Multi-tier App — Python (FastAPI) + React (Vite) — Starter Scaffold

Below is a **ready-to-copy repo scaffold** with minimal but working code for: - **Backend:** Python FastAPI - **Frontend:** React (Vite, plain JS) - **Dockerfiles** for both - **Kubernetes** (Helm umbrella chart + values for local) - **Makefile** for kind + ingress + deploy - **GitHub Actions** (build & push images)

> Replace `ghcr.io/YOUR_GH_USER/multi-tier-k8s` everywhere.

---

1) Repo structure

```
multi-tier-k8s/
├─ apps/
│  ├─ backend-fastapi/
│  │  ├─ app/
│  │  │  ├─ main.py
│  │  │  └─ db.py
│  │  ├─ pyproject.toml
│  │  └─ Dockerfile
│  └─ frontend/
│     ├─ index.html
│     ├─ package.json
│     ├─ vite.config.js
│     ├─ src/
│     │  ├─ main.jsx
│     │  └─ App.jsx
│     └─ Dockerfile
├─ deploy/
│  └─ helm/
│     └─ app/
│        ├─ Chart.yaml
│        ├─ values.yaml
│        └─ templates/
│           ├─ namespace.yaml
│           ├─ backend-deploy.yaml
│           ├─ backend-svc.yaml
│           ├─ frontend-deploy.yaml
│           ├─ frontend-svc.yaml
│           └─ ingress.yaml
├─ ops/
│  └─ scripts/
│     └─ kind-config.yaml
├─ .github/workflows/ci.yml
```

```
├─ Makefile
└─ README.md
```

---

2) Backend (FastAPI)

`apps/backend-fastapi/app/db.py`

```python
import os
import sqlite3
from contextlib import contextmanager

DB_PATH = os.getenv("DB_PATH", "/data/app.db")

os.makedirs(os.path.dirname(DB_PATH), exist_ok=True)
conn = sqlite3.connect(DB_PATH)
conn.execute("CREATE TABLE IF NOT EXISTS todos (id INTEGER PRIMARY KEY, title
TEXT, done INTEGER DEFAULT 0)")
conn.commit()
conn.close()

@contextmanager
def get_db():
    conn = sqlite3.connect(DB_PATH)
    try:
        yield conn
    finally:
        conn.close()
```

`apps/backend-fastapi/app/main.py`

```python
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
from .db import get_db

app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

class TodoIn(BaseModel):
    title: str
```

```
    done: bool = False

class TodoOut(TodoIn):
    id: int

@app.get("/healthz")
def healthz():
    return {"status": "ok"}

@app.get("/livez")
def livez():
    return {"status": "alive"}

@app.get("/api/todos")
def list_todos():
    with get_db() as db:
        cur = db.execute("SELECT id, title, done FROM todos ORDER BY id
DESC")
        rows = cur.fetchall()
        return [
            {"id": r[0], "title": r[1], "done": bool(r[2])}
            for r in rows
        ]

@app.post("/api/todos", response_model=TodoOut)
def create(todo: TodoIn):
    with get_db() as db:
        cur = db.execute(
            "INSERT INTO todos (title, done) VALUES (?, ?)",
            (todo.title, 1 if todo.done else 0),
        )
        db.commit()
        return {"id": cur.lastrowid, **todo.model_dump()}

@app.patch("/api/todos/{todo_id}")
def update(todo_id: int, todo: TodoIn):
    with get_db() as db:
        cur = db.execute(
            "UPDATE todos SET title=?, done=? WHERE id=?",
            (todo.title, 1 if todo.done else 0, todo_id),
        )
        db.commit()
        if cur.rowcount == 0:
            raise HTTPException(404, "Not found")
        return {"id": todo_id, **todo.model_dump()}

@app.delete("/api/todos/{todo_id}")
def delete(todo_id: int):
    with get_db() as db:
        cur = db.execute("DELETE FROM todos WHERE id=?", (todo_id,))
        db.commit()
```

```
        if cur.rowcount == 0:
            raise HTTPException(404, "Not found")
        return {"status": "deleted", "id": todo_id}
```

