Post-Mortem: Simulated Backend Service Outage (June 19, 2025)

Date of Incident: June 19, 2025, 8:45 PM - 9:00 PM +04 (Tbilisi Time)

Date of Post-Mortem: June 19, 2025

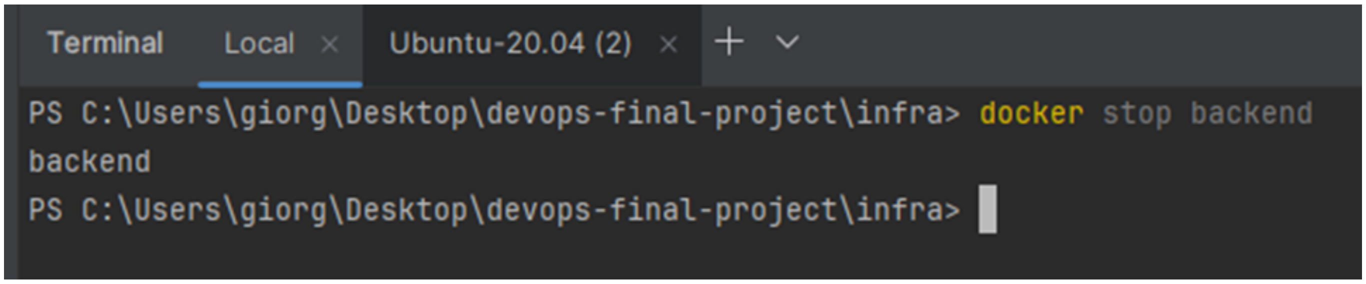
Prepared By: Giorgi Gagnidze

1. Incident Summary

On June 19, 2025, from 8:45 PM to 9:15 PM +04, **backend service (`backend` container)** was intentionally taken offline as part of a planned incident simulation. This action immediately rendered all backend functionalities, such as managing user tasks, creating and updating them. The primary goal of this exercise was to test the monitoring, alerting, and logging infrastructure, as well as observe the cascading effects on dependent services like the frontend or Gatling.

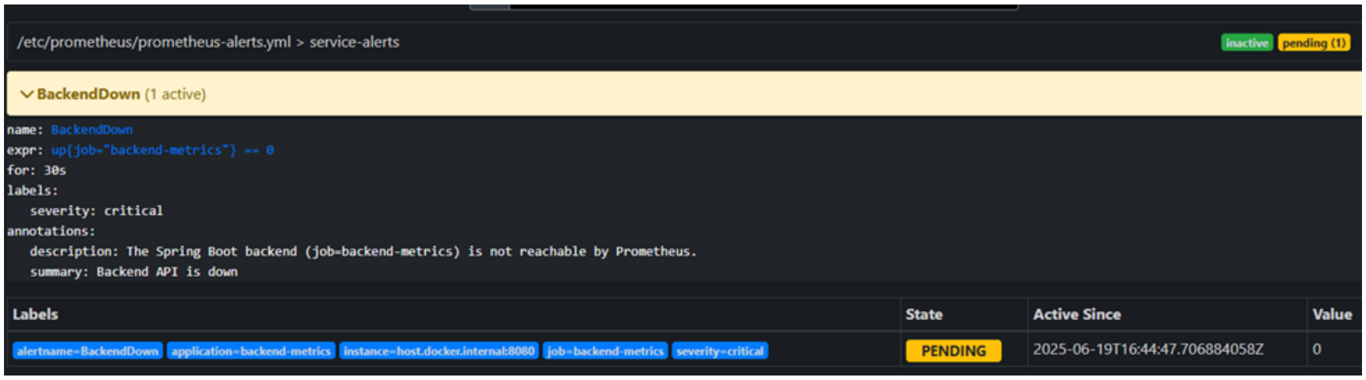
2. Timeline of Events

- **8:45 PM:** The `backend` Docker container was manually stopped using `docker stop backend`, simulating a critical service failure.



```
PS C:\Users\giorg\Desktop\devops-final-project\infra> docker stop backend
backend
PS C:\Users\giorg\Desktop\devops-final-project\infra>
```

- **8:46 PM: Prometheus** began detecting failed scrapes and increased error rates for the `backend` service's endpoints.



