Level 2: Intermediate Docker and Kubernetes Workshop – Scaling and Deploying Your Web App

Hello class! Welcome back to Level 2 of our Docker and Kubernetes series. In Level 1, we built a simple "Hello World" web app using Python Flask (generated via LLM), containerized it with Docker, and deployed it locally with Minikube. Now, we'll take that same project, enhance it with more features, and dive into intermediate Kubernetes concepts like scalability, autoscaling, and cloud deployment. We'll use free or open-source tools so you can try this at home or in class without costs.

This workshop builds directly on Level 1—keep your `hello-app` folder handy. We'll add endpoints for time and metrics, simulate delays to mimic real loads, and deploy to a lightweight local cluster (K3d) or free cloud platforms (Render.com or Railway.app, which have free tiers as of September 2025). These platforms let you expose your app on the internet quickly, making it feel "real."

Why Level 2? To show how Kubernetes shines in handling growth—like scaling from 1 user to 100 without crashing. We'll use analogies: Think of Kubernetes as a restaurant manager scaling waiters (pods) during rush hour.

Prerequisites:

- Level 1 completed (or basic Docker/K8s knowledge).
- Python 3.8+, Docker Desktop, kubectl (Kubernetes CLI).
- Free accounts: Docker Hub (or GitHub), Render.com, Railway.app.
- Install K3d: `curl -s https://raw.githubusercontent.com/k3d-io/k3d/main/install.s | bash` (free, open-source).
- Time: 120–150 minutes, with hands-on breaks.

Key Concepts Recap/Advance:

- **Deployment:** A blueprint for creating pods (running containers).
- **ReplicaSet:** Ensures the right number of pods are running (part of Deployment).
- **Service:** A stable "front door" for pods.
- **Ingress:** Routes external traffic to Services (like a traffic cop).
- **HPA (Horizontal Pod Autoscaler):** Automatically adds/removes pods based on load (e.g., CPU > 50%).
- **Probes:** Health checks to ensure pods are ready/alive.

Let's enhance our project!

1 How to Generate the Enhanced Dummy Project Using LLM

We'll use an LLM again to generate the updated code—it's quick and lets us focus on deployment. Input this exact prompt into an LLM (like Grok or ChatGPT):

> Act as a code generator and create an enhanced web application using Python Flask. The app should:
>
> 1. Serve a 'Hello, World!' message at the root endpoint (/).
> 2. Provide another endpoint (/time) that returns the current server time in JSON format.
> 3. Include an endpoint (/metrics) that exposes simple metrics like request count.
> 4. Simulate latency by adding optional random delays to requests.
> 5. Include proper logging and comments explaining each part.
> 6. Be structured in a way that is easy for beginners to follow.
> 7. Include a requirements.txt or package.json with dependencies.
> 8. Provide instructions on how to run it locally, containerize it with Docker, and push the image to Docker Hub or another free container registry.
>
> Format the folder structure at the top and include comments in the code.

Expected Output: The LLM will give you `app.py` and `requirements.txt`. Here's a sample (generate it live in class!):

Folder Structure:

```
hello-app/
├── app.py
├── requirements.txt
```

1.1 app.py

```python
# Import necessary modules: Flask for web app, datetime for time, random for latency simulation, logging for logs.
from flask import Flask, jsonify
import datetime
import random
import time
import logging

# Set up logging: This helps debug by printing info to console.
logging.basicConfig(level=logging.INFO)

# Create the app instance.
app = Flask(__name__)

# Global counter for requests: Simple metric to track total requests.
request_count = 0

# Function to simulate latency: Adds a random delay (0-2 seconds) to mimic real-world slowdowns.
def simulate_latency():
```

```
    delay = random.uniform(0, 2) # Random delay in seconds.
    time.sleep(delay)
    logging.info(f"Simulated latency: {delay} seconds")

# Root endpoint: Serves a simple message.
@app.route('/')
def hello_world():
    global request_count
    request_count += 1
    simulate_latency() # Add delay for realism.
    logging.info("Handled / request")
    return 'Hello World from our enhanced workshop!'

# /time endpoint: Returns current server time in JSON.
@app.route('/time')
def get_time():
    global request_count
    request_count += 1
    simulate_latency()
    logging.info("Handled /time request")
    current_time = datetime.datetime.now().isoformat()
    return jsonify({'time': current_time})

# /metrics endpoint: Exposes basic stats like request count.
@app.route('/metrics')
def metrics():
    global request_count
    logging.info("Handled /metrics request")
    return jsonify({'request_count': request_count})

# Run the app: Starts the server.
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

1.2 requirements.txt

