**1. JavaScript & TypeScript Mastery**

**1.1 Async Programming & Event Loop**

 Task:

o Implement a function schedule(fn, delay) that:

o Ensures fn is executed after a given delay while respecting a rate limit:

no more than N executions in a rolling 1-second window.

 Focus:

Async execution, event loop behavior, and Promises. Demonstrate

understanding of task/microtask queues and rate-limiting concepts.

class RateLimiter {

constructor(maxExecutions, windowMs = 1000) {

this.maxExecutions = maxExecutions;

this.windowMs = windowMs;

this.queue = []; // Queue of tasks waiting to run

this.timestamps = []; // Log of recent execution times

}

schedule(fn, delay) {

const task = {

fn,

scheduledAt: Date.now(),

delay,

};

this.queue.push(task);

this.processQueue();

}

processQueue() {

if (this.queue.length === 0) return;

const now = Date.now();

// Remove timestamps older than windowMs (rolling window)

this.timestamps = this.timestamps.filter(ts => now - ts < this.windowMs);

if (this.timestamps.length < this.maxExecutions) {

const task = this.queue.shift();

const executeAt = task.scheduledAt + task.delay;

const waitTime = Math.max(0, executeAt - now);

setTimeout(() => {

task.fn();

this.timestamps.push(Date.now());

this.processQueue(); // Process next task

}, waitTime);

} else {

// Wait until the oldest timestamp falls out of the 1s window

const timeUntilNext = this.windowMs - (now - this.timestamps[0]);

setTimeout(() => this.processQueue(), timeUntilNext);

}

}

}

// Example usage

const limiter = new RateLimiter(3); // Max 3 executions per 1000ms

for (let i = 0; i < 10; i++) {

limiter.schedule(() => {

console.log(`Task ${i} executed at ${new Date().toISOString()}`);

}, i \* 100); // Schedule each task with a 100ms offset

}

**1.2 Required Keys Utility Type**

 Task:

Create a utility type RequiredKeysT> that extracts only the required keys

from a given object type. Then implement a function that accepts an object of

type T and ensures all RequiredKeysT> are present and correctly typed.

 Focus: Mapped types, conditional types, type-level computation.

/\*\*

\* Validates that all required keys exist in the given object

\* @param {Object} obj - The object to validate

\* @param {string[]} requiredKeys - Array of keys that must be present

\* @returns {boolean} true if all keys are present, throws error otherwise

\*/

function validateRequiredKeys(obj, requiredKeys) {

for (const key of requiredKeys) {

if (!(key in obj) || obj[key] === undefined) {

throw new Error(`Missing required key: ${key}`);

}

}

return true;

}

// Example schema: keys that must be present

const requiredKeys = ["id", "email"];

// Example object - valid

const user1 = {

id: 101,

email: "hello@example.com",

name: "Alice", // optional

};

// Example object - missing required key

const user2 = {

id: 102,

name: "Bob",

};

try {

validateRequiredKeys(user1, requiredKeys);

console.log(" user1 is valid");

} catch (e) {

console.error("user1 error:", e.message);

}

try {

validateRequiredKeys(user2, requiredKeys);

console.log("user2 is valid");

} catch (e) {

console.error("user2 error:", e.message);

}

**1.3 Function Composition & Closures**

 Task:

Implement a function composition utility compose(fns: Function[]) that chains

multiple functions (either synchronous or asynchronous). The composed

function should return a new function where the output of one function

becomes the input to the next.

 Focus: Closures, functional composition, type inference, generics,

understanding of synchronous and asynchronous function chains.

// Compose utility: handles sync and async functions

function compose(fns) {

return function (input) {

return fns.reduce((acc, fn) => {

return Promise.resolve(acc).then(fn);

}, input);

};

}

// Example synchronous functions

const add = x => {

console.log("add:", x);

return x + 1;

};

const double = x => {

console.log("double:", x);

return x \* 2;

};

const square = x => {

console.log("square:", x);

return x \* x;

};

// Example asynchronous functions

const asyncAdd = async x => {

console.log("asyncAdd:", x);

await delay(100);

return x + 5;

};

const asyncDouble = async x => {

console.log("asyncDouble:", x);

await delay(100);

return x \* 3;

};

