🏠 Local Deployment Guide - Bespoke Labs Assignment

📖 Overview

This guide will help you deploy and test the Wiki Service application on your **local system** (not the DeepAgent environment). Whether you want to quickly test the FastAPI application or deploy the full Kubernetes stack, this guide has you covered.

🎯 Two Deployment Paths

Path 1: Quick Local Testing (Current State)

Use this to: Understand the application, test endpoints, and verify functionality before completing all phases.
- ✅ Run FastAPI with SQLite
- ✅ Test API endpoints
- ✅ View Prometheus metrics
- ⏱️ **Time**: 5-10 minutes

Path 2: Full Kubernetes Deployment (After All Phases)

Use this to: Deploy the complete production-ready stack after finishing all assignment phases.
- ✅ PostgreSQL database
- ✅ FastAPI service
- ✅ Prometheus monitoring
- ✅ Grafana dashboards
- ✅ Ingress routing
- ⏱️ **Time**: 30-45 minutes

📋 Prerequisites

For Quick Local Testing (Path 1)

```
# Required
- Python 3.13+ (check: python --version)
- Git (check: git --version)

# Optional but recommended
- Docker (check: docker --version)
```

For Full Kubernetes Deployment (Path 2)

```
# All of the above, plus:
- Docker Desktop or Docker Engine
- Minikube (check: minikube version)
- kubectl (check: kubectl version --client)
- Helm 3+ (check: helm version)
```

Installation Instructions

Install Python 3.13+

```
# macOS (using Homebrew)
brew install python@3.13

# Ubuntu/Debian
sudo apt update
sudo apt install python3.13 python3.13-venv python3-pip

# Windows
# Download from https://www.python.org/downloads/
```

Install UV (Python Package Manager)

```
# macOS/Linux
curl -LsSf https://astral.sh/uv/install.sh | sh

# Windows (PowerShell)
powershell -c "irm https://astral.sh/uv/install.ps1 | iex"

# Verify installation
uv --version
```

Install Docker Desktop

```
# macOS
brew install --cask docker

# Windows/Linux
# Download from https://www.docker.com/products/docker-desktop/

# Verify installation
docker --version
docker ps
```

Install Minikube

```
# macOS
brew install minikube

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

# Windows (PowerShell as Admin)
choco install minikube

# Verify installation
minikube version
```

Install kubectl

```
# macOS
brew install kubectl

# Linux
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/
stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# Windows (PowerShell as Admin)
choco install kubernetes-cli

# Verify installation
kubectl version --client
```

Install Helm

```
# macOS
brew install helm

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

# Windows (PowerShell as Admin)
choco install kubernetes-helm

# Verify installation
helm version
```

🚀 Path 1: Quick Local Testing

Step 1: Clone Your Repository

```
# Clone your GitHub repository
git clone https://github.com/YOUR_USERNAME/assignment-part1.git
cd assignment-part1
```

Step 2: Option A - Run with Python (Recommended for Development)

```
# Navigate to the service directory
cd wiki-service

# Create a virtual environment (optional but recommended)
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies using UV
uv sync

# OR install with pip
pip install -e .

# Run the application
python main.py
```

Expected Output:

```
INFO:     Started server process [12345]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

Step 3: Test the API

Open a new terminal and test the endpoints:

```
# Navigate to the wiki-service directory
cd assignment-part1/wiki-service

# Make the test script executable
chmod +x test_api.sh

# Run the test script
./test_api.sh
```

Or test manually with curl:

```
# 1. Check the root endpoint
curl http://localhost:8000/

# 2. Create a user
curl -X POST http://localhost:8000/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Alice"}'

# Expected response:
# {"id":1,"name":"Alice","created_time":"2026-01-31T15:30:00.123456"}

# 3. Create a post
curl -X POST http://localhost:8000/posts \
  -H "Content-Type: application/json" \
  -d '{"user_id": 1, "content": "Hello World!"}'

# Expected response:
# {"post_id":1,"content":"Hello World!","user_id":
1,"created_time":"2026-01-31T15:30:05.123456"}

# 4. Get user by ID
curl http://localhost:8000/user/1

# 5. Get post by ID
curl http://localhost:8000/posts/1

# 6. View Prometheus metrics
curl http://localhost:8000/metrics
```

Step 4: Access Interactive Documentation

Open your browser and visit:
- **Swagger UI**: http://localhost:8000/docs
- **ReDoc**: http://localhost:8000/redoc
- **Metrics**: http://localhost:8000/metrics

Step 5: Option B - Run with Docker (Simple Container)

If you want to test containerization before completing all phases:

