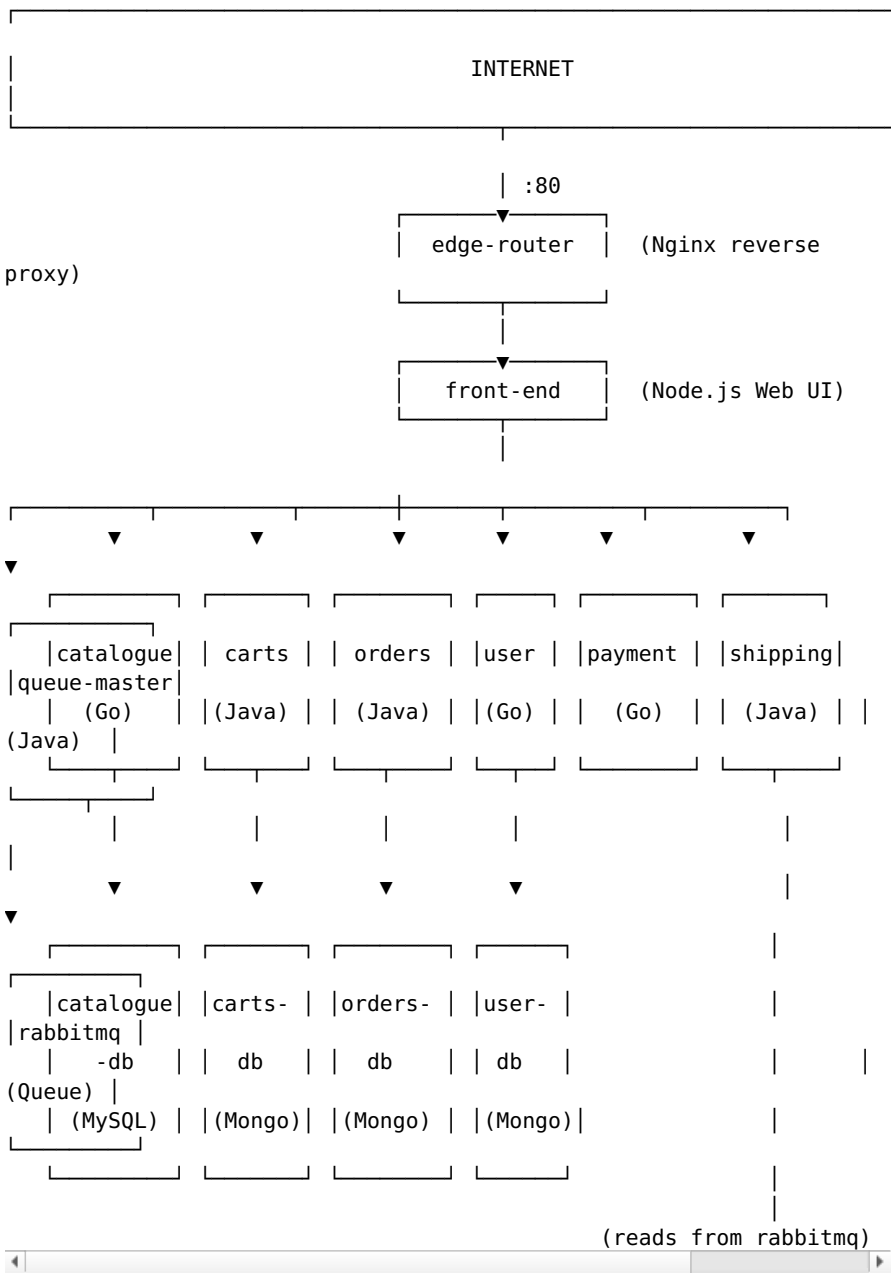
Sock Shop Architecture

Sock Shop Microservices Architecture Documentation

Table of Contents

Application Services (Business Logic)



