I'm not a robot

webpack-dev-server error: "Invalid Host header" occurs when starting server, resolved by disabling host check or setting allowedHosts property. ========== To access your local Vue project on the internet, you can use ngrok to generate a link that works across different networks. However, the Webpack dev server may throw an error due to its host checking mechanism. Disabling Host Check ---------- You can resolve this issue by disabling the host check in your webpack.config.js file or setting the allowedHosts property to all. For versions greater than 4.0.0, set the allowedHosts property to all: ```javascript module.exports = { // ... rest devServer: { compress: true, port: 9000, // set this property allowedHosts: 'all', }, }; ``` Setting allowedHosts to all bypasses host checking. Allowing Specific Hosts ---------- If you want your server to be accessible externally, specify the host that should be used in your webpack.config.js file: ```javascript module.exports = { // ... rest devServer: { compress: true, // set this property host: 'example.com', }, }; ``` Using 0.0.0.0 to Specify All IPV4 Addresses ---------- You can use 0.0.0.0 to specify all IPV4 addresses on the local machine: ```javascript module.exports = { // ... rest devServer: { compress: true, // set this property host: '0.0.0.0', }, }; ``` Using Webpack CLI to Specify Host ---------- You can also use the Webpack CLI to specify the host: ```bash npx webpack serve --host 0.0.0.0 ``` Specifying Allowed Hosts in allowedHosts Array ---------- You can resolve the error by explicitly specifying the allowed hosts in the allowedHosts array in your webpack.config.js file: ```javascript module.exports = { // ... rest devServer: { compress: true, port: 9000, // set this property allowedHosts: ['.ngrok.io', '.amazonaws.com', 'host.com', 'subdomain.host.com', 'subdomain2.host.com', 'host2.com',], }, }; ``` Setting Allowed Hosts in Vue.config.js ---------- If you use Vue.js, you would set the allowedHosts property in your vue.config.js file instead of in webpack.config.js. Allowing Wildcard Domains ---------- A value beginning with . can be used as a subdomain wildcard. For example, .host.com will match host.com, www.host.com, and any other subdomain of host.com. Setting allowedHosts to All ---------- If none of the suggestions helped, you can try setting allowedHosts to all: ```javascript module.exports = { // ... rest devServer: { compress: true, port: 9000, // set this property allowedHosts: 'all', }, }; ``` This approach bypasses host checking and should be used only for local development, as apps that don't check the host are vulnerable to DNS rebinding attacks. Disable Host Check in Create React App Project ---------- If you got the error in a Create React App project, set the DANGEROUSLY_DISABLE_HOST_CHECK environment variable to true in your .env file: ```bash DANGEROUSLY_DISABLE_HOST_CHECK=true ``` Please note that disabling host checking can pose security risks.Resolve the Invalid Host Header Error in Local Development ========== When the DANGEROUSLY_DISABLE_HOST_CHECK environment variable is set to true, the frontend and backend are treated as if they are on the same host and port. This can help resolve the invalid host header error in Angular applications. Setting the --disable-host-check option when issuing the ng serve command can also fix the issue. You can update the start script in your package.json file to include this option: You may get an invalid host header error. You are using a proxy server. If you are using a proxy server, it may be intercepting your requests and changing the host header. This can cause ngrok to think that the request is coming from an invalid host. How to fix it There are several thing you can do to fix an invalid host header error. Firstly, checck the spelling of the host header. Make sure you are typing it correctly. Also, make shure the host header is a valid domain name. A vaid domain name is a name that can be used to access a website. Another thing you can do is to check if the host header is associated with the server that is receiving the request. If you are using a custom ngrok domain, you need to make sure it is properly configured. If you are using a proxy server, try disabling it. Disabling the proxy server may fix the problem. To prevent this error in the future, use the correct ngrok tunnel and make shure you are using a valid domain name. When creating a custom ngrok domain, make shure the domain name is vaid. Also, configure your proxy server properly. By following these tips, you can help to prevent invalid host header errors.To avoid an invalid host header error, it is essential to identify and address the root cause of the issue. This can be achieved by checking the server logs, which may provide valuable information about the error. Additionally, contacting the web hosting provider can help in resolving the problem. It is also crucial to use a different browser if the issue persists. ========== An invalid host header error occurs when a web server receives a request with a host header that does not match its hostname. This discrepancy can be caused by various factors, such as a misconfigured proxy server or a browser or application issue. ========== To resolve an invalid host header error, it is vital to determine the cause of the problem and take corrective action. This may involve checking the client's configuration, ensuring the server's hostname is correctly