# MCP UI (kubectl via MCP) — Full project

This repository contains a minimal full-stack app you can deploy to Kubernetes to expose a web UI that runs kubectl commands via an MCP-style backend. The backend runs kubectl (using a kubeconfig mounted into the pod) and returns results to the frontend.

**Important security note:** This example is intentionally simple for demonstration. Running arbitrary kubectl from a web UI is dangerous. Protect the UI with network controls, authentication, and RBAC. Consider restricting allowed commands and using read-only service accounts.

## Files included

* README.md — this file (instructions)
* Dockerfile — multi-stage build, builds frontend and backend
* backend/package.json
* backend/index.js — Express backend and API
* frontend/package.json
* frontend/src/App.js — React app
* k8s/deployment.yaml — Deployment + Service + Secret example
* k8s/ingress.yaml — optional Ingress manifest

## README

### Quick overview

1. Create a Kubernetes secret containing your kubeconfig:

kubectl create secret generic kubeconfig-secret --from-file=kubeconfig=/path/to/admin.conf -n default

1. Build and push Docker image (replace <repo>):

docker build -t <repo>/mcp-ui:latest .  
docker push <repo>/mcp-ui:latest

1. Deploy to cluster:

kubectl apply -f k8s/deployment.yaml  
# optional ingress  
kubectl apply -f k8s/ingress.yaml

1. Access the UI:

* If NodePort: http://<NodeIP>:30080
* If Ingress: http://mcp-ui.example.com

## Dockerfile

# Stage 1: build frontend  
FROM node:18-alpine AS frontend-builder  
WORKDIR /app/frontend  
COPY frontend/package\*.json ./  
RUN npm install --silent  
COPY frontend ./  
RUN npm run build  
  
# Stage 2: build backend  
FROM node:18-alpine AS backend  
WORKDIR /app  
COPY backend/package\*.json ./  
RUN npm install --production --silent  
COPY backend ./  
# copy frontend build into backend public  
COPY --from=frontend-builder /app/frontend/build ./public  
  
ENV PORT=3000  
EXPOSE 3000  
CMD ["node", "index.js"]

## backend/package.json

{  
 "name": "mcp-ui-backend",  
 "version": "1.0.0",  
 "main": "index.js",  
 "scripts": {  
 "start": "node index.js"  
 },  
 "dependencies": {  
 "express": "^4.18.2",  
 "body-parser": "^1.20.2",  
 "cors": "^2.8.5"  
 }  
}

## backend/index.js

const express = require('express');  
const bodyParser = require('body-parser');  
const { spawn } = require('child\_process');  
const path = require('path');  
const fs = require('fs');  
  
const app = express();  
app.use(bodyParser.json());  
app.use(require('cors')());  
  
const ADMIN\_TOKEN = process.env.ADMIN\_TOKEN || 'change-me-token';  
const KUBECONFIG\_PATH = process.env.KUBECONFIG\_PATH || '/etc/kubeconfig/kubeconfig';  
  
// Serve frontend static  
app.use('/', express.static(path.join(\_\_dirname, 'public')));  
  
// health  
app.get('/healthz', (req, res) => res.json({ ok: true }));  
  
// simple authenticated endpoint to run kubectl commands  
app.post('/api/exec', (req, res) => {  
 const token = req.headers['x-api-token'] || '';  
 if (token !== ADMIN\_TOKEN) {  
 return res.status(401).json({ error: 'unauthorized' });  
 }  
  
 const { command } = req.body;  
 if (!command || typeof command !== 'string') {  
 return res.status(400).json({ error: 'command required' });  
 }  
  
 // Basic allowlist: allow get/describe/top/logs only (adjust as needed)  
 const allowRegex = /^(kubectl\s+)?(get|describe|top|logs)\b/i;  
 if (!allowRegex.test(command.trim())) {  
 return res.status(400).json({ error: 'command not allowed' });  
 }  
  
 // Split command into args safely  
 // If user passed a full kubectl string, strip leading 'kubectl'  
 let cmd = command.trim();  
 if (cmd.startsWith('kubectl ')) cmd = cmd.replace(/^kubectl\s+/, '');  
 const args = cmd.split(/\s+/);  
  
 // Ensure kubeconfig exists  
 if (!fs.existsSync(KUBECONFIG\_PATH)) {  
 return res.status(500).json({ error: 'kubeconfig not found on server' });  
 }  
  
 const kubectl = spawn('kubectl', args, { env: { ...process.env, KUBECONFIG: KUBECONFIG\_PATH } });  
 let stdout = '';  
 let stderr = '';  
 kubectl.stdout.on('data', (d) => (stdout += d.toString()));  
 kubectl.stderr.on('data', (d) => (stderr += d.toString()));  
 kubectl.on('close', (code) => {  
 res.json({ code, stdout, stderr });  
 });  
});  
  
