juniper_canopy/docs/deployment/KUBERNETES_DEPLOYMENT_PLAN.md

Juniper Canopy Kubernetes Deployment Plan

Version: 1.0.0
Date: 2025-12-30
Author: AI Architect / DevOps Engineer
Status: Planning Document - No Implementation Without Approval

---

Table of Contents

Executive Summary

This document provides a comprehensive plan to containerize and deploy the Juniper Canopy application using Kubernetes for container orchestration. The plan includes:

- **Kubernetes Architecture**: Production-grade deployment with proper resource management, health checks, and autoscaling
- **CI/CD Pipeline**: Robust GitHub Actions pipeline with multi-stage builds, security scanning, and artifact storage

- **Secrets Management**: Secure handling of sensitive configuration using Kubernetes Secrets with optional external secret manager integration
- **Multi-Environment Strategy**: Development → Staging → Production promotion workflow

Key Recommendations Summary

Area	Current State	Recommended State	Priority
Docker Base Image	Python 3.9-slim	Python 3.11-slim (multi-stage)	**Critical**
Dockerfile Issues	apt-get runs as non-root user	Multi-stage build with proper ordering	**Critical**
Health Endpoints	`/health` (not implemented)	`/healthz`, `/readyz` endpoints	**High**
Secrets	Env var substitution only	K8s Secrets + External Secret Manager	**High**
Container Registry	None configured	GitHub Container Registry (GHCR)	**High**
WebSocket Support	Basic	Proper Ingress annotations for WebSocket	**Medium**
Autoscaling	None	HPA with conservative scale-down	**Medium**

Current State Analysis

2.1 Application Overview

Juniper Canopy is a real-time monitoring and diagnostic frontend for the Cascade Correlation Neural Network (CasCor) prototype.

- **Technology Stack:**
  - FastAPI (ASGI web framework)
  - Dash (Plotly dashboard framework)

  - WebSockets (real-time communication)
  - Redis (caching layer)
  - Python 3.11+ requirement

- **Key Features:**

  - Real-time training metrics visualization
  - Network topology visualization
  - Decision boundary plotting
  - WebSocket-based live updates

2.2 Current Docker Configuration

Location: `conf/Dockerfile`

Issues Identified:

Issue	Description	Impact	Fix Required
Python Version Mismatch	Uses `python:3.9-slim` but app requires `>=3.11`	**Critical** - App will fail	Update to `python:3.11-slim`
apt-get as non-root	`USER juniper` set before `apt-get install`	**Critical** - Build fails	Move USER directive after apt-get
No multi-stage build	Single stage includes build tools in final image	Medium - Larger image, security risk	Implement multi-stage
Health check endpoint	References `/health` which doesn't exist	High - K8s probes will fail	Implement `/healthz`, `/readyz`
No .dockerignore	All files copied including unnecessary ones	Low - Slower builds	Create .dockerignore

2.3 Current CI/CD Configuration

Location: `.github/workflows/ci.yml`

Strengths:

- Multi-Python version testing (3.14 configured)
- Comprehensive test suite with markers
- Code quality checks (Black, isort, Flake8, MyPy)
- Coverage reporting with Codecov
- Quality gates and notifications

Gaps:

- No Docker image building
- No container registry push
- No Kubernetes deployment stages
- No security scanning (Trivy, Snyk)
- No artifact versioning/tagging

2.4 Identified Secrets

Secret	Current Handling	Location	Risk Level
`JWT_SECRET_KEY`	`${JWT_SECRET_KEY}` substitution	app_config.yaml:334	High
`LOG_API_KEY`	`${LOG_API_KEY}` substitution	app_config.yaml:132	Medium
`REDIS_URL`	Hardcoded with optional env override	app_config.yaml:283	Medium
`CASCOR_BACKEND_URL`	Environment variable	docker-compose.yaml:15	Low
`CODECOV_TOKEN`	GitHub Secret	CI workflow	Low
`CASSANDRA_PASSWORD`	Documentation only (future)	CASSANDRA docs	High (future)