```
# Navigate to wiki-service directory
cd assignment-part1/wiki-service

# Create a simple Dockerfile (if not exists)
cat > Dockerfile.dev << 'EOF'
FROM python:3.13-slim

WORKDIR /app

# Copy application files
COPY . .

# Install dependencies
RUN pip install --no-cache-dir -e .

# Expose port
EXPOSE 8000

# Run the application
CMD ["python", "main.py"]
EOF

# Build the Docker image
docker build -f Dockerfile.dev -t wiki-service:dev .

# Run the container
docker run -d -p 8000:8000 --name wiki-service wiki-service:dev

# Check logs
docker logs wiki-service

# Test the API (same curl commands as above)
curl http://localhost:8000/

# Stop and remove container when done
docker stop wiki-service
docker rm wiki-service
```

🎯 Path 2: Full Kubernetes Deployment

> **Note**: Complete this path **after** you've finished all assignment phases (PostgreSQL migration, Dockerfile, Helm charts, monitoring setup).

Step 1: Start Minikube

```
# Start Minikube with sufficient resources
minikube start --cpus=4 --memory=8192 --driver=docker

# Enable Ingress addon
minikube addons enable ingress

# Verify cluster is running
kubectl cluster-info
kubectl get nodes
```

Expected Output:

```
😄  minikube v1.32.0 on Darwin 14.0
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🔄  Restarting existing docker container for "minikube" ...
🐳  Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
🌟  Enabled addons: storage-provisioner, default-storageclass, ingress
🏄  Done! kubectl is now configured to use "minikube" cluster
```

Step 2: Build Docker Images

```
# Navigate to project root
cd assignment-part1

# Point Docker to Minikube's Docker daemon (important!)
eval $(minikube docker-env)

# Build the wiki-service image
cd wiki-service
docker build -t wiki-service:latest .

# Verify image is built
docker images | grep wiki-service

# Return to project root
cd ..
```

Troubleshooting: If you get “image not found” errors later, make sure you ran `eval $(minikube docker-env)` before building.

Step 3: Install Helm Chart

```
# Navigate to project root
cd assignment-part1

# Install the Helm chart
helm install wiki-release ./wiki-chart

# Check deployment status
kubectl get pods
kubectl get services
kubectl get ingress

# Wait for all pods to be ready (may take 1-2 minutes)
kubectl wait --for=condition=ready pod --all --timeout=300s
```

Expected Output:

```
NAME                                READY   STATUS    RESTARTS   AGE
wiki-service-deployment-xxx         1/1     Running   0          30s
postgresql-xxx                      1/1     Running   0          30s
prometheus-server-xxx               1/1     Running   0          30s
grafana-xxx                         1/1     Running   0          30s
```

Step 4: Access Services via Ingress

```
# Get Minikube IP
minikube ip

# Add entries to /etc/hosts (replace <MINIKUBE_IP> with actual IP)
echo "<MINIKUBE_IP> wiki.local" | sudo tee -a /etc/hosts
echo "<MINIKUBE_IP> grafana.local" | sudo tee -a /etc/hosts
echo "<MINIKUBE_IP> prometheus.local" | sudo tee -a /etc/hosts

# Example:
# 192.168.49.2 wiki.local
# 192.168.49.2 grafana.local
# 192.168.49.2 prometheus.local
```

On Windows, edit `C:\Windows\System32\drivers\etc\hosts` as Administrator.

Step 5: Test All Components

Test FastAPI Service

```
# Create a user
curl -X POST http://wiki.local/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Bob"}'

# Create a post
curl -X POST http://wiki.local/posts \
  -H "Content-Type: application/json" \
  -d '{"user_id": 1, "content": "Testing Kubernetes deployment!"}'

# Get user
curl http://wiki.local/user/1

# Get post
curl http://wiki.local/posts/1

# Check metrics
curl http://wiki.local/metrics
```

Access Grafana Dashboard

```
# Open in browser
open http://grafana.local  # macOS
xdg-open http://grafana.local  # Linux
start http://grafana.local  # Windows

# Default credentials (check your Helm values):
# Username: admin
# Password: (check with: kubectl get secret grafana-admin -o json-
path="{.data.password}" | base64 --decode)
```

Access Prometheus

