MCB Orchestration Guide for RedHat GBOGEB-RH

Executive Summary

This comprehensive guide provides detailed instructions for deploying MCB (Multi-Cloud Broker) orchestration on RedHat GBOGEB-RH infrastructure with GitHub authentication integration. The guide covers container orchestration, resource management, and integration patterns specific to RedHat OpenShift environments.

1. Architecture Overview

1.1 GBOGEB-RH Infrastructure Components

```yaml
# GBOGEB-RH Architecture Definition
apiVersion: v1
kind: ConfigMap
metadata:
  name: gbogeb-rh-config
  namespace: mcb-orchestration
data:
  architecture.yaml: |
    components:
      - name: "OpenShift Container Platform"
        version: "4.14+"
        role: "Container Orchestration"
      - name: "Red Hat Developer Hub"
        version: "1.3+"
        role: "Developer Platform"
      - name: "GitHub Enterprise"
        version: "3.9+"
        role: "Source Control & Authentication"
      - name: "MCB Orchestrator"
        version: "2.1+"
        role: "Multi-Cloud Workflow Management"
      - name: "GBOGEB Authentication Service"
        version: "1.0+"
        role: "Identity & Access Management"
```

1.2 Network Architecture

```
graph TB
    subgraph "GBOGEB-RH Infrastructure"
        subgraph "DMZ Zone"
            LB[Load Balancer]
            GW[API Gateway]
        end

        subgraph "Application Zone"
            OCP[OpenShift Cluster]
            RDH[Red Hat Developer Hub]
            MCB[MCB Orchestrator]
        end

        subgraph "Data Zone"
            DB[(Database)]
            CACHE[(Redis Cache)]
            STORAGE[(Persistent Storage)]
        end

        subgraph "External Services"
            GH[GitHub Enterprise]
            AUTH[GBOGEB Auth Service]
        end
    end

    LB --> GW
    GW --> OCP
    GW --> RDH
    OCP --> MCB
    MCB --> DB
    MCB --> CACHE
    MCB --> STORAGE
    RDH --> GH
    RDH --> AUTH
    MCB --> GH
```

2. GitHub Authentication Setup

2.1 GitHub App Configuration

```bash
#!/bin/bash
# GitHub App Setup for GBOGEB-RH

# Create GitHub App configuration
cat > github-app-config.json << EOF
{
  "name": "GBOGEB-RH-MCB-Orchestrator",
  "description": "MCB Orchestration platform for GBOGEB-RH infrastructure",
  "homepage_url": "https://mcb.gbogeb-rh.internal",
  "callback_urls": [
    "https://mcb.gbogeb-rh.internal/auth/github/callback",
    "https://rhdh.gbogeb-rh.internal/api/auth/github/handler/frame"
  ],
  "webhook_url": "https://mcb.gbogeb-rh.internal/webhooks/github",
  "permissions": {
    "repository_administration": "read",
    "repository_contents": "write",
    "repository_metadata": "read",
    "repository_pull_requests": "write",
    "organization_members": "read",
    "organization_administration": "read"
  },
  "events": [
    "push",
    "pull_request",
    "workflow_run",
    "deployment_status"
  ]
}
EOF

# Generate private key and store securely
openssl genpkey -algorithm RSA -out github-app-private-key.pem -pkcs8 -pass pass:$GIT-
HUB_APP_PASSPHRASE
```

