Kubernetes Deployment Guide for Archivist

This guide explains how to deploy the Archivist application on Kubernetes for production-grade scalability, high availability, and orchestration.

Table of Contents

Architecture Overview

Archivist's microservices architecture maps naturally to Kubernetes:

```
┌─────────────────────────────────────────────────────────────────────────
┐
│                        KUBERNETES CLUSTER
│
│  ┌──────────────────────────────────────────────────────────────────┐
│
│  │                      INGRESS CONTROLLER                           │
│
│  │                  (nginx-ingress / traefik)                        │
│
│  └────────────────────────────┬─────────────────────────────────────┘
│
│                               │
│
```

```
│  ┌────────────────────────────┼──────────────────────────────────────┐
│  │                    APPLICATION NAMESPACE                           │
│  │  ┌─────────────┐  ┌─────────────┐  ┌─────────────────────────┐    │
│  │  │ RAG API     │  │ Graph       │  │ Archivist Worker        │    │
│  │  │ Deployment  │  │ Service     │  │ Jobs / CronJobs         │    │
│  │  │ (FastAPI)   │  │ Deployment  │  │ (PDF Processing)        │    │
│  │  │ Replicas: 3 │  │ Replicas: 2 │  │                         │    │
│  │  └─────────────┘  └─────────────┘  └─────────────────────────┘    │
│  └───────────────────────────────────────────────────────────────────┘
│
│  ┌───────────────────────────────────────────────────────────────────┐
│  │                     DATA NAMESPACE                                 │
│  │  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐  ┌──────────┐  │
│  │  │ Neo4j       │  │ Qdrant      │  │ Redis       │  │ Kafka    │  │
│  │  │ StatefulSet │  │ StatefulSet │  │ StatefulSet │  │ (Strimzi)│  │
│  │  │ (Graph DB)  │  │ (Vector DB) │  │ (Cache)     │  │          │  │
│  │  └─────────────┘  └─────────────┘  └─────────────┘  └──────────┘  │
│  └───────────────────────────────────────────────────────────────────┘
│
│  ┌───────────────────────────────────────────────────────────────────┐
│  │                    PERSISTENT STORAGE                              │
```

```
│
│   │  StorageClass: SSD (gp3/premium-lrs) for databases            │
│
│   │  PersistentVolumeClaims for: Neo4j, Qdrant, Redis, PDF storage │
│
│   └───────────────────────────────────────────────────────────────┘
│
└──────────────────────────────────────────────────────────────────────
┘
```

Why Kubernetes for Archivist

Current Challenges (Docker Compose)

Challenge	Impact
Single-node deployment	No fault tolerance
Manual scaling	Cannot handle traffic spikes
No self-healing	Service failures require manual restart
Resource contention	All services share same host resources
No rolling updates	Downtime during deployments

Kubernetes Benefits

Benefit	Archivist Use Case
Horizontal Pod Autoscaling	Scale RAG API pods during high query load
StatefulSets	Reliable Neo4j, Qdrant, Redis deployments with persistent storage
Jobs/CronJobs	Batch PDF processing, scheduled index rebuilds
Service Discovery	Automatic DNS for inter-service communication
ConfigMaps/Secrets	Centralized config, secure API key management
Health Checks	Auto-restart failed containers
Resource Quotas	Prevent runaway Gemini API calls
Rolling Updates	Zero-downtime deployments
Multi-zone HA	Survive data center failures

Prerequisites

Required Tools

```
# kubectl - Kubernetes CLI
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/releas

# helm - Package manager for Kubernetes
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

# Optional: k9s for cluster management
brew install derailed/k9s/k9s
```

Cluster Options

Option	Best For	Notes
Minikube	Local development	Single-node, limited resources
kind	CI/CD testing	Docker-based, ephemeral
EKS (AWS)	Production	Managed, integrates with AWS services
GKE (GCP)	Production	Best for Gemini API (same network)
AKS (Azure)	Production	Managed, enterprise features

Kubernetes Components

Namespace Organization

```
# namespaces.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: archivist
  labels:
    app: archivist
    environment: production
---
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: archivist-data
  labels:
    app: archivist
    tier: data
```