Target Architecture

3.1 Kubernetes Architecture Diagram

```
┌─────────────────────────────────────────────────────────────────────────┐
│                           KUBERNETES CLUSTER                            │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  ┌──────────────────────────────────────────────────────────────────┐   │
│  │                    INGRESS CONTROLLER (nginx)                    │   │
│  │  • WebSocket support (proxy-read-timeout: 3600s)                 │   │
│  │  • TLS termination (cert-manager + Let's Encrypt)                │   │
│  │  • Path-based routing (/, /ws, /dashboard)                       │   │
│  └──────────────────────┬───────────────────────────────────────────┘   │
│                         │                                               │
│  ┌──────────────────────▼───────────────────────────────────────────┐   │
│  │                    juniper-canopy (Service)                      │   │
│  │  ClusterIP: 8050                                                 │   │
│  └──────────────────────┬───────────────────────────────────────────┘   │
│                         │                                               │
│  ┌──────────────────────▼───────────────────────────────────────────┐   │
│  │              juniper-canopy (Deployment)                         │   │
│  │                                                                  │   │
│  │  ┌─────────────────┐  ┌─────────────────┐  ┌─────────────────┐   │   │
│  │  │     Pod 1       │  │     Pod 2       │  │     Pod N       │   │   │
│  │  │  FastAPI+Dash   │  │  FastAPI+Dash   │  │  FastAPI+Dash   │   │   │
│  │  │  Port: 8050     │  │  Port: 8050     │  │  Port: 8050     │   │   │
│  │  └─────────────────┘  └─────────────────┘  └─────────────────┘   │   │
│  │                                                                  │   │
│  │  HPA: min=2, max=10, CPU target=60%                              │   │
│  │  PDB: minAvailable=2                                             │   │
│  └──────────────────────────────────────────────────────────────────┘   │
│                         │                                               │
│  ┌──────────────────────▼───────────────────────────────────────────┐   │
│  │                    redis (StatefulSet)                           │   │
│  │                                                                  │   │
│  │  ┌─────────────────┐                                             │   │
│  │  │   Redis Pod     │  PVC: redis-data                            │   │
│  │  │   Port: 6379    │  --appendonly yes                           │   │
│  │  └─────────────────┘                                             │   │
│  │                                                                  │   │
│  └──────────────────────────────────────────────────────────────────┘   │
│                                                                         │
│  ┌────────────────────────────┐  ┌────────────────────────────────┐     │
│  │     ConfigMap              │  │     Secret                     │     │
│  │  • LOG_LEVEL               │  │  • JWT_SECRET_KEY              │     │
│  │  • CASCOR_ENV              │  │  • REDIS_URL                   │     │
│  │  • WEBSOCKET_HEARTBEAT     │  │  • LOG_API_KEY                 │     │
│  └───────────────────────┘  └──────────────────────────┘    │
│                                                              │
└──────────────────────────────────────────────────────────────┘
```

3.2 Component Specifications

3.2.1 Juniper Canopy Deployment

Specification	Value	Rationale
Replicas (min)	2	High availability, zero single points of failure
Replicas (max)	10	Accommodate traffic spikes
CPU Request	250m	Baseline for scheduling
CPU Limit	1000m	Prevent runaway processes
Memory Request	512Mi	Dash visualizations need memory
Memory Limit	1Gi	Prevent OOM kills in cluster
Update Strategy	RollingUpdate	Zero-downtime deployments
maxSurge	1	Conservative resource usage
maxUnavailable	0	Strict zero-downtime

3.2.2 Redis StatefulSet

Specification	Value	Rationale
Replicas	1	Single-node for simplicity; upgrade to cluster for HA
CPU Request	100m	Redis is lightweight
Memory Request	256Mi	Cache workload
Storage	10Gi PVC	Persistent cache data
Persistence	appendonly yes	Durability for cache state

Alternative: Use managed Redis (AWS ElastiCache, GCP Memorystore) for production.

3.2.3 Ingress Configuration (WebSocket-Aware)

```
# Key annotations for WebSocket support
annotations:
  nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
  nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
  nginx.ingress.kubernetes.io/websocket-services: "juniper-canopy"
  nginx.ingress.kubernetes.io/proxy-http-version: "1.1"
```

Kubernetes Deployment Procedure

4.1 Prerequisites

4.1.1 Required Tools

```
# Install 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/

# Install Helm 3
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# Install Docker
sudo apt-get update && sudo apt-get install -y docker.io
sudo usermod -aG docker $USER
```

4.1.2 Cluster Requirements

Requirement	Minimum	Recommended
Kubernetes Version	1.28+	1.30+

Requirement	Minimum	Recommended
Nodes	2	3+
Node CPU	2 cores	4 cores
Node Memory	4 GB	8 GB
Storage Class	Standard	SSD-backed

4.1.3 Cluster Setup Options

Option A: Local Development (minikube):

```
# Install minikube
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube

# Start cluster
minikube start --cpus=4 --memory=8192 --driver=docker

# Enable addons
minikube addons enable ingress
minikube addons enable metrics-server
```

Option B: Cloud Provider (AWS EKS):

```
# Install eksctl
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin

# Create cluster
eksctl create cluster \
  --name juniper-canopy \
  --region us-east-1 \
  --nodegroup-name standard-workers \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 5 \
  --managed
```

Option C: Cloud Provider (GCP GKE):

```
# Create cluster
gcloud container clusters create juniper-canopy \
  --zone us-central1-a \
  --num-nodes 3 \
  --machine-type e2-medium \
  --enable-autoscaling \
  --min-nodes 2 \
  --max-nodes 5
```

4.2 Namespace Setup

```
# Create namespaces for each environment
kubectl create namespace juniper-dev
kubectl create namespace juniper-staging
kubectl create namespace juniper-prod

# Set default namespace for development
kubectl config set-context --current --namespace=juniper-dev

# Apply resource quotas (production)
kubectl apply -f - <<EOF
apiVersion: v1
kind: ResourceQuota
metadata:
  name: juniper-quota
  namespace: juniper-prod
spec:
  hard:
    requests.cpu: "10"
    requests.memory: "20Gi"
    limits.cpu: "20"
    limits.memory: "40Gi"
    pods: "50"
    persistentvolumeclaims: "10"
EOF
```