2.2 OpenShift Secret Configuration

```yaml
# GitHub Authentication Secrets
apiVersion: v1
kind: Secret
metadata:
  name: github-auth-secrets
  namespace: mcb-orchestration
type: Opaque
stringData:
  GITHUB_APP_ID: "${GITHUB_APP_ID}"
  GITHUB_CLIENT_ID: "${GITHUB_CLIENT_ID}"
  GITHUB_CLIENT_SECRET: "${GITHUB_CLIENT_SECRET}"
  GITHUB_PRIVATE_KEY: |
    -----BEGIN PRIVATE KEY-----
    ${GITHUB_PRIVATE_KEY_CONTENT}
    -----END PRIVATE KEY-----
  GITHUB_WEBHOOK_SECRET: "${GITHUB_WEBHOOK_SECRET}"
  GBOGEB_AUTH_TOKEN: "${GBOGEB_AUTH_TOKEN}"

---
# GitHub Integration ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: github-integration-config
  namespace: mcb-orchestration
data:
  config.yaml: |
    github:
      host: "github.gbogeb-rh.internal"
      api_url: "https://api.github.gbogeb-rh.internal"
      organization: "GBOGEB"
      repositories:
        - "pipeline-automation-hub"
        - "dmaic-recursive-pipeline"
        - "orchestration-sandbox"
      authentication:
        type: "github_app"
        app_id: "${GITHUB_APP_ID}"
        installation_id: "${GITHUB_INSTALLATION_ID}"
      webhooks:
        enabled: true
        events: ["push", "pull_request", "workflow_run"]
        endpoint: "/webhooks/github"
```

3. Container Orchestration Setup

3.1 MCB Orchestrator Deployment

```yaml
# MCB Orchestrator Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mcb-orchestrator
  namespace: mcb-orchestration
  labels:
    app: mcb-orchestrator
    component: orchestration-engine
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mcb-orchestrator
  template:
    metadata:
      labels:
        app: mcb-orchestrator
    spec:
      serviceAccountName: mcb-orchestrator-sa
      containers:
      - name: orchestrator
        image: registry.gbogeb-rh.internal/mcb/orchestrator:2.1.0
        ports:
        - containerPort: 8080
          name: http
        - containerPort: 9090
          name: metrics
        env:
        - name: GITHUB_APP_ID
          valueFrom:
            secretKeyRef:
              name: github-auth-secrets
              key: GITHUB_APP_ID
        - name: GITHUB_PRIVATE_KEY
          valueFrom:
            secretKeyRef:
              name: github-auth-secrets
              key: GITHUB_PRIVATE_KEY
        - name: GBOGEB_AUTH_ENDPOINT
          value: "https://auth.gbogeb-rh.internal"
        - name: REDIS_URL
          value: "redis://redis-cluster.mcb-orchestration.svc.cluster.local:6379"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: database-secrets
              key: connection-string
        volumeMounts:
        - name: config-volume
          mountPath: /etc/mcb/config
        - name: github-key-volume
          mountPath: /etc/mcb/github
          readOnly: true
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          initialDelaySeconds: 5
          periodSeconds: 5
      volumes:
      - name: config-volume
        configMap:
          name: github-integration-config
      - name: github-key-volume
        secret:
          secretName: github-auth-secrets
          items:
          - key: GITHUB_PRIVATE_KEY
            path: private-key.pem

---
# Service for MCB Orchestrator
apiVersion: v1
kind: Service
metadata:
  name: mcb-orchestrator-service
  namespace: mcb-orchestration
spec:
  selector:
    app: mcb-orchestrator
  ports:
  - name: http
    port: 80
    targetPort: 8080
  - name: metrics
    port: 9090
    targetPort: 9090
  type: ClusterIP

---
# Route for external access
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: mcb-orchestrator-route
  namespace: mcb-orchestration
spec:
  host: mcb.gbogeb-rh.internal
  to:
    kind: Service
    name: mcb-orchestrator-service
  port:
    targetPort: http
  tls:
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
```

3.2 Worker Node Configuration

```yaml
# MCB Worker Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mcb-worker
  namespace: mcb-orchestration
spec:
  replicas: 5
  selector:
    matchLabels:
      app: mcb-worker
  template:
    metadata:
      labels:
        app: mcb-worker
    spec:
      serviceAccountName: mcb-worker-sa
      containers:
      - name: worker
        image: registry.gbogeb-rh.internal/mcb/worker:2.1.0
        env:
        - name: ORCHESTRATOR_ENDPOINT
          value: "http://mcb-orchestrator-service.mcb-orchestration.svc.cluster.local"
        - name: WORKER_ID
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: GITHUB_TOKEN
          valueFrom:
            secretKeyRef:
              name: github-auth-secrets
              key: GITHUB_CLIENT_SECRET
        resources:
          requests:
            memory: "256Mi"
            cpu: "100m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        volumeMounts:
        - name: workspace
          mountPath: /workspace
        - name: docker-socket
          mountPath: /var/run/docker.sock
      volumes:
      - name: workspace
        emptyDir: {}
      - name: docker-socket
        hostPath:
          path: /var/run/docker.sock
          type: Socket

---
# Horizontal Pod Autoscaler for Workers
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: mcb-worker-hpa
  namespace: mcb-orchestration
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mcb-worker
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

4. Resource Management

4.1 Resource Quotas and Limits