```
# Open in browser
open http://prometheus.local  # macOS

# Check targets are being scraped
# Navigate to Status > Targets
```

Step 6: View Logs and Debug

```
# View wiki-service logs
kubectl logs -l app=wiki-service --tail=50 -f

# View PostgreSQL logs
kubectl logs -l app=postgresql --tail=50

# View Prometheus logs
kubectl logs -l app=prometheus --tail=50

# Describe a pod for detailed info
kubectl describe pod <pod-name>

# Execute commands inside a pod
kubectl exec -it <wiki-service-pod-name> -- /bin/sh

# Check database connection
kubectl exec -it <postgresql-pod-name> -- psql -U wikiuser -d wikidb -c "\dt"
```

Step 7: Port Forwarding (Alternative Access Method)

If Ingress isn't working, use port forwarding:

```
# Forward wiki-service
kubectl port-forward service/wiki-service 8000:8000 &

# Forward Grafana
kubectl port-forward service/grafana 3000:3000 &

# Forward Prometheus
kubectl port-forward service/prometheus-server 9090:9090 &

# Now access via localhost:
# - API: http://localhost:8000
# - Grafana: http://localhost:3000
# - Prometheus: http://localhost:9090
```

Step 8: Cleanup

```
# Uninstall Helm release
helm uninstall wiki-release

# Verify all resources are deleted
kubectl get all

# Stop Minikube (optional)
minikube stop

# Delete Minikube cluster (optional)
minikube delete
```

🔧 Troubleshooting Common Issues

Issue 1: Port Already in Use

Symptom:

```
Error: bind: address already in use
```

Solution:

```
# Find process using port 8000
lsof -i :8000  # macOS/Linux
netstat -ano | findstr :8000  # Windows

# Kill the process
kill -9 <PID>  # macOS/Linux
taskkill /PID <PID> /F  # Windows

# Or use a different port
uvicorn app.main:app --port 8001
```

Issue 2: Minikube Won’t Start

Symptom:

```
❌ Exiting due to PROVIDER_DOCKER_NOT_RUNNING
```

Solution:

```
# Make sure Docker Desktop is running
docker ps

# If Docker is running, try:
minikube delete
minikube start --driver=docker

# On macOS with Apple Silicon:
minikube start --driver=docker --container-runtime=containerd

# Check Minikube status
minikube status
```

Issue 3: Docker Build Fails

Symptom:

```
ERROR: failed to solve: failed to compute cache key
```

Solution:

```
# Clear Docker cache
docker builder prune -a

# Rebuild without cache
docker build --no-cache -t wiki-service:latest .

# Check Dockerfile syntax
docker build --progress=plain -t wiki-service:latest .
```

Issue 4: Pods Not Starting

Symptom:

```
kubectl get pods
NAME                    READY   STATUS             RESTARTS   AGE
wiki-service-xxx        0/1     ImagePullBackOff   0          2m
```

Solution:

```
# Check pod details
kubectl describe pod <pod-name>

# Common causes:
# 1. Image not found - make sure you ran: eval $(minikube docker-env)
# 2. Wrong image name in Helm values
# 3. Image pull policy - set to IfNotPresent or Never for local images

# Fix: Update values.yaml
image:
  repository: wiki-service
  tag: latest
  pullPolicy: Never  # Important for local images!
```

Issue 5: Ingress Not Working

Symptom:

```
curl: (7) Failed to connect to wiki.local port 80
```

Solution:

```
# 1. Check Ingress addon is enabled
minikube addons list | grep ingress

# 2. Enable if not enabled
minikube addons enable ingress

# 3. Wait for Ingress controller to be ready
kubectl get pods -n ingress-nginx

# 4. Check Ingress resource
kubectl get ingress
kubectl describe ingress wiki-ingress

# 5. Verify /etc/hosts entry
cat /etc/hosts | grep wiki.local

# 6. Use port-forward as alternative (see Step 7 above)
```

Issue 6: Database Connection Errors

Symptom:

```
sqlalchemy.exc.OperationalError: could not connect to server
```

Solution:

```
# 1. Check PostgreSQL pod is running
kubectl get pods -l app=postgresql

# 2. Check PostgreSQL logs
kubectl logs -l app=postgresql

# 3. Verify database credentials in ConfigMap/Secret
kubectl get configmap wiki-config -o yaml
kubectl get secret wiki-secret -o yaml

# 4. Test connection from wiki-service pod
kubectl exec -it <wiki-service-pod> -- sh
# Inside pod:
apk add postgresql-client
psql -h postgresql -U wikiuser -d wikidb
```

Issue 7: Metrics Not Showing in Prometheus

Symptom:

- Prometheus shows targets as “down”
- No metrics visible in Grafana

Solution:

```
# 1. Check Prometheus is scraping the service
kubectl port-forward service/prometheus-server 9090:9090
# Open http://localhost:9090/targets

# 2. Verify service has correct labels/annotations
kubectl get service wiki-service -o yaml

# 3. Check Prometheus configuration
kubectl get configmap prometheus-config -o yaml

# 4. Verify metrics endpoint is accessible
kubectl exec -it <wiki-service-pod> -- curl localhost:8000/metrics
```

Issue 8: UV/Python Dependency Issues

Symptom:

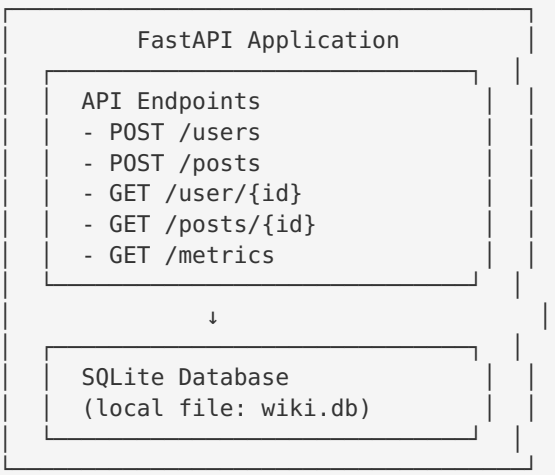
```
error: Failed to download distributions
```

Solution:

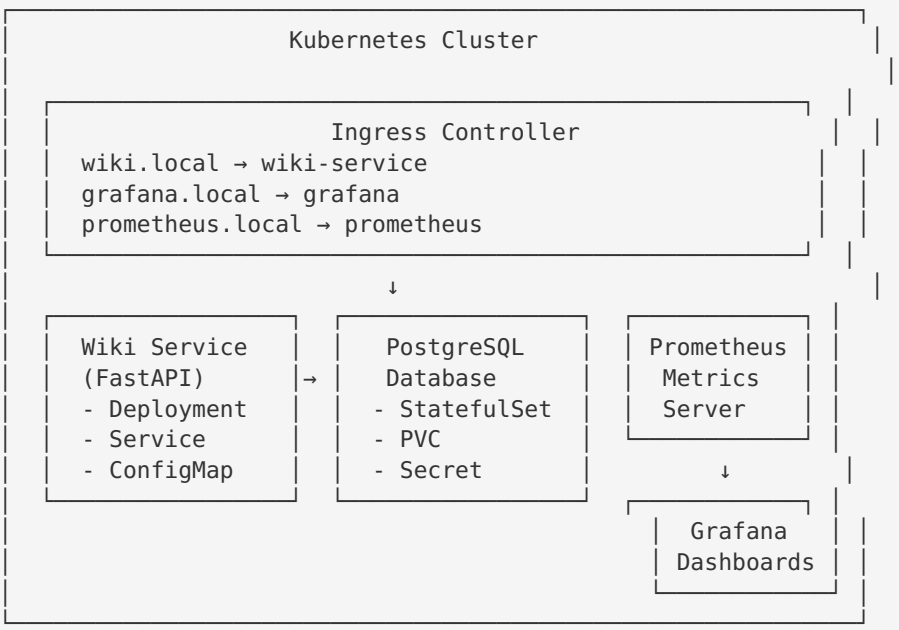
```
# 1. Update UV
curl -LsSf https://astral.sh/uv/install.sh | sh

# 2. Clear UV cache
uv cache clean

# 3. Use pip as fallback
pip install -e .

# 4. Check Python version
python --version  # Should be 3.13+
```

🎓 Understanding the Architecture

Current State (Phase 1)



Target State (After All Phases)



📚 Next Steps After Successful Deployment

1. Explore the Application

- [ ] Test all API endpoints with different data
- [ ] View metrics in Prometheus
- [ ] Create custom Grafana dashboards
- [ ] Monitor resource usage in Kubernetes

2. Experiment with Kubernetes

```
# Scale the deployment
kubectl scale deployment wiki-service --replicas=3

# Update the application
# (make code changes, rebuild image, upgrade Helm release)
helm upgrade wiki-release ./wiki-chart

