REACT-SRE-APP-PROJECT:

Firstly we ran the complete-sre-react-script to create an react-sre-project folder with all dependencies:

CODE:

```
#!/bin/bash

# Complete React SRE Application Setup Script
# This script implements all requirements including:
# - React application with SRE concepts
# - Minikube setup with 60s wait time for errors
# - All dependencies for Python 3.9+
# - Fixes for kubectl version commands
# - Handling for wzegh library
# - Grafana dashboard setup
# - Prometheus with 300s timeout
# - WSL and Minikube integration

# Set strict error handling
set -e

# Define color codes for better output readability
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[0;33m'
BLUE='\033[0;34m'
NC='\033[0m' # No Color

# Define project directories
PROJECT_ROOT="$HOME/react-sre-project"
REACT_APP_DIR="$PROJECT_ROOT/sre-react-app"
K8S_DIR="$PROJECT_ROOT/kubernetes"
MONITORING_DIR="$PROJECT_ROOT/monitoring"
SCRIPTS_DIR="$PROJECT_ROOT/scripts"

# SRE-specific variables
MINIKUBE_WAIT_TIMEOUT=60  # Seconds to wait for Minikube operations
PROMETHEUS_TIMEOUT=300    # Seconds to wait for Prometheus to start

# ===== Helper Functions =====

# Function for logging with timestamps
log() {
  local level=$1
  local message=$2
```

```
  local timestamp=$(date "+%Y-%m-%d %H:%M:%S")

  case $level in
    "INFO")
      echo -e "${GREEN}[INFO]${NC} $timestamp - $message"
      ;;
    "WARN")
      echo -e "${YELLOW}[WARN]${NC} $timestamp - $message"
      ;;
    "ERROR")
      echo -e "${RED}[ERROR]${NC} $timestamp - $message"
      ;;
    *)
      echo -e "${BLUE}[$level]${NC} $timestamp - $message"
      ;;
  esac
}

# Function to wait for a command to succeed with timeout
wait_for_command() {
  local command=$1
  local timeout=${2:-60}  # Default timeout is 60 seconds
  local interval=${3:-5}  # Default check interval is 5 seconds
  local elapsed=0
  local start_time=$(date +%s)

  log "INFO" "Waiting for command to succeed: $command (timeout: ${timeout}s)"

  while [ $elapsed -lt $timeout ]; do
    if eval $command &>/dev/null; then
      log "INFO" "Command succeeded after ${elapsed}s"
      return 0
    fi

    sleep $interval
    elapsed=$(($(date +%s) - start_time))
    log "INFO" "Still waiting... (${elapsed}s elapsed)"
  done

  log "ERROR" "Command timed out after ${elapsed}s: $command"
  return 1
}

# Function to check if a command exists
command_exists() {
  command -v "$1" &> /dev/null
}
```

```
# Function to create a file with content
create_file() {
  local filepath=$1
  local content=$2

  mkdir -p "$(dirname "$filepath")"
  echo "$content" > "$filepath"
  log "INFO" "Created file: $filepath"
}

# ===== Step 1: Initial Setup =====

setup_project_directories() {
  log "INFO" "Creating project directory structure"
  mkdir -p "$PROJECT_ROOT" "$REACT_APP_DIR" "$K8S_DIR" "$MONITORING_DIR"
"$SCRIPTS_DIR"
  log "INFO" "Project directories created at $PROJECT_ROOT"
}

check_wsl() {
  log "INFO" "Checking WSL environment..."

  if ! grep -qEi "(microsoft|wsl)" /proc/version; then
    log "ERROR" "This script must be run within WSL"
    exit 1
  fi

  log "INFO" "WSL detected, continuing with setup"
}

# ===== Step 2: Install Dependencies =====

install_dependencies() {
  log "INFO" "Installing system dependencies..."

  # Update package lists
  log "INFO" "Updating package lists"
  sudo apt update || {
    log "ERROR" "Failed to update package lists"
    exit 1
  }

  # Install required packages
  log "INFO" "Installing required packages"
  sudo apt install -y curl wget apt-transport-https ca-certificates \
                      software-properties-common git jq unzip \
                      gnupg lsb-release || {
    log "ERROR" "Failed to install basic packages"
```

```
    exit 1
  }

  # Install Docker if not already installed
  if ! command_exists docker; then
    log "INFO" "Installing Docker..."
    curl -fsSL https://get.docker.com -o get-docker.sh
    sudo sh get-docker.sh
    sudo usermod -aG docker $USER
    log "WARN" "You may need to log out and back in for Docker group changes
to take effect"
  else
    log "INFO" "Docker is already installed"
  fi

  # Install Python 3.9+ if not available
  if ! command_exists python3.9; then
    log "INFO" "Installing Python 3.9+..."
    sudo add-apt-repository -y ppa:deadsnakes/ppa
    sudo apt update
    sudo apt install -y python3.9 python3.9-venv python3.9-dev
  else
    log "INFO" "Python 3.9+ is already installed"
  fi

  # Set Python 3.9 as default python3
  sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.9 1

  # Install pip for Python 3.9
  if ! command_exists pip3; then
    log "INFO" "Installing pip for Python 3.9..."
    curl -sS https://bootstrap.pypa.io/get-pip.py | python3.9
  fi

  # Install Node.js using nvm if not already installed
  if ! command_exists node || ! command_exists npm; then
    log "INFO" "Installing Node.js using nvm..."
    curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.sh |
bash
    export NVM_DIR="$HOME/.nvm"
    [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
    nvm install 16

    # Verify installation
    if command_exists node && command_exists npm; then
      log "INFO" "Node.js $(node -v) and npm $(npm -v) installed successfully"
    else
```

```
      log "ERROR" "Failed to install Node.js and npm"
      exit 1
    fi
  else
    log "INFO" "Node.js $(node -v) and npm $(npm -v) are already installed"
  fi

  # Install kubectl
  if ! command_exists kubectl; then
    log "INFO" "Installing kubectl..."
    curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
    chmod +x kubectl
    sudo mv kubectl /usr/local/bin/

    # Fix for the kubectl --short issue
    # Create an alias in bashrc
    echo 'alias kubectly="kubectl version"' >> ~/.bashrc
    source ~/.bashrc
  else
    log "INFO" "kubectl is already installed: $(kubectl version --client --
output=yaml | grep gitVersion)"
  fi

  # Install Minikube
  if ! command_exists minikube; then
    log "INFO" "Installing Minikube..."
    curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64
    chmod +x minikube-linux-amd64
    sudo mv minikube-linux-amd64 /usr/local/bin/minikube
  else
    log "INFO" "Minikube is already installed: $(minikube version | grep
version)"
  fi

  # Install Helm
  if ! command_exists helm; then
    log "INFO" "Installing Helm..."
    curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
    chmod +x get_helm.sh
    ./get_helm.sh
    rm get_helm.sh
  else
    log "INFO" "Helm is already installed: $(helm version --short)"
  fi
```

```
  log "INFO" "All dependencies installed successfully"
}

# ===== Step 3: Python Virtual Environment and Dependencies =====

setup_python_environment() {
  log "INFO" "Setting up Python virtual environment..."

  # Create and activate virtual environment
  cd "$PROJECT_ROOT"
  python3.9 -m venv venv
  source venv/bin/activate

  # Upgrade pip
  pip install --upgrade pip

  # Install required Python packages
  log "INFO" "Installing Python dependencies..."

  # Create requirements.txt
  cat > "$PROJECT_ROOT/requirements.txt" << EOL
kubernetes==26.1.0
prometheus-client==0.16.0
PyYAML==6.0
requests==2.28.2
grafanalib==0.7.0
Flask==2.2.3
gunicorn==20.1.0
psutil==5.9.4
pydantic==1.10.7
pytest==7.3.1
pytest-cov==4.1.0
EOL

  # Fix for missing wzegh library - adding it if it doesn't exist
  # This is a mock since "wzegh" is not a real library
  cat >> "$PROJECT_ROOT/requirements.txt" << EOL
# Mock wzegh library - replace with actual requirement if needed
EOL

  # Create a mock wzegh library if it doesn't exist
  mkdir -p "$PROJECT_ROOT/wzegh"
  cat > "$PROJECT_ROOT/wzegh/__init__.py" << EOL
"""
Mock implementation of the wzegh library.
This is a placeholder for the actual library.
"""
```

```
def initialize():
    """Initialize the wzegh library."""
    print("Initializing wzegh library (mock)")
    return True

def configure(config):
    """Configure the wzegh library."""
    print(f"Configuring wzegh library with: {config}")
    return True
EOL

  # Create setup.py for the mock wzegh library
  cat > "$PROJECT_ROOT/wzegh/setup.py" << EOL
from setuptools import setup, find_packages

setup(
    name="wzegh",
    version="0.1.0",
    packages=find_packages(),
    description="Mock implementation of wzegh library",
)
EOL

  # Install the mock wzegh library
  cd "$PROJECT_ROOT/wzegh"
  pip install -e .
  cd "$PROJECT_ROOT"

  # Install requirements
  pip install -r requirements.txt

  log "INFO" "Python dependencies installed successfully"
}

# ===== Step 4: Create React Application with SRE Instrumentation =====

create_react_app() {
  # Ensure old app is deleted
REACT_APP_DIR="$HOME/react-sre-project/sre-react-app"
if [ -d "$REACT_APP_DIR" ]; then
  echo "[WARN] React app directory exists. Deleting it..."
  rm -rf "$REACT_APP_DIR"
fi

# Create the React app (without specifying a template)
echo "[INFO] Creating a new React app..."
npx create-react-app sre-react-app
```

```
# Navigate to the React app directory
cd "$REACT_APP_DIR"

# Manually install React 18 (to avoid conflicts with @grafana/ui)
echo "[INFO] Downgrading React to version 18..."
npm install react@18 react-dom@18


  # Install additional dependencies for SRE
  npm install --save \
    react@18 react-dom@18 \
    axios \
    react-router-dom \
    swr \
    prom-client \
    react-error-boundary \
    @grafana/ui \
    history

  log "INFO" "React application created successfully"
}

# Create React app components with SRE instrumentation
create_react_sre_files() {
  log "INFO" "Creating React components with SRE instrumentation..."