```yaml
# Namespace Resource Quota
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mcb-orchestration-quota
  namespace: mcb-orchestration
spec:
  hard:
    requests.cpu: "10"
    requests.memory: 20Gi
    limits.cpu: "20"
    limits.memory: 40Gi
    persistentvolumeclaims: "10"
    services: "20"
    secrets: "50"
    configmaps: "50"

---
# Limit Range for Pods
apiVersion: v1
kind: LimitRange
metadata:
  name: mcb-orchestration-limits
  namespace: mcb-orchestration
spec:
  limits:
  - type: Pod
    max:
      cpu: "2"
      memory: "4Gi"
    min:
      cpu: "100m"
      memory: "128Mi"
  - type: Container
    default:
      cpu: "500m"
      memory: "512Mi"
    defaultRequest:
      cpu: "100m"
      memory: "128Mi"
    max:
      cpu: "2"
      memory: "4Gi"
    min:
      cpu: "50m"
      memory: "64Mi"
```

4.2 Persistent Storage Configuration

```yaml
# Storage Class for MCB Data
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: mcb-ssd-storage
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp3
  iops: "3000"
  throughput: "125"
  encrypted: "true"
allowVolumeExpansion: true
reclaimPolicy: Retain

---
# Persistent Volume Claim for Database
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mcb-database-pvc
  namespace: mcb-orchestration
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: mcb-ssd-storage
  resources:
    requests:
      storage: 100Gi

---
# Persistent Volume Claim for Redis
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mcb-redis-pvc
  namespace: mcb-orchestration
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: mcb-ssd-storage
  resources:
    requests:
      storage: 50Gi
```

5. Integration Patterns

5.1 DMAIC Integration Pattern

```yaml
# DMAIC Workflow ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: dmaic-workflow-config
  namespace: mcb-orchestration
data:
  dmaic-pipeline.yaml: |
    workflows:
      dmaic_standard:
        phases:
          - name: "define"
            tasks:
              - name: "problem_definition"
                type: "conversation"
                github_integration:
                  repository: "pipeline-automation-hub"
                  path: "workflows/define"
                  trigger: "manual"
              - name: "stakeholder_analysis"
                type: "analysis"
                dependencies: ["problem_definition"]

          - name: "measure"
            tasks:
              - name: "data_collection"
                type: "orchestration"
                github_integration:
                  repository: "pipeline-automation-hub"
                  path: "workflows/measure"
                  trigger: "webhook"
              - name: "baseline_metrics"
                type: "analysis"
                dependencies: ["data_collection"]

          - name: "analyze"
            tasks:
              - name: "root_cause_analysis"
                type: "conversation"
                multi_ai_handover: true
                agents: ["analyst", "domain_expert"]
              - name: "hypothesis_testing"
                type: "analysis"
                dependencies: ["root_cause_analysis"]

          - name: "improve"
            tasks:
              - name: "solution_design"
                type: "orchestration"
                github_integration:
                  repository: "pipeline-automation-hub"
                  path: "workflows/improve"
                  auto_pr: true
              - name: "pilot_implementation"
                type: "deployment"
                dependencies: ["solution_design"]

          - name: "control"
            tasks:
              - name: "monitoring_setup"
                type: "orchestration"
                continuous: true
              - name: "documentation"
                type: "conversation"
                github_integration:
                  repository: "pipeline-automation-hub"
                  path: "docs/control"
                  auto_commit: true
```

5.2 Multi-AI Handover Configuration