set in its configuration file, and disabling host header validation if necessary.ngrok invalid host header error occurs when ngrok is unable to resolve the host header in the request. This can happen for a variety of reasons, such as incorrect configuration, DNS issues, or network problems. To fix the error, you can try checking your ngrok configuration to make sure that the host header is correct, and then verify that your DNS settings are resolving correctly. ========== When running a React app with Ngrok, the server rejects the request with an "Invalid Host Header" error because it thinks the request is coming from an unauthorized source. To fix this, you can try one of several solutions depending on your setup. The most straightforward approach is to use the --host flag when starting your React development server. If you're using create-react-app or a similar setup, find the start script in your package.json and add the --host flag. This tells the server to accept connections from any host (or a specific one), which resolves the host header mismatch. For example, if your start script looks like this: "scripts": { "start": "react-scripts start", } You can modify it to include the --host flag like so: "scripts": { "start": "react-scripts start --host 0.0.0.0", } This way, when you restart your React server, Ngrok should be able to connect without any issues. Another approach is to set the PUBLIC_URL environment variable using create-react-app. This helps with routing and asset loading, especially if you have routing configured. While not directly solving the host header issue, it can sometimes be necessary for a solution, particularly when dealing with routing problems. If your setup involves a proxy server like http-proxy-middleware, you might need to configure the proxy to forward the Host header correctly. This is less common in standard React development setups but becomes relevant if you have a backend server involved. For most create-react-app projects, using the --host 0.0.0.0 flag is usually the simplest and recommended solution. If you're experiencing routing issues alongside the host header problem, setting the PUBLIC_URL can also be beneficial.Looking forward to seeing everyone at the meeting tomorrow and discussing our strategies. ========== To access your React app, you can use a URL provided by ngrok, which is a tool that creates a secure tunnel to your development server. When you start your React app using npm start or yarn start, it starts on port 3000 by default. ========== In order for this to work, you need to run ngrok first and get the URL that it provides. This URL will be used in place of localhost in your React app. ========== For example, if you run ngrok using the command "ngrok http 3000", it will give you a URL like . You can then use this URL to access your React app instead of localhost. ========== To make this work, you also need to set an environment variable called PUBLIC_URL in your package.json file. This variable tells React where the app will eventually be deployed, which is important for routing and asset loading. ========== You can set the PUBLIC_URL using either a .env file or directly in the terminal. If you use a .env file, you would add the following line to the .env file: "PUBLIC_URL= ". You then start your React app using npm start or yarn start. ========== If you prefer to set the PUBLIC_URL directly in the terminal, you can run ngrok first and then use the command "npm start" with the PUBLIC_URL variable set. This will tell React to generate the correct URLs for your application's resources. ========== Make sure that the port you're using with ngrok matches the port your React development server is running on (usually 3000). If the ports don't match, it can cause issues with routing and asset loading in your app. ========== Also, keep in mind that the ngrok URL changes every time you restart ngrok. This means that you'll need to update the PUBLIC_URL variable whenever you make changes to package.json or set environment variables. You should usually prefer using 0.0.0.0 instead of a specific ngrok URL to avoid this issue. ========== Finally, after making changes to package.json or setting environment variables, you need to restart your React development server for the changes to take effect. If you're using a proxy server in front of your React development server, you might also need to configure the proxy to correctly forward the Host header and set changeOrigin to true to handle CORS issues properly.ngrok's ease of use can be hindered by issues like host header mismatches, but there are ways to troubleshoot and resolve these problems. When developing with ngrok, it allows requests from any host, which is helpful, but in production, this should not be the case. The Host header must match a list of allowed hosts; otherwise, an "Invalid Host Header" error occurs. This error can be frustrating when trying to connect to a React development server via ngrok. To resolve this issue, one solution is to manually validate the Host header in an Express.js server. However, if you're using create-react-app, a simpler approach is to use the --host option with your ngrok subdomain. For example, "start": "react-scripts start --host your-ngrok-subdomain.ngrok.io". This sets the host header correctly and allows ngrok to connect to your React development server without any issues.Creating Cloud Endpoints and Forwarding Traffic When creating cloud endpoints