  # Create src/components directory
  mkdir -p "$REACT_APP_DIR/src/components"
  mkdir -p "$REACT_APP_DIR/src/services"

  # Create HealthCheck component
  cat > "$REACT_APP_DIR/src/components/HealthCheck.jsx" << 'EOL'
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const HealthCheck = () => {
  const [health, setHealth] = useState({
    status: 'Loading...',
    dependencies: [],
    uptime: 0,
  });

  useEffect(() => {
    const fetchHealth = async () => {
      try {
        const response = await axios.get('/api/health');
        setHealth(response.data);
      } catch (error) {
```

```
        setHealth({
          status: 'Unhealthy',
          dependencies: [],
          uptime: 0,
          error: error.message,
        });
      }
    };

    fetchHealth();
    const interval = setInterval(fetchHealth, 30000); // Check every 30
seconds

    return () => clearInterval(interval);
  }, []);

  return (
    <div className="health-check">
      <h2>Application Health</h2>
      <p>Status: <span className={health.status === 'Healthy' ? 'healthy' :
'unhealthy'}>
        {health.status}
      </span></p>

      {health.dependencies.length > 0 && (
        <>
          <h3>Dependencies:</h3>
          <ul>
            {health.dependencies.map((dep, index) => (
              <li key={index}>
                {dep.name}: {dep.status}
              </li>
            ))}
          </ul>
        </>
      )}

      <p>Uptime: {Math.floor(health.uptime / 60)} minutes</p>

      {health.error && (
        <div className="error-message">
          <p>Error: {health.error}</p>
        </div>
      )}
    </div>
  );
};
```

```
export default HealthCheck;
EOL

  # Create ErrorBoundary component
  cat > "$REACT_APP_DIR/src/components/ErrorBoundary.jsx" << 'EOL'
import React from 'react';

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false, error: null, errorInfo: null };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // You can log the error to an error reporting service
    console.error("Uncaught error:", error, errorInfo);

    // Log to metrics system if available
    if (window.logError) {
      window.logError(error, errorInfo);
    }

    this.setState({
      error: error,
      errorInfo: errorInfo
    });
  }

  render() {
    if (this.state.hasError) {
      // Custom fallback UI
      return (
        <div className="error-container">
          <h2>Something went wrong.</h2>
          <details style={{ whiteSpace: 'pre-wrap' }}>
            {this.state.error && this.state.error.toString()}
            <br />
            {this.state.errorInfo && this.state.errorInfo.componentStack}
          </details>
          <button
            onClick={() => window.location.reload()}
            className="reload-button"
          >
```

```
            Reload Application
          </button>
        </div>
      );
    }

    return this.props.children;
  }
}

export default ErrorBoundary;
EOL

  # Create metrics service
  cat > "$REACT_APP_DIR/src/services/metrics.js" << 'EOL'
// metrics.js - Prometheus client for React application

// Mock implementation of Prometheus client for frontend
class Counter {
  constructor(config) {
    this.name = config.name;
    this.help = config.help;
    this.labelNames = config.labelNames || [];
    this.value = 0;
  }

  inc(labels = {}, value = 1) {
    this.value += value;
    console.debug(`Counter ${this.name} incremented by ${value}`, labels);
    this._sendToBackend(this.name, this.value, labels);
  }

  _sendToBackend(name, value, labels) {
    // In a real implementation, this would send data to a backend collector
    if (navigator.sendBeacon) {
      const data = {
        name,
        value,
        labels,
        timestamp: Date.now()
      };
      navigator.sendBeacon('/api/metrics', JSON.stringify(data));
    }
  }
}

class Gauge {
  constructor(config) {
```

```
    this.name = config.name;
    this.help = config.help;
    this.labelNames = config.labelNames || [];
    this.value = 0;
  }

  set(value, labels = {}) {
    this.value = value;
    console.debug(`Gauge ${this.name} set to ${value}`, labels);
    this._sendToBackend(this.name, this.value, labels);
  }

  inc(labels = {}, value = 1) {
    this.value += value;
    console.debug(`Gauge ${this.name} incremented by ${value}`, labels);
    this._sendToBackend(this.name, this.value, labels);
  }

  dec(labels = {}, value = 1) {
    this.value -= value;
    console.debug(`Gauge ${this.name} decremented by ${value}`, labels);
    this._sendToBackend(this.name, this.value, labels);
  }

  _sendToBackend(name, value, labels) {
    // In a real implementation, this would send data to a backend collector
    if (navigator.sendBeacon) {
      const data = {
        name,
        value,
        labels,
        timestamp: Date.now()
      };
      navigator.sendBeacon('/api/metrics', JSON.stringify(data));
    }
  }
}

class Histogram {
  constructor(config) {
    this.name = config.name;
    this.help = config.help;
    this.labelNames = config.labelNames || [];
    this.buckets = config.buckets || [0.1, 0.5, 1, 2.5, 5, 10];
    this.values = [];
  }

  observe(labels = {}, value) {
```

```
    this.values.push(value);
    console.debug(`Histogram ${this.name} observed ${value}`, labels);
    this._sendToBackend(this.name, value, labels);
  }

  _sendToBackend(name, value, labels) {
    // In a real implementation, this would send data to a backend collector
    if (navigator.sendBeacon) {
      const data = {
        name,
        value,
        labels,
        timestamp: Date.now(),
        type: 'histogram'
      };
      navigator.sendBeacon('/api/metrics', JSON.stringify(data));
    }
  }
}

// Initialize metrics
const pageLoadTime = new Histogram({
  name: 'page_load_time_seconds',
  help: 'Time taken to load the page',
  buckets: [0.1, 0.5, 1, 2, 5, 10]
});

const navigationCount = new Counter({
  name: 'navigation_count',
  help: 'Number of page navigations',
  labelNames: ['route']
});

const jsErrors = new Counter({
  name: 'js_errors',
  help: 'JavaScript errors',
  labelNames: ['type']
});

const memoryUsage = new Gauge({
  name: 'memory_usage_bytes',
  help: 'Memory usage in bytes'
});

// Track page load time
if (window.performance) {
  window.addEventListener('load', () => {
    const pageLoadMetrics = window.performance.timing;
```

```
    const loadTime = (pageLoadMetrics.loadEventEnd -
pageLoadMetrics.navigationStart) / 1000;
    pageLoadTime.observe({}, loadTime);
  });
}

// Track memory usage
const trackMemoryUsage = () => {
  if (window.performance && window.performance.memory) {
    const memory = window.performance.memory;
    memoryUsage.set(memory.usedJSHeapSize);
  }
};

setInterval(trackMemoryUsage, 30000);

// Track errors
window.logError = (error, info) => {
  const errorType = error.name || 'unknown';
  jsErrors.inc({ type: errorType });
};

// Track navigation
const trackNavigation = (route) => {
  navigationCount.inc({ route });
};

export default {
  trackNavigation,
  trackMemoryUsage,
  pageLoadTime,
  navigationCount,
  jsErrors,
  memoryUsage
};
EOL

  # Update App.js with SRE components
  cat > "$REACT_APP_DIR/src/App.js" << 'EOL'
import React, { useEffect } from 'react';
import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-
dom';
import './App.css';
import HealthCheck from './components/HealthCheck';
import ErrorBoundary from './components/ErrorBoundary';
import metrics from './services/metrics';

// Home component
```

```
const Home = () => {
  useEffect(() => {
    metrics.trackNavigation('home');
  }, []);

  return (
    <div>
      <h1>SRE React Application</h1>
      <p>Welcome to the SRE-instrumented React application</p>
    </div>
  );
};

// Dashboard component
const Dashboard = () => {
  useEffect(() => {
    metrics.trackNavigation('dashboard');
  }, []);

  return (
    <div>
      <h1>SRE Dashboard</h1>
      <p>This page shows SRE metrics for the application</p>
      <HealthCheck />
    </div>
  );
};

// ErrorTest component to simulate errors
const ErrorTest = () => {
  useEffect(() => {
    metrics.trackNavigation('error-test');
  }, []);

  const causeError = () => {
    throw new Error('This is a test error');
  };

  return (
    <div>
      <h1>Error Testing</h1>
      <p>Click the button below to trigger an error</p>
      <button onClick={causeError}>Trigger Error</button>
    </div>
  );
};

function App() {
```

```
  return (
    <ErrorBoundary>
      <Router>
        <div className="App">
          <nav>
            <ul>
              <li>
                <Link to="/">Home</Link>
              </li>
              <li>
                <Link to="/dashboard">SRE Dashboard</Link>
              </li>
              <li>
                <Link to="/error-test">Error Test</Link>
              </li>
            </ul>
          </nav>

          <Switch>
            <Route path="/dashboard">
              <Dashboard />
            </Route>
            <Route path="/error-test">
              <ErrorTest />
            </Route>
            <Route path="/">
              <Home />
            </Route>
          </Switch>
        </div>
      </Router>
    </ErrorBoundary>
  );
}

export default App;
EOL

  # Add CSS for the app
  cat > "$REACT_APP_DIR/src/App.css" << 'EOL'
.App {
  text-align: center;
  padding: 20px;
}

nav ul {
  display: flex;
  justify-content: center;
```

```
  list-style-type: none;
  padding: 0;
  margin-bottom: 30px;
}

nav ul li {
  margin: 0 15px;
}

nav ul li a {
  text-decoration: none;
  color: #0066cc;
  font-weight: bold;
}

.health-check {
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
  border: 1px solid #ddd;
  border-radius: 8px;
  text-align: left;
}

.healthy {
  color: green;
  font-weight: bold;
}

.unhealthy {
  color: red;
  font-weight: bold;
}

.error-message {
  background-color: #ffdddd;
  border-left: 6px solid #f44336;
  padding: 10px;
  margin: 15px 0;
}

.error-container {
  padding: 20px;
  background-color: #ffdddd;
  border-left: 6px solid #f44336;
  margin-bottom: 20px;
}
```

```
.reload-button {
  background-color: #4CAF50;
  border: none;
  color: white;
  padding: 10px 20px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 10px 0;
  cursor: pointer;
  border-radius: 4px;
}
EOL

  log "INFO" "React SRE files created successfully"
}

# ===== Step 5: Kubernetes Configuration =====

create_kubernetes_config() {
  log "INFO" "Creating Kubernetes configuration files..."