```
# Multi-AI Agent Configuration
apiVersion: v1
kind: ConfigMap
metadata:
  name: multi-ai-config
  namespace: mcb-orchestration
data:
  agents.yaml: |
    agents:
      - id: "conversation_agent"
        type: "conversational"
        capabilities: ["natural_language", "context_management"]
        github_integration:
          permissions: ["read", "comment"]
        handover_rules:
          - condition: "technical_analysis_required"
            target_agent: "technical_analyst"
          - condition: "code_review_needed"
            target_agent: "code_reviewer"

      - id: "technical_analyst"
        type: "analytical"
        capabilities: ["data_analysis", "pattern_recognition"]
        github_integration:
          permissions: ["read", "write", "create_branch"]
        handover_rules:
          - condition: "implementation_ready"
            target_agent: "orchestration_agent"

      - id: "orchestration_agent"
        type: "orchestration"
        capabilities: ["workflow_management", "resource_allocation"]
        github_integration:
          permissions: ["admin"]
        handover_rules:
          - condition: "deployment_complete"
            target_agent: "conversation_agent"

      - id: "code_reviewer"
        type: "review"
        capabilities: ["code_analysis", "security_scanning"]
        github_integration:
          permissions: ["read", "review", "approve"]
        handover_rules:
          - condition: "review_complete"
            target_agent: "orchestration_agent"
```

6. Monitoring and Observability

6.1 Prometheus Configuration

```yaml
# Prometheus ServiceMonitor
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: mcb-orchestrator-monitor
  namespace: mcb-orchestration
spec:
  selector:
    matchLabels:
      app: mcb-orchestrator
  endpoints:
  - port: metrics
    interval: 30s
    path: /metrics

---
# Grafana Dashboard ConfigMap
apiVersion: v1
kind: ConfigMap
metadata:
  name: mcb-dashboard
  namespace: mcb-orchestration
data:
  dashboard.json: |
    {
      "dashboard": {
        "title": "MCB Orchestration Dashboard",
        "panels": [
          {
            "title": "Task Execution Rate",
            "type": "graph",
            "targets": [
              {
                "expr": "rate(mcb_tasks_completed_total[5m])",
                "legendFormat": "Tasks/sec"
              }
            ]
          },
          {
            "title": "GitHub API Rate Limit",
            "type": "singlestat",
            "targets": [
              {
                "expr": "github_api_rate_limit_remaining",
                "legendFormat": "Remaining"
              }
            ]
          },
          {
            "title": "Agent Handover Success Rate",
            "type": "graph",
            "targets": [
              {
                "expr": "rate(mcb_handover_success_total[5m]) /
rate(mcb_handover_attempts_total[5m])",
                "legendFormat": "Success Rate"
              }
            ]
          }
        ]
      }
    }
```

6.2 Alerting Rules

```yaml
# PrometheusRule for MCB Alerts
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: mcb-orchestration-alerts
  namespace: mcb-orchestration
spec:
  groups:
  - name: mcb.rules
    rules:
    - alert: MCBOrchestratorDown
      expr: up{job="mcb-orchestrator"} == 0
      for: 1m
      labels:
        severity: critical
      annotations:
        summary: "MCB Orchestrator is down"
        description: "MCB Orchestrator has been down for more than 1 minute"

    - alert: GitHubAPIRateLimitLow
      expr: github_api_rate_limit_remaining < 100
      for: 5m
      labels:
        severity: warning
      annotations:
        summary: "GitHub API rate limit is low"
        description: "GitHub API rate limit remaining: {{ $value }}"

    - alert: HighTaskFailureRate
      expr: rate(mcb_tasks_failed_total[5m]) / rate(mcb_tasks_total[5m]) > 0.1
      for: 2m
      labels:
        severity: warning
      annotations:
        summary: "High task failure rate detected"
        description: "Task failure rate is {{ $value | humanizePercentage }}"
```

7. Security Configuration

7.1 Network Policies

