TP5: Déploiement avec Infrastructure as Code

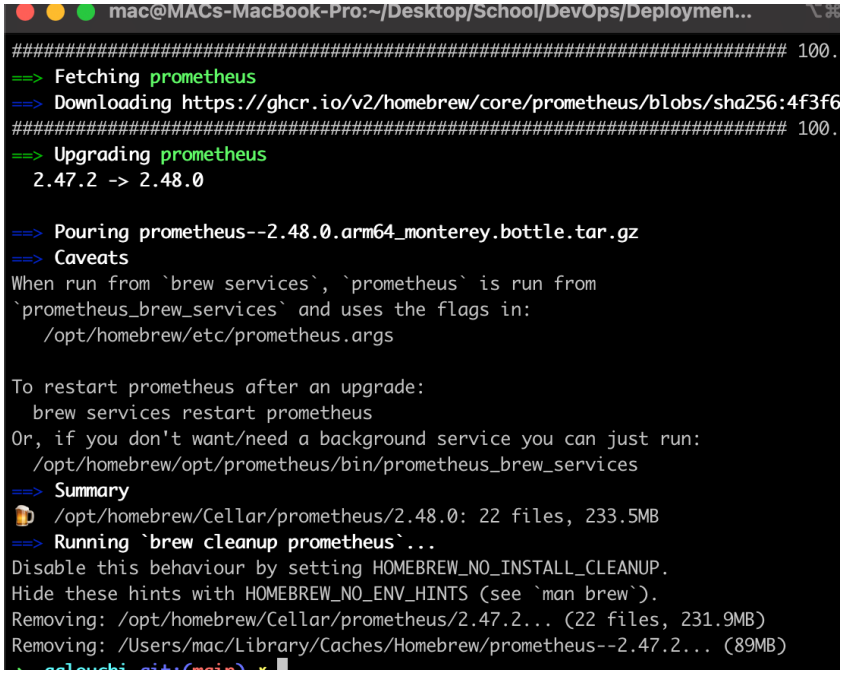
PARTIE B: Monitoring

Report Realized By Group 3 Members

Houssem Saafi
Emna Boukhris
Safa Kedidi

Step 6:

6. Configurez l'outil de monitoring avec Prometheus et Grafana.
- Installez Prometheus pour collecter les métriques de votre infrastructure et de votre application.
- Installez Grafana pour visualiser les métriques collectées à l'aide de tableaux de bord personnalisés.



```
mac@MACs-MacBook-Pro:~/Desktop/School/DevOps/Deploymen...
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
Removing: /opt/homebrew/Cellar/prometheus/2.47.2... (22 files, 231.9MB)
Removing: /Users/mac/Library/Caches/Homebrew/prometheus--2.47.2... (89MB)
➜  aalouchi git:(main) ✗ brew install grafana
==> Downloading https://ghcr.io/v2/homebrew/core/grafana/manifests/10.2.1
######################################################################## 100.
==> Fetching grafana
==> Downloading https://ghcr.io/v2/homebrew/core/grafana/blobs/sha256:cfc49297
######################################################################## 100.
==> Pouring grafana--10.2.1.arm64_monterey.bottle.tar.gz
==> Caveats
To start grafana now and restart at login:
  brew services start grafana
Or, if you don't want/need a background service you can just run:
  /opt/homebrew/opt/grafana/bin/grafana server --config /opt/homebrew/etc/graf
a/grafana.ini --homepath /opt/homebrew/opt/grafana/share/grafana --packaging\=
ew cfg:default.paths.logs\=/opt/homebrew/var/log/grafana cfg:default.paths.dat
=/opt/homebrew/var/lib/grafana cfg:default.paths.plugins\=/opt/homebrew/var/li
grafana/plugins
==> Summary
🍺  /opt/homebrew/Cellar/grafana/10.2.1: 7,579 files, 346.2MB
==> Running `brew cleanup grafana`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
```

Step 7:

7. Intégrez l'instrumentation de l'application pour exporter les métriques.
- Modifiez votre code pour intégrer des bibliothèques de monitoring telles que Prometheus Exporter ou OpenTelemetry.
- Configurez ces bibliothèques pour exporter les métriques et les journaux de votre application vers Prometheus.

- First, we will need to install prom-client into our react application:

```
➜  aalouchi git:(main) ✗ npm install prom-client
npm WARN deprecated workbox-cacheable-response@6.6.0: workbox-background-sync@6.6.0
npm WARN deprecated w3c-hr-time@1.0.2: Use your platform's native performance.now() and performance.timeOrigin.
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.
npm WARN deprecated stable@0.1.8: Modern JS already guarantees Array#sort() is a stable sort, so this library is deprecated. See the compatibility
/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort#browser_compatibility
npm WARN deprecated sourcemap-codec@1.4.8: Please use @jridgewell/sourcemap-codec instead
npm WARN deprecated rollup-plugin-terser@7.0.2: This package has been deprecated and is no longer maintained. Please use @rollup/plugin-terser
npm WARN deprecated web-animation-club@0.6.0: WARNING: This project has been renamed to @rcaferati/wac; install it using @rcaferati/wac instead.

added 1540 packages, and audited 1789 packages in 18s
```

- Then we will need to integrate the collection of metrics into our application. Given that our application is a frontend application. To collect application metrics, we will need to create a backend server that listens to our application posted metrics and Prometheus would retrieve them later from our backend server.

  - We will be creating an express backend server named metrics-server. This server must have a get route, which is the route used by Prometheus for data retrieval and a post route used by the frontend application to send metrics to the server.