// fallback route to serve index.html for SPA routing  
app.get('\*', (req, res) => res.sendFile(path.join(\_\_dirname, 'public', 'index.html')));  
  
const port = process.env.PORT || 3000;  
app.listen(port, () => console.log('MCP UI backend listening on', port));

## frontend/package.json

{  
 "name": "mcp-ui-frontend",  
 "version": "1.0.0",  
 "private": true,  
 "dependencies": {  
 "react": "18.2.0",  
 "react-dom": "18.2.0",  
 "react-scripts": "5.0.1"  
 },  
 "scripts": {  
 "start": "react-scripts start",  
 "build": "react-scripts build"  
 }  
}

## frontend/src/App.js

import React, { useState } from 'react';  
import './App.css';  
  
function App() {  
 const [cmd, setCmd] = useState('kubectl get pods -A');  
 const [output, setOutput] = useState('');  
 const [loading, setLoading] = useState(false);  
  
 async function run() {  
 setLoading(true);  
 setOutput('');  
 const resp = await fetch('/api/exec', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json', 'x-api-token': prompt('Enter API token') },  
 body: JSON.stringify({ command: cmd })  
 });  
 const j = await resp.json();  
 setOutput(JSON.stringify(j, null, 2));  
 setLoading(false);  
 }  
  
 return (  
 <div style={{ padding: 20, fontFamily: 'Arial, sans-serif' }}>  
 <h2>MCP UI — kubectl executor</h2>  
 <p>Allowed commands: <code>get</code>, <code>describe</code>, <code>top</code>, <code>logs</code></p>  
  
 <textarea value={cmd} onChange={(e) => setCmd(e.target.value)} style={{ width: '100%', height: 80 }} />  
 <div style={{ marginTop: 8 }}>  
 <button onClick={run} disabled={loading}>Run</button>  
 </div>  
  
 <pre style={{ background: '#111', color: '#eee', padding: 10, marginTop: 12, whiteSpace: 'pre-wrap' }}>{output}</pre>  
 </div>  
 );  
}  
  
export default App;

## k8s/deployment.yaml

apiVersion: v1  
kind: Secret  
metadata:  
 name: kubeconfig-secret  
 namespace: default  
type: Opaque  
stringData:  
 # create this secret with: kubectl create secret generic kubeconfig-secret --from-file=kubeconfig=/path/to/admin.conf  
 kubeconfig: |  
 # placeholder - replace by creating the secret from file as instructed in README  
  
---  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: mcp-ui  
 namespace: default  
spec:  
 replicas: 1  
 selector:  
 matchLabels:  
 app: mcp-ui  
 template:  
 metadata:  
 labels:  
 app: mcp-ui  
 spec:  
 containers:  
 - name: mcp-ui  
 image: <your-repo>/mcp-ui:latest  
 imagePullPolicy: IfNotPresent  
 ports:  
 - containerPort: 3000  
 env:  
 - name: ADMIN\_TOKEN  
 valueFrom:  
 secretKeyRef:  
 name: mcp-ui-secret  
 key: admin-token  
 - name: KUBECONFIG\_PATH  
 value: /etc/kubeconfig/kubeconfig  
 volumeMounts:  
 - name: kubeconfig  
 mountPath: /etc/kubeconfig  
 readOnly: true  
 volumes:  
 - name: kubeconfig  
 secret:  
 secretName: kubeconfig-secret  
  
---  
apiVersion: v1  
kind: Service  
metadata:  
 name: mcp-ui  
 namespace: default  
spec:  
 type: NodePort  
 selector:  
 app: mcp-ui  
 ports:  
 - port: 3000  
 targetPort: 3000  
 nodePort: 30080

## k8s/ingress.yaml (optional)

apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
 name: mcp-ui-ingress  
 annotations:  
 nginx.ingress.kubernetes.io/rewrite-target: /  
spec:  
 rules:  
 - host: mcp-ui.example.com  
 http:  
 paths:  
 - path: /  
 pathType: Prefix  
 backend:  
 service:  
 name: mcp-ui  
 port:  
 number: 3000

## Security Recommendations

* Use a Kubernetes ServiceAccount with least privilege instead of mounting admin.conf.
* Restrict allowed kubectl subcommands (the backend currently allows only get|describe|top|logs).
* Protect the UI with authentication (API tokens, OAuth, or network ACLs).
* Run the backend with a non-root user.

If you want, I can also: - generate Kubernetes manifests that use a ServiceAccount + Role/RoleBinding instead of kubeconfig secret - add example kubectl create secret commands - create a CI-friendly build-and-push.sh script

Tell me which extras you want and I will update the project.