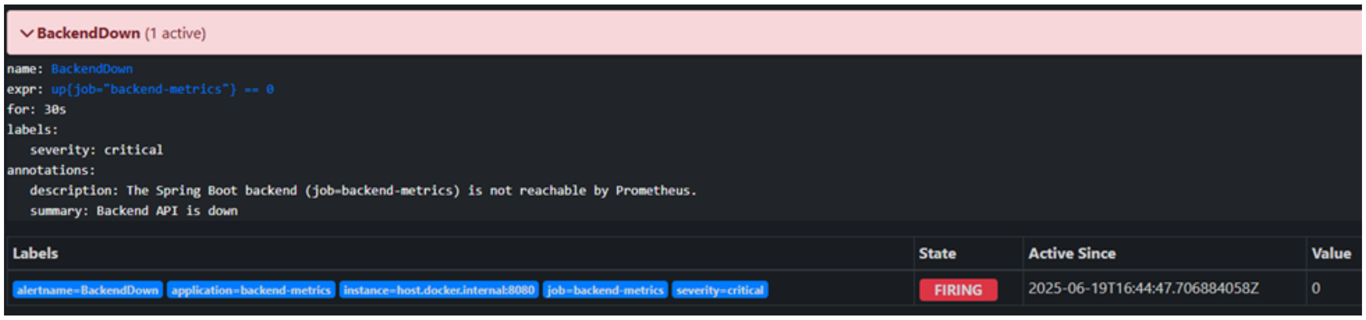
/etc/prometheus/prometheus-alerts.yml > service-alerts

BackendDown (1 active)

```
name: BackendDown
expr: up{job="backend-metrics"} == 0
for: 30s
labels:
  severity: critical
annotations:
  description: The Spring Boot backend (job=backend-metrics) is not reachable by Prometheus.
  summary: Backend API is down
```

Labels	State	Active Since	Value
alertname=BackendDown application=backend-metrics instance=host.docker.internal:8080 job=backend-metrics severity=critical	PENDING	2025-06-19T16:44:47.706884058Z	0

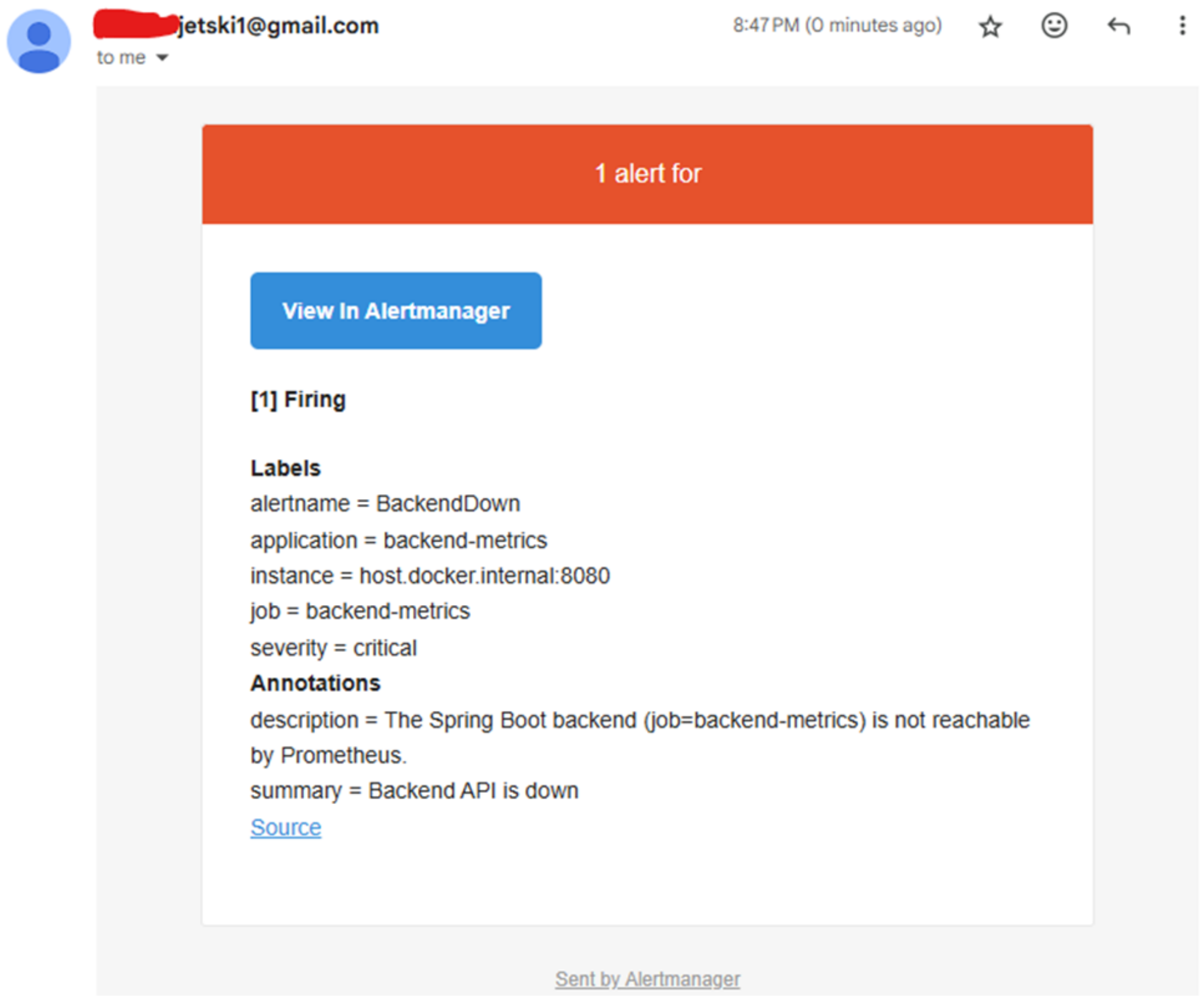
- **8:47 PM: Alertmanager** fired critical alerts, notifying the team of the `backend` service's unavailability based on predefined Prometheus alerting rules, in this case EMAIL.