4.3 Deploy Supporting Infrastructure

4.3.1 Install NGINX Ingress Controller

```
# Add ingress-nginx repository
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update

# Install ingress-nginx
helm install ingress-nginx ingress-nginx/ingress-nginx \
  --namespace ingress-nginx \
  --create-namespace \
  --set controller.config.proxy-read-timeout=3600 \
  --set controller.config.proxy-send-timeout=3600 \
  --set controller.config.upstream-keepalive-connections=32

# Verify installation
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=120s
```

4.3.2 Install cert-manager (TLS Certificates)

```
# Install cert-manager
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.14.0/cert-manager.yaml

# Wait for cert-manager
kubectl wait --for=condition=Available deployment --all -n cert-manager --timeout=120s

# Create Let's Encrypt ClusterIssuer
kubectl apply -f - <<EOF
apiVersion: cert-manager.io/v1
```

```
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: admin@example.com  # CHANGE THIS
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
EOF
```

4.3.3 Deploy Redis

```
# Option A: Helm chart (recommended for production)
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install redis bitnami/redis \
  --namespace juniper-prod \
  --set auth.enabled=true \
  --set auth.password="CHANGE_THIS_PASSWORD" \
  --set replica.replicaCount=0 \
  --set master.persistence.size=10Gi

# Option B: Simple StatefulSet (development)
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis
  namespace: juniper-dev
spec:
  serviceName: redis
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - name: redis
        image: redis:7-alpine
        ports:
        - containerPort: 6379
        command: ["redis-server", "--appendonly", "yes"]
        volumeMounts:
        - name: data
          mountPath: /data
        resources:
          requests:
            cpu: 100m
            memory: 256Mi
          limits:
            cpu: 500m
            memory: 512Mi
        livenessProbe:
          tcpSocket:
            port: 6379
          initialDelaySeconds: 10
          periodSeconds: 20
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 10Gi
---
apiVersion: v1
kind: Service
metadata:
  name: redis
  namespace: juniper-dev
spec:
  ports:
```

```
  - port: 6379
  selector:
    app: redis
EOF
```

4.4 Application Deployment

4.4.1 Create ConfigMap

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: juniper-canopy-config
  namespace: juniper-dev
data:
  CASCOR_ENV: "development"
  CASCOR_DEBUG: "true"
  CASCOR_DEMO_MODE: "1"
  LOG_LEVEL: "DEBUG"
  WEBSOCKET_HEARTBEAT_INTERVAL: "30"
  WEBSOCKET_MAX_CONNECTIONS: "50"
  PYTHONPATH: "/app"
EOF
```

4.4.2 Create Secret

```
# Create secret (use --from-literal or --from-env-file in production)
kubectl create secret generic juniper-canopy-secrets \
  --namespace=juniper-dev \
  --from-literal=JWT_SECRET_KEY='your-secure-jwt-secret-key-change-this' \
  --from-literal=LOG_API_KEY='your-log-api-key' \
  --from-literal=REDIS_URL='redis://redis:6379/0'

# Verify secret
kubectl get secret juniper-canopy-secrets -n juniper-dev -o yaml
```

4.4.3 Deploy Application

```
kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: juniper-canopy
  namespace: juniper-dev
  labels:
    app: juniper-canopy
    version: v1
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: juniper-canopy
  template:
    metadata:
      labels:
        app: juniper-canopy
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8050"
        prometheus.io/path: "/api/health"
    spec:
      terminationGracePeriodSeconds: 30
      securityContext:
        runAsNonRoot: true
        runAsUser: 1000
        fsGroup: 1000
      containers:
      - name: juniper-canopy
        image: ghcr.io/pcalnon/juniper-canopy:latest  # UPDATE THIS
        imagePullPolicy: Always
        ports:
        - name: http
```

```
          containerPort: 8050
          protocol: TCP
        envFrom:
        - configMapRef:
            name: juniper-canopy-config
        - secretRef:
            name: juniper-canopy-secrets
        env:
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        livenessProbe:
          httpGet:
            path: /healthz
            port: 8050
          initialDelaySeconds: 30
          periodSeconds: 30
          timeoutSeconds: 5
          failureThreshold: 3
        readinessProbe:
          httpGet:
            path: /readyz
            port: 8050
          initialDelaySeconds: 10
          periodSeconds: 10
          timeoutSeconds: 3
          failureThreshold: 3
        startupProbe:
          httpGet:
            path: /healthz
            port: 8050
          initialDelaySeconds: 0
          periodSeconds: 2
          failureThreshold: 30
        resources:
          requests:
            cpu: 250m
            memory: 512Mi
          limits:
            cpu: 1000m
            memory: 1Gi
        securityContext:
          allowPrivilegeEscalation: false
          readOnlyRootFilesystem: false
          capabilities:
            drop:
              - ALL
        volumeMounts:
        - name: logs
          mountPath: /app/logs
        - name: tmp
          mountPath: /tmp
        lifecycle:
          preStop:
            exec:
              command: ["/bin/sh", "-c", "sleep 10"]
      volumes:
      - name: logs
        emptyDir: {}
      - name: tmp
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: juniper-canopy
  namespace: juniper-dev
  labels:
    app: juniper-canopy
spec:
  type: ClusterIP
  ports:
  - name: http
    port: 8050
    targetPort: 8050
    protocol: TCP
  selector:
    app: juniper-canopy
EOF
```

4.4.4 Create Ingress

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: juniper-canopy
  namespace: juniper-dev
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    nginx.ingress.kubernetes.io/websocket-services: "juniper-canopy"
    nginx.ingress.kubernetes.io/proxy-http-version: "1.1"
    nginx.ingress.kubernetes.io/upstream-vhost: "juniper-canopy.juniper-dev.svc.cluster.local"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - juniper-dev.example.com  # CHANGE THIS
    secretName: juniper-canopy-tls
  rules:
  - host: juniper-dev.example.com  # CHANGE THIS
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: juniper-canopy
            port:
              number: 8050
EOF
```