1. RAG API Service (Deployment)

```
# rag-api-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rag-api
  namespace: archivist
  labels:
    app: rag-api
    tier: api
spec:
  replicas: 3
  selector:
    matchLabels:
      app: rag-api
  template:
    metadata:
      labels:
        app: rag-api
    spec:
      containers:
      - name: rag-api
        image: archivist/rag-api:latest
        ports:
        - containerPort: 8000
        env:
        - name: GEMINI_API_KEY
          valueFrom:
            secretKeyRef:
              name: archivist-secrets
              key: gemini-api-key
        - name: QDRANT_HOST
          value: "qdrant.archivist-data.svc.cluster.local"
```

```yaml
        - name: QDRANT_PORT
          value: "6333"
        - name: REDIS_HOST
          value: "redis.archivist-data.svc.cluster.local"
        - name: NEO4J_URI
          value: "bolt://neo4j.archivist-data.svc.cluster.local:7687"
        resources:
          requests:
            memory: "512Mi"
            cpu: "250m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 8000
          initialDelaySeconds: 5
          periodSeconds: 5
---
apiVersion: v1
kind: Service
metadata:
  name: rag-api
  namespace: archivist
spec:
  selector:
    app: rag-api
  ports:
  - port: 8000
    targetPort: 8000
  type: ClusterIP
```

2. Graph Service (Deployment)

```yaml
# graph-service-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: graph-service
  namespace: archivist
  labels:
    app: graph-service
spec:
  replicas: 2
  selector:
    matchLabels:
      app: graph-service
  template:
    metadata:
      labels:
        app: graph-service
    spec:
      containers:
      - name: graph-service
        image: archivist/graph-service:latest
        ports:
        - containerPort: 8081
        env:
        - name: GEMINI_API_KEY
          valueFrom:
            secretKeyRef:
              name: archivist-secrets
              key: gemini-api-key
        - name: NEO4J_URI
          value: "bolt://neo4j.archivist-data.svc.cluster.local:7687"
        - name: NEO4J_USER
          value: "neo4j"
        - name: NEO4J_PASSWORD
          valueFrom:
            secretKeyRef:
              name: archivist-secrets
              key: neo4j-password
        - name: KAFKA_BOOTSTRAP_SERVERS
          value: "kafka-cluster-kafka-bootstrap.archivist-data.svc.cluste
        resources:
          requests:
            memory: "256Mi"
            cpu: "100m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8081
          initialDelaySeconds: 30
          periodSeconds: 10
```

3. Neo4j (StatefulSet)

```
# neo4j-statefulset.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: neo4j
  namespace: archivist-data
spec:
  serviceName: neo4j
  replicas: 1  # Use Neo4j Cluster for HA (3+ replicas)
  selector:
    matchLabels:
      app: neo4j
  template:
    metadata:
      labels:
        app: neo4j
    spec:
      containers:
      - name: neo4j
        image: neo4j:5.13-community
        ports:
        - containerPort: 7474
          name: http
        - containerPort: 7687
          name: bolt
        env:
        - name: NEO4J_AUTH
          valueFrom:
            secretKeyRef:
              name: neo4j-credentials
              key: auth
        - name: NEO4J_PLUGINS
          value: '["graph-data-science", "apoc"]'
        - name: NEO4J_dbms_memory_heap_max__size
          value: "2G"
        volumeMounts:
        - name: neo4j-data
          mountPath: /data
        - name: neo4j-logs
          mountPath: /logs
        resources:
          requests:
            memory: "2Gi"
            cpu: "500m"
          limits:
            memory: "4Gi"
            cpu: "2000m"
  volumeClaimTemplates:
  - metadata:
      name: neo4j-data
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: ssd
      resources:
        requests:
          storage: 50Gi
  - metadata:
      name: neo4j-logs
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: standard
      resources:
        requests:
          storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: neo4j
  namespace: archivist-data
spec:
  selector:
    app: neo4j
  ports:
```

```
  - port: 7474
    targetPort: 7474
    name: http
  - port: 7687
    targetPort: 7687
    name: bolt
  type: ClusterIP
```