```
Flask==3.0.3
```

Local Run Instructions (from LLM):

- `pip install -r requirements.txt`
- `python app.py`
- Visit `http://localhost:5000/`, `/time`, `/metrics`.

Hands-On: Generate and run it now. Hit `/metrics` after a few requests—see the count grow!

2 Containerizing the Updated Project

Why? Docker packages everything (code, deps, latency sim) into a portable image.

1. Update/Create `Dockerfile` in `hello-app`:

```
# Base image with Python.
FROM python:3.9-slim
```

```
3
4  # Working dir.
5  WORKDIR /app
6
7  # Copy and install deps.
8  COPY requirements.txt .
9  RUN pip install -r requirements.txt
10
11 # Copy code.
12 COPY app.py .
13
14 # Expose port.
15 EXPOSE 5000
16
17 # Run command.
18 CMD ["python", "app.py"]
```

2. Build: `docker build -t enhanced-app:latest .`
3. Run Locally: `docker run -p 5000:5000 enhanced-app:latest`
   - Test endpoints.
4. Push to Free Registry (e.g., Docker Hub):
   - Create free Docker Hub account.
   - `docker login`
   - `docker tag enhanced-app:latest yourusername/enhanced-app:latest`
   - `docker push yourusername/enhanced-app:latest`
   - Alternatives: GitHub Container Registry (ghcr.io—free with GitHub account) or Quay.io (free open-source).

Why Push? For Kubernetes to pull the image from anywhere.

3 Setting Up a Kubernetes Environment and Introducing Concepts

We'll use K3d (free, lightweight local K8s) for local testing, then Render.com (free tier for web services) for cloud exposure.

Install K3d: `k3d cluster create mycluster –agents 2` (creates a cluster with 2 worker nodes).

- Why K3d? It's faster than Minikube, free, and simulates multi-node scaling.

Kubernetes Concepts with Analogies:

- **Deployment & ReplicaSet:** Deployment is a recipe; ReplicaSet ensures X copies (replicas) of pods follow it. Why? If one "waiter" (pod) quits, it hires another.
- **HPA:** Like auto-hiring more waiters when the restaurant gets busy (high CPU).
- **Service:** A VIP entrance that routes customers to any available waiter.
- **Ingress:** The restaurant's signboard, directing street traffic inside.

- **Probes:** Health checks—liveness (is the pod alive?) restarts if sick; readiness (is it ready to serve?) waits before sending traffic.

Diagram: Traffic Flow (ASCII):

```
Internet/User --> LoadBalancer/Ingress --> Service --> Pods (Replicas)
                  (Routes external)       (Balances)   (Running Apps)
```

Autoscaling Diagram:

```
Low Load: 1 Pod
High Load (CPU > 50%): HPA --> Add Pods (up to max)
```

3.1 Create Enhanced Manifests

Update from Level 1.

3.1.1 deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: enhanced-deployment
spec:
  replicas: 2 # Start with 2 pods for redundancy.
  selector:
    matchLabels:
      app: enhanced-app
  template:
    metadata:
      labels:
        app: enhanced-app
    spec:
      containers:
      - name: enhanced-container
        image: yourusername/enhanced-app:latest # Your pushed image.
        ports:
        - containerPort: 5000
        livenessProbe: # Checks if alive.
          httpGet:
            path: / # Hit root to check.
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 10
        readinessProbe: # Checks if ready.
          httpGet:
            path: /metrics # Use metrics as it's quick.
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 5
        resources: # For HPA.
          requests:
            cpu: "100m" # Min CPU.
          limits:
            cpu: "500m" # Max.
```

Why Probes? Prevents sending traffic to unhealthy pods.