# Rollback if needed
helm rollback wiki-release

# View revision history
helm history wiki-release
```

3. Learn More

- [ ] Read Kubernetes documentation on Deployments, Services, Ingress
- [ ] Explore Helm templating and values
- [ ] Study Prometheus query language (PromQL)
- [ ] Create custom Grafana dashboards

4. Enhance the Application

- [ ] Add authentication/authorization
- [ ] Implement caching (Redis)
- [ ] Add more API endpoints
- [ ] Create integration tests
- [ ] Set up CI/CD pipeline

5. Production Readiness Checklist

- [ ] Add health checks (liveness/readiness probes)
- [ ] Configure resource limits and requests
- [ ] Set up horizontal pod autoscaling
- [ ] Implement proper logging (ELK stack)
- [ ] Add backup strategy for PostgreSQL
- [ ] Configure TLS/SSL certificates
- [ ] Set up network policies
- [ ] Implement secrets management (Vault)

🆘 Getting Help

Useful Commands Reference

```
# Kubernetes
kubectl get all                            # List all resources
kubectl describe <resource> <name>         # Detailed info
kubectl logs <pod-name> -f                 # Stream logs
kubectl exec -it <pod-name> -- sh          # Shell into pod
kubectl port-forward <pod> 8000:8000       # Port forwarding

# Helm
helm list                                  # List releases
helm status <release-name>                 # Release status
helm get values <release-name>             # View values
helm upgrade <release> <chart>             # Update release
helm rollback <release> <revision>         # Rollback

# Minikube
minikube status                            # Cluster status
minikube dashboard                         # Open dashboard
minikube service list                      # List services
minikube logs                              # Cluster logs
minikube ssh                               # SSH into node

# Docker
docker ps                                  # List containers
docker logs <container-id>                 # View logs
docker exec -it <container> sh             # Shell into container
docker images                              # List images
```

Documentation Links

- FastAPI Docs (https://fastapi.tiangolo.com/)
- Kubernetes Docs (https://kubernetes.io/docs/)
- Helm Docs (https://helm.sh/docs/)
- Prometheus Docs (https://prometheus.io/docs/)
- Grafana Docs (https://grafana.com/docs/)
- Minikube Docs (https://minikube.sigs.k8s.io/docs/)

Common Questions

Q: Can I use Kind or K3s instead of Minikube?
A: Yes! The Helm chart should work with any Kubernetes cluster. Just adjust the Ingress configuration accordingly.

Q: Do I need to use UV, or can I use pip?
A: Either works! UV is faster and handles dependencies better, but pip is perfectly fine.

Q: Can I deploy this to a cloud provider (AWS, GCP, Azure)?
A: Yes! After completing all phases, you can deploy to any Kubernetes cluster. You'll need to:
- Push Docker images to a container registry (Docker Hub, ECR, GCR)
- Update Helm values with the registry URL
- Configure cloud-specific Ingress (ALB, GCE Ingress)

Q: How do I persist data between Minikube restarts?
A: Use PersistentVolumeClaims (PVCs) for PostgreSQL. The Helm chart should include this in later phases.

Q: Can I run multiple instances of the wiki-service?
A: Yes! Use `kubectl scale` or set `replicaCount` in Helm values.

✅ Verification Checklist

Path 1: Quick Local Testing

- [ ] Python 3.13+ installed
- [ ] Repository cloned
- [ ] Dependencies installed (uv sync or pip install)
- [ ] Application runs without errors
- [ ] Can create users via API
- [ ] Can create posts via API
- [ ] Can retrieve users and posts
- [ ] Metrics endpoint returns data
- [ ] Interactive docs accessible at /docs

Path 2: Full Kubernetes Deployment

- [ ] Docker installed and running
- [ ] Minikube installed and started
- [ ] kubectl configured for Minikube
- [ ] Helm installed
- [ ] Ingress addon enabled
- [ ] Docker images built in Minikube context
- [ ] Helm chart installed successfully
- [ ] All pods in Running state
- [ ] Services created and accessible
- [ ] Ingress configured with /etc/hosts entries
- [ ] Can create/retrieve users and posts via Ingress
- [ ] Prometheus scraping metrics
- [ ] Grafana accessible and showing data
- [ ] Can view logs from all pods

🎉 Success Criteria

You've successfully deployed the application when:

For Path 1 (Quick Testing):

✅ Application starts without errors
✅ All API endpoints respond correctly

✅ Metrics endpoint shows Prometheus data
✅ Interactive documentation loads

For Path 2 (Full Deployment):

✅ All Kubernetes pods are in Running state
✅ Services are accessible via Ingress or port-forward
✅ Can perform CRUD operations on users and posts
✅ Data persists in PostgreSQL
✅ Prometheus is collecting metrics
✅ Grafana displays application metrics
✅ No errors in pod logs

Good luck with your deployment! 🚀

Last Updated: January 31, 2026