Cross Origin Resource Sharing Implementation

Use Case:

Cross origin resource sharing is required when you are dealing with multiple domains and all of them need to be able to make calls to specific sub-domain or the API layer. Many times we even need to allow the Partner networks to have access to such API sub-domains. One can do this on backend servers but it gets complicated quickly and every change needs to be replicated on multiple backend servers in the setup. Doing the same through the load balancing setup is a much simpler way to get there.

F5 iRules:

```
# Domains that are allowed to make cross-domain calls to example.com
class allowed_origins {
    ".example.com"
    ".example2.com"
    ".goodpartner.com"
}

when HTTP_REQUEST {
    unset -nocomplain cors_origin
    if { class match [HTTP::header Origin] ends_with allowed_origins } {
        if { ( [HTTP::method] equals "OPTIONS" ) and ( [HTTP::header exists "Access-Control-Request-Method"] ) } {
            # CORS preflight request - return response immediately
            HTTP::respond 200 "Access-Control-Allow-Origin" [HTTP::header "Origin"] \
                              "Access-Control-Allow-Methods" "POST, GET, OPTIONS" \
                              "Access-Control-Allow-Headers" [HTTP::header "Access-Control-Request-Headers"] \
                              "Access-Control-Max-Age" "86400" \
                              "Vary" "Origin"
        } else {
            # CORS GET/POST requests - set cors_origin variable
            set cors_origin [HTTP::header "Origin"]
```

```
        }
    }
}

when HTTP_RESPONSE {
    # CORS GET/POST response - check cors_origin variable set in request
    if { [info exists cors_origin] } {
        HTTP::header insert "Access-Control-Allow-Origin" $cors_origin
        HTTP::header insert "Access-Control-Allow-Credentials" "true"
        HTTP::header insert "Vary" "Origin"
    }
}
```

URL: https://devcentral.f5.com/codeshare/cors-implementation

NetScaler Solution:

```
add responder action respond_act respondwith '"HTTP/1.1 200
OK\r\nAccess-Control-Allow-Origin:
"+HTTP.REQ.HEADER("Origin")+"\r\nAccess-Control-Allow-Methods: POST,
GET, OPTIONS\r\nAccess-Control-Allow-Headers:
"+HTTP.REQ.HEADER("Access-Control-Request-Headers")+"\r\nAccess-
Control-Max-Age: 86400\r\nVary: Origin\r\n\r\n"' -bypassSafetyCheck
YES


add responder policy allow_domains_pol
'HTTP.REQ.HEADER("Origin").ENDSWITH_ANY("allowed_domains") &&
HTTP.REQ.METHOD.EQ("OPTIONS") && HTTP.REQ.HEADER("Access-Control-
Request-Method").EXISTS' respond_act
```

Bind the Responder policy to specific VSERVER or to Global responder bind point.

```
add rewrite action insert_headers insert_after
HTTP.RES.FULL_HEADER.BEFORE_STR("\r\n\r\n") '"\r\nAccess-Control-
Allow-Origin: "+HTTP.REQ.HEADER("Origin")+"\r\nAccess-Control-Allow-
Credentials: true\r\nVary: Origin"' -bypassSafetyCheck YES


add rewrite policy insert_headers_pol
HTTP.REQ.HEADER("Origin").ENDSWITH_ANY("allowed_domains")
```

`insert_headers`

Bind the Rewrite policy to specific VSERVER or to Global rewrite bind point on response flow.

NetScaler advance policy infrastructure provides you with many cool modules. Responder and Rewrite and the commonly used ones where Responder module processes the requests and helps generate a response from NetScaler itself. If the response is generated from NetScaler then this request does not reach backend infrastructure. In this example we are generating specific response with custom headers like “Access-Control-Allow-Origin”, “Access-Control-Allow-Methods” and “Access-Control-Allow-Headers” etc.

While the Rewrite module helps with changing the request or response based on defined action. In this example we are able to insert custom headers like “Access-Control-Allow-Origin” and “Access-Control-Allow-Credentials” in the response coming from backend server. You can very well define where the insertion needs to take place in order to achieve the impact required.