4.4.5 Create HPA and PDB

```
# Horizontal Pod Autoscaler
kubectl apply -f - <<EOF
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: juniper-canopy-hpa
  namespace: juniper-dev
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: juniper-canopy
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 60
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 50
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 60
      policies:
      - type: Pods
        value: 2
        periodSeconds: 60
EOF

# Pod Disruption Budget
kubectl apply -f - <<EOF
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: juniper-canopy-pdb
  namespace: juniper-dev
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: juniper-canopy
EOF
```

4.5 Verification Steps

```
# 1. Check all resources
kubectl get all -n juniper-dev

# 2. Check pods are running
kubectl get pods -n juniper-dev -l app=juniper-canopy -w

# 3. Check pod logs
kubectl logs -n juniper-dev -l app=juniper-canopy --tail=100 -f

# 4. Test health endpoints (port-forward)
kubectl port-forward -n juniper-dev svc/juniper-canopy 8050:8050 &
curl http://localhost:8050/healthz
curl http://localhost:8050/readyz
curl http://localhost:8050/api/health

# 5. Test WebSocket connection
websocat ws://localhost:8050/ws/training

# 6. Check ingress
kubectl get ingress -n juniper-dev
curl -k https://juniper-dev.example.com/api/health

# 7. Check HPA status
kubectl get hpa -n juniper-dev

# 8. Describe deployment for events
kubectl describe deployment juniper-canopy -n juniper-dev
```

4.6 Troubleshooting Guide

Symptom	Possible Cause	Solution
Pod stuck in `Pending`	Insufficient resources	Check node resources: `kubectl describe nodes`
Pod in `CrashLoopBackOff`	Application error	Check logs: `kubectl logs <pod>`
Readiness probe failing	App not responding	Check `/readyz` implementation, increase initialDelaySeconds
WebSocket disconnects	Ingress timeout	Increase proxy-read-timeout annotation
502 Bad Gateway	Service not ready	Check pod readiness, service selector
OOMKilled	Memory limit too low	Increase memory limit

CI/CD Pipeline Design

5.1 Pipeline Architecture

```
┌─────────────────────────────────────────────────────────────────────────┐
│                        GitHub Actions CI/CD Pipeline                    │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  ┌──────────┐    ┌──────────┐    ┌──────────┐    ┌───────────┐          │
│  │  Code    │    │  Lint    │    │  Test    │    │  Build    │          │
│  │ Commit   │───▶│ & SAST   │───▶│ & Scan   │───▶│  Image    │          │
│  └──────────┘    └──────────┘    └──────────┘    └─────┬─────┘          │
│                                                        │                │
│                                   ┌────────────────────┼────────────┐   │
│                                   │                    ▼            │   │
│  ┌──────────────────────────────────────────────────────────────┐   │   │
│  │                    Security Scanning                         │   │   │
│  │  • Trivy (container vulnerabilities)                         │   │   │
│  │  • CodeQL (SAST)                                             │   │   │
│  │  • Gitleaks (secret scanning)                                │   │   │
│  │  • Snyk (dependency scanning)                                │   │   │
│  └──────────────────────────────────────────────────────────────┘   │   │
│                                                        ┌────────────┘   │
│                                                        │                │
│                                                        ▼                │
│  ┌──────────────────────────────────────────────────────────────────┐   │
│  │                     Container Registry (GHCR)                    │   │
│  │  ghcr.io/pcalnon/juniper-canopy:sha-abc1234                      │   │
│  │  ghcr.io/pcalnon/juniper-canopy:develop                          │   │
│  │  ghcr.io/pcalnon/juniper-canopy:v1.0.0                           │   │
│  └──────────────────────────────────────────────────────────────────┘   │
│                          │                                              │
│         ┌────────────────┼────────────────┐                             │
│         ▼                ▼                ▼                             │
│  ┌────────────┐   ┌────────────┐   ┌────────────┐                       │
│  │    DEV     │   │  STAGING   │   │    PROD    │                       │
│  │  Auto-     │   │  Manual    │   │  Manual +  │                       │
│  │  deploy    │   │  approval  │   │  Tag-based │                       │
│  └────────────┘   └────────────┘   └────────────┘                       │
│                                                                         │
└─────────────────────────────────────────────────────────────────────────┘
```