- Here is a screenshot of the server code:First it imports expressed prom-client which are needed to have the needed functionalities.

```
index.js 1   reportWebVitals.js   prometheus.yml src U   prometheus.yml prometheus U   metrics-server.js M   package.json
rc > metrics-server.js > app.post('/metrics') callback
You, 3 minutes ago | 1 author (You)
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const Prom = require('prom-client');
const { collectDefaultMetrics } = require('prom-client');

const app = express();
const PORT = 3001;

// Middleware
app.use(bodyParser.json());
app.use(cors());

// Create a custom metric registry
//const customRegistry = new Registry();

// Endpoint to expose metrics for Prometheus
app.get('/metrics', async (req, res) => {
  try {
    // Collect metrics from the custom registry
    res.set('Content-Type', Prom.register.contentType);

    // Await the resolution of the Promise returned by metrics()
    const metrics = await Prom.register.metrics();

    res.end(metrics);
  } catch (error) {
    console.error('Error generating metrics:', error);
    res.status(500).send('Internal Server Error');
  }
});


// Endpoint to handle performance metrics sent from the React app
app.post('/metrics', (req, res) => {
  const metricData = req.body;
  console.log('Received performance metric:', metricData);
```

- The frontend app will make use of the post call implemented in the server to send its metrics as shown below in the screenshot:

```
// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
// Import axios for making HTTP requests
function sendToServer(metric) {
  const url = 'http://localhost:3001/metrics';

  // Adjust the payload or structure based on your server's expectations
  const payload = {
    metricName: metric.name,  // Adjust based on the metric data structure
    metricValue: metric.value, // Adjust based on the metric data structure
  };

  // Use axios to send a POST request to the server
  axios.post(url, payload)
    .then(response => {
      console.log('Metric sent successfully:', response.data);
    })
    .catch(error => {
      console.error('Error sending metric:', error);
    });
}

// Call sendToServer instead of navigator.sendBeacon      You, yesterday • terraform
reportWebVitals(sendToServer);
```

- Finally, we will need to setup Prometheus to retrieve data from the server metrics get route in the “prometheus.yml” file:

rometheus > ! prometheus.yml > [ ] scrape_configs > {} 0

prometheus.json - Prometheus configuration file (prometheus.json)

```yaml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'react-app'
    static_configs:
      - targets: ['localhost:3001']
```

- This sets Prometheus to retrieve data from the backend server every 15 seconds.

- Here is a screenshot of the metrics shown when calling the get route of the backend metrics server (started by running this command: node src/metrics-server.js)

localhost:3001/metrics

Gmail YouTube Netflix Traduire GitHub Diet & Workouts Opportunities DataScience us-startups

```
nodejs_heap_space_size_available_bytes{space="map"} 342848
nodejs_heap_space_size_available_bytes{space="large_object"} 0
nodejs_heap_space_size_available_bytes{space="code_large_object"} 0
nodejs_heap_space_size_available_bytes{space="new_large_object"} 1030976
nodejs_heap_space_size_available_bytes{space="new"} 326224

# HELP nodejs_version_info Node.js version info.
# TYPE nodejs_version_info gauge
nodejs_version_info{version="v18.18.0",major="18",minor="18",patch="0"} 1

# HELP nodejs_gc_duration_seconds Garbage collection duration by kind, one of major, minor, incremental or weakcb.
# TYPE nodejs_gc_duration_seconds histogram
nodejs_gc_duration_seconds_bucket{le="0.001",kind="incremental"} 4
nodejs_gc_duration_seconds_bucket{le="0.01",kind="incremental"} 4
nodejs_gc_duration_seconds_bucket{le="0.1",kind="incremental"} 4
nodejs_gc_duration_seconds_bucket{le="1",kind="incremental"} 4
nodejs_gc_duration_seconds_bucket{le="2",kind="incremental"} 4
nodejs_gc_duration_seconds_bucket{le="5",kind="incremental"} 4
nodejs_gc_duration_seconds_bucket{le="+Inf",kind="incremental"} 4
nodejs_gc_duration_seconds_sum{kind="incremental"} 0.0014418759346008303
nodejs_gc_duration_seconds_count{kind="incremental"} 4
nodejs_gc_duration_seconds_bucket{le="0.001",kind="major"} 0
nodejs_gc_duration_seconds_bucket{le="0.01",kind="major"} 2
nodejs_gc_duration_seconds_bucket{le="0.1",kind="major"} 2
nodejs_gc_duration_seconds_bucket{le="1",kind="major"} 2
nodejs_gc_duration_seconds_bucket{le="2",kind="major"} 2
nodejs_gc_duration_seconds_bucket{le="5",kind="major"} 2
nodejs_gc_duration_seconds_bucket{le="+Inf",kind="major"} 2
nodejs_gc_duration_seconds_sum{kind="major"} 0.004941291332244873
nodejs_gc_duration_seconds_count{kind="major"} 2
nodejs_gc_duration_seconds_bucket{le="0.001",kind="minor"} 0
nodejs_gc_duration_seconds_bucket{le="0.01",kind="minor"} 1
nodejs_gc_duration_seconds_bucket{le="0.1",kind="minor"} 1
nodejs_gc_duration_seconds_bucket{le="1",kind="minor"} 1
nodejs_gc_duration_seconds_bucket{le="2",kind="minor"} 1
nodejs_gc_duration_seconds_bucket{le="5",kind="minor"} 1
nodejs_gc_duration_seconds_bucket{le="+Inf",kind="minor"} 1
nodejs_gc_duration_seconds_sum{kind="minor"} 0.0010542078018188476
nodejs_gc_duration_seconds_count{kind="minor"} 1

# HELP FCP Metric received from React app: FCP
# TYPE FCP gauge
FCP 688.3000001907349

# HELP TTFB Metric received from React app: TTFB
# TYPE TTFB gauge
TTFB 13.600000143051147

# HELP LCP Metric received from React app: LCP
# TYPE LCP gauge
LCP 810.8000001907349

# HELP FID Metric received from React app: FID
# TYPE FID gauge
FID 0.9000000953674316

# HELP CLS Metric received from React app: CLS
# TYPE CLS gauge
CLS 0.9299059966459554
```