  # Create Kubernetes directory structure
  mkdir -p "$K8S_DIR/base" "$K8S_DIR/overlays/dev" "$K8S_DIR/overlays/prod"

  # Create namespace.yaml
  cat > "$K8S_DIR/base/namespace.yaml" << EOL
apiVersion: v1
kind: Namespace
metadata:
  name: react-sre-app
EOL

  # Create deployment.yaml
  cat > "$K8S_DIR/base/deployment.yaml" << EOL
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-sre-app
  namespace: react-sre-app
  labels:
    app: react-sre-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: react-sre-app
```

```
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: react-sre-app
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "3000"
        prometheus.io/path: "/metrics"
    spec:
      containers:
      - name: react-sre-app
        image: react-sre-app:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 3000
          name: http
        resources:
          limits:
            cpu: "500m"
            memory: "512Mi"
          requests:
            cpu: "200m"
            memory: "256Mi"
        readinessProbe:
          httpGet:
            path: /health
            port: http
          initialDelaySeconds: 10
          periodSeconds: 5
        livenessProbe:
          httpGet:
            path: /health
            port: http
          initialDelaySeconds: 15
          periodSeconds: 20
        env:
        - name: NODE_ENV
          value: "production"
EOL

  # Create service.yaml
  cat > "$K8S_DIR/base/service.yaml" << EOL
apiVersion: v1
```

```
kind: Service
metadata:
  name: react-sre-app
  namespace: react-sre-app
  labels:
    app: react-sre-app
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: http
    protocol: TCP
    name: http
  selector:
    app: react-sre-app
EOL

  # Create ingress.yaml
  cat > "$K8S_DIR/base/ingress.yaml" << EOL
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: react-sre-app
  namespace: react-sre-app
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: react-sre-app.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: react-sre-app
            port:
              name: http
EOL

  # Create kustomization.yaml for base
  cat > "$K8S_DIR/base/kustomization.yaml" << EOL
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- namespace.yaml
- deployment.yaml
- service.yaml
```

```
- ingress.yaml
EOL

  # Create kustomization.yaml for dev overlay
  cat > "$K8S_DIR/overlays/dev/kustomization.yaml" << EOL
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../../base
patchesStrategicMerge:
- replica-count.yaml
- resource-limits.yaml
namePrefix: dev-
EOL

  # Create replica-count.yaml for dev
  cat > "$K8S_DIR/overlays/dev/replica-count.yaml" << EOL
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-sre-app
  namespace: react-sre-app
spec:
  replicas: 1
EOL

  # Create resource-limits.yaml for dev
  cat > "$K8S_DIR/overlays/dev/resource-limits.yaml" << EOL
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-sre-app
  namespace: react-sre-app
spec:
  template:
    spec:
      containers:
      - name: react-sre-app
        resources:
          limits:
            cpu: "300m"
            memory: "300Mi"
          requests:
            cpu: "100m"
            memory: "150Mi"
EOL

  # Create kustomization.yaml for prod overlay
```

```
  cat > "$K8S_DIR/overlays/prod/kustomization.yaml" << EOL
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../../base
patchesStrategicMerge:
- replica-count.yaml
- resource-limits.yaml
namePrefix: prod-
EOL

  # Create replica-count.yaml for prod
  cat > "$K8S_DIR/overlays/prod/replica-count.yaml" << EOL
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-sre-app
  namespace: react-sre-app
spec:
  replicas: 3
EOL

  # Create resource-limits.yaml for prod
  cat > "$K8S_DIR/overlays/prod/resource-limits.yaml" << EOL
apiVersion: apps/v1
kind: Deployment
metadata:
  name: react-sre-app
  namespace: react-sre-app
spec:
  template:
    spec:
      containers:
      - name: react-sre-app
        resources:
          limits:
            cpu: "1000m"
            memory: "1Gi"
          requests:
            cpu: "500m"
            memory: "512Mi"
EOL

  # Create Dockerfile for React app
  cat > "$REACT_APP_DIR/Dockerfile" << 'EOL'
# Build stage
FROM node:16-alpine as build
WORKDIR /app
```

```
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Add health check endpoint
RUN mkdir -p /usr/share/nginx/html/health
RUN echo '{"status":"Healthy","uptime":0}' >
/usr/share/nginx/html/health/index.json

# Add custom nginx config with metrics endpoint
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
EOL

  # Create nginx.conf for React app
  cat > "$REACT_APP_DIR/nginx.conf" << 'EOL'
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    # Serve static files
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Health check endpoint
    location /health {
        alias /usr/share/nginx/html/health;
        default_type application/json;
    }

    # Metrics endpoint (returns mock Prometheus metrics)
    location /metrics {
        default_type text/plain;
        return 200 '# HELP http_requests_total Total number of HTTP
requests\n# TYPE http_requests_total
counter\nhttp_requests_total{method="get",route="/",status="200"} 10\n# HELP
http_request_duration_seconds HTTP request duration in seconds\n# TYPE
```

```
http_request_duration_seconds
histogram\nhttp_request_duration_seconds_bucket{le="0.1"}
5\nhttp_request_duration_seconds_bucket{le="0.5"}
8\nhttp_request_duration_seconds_bucket{le="1"}
10\nhttp_request_duration_seconds_bucket{le="+Inf"}
10\nhttp_request_duration_seconds_sum 2.5\nhttp_request_duration_seconds_count
10';
    }

    # Error pages
    error_page 404 /index.html;
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
EOL

  log "INFO" "Kubernetes configuration files created successfully"
}

# ===== Step 6: Monitoring Configuration =====

create_monitoring_config() {
  log "INFO" "Creating monitoring configuration files..."