4. Qdrant (StatefulSet)

```
# qdrant-statefulset.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: qdrant
  namespace: archivist-data
spec:
  serviceName: qdrant
  replicas: 1  # Can scale to 3+ for distributed mode
  selector:
    matchLabels:
      app: qdrant
  template:
    metadata:
      labels:
        app: qdrant
    spec:
      containers:
      - name: qdrant
        image: qdrant/qdrant:v1.7.4
        ports:
        - containerPort: 6333
          name: http
        - containerPort: 6334
          name: grpc
        volumeMounts:
        - name: qdrant-storage
          mountPath: /qdrant/storage
        resources:
          requests:
            memory: "1Gi"
            cpu: "250m"
          limits:
            memory: "4Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /healthz
            port: 6333
          initialDelaySeconds: 30
          periodSeconds: 10
  volumeClaimTemplates:
  - metadata:
      name: qdrant-storage
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: ssd
      resources:
        requests:
          storage: 100Gi
---
apiVersion: v1
kind: Service
metadata:
  name: qdrant
  namespace: archivist-data
spec:
  selector:
    app: qdrant
  ports:
  - port: 6333
    targetPort: 6333
    name: http
  - port: 6334
    targetPort: 6334
    name: grpc
```

5. Redis (StatefulSet)

```
# redis-statefulset.yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis
  namespace: archivist-data
spec:
  serviceName: redis
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis
        image: redis:7.2-alpine
        ports:
        - containerPort: 6379
        command:
        - redis-server
        - --appendonly
        - "yes"
        - --maxmemory
        - "512mb"
        - --maxmemory-policy
        - "allkeys-lru"
        volumeMounts:
        - name: redis-data
          mountPath: /data
        resources:
          requests:
            memory: "256Mi"
            cpu: "100m"
          limits:
            memory: "1Gi"
            cpu: "500m"
        livenessProbe:
          exec:
            command:
            - redis-cli
            - ping
          initialDelaySeconds: 30
          periodSeconds: 10
  volumeClaimTemplates:
  - metadata:
      name: redis-data
    spec:
      accessModes: ["ReadWriteOnce"]
      storageClassName: ssd
      resources:
        requests:
          storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: redis
  namespace: archivist-data
spec:
  selector:
    app: redis
  ports:
  - port: 6379
    targetPort: 6379
```

6. PDF Processing Job

```
# pdf-processor-job.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pdf-processor-batch
  namespace: archivist
spec:
  parallelism: 4  # Process 4 PDFs concurrently
  completions: 10 # Total PDFs to process
  backoffLimit: 3
  template:
    spec:
      containers:
      - name: processor
        image: archivist/archivist:latest
        command: ["./archivist", "process", "--batch"]
        env:
        - name: GEMINI_API_KEY
          valueFrom:
            secretKeyRef:
              name: archivist-secrets
              key: gemini-api-key
        volumeMounts:
        - name: pdf-storage
          mountPath: /app/lib
        - name: output-storage
          mountPath: /app/tex_files
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "4Gi"
            cpu: "2000m"
      volumes:
      - name: pdf-storage
        persistentVolumeClaim:
          claimName: pdf-input-pvc
      - name: output-storage
        persistentVolumeClaim:
          claimName: latex-output-pvc
      restartPolicy: OnFailure
```

7. Index Rebuild CronJob

```
# index-rebuild-cronjob.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: index-rebuild
  namespace: archivist
spec:
  schedule: "0 2 * * *"  # Daily at 2 AM
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: indexer
            image: archivist/rag-api:latest
            command: ["python", "-m", "indexer", "--rebuild"]
            env:
```

```
            - name: GEMINI_API_KEY
              valueFrom:
                secretKeyRef:
                  name: archivist-secrets
                  key: gemini-api-key
            - name: QDRANT_HOST
              value: "qdrant.archivist-data.svc.cluster.local"
            resources:
              requests:
                memory: "2Gi"
                cpu: "500m"
              limits:
                memory: "8Gi"
                cpu: "2000m"
          restartPolicy: OnFailure
```