// Utility function for artificial delay

function delay(ms) {

return new Promise(resolve => setTimeout(resolve, ms));

}

// Test 1: Compose with sync functions

const syncPipeline = compose([add, double, square]);

syncPipeline(2).then(result => {

console.log(" Sync Result:", result); // Output: ((2 + 1) \* 2)^2 = 36

});

// Test 2: Compose with mixed sync + async functions

const asyncPipeline = compose([asyncAdd, double, asyncDouble]);

asyncPipeline(3).then(result => {

console.log(" Async Result:", result); // Output: (((3 + 5) = 8) \* 2 = 16) \* 3 = 48

});

**2. Angular-Focused Exercises**

**2.1 Change Detection, OnPush Lifecycle Hooks**

 Task: Given a parent and child Angular component, implement:

o The child with ChangeDetectionStrategy.OnPush

o Track and log change detection runs

o Demonstrate how ngOnChanges is triggered when parent input

changes

 Then:

o Optimize unnecessary renders using trackBy, pure pipes, and input

immutability

 Focus: Change detection, OnPush, immutability, lifecycle hooks

**child.component.ts**

import { Component, Input, OnChanges, SimpleChanges, ChangeDetectionStrategy } from '@angular/core';

@Component({

selector: 'app-child',

template: `

<div>

<p>Child received: {{ item.name }}</p>

</div>

`,

changeDetection: ChangeDetectionStrategy.OnPush

})

export class ChildComponent implements OnChanges {

@Input() item: { id: number; name: string };

ngOnChanges(changes: SimpleChanges) {

console.log('Child ngOnChanges triggered:', changes);

}

}

**parent.component.ts**

import { Component } from '@angular/core';

@Component({

selector: 'app-parent',

template: `

<h3>Parent Component</h3>

<button (click)="updateItem()">Update Item</button>

<button (click)="replaceItem()">Replace Item</button>

<app-child [item]="childItem"></app-child>

<hr />

<ul>

<li \*ngFor="let user of users; trackBy: trackById">

{{ user.name | uppercase }}

</li>

</ul>

`

})

export class ParentComponent {

childItem = { id: 1, name: 'Alpha' };

users = [

{ id: 1, name: 'Anna' },

{ id: 2, name: 'Ben' },

{ id: 3, name: 'Cara' }

];

// Mutates the existing object – OnPush won't detect

updateItem() {

this.childItem.name = 'Alpha Updated';

}

// Replaces the object reference – OnPush detects change

replaceItem() {

this.childItem = { ...this.childItem, name: 'Alpha Replaced' };

}

// Used with \*ngFor to avoid unnecessary DOM updates

trackById(index: number, user: { id: number }) {

return user.id;

}

}

**2.2 NgRx (or NGXS) State Handling**

 Task: Create a state management flow for a shopping cart using NgRx or

NGXS:

o Implement actions to add/remove/update cart items.

o Use selectors to derive total price and item count

o Persist the cart state to localStorage.

 Focus: NgRx/NGXS patterns, selectors, and side effects.

 Bonus: Implement lazy-loaded modules with isolated state.

**2.2.1. Define Cart Models (cart.model.ts)**

export interface CartItem {

id: string;

name: string;

quantity: number;

price: number;

}

**2.2.2. Define Actions (cart.actions.ts)**

import { createAction, props } from '@ngrx/store';

import { CartItem } from './cart.model';

export const addItem = createAction('[Cart] Add Item', props<{ item: CartItem }>());

export const removeItem = createAction('[Cart] Remove Item', props<{ id: string }>());

export const updateItem = createAction('[Cart] Update Item', props<{ item: CartItem }>());

export const loadCart = createAction('[Cart] Load from Storage', props<{ items: CartItem[] }>());

**2.2.3. Cart Reducer (cart.reducer.ts)**

import { createReducer, on } from '@ngrx/store';

import { CartItem } from './cart.model';

import \* as CartActions from './cart.actions';

export interface CartState {

items: CartItem[];

}

export const initialState: CartState = {

items: []

};

export const cartReducer = createReducer(

initialState,

on(CartActions.addItem, (state, { item }) => ({

...state,

items: [...state.items, item]