5.2 Recommended CI/CD Workflow

Proposed File: `.github/workflows/cicd.yml`

```
# This is the PROPOSED workflow - DO NOT implement without approval
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop, staging]
    tags: ['v*']
  pull_request:
    branches: [main, develop]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}/juniper-canopy
  PYTHON_VERSION: "3.11"

permissions:
  contents: read
  packages: write
  security-events: write
  id-token: write

jobs:
  # Stage 1: Code Quality
  lint:
    name: Code Quality
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: ${{ env.PYTHON_VERSION }}
          cache: 'pip'
      - name: Install linting tools
        run: pip install black isort flake8 mypy
      - name: Run Black
        run: black --check --diff src/
      - name: Run isort
        run: isort --check-only --diff src/
      - name: Run Flake8
        run: flake8 src/ --max-line-length=120 --exit-zero
      - name: Run MyPy
        run: mypy src/ --ignore-missing-imports

  # Stage 2: Security Scanning (SAST)
  security-scan:
    name: Security Scan
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - name: Initialize CodeQL
        uses: github/codeql-action/init@v2
        with:
          languages: python
      - name: CodeQL Analysis
        uses: github/codeql-action/analyze@v2
      - name: Gitleaks Secret Scan
        uses: gitleaks/gitleaks-action@v2

  # Stage 3: Test Suite
  test:
    name: Test Suite
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: ${{ env.PYTHON_VERSION }}
          cache: 'pip'
      - name: Install dependencies
        run: |
          pip install -r conf/requirements.txt
          pip install pytest pytest-cov pytest-asyncio
      - name: Create directories
        run: mkdir -p logs reports/junit
      - name: Run tests
        env:
          CASCOR_DEMO_MODE: "1"
        run: |
          cd src
          pytest tests/ \
            -v \
            -m "not requires_cascor and not requires_server and not slow" \
            --cov=. \
            --cov-report=xml:../coverage.xml \
            --cov-report=term-missing \
            --junit-xml=../reports/junit/results.xml
      - name: Upload coverage
        uses: codecov/codecov-action@v4
        with:
          file: ./coverage.xml
          token: ${{ secrets.CODECOV_TOKEN }}

  # Stage 4: Build Docker Image
  build:
    name: Build & Push Image
    runs-on: ubuntu-latest
    needs: [lint, security-scan, test]
    outputs:
      image: ${{ steps.build.outputs.imageid }}
      version: ${{ steps.meta.outputs.version }}
    steps:
      - uses: actions/checkout@v4
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      - name: Log in to GHCR
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
          tags: |
            type=ref,event=branch
            type=semver,pattern={{version}}
            type=semver,pattern={{major}}.{{minor}}
            type=sha,prefix=sha-
            type=raw,value=latest,enable={{is_default_branch}}
      - name: Build and push
        id: build
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./conf/Dockerfile
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max
          build-args: |
            BUILD_DATE=${{ github.event.head_commit.timestamp }}
            VCS_REF=${{ github.sha }}
            VERSION=${{ steps.meta.outputs.version }}
```

```
  # Stage 5: Container Security Scan
  container-scan:
    name: Container Security Scan
    runs-on: ubuntu-latest
    needs: [build]
    steps:
      - name: Run Trivy vulnerability scanner
        uses: aquasecurity/trivy-action@master
        with:
          image-ref: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:sha-${{ github.sha }}
          format: 'sarif'
          output: 'trivy-results.sarif'
          severity: 'CRITICAL,HIGH'
          exit-code: '0'  # Don't fail build, just report
      - name: Upload Trivy results
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: 'trivy-results.sarif'

  # Stage 6: Deploy to Development
  deploy-dev:
    name: Deploy to Development
    runs-on: ubuntu-latest
    needs: [build, container-scan]
    if: github.ref == 'refs/heads/develop'
    environment:
      name: development
      url: https://juniper-dev.example.com
    steps:
      - uses: actions/checkout@v4
      - name: Deploy to dev namespace
        run: |
          echo "Deploying ${{ needs.build.outputs.version }} to development"
          # kubectl set image deployment/juniper-canopy \
          #   juniper-canopy=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:sha-${{ github.sha }} \
          #   -n juniper-dev

  # Stage 7: Deploy to Staging (requires approval)
  deploy-staging:
    name: Deploy to Staging
    runs-on: ubuntu-latest
    needs: [build, container-scan]
    if: github.ref == 'refs/heads/staging'
    environment:
      name: staging
      url: https://juniper-staging.example.com
    steps:
      - uses: actions/checkout@v4
      - name: Deploy to staging namespace
        run: |
          echo "Deploying ${{ needs.build.outputs.version }} to staging"
          # kubectl set image deployment/juniper-canopy \
          #   juniper-canopy=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:sha-${{ github.sha }} \
          #   -n juniper-staging

  # Stage 8: Deploy to Production (requires tag + approval)
  deploy-prod:
    name: Deploy to Production
    runs-on: ubuntu-latest
    needs: [build, container-scan]
    if: startsWith(github.ref, 'refs/tags/v')
    environment:
      name: production
      url: https://juniper.example.com
    steps:
      - uses: actions/checkout@v4
      - name: Deploy to production namespace
        run: |
          echo "Deploying ${{ needs.build.outputs.version }} to production"
          # kubectl set image deployment/juniper-canopy \
          #   juniper-canopy=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}:${{ github.ref_name }} \
          #   -n juniper-prod
```