BackendDown (1 active)

```
name: BackendDown
expr: up{job="backend-metrics"} == 0
for: 30s
labels:
  severity: critical
annotations:
  description: The Spring Boot backend (job=backend-metrics) is not reachable by Prometheus.
  summary: Backend API is down
```

Labels	State	Active Since	Value
alertname=BackendDown application=backend-metrics instance=host.docker.internal:8080 job=backend-metrics severity=critical	FIRING	2025-06-19T16:44:47.706884058Z	0



[redacted]jetski1@gmail.com
to me
8:47 PM (0 minutes ago)

1 alert for

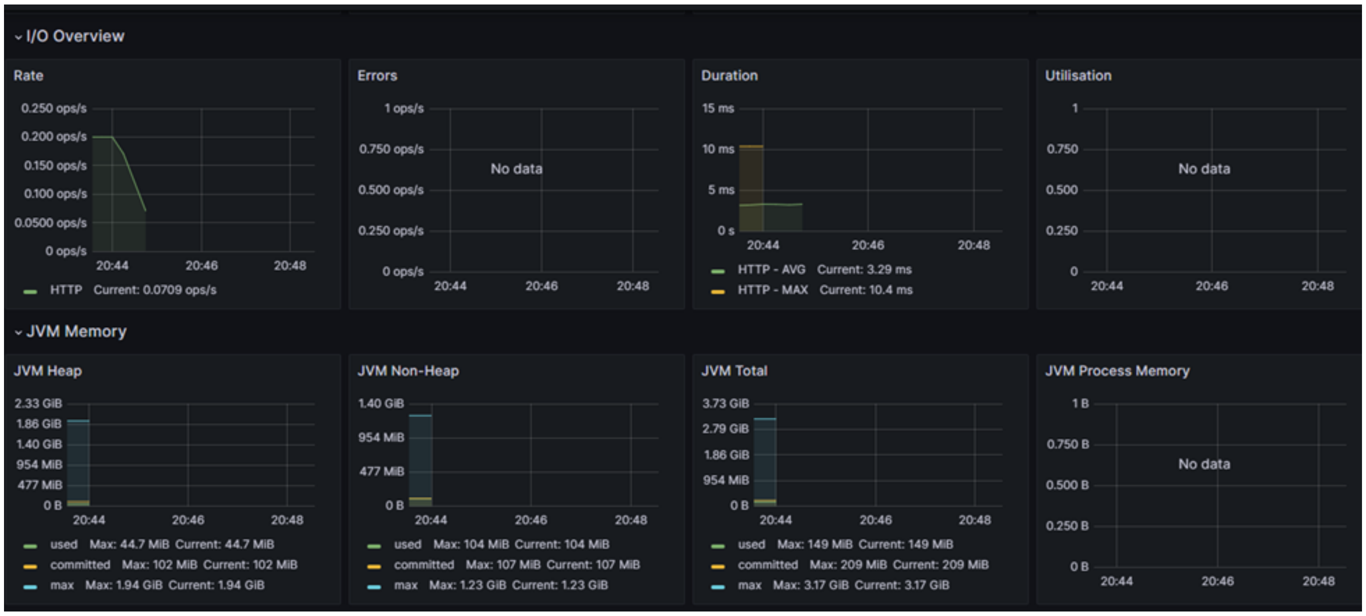
View In Alertmanager

[1] Firing

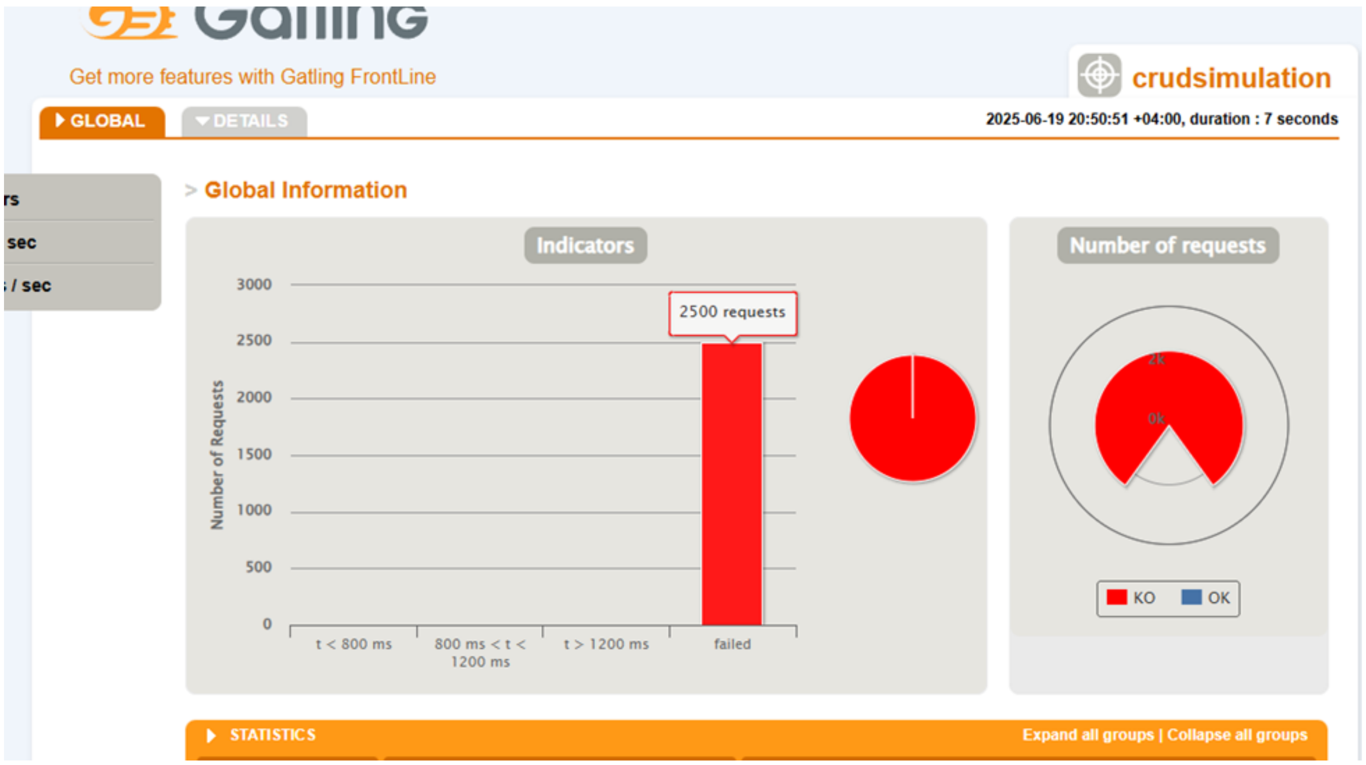
Labels
alertname = BackendDown
application = backend-metrics
instance = host.docker.internal:8080
job = backend-metrics
severity = critical
Annotations
description = The Spring Boot backend (job=backend-metrics) is not reachable by Prometheus.
summary = Backend API is down
Source