})),

on(CartActions.removeItem, (state, { id }) => ({

...state,

items: state.items.filter(i => i.id !== id)

})),

on(CartActions.updateItem, (state, { item }) => ({

...state,

items: state.items.map(i => (i.id === item.id ? item : i))

})),

on(CartActions.loadCart, (state, { items }) => ({

...state,

items

}))

);

**2.2.4. Selectors (cart.selectors.ts)**

import { createSelector, createFeatureSelector } from '@ngrx/store';

import { CartState } from './cart.reducer';

export const selectCart = createFeatureSelector<CartState>('cart');

export const selectCartItems = createSelector(selectCart, state => state.items);

export const selectCartTotal = createSelector(selectCartItems, items =>

items.reduce((acc, item) => acc + item.price \* item.quantity, 0)

);

export const selectItemCount = createSelector(selectCartItems, items =>

items.reduce((count, item) => count + item.quantity, 0)

);

**2.2.5. Persist to localStorage (cart.effects.ts)**

import { Injectable } from '@angular/core';

import { Actions, createEffect, ofType } from '@ngrx/effects';

import { tap, withLatestFrom, map } from 'rxjs/operators';

import \* as CartActions from './cart.actions';

import { Store } from '@ngrx/store';

import { selectCartItems } from './cart.selectors';

@Injectable()

export class CartEffects {

constructor(private actions$: Actions, private store: Store) {}

saveCart$ = createEffect(() =>

this.actions$.pipe(

ofType(CartActions.addItem, CartActions.removeItem, CartActions.updateItem),

withLatestFrom(this.store.select(selectCartItems)),

tap(([action, items]) => {

localStorage.setItem('cart', JSON.stringify(items));

})

),

{ dispatch: false }

);

loadCart$ = createEffect(() =>

this.actions$.pipe(

ofType('@ngrx/effects/init'),

map(() => {

const data = localStorage.getItem('cart');

const items = data ? JSON.parse(data) : [];

return CartActions.loadCart({ items });

})

)

);

}

**2.2.6. NgRx Store Module Setup**

**In your cart.module.ts (lazy-loaded!):**

import { NgModule } from '@angular/core';

import { StoreModule } from '@ngrx/store';

import { EffectsModule } from '@ngrx/effects';

import { cartReducer } from './cart.reducer';

import { CartEffects } from './cart.effects';

@NgModule({

imports: [

StoreModule.forFeature('cart', cartReducer),

EffectsModule.forFeature([CartEffects])

]

})

export class CartModule {}

**2.2.7. Using the Cart in a Component**

import { Component } from '@angular/core';

import { Store } from '@ngrx/store';

import { addItem, removeItem } from './cart.actions';

import { selectCartItems, selectCartTotal } from './cart.selectors';

@Component({

selector: 'app-cart',

template: `

<h3>🛒 Cart</h3>

<ul>

<li \*ngFor="let item of items$ | async">

{{ item.name }} - {{ item.quantity }} x {{ item.price }}

<button (click)="remove(item.id)">Remove</button>

</li>

</ul>

<p>Total: {{ total$ | async }}</p>

`

})

export class CartComponent {

items$ = this.store.select(selectCartItems);

total$ = this.store.select(selectCartTotal);

constructor(private store: Store) {}

remove(id: string) {

this.store.dispatch(removeItem({ id }));

}

}

**3. React-Focused Exercises**

**3.1 Custom Hooks & useMemo/useCallback**

 Task: Build a custom hook useFetchWithCache(url) that:

o Caches fetched results in memory

o Avoids re-fetching if the same URL is passed again

o Uses useCallback and useMemo to avoid unnecessary re-renders

 Focus: Hook composition, memoization, cache handling, performance tuning

**useFetchWithCache.js**

import { useEffect, useState, useMemo, useCallback } from 'react';

// Simple in-memory cache

const cache = new Map();

export function useFetchWithCache(url) {

const [data, setData] = useState(null);

const [loading, setLoading] = useState(!cache.has(url));

const [error, setError] = useState(null);

const fetchData = useCallback(async () => {

if (cache.has(url)) {

console.log('💾 Using cached data for:', url);

setData(cache.get(url));

setLoading(false);