5.3 Environment Promotion Strategy

Environment	Branch/ Tag	Deployment	Approval	URL Pattern

Environment	Branch/ Tag	Deployment	Approval	URL Pattern
Development	`develop`	Automatic	None	`juniper-dev.example.com`
Staging	`staging`	Automatic	Environment protection	`juniper-staging.example.com`
Production	`v*` tags	Automatic	Environment protection + reviewers	`juniper.example.com`

5.4 Rollback Procedure

```
# Option 1: kubectl rollback
kubectl rollout undo deployment/juniper-canopy -n juniper-prod

# Option 2: Deploy specific version
kubectl set image deployment/juniper-canopy \
  juniper-canopy=ghcr.io/pcalnon/juniper-canopy:v1.0.0 \
  -n juniper-prod

# Option 3: View rollout history
kubectl rollout history deployment/juniper-canopy -n juniper-prod

# Rollback to specific revision
kubectl rollout undo deployment/juniper-canopy --to-revision=2 -n juniper-prod
```

Secrets Management Strategy

6.1 Secrets Classification

Category	Secrets	Storage	Rotation Frequency
Application Secrets	JWT_SECRET_KEY, LOG_API_KEY	K8s Secret	90 days

Category	Secrets	Storage	Rotation Frequency
Infrastructure Secrets	REDIS_URL, CASSANDRA_PASSWORD	K8s Secret	90 days
CI/CD Secrets	CODECOV_TOKEN, GHCR credentials	GitHub Secrets	1 year
Cloud Provider	AWS/GCP credentials	OIDC (no secrets)	N/A

6.2 Implementation Approach

Phase 1: Kubernetes Secrets (Immediate)

```
# Production secrets structure
apiVersion: v1
kind: Secret
metadata:
  name: juniper-canopy-secrets
  namespace: juniper-prod
  annotations:
    # Track secret versions
    secrets.juniper.io/version: "1"
    secrets.juniper.io/last-rotated: "2025-12-30"
type: Opaque
stringData:
  JWT_SECRET_KEY: "${GENERATED_SECRET}"  # 256-bit minimum
  LOG_API_KEY: "${GENERATED_KEY}"
  REDIS_URL: "redis://:${REDIS_PASSWORD}@redis:6379/0"
  CASCOR_BACKEND_URL: "http://cascor-backend:8000"
```

Secret Generation Script:

```
#!/bin/bash
# scripts/generate-secrets.sh

# Generate cryptographically secure secrets
```

```
JWT_SECRET=$(openssl rand -base64 32)
LOG_API_KEY=$(openssl rand -hex 16)
REDIS_PASSWORD=$(openssl rand -base64 24)

# Create secret
kubectl create secret generic juniper-canopy-secrets \
  --namespace=juniper-prod \
  --from-literal=JWT_SECRET_KEY="$JWT_SECRET" \
  --from-literal=LOG_API_KEY="$LOG_API_KEY" \
  --from-literal=REDIS_URL="redis://:$REDIS_PASSWORD@redis:6379/0" \
  --dry-run=client -o yaml | kubectl apply -f -

echo "Secrets generated and applied successfully"
```

Phase 2: External Secrets Operator (Future)

```
# Install External Secrets Operator
helm repo add external-secrets https://charts.external-secrets.io
helm install external-secrets external-secrets/external-secrets \
  --namespace external-secrets \
  --create-namespace

# Example: AWS Secrets Manager integration
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: aws-secrets-manager
  namespace: juniper-prod
spec:
  provider:
    aws:
      service: SecretsManager
      region: us-east-1
      auth:
        jwt:
          serviceAccountRef:
            name: external-secrets

---
```

```
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: juniper-canopy-secrets
  namespace: juniper-prod
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: aws-secrets-manager
    kind: SecretStore
  target:
    name: juniper-canopy-secrets
  data:
  - secretKey: JWT_SECRET_KEY
    remoteRef:
      key: juniper-canopy/prod
      property: jwt_secret_key
  - secretKey: REDIS_URL
    remoteRef:
      key: juniper-canopy/prod
      property: redis_url
```

6.3 Secret Rotation Procedure

```
#!/bin/bash
# scripts/rotate-secrets.sh

NAMESPACE="${1:-juniper-prod}"
SECRET_NAME="juniper-canopy-secrets"

echo "Rotating secrets for $SECRET_NAME in $NAMESPACE"

# 1. Generate new secrets
NEW_JWT_SECRET=$(openssl rand -base64 32)
NEW_LOG_API_KEY=$(openssl rand -hex 16)

# 2. Backup current secret
kubectl get secret $SECRET_NAME -n $NAMESPACE -o yaml > /tmp/secret-backup.yaml
echo "Backup saved to /tmp/secret-backup.yaml"
```

```
# 3. Patch secret with new values
kubectl patch secret $SECRET_NAME -n $NAMESPACE -p "{
  \"stringData\": {
    \"JWT_SECRET_KEY\": \"$NEW_JWT_SECRET\",
    \"LOG_API_KEY\": \"$NEW_LOG_API_KEY\"
  }
}"

# 4. Trigger rolling restart
kubectl rollout restart deployment/juniper-canopy -n $NAMESPACE

# 5. Wait for rollout
kubectl rollout status deployment/juniper-canopy -n $NAMESPACE --timeout=300s

# 6. Verify health
kubectl exec -n $NAMESPACE deployment/juniper-canopy -- curl -s localhost:8050/readyz

echo "Secret rotation complete"
```