through ngrok's Dashboard or API, it's essential to understand how they handle traffic. Endpoints are persistent until deletion and have a built-in traffic policy that controls how incoming requests are handled. HTTPS Endpoints To create an HTTPS endpoint, you need to specify both a scheme (https) and a hostname. The default port for https is 443, but if you omit it, the default will be used automatically. For example, creating an endpoint with the URL ` ` would forward traffic to this domain. Valid and Invalid URLs When setting up public endpoints, the hostname must have a valid public suffix. Ports can also have specific requirements depending on the scheme. Using an invalid port or hostname will result in an error. Endpoint Defaults If you create an agent endpoint without specifying a complete URL, ngrok defaults to constructing an endpoint URL based on certain values. Cloud endpoints require both a scheme and hostname when created, so always specify them explicitly. Listening for Both HTTP and HTTPS Traffic To listen for both http and https traffic simultaneously, you can create two separate endpoints with their respective protocols. Domains and BYOD (Bring Your Own Domain) Domains are crucial in setting up public endpoints. They enable branded domains and TLS certificate management. You may also use wildcard domains to create public endpoints that receive traffic for all subdomains matching the given domain. Wildcard Endpoints You can create an endpoint that receives traffic for multiple subdomains by using a wildcard domain, such as `*.example.com`. This allows you to route connections from various URLs to a single endpoint. However, ensure you've created a wildcard domain before setting up a public wildcard endpoint. Note: Connections are routed to the most specific online endpoint available.ngrok provides flexibility in creating public endpoints for subdomains of a wildcard domain. For instance, if you reserve *.example.com, you can route requests to specific subdomains like api1.example.com or api2.example.com via the CLI. ngrok will handle traffic routing even if the upstream services are running on different ports. If one endpoint is at port 80 while another is at port 81, specifying subdomains through the CLI ensures correct routing. When creating public endpoints for subdomains, a domain is necessary. However, internal endpoints don't require a domain name; ngrok will assign a random hostname automatically if you omit it in the CLI command. For example, running `ngrok http 80` might create an endpoint like . For security and authentication, Traffic Policy can be attached to endpoints for routing, authenticating, and transforming traffic. Public endpoints are accessible by default unless secured with authentication. Basic Auth or OAuth policies can be used for securing endpoints with Traffic Policy actions. ngrok also allows rewriting the host header of incoming requests to ensure compatibility with application servers that expect a specific hostname. When rewriting the host header, ngrok automatically rewrites the location header of HTTP responses to match the endpoint URL's hostname. Traffic forwarding from the ngrok agent to upstream services is handled through specifying a URL or port number in the CLI command. The scheme (http or https) used for forwarding depends on the specified upstream URL; if none is provided, http is used by default except when forwarding to port 443, in which case https is chosen. When HTTP2 forwarding is enabled, all requests are transmitted over HTTP/2 Cleartext as TLS was already terminated at the ngrok cloud service. We cannot use TLS-ALPN currently. We rely on HTTP/2 with Prior Knowledge serving file directories. The ngrok agent supports the file:// scheme in a forwarding URL. When using this scheme, the ngrok agent serves local file system directories by its own built-in file server. Paths in file:// URLs must be specified as absolute paths. Serve files in /var/log The ngrok agent can serve files on Windows. Serve files in your current working directory The agent also supports serving files from the current working directory. Traffic Observability The Traffic Inspector provides a real-time view of HTTP traffic flowing through your endpoints in the dashboard. You can choose to capture either request metadata or full request and response bodies. Log Export Events Logs of traffic can be exported to HTTP/S endpoints with ngrok's events system. Advanced HTTP/S endpoints are standards-compliant reverse proxies. Versions HTTPS endpoints support HTTP/1.1, while HTTP/S endpoints support HTTP/2, which is used automatically when the client supports it. Client support is determined via standard ALPN negotiation. Even if your upstream service does not support HTTP/2, HTTP/2 is still used between the client and endpoint. Websockets Websocket connections are supported without configuration. ngrok does not forward hop-by-hop headers to the upstream service except for Connection: upgrade headers. Persistent connections When a connection is made with HTTP/1.1, ngrok may choose persistent connections to improve future requests from the same client if it supports it. Well Known URIs The ngrok agent takes over handling of /.well-known/acme-challenge path in any matching Domain with automated certificate management enabled. You can disable this behavior by uploading your own certificate on the matching Domain. TLS Ngrok handles TLS (SSL) certificate management