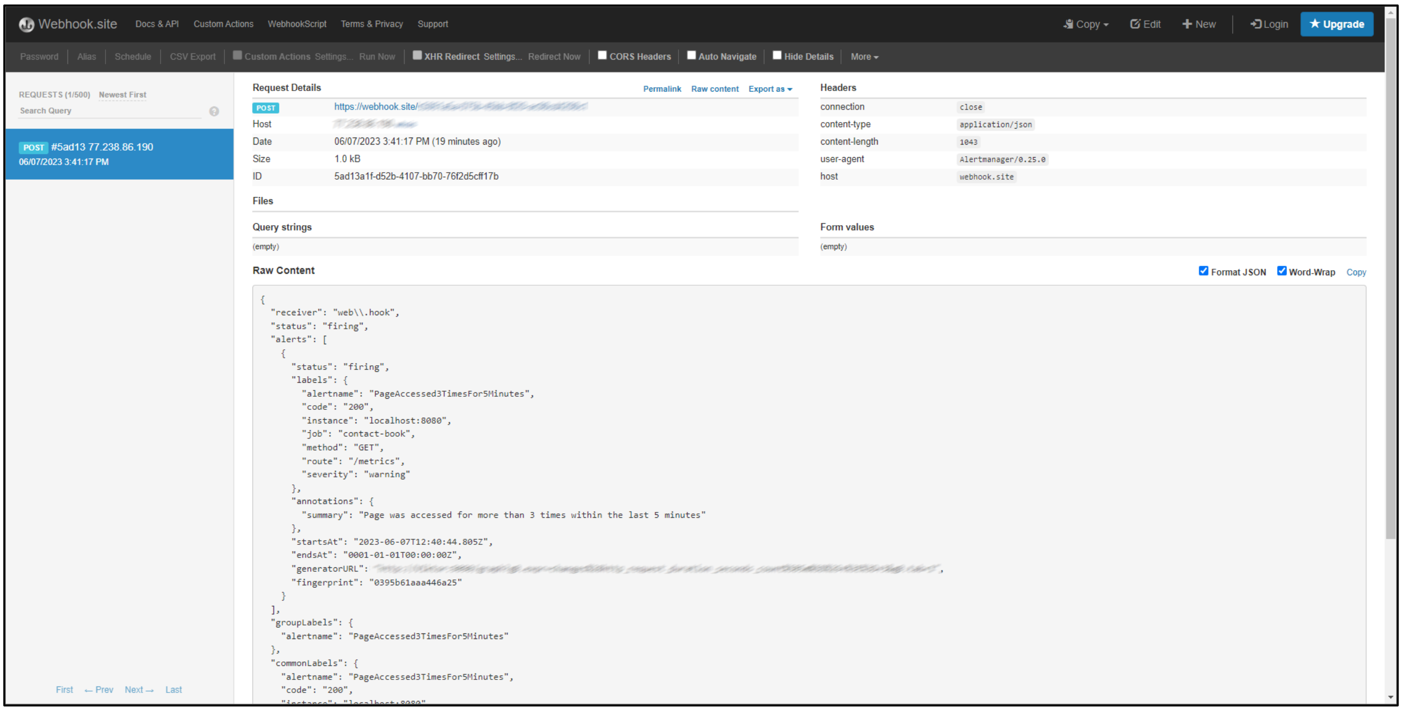


alertname="PageAccessed3TimesFor5Minutes" 1 alert

2023-06-07T12:40:44.805Z + Info Source Silence Link

code="200" instance="localhost:8080" job="contact-book" method="GET" route="/metrics" severity="warning"

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



POST #5ad13 77.238.86.190
06/07/2023 3:41:17 PM

Request Details

POST	https://webhook.site/[redacted]
Host	[redacted]
Date	06/07/2023 3:41:17 PM (19 minutes ago)
Size	1.0 kB
ID	5ad13a1f-d52b-4107-bb70-76f2d5cff17b

Headers

connection	close
content-type	application/json
content-length	1043
user-agent	Alertmanager/0.25.0
host	webhook.site

Files

Query strings

(empty)

Form values

(empty)

Raw Content

☑ Format JSON ☑ Word-Wrap Copy

```
{
  "receiver": "web\\.hook",
  "status": "firing",
  "alerts": [
    {
      "status": "firing",
      "labels": {
        "alertname": "PageAccessed3TimesFor5Minutes",
        "code": "200",
        "instance": "localhost:8080",
        "job": "contact-book",
        "method": "GET",
        "route": "/metrics",
        "severity": "warning"
      },
      "annotations": {
        "summary": "Page was accessed for more than 3 times within the last 5 minutes"
      },
      "startsAt": "2023-06-07T12:40:44.805Z",
      "endsAt": "0001-01-01T00:00:00Z",
      "generatorURL": [redacted],
      "fingerprint": "0395b61aaa446a25"
    }
  ],
  "groupLabels": {
    "alertname": "PageAccessed3TimesFor5Minutes"
  },
  "commonLabels": {
    "alertname": "PageAccessed3TimesFor5Minutes",
    "code": "200",
    "instance": "localhost:8080"
```