Configuration Management

Secrets

```
# secrets.yaml
apiVersion: v1
kind: Secret
metadata:
  name: archivist-secrets
  namespace: archivist
type: Opaque
stringData:
  gemini-api-key: "your-gemini-api-key"
  neo4j-password: "your-neo4j-password"
---
apiVersion: v1
kind: Secret
metadata:
  name: neo4j-credentials
  namespace: archivist-data
type: Opaque
stringData:
  auth: "neo4j/your-neo4j-password"
```

ConfigMap

```
# configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: archivist-config
  namespace: archivist
data:
  config.yaml: |
    processing:
      max_workers: 8
      batch_size: 10
      timeout_per_paper: 600s

    gemini:
      model: "models/gemini-2.0-flash-exp"
      temperature: 0.3
      agentic_workflow: true

    graph:
      async_building: true
      max_graph_workers: 2
      search:
        vector_weight: 0.5
        graph_weight: 0.3
        keyword_weight: 0.2

    qdrant:
      collection_name: "archivist_papers"
      vector_size: 768
      distance: "Cosine"

    cache:
      ttl_days: 30
```

Scaling Strategies

Horizontal Pod Autoscaler (HPA)

```yaml
# hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: rag-api-hpa
  namespace: archivist
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: rag-api
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 10
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 0
      policies:
      - type: Percent
        value: 100
        periodSeconds: 15
```

Vertical Pod Autoscaler (VPA)

```
# vpa.yaml
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: rag-api-vpa
  namespace: archivist
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: rag-api
  updatePolicy:
    updateMode: "Auto"
  resourcePolicy:
    containerPolicies:
    - containerName: rag-api
      minAllowed:
        cpu: 100m
        memory: 256Mi
      maxAllowed:
        cpu: 4
        memory: 8Gi
```

Monitoring and Observability

Prometheus ServiceMonitor

```
# servicemonitor.yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: archivist-metrics
  namespace: archivist
spec:
  selector:
    matchLabels:
      app: rag-api
  endpoints:
  - port: http
```

```
    path: /metrics
    interval: 30s
```

Key Metrics to Monitor

Service	Metric	Alert Threshold
RAG API	Response latency p99	> 2s
RAG API	Error rate	> 5%
Qdrant	Memory usage	> 80%
Neo4j	Active connections	> 100
Redis	Memory fragmentation	> 1.5
Kafka	Consumer lag	> 1000
All	Pod restarts	> 3/hour

Production Considerations

High Availability Setup

```
Recommended Production Configuration:
- RAG API: 3+ replicas across availability zones
- Graph Service: 2+ replicas
- Neo4j: 3-node cluster (Enterprise)
- Qdrant: 3-node distributed cluster
- Redis: 3-node Sentinel or Redis Cluster
- Kafka: 3+ brokers via Strimzi operator
```

Resource Allocation Guide

Component	CPU Request	CPU Limit	Memory Request	Memory Limit	Storage
RAG API	250m	1000m	512Mi	2Gi	-
Graph Service	100m	500m	256Mi	1Gi	-
Neo4j	500m	2000m	2Gi	4Gi	50Gi SSD
Qdrant	250m	1000m	1Gi	4Gi	100Gi SSD

Component	CPU Request	CPU Limit	Memory Request	Memory Limit	Storage
Redis	100m	500m	256Mi	1Gi	10Gi SSD
PDF Processor	500m	2000m	1Gi	4Gi	-

Network Policies

```yaml
# network-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: rag-api-policy
  namespace: archivist
spec:
  podSelector:
    matchLabels:
      app: rag-api
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - port: 8000
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          app: archivist
  - to:
    - namespaceSelector:
        matchLabels:
          tier: data
  # Allow external access to Gemini API
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
    ports:
    - port: 443
```