and termination for you automatically, without setup or configuration. TLS connections to https endpoints are terminated at ngrok's cloud service.The final value of the header in your application code is crucial in reading the values injected by ngrok. You may remove these headers with the Remove Headers Traffic Policy action if needed. Connection Limits & Timeouts Contact us to configure limits and timeouts on connections to HTTP endpoints. ========== Connection LimitNameNotes5 minutesClient Idle TimeoutTime since data was last transmitted by the upstream service5 minutesServer Idle TimeoutTime since data was last transmitted by the upstream serviceNo limitData transmittedData transmitted by the client or upstream service TLS LimitNameNotes60 secondsTLS Handshake DurationTime between ClientHello received and handshake completion64 KBHandshake Message SizeMax size of non-certificate handshake messages256 KBCertificate Message SizeMax size of certificate handshake messages16 KBRecord Payload Size HTTP LimitNameNotesNo timeoutRound Trip TimeoutTime for the entire HTTP request and response HTTP Request LimitNameNotes1 MBRequest Header SizeIncludes method, URI and headers1 MBRequest URI LengthLimited by the size of the request headerNo timeoutRequest TimeoutTime to read the entire HTTP request from the clientNo timeoutRequest Header TimeoutTime to read the HTTP request header from the clientNo limitRequest Body Size HTTP Response LimitNameNotes1 MBResponse Header SizeIncludes method, URI and headersNo timeoutResponse TimeoutTime to read the entire HTTP response from the serverNo timeoutResponse Header TimeoutTime to read the HTTP response header from the serverNo limitResponse Body Size Errors If ngrok fails to handle an HTTP request it will set the ngrok-error-code header in the HTTP response with a unique ngrok Error Code describing the failure. ngrok guarantees that the upstream service may never set the ngrok-error-code HTTP response header so you know reliably that it was set by ngrok. ngrok may return an error under the following conditions: Your upstream service timed out or rejected the connection Your upstream service returned a response that was not valid HTTP A Traffic Policy action rejected the request. Traffic Policy execution encountered a runtime error. ngrok encountered an internal error API HTTP/S Endpoints can be created programatically. Consult the documentation on Endpoint APIs. Pricing HTTP/S endpoints are available on all plans. Consult the Endpoints Pricing documentation for billing details. See Domains pricing for details on pricing for custom domains, wildcard domains and more. When using ngrok with a React development server, you might encounter the "Invalid Host Header" error due to security measures implemented in webpack-dev-server. This error occurs because webpack-dev-server validates the Host header in incoming requests to prevent DNS rebinding attacks, and the Host header from ngrok might not match what webpack-dev-server expects. There are a few ways to resolve this issue, each with its own advantages and disadvantages: Disabling Host Check: This is the simplest solution but also the least secure. You can disable the Host header check by adding disableHostCheck: true to your webpack-dev-server configuration. However, this makes your development server vulnerable to DNS rebinding attacks, so it should only be used in development environments with caution. Setting Allowed Hosts: A more secure approach is to explicitly specify the allowed hosts in your webpack-dev-server configuration. You can add the ngrok URL to the allowedHosts array to allow requests from that specific URL while maintaining security for other hosts. Using a Custom Server: For more advanced control, you can create a custom Node.js server using Express or a similar framework. This server can proxy requests to your React development server and ensure that the Host header is set correctly. This method provides more flexibility and allows for additional customization.ngrok with React development server "Host Header" error often occurs due to security measures in webpack-dev-server. This issue can be resolved by understanding the problem, its causes and solutions. ========== Understanding the Issue: webpack-dev-server security validates Host headers to prevent DNS rebinding attacks. ngrok exposes local servers with public URLs but sometimes mismatched Host headers cause errors. Solutions: a) Disabling Host Check (Less Secure) Locate webpack-dev-server configuration: Find it in webpack.config.js, package.json scripts or a custom setup. Add disableHostCheck option: ```javascript module.exports = { // ... other configurations devServer: { disableHostCheck: true, // ... other devServer options }, } ``` Caution: This disables security check and makes development server vulnerable to DNS rebinding attacks. b) Setting Allowed Hosts (More Secure) Identify ngrok URL: Start ngrok and note the generated public URL. Add it to allowed hosts in webpack-dev-server configuration: ```javascript module.exports = { // ... other configurations devServer: { allowedHosts: ['your-unique-id.ngrok.io', // Replace with your ngrok URL 'localhost',], // ... other devServer options }, } ``` This allows requests from specific ngrok URLs while maintaining security for other hosts. c) Using a Custom Server (Advanced) Create a Node.js server using Express or similar framework. Proxy requests to React development server, ensuring