} else {

try {

console.log(' Fetching:', url);

setLoading(true);

const res = await fetch(url);

if (!res.ok) throw new Error('Network error');

const result = await res.json();

cache.set(url, result);

setData(result);

} catch (err) {

setError(err.message);

} finally {

setLoading(false);

}

}

}, [url]);

useEffect(() => {

fetchData();

}, [fetchData]);

// Memoize return object to prevent unnecessary re-renders

return useMemo(() => ({ data, loading, error }), [data, loading, error]);

}

**App.js**

import React, { useState } from 'react';

import { useFetchWithCache } from './useFetchWithCache';

export default function App() {

const [url, setUrl] = useState('https://jsonplaceholder.typicode.com/posts/1');

const { data, loading, error } = useFetchWithCache(url);

return (

<div style={{ padding: '1rem' }}>

<h1>🪝 useFetchWithCache Demo</h1>

<button onClick={() => setUrl('https://jsonplaceholder.typicode.com/posts/1')}>

Load Post 1

</button>

<button onClick={() => setUrl('https://jsonplaceholder.typicode.com/posts/2')}>

Load Post 2

</button>

{loading && <p>Loading...</p>}

{error && <p>Error: {error}</p>}

{data && (

<div style={{ border: '1px solid #ccc', padding: '1rem', marginTop: '1rem' }}>

<h2>{data.title}</h2>

<p>{data.body}</p>

</div>

)}

</div>

);

}

**3.2 Performance Optimization & Virtualized Lists**

 Task: Render a list of 10,000 items using a virtualization strategy (e.g., react-

window):

o Profile the performance using React Profiler before and after

optimization.

o Optimize unnecessary re-renders using React.memo and useCallback.

 Focus: Virtualization, re-render minimization, profiling

**VirtualList.js**

import React, { useCallback } from 'react';

import { FixedSizeList as List } from 'react-window';

// Simulate a large data set

const items = Array.from({ length: 10000 }, (\_, index) => ({

id: index,

name: `Item #${index + 1}`

}));

// Memoized row component

const Row = React.memo(({ index, style, data }) => {

const item = data[index];

return (

<div style={style}>

{item.name}

</div>

);

});

export default function VirtualList() {

// Memoize item data

const itemData = items;

// Memoized click handler (if needed later)

const handleClick = useCallback((index) => {

console.log('Clicked item', index);

}, []);

return (

<div>

<h2>🧠 Virtualized List (10,000 items)</h2>

<List

height={500} // container height

itemCount={itemData.length}

itemSize={35} // each row height

width={'100%'}

itemData={itemData}

>

{Row}

</List>

</div>

);

}

**App.js**

import React from 'react';

import VirtualList from './VirtualList';

function App() {

return (

<div className="App">

<VirtualList />

</div>

);

}

export default App;

**3.3 Higher-Order Components (HoC)**

 Task: Create a Higher-Order Component withErrorBoundary that wraps any

component and catches rendering errors.

 Add fallback UI and optional retry mechanism.

 Focus: HoC patterns, error boundaries, reusable patterns in React

**withErrorBoundary.js**

import React from 'react';

// This is our Error Boundary component

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false, error: null };

}

static getDerivedStateFromError(error) {

// Update state so next render shows fallback UI

return { hasError: true, error };

}

componentDidCatch(error, info) {

console.error('🧨 Error caught in boundary:', error, info);

// You can log the error to an external service here

}

handleRetry = () => {

this.setState({ hasError: false, error: null });

};

render() {

if (this.state.hasError) {

return (

<div style={{ border: '1px solid red', padding: 20 }}>

<h3>Something went wrong.</h3>

<p>{this.state.error?.message}</p>

<button onClick={this.handleRetry}>🔁 Try Again</button>

</div>

);

}

return this.props.children;

}

}

// Higher-Order Component

export function withErrorBoundary(WrappedComponent) {

return function WrappedWithErrorBoundary(props) {

return (

<ErrorBoundary>

<WrappedComponent {...props} />

</ErrorBoundary>

);

};

}

**App.js**

import React from 'react';

import { withErrorBoundary } from './withErrorBoundary'; // your HOC

import './App.css'; // optional styling

// Example component that throws an error

function BrokenComponent() {

throw new Error("This component is broken!");

return <div>Should never be seen</div>;

}

// Wrap the component with the HOC

const SafeComponent = withErrorBoundary(BrokenComponent);

function App() {

return (

<div className="App">

<h2>Error Boundary HoC Example</h2>

<SafeComponent />

</div>

);

}

export default App;

**4. End-to-End Frontend Integration**

**4.1 Micro-Frontend Router Strategy**

 Task:

o Design a micro-frontend router strategy (can be Angular or React).

o Each micro-frontend should have its own route and be loaded

dynamically at runtime.

o Implement a shell app that lazy-loads remote apps based on route.