6.4 GitHub Secrets Configuration

Secret Name	Purpose	Where Used
`CODECOV_TOKEN`	Coverage upload	Test job
`GHCR_TOKEN`	Container registry (auto)	`GITHUB_TOKEN`
`KUBE_CONFIG`	Kubernetes deployment	Deploy jobs

Build Artifact Storage

7.1 Container Registry Strategy

Primary Registry: GitHub Container Registry (GHCR)

Aspect	Configuration
Registry URL	`ghcr.io/pcalnon/juniper-canopy`
Authentication	`GITHUB_TOKEN` (automatic)
Visibility	Private (organization default)

Aspect	Configuration
Retention	Latest + last 5 versions per branch

Image Tagging Convention:

Tag Pattern	Example	Purpose
`sha-<short-sha>`	`sha-abc1234`	Immutable commit reference
`<branch>`	`develop`, `main`	Latest from branch
`v<semver>`	`v1.0.0`, `v1.2.3`	Release versions
`latest`	`latest`	Latest from main branch

7.2 Build Caching Strategy

```
# In docker/build-push-action
- name: Build and push
  uses: docker/build-push-action@v5
  with:
    context: .
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    # GitHub Actions cache for layers
    cache-from: type=gha
    cache-to: type=gha,mode=max
```

7.3 Artifact Retention

```
# Test artifacts (coverage, reports)
- name: Upload test artifacts
  uses: actions/upload-artifact@v4
  with:
    name: test-results
    path: |
      coverage.xml
      reports/
    retention-days: 30
```

```
# Container images (registry-side)
# Configure in GHCR settings: keep 5 versions per tag pattern
```

Implementation Phases

Phase 1: Foundation (Week 1-2)

Priority: Critical:

Task	Owner	Status	Deliverable
Fix Dockerfile issues	DevOps	Not Started	Updated multi-stage Dockerfile
Implement health endpoints	Dev	Not Started	`/healthz`, `/readyz` in main.py
Create .dockerignore	DevOps	Not Started	.dockerignore file
Set up GHCR	DevOps	Not Started	Registry access configured
Basic K8s manifests	DevOps	Not Started	Deployment, Service, ConfigMap, Secret

Phase 2: CI/CD Pipeline (Week 2-3)

Priority: High:

Task	Owner	Status	Deliverable
Update ci.yml for Docker build	DevOps	Not Started	Build and push stages
Add Trivy scanning	DevOps	Not Started	Container security scanning
Environment configuration	DevOps	Not Started	GitHub environments (dev/staging/prod)

Task	Owner	Status	Deliverable
Deployment automation	DevOps	Not Started	Deploy stages in workflow

Phase 3: Kubernetes Infrastructure (Week 3-4)

Priority: High:

Task	Owner	Status	Deliverable
Set up K8s cluster	Platform	Not Started	EKS/GKE cluster
Install ingress-nginx	Platform	Not Started	Ingress controller
Install cert-manager	Platform	Not Started	TLS automation
Deploy Redis	Platform	Not Started	Cache layer
Network policies	Platform	Not Started	Security isolation

Phase 4: Production Readiness (Week 4-5)

Priority: Medium:

Task	Owner	Status	Deliverable
HPA configuration	DevOps	Not Started	Autoscaling
PDB configuration	DevOps	Not Started	Disruption budget
Monitoring setup	Platform	Not Started	Prometheus/Grafana
Log aggregation	Platform	Not Started	Loki/ELK
Secret rotation automation	DevOps	Not Started	Rotation scripts

Phase 5: Advanced Features (Week 5+)

Priority: Low:

Task	Owner	Status	Deliverable
GitOps (ArgoCD)	Platform	Not Started	GitOps deployment
External Secrets Operator	Platform	Not Started	Vault integration

Task	Owner	Status	Deliverable
Service mesh evaluation	Platform	Not Started	Istio/Linkerd assessment
Multi-cluster strategy	Platform	Not Started	DR/HA architecture

---

Risk Assessment

Technical Risks

Risk	Probability	Impact	Mitigation
WebSocket connection drops during scaling	Medium	High	Conservative scale-down policy, session affinity
Database connection pool exhaustion	Low	High	Connection pooling, proper limits
Secret exposure in logs/errors	Low	Critical	Secret masking, log sanitization
Container image vulnerabilities	Medium	Medium	Regular scanning, base image updates
Ingress timeout misconfiguration	Medium	Medium	Thorough testing, monitoring

Operational Risks

Risk	Probability	Impact	Mitigation
Deployment rollback needed	Medium	Medium	Maintain 5 revisions, test rollback procedure
Certificate expiration	Low	High	cert-manager automation, monitoring
Resource exhaustion	Medium	Medium	Resource quotas, monitoring alerts
Secret rotation failures	Low	High	Automated testing, backup procedure