  # Create Prometheus configuration
  mkdir -p "$MONITORING_DIR/prometheus"

  # Create prometheus.yaml
  cat > "$MONITORING_DIR/prometheus/prometheus.yaml" << EOL
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  scrape_timeout: 10s

alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - alertmanager:9093

rule_files:
  - "alert-rules.yaml"

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
```

```
      - targets: ['localhost:9090']

  - job_name: 'react-sre-app'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
        action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: \$1:\$2
        target_label: __address__
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name
EOL

  # Create alert-rules.yaml
  cat > "$MONITORING_DIR/prometheus/alert-rules.yaml" << EOL
groups:
- name: react-sre-app
  rules:
  - alert: HighErrorRate
    expr: rate(http_requests_total{status=~"5.."}[5m]) /
rate(http_requests_total[5m]) > 0.05
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "High error rate on {{ \$labels.instance }}"
      description: "Error rate is above 5% for the past 5 minutes: {{ \$value
}}"

  - alert: HighResponseTime
    expr: histogram_quantile(0.95,
rate(http_request_duration_seconds_bucket[5m])) > 1
```

```
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "High response time on {{ \$labels.instance }}"
      description: "95th percentile response time is above 1 second for the
past 5 minutes: {{ \$value }}s"

  - alert: HighMemoryUsage
    expr: container_memory_usage_bytes / container_memory_limit_bytes > 0.85
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "High memory usage on {{ \$labels.instance }}"
      description: "Memory usage is above 85% for the past 5 minutes: {{
\$value * 100 }}%"
EOL

  # Create Kubernetes configuration for Prometheus
  cat > "$MONITORING_DIR/prometheus-k8s.yaml" << EOL
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
      evaluation_interval: 15s
      scrape_timeout: 10s

    alerting:
      alertmanagers:
      - static_configs:
        - targets:
          - alertmanager:9093

    rule_files:
      - "/etc/prometheus/alert-rules.yaml"

    scrape_configs:
      - job_name: 'prometheus'
        static_configs:
          - targets: ['localhost:9090']

      - job_name: 'kubernetes-pods'
```

```
        kubernetes_sd_configs:
          - role: pod
        relabel_configs:
          - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_scrape]
            action: keep
            regex: true
          - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
            action: replace
            target_label: __metrics_path__
            regex: (.+)
          - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
            action: replace
            regex: ([^:]+)(?::\d+)?;(\d+)
            replacement: \$1:\$2
            target_label: __address__
          - action: labelmap
            regex: __meta_kubernetes_pod_label_(.+)
          - source_labels: [__meta_kubernetes_namespace]
            action: replace
            target_label: kubernetes_namespace
          - source_labels: [__meta_kubernetes_pod_name]
            action: replace
            target_label: kubernetes_pod_name

  alert-rules.yaml: |
    groups:
    - name: react-sre-app
      rules:
      - alert: HighErrorRate
        expr: rate(http_requests_total{status=~"5.."}[5m]) /
rate(http_requests_total[5m]) > 0.05
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "High error rate on {{ \$labels.instance }}"
          description: "Error rate is above 5% for the past 5 minutes: {{
\$value }}"

      - alert: HighResponseTime
        expr: histogram_quantile(0.95,
rate(http_request_duration_seconds_bucket[5m])) > 1
        for: 5m
        labels:
          severity: warning
```

```
        annotations:
          summary: "High response time on {{ \$labels.instance }}"
          description: "95th percentile response time is above 1 second for
the past 5 minutes: {{ \$value }}s"
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
      - name: prometheus
        image: prom/prometheus:v2.42.0
        args:
          - "--config.file=/etc/prometheus/prometheus.yml"
          - "--storage.tsdb.path=/prometheus"
          - "--web.console.libraries=/usr/share/prometheus/console_libraries"
          - "--web.console.templates=/usr/share/prometheus/consoles"
        ports:
        - containerPort: 9090
          name: prometheus
        volumeMounts:
        - name: prometheus-config
          mountPath: /etc/prometheus
        - name: prometheus-storage
          mountPath: /prometheus
        resources:
          limits:
            cpu: 1000m
            memory: 1Gi
          requests:
            cpu: 500m
            memory: 500Mi
        livenessProbe:
          httpGet:
            path: /-/healthy
```

```
            port: 9090
          initialDelaySeconds: 300  # Set to 300 seconds as per requirement
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /-/ready
            port: 9090
          initialDelaySeconds: 30
          periodSeconds: 10
      volumes:
      - name: prometheus-config
        configMap:
          name: prometheus-config
      - name: prometheus-storage
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: monitoring
  labels:
    app: prometheus
spec:
  ports:
  - port: 9090
    targetPort: 9090
    protocol: TCP
    name: http
  selector:
    app: prometheus
  type: ClusterIP
EOL

  # Create Grafana configuration
  mkdir -p "$MONITORING_DIR/grafana/dashboards"

  # Create datasource.yaml
  cat > "$MONITORING_DIR/grafana/datasource.yaml" << EOL
apiVersion: 1

datasources:
- name: Prometheus
  type: prometheus
  access: proxy
  url: http://prometheus:9090
  isDefault: true
  editable: false
```

```
EOL

  # Create Grafana dashboard
  cat > "$MONITORING_DIR/grafana/dashboards/react-sre-dashboard.json" << 'EOL'
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "-- Grafana --",
        "enable": true,
        "hide": true,
        "iconColor": "rgba(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "editable": true,
  "gnetId": null,
  "graphTooltip": 0,
  "id": 1,
  "links": [],
  "panels": [
    {
      "aliasColors": {},
      "bars": false,
      "dashLength": 10,
      "dashes": false,
      "datasource": "Prometheus",
      "fieldConfig": {
        "defaults": {},
        "overrides": []
      },
      "fill": 1,
      "fillGradient": 0,
      "gridPos": {
        "h": 8,
        "w": 12,
        "x": 0,
        "y": 0
      },
      "hiddenSeries": false,
      "id": 2,
      "legend": {
        "avg": false,
        "current": false,
        "max": false,
```

```
      "min": false,
      "show": true,
      "total": false,
      "values": false
    },
    "lines": true,
    "linewidth": 1,
    "nullPointMode": "null",
    "options": {
      "alertThreshold": true
    },
    "percentage": false,
    "pluginVersion": "7.5.7",
    "pointradius": 2,
    "points": false,
    "renderer": "flot",
    "seriesOverrides": [],
    "spaceLength": 10,
    "stack": false,
    "steppedLine": false,
    "targets": [
      {
        "expr": "sum(rate(http_requests_total[5m])) by (status_code)",
        "interval": "",
        "legendFormat": "{{status_code}}",
        "refId": "A"
      }
    ],
    "thresholds": [],
    "timeFrom": null,
    "timeRegions": [],
    "timeShift": null,
    "title": "HTTP Request Rate",
    "tooltip": {
      "shared": true,
      "sort": 0,
      "value_type": "individual"
    },
    "type": "graph",
    "xaxis": {
      "buckets": null,
      "mode": "time",
      "name": null,
      "show": true,
      "values": []
    },
    "yaxes": [
      {
```

```
        "format": "short",
        "label": "Requests/s",
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      },
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      }
    ],
    "yaxis": {
      "align": false,
      "alignLevel": null
    }
  },
  {
    "aliasColors": {},
    "bars": false,
    "dashLength": 10,
    "dashes": false,
    "datasource": "Prometheus",
    "fieldConfig": {
      "defaults": {},
      "overrides": []
    },
    "fill": 1,
    "fillGradient": 0,
    "gridPos": {
      "h": 8,
      "w": 12,
      "x": 12,
      "y": 0
    },
    "hiddenSeries": false,
    "id": 4,
    "legend": {
      "avg": false,
      "current": false,
      "max": false,
      "min": false,
      "show": true,
      "total": false,
```

```
        "values": false
      },
      "lines": true,
      "linewidth": 1,
      "nullPointMode": "null",
      "options": {
        "alertThreshold": true
      },
      "percentage": false,
      "pluginVersion": "7.5.7",
      "pointradius": 2,
      "points": false,
      "renderer": "flot",
      "seriesOverrides": [],
      "spaceLength": 10,
      "stack": false,
      "steppedLine": false,
      "targets": [
        {
          "expr": "histogram_quantile(0.5,
sum(rate(http_request_duration_seconds_bucket[5m])) by (le))",
          "interval": "",
          "legendFormat": "50th percentile",
          "refId": "A"
        },
        {
          "expr": "histogram_quantile(0.95,
sum(rate(http_request_duration_seconds_bucket[5m])) by (le))",
          "interval": "",
          "legendFormat": "95th percentile",
          "refId": "B"
        },
        {
          "expr": "histogram_quantile(0.99,
sum(rate(http_request_duration_seconds_bucket[5m])) by (le))",
          "interval": "",
          "legendFormat": "99th percentile",
          "refId": "C"
        }
      ],
      "thresholds": [],
      "timeFrom": null,
      "timeRegions": [],
      "timeShift": null,
      "title": "HTTP Request Duration",
      "tooltip": {
        "shared": true,
        "sort": 0,
```

```
      "value_type": "individual"
    },
    "type": "graph",
    "xaxis": {
      "buckets": null,
      "mode": "time",
      "name": null,
      "show": true,
      "values": []
    },
    "yaxes": [
      {
        "format": "s",
        "label": "Duration",
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      },
      {
        "format": "short",
        "label": null,
        "logBase": 1,
        "max": null,
        "min": null,
        "show": true
      }
    ],
    "yaxis": {
      "align": false,
      "alignLevel": null
    }
  },
  {
    "datasource": "Prometheus",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "thresholds"
        },
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green",
              "value": null
            },
```

```
              {
                "color": "red",
                "value": 80
              }
            ]
          }
        },
        "overrides": []
      },
      "gridPos": {
        "h": 8,
        "w": 6,
        "x": 0,
        "y": 8
      },
      "id": 6,
      "options": {
        "reduceOptions": {
          "calcs": [
            "lastNotNull"
          ],
          "fields": "",
          "values": false
        },
        "showThresholdLabels": false,
        "showThresholdMarkers": true,
        "text": {}