Ingress Configuration

```
# ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: archivist-ingress
  namespace: archivist
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    cert-manager.io/cluster-issuer: letsencrypt-prod
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - api.archivist.example.com
    secretName: archivist-tls
  rules:
  - host: api.archivist.example.com
    http:
      paths:
      - path: /rag
        pathType: Prefix
        backend:
          service:
            name: rag-api
            port:
              number: 8000
      - path: /graph
        pathType: Prefix
        backend:
          service:
            name: graph-service
            port:
              number: 8081
```

Deployment Commands

Initial Setup

```
# Create namespaces
kubectl apply -f namespaces.yaml

# Deploy secrets (use sealed-secrets or external-secrets in production)
kubectl apply -f secrets.yaml

# Deploy ConfigMaps
kubectl apply -f configmap.yaml

# Deploy data layer
kubectl apply -f neo4j-statefulset.yaml
kubectl apply -f qdrant-statefulset.yaml
kubectl apply -f redis-statefulset.yaml

# Wait for data services to be ready
kubectl wait --for=condition=ready pod -l app=neo4j -n archivist-data --t
kubectl wait --for=condition=ready pod -l app=qdrant -n archivist-data --
kubectl wait --for=condition=ready pod -l app=redis -n archivist-data --t

# Deploy application layer
kubectl apply -f rag-api-deployment.yaml
kubectl apply -f graph-service-deployment.yaml

# Deploy autoscaling
kubectl apply -f hpa.yaml

# Deploy ingress
kubectl apply -f ingress.yaml
```

Useful Commands

```
# Check pod status
kubectl get pods -n archivist
kubectl get pods -n archivist-data

# View logs
kubectl logs -f deployment/rag-api -n archivist

# Scale manually
kubectl scale deployment rag-api --replicas=5 -n archivist

# Run one-off PDF processing job
```

```
kubectl create job --from=cronjob/pdf-processor pdf-batch-$(date +%s) -n

# Port-forward for local access
kubectl port-forward svc/rag-api 8000:8000 -n archivist
kubectl port-forward svc/neo4j 7474:7474 7687:7687 -n archivist-data

# Check HPA status
kubectl get hpa -n archivist

# View resource usage
kubectl top pods -n archivist
```

Helm Chart Structure (Recommended)

For production deployments, package the manifests as a Helm chart:

```
archivist-chart/
├── Chart.yaml
├── values.yaml
├── values-production.yaml
├── templates/
│   ├── _helpers.tpl
│   ├── namespaces.yaml
│   ├── secrets.yaml
│   ├── configmap.yaml
│   ├── rag-api/
│   │   ├── deployment.yaml
│   │   ├── service.yaml
│   │   └── hpa.yaml
│   ├── graph-service/
│   │   ├── deployment.yaml
│   │   └── service.yaml
│   ├── neo4j/
│   │   ├── statefulset.yaml
│   │   └── service.yaml
│   ├── qdrant/
│   │   ├── statefulset.yaml
│   │   └── service.yaml
│   ├── redis/
│   │   ├── statefulset.yaml
│   │   └── service.yaml
│   ├── jobs/
│   │   ├── pdf-processor.yaml
│   │   └── index-rebuild.yaml
│   ├── ingress.yaml
│   └── networkpolicy.yaml
└── README.md
```

Install with:

```
helm install archivist ./archivist-chart -f values-production.yaml
```

Migration from Docker Compose

1. **Build container images** and push to a registry (Docker Hub, ECR, GCR)
2. **Export data** from local volumes to cloud storage or PVCs
3. **Test in staging** cluster before production
4. **Use blue-green deployment** for zero-downtime migration
5. **Monitor closely** during initial production rollout

Cost Optimization Tips

1. Use **Spot/Preemptible instances** for PDF processing jobs
2. Enable **cluster autoscaler** to scale down during off-peak hours
3. Use **resource quotas** to prevent runaway costs
4. Consider **serverless** options (Knative, Cloud Run) for bursty workloads
5. Use **managed services** (Cloud Memorystore, Neo4j Aura) to reduce ops overhead

---

Next Steps

1. Set up CI/CD pipeline for automated deployments
2. Configure backup strategies for databases
3. Implement disaster recovery procedures
4. Set up log aggregation (ELK/Loki)
5. Configure alerting (PagerDuty/Slack integration)