apps/backend-fastapi/pyproject.toml

```
[tool.poetry]
name = "backend-fastapi"
version = "0.1.0"
description = "K8s demo backend"
authors = ["YOU <you@example.com>"]

[tool.poetry.dependencies]
python = "^3.12"
fastapi = "^0.115.0"
uvicorn = {extras = ["standard"], version = "^0.30.0"}
pydantic = "^2.9.0"

[tool.poetry.group.dev.dependencies]
pytest = "^8.3.0"

[build-system]
requires = ["poetry-core>=1.0.0"]
build-backend = "poetry.core.masonry.api"
```

apps/backend-fastapi/Dockerfile

```
FROM python:3.12-slim
ENV PYTHONDONTWRITEBYTECODE=1 PYTHONUNBUFFERED=1
WORKDIR /app
RUN pip install --no-cache-dir poetry
COPY pyproject.toml ./
RUN poetry config virtualenvs.create false && poetry install --no-
interaction --no-ansi --no-root
COPY app/ ./app/
EXPOSE 8080
VOLUME ["/data"]
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

---

3) Frontend (React + Vite, JS)

apps/frontend/package.json

```
{
  "name": "frontend",
  "private": true,
```

```
  "version": "0.0.0",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview --port 5173"
  },
  "dependencies": {
    "react": "^18.3.1",
    "react-dom": "^18.3.1"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^4.3.1",
    "vite": "^5.4.0"
  }
}
```

apps/frontend/vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: { port: 5173 },
})
```

apps/frontend/index.html

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Todos</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

apps/frontend/src/main.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
createRoot(document.getElementById('root')).render(<App />)
```

apps/frontend/src/App.jsx

```
import React, { useEffect, useState } from 'react'

const API_BASE = import.meta.env.VITE_API_BASE || '' // same host via Ingress

export default function App() {
  const [todos, setTodos] = useState([])
  const [title, setTitle] = useState('')

  const load = async () => {
    const res = await fetch(`${API_BASE}/api/todos`)
    setTodos(await res.json())
  }

  const add = async () => {
    if (!title.trim()) return
    await fetch(`${API_BASE}/api/todos`, {
      method: 'POST', headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ title })
    })
    setTitle('')
    await load()
  }

  useEffect(() => { load() }, [])

  return (
    <div style={{ maxWidth: 600, margin: '3rem auto', fontFamily: 'sans-
serif' }}>
      <h1>Todos</h1>
      <div style={{ display: 'flex', gap: 8 }}>
        <input value={title} onChange={e => setTitle(e.target.value)}
placeholder="Add todo" style={{ flex: 1 }} />
        <button onClick={add}>Add</button>
      </div>
      <ul>
        {todos.map(t => (
          <li key={t.id}>{t.title} {t.done ? '✅' : ''}</li>
        ))}
      </ul>
    </div>
  )
}
```

apps/frontend/Dockerfile

```
FROM node:20 AS build
WORKDIR /app
```

```
COPY package.json package-lock.json* ./
RUN npm ci || npm i
COPY . .
RUN npm run build

FROM nginx:1.27-alpine
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 80
```

---

4) Helm (umbrella chart)

`deploy/helm/app/Chart.yaml`

```
apiVersion: v2
name: app
version: 0.1.0
appVersion: "0.1.0"
```

`deploy/helm/app/values.yaml`

```
namespace: app
image:
  backend: ghcr.io/YOUR_GH_USER/multi-tier-k8s/backend:local
  frontend: ghcr.io/YOUR_GH_USER/multi-tier-k8s/frontend:local
replicas:
  backend: 2
  frontend: 1
service:
  type: ClusterIP
host: app.local
```

`deploy/helm/app/templates/namespace.yaml`

```
apiVersion: v1
kind: Namespace
metadata:
  name: {{ .Values.namespace }}
```

`deploy/helm/app/templates/backend-deploy.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: {{ .Values.namespace }}
```

```
spec:
  replicas: {{ .Values.replicas.backend }}
  selector:
    matchLabels: { app: backend }
  template:
    metadata:
      labels: { app: backend }
    spec:
      containers:
      - name: api
        image: {{ .Values.image.backend }}
        ports: [{ containerPort: 8080 }]
        volumeMounts:
        - name: data
          mountPath: /data
        readinessProbe:
          httpGet: { path: /healthz, port: 8080 }
        livenessProbe:
          httpGet: { path: /livez, port: 8080 }
      volumes:
      - name: data
        emptyDir: {}
```