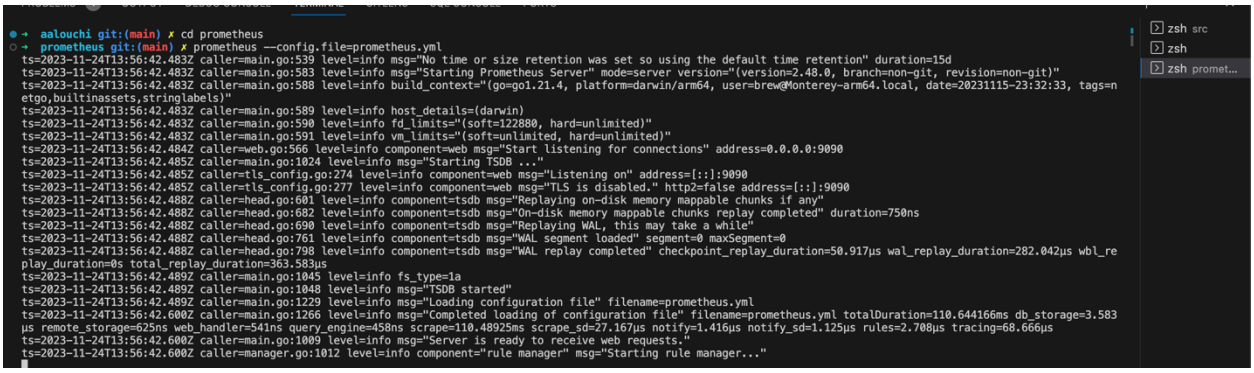
1

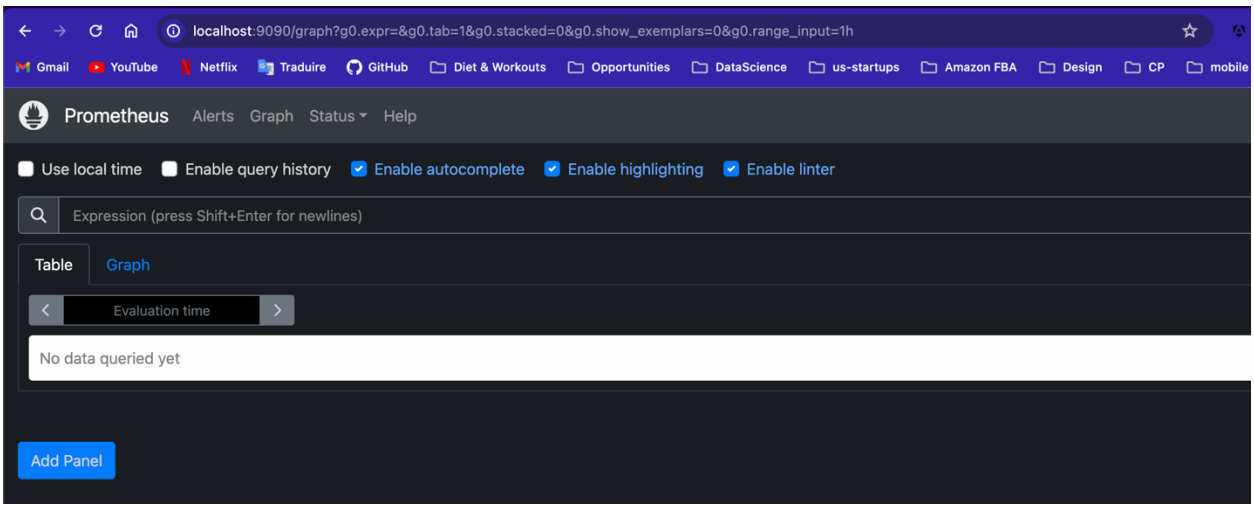
2

As shown in the screenshot; the backend server sends:

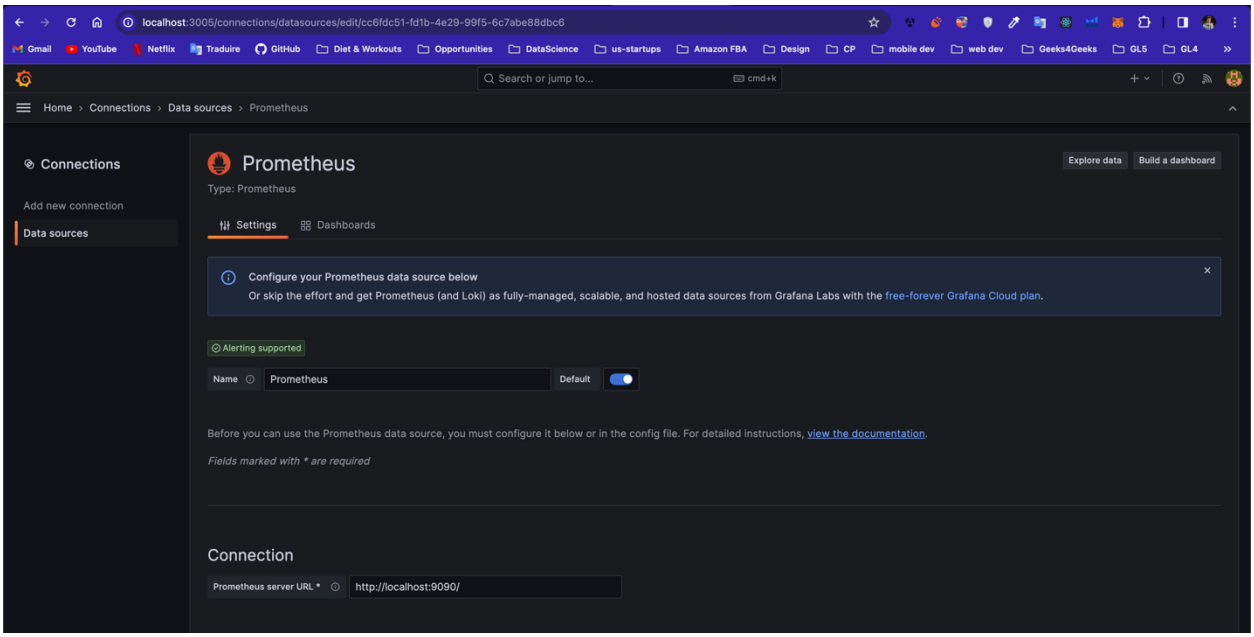
1. General or default metrics related to CPU usage, memory usage, volume, execution time, etc…
2. Frontend application related metrics such as FCP, TTFB, LCP, FID, and CLS.