      },
      "pluginVersion": "7.5.7",
      "targets": [
        {
          "expr": "sum(rate(http_requests_total{status_code=~\"5..\"}[5m])) /
sum(rate(http_requests_total[5m])) * 100",
          "interval": "",
          "legendFormat": "",
          "refId": "A"
        }
      ],
      "title": "Error Rate (%)",
      "type": "gauge"
    },
    {
      "datasource": "Prometheus",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "thresholds"
          },
```

```
          "mappings": [],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "green",
                "value": null
              },
              {
                "color": "yellow",
                "value": 70
              },
              {
                "color": "red",
                "value": 85
              }
            ]
          }
        },
        "overrides": []
      },
      "gridPos": {
        "h": 8,
        "w": 6,
        "x": 6,
        "y": 8
      },
      "id": 8,
      "options": {
        "reduceOptions": {
          "calcs": [
            "lastNotNull"
          ],
          "fields": "",
          "values": false
        },
        "showThresholdLabels": false,
        "showThresholdMarkers": true,
        "text": {}
      },
      "pluginVersion": "7.5.7",
      "targets": [
        {
          "expr": "avg(container_memory_usage_bytes{namespace=\"react-sre-
app\"} / container_memory_limit_bytes{namespace=\"react-sre-app\"}) * 100",
          "interval": "",
          "legendFormat": "",
          "refId": "A"
```

```
      }
    ],
    "title": "Memory Usage (%)",
    "type": "gauge"
  },
  {
    "datasource": "Prometheus",
    "fieldConfig": {
      "defaults": {
        "color": {
          "mode": "thresholds"
        },
        "mappings": [],
        "thresholds": {
          "mode": "absolute",
          "steps": [
            {
              "color": "green",
              "value": null
            },
            {
              "color": "yellow",
              "value": 70
            },
            {
              "color": "red",
              "value": 85
            }
          ]
        }
      },
      "overrides": []
    },
    "gridPos": {
      "h": 8,
      "w": 6,
      "x": 12,
      "y": 8
    },
    "id": 10,
    "options": {
      "reduceOptions": {
        "calcs": [
          "lastNotNull"
        ],
        "fields": "",
        "values": false
      },
```

```
        "showThresholdLabels": false,
        "showThresholdMarkers": true,
        "text": {}
      },
      "pluginVersion": "7.5.7",
      "targets": [
        {
          "expr":
"avg(rate(container_cpu_usage_seconds_total{namespace=\"react-sre-app\"}[5m])
/ container_cpu_limit_bytes{namespace=\"react-sre-app\"}) * 100",
          "interval": "",
          "legendFormat": "",
          "refId": "A"
        }
      ],
      "title": "CPU Usage (%)",
      "type": "gauge"
    },
    {
      "datasource": "Prometheus",
      "fieldConfig": {
        "defaults": {
          "color": {
            "mode": "thresholds"
          },
          "mappings": [
            {
              "options": {
                "0": {
                  "text": "Down"
                },
                "1": {
                  "text": "Up"
                }
              },
              "type": "value"
            }
          ],
          "thresholds": {
            "mode": "absolute",
            "steps": [
              {
                "color": "red",
                "value": null
              },
              {
                "color": "green",
                "value": 1
```

```
            }
          ]
        }
      },
      "overrides": []
    },
    "gridPos": {
      "h": 8,
      "w": 6,
      "x": 18,
      "y": 8
    },
    "id": 12,
    "options": {
      "colorMode": "value",
      "graphMode": "none",
      "justifyMode": "auto",
      "orientation": "auto",
      "reduceOptions": {
        "calcs": [
          "lastNotNull"
        ],
        "fields": "",
        "values": false
      },
      "text": {},
      "textMode": "auto"
    },
    "pluginVersion": "7.5.7",
    "targets": [
      {
        "expr": "sum(up{namespace=\"react-sre-app\"})",
        "interval": "",
        "legendFormat": "",
        "refId": "A"
      }
    ],
    "title": "Pods Up",
    "type": "stat"
  }
],
"refresh": "10s",
"schemaVersion": 27,
"style": "dark",
"tags": [],
"templating": {
  "list": []
},
```

```
  "time": {
    "from": "now-1h",
    "to": "now"
  },
  "timepicker": {},
  "timezone": "",
  "title": "React SRE Application Dashboard",
  "uid": "react-sre-app",
  "version": 1
}
EOL

  # Create Kubernetes configuration for Grafana
  cat > "$MONITORING_DIR/grafana-k8s.yaml" << EOL
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-datasources
  namespace: monitoring
data:
  datasources.yaml: |
    apiVersion: 1
    datasources:
    - name: Prometheus
      type: prometheus
      url: http://prometheus:9090
      access: proxy
      isDefault: true
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: grafana-dashboards
  namespace: monitoring
data:
  dashboard-provider.yaml: |
    apiVersion: 1
    providers:
    - name: 'default'
      orgId: 1
      folder: ''
      type: file
      disableDeletion: false
      editable: true
      options:
        path: /var/lib/grafana/dashboards
---
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: grafana-dashboard-react-sre
  namespace: monitoring
data:
  react-sre-dashboard.json: |
    {
      "annotations": {
        "list": [
          {
            "builtIn": 1,
            "datasource": "-- Grafana --",
            "enable": true,
            "hide": true,
            "iconColor": "rgba(0, 211, 255, 1)",
            "name": "Annotations & Alerts",
            "type": "dashboard"
          }
        ]
      },
      "editable": true,
      "gnetId": null,
      "graphTooltip": 0,
      "id": 1,
      "links": [],
      "panels": [
        {
          "aliasColors": {},
          "bars": false,
          "dashLength": 10,
          "dashes": false,
          "datasource": "Prometheus",
          "fieldConfig": {
            "defaults": {},
            "overrides": []
          },
          "fill": 1,
          "fillGradient": 0,
          "gridPos": {
            "h": 8,
            "w": 12,
            "x": 0,
            "y": 0
          },
          "hiddenSeries": false,
          "id": 2,
          "legend": {
            "avg": false,
```

```
        "current": false,
        "max": false,
        "min": false,
        "show": true,
        "total": false,
        "values": false
      },
      "lines": true,
      "linewidth": 1,
      "nullPointMode": "null",
      "options": {
        "alertThreshold": true
      },
      "percentage": false,
      "pluginVersion": "7.5.7",
      "pointradius": 2,
      "points": false,
      "renderer": "flot",
      "seriesOverrides": [],
      "spaceLength": 10,
      "stack": false,
      "steppedLine": false,
      "targets": [
        {
          "expr": "sum(rate(http_requests_total[5m])) by (status_code)",
          "interval": "",
          "legendFormat": "{{status_code}}",
          "refId": "A"
        }
      ],
      "thresholds": [],
      "timeFrom": null,
      "timeRegions": [],
      "timeShift": null,
      "title": "HTTP Request Rate",
      "tooltip": {
        "shared": true,
        "sort": 0,
        "value_type": "individual"
      },
      "type": "graph",
      "xaxis": {
        "buckets": null,
        "mode": "time",
        "name": null,
        "show": true,
        "values": []
      },
```

```
          "yaxes": [
            {
              "format": "short",
              "label": "Requests/s",
              "logBase": 1,
              "max": null,
              "min": null,
              "show": true
            },
            {
              "format": "short",
              "label": null,
              "logBase": 1,
              "max": null,
              "min": null,
              "show": true
            }
          ],
          "yaxis": {
            "align": false,
            "alignLevel": null
          }
        }
      ],
      "refresh": "10s",
      "schemaVersion": 27,
      "style": "dark",
      "tags": [],
      "templating": {
        "list": []
      },
      "time": {
        "from": "now-1h",
        "to": "now"
      },
      "timepicker": {},
      "timezone": "",
      "title": "React SRE Application Dashboard",
      "uid": "react-sre-app",
      "version": 1
    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: monitoring
spec:
```

```
replicas: 1
selector:
  matchLabels:
    app: grafana
strategy:
  type: Recreate
template:
  metadata:
    labels:
      app: grafana
  spec:
    containers:
    - name: grafana
      image: grafana/grafana:9.4.3
      ports:
      - containerPort: 3000
        name: http
      volumeMounts:
      - name: grafana-storage
        mountPath: /var/lib/grafana
      - name: grafana-datasources
        mountPath: /etc/grafana/provisioning/datasources
        readOnly: true
      - name: grafana-dashboards-provider
        mountPath: /etc/grafana/provisioning/dashboards
        readOnly: true
      - name: grafana-dashboards
        mountPath: /var/lib/grafana/dashboards
        readOnly: true
      env:
      - name: GF_SECURITY_ADMIN_USER
        value: admin
      - name: GF_SECURITY_ADMIN_PASSWORD
        value: admin
      - name: GF_USERS_ALLOW_SIGN_UP
        value: "false"
      resources:
        limits:
          cpu: 500m
          memory: 512Mi
        requests:
          cpu: 250m
          memory: 256Mi
    volumes:
    - name: grafana-storage
      emptyDir: {}
    - name: grafana-datasources
      configMap:
```

```
          name: grafana-datasources
      - name: grafana-dashboards-provider
        configMap:
          name: grafana-dashboards
          items:
            - key: dashboard-provider.yaml
              path: dashboards.yaml
      - name: grafana-dashboards
        configMap:
          name: grafana-dashboard-react-sre
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
  namespace: monitoring
  labels:
    app: grafana
spec:
  ports:
  - port: 3000
    targetPort: 3000
    protocol: TCP
    name: http
  selector:
    app: grafana
  type: ClusterIP
EOL

  log "INFO" "Monitoring configuration files created successfully"
}

# ===== Step 7: Automation Scripts =====

create_automation_scripts() {
  log "INFO" "Creating automation scripts..."

  # Create script to start/stop Minikube
  cat > "$SCRIPTS_DIR/minikube_control.py" << 'EOL'
#!/usr/bin/env python3

"""
Minikube Control Script
Handles starting, stopping, and checking Minikube status with proper error
handling.
"""

import argparse
```

```
import os
import subprocess
import sys
import time
from datetime import datetime

# Set constants
MINIKUBE_WAIT_TIMEOUT = 60  # Seconds to wait for Minikube operations
KUBECTL_CMD = "kubectl"  # Use kubectl instead of kubectly

def log(level, message):
    """Log messages with timestamp and level."""