correct Host header: ```javascript const express = require('express'); const { createProxyMiddleware } = require('http-proxy-middleware'); const app = express(); app.use('/', createProxyMiddleware({ target: ' ', // Your React dev server URL changeOrigin: true, })); app.listen(5000); // Port for your custom server ``` This provides more control over headers and allows additional customization. Additional Tips: Restart Dev Server: After making configuration changes, restart React development server for them to take effect. ngrok Subdomains: If using paid ngrok features, consider using a custom subdomain for better control and stability. Security Best Practices: Always prioritize security, especially when exposing your development server publicly. Choose the solution that balances convenience and safety for your development workflow.const path = require('path'); module.exports = { // ... other configurations devServer: { allowedHosts: ['your-unique-id.ngrok.io', // Replace with your ngrok URL 'localhost',], // ... other devServer options }, }; ========== c) Using a Custom Server (Advanced): const express = require('express'); const { createProxyMiddleware } = require('http-proxy-middleware'); const app = express(); // Assuming your React dev server is running on port 3000 app.use('/', createProxyMiddleware({ target: ' ', changeOrigin: true, })); app.listen(5000, () => { console.log('Custom server listening on port 5000'); }); ========== Explanation: a) Disabling Host Check: This code snippet adds the disableHostCheck: true option to your webpack-dev-server configuration, effectively disabling the host header check. b) Setting Allowed Hosts: This code snippet adds your ngrok URL and 'localhost' to the allowedHosts array within the devServer configuration. This allows requests from these specific hosts while maintaining security for others. c) Using a Custom Server: This code snippet creates a simple Express server that acts as a proxy. It forwards incoming requests to your React development server (running on port 3000 in this example) while ensuring the Host header is set correctly. ========== Important Considerations: - Replace 'your-unique-id.ngrok.io' with your actual ngrok URL in option b). - After making changes to your webpack configuration, restart your development server for the changes to take effect. - Option a) is the least secure and should only be used in development environments with caution. - Option b) provides a good balance between security and convenience. - Option c) offers more flexibility and control but requires setting up a separate server. - Choose the solution that best fits your needs and security requirements. ========== Alternative Solutions: - Environment Variables: Instead of hardcoding the ngrok URL, consider using environment variables in your webpack configuration. This allows for more flexibility and avoids the need to modify the code when the ngrok URL changes. Example in webpack.config.js module.exports = { // ... other configurations devServer: { allowedHosts: [process.env.NGROK_URL, 'localhost',], // ... other devServer options }, }; ========== Localhost Tunneling Alternatives: - Explore other localhost tunneling tools like localtunnel or Forwarder, which might offer different security features or better integration with your workflow. ========== Troubleshooting Tips: - Check ngrok Logs: If you're still facing issues, examine the ngrok logs for any errors or warnings that might provide clues about the problem. - Verify Configuration: Double-check your webpack-dev-server and ngrok configurations to ensure everything is set up correctly. - Network Connectivity: Ensure your network connection is stable and that there are no firewall rules blocking communication. ========== Security Considerations: - Development vs. Production: Remember that the solutions mentioned here are primarily for development environments. In production, you should use a proper domain name and SSL certificate for secure communication. - HTTPS with ngrok: If you need HTTPS for your development server, consider using ngrok's paid plans that offer custom subdomains and SSL certificates. ========== Additional Resources: By understanding the cause of the "Invalid Host Header" error and exploring the various solutions, you can effectively address the issue and continue developing your React applications with ngrok.To resolve the "Invalid Host Header" error when using ngrok with a React development server, consider disabling host checking as the simplest option but be aware of its security implications. Alternatively, setting allowed hosts provides a balance between security and convenience. For more control and customization, create a custom server to proxy requests to your React dev server, ensuring correct Host header. ==========___ When ngrok tries to connect to Angular or React dev server, Invalid Host Header is displayed. The issue can also arise when using webpack-dev-server with params --host 0.0.0.0. Disabling host check in ngrok may be the simplest solution but raises concerns about security. On the other hand, setting allowed hosts offers a balance between security and convenience. ==========ngrok.connect({ addr: configstrp.ngrok.port, authtoken: configstrp.ngrok.authtoken, host_header:3000 }, (err, url) => { if (!err) { console.log('Connected to ngrok'); var ngrokurl = 'http://'+ url; console.log(ngrokurl); } }); // dont pass a subdomain unless u have a custom domain set