Sent by Alertmanager

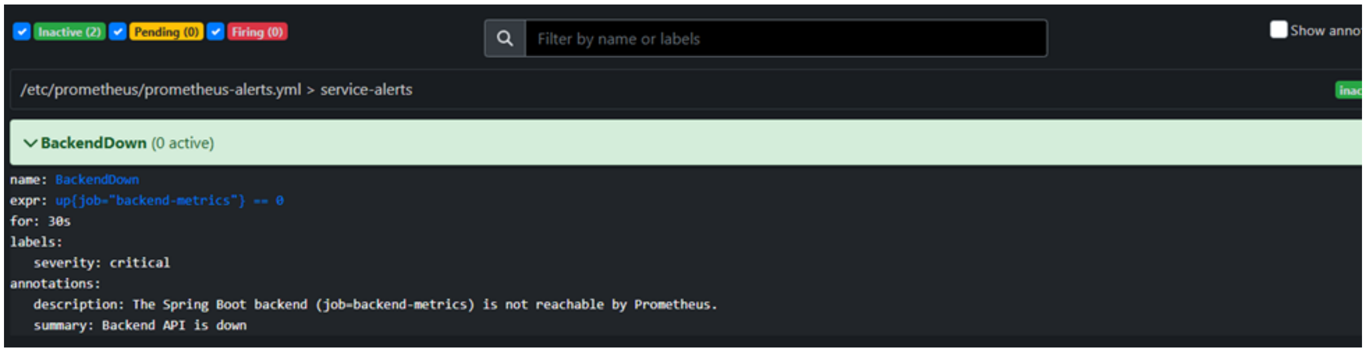
- **8:48 PM: Grafana dashboards** immediately reflected the outage, showing a complete drop in backend metrics.


I/O Overview
Rate
Errors
No data
Duration
Utilisation
No data
HTTP Current: 0.0709 ops/s
HTTP - AVG Current: 3.29 ms
HTTP - MAX Current: 10.4 ms
JVM Memory
JVM Heap
JVM Non-Heap
JVM Total
JVM Process Memory
No data
used Max: 44.7 MiB Current: 44.7 MiB
committed Max: 102 MiB Current: 102 MiB
max Max: 1.94 GiB Current: 1.94 GiB
used Max: 104 MiB Current: 104 MiB
committed Max: 107 MiB Current: 107 MiB
max Max: 1.23 GiB Current: 1.23 GiB
used Max: 149 MiB Current: 149 MiB
committed Max: 209 MiB Current: 209 MiB
max Max: 3.17 GiB Current: 3.17 GiB
20:44
20:46
20:48

- **8:50 PM: Gatling**, which was configured to run a `CrudSimulation` targeting the backend, began reporting 100% request failures, validating its ability to detect service degradation.