```yaml
# Network Policy for MCB Orchestration
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: mcb-orchestration-netpol
  namespace: mcb-orchestration
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: openshift-ingress
    ports:
    - protocol: TCP
      port: 8080
  - from:
    - podSelector:
        matchLabels:
          app: mcb-worker
    ports:
    - protocol: TCP
      port: 8080
  egress:
  - to: []
    ports:
    - protocol: TCP
      port: 443  # HTTPS to GitHub
    - protocol: TCP
      port: 53   # DNS
    - protocol: UDP
      port: 53   # DNS
  - to:
    - podSelector:
        matchLabels:
          app: redis
    ports:
    - protocol: TCP
      port: 6379
  - to:
    - podSelector:
        matchLabels:
          app: postgresql
    ports:
    - protocol: TCP
      port: 5432
```

7.2 Security Context Constraints

```yaml
# Security Context Constraint for MCB
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: mcb-orchestrator-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
  ranges:
  - min: 1000
  - max: 65535
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- MKNOD
- SETUID
- SETGID
runAsUser:
  type: MustRunAsRange
  uidRangeMin: 1000
  uidRangeMax: 65535
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

8. Deployment Scripts

8.1 Automated Deployment Script

```bash
#!/bin/bash
# MCB Orchestration Deployment Script for GBOGEB-RH

set -euo pipefail

# Configuration
NAMESPACE="mcb-orchestration"
GITHUB_ORG="GBOGEB"
REGISTRY="registry.gbogeb-rh.internal"

# Colors for output
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

log() {
    echo -e "${GREEN}[$(date +'%Y-%m-%d %H:%M:%S')] $1${NC}"
}

warn() {
    echo -e "${YELLOW}[$(date +'%Y-%m-%d %H:%M:%S')] WARNING: $1${NC}"
}

error() {
    echo -e "${RED}[$(date +'%Y-%m-%d %H:%M:%S')] ERROR: $1${NC}"
    exit 1
}

# Check prerequisites
check_prerequisites() {
    log "Checking prerequisites..."

    # Check if oc is installed and logged in
    if ! command -v oc &> /dev/null; then
        error "OpenShift CLI (oc) is not installed"
    fi

    if ! oc whoami &> /dev/null; then
        error "Not logged in to OpenShift cluster"
    fi

    # Check if required environment variables are set
    required_vars=("GITHUB_APP_ID" "GITHUB_CLIENT_ID" "GITHUB_CLIENT_SECRET" "GIT-
HUB_PRIVATE_KEY" "GBOGEB_AUTH_TOKEN")
    for var in "${required_vars[@]}"; do
        if [[ -z "${!var:-}" ]]; then
            error "Environment variable $var is not set"
        fi
    done

    log "Prerequisites check passed"
}

# Create namespace and RBAC
setup_namespace() {
    log "Setting up namespace and RBAC..."

    # Create namespace
    oc create namespace $NAMESPACE --dry-run=client -o yaml | oc apply -f -

    # Label namespace for monitoring
    oc label namespace $NAMESPACE monitoring=enabled --overwrite

    # Create service accounts
    cat <<EOF | oc apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mcb-orchestrator-sa
  namespace: $NAMESPACE
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: mcb-worker-sa
  namespace: $NAMESPACE
EOF

    # Create cluster role and binding
    cat <<EOF | oc apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: mcb-orchestrator-role
rules:
- apiGroups: [""]
  resources: ["pods", "services", "configmaps", "secrets"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["apps"]
  resources: ["deployments", "replicasets"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["batch"]
  resources: ["jobs", "cronjobs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: mcb-orchestrator-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: mcb-orchestrator-role
subjects:
- kind: ServiceAccount
  name: mcb-orchestrator-sa
  namespace: $NAMESPACE
EOF

    log "Namespace and RBAC setup completed"
}

# Deploy secrets and config maps
deploy_config() {
    log "Deploying configuration..."

    # Create GitHub auth secrets
    oc create secret generic github-auth-secrets \
        --from-literal=GITHUB_APP_ID="$GITHUB_APP_ID" \
        --from-literal=GITHUB_CLIENT_ID="$GITHUB_CLIENT_ID" \
        --from-literal=GITHUB_CLIENT_SECRET="$GITHUB_CLIENT_SECRET" \
        --from-literal=GITHUB_PRIVATE_KEY="$GITHUB_PRIVATE_KEY" \
        --from-literal=GBOGEB_AUTH_TOKEN="$GBOGEB_AUTH_TOKEN" \
        --namespace=$NAMESPACE \
        --dry-run=client -o yaml | oc apply -f -

    # Apply all configuration files
    find ./config -name "*.yaml" -exec oc apply -f {} \;

    log "Configuration deployment completed"
}

# Deploy MCB components
deploy_mcb() {
    log "Deploying MCB Orchestrator components..."

    # Apply deployment manifests
    oc apply -f ./manifests/

    # Wait for deployments to be ready
    log "Waiting for deployments to be ready..."
    oc rollout status deployment/mcb-orchestrator -n $NAMESPACE --timeout=300s
    oc rollout status deployment/mcb-worker -n $NAMESPACE --timeout=300s

    log "MCB components deployment completed"
}

# Setup monitoring
setup_monitoring() {
    log "Setting up monitoring..."

    # Apply monitoring configuration
    oc apply -f ./monitoring/

    log "Monitoring setup completed"
}

# Verify deployment
verify_deployment() {
    log "Verifying deployment..."

    # Check pod status
    oc get pods -n $NAMESPACE

    # Check service endpoints
    oc get endpoints -n $NAMESPACE

    # Test MCB orchestrator health endpoint
    MCB_URL=$(oc get route mcb-orchestrator-route -n $NAMESPACE -o jsonpath='{.spec.hos
t}')
    if curl -f "https://$MCB_URL/health" > /dev/null 2>&1; then
        log "MCB Orchestrator health check passed"
    else
        warn "MCB Orchestrator health check failed"
    fi

    log "Deployment verification completed"
}

# Main deployment function
main() {
    log "Starting MCB Orchestration deployment for GBOGEB-RH"

    check_prerequisites
    setup_namespace
    deploy_config
    deploy_mcb
    setup_monitoring
    verify_deployment

    log "MCB Orchestration deployment completed successfully!"
    log "Access the MCB Orchestrator at: https://$(oc get route mcb-orchestrator-route
-n $NAMESPACE -o jsonpath='{.spec.host}')"
}

# Run main function
main "$@"
```