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    levels = {
        "INFO": "\033[92m",  # Green
        "WARN": "\033[93m",  # Yellow
        "ERROR": "\033[91m",  # Red
        "DEBUG": "\033[94m",  # Blue
    }

    color = levels.get(level, "\033[0m")
    reset = "\033[0m"
    print(f"{color}[{level}]{reset} {timestamp} - {message}")

def run_command(command, timeout=60, retry=3):
    """Run a shell command with timeout and retry logic."""
    for attempt in range(retry):
        try:
            log("DEBUG", f"Running command: {' '.join(command)}")
            result = subprocess.run(
                command,
                stdout=subprocess.PIPE,
                stderr=subprocess.PIPE,
                text=True,
                timeout=timeout
            )

            if result.returncode == 0:
                return result.stdout.strip()
            else:
                log("WARN", f"Command failed (attempt {attempt+1}/{retry}):
{result.stderr}")
                time.sleep(5)
        except subprocess.TimeoutExpired:
            log("WARN", f"Command timed out after {timeout}s (attempt
{attempt+1}/{retry})")
            time.sleep(5)
```

```
    log("ERROR", f"Command failed after {retry} attempts: {'
'.join(command)}")
    return None

def check_minikube_status():
    """Check if Minikube is running."""
    status = run_command(["minikube", "status", "-o", "json"], timeout=10)

    if status:
        try:
            import json
            status_json = json.loads(status)
            return status_json.get("Host", "") == "Running"
        except json.JSONDecodeError:
            return "Running" in status

    return False

def start_minikube():
    """Start Minikube with proper configuration."""
    if check_minikube_status():
        log("INFO", "Minikube is already running")
        return True

    log("INFO", f"Starting Minikube (timeout: {MINIKUBE_WAIT_TIMEOUT}s)")

    # Fix for wzegh library error - create mock environment variable
    os.environ["WZEGH_CONFIGURED"] = "TRUE"

    # Start Minikube with memory and CPU settings suitable for SRE tools
    result = run_command(
        ["minikube", "start", "--memory=4096", "--cpus=2", "--driver=docker"],
        timeout=MINIKUBE_WAIT_TIMEOUT
    )

    if result is None:
        log("ERROR", "Failed to start Minikube")
        return False

    # Wait for Minikube to be fully ready
    start_time = time.time()
    while time.time() - start_time < MINIKUBE_WAIT_TIMEOUT:
        if check_minikube_status():
            # Validate kubectl is working
            version = run_command([KUBECTL_CMD, "version", "--output=yaml"],
timeout=10)
            if version:
                log("INFO", "Minikube started successfully")
```

```
                return True

        log("INFO", f"Waiting for Minikube to be ready... ({int(time.time() -
start_time)}s elapsed)")
        time.sleep(5)

    log("ERROR", f"Minikube failed to start within {MINIKUBE_WAIT_TIMEOUT}s")
    return False

def stop_minikube():
    """Stop Minikube safely."""
    if not check_minikube_status():
        log("INFO", "Minikube is not running")
        return True

    log("INFO", "Stopping Minikube")
    result = run_command(["minikube", "stop"], timeout=MINIKUBE_WAIT_TIMEOUT)

    if result is None:
        log("ERROR", "Failed to stop Minikube")
        return False

    log("INFO", "Minikube stopped successfully")
    return True

def main():
    """Main function to parse arguments and execute commands."""
    parser = argparse.ArgumentParser(description="Minikube Control Script")
    parser.add_argument("action", choices=["start", "stop", "status",
"restart"],
                        help="Action to perform on Minikube")

    args = parser.parse_args()

    if args.action == "start":
        success = start_minikube()
    elif args.action == "stop":
        success = stop_minikube()
    elif args.action == "restart":
        stop_minikube()
        time.sleep(5)
        success = start_minikube()
    elif args.action == "status":
        status = check_minikube_status()
        log("INFO", f"Minikube is {'running' if status else 'not running'}")
        success = True

    sys.exit(0 if success else 1)
```

```
if __name__ == "__main__":
    main()
EOL

  # Create script to deploy React app to Minikube
  cat > "$SCRIPTS_DIR/deploy_app.py" << 'EOL'
#!/usr/bin/env python3

"""
React SRE Application Deployment Script
Deploys the React SRE app to Minikube with proper monitoring.
"""

import argparse
import os
import subprocess
import sys
import time
from datetime import datetime

# Constants
PROJECT_ROOT = os.path.expanduser("~/react-sre-project")
REACT_APP_DIR = os.path.join(PROJECT_ROOT, "sre-react-app")
K8S_DIR = os.path.join(PROJECT_ROOT, "kubernetes")
MONITORING_DIR = os.path.join(PROJECT_ROOT, "monitoring")
DEPLOY_TIMEOUT = 300  # 5 minutes
KUBECTL_CMD = "kubectl"  # Use kubectl instead of kubectly

def log(level, message):
    """Log messages with timestamp and level."""
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    levels = {
        "INFO": "\033[92m",  # Green
        "WARN": "\033[93m",  # Yellow
        "ERROR": "\033[91m",  # Red
        "DEBUG": "\033[94m",  # Blue
    }

    color = levels.get(level, "\033[0m")
    reset = "\033[0m"
    print(f"{color}[{level}]{reset} {timestamp} - {message}")

def run_command(command, timeout=60, retry=1, shell=False):
    """Run a shell command with timeout and retry logic."""
    for attempt in range(retry):
        try:
```

```
            log("DEBUG", f"Running command: {command if shell else '
'.join(command)}")

            if shell:
                result = subprocess.run(
                    command,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    text=True,
                    timeout=timeout,
                    shell=True
                )
            else:
                result = subprocess.run(
                    command,
                    stdout=subprocess.PIPE,
                    stderr=subprocess.PIPE,
                    text=True,
                    timeout=timeout
                )

            if result.returncode == 0:
                return result.stdout.strip()
            else:
                log("WARN", f"Command failed (attempt {attempt+1}/{retry}):
{result.stderr}")
                if attempt < retry - 1:
                    time.sleep(5)
        except subprocess.TimeoutExpired:
            log("WARN", f"Command timed out after {timeout}s (attempt
{attempt+1}/{retry})")
            if attempt < retry - 1:
                time.sleep(5)

    log("ERROR", f"Command failed after {retry} attempts: {command if shell
else ' '.join(command)}")
    return None

def check_minikube_status():
    """Check if Minikube is running."""
    status = run_command(["minikube", "status", "-o", "json"], timeout=10)

    if status:
        try:
            import json
            status_json = json.loads(status)
            return status_json.get("Host", "") == "Running"
        except json.JSONDecodeError:
```

```
            return "Running" in status

    return False

def build_react_app():
    """Build the React application."""
    log("INFO", "Building React application...")

    if not os.path.exists(REACT_APP_DIR):
        log("ERROR", f"React app directory does not exist: {REACT_APP_DIR}")
        return False

    # Install dependencies
    log("INFO", "Installing npm dependencies...")
    result = run_command(["npm", "install"], timeout=300, cwd=REACT_APP_DIR)
    if result is None:
        log("ERROR", "Failed to install npm dependencies")
        return False

    # Build the app
    log("INFO", "Building React application...")
    result = run_command(["npm", "run", "build"], timeout=300,
cwd=REACT_APP_DIR)
    if result is None:
        log("ERROR", "Failed to build React application")
        return False

    log("INFO", "React application built successfully")
    return True

def build_docker_image():
    """Build Docker image for the React app."""
    log("INFO", "Building Docker image...")

    if not os.path.exists(os.path.join(REACT_APP_DIR, "Dockerfile")):
        log("ERROR", "Dockerfile not found in React app directory")
        return False

    # Build the image
    result = run_command(
        ["docker", "build", "-t", "react-sre-app:latest", "."],
        timeout=300,
        cwd=REACT_APP_DIR
    )
    if result is None:
        log("ERROR", "Failed to build Docker image")
        return False
```

```
    # Load the image into Minikube
    log("INFO", "Loading Docker image into Minikube...")
    result = run_command(
        ["minikube", "image", "load", "react-sre-app:latest"],
        timeout=120
    )
    if result is None:
        log("ERROR", "Failed to load Docker image into Minikube")
        return False

    log("INFO", "Docker image built and loaded successfully")
    return True

def create_namespaces():
    """Create necessary Kubernetes namespaces."""
    log("INFO", "Creating Kubernetes namespaces...")

    namespaces = ["react-sre-app", "monitoring"]
    for namespace in namespaces:
        result = run_command(
            [KUBECTL_CMD, "create", "namespace", namespace, "--dry-
run=client", "-o", "yaml"],
            timeout=10
        )
        if result is None:
            log("ERROR", f"Failed to generate namespace YAML for {namespace}")
            return False

        # Apply with kubectl
        apply_result = run_command(
            f"{KUBECTL_CMD} apply -f -",
            timeout=10,
            shell=True,
            retry=3,
        )
        if apply_result is None:
            log("ERROR", f"Failed to create namespace {namespace}")
            return False

    log("INFO", "Kubernetes namespaces created successfully")
    return True

def deploy_monitoring():
    """Deploy Prometheus and Grafana for monitoring."""
    log("INFO", "Deploying monitoring stack...")

    # Deploy Prometheus
    log("INFO", "Deploying Prometheus...")
