## **Barista project infrastructure and monitoring stack**

## 

## **☕ Barista Café DevOps Stack Overview**

### **🧱 Core Components**

| **Component** | **Usage** | **Why Chosen** | **Alternatives** |
| --- | --- | --- | --- |
| **Docker** | Containerized frontend & backend | Easy packaging, portability | Podman, Buildah, containerd |
| **Kubernetes** | Orchestrate frontend, backend, and DB | Industry standard, auto-scaling, mgmt | Docker Swarm, Nomad, OpenShift |
| **PostgreSQL** | Backend database | Stable, SQL-compliant, great tooling | MySQL, SQLite, MongoDB |
| **Flask** | Python web API backend | Lightweight, simple REST support | FastAPI, Django, Express.js (Node.js) |
| **HTML + JS** | Frontend for reservation | Custom design via Barista template | React, Vue.js, Angular |

## **📊 Monitoring Stack**

| **Component** | **Usage** | **Why Chosen** | **Alternatives** |
| --- | --- | --- | --- |
| **Prometheus** | Time-series metrics scraper | Native with Kubernetes, simple setup | Zabbix, InfluxDB + Telegraf, Graphite |
| **Node Exporter** | Expose Linux server hardware metrics | Official Prometheus agent, very lightweight | Netdata, Collectd, Telegraf |
| **Grafana** | Visualize Prometheus metrics | Powerful dashboards, Prometheus native | Kibana (Elasticsearch), Chronograf |

## 

## 

## 

## **🌐 Server-Level Setup**

### **✅ You have:**

| **Server IP** | **Purpose** |
| --- | --- |
| 192.168.48.151 | Node Exporter installed |
| 192.168.48.142 | Prometheus + Grafana server |

## **🌐 Server-Level Setup**

### **✅ You have:**

| **Server IP** | **Purpose** |
| --- | --- |
| 192.168.48.151 | Node Exporter installed |
| 192.168.48.142 | Prometheus + Grafana server |

### **2. Prometheus**

* **Used**: To collect metrics from Node Exporter and make them queryable.
* **Why**: Simple config, wide community adoption, and works with Grafana out of the box.
* **Alternative**: InfluxDB (especially with Telegraf), Zabbix (more enterprise), OpenTelemetry.

### **3. Grafana**

* **Used**: To visualize data from Prometheus.
* **Why**: Customizable dashboards, alerting, and open-source.
* **Alternative**: Chronograf (with InfluxDB), Kibana (for logs + metrics with Elasticsearch).

### **4. Manual Prometheus Configuration**

* **Used**: Local prometheus.yml with static targets.
* **Why**: Simple and direct for your small-scale setup.
* **Alternative**:  
  + **Kubernetes**: Use ServiceMonitors + Prometheus Operator
  + **Dynamic service discovery** via Consul or file-based SD

### **5. Manual Binary Runs & Systemd Services**

* **Used**: To run Prometheus and Node Exporter as background services.
* **Why**: Easy to understand and control, works well on Ubuntu.
* **Alternative**: Run both in Docker or Kubernetes (with Helm charts), or as snap packages.

## **🧠 DevOps Good Practice Summary**

* ✅ You containerized apps using Docker → good for versioning and portability
* ✅ You orchestrated with Kubernetes → great for scaling
* ✅ You set up Prometheus + Grafana monitoring → observability in place
* ✅ You kept infrastructure flexible and OS-native (Ubuntu + systemd)

## **🛠️ Next Steps You Could Consider**

* Monitor **Flask backend** metrics via prometheus\_client
* Use a **PostgreSQL Exporter** for DB health (dashboard: 9628)
* Set up **alerts in Grafana** (CPU, memory, reservation spikes)
* Use **Helm** to manage Grafana + Prometheus (for future clusters)