3.1.2 service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: enhanced-service
spec:
  selector:
    app: enhanced-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: LoadBalancer # Exposes externally.
```

3.1.3 hpa.yaml (for Autoscaling)

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: enhanced-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: enhanced-deployment
  minReplicas: 1
  maxReplicas: 5 # Scale up to 5.
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50 # Scale if CPU > 50%.
```

3.2 Apply and Test Locally with K3d

1. `kubectl apply -f deployment.yaml -f service.yaml -f hpa.yaml`
2. Get URL: `kubectl get svc enhanced-service` (use external IP/port).
3. Test: Browser to the URL—hit endpoints, see load balancing (refresh to hit different pods).

For Ingress (Optional): Install Ingress controller in K3d: `kubectl apply -f https://raw.github` (then add ingress.yaml for domain routing).

4 Cloud Deployment with Free Services

Why Cloud? To see real internet access and scaling without local limits.

4.1 Option 1: Render.com (Free Tier: 512MB RAM services, always-on)

1. Sign up (free).

2. New → Web Service → Docker → Paste your Docker Hub image URL.
3. Set env vars (if needed, e.g., for secrets).
4. Deploy—Render handles building/pulling.
5. Access: Gets a free subdomain (e.g., yourapp.onrender.com).
6. For K8s-like: Render is PaaS, but add autoscaling in settings (free tier limited).

Configure LoadBalancer: Render auto-provides; no manual Ingress needed.

4.2 Option 2: Railway.app (Free: $1 credit/month for small apps)

1. Sign up.
2. New Project → Docker Image → Your image.
3. Deploy.
4. Access: Free subdomain.
5. Scaling: Set replicas in UI; limited autoscaling on free.

Hands-On: Deploy to one now—share your live URL in class!

5 Simulate Real-World Usage

Why? To see Kubernetes react to "traffic jams."

1. **Install Load Tool:** `go install github.com/rakyll/hey@latest` (free; or use `ab -n 1000 -c 10 http://your-url/`).
2. **Generate Load:** `hey -n 1000 -c 50 http://your-url/time` (1000 requests, 50 concurrent).
3. **Observe Scaling:** `kubectl get hpa –watch`—watch replicas increase.
4. **Troubleshoot:** If not scaling, check metrics: `kubectl top pods` (needs metrics-server: `kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/rel`

Analogy: Like cars on a highway—HPA adds lanes (pods) when jammed.

6 Best Practices and Security

- **Env Vars/ConfigMaps/Secrets:** Use ConfigMap for non-sensitive (e.g., app config): Create configmap.yaml with data, mount in deployment. Secrets for API keys: `kubectl create secret generic my-secret –from-literal=key=value`.
- **Why?** Keeps configs separate from code; secure (secrets encrypted).
- Production Tip: Use resource limits to prevent one pod hogging CPU.

7 Diagrams and Explanations

Traffic Flow:

```
User --> Internet --> Cloud LoadBalancer --> Ingress --> Service --> Pod1, Pod2.
```

Autoscaling:

- Monitor: CPU high? HPA signals Deployment to create more ReplicaSets/pods.
- Distributes: Service round-robins requests.

8 Exercises and Challenges

1. **Basic:** Deploy to Fly.io (usage-based free for tiny apps) instead—compare ease.
2. **Intermediate:** Simulate spikes: Run load tests with varying concurrency; graph CPU with `kubectl top`.
3. **Challenge:** Add Prometheus (free open-source): Install via Helm (`helm install prom prometheus-community/prometheus`), monitor metrics.
4. **Advanced:** Add a database (free Postgres on Render), connect via env vars/Secrets.

9 Troubleshooting and FAQs

- **Pods Not Scaling:** Ensure metrics-server installed; check `kubectl describe hpa` for errors.
- **Logs/Metrics:** `kubectl logs <pod-name>`; `kubectl top pods`.
- **Networking Issues:** On free services, check firewall/ports; for local, `kubectl port-forward svc/enhanced-service 5000:80`.
- Q: Why no free forever on Fly.io? A: Usage-based; monitor credits.
- Q: App not accessible? A: Verify Service type (LoadBalancer) and external IP.

Cleanup: `k3d cluster delete mycluster`; delete cloud deployments.

Awesome work! You've scaled a real app. Share your live links—next level: CI/CD. Questions?