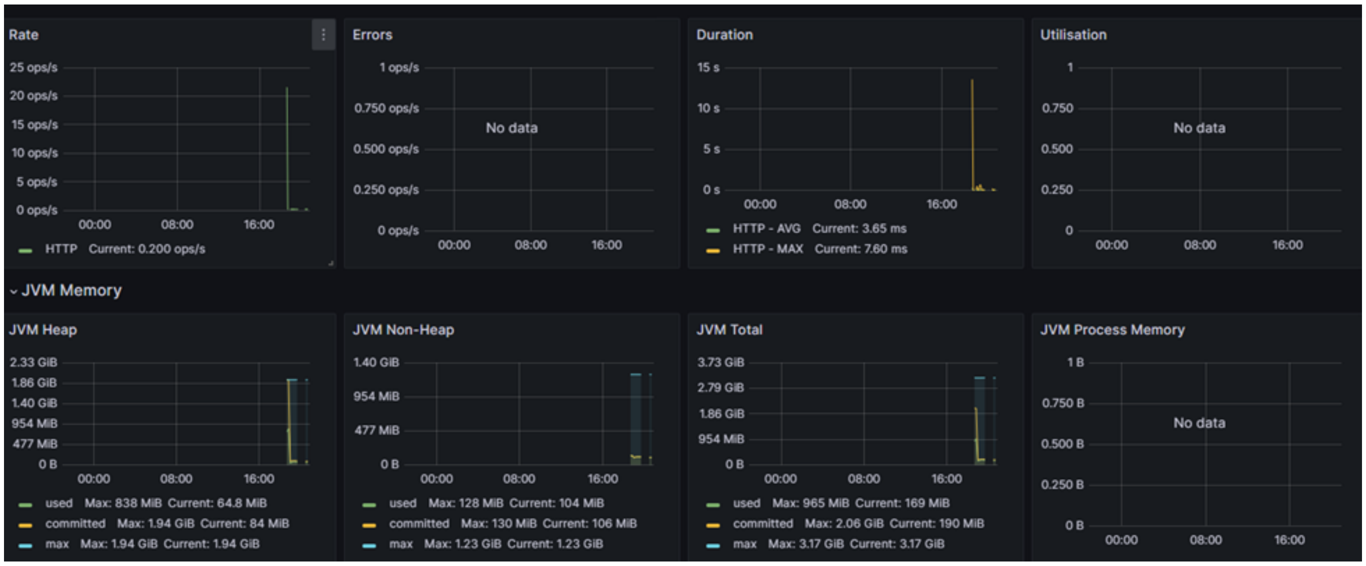
- 8:52 PM:** The `backend` Docker container was manually restarted using `docker start backend`.
- **8:54 PM:** Monitoring systems (Prometheus, Grafana) showed that the `backend` service was operational, with metrics returning to normal levels.



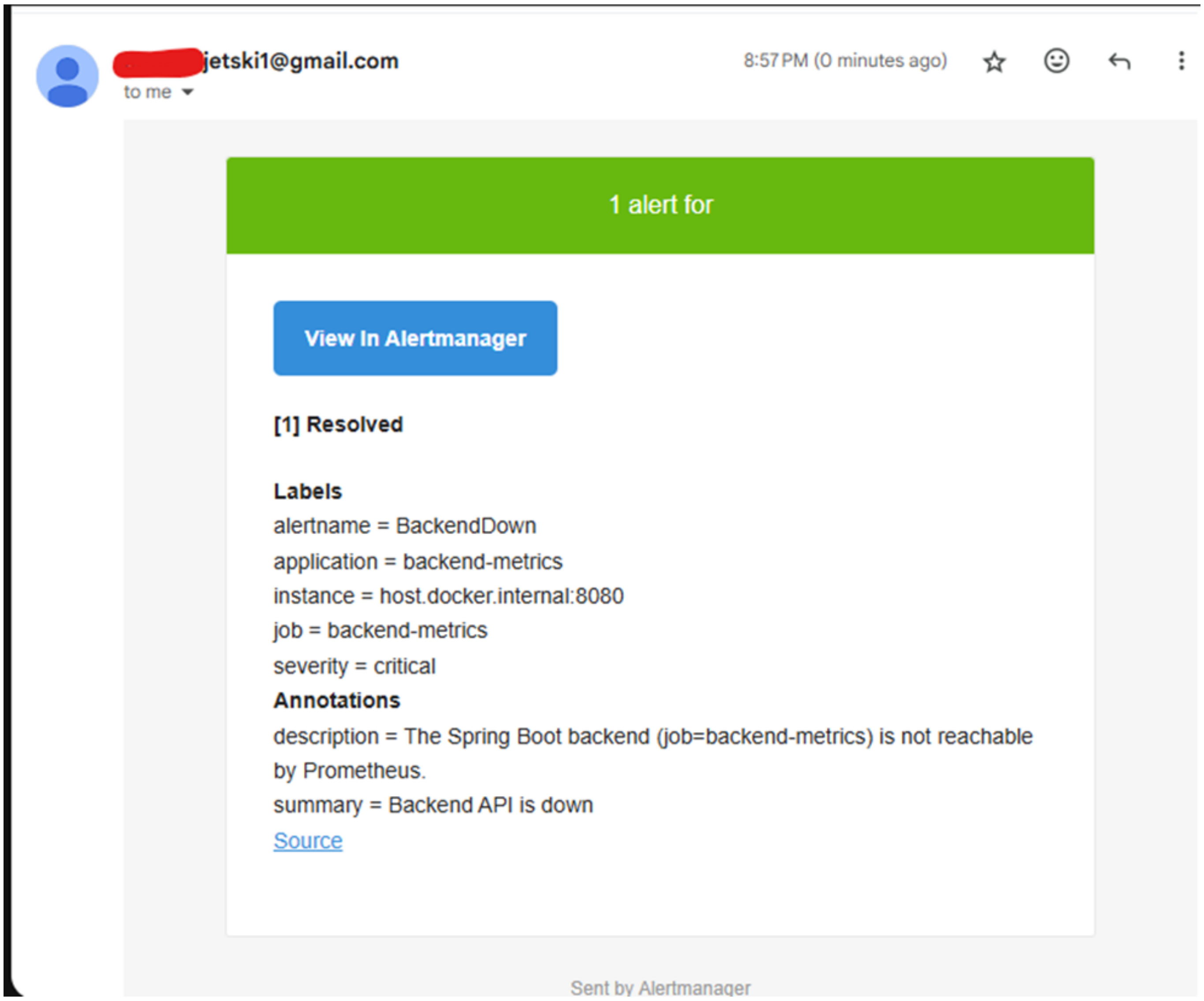
/etc/prometheus/prometheus-alerts.yml > service-alerts