 Focus: Dynamic routing, lazy-loading, inter-app communication

**Shell App — webpack.config.js**

const ModuleFederationPlugin = require('webpack/lib/container/ModuleFederationPlugin');

const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: './src/bootstrap.js',

mode: 'development',

devServer: {

port: 3000

},

plugins: [

new ModuleFederationPlugin({

name: 'shell',

remotes: {

remote1: 'remote1@http://localhost:3001/remoteEntry.js'

}

}),

new HtmlWebpackPlugin({

template: './public/index.html'

})

]

};

**Remote App — webpack.config.js**

const ModuleFederationPlugin = require('webpack/lib/container/ModuleFederationPlugin');

const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: './src/bootstrap.js',

mode: 'development',

devServer: {

port: 3001

},

plugins: [

new ModuleFederationPlugin({

name: 'remote1',

filename: 'remoteEntry.js',

exposes: {

'./RemoteApp': './src/App'

}

}),

new HtmlWebpackPlugin({

template: './public/index.html'

})

]

};

**Shell App Routing (App.js)**

import React, { Suspense, lazy } from 'react';

import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';

const RemoteApp = lazy(() => import('remote1/RemoteApp'));

export default function App() {

return (

<Router>

<nav>

<Link to="/">Home</Link> | <Link to="/remote">Remote App</Link>

</nav>

<Suspense fallback={<div>Loading remote...</div>}>

<Routes>

<Route path="/" element={<div>Welcome to Shell App</div>} />

<Route path="/remote" element={<RemoteApp />} />

</Routes>

</Suspense>

</Router>

);

}

**Remote App (App.js)**

import React from 'react';

export default function RemoteApp() {

return <div style={{ color: 'green' }}>This is Remote App 1!</div>;

}

**4.2 Bundle Optimization & SSR Hydration**

 Task: Configure a simple React or Angular app with:

o Code splitting using dynamic imports.

o Lazy loading for routes/components.

o SSR hydration for the first page load.

 Focus: Bundle optimization, SSR hydration, lazy loading, and code-splitting.

**Add Lazy-Loaded Routes (App.jsx)**

import React, { Suspense, lazy } from 'react';

import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';

const Home = lazy(() => import('./routes/Home'));

const About = lazy(() => import('./routes/About'));

export default function App() {

return (

<BrowserRouter>

<nav>

<Link to="/">Home</Link> | <Link to="/about">About</Link>

</nav>

<Suspense fallback={<div>Loading...</div>}>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</Suspense>

</BrowserRouter>

);

}

**Create Routes**

**routes/Home.jsx**

export default function Home() {

return <h2>🏠 Home (SSR + Lazy Load)</h2>;

}

**routes/About.jsx**

export default function About() {

return <h2>About (Lazy Loaded)</h2>;

}

**Client Entry (main.jsx)**

import React from 'react';

import { hydrateRoot } from 'react-dom/client';

import App from './App';

hydrateRoot(document.getElementById('root'), <App />);

**Server Entry (entry-server.jsx)**

import React from 'react';

import ReactDOMServer from 'react-dom/server';

import App from './App';

export function render() {

return ReactDOMServer.renderToString(<App />);

}

**Server for SSR (server.js)**

import express from 'express';

import fs from 'fs';

import path from 'path';

import { fileURLToPath } from 'url';

import { render } from './dist/server/entry-server.js';

const app = express();

const \_\_dirname = path.dirname(fileURLToPath(import.meta.url));

const indexHTML = fs.readFileSync(path.resolve(\_\_dirname, './dist/client/index.html'), 'utf-8');

app.use('/assets', express.static(path.resolve(\_\_dirname, './dist/client/assets')));

app.get('\*', (req, res) => {

const appHTML = render();

const html = indexHTML.replace(`<!--app-html-->`, appHTML);

res.status(200).set({ 'Content-Type': 'text/html' }).end(html);