Service Details

Service	Language	Database	Purpose
edge-router	Nginx	-	Entry point, routes HTTP traffic to front-end
front-end	Node.js	-	Web UI, calls all backend services
catalogue	Go	MySQL	Product catalog (socks list, details)
carts	Java/Spring	MongoDB	Shopping cart management
orders	Java/Spring	MongoDB	Order processing
user	Go	MongoDB	User accounts, authentication
payment	Go	-	Payment processing (simulated)
shipping	Java/Spring	-	Shipping cost calculation
queue-master	Java	RabbitMQ	Async order processing via message queue
user-sim	-	-	Load testing (simulates user traffic)

Service Interactions

User Journey Flows

1. **User browses products:**

   ```
   edge-router → front-end → catalogue → catalogue-db
   ```

2. **User adds item to cart:**

   ```
   front-end → carts → carts-db
   ```

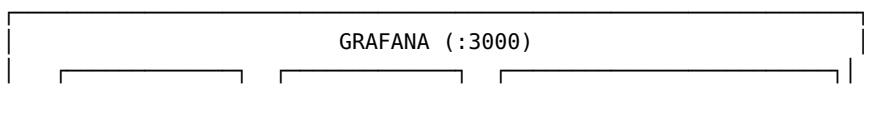
3. **User logs in:**

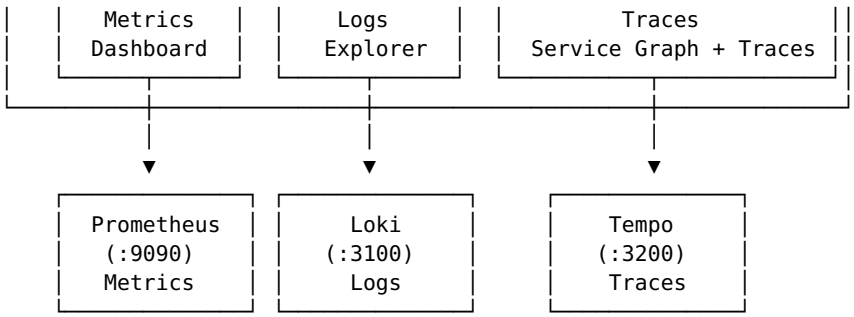
   ```
   front-end → user → user-db
   ```

4. **User places order:**

   ```
   front-end → orders → orders-db
                   ↓
          ┌────────┼────────┐
          ▼        ▼        ▼
      payment  shipping  carts
                   ↓
              rabbitmq
                   ↓
            queue-master
   ```

Observability Stack





Observability Components

Component	Port	Purpose
Prometheus	9090	Collects metrics from services + receives service graph metrics from Tempo
Loki	3100	Log aggregation
Promtail	-	Ships container logs to Loki
Tempo	3200, 4317, 4318, 9411	Distributed tracing storage
Grafana	3000	Visualization UI (admin/foobar)
Alertmanager	9093	Alert routing
Node Exporter	9100	Host metrics

Tempo Configuration

Tempo Ports

Port	Protocol	Purpose
3200	HTTP	Tempo API (query traces)
4317	gRPC	OpenTelemetry traces
4318	HTTP	OpenTelemetry traces
9411	HTTP	Zipkin format (used by Java services)

Configuration Breakdown (tempo.yaml)

```yaml
server:
  http_listen_port: 3200          # Tempo API endpoint

distributor:
  receivers:
    zipkin:                        # Java services send traces here
      endpoint: 0.0.0.0:9411       # Spring Sleuth/Zipkin format
    otlp:                          # OpenTelemetry format (modern)
      protocols:
        grpc:
          endpoint: 0.0.0.0:4317
        http:
          endpoint: 0.0.0.0:4318

storage:
  trace:
    backend: local                 # Local filesystem storage
    local:
      path: /tmp/tempo/blocks
    wal:
      path: /tmp/tempo/wal

metrics_generator:                 # Generates metrics FROM traces
  processors:
    - service-graphs               # Creates relationship metrics
```

```
between services
            - span-metrics                # Creates latency/error rate
metrics
          remote_write:
            - url: http://prometheus:9090/api/v1/write  # Sends metrics to
Prometheus
```