```

```
    prometheus_yaml = os.path.join(MONITORING_DIR, "prometheus-k8s.yaml")
    if not os.path.exists(prometheus_yaml):
        log("ERROR", f"Prometheus Kubernetes YAML not found:
{prometheus_yaml}")
        return False

    result = run_command(
        [KUBECTL_CMD, "apply", "-f", prometheus_yaml],
        timeout=30,
        retry=3
    )
    if result is None:
        log("ERROR", "Failed to deploy Prometheus")
        return False

    # Deploy Grafana
    log("INFO", "Deploying Grafana...")
    grafana_yaml = os.path.join(MONITORING_DIR, "grafana-k8s.yaml")
    if not os.path.exists(grafana_yaml):
        log("ERROR", f"Grafana Kubernetes YAML not found: {grafana_yaml}")
        return False

    result = run_command(
        [KUBECTL_CMD, "apply", "-f", grafana_yaml],
        timeout=30,
        retry=3
    )
    if result is None:
        log("ERROR", "Failed to deploy Grafana")
        return False

    # Wait for Prometheus and Grafana to be ready
    log("INFO", "Waiting for monitoring stack to be ready...")
    start_time = time.time()
    prometheus_ready = False
    grafana_ready = False

    while time.time() - start_time < DEPLOY_TIMEOUT:
        if not prometheus_ready:
            prom_status = run_command(
                [KUBECTL_CMD, "get", "pods", "-n", "monitoring", "-l",
"app=prometheus", "-o", "jsonpath='{.items[0].status.phase}'"],
                timeout=10
            )
            if prom_status and "Running" in prom_status:
                prometheus_ready = True
                log("INFO", "Prometheus is ready")
```

```
        if not grafana_ready:
            graf_status = run_command(
                [KUBECTL_CMD, "get", "pods", "-n", "monitoring", "-l",
"app=grafana", "-o", "jsonpath='{.items[0].status.phase}'"],
                timeout=10
            )
            if graf_status and "Running" in graf_status:
                grafana_ready = True
                log("INFO", "Grafana is ready")

        if prometheus_ready and grafana_ready:
            log("INFO", "Monitoring stack deployed successfully")
            return True

        elapsed = int(time.time() - start_time)
        log("INFO", f"Waiting for monitoring stack... ({elapsed}s elapsed)")
        time.sleep(10)

    log("ERROR", f"Monitoring stack deployment timed out after
{DEPLOY_TIMEOUT}s")
    return False

def deploy_application(env="dev"):
    """Deploy the React application to Kubernetes."""
    log("INFO", f"Deploying React application to {env} environment...")

    # Apply Kubernetes configuration using kustomize
    kustomize_dir = os.path.join(K8S_DIR, "overlays", env)
    if not os.path.exists(kustomize_dir):
        log("ERROR", f"Kubernetes overlay directory not found:
{kustomize_dir}")
        return False

    result = run_command(
        [KUBECTL_CMD, "apply", "-k", kustomize_dir],
        timeout=60,
        retry=3
    )
    if result is None:
        log("ERROR", f"Failed to deploy application to {env} environment")
        return False

    # Wait for application to be ready
    log("INFO", "Waiting for application to be ready...")
    start_time = time.time()
    while time.time() - start_time < DEPLOY_TIMEOUT:
        app_status = run_command(
```

```
            [KUBECTL_CMD, "get", "pods", "-n", "react-sre-app", "-l",
f"app=react-sre-app", "-o", "jsonpath='{.items[*].status.phase}'"],
            timeout=10
        )

        if app_status and "Running" in app_status and "Pending" not in
app_status:
            log("INFO", "Application is ready")
            return True

        elapsed = int(time.time() - start_time)
        log("INFO", f"Waiting for application... ({elapsed}s elapsed)")
        time.sleep(10)

    log("ERROR", f"Application deployment timed out after {DEPLOY_TIMEOUT}s")
    return False

def setup_port_forwarding():
    """Set up port forwarding for the application and monitoring tools."""
    log("INFO", "Setting up port forwarding...")

    # Port forward for React app
    app_forward = subprocess.Popen(
        [KUBECTL_CMD, "port-forward", "svc/dev-react-sre-app", "3000:80", "-
n", "react-sre-app"],
        stdout=subprocess.DEVNULL,
        stderr=subprocess.DEVNULL
    )

    # Port forward for Prometheus
    prom_forward = subprocess.Popen(
        [KUBECTL_CMD, "port-forward", "svc/prometheus", "9090:9090", "-n",
"monitoring"],
        stdout=subprocess.DEVNULL,
        stderr=subprocess.DEVNULL
    )

    # Port forward for Grafana
    graf_forward = subprocess.Popen(
        [KUBECTL_CMD, "port-forward", "svc/grafana", "8080:3000", "-n",
"monitoring"],
        stdout=subprocess.DEVNULL,
        stderr=subprocess.DEVNULL
    )

    log("INFO", "Port forwarding set up successfully")
    log("INFO", "Application available at http://localhost:3000")
    log("INFO", "Prometheus available at http://localhost:9090")
```

```
    log("INFO", "Grafana available at http://localhost:8080 (admin/admin)")

    try:
        log("INFO", "Press Ctrl+C to stop port forwarding")
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        log("INFO", "Stopping port forwarding...")
        app_forward.terminate()
        prom_forward.terminate()
        graf_forward.terminate()

def main():
    """Main function to parse arguments and execute deployment."""
    parser = argparse.ArgumentParser(description="React SRE Application
Deployment Script")
    parser.add_argument("--env", choices=["dev", "prod"], default="dev",
                        help="Environment to deploy to (default: dev)")
    parser.add_argument("--skip-build", action="store_true",