---

Appendices

A. Proposed Dockerfile (Multi-Stage)

```
# ============================================================
# Juniper Canopy - Production Dockerfile
# Multi-stage build for optimized image size and security
# ============================================================

# -------------------- Builder Stage --------------------
FROM python:3.11-slim-bookworm AS builder

WORKDIR /app

# Install build dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    g++ \
    make \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Copy requirements first for layer caching
COPY conf/requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir --upgrade pip && \
    pip wheel --no-cache-dir --wheel-dir /app/wheels -r requirements.txt

# -------------------- Runtime Stage --------------------
FROM python:3.11-slim-bookworm AS runtime

# Build arguments for versioning
ARG BUILD_DATE
ARG VCS_REF
ARG VERSION

LABEL org.opencontainers.image.title="Juniper Canopy" \
      org.opencontainers.image.description="Real-time monitoring dashboard for CasCor neural networks" \
      org.opencontainers.image.version="${VERSION}" \
      org.opencontainers.image.created="${BUILD_DATE}" \
      org.opencontainers.image.revision="${VCS_REF}" \
      org.opencontainers.image.vendor="Paul Calnon" \
      org.opencontainers.image.licenses="MIT"

# Create non-root user
RUN groupadd -r juniper && \
    useradd -r -g juniper -d /home/juniper -s /bin/bash juniper

WORKDIR /app

# Environment variables
ENV PYTHONPATH=/app \
    PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    CASCOR_ENV=production \
    CASCOR_DEBUG=false

# Copy wheels from builder and install
COPY --from=builder /app/wheels /wheels
COPY --from=builder /app/requirements.txt .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir /wheels/* && \
    rm -rf /wheels

# Copy application code
COPY --chown=juniper:juniper src/ ./src/
COPY --chown=juniper:juniper conf/app_config.yaml ./conf/
COPY --chown=juniper:juniper conf/logging_config.yaml ./conf/

# Create required directories
RUN mkdir -p logs data images && \
    chown -R juniper:juniper logs data images

# Switch to non-root user
USER juniper

# Expose port
EXPOSE 8050

# Health check (for Docker, not K8s)
```

```
HEALTHCHECK --interval=30s --timeout=10s --start-period=30s --retries=3 \
    CMD curl -f http://localhost:8050/healthz || exit 1

# Run application
CMD ["uvicorn", "src.main:app", "--host", "0.0.0.0", "--port", "8050"]
```

B. Proposed .dockerignore

```
# Git
.git
.gitignore

# Python
__pycache__
*.py[cod]
*$py.class
*.so
.Python
.eggs
*.egg-info
*.egg
.mypy_cache
.pytest_cache
.coverage
htmlcov
.tox

# Virtual environments
venv
.venv
ENV
env

# IDE
.idea
.vscode
*.swp
*.swo

# Build artifacts
```

```
dist
build
*.wheel

# Documentation
docs
*.md
!README.md

# Test data
tests
src/tests

# Logs and reports
logs
reports
*.log

# Development files
.pre-commit-config.yaml
.trunk
.github

# Local configuration
*.local
.env
.env.*

# Images and data (mount as volumes)
images
data

# Misc
*.bak
*.tmp
Makefile
```

C. Proposed Health Endpoints

```
# Add to src/main.py
```

```
@app.get("/healthz", tags=["health"])
async def healthz():
    """
    Kubernetes liveness probe.
    Returns 200 if the application process is alive.
    """
    return {"status": "ok"}


@app.get("/readyz", tags=["health"])
async def readyz():
    """
    Kubernetes readiness probe.
    Returns 200 only if the application is ready to serve traffic.
    Checks dependent services (Redis, etc.) if configured.
    """
    checks = {"app": "ok"}
    status_code = 200

    # Check Redis connectivity if configured
    try:
        # Optional: Add Redis health check when cache is implemented
        # from backend.cache import get_redis_client
        # redis_client = get_redis_client()
        # redis_client.ping()
        checks["redis"] = "ok"
    except Exception as e:
        checks["redis"] = f"error: {type(e).__name__}"
        status_code = 503

    # Check demo mode or backend availability
    global demo_mode_active, demo_mode_instance
    if demo_mode_active:
        checks["mode"] = "demo"
    elif cascor_integration:
        checks["mode"] = "cascor_backend"
    else:
        checks["mode"] = "no_backend"
        status_code = 503

    return JSONResponse(checks, status_code=status_code)
```

D. Helm Chart Structure (Future)

```
juniper-canopy-helm/
├── Chart.yaml
├── values.yaml
├── values-dev.yaml
├── values-staging.yaml
├── values-prod.yaml
├── templates/
│   ├── _helpers.tpl
│   ├── deployment.yaml
│   ├── service.yaml
│   ├── ingress.yaml
│   ├── configmap.yaml
│   ├── secret.yaml
│   ├── hpa.yaml
│   ├── pdb.yaml
│   ├── networkpolicy.yaml
│   ├── serviceaccount.yaml
│   └── tests/
│       └── test-connection.yaml
├── charts/
│   └── redis/          # Subchart dependency
└── README.md
```

Approval and Sign-Off

This document is a **planning document only**. No changes should be made to the codebase without explicit approval.

Required Approvals:

Role	Name	Date	Signature
Project Owner			

Role	Name	Date	Signature
Lead Developer			
DevOps Engineer			
Security Review			

Document History

Version	Date	Author	Changes
1.0.0	2025-12-30	AI Architect	Initial planning document