9. Troubleshooting Guide

9.1 Common Issues and Solutions

```
# Troubleshooting Script
#!/bin/bash

troubleshoot_github_auth() {
    echo "Troubleshooting GitHub authentication..."

    # Check GitHub App configuration
    oc get secret github-auth-secrets -n mcb-orchestration -o yaml

    # Test GitHub API connectivity
    kubectl exec -n mcb-orchestration deployment/mcb-orchestrator -- \
        curl -H "Authorization: Bearer $(oc get secret github-auth-secrets -n mcb-or-
chestration -o jsonpath='{.data.GITHUB_CLIENT_SECRET}' | base64 -d)" \
        https://api.github.gbogeb-rh.internal/user
}

troubleshoot_networking() {
    echo "Troubleshooting network connectivity..."

    # Check network policies
    oc get networkpolicy -n mcb-orchestration

    # Test internal service connectivity
    oc exec -n mcb-orchestration deployment/mcb-worker -- \
        curl -f http://mcb-orchestrator-service.mcb-orchestration.svc.cluster.local/
health
}

troubleshoot_resources() {
    echo "Troubleshooting resource issues..."

    # Check resource usage
    oc top pods -n mcb-orchestration
    oc describe quota -n mcb-orchestration

    # Check PVC status
    oc get pvc -n mcb-orchestration
}
```

10. Conclusion

This comprehensive guide provides the foundation for deploying MCB orchestration on RedHat GBOGEB-RH infrastructure with robust GitHub authentication integration. The setup includes:

- **Secure Authentication**: GitHub App integration with proper secret management
- **Scalable Architecture**: Container orchestration with auto-scaling capabilities
- **Resource Management**: Proper quotas, limits, and storage configuration
- **Monitoring & Observability**: Comprehensive metrics and alerting
- **Security**: Network policies and security context constraints
- **Integration Patterns**: DMAIC workflow and multi-AI handover support

The deployment is production-ready and follows RedHat OpenShift best practices for enterprise environments.