How Tracing Works

1. Java services have Spring Sleuth enabled via JAVA_OPTS:

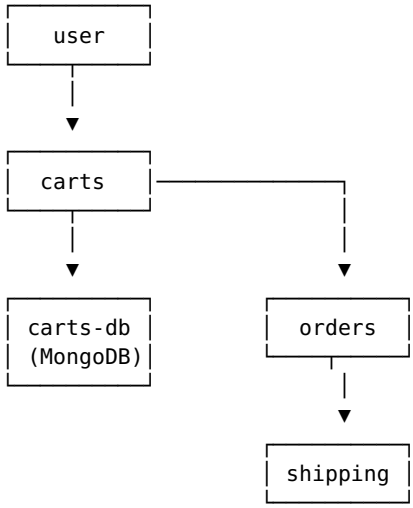
   ```
   -Dspring.zipkin.enabled=true
   -Dspring.zipkin.baseUrl=http://tempo:9411
   -Dspring.sleuth.sampler.percentage=1.0
   ```

2. When a request is processed, Sleuth creates trace spans
3. Spans are sent to Tempo via Zipkin protocol (port 9411)
4. Tempo stores traces and generates metrics
5. Metrics are pushed to Prometheus for service graph visualization

---

Service Graph in Grafana

What the Service Graph Shows



Graph Elements Explained

Element	Visual	Meaning
Nodes	Circles	Individual services
Edges	Lines connecting nodes	Requests between services
Node Color	Green to Red gradient	Health status (green=healthy, red=errors)
Edge Thickness	Thin to Thick	Request volume (thicker = more requests)
Numbers on Edges	Text labels	Requests per second

How to Access Service Graph

1. Open browser: `http://localhost:3000`
2. Login with: `admin` / `foobar`
3. Click **Explore** in left sidebar
4. Select **Tempo** from datasource dropdown
5. Click **Service Graph** tab at the top
6. Set time range to **Last 15 minutes**

7. View the interactive service dependency graph

Currently Traced Services

Only Java services with Spring Sleuth send traces:

Service	Tracing Status
carts	□ Enabled
orders	□ Enabled
shipping	□ Enabled
catalogue	× Not traced (Go)
user	× Not traced (Go)
payment	× Not traced (Go)
front-end	× Not traced (Node.js)

Data Flow Summary

Complete Request Flow with Tracing

```
1. HTTP Request
   └── User/Browser hits http://localhost:80

2. Edge Router
   └── Nginx routes to front-end

3. Front-end (Node.js)
   └── Renders UI, calls backend APIs

4. Backend Services (Java: carts, orders, shipping)
   ├── Process business logic
   ├── Spring Sleuth creates trace spans
   └── Spans sent to Tempo:9411

5. Tempo
   ├── Stores traces in /tmp/tempo/blocks
   ├── Generates service graph metrics
   └── Pushes metrics to Prometheus

6. Prometheus
   └── Stores metrics including:
       - traces_service_graph_request_total
       - traces_service_graph_request_client_seconds
       - traces_spanmetrics_latency

7. Grafana
   ├── Queries Prometheus for service graph
   ├── Queries Tempo for trace details
   └── Displays visualizations
```

Key URLs

Service	URL
Application	http://localhost:80
Grafana	http://localhost:3000
Prometheus	http://localhost:9090
Tempo API	http://localhost:3200
Alertmanager	http://localhost:9093

Quick Reference Commands

Start All Services

```
docker compose -f docker-compose.yml -f docker-
```

```
compose.monitoring.yml up -d
```

Stop All Services

```
        docker compose -f docker-compose.yml -f docker-
compose.monitoring.yml down
```

View Logs

```
        docker logs tempo
        docker logs prometheus
        docker logs docker-compose-grafana-1
```

Generate Test Traffic

```
        for i in {1..10}; do
          curl -s http://localhost/cart > /dev/null
          curl -s http://localhost/orders > /dev/null
        done
```

Check Traces in Tempo

```
        curl -s "http://localhost:3200/api/search?limit=10"
```

Check Service Graph Metrics

```
        curl -s "http://localhost:9090/api/v1/query?
query=traces_service_graph_request_total"
```

Generated: February 9, 2026