BackendDown (0 active)

```
name: BackendDown
expr: up{job="backend-metrics"} == 0
for: 30s
labels:
   severity: critical
annotations:
   description: The Spring Boot backend (job=backend-metrics) is not reachable by Prometheus.
   summary: Backend API is down
```



- **8:55 PM:** Frontend functionality was fully restored, and Gatling began reporting successful requests again.
- 8:57 PM: **Alertmanager** reported that the system was back to normal and the issue was resolved. The incident simulation concluded.



[redacted]jetski1@gmail.com
to me
8:57 PM (0 minutes ago)

1 alert for

[1] Resolved

Labels
alertname = BackendDown
application = backend-metrics
instance = host.docker.internal:8080
job = backend-metrics
severity = critical
Annotations
description = The Spring Boot backend (job=backend-metrics) is not reachable by Prometheus.
summary = Backend API is down
Source

Sent by Alertmanager

3. Root Cause Analysis

The root cause of this incident was the **intentional manual shutdown of the `backend` Docker container**. This action was a deliberate part of a planned DevOps simulation to evaluate the observability stack and incident response capabilities, rather than surprise system failure.

4. Impact

- **System Functionality:** All functionalities reliant on the backend service were completely unavailable for 8 minutes.
- **User Experience:** The frontend application was unfunctional due to backend outage.
- **Observability Systems:**
  - **Prometheus & Alertmanager:** Successfully detected and alerted on the backend's unavailability.
  - **Grafana:** Provided an immediate and clear visual representation of the outage.
  - **Logstash & Kibana:** Displayed the shutdown of the service in logs

- **Performance Testing: Gatling** accurately reported a 100% failure rate for all simulated API calls, confirming its effectiveness in detecting service downtime.

5. What Went Well

- **Rapid Detection: Prometheus** monitoring and **Alertmanager** configuration proved highly effective, triggering alerts within seconds of the backend going offline.
- **Dependency Awareness: Gatling** service correctly reported failures due to the backend's unavailability, demonstrating a clear understanding of service dependencies within the ecosystem.
- **Efficient Recovery:** Restarting the `backend` container was swift and straightforward, leading to a quick service restoration.

6. What Went Wrong

- **No Automated Restart:** the recovery required manual intervention. For real-world scenarios, an automated restart mechanism would reduce recovery time.
- **Frontend Degradation:** Front end did not display alerts, errors and didn't handle error statuses due to backend outage

7. Lessons Learned

- Current monitoring and logging infrastructure provides a solid foundation for detecting and understanding critical service outages.
- Automated incident response mechanisms are crucial for minimizing downtime in production environments.
- Regular simulations are invaluable for validating tools, processes, and readiness for real-world incidents.