});

app.listen(3000, () => {

console.log('SSR server running at http://localhost:3000');

});

**5. Helm & CI Integration**

**5.1 Helm Chart for SPA**

 Task:

o Write a Helm chart for deploying a frontend SPA.

o Include service.enabled, ingress.enabled, and extraEnvVars from

values.yaml.

o Use conditionals and named templates in \_helpers.tpl

 Focus: Helm templating, conditionals, DRY templates, deployment config

**values.yaml**

replicaCount: 1

image:

repository: myregistry.com/my-spa

tag: latest

pullPolicy: IfNotPresent

service:

enabled: true

type: ClusterIP

port: 80

ingress:

enabled: true

className: nginx

host: my-spa.local

path: /

extraEnvVars:

- name: REACT\_APP\_API\_BASE

value: https://api.my-spa.com

resources: {}

**\_helpers.tpl**

{{/\*

Return the full name of the release

\*/}}

{{- define "spa-chart.fullname" -}}

{{ .Release.Name }}-spa

{{- end }}

{{/\*

Labels used by all resources

\*/}}

{{- define "spa-chart.labels" -}}

app.kubernetes.io/name: {{ include "spa-chart.fullname" . }}

app.kubernetes.io/instance: {{ .Release.Name }}

{{- end }}

**deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: {{ include "spa-chart.fullname" . }}

labels:

{{ include "spa-chart.labels" . | nindent 4 }}

spec:

replicas: {{ .Values.replicaCount }}

selector:

matchLabels:

app: {{ include "spa-chart.fullname" . }}

template:

metadata:

labels:

app: {{ include "spa-chart.fullname" . }}

spec:

containers:

- name: spa

image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"

imagePullPolicy: {{ .Values.image.pullPolicy }}

ports:

- containerPort: 80

env:

{{- range .Values.extraEnvVars }}

- name: {{ .name }}

value: {{ .value | quote }}

{{- end }}

**service.yaml**

{{- if .Values.service.enabled }}

apiVersion: v1

kind: Service

metadata:

name: {{ include "spa-chart.fullname" . }}

spec:

type: {{ .Values.service.type }}

selector:

app: {{ include "spa-chart.fullname" . }}

ports:

- protocol: TCP

port: {{ .Values.service.port }}

targetPort: 80

{{- end }}

**ingress.yaml**

{{- if .Values.ingress.enabled }}

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: {{ include "spa-chart.fullname" . }}

annotations:

kubernetes.io/ingress.class: {{ .Values.ingress.className }}

spec:

rules:

- host: {{ .Values.ingress.host }}

http:

paths:

- path: {{ .Values.ingress.path }}

pathType: Prefix

backend:

service:

name: {{ include "spa-chart.fullname" . }}

port:

number: {{ .Values.service.port }}

{{- end }}

**5.2 Helm Hooks for Frontend Cache Busting**

 Task: Write a Helm post-upgrade hook that:

o Sends a request to Cloudflare/AWS to purge CDN cache or update an

S3 website deployment with cache-busted filenames.

 Include helm.sh/hook, hook-weight, and hook-delete-policy

 Focus: Deployment automation, hooks, CDN/cache invalidation

apiVersion: batch/v1

kind: Job

metadata:

name: {{ include "spa-chart.fullname" . }}-cache-buster

annotations:

"helm.sh/hook": post-upgrade

"helm.sh/hook-weight": "5"

"helm.sh/hook-delete-policy": before-hook-creation,hook-succeeded

spec:

template:

spec:

restartPolicy: Never

containers:

- name: purge-cache

image: curlimages/curl:latest

command: ["/bin/sh", "-c"]

args:

- >

echo "Purging CDN cache..." &&

curl -X POST "https://api.cloudflare.com/client/v4/zones/{{ .Values.cloudflare.zoneId }}/purge\_cache"

-H "Authorization: Bearer {{ .Values.cloudflare.apiToken }}"

-H "Content-Type: application/json"

--data '{"purge\_everything":true}'

# optional: add resources/securityContext/etc.