`deploy/helm/app/templates/backend-svc.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: backend
  namespace: {{ .Values.namespace }}
spec:
  selector: { app: backend }
  ports:
  - port: 80
    targetPort: 8080
```

`deploy/helm/app/templates/frontend-deploy.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: {{ .Values.namespace }}
spec:
  replicas: {{ .Values.replicas.frontend }}
  selector:
    matchLabels: { app: frontend }
  template:
    metadata:
```

```
      labels: { app: frontend }
    spec:
      containers:
      - name: web
        image: {{ .Values.image.frontend }}
        ports: [{ containerPort: 80 }]
```

`deploy/helm/app/templates/frontend-svc.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: {{ .Values.namespace }}
spec:
  selector: { app: frontend }
  ports:
  - port: 80
    targetPort: 80
```

`deploy/helm/app/templates/ingress.yaml`

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: {{ .Values.namespace }}
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: {{ .Values.host }}
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service: { name: frontend, port: { number: 80 } }
      - path: /api
        pathType: Prefix
        backend:
          service: { name: backend, port: { number: 80 } }
```

---

5) kind config & Makefile

`ops/scripts/kind-config.yaml`

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
```

`Makefile`

```
REG ?= ghcr.io/YOUR_GH_USER/multi-tier-k8s
TAG ?= local
KIND_CLUSTER ?= multi

.PHONY: kind ingress images deploy echo-hosts clean

kind:
    kind create cluster --name $(KIND_CLUSTER) --config ops/scripts/kind-
config.yaml || true
    kubectl cluster-info --context kind-$(KIND_CLUSTER)

ingress:
    helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
    helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx -n
ingress-nginx --create-namespace

echo-hosts:
    echo "127.0.0.1 app.local" | sudo tee -a /etc/hosts

images:
    docker build -t $(REG)/backend:$(TAG) apps/backend-fastapi
    docker build -t $(REG)/frontend:$(TAG) apps/frontend

push:
    docker push $(REG)/backend:$(TAG)
    docker push $(REG)/frontend:$(TAG)

deploy:
    helm upgrade --install app deploy/helm/app
      --namespace app --create-namespace
      --set image.backend=$(REG)/backend:$(TAG)
      --set image.frontend=$(REG)/frontend:$(TAG)

clean:
    kind delete cluster --name $(KIND_CLUSTER)
```

6) GitHub Actions (CI)

`.github/workflows/ci.yml`

```
name: ci
on:
  push:
    branches: [ main ]
  pull_request:

jobs:
  build:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write
    steps:
    - uses: actions/checkout@v4
    - uses: docker/setup-qemu-action@v3
    - uses: docker/setup-buildx-action@v3
    - uses: docker/login-action@v3
      with:
        registry: ghcr.io
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}
    - name: Build & push backend
      uses: docker/build-push-action@v6
      with:
        context: apps/backend-fastapi
        push: true
        tags: ghcr.io/${{ github.repository }}/backend:$(echo $GITHUB_SHA |
cut -c1-7)
    - name: Build & push frontend
      uses: docker/build-push-action@v6
      with:
        context: apps/frontend
        push: true
        tags: ghcr.io/${{ github.repository }}/frontend:$(echo $GITHUB_SHA |
cut -c1-7)
```

7) README quick start

```
# Quick start (local)

## Prereqs
- Docker, kubectl, Helm, kind
```

```
- (Optional) ghcr login:
`echo $CR_PAT | docker login ghcr.io -u YOUR_GH_USER --password-stdin`

## Steps
1. `make kind`
2. `make ingress echo-hosts`
3. `make images`
4. `make deploy`
5. Open http://app.local (frontend) → it calls `/api/todos` via Ingress.

## Dev mode (without k8s)
- Backend:
`cd apps/backend-fastapi && poetry install && uvicorn app.main:app --reload
--port 8080`
- Frontend: `cd apps/frontend && npm i && npm run dev` → then set
`VITE_API_BASE=http://localhost:8080` in `.env.local`
```

---

8) Next additions (when ready)

- Add **PostgreSQL** StatefulSet and switch backend DB from SQLite → Postgres
- Add **HPA** for backend
- Install **Prometheus/Grafana** + **Loki/Promtail**
- Wire **Argo CD** for GitOps
- Add **NetworkPolicies** and **cert-manager**