Finally by running the `prometheus --config.file=prometheus.yml` the scraping would be finally in action.

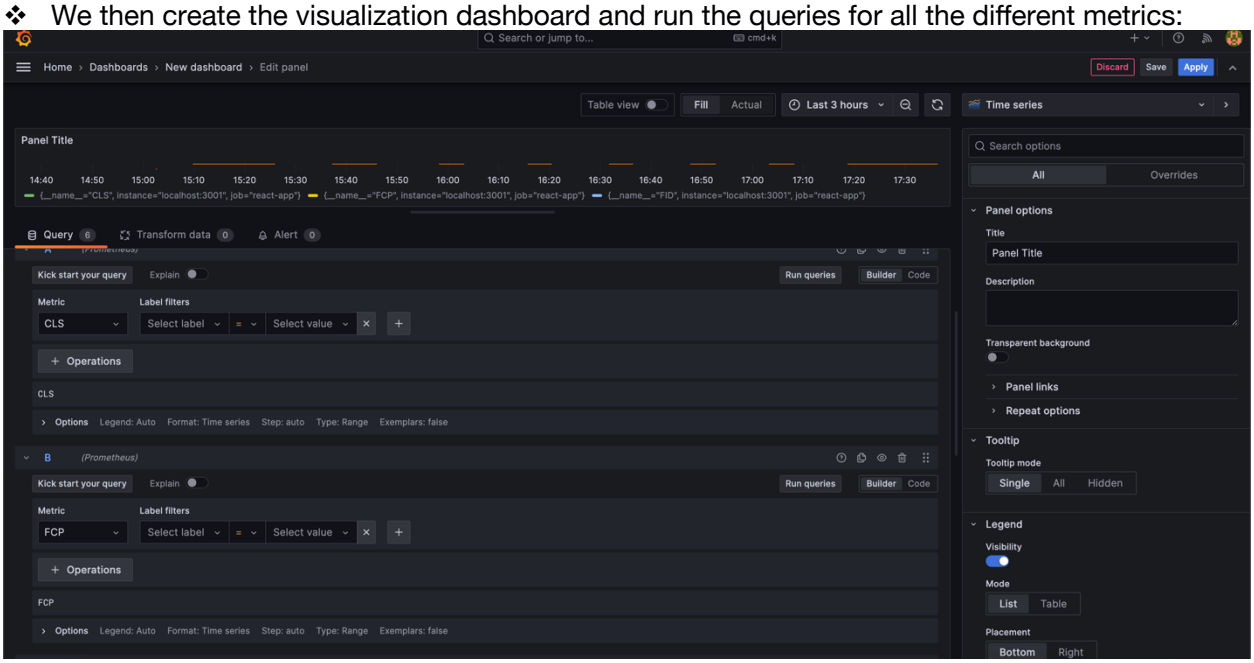




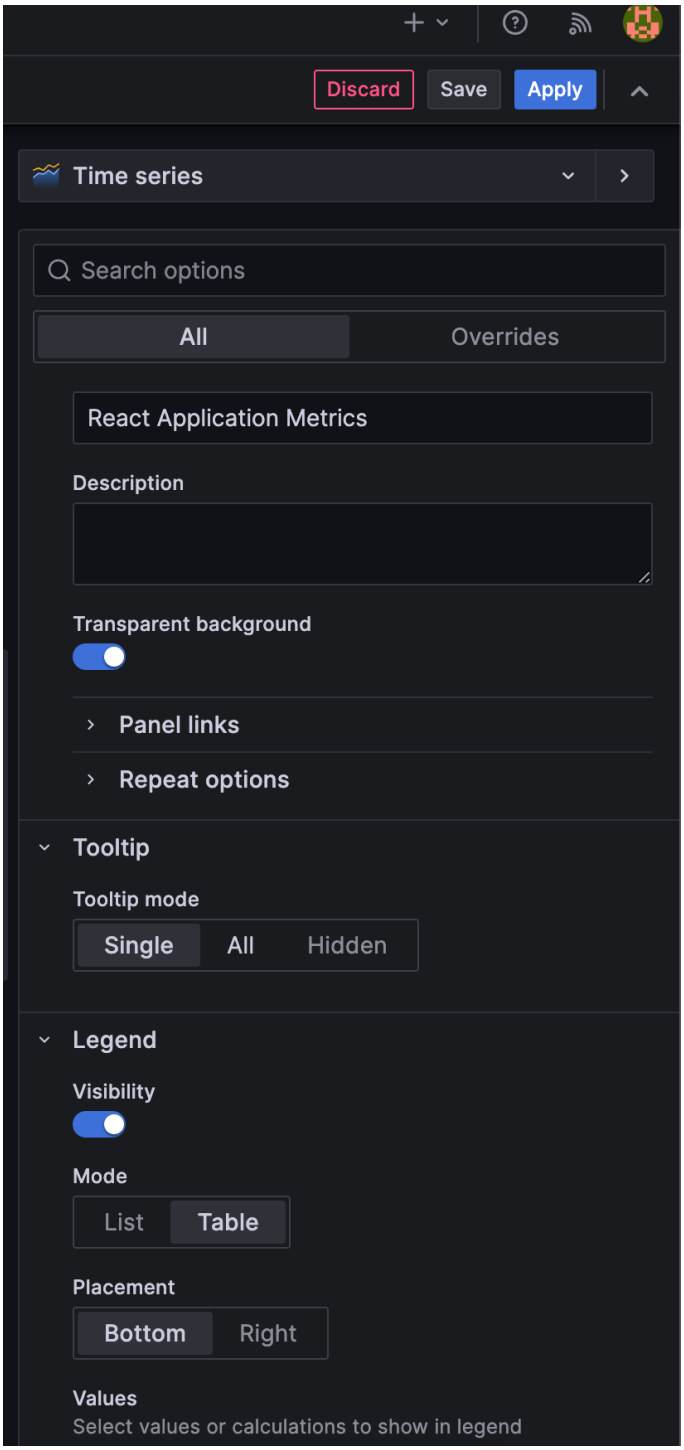
Finally, having installed grafana already, we should go to localhost:3005 (modified default port because that's where our react app is running) and connect grafana to prometheus:



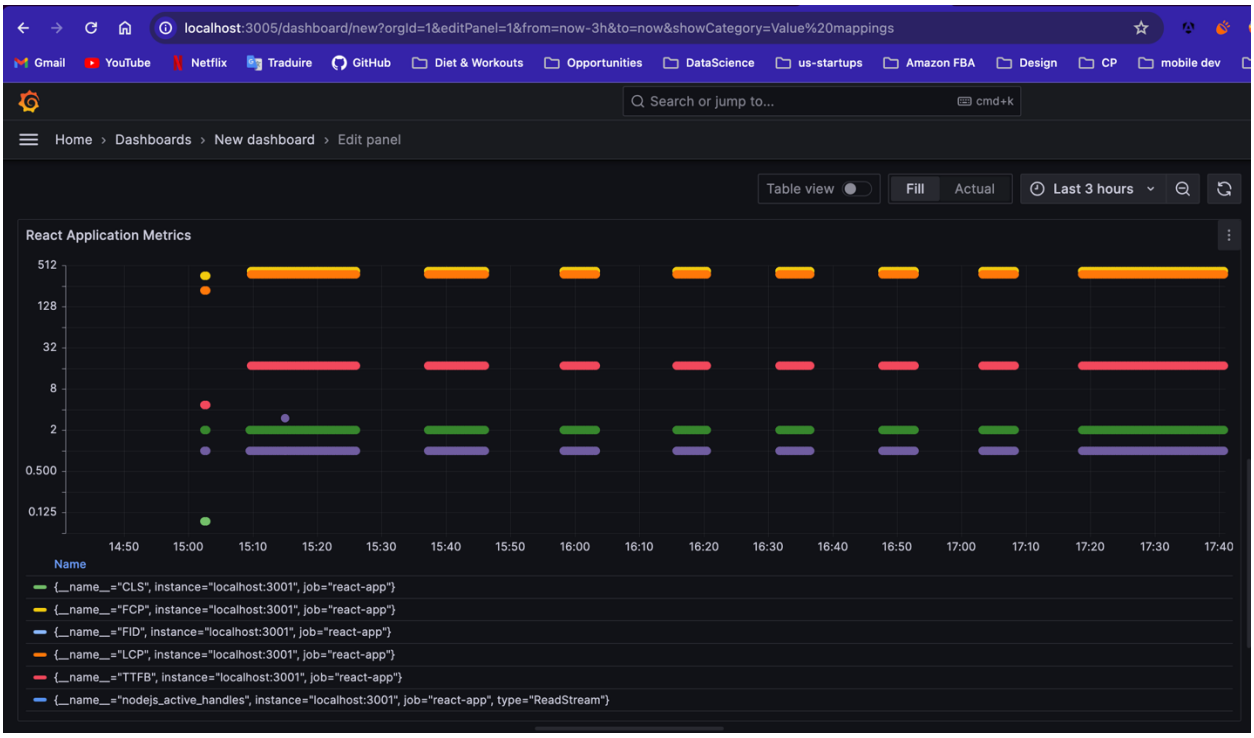
❖ We then create the visualization dashboard and run the queries for all the different metrics:

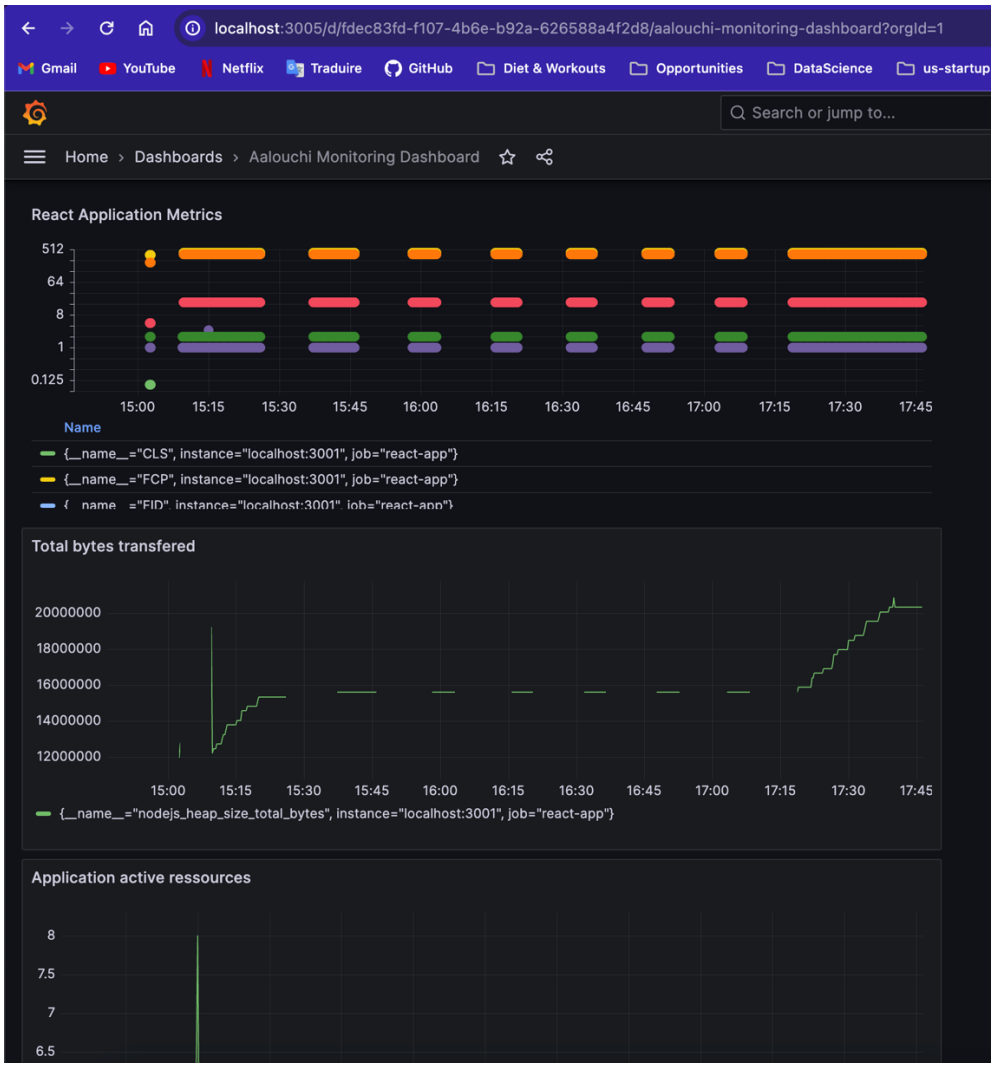


❖ Then we make some modification to make our visualization easier to view and analyze:



❖ Finally, here is a look at our application metrics:

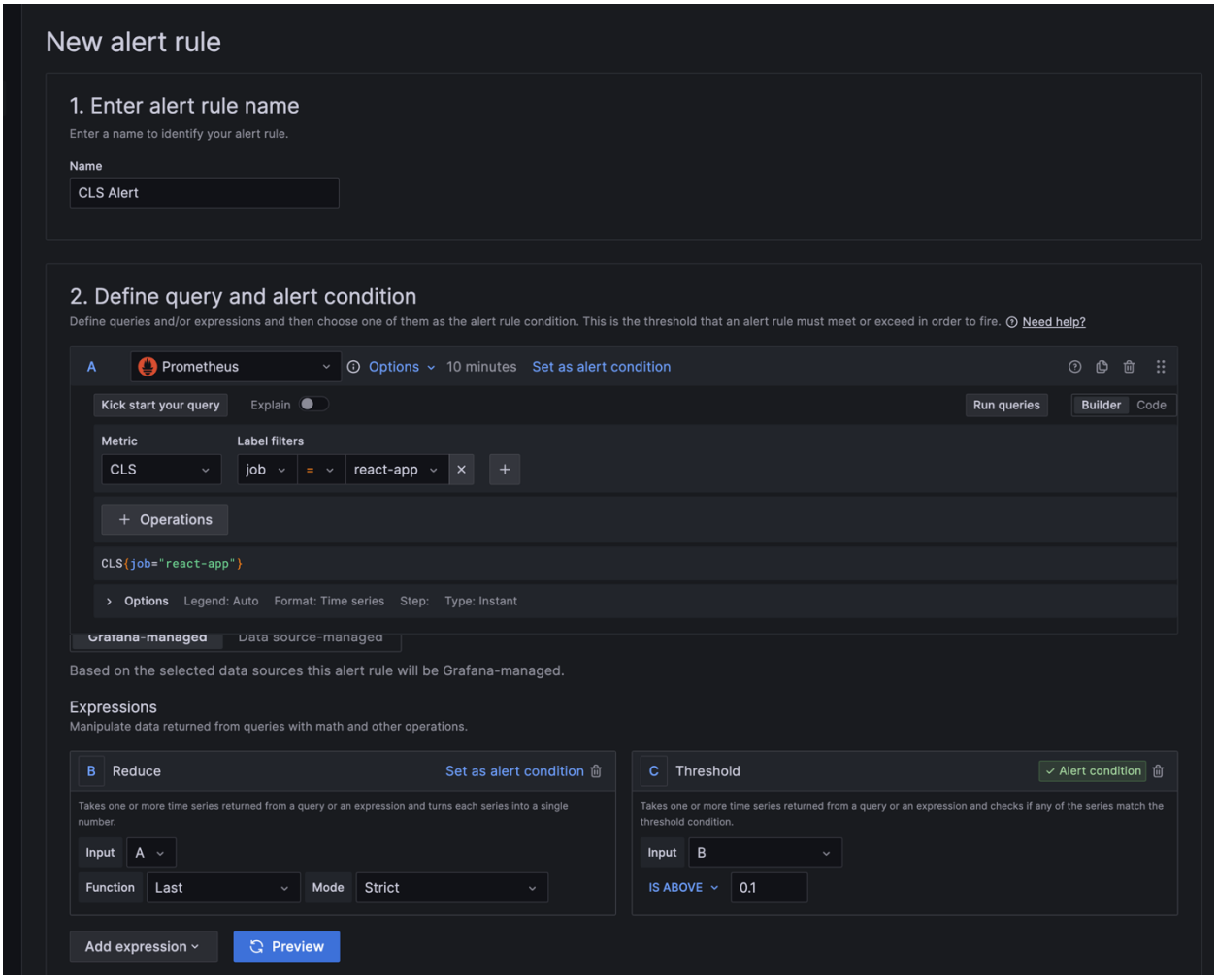




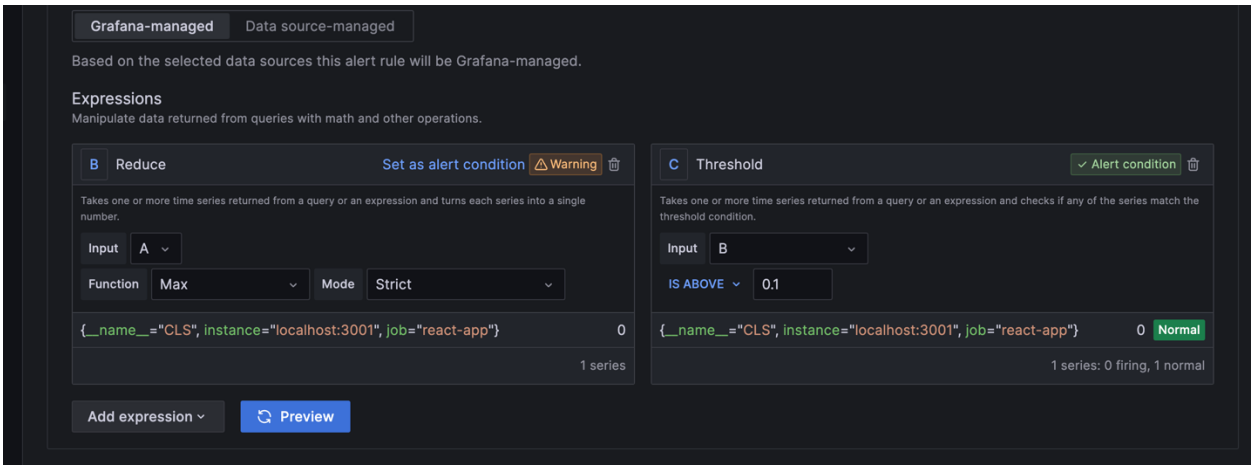
- Final view on our React application metrics and monitoring dashboard which has metrics in relation to application's performance as well as information about the infrastructure such as CPU usage, bytes transferred as well as memory used.
  - Those construct valuable information as if needed they can help us determine if the application is being slow and determine when a good time is to scale up or scale down.

Defining alerts: CLS metric:

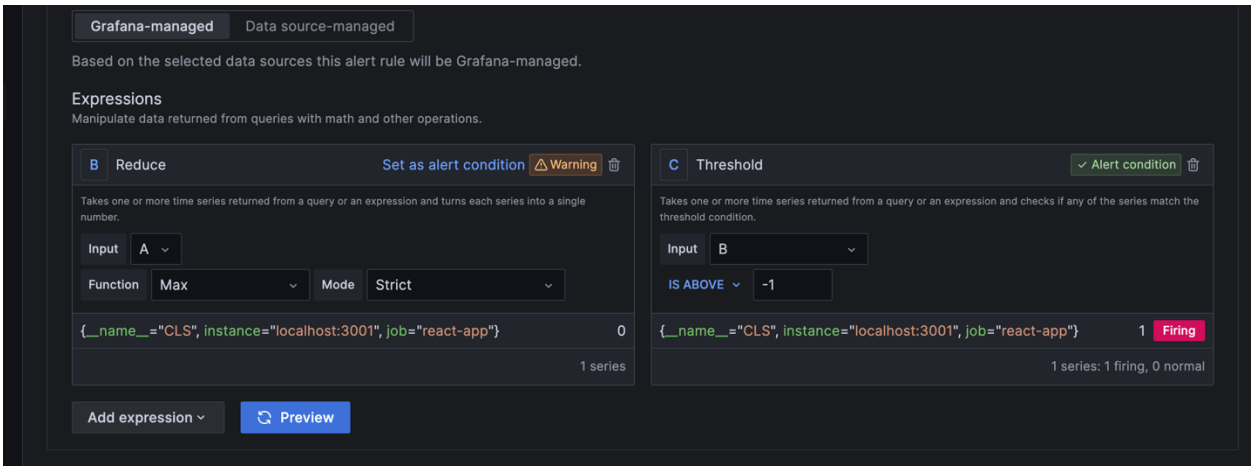
Cumulative Layout Shift (CLS) is a stable Core Web Vital metric. It is an important, user-centric metric for measuring visual stability because it helps quantify how often users experience unexpected layout shifts—a low CLS helps ensure that the page is delightful.



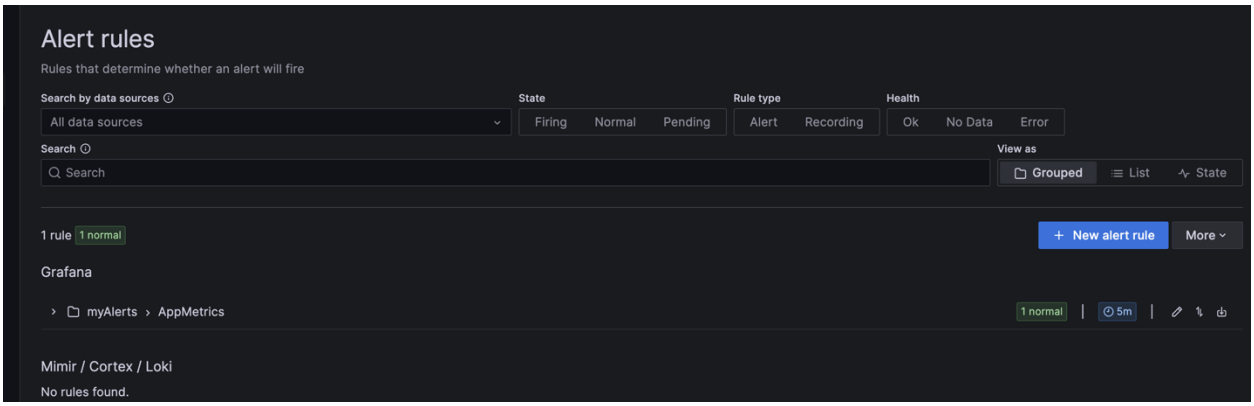
Here if the CLS becomes more than 0.1 which means that it went out of the “good” region into the “needs improvement” region. The alert will be activated.



Given our latest sent CLS is 0 for our condition we got normal as a return. If our condition was -1 for example this should fire the alert:



The alert is finally added checking every 5 minutes for any possible issues in order to be fired if needed.



Jenkins Final Pipeline:

Code Blame 112 lines (100 loc) · 4.23 KB Code 55% faster with GitHub Copilot Raw

```
pipeline {
    agent any
    environment {
        DOCKER_HOME = '/Users/mac/.jenkins/tools/org.jenkinsci.plugins.docker.commons.tools.DockerTool/docker/bin'
    }
    tools {
        dockerTool 'docker'
        maven 'Maven'
    }
    stages {
        stage('Pull from GitHub') {
            steps {
                git url: 'https://github.com/HoussemSaafi/aalouchi', branch: 'main',
                credentialsId: 'github-credentials'
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    echo "Building the Docker image"
                    sh "which docker"
                    sh "mv ~/.docker/config.json ~/.docker/config.json_bkp"
                    withCredentials([usernamePassword(credentialsId: 'docker-credential', usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD')]) {
                        echo USERNAME
                        echo "Logging into Docker Hub"
                        sh "${DOCKER_HOME}/docker login -u $USERNAME -p $PASSWORD"
                    }
                    echo 'Starting Docker service...'
                    sh "${DOCKER_HOME}/docker build -t allouchijenkins ."
                }
            }
        }

        stage('Application Testing') {
            steps {
                script {
                    echo "Testing the Docker image"
                    sh "${DOCKER_HOME}/docker run -d -p 9292:80 --name test-container allouchijenkins"

                    def nagiosCheckCommand = "/opt/homebrew/sbin/check_http -H localhost -p 9292 -u /"
```

aalouchi / Jenkinsfile ↑ Top

Code Blame 112 lines (100 loc) · 4.23 KB Code 55% faster with GitHub Copilot Raw

```
                    def nagiosCheckCommand = "/opt/homebrew/sbin/check_http -H localhost -p 9292 -u /"
                    def nagiosExitCode = sh(script: nagiosCheckCommand, returnStatus: true)

                    if (nagiosExitCode == 0) {
                        echo "Nagios check passed. Application is healthy. The new image will be pushed to the docker hub"
                    } else {
                        error "Nagios check failed. Application is not healthy. The new image won't be pushed to the docker hub."
                    }
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    echo "Tagging and pushing the Docker image to Docker Hub"
                    sh "docker tag allouchijenkins houssemsaafi/allouchijenkins"
                    sh "docker push houssemsaafi/allouchijenkins"
                }
            }
        }

        stage('Sonarqube Security and Bug Tests') {
            steps {
                script {
                    sh "/opt/homebrew/bin/sonar-scanner -Dsonar.projectKey=allouchi -Dsonar.sources=. -Dsonar.host.url=http://localhost:9000 -Dsonar.token=sqp_ab052dd733221beefa
                }
            }
        }

        stage('Terraform Apply') {
            steps {
                script {
                    echo 'Running Terraform apply...'
                    sh '/opt/homebrew/bin/terraform destroy -auto-approve'
                    sh '/opt/homebrew/bin/terraform init'
                    sh '/opt/homebrew/bin/terraform apply -auto-approve'
                }
            }
        }
```

```
        stage('Nagios Check - After Deployment') {
            steps {
                script {
                    // Run a Nagios check using a specific plugin, e.g., check_http
                    def nagiosCheckCommand = "/opt/homebrew/sbin/check_http -H localhost -p 9191 -u /"
                    def nagiosExitCode = sh(script: nagiosCheckCommand, returnStatus: true)

                    if (nagiosExitCode == 0) {
                        echo "Nagios check passed. Deployed Application is healthy."
                    } else {
                        error "Nagios check failed. Deployed Application is not healthy."
                    }
                }
            }
        }
        stage('Monitoring') {
            steps {
                echo "Monitoring Stage..."

                sh "node src/metrics-server.js &"
                echo "Node metrics sending server started."

                sh "/opt/homebrew/bin/prometheus --config.file=prometheus.yml &"
                echo "Prometheus started."

                sh "/opt/homebrew/bin/brew services start grafana"
                echo "Grafana started."
            }
        }
    }
}
```

	Declarative: Checkout SCM	Declarative: Tool Install	Pull from GitHub	Build Docker Image	Application Testing	Push Docker Image	Sonarqube Security and Bug Tests	Terraform Apply	Nagios Check - After Deployment	Monitoring
Average stage times: (Average full run time: ~1min 50s)	4s	1s	2s	43s	2s	5s	21s	2s	1s	3s
#10 Nov 29 19:22 1 commit	1s	1s	2s	48s	3s	7s	37s	4s	1s	3s

Conclusion:

- In conclusion, the implementation of monitoring and instrumentation for the React application has been successfully executed, incorporating Prometheus and Grafana to collect, visualize, and analyze both infrastructure and application metrics. The step-by-step process involved installing prom-client in the React application, creating a dedicated backend server (metrics-server) for metric collection, and configuring Prometheus to scrape data from the server.

- The instrumentation of the React application included the integration of monitoring libraries such as Prometheus Exporter, allowing the application to send metrics to the backend server. This provides valuable insights into various metrics, including CPU usage, memory usage, execution time, and application-specific metrics like FCP, TTFB, LCP, FID, and CLS.

- The Grafana dashboard, connected to Prometheus, serves as a centralized visualization tool for monitoring the application's performance and infrastructure metrics. The ability to define alerts based on critical metrics, such as Cumulative Layout Shift (CLS), adds a proactive element to the monitoring strategy. Alerts are configured to trigger when metrics surpass predefined thresholds, enabling the identification and resolution of potential issues before they impact the user experience.

- The monitoring and alerting system provides a comprehensive overview of the React application, aiding in the identification of performance bottlenecks, infrastructure issues, and potential scalability concerns. The integration of monitoring practices aligns with best practices in DevOps, ensuring a proactive and data-driven approach to maintaining the health and reliability of the application.

- This comprehensive monitoring setup, integrated seamlessly into the React application's CI/CD pipeline alongside Jenkins, contributes significantly to the overall efficiency, reliability, and scalability of the software development and deployment lifecycle. The combination of Prometheus, Grafana, and Jenkins provides a powerful toolset for continuous monitoring, feedback, and improvement.