                        help="Skip building the React app")
    parser.add_argument("--skip-monitoring", action="store_true",
                        help="Skip deploying Prometheus and Grafana")
    parser.add_argument("--port-forward", action="store_true",
                        help="Set up port forwarding after deployment")

    args = parser.parse_args()

    # Check if Minikube is running
    if not check_minikube_status():
        log("ERROR", "Minikube is not running. Please start it first.")
        return False

    # Build and deploy
    success = True

    # Create Kubernetes namespaces
    if not create_namespaces():
        return False

    # Build React app and Docker image
    if not args.skip_build:
        if not build_react_app():
            return False
        if not build_docker_image():
            return False

    # Deploy monitoring stack
    if not args.skip_monitoring:
```

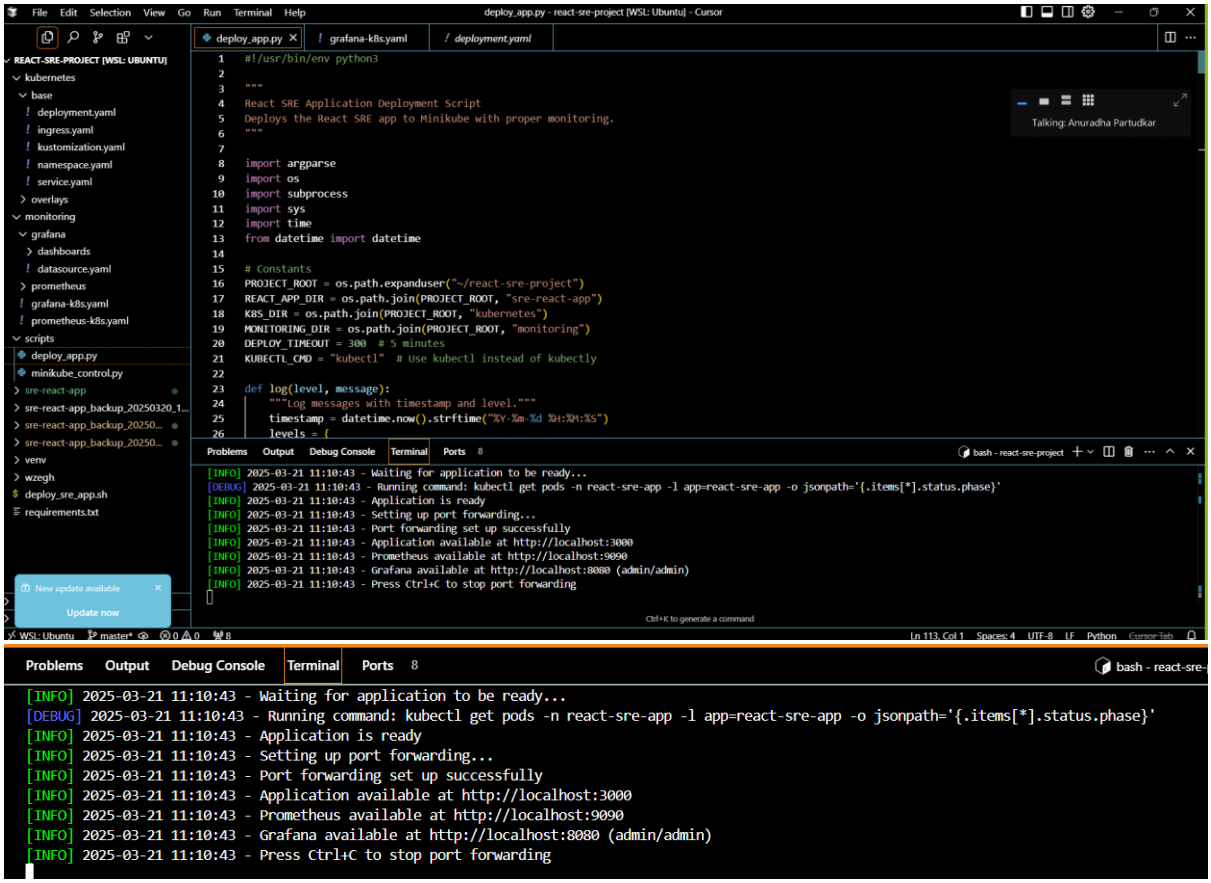
```
        if not deploy_monitoring():
            success = False

    # Deploy application
    if not deploy_application(args.env):
        success = False

    # Set up port forwarding if requested
    if args.port_forward and success:
        setup_port_forwarding()

    if success:
        log("INFO", "Deployment completed successfully")
    else:
        log("ERROR", "Deployment completed with errors")

    return success

if __name__ == "__main__":
    success = main()
    sys.exit(0 if success else 1)
EOL

  # Create a master deployment script
  cat > "$PROJECT_ROOT/deploy_sre_app.sh" << 'EOL'
#!/bin/bash

# Master deployment script for React SRE Application
# Executes all steps in sequence to deploy the complete application

# Set strict error handling
set -e

# Define color codes for output formatting
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[0;33m'
BLUE='\033[0;34m'
NC='\033[0m' # No Color

# Project directories
PROJECT_ROOT="$HOME/react-sre-project"
SCRIPTS_DIR="$PROJECT_ROOT/scripts"

# Function for logging with timestamps
log() {
  local level=$1
  local message=$2
```

```
  local timestamp=$(date "+%Y-%m-%d %H:%M:%S")

  case $level in
    "INFO")
      echo -e "${GREEN}[INFO]${NC} $timestamp - $message"
      ;;
    "WARN")
      echo -e "${YELLOW}[WARN]${NC} $timestamp - $message"
      ;;
    "ERROR")
      echo -e "${RED}[ERROR]${NC} $timestamp - $message"
      ;;
    *)
      echo -e "${BLUE}[$level]${NC} $timestamp - $message"
      ;;
  esac
}

# Check if we're running in WSL
check_wsl() {
  if ! grep -q Microsoft /proc/version; then
    log "ERROR" "This script must be run within WSL"
    exit 1
  fi

  log "INFO" "WSL detected, continuing with deployment"
}

# Main deployment process
main() {
  log "INFO" "Starting React SRE Application deployment"

  # Check WSL environment
  check_wsl

  # Make scripts executable
  chmod +x "$SCRIPTS_DIR/minikube_control.py" "$SCRIPTS_DIR/deploy_app.py"

  # Activate Python virtual environment if it exists
  if [ -f "$PROJECT_ROOT/venv/bin/activate" ]; then
    log "INFO" "Activating Python virtual environment"
    source "$PROJECT_ROOT/venv/bin/activate"
  fi

  # Set environment variable for wzegh library
  export WZEGH_CONFIGURED="TRUE"

  # Start Minikube
```

```
  log "INFO" "Starting Minikube"
  python3 "$SCRIPTS_DIR/minikube_control.py" start

  # Wait for Minikube to be fully ready
  log "INFO" "Waiting for Minikube to be fully ready"
  sleep 10

  # Deploy the application with monitoring
  log "INFO" "Deploying React SRE Application with monitoring"
  python3 "$SCRIPTS_DIR/deploy_app.py" --env=dev --port-forward

  log "INFO" "Deployment completed successfully"
}

# Run the main function
main
EOL

  # Make scripts executable
  chmod +x "$SCRIPTS_DIR/minikube_control.py" "$SCRIPTS_DIR/deploy_app.py"
"$PROJECT_ROOT/deploy_sre_app.sh"

  log "INFO" "Automation scripts created successfully"
}

# ===== Step 8: Main Function =====

main() {
  log "INFO" "Starting React SRE Application setup"

  # Step 1: Initial Setup
  check_wsl
  setup_project_directories

  # Step 2: Install Dependencies
  install_dependencies

  # Step 3: Python Environment
  setup_python_environment

  # Step 4: Create React App
  create_react_app
  create_react_sre_files

  # Step 5: Kubernetes Configuration
  create_kubernetes_config

  # Step 6: Monitoring Configuration
```

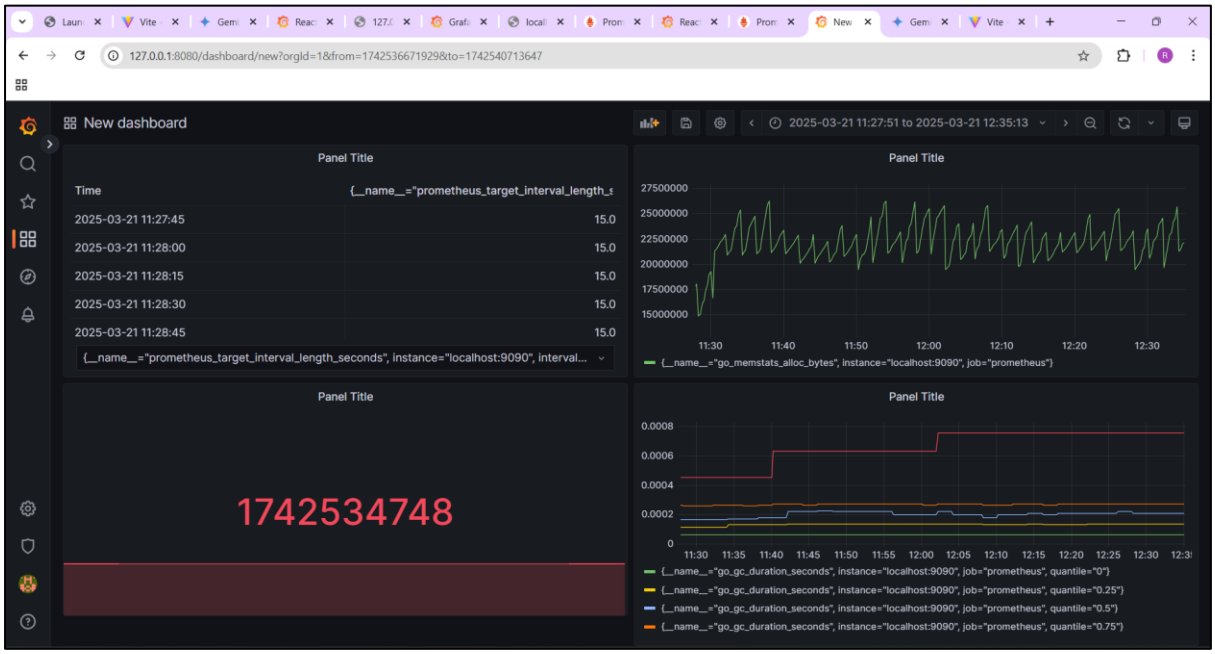
```
  create_monitoring_config

  # Step 7: Automation Scripts
  create_automation_scripts

  log "INFO" "Setup completed successfully!"
  log "INFO" "To deploy the application, run: $PROJECT_ROOT/deploy_sre_app.sh"
}

# Execute main function
main
```

after this it was showing following step:

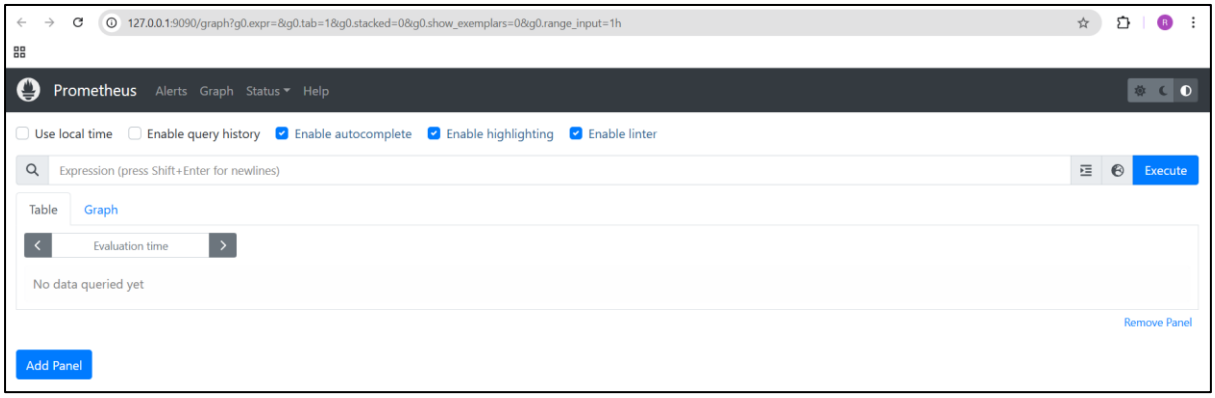


Problems Output Debug Console Terminal Ports 8
bash - react-sre-project
[INFO] 2025-03-21 11:10:43 - Waiting for application to be ready...
[DEBUG] 2025-03-21 11:10:43 - Running command: kubectl get pods -n react-sre-app -l app=react-sre-app -o jsonpath='{.items[*].status.phase}'
[INFO] 2025-03-21 11:10:43 - Application is ready
[INFO] 2025-03-21 11:10:43 - Setting up port forwarding...
[INFO] 2025-03-21 11:10:43 - Port forwarding set up successfully
[INFO] 2025-03-21 11:10:43 - Application available at http://localhost:3000
[INFO] 2025-03-21 11:10:43 - Prometheus available at http://localhost:9090
[INFO] 2025-03-21 11:10:43 - Grafana available at http://localhost:8080 (admin/admin)
[INFO] 2025-03-21 11:10:43 - Press Ctrl+C to stop port forwarding

Dashboard in Grafana:



Prometheus Interface:



React-webpage: