Adobe Summit

LAB
WORKBOOK

L537 – Implement Adobe Target for Optimal Performance

Adobe Summit 2025

Lab Overview

In this lab you will learn a new implementation method for Adobe Target that will allow you to perform CDN experimentation. This new method will allow you to create A/B and XT Target activities and perform the decisioning work on your CDN for the fastest possible results.

Key Takeaways

- Understand the different Target implementation methods and how using edge workers can lead to better performance
- Using the Target Properties learn how to be selective about which activities are included in your rules bundle
- Get familiar with setting up the local environment for development
- Understand how to configure an Akamai edge worker for Target experimentation
- Learn the basics of how to code in the CDN environment

Prerequisites

- Adobe Target with product admin rights for editing workspaces and properties
- Enable the on-device decisioning capability in Adobe Target
- Access to your preferred CDN edge worker environment with the appropriate rights to configure settings
- Visual Studio Code software or another development environment of your choice

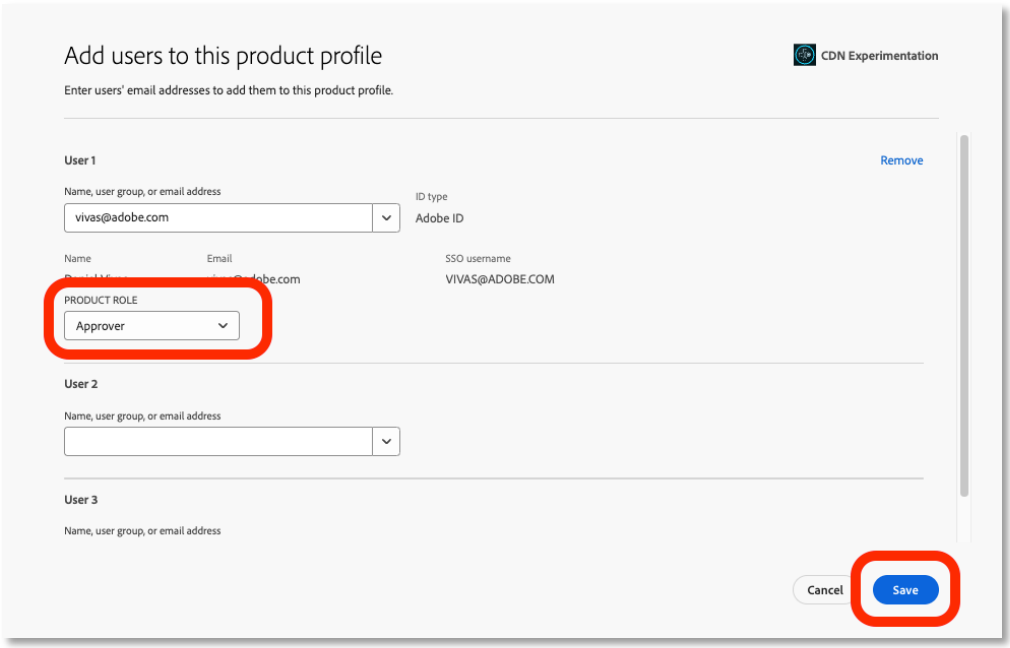
Part 1: Create On-Device Decisioning (ODD) activities in Target and assign it to CDN experimentation property

Section overview: In this section, we'll start by reviewing the On-device decisioning features available in Adobe Target. Next, we'll create a CDN Experimentation Target property (optional step for premium users). Then, using the provided lab user credentials, we will create an A/B Target test that will be applied on the lab test website.
Intro: ODD in Adobe Target

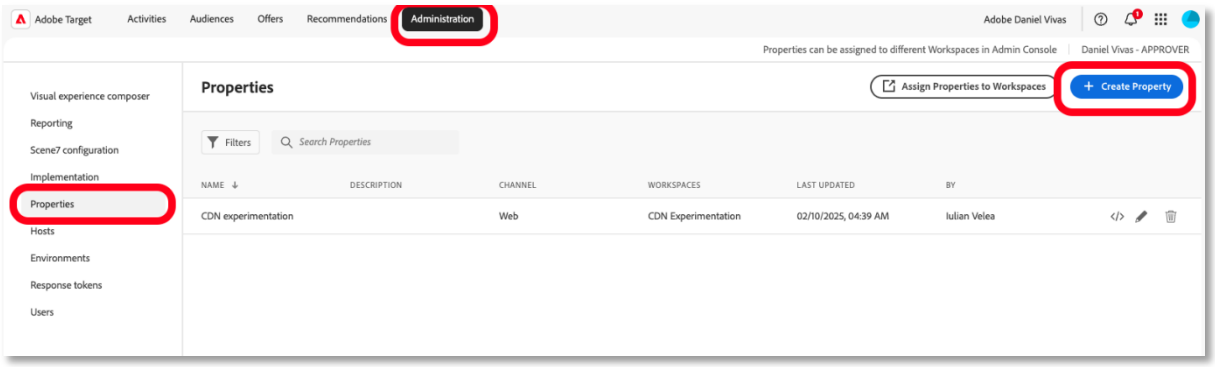
- Enables near-zero latency content delivery by performing in-memory decisioning on the device it operates on.
- Caches A/B Test and Experience Targeting (XT) activity data
- Requires a rules artifact

Exercise 1: Create a CDN Experimentation Target property for optimizing the ODD rules package size (preconfigured)

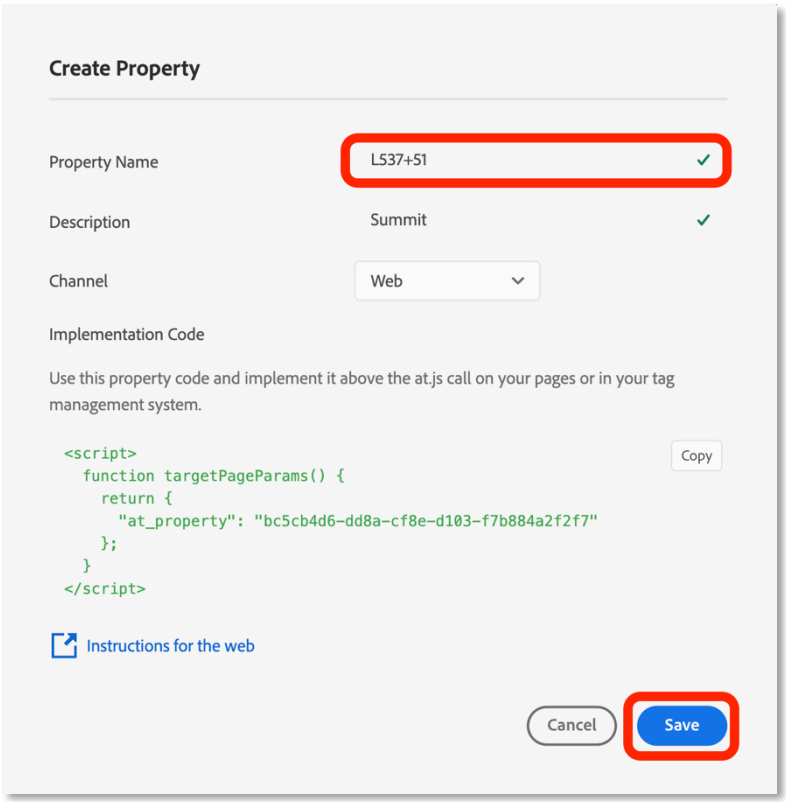
Step 1 [preconfigured on the lab machines]: Confirm the user Product Role for the workspace is set to Approver and Save



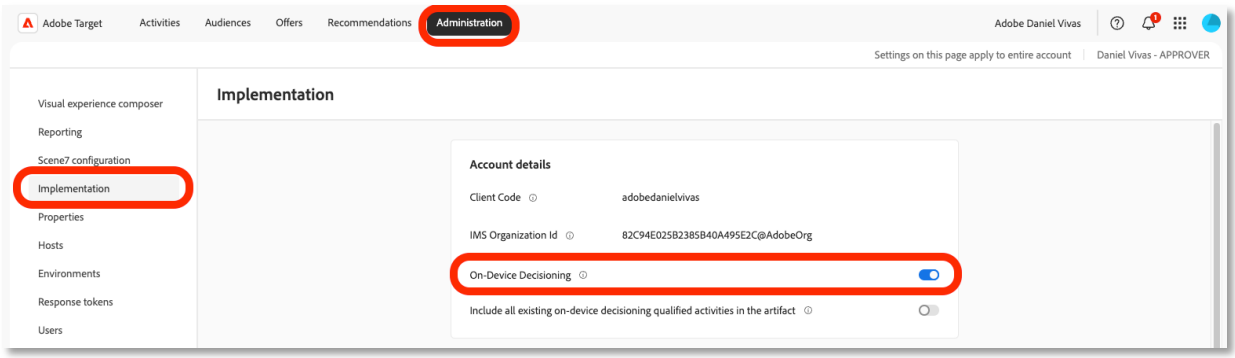
Step 2 [preconfigured on the lab machines]: In the Target UI navigate to the **Administration/Properties** section and click on Create Property button



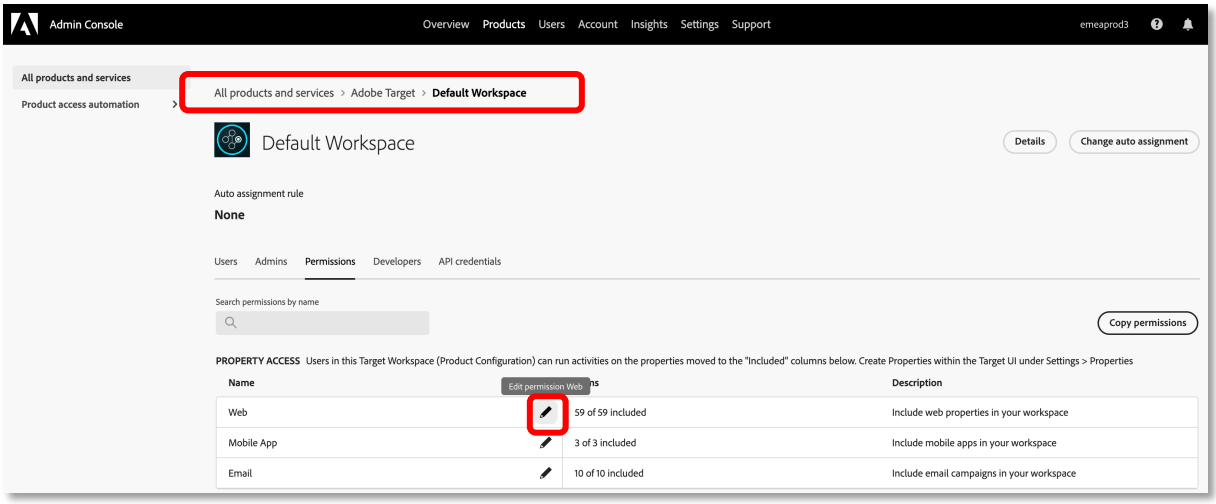
Step 3 [preconfigured on the lab machines]: Edit the properties details (your property will have the lab and seat number in the name e.g: L537+##) and click Save



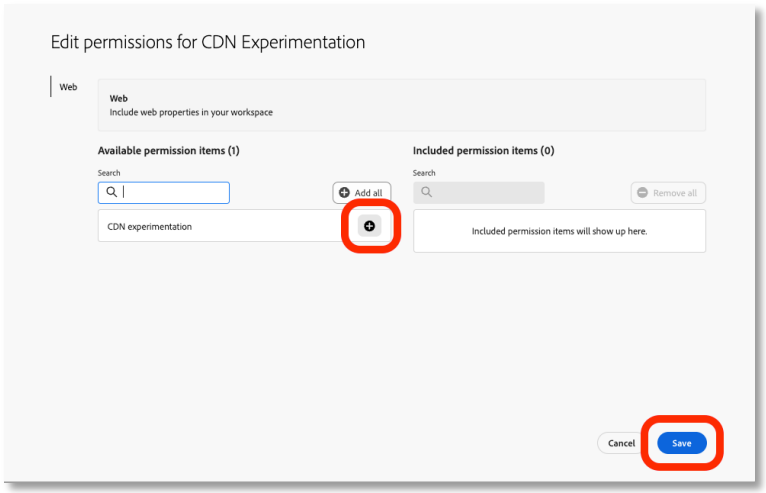
Step 4 [preconfigured on the lab machines]: Confirm the on-device decisioning setting is toggled on; this allows us to use the activities for On-Device Decisioning



Step 5 [preconfigured on the lab machines]: Navigate back to the **Admin Console/Adobe Target/Default Workspace** profile settings and edit the Profile Permissions by clicking on the pencil icon next to **"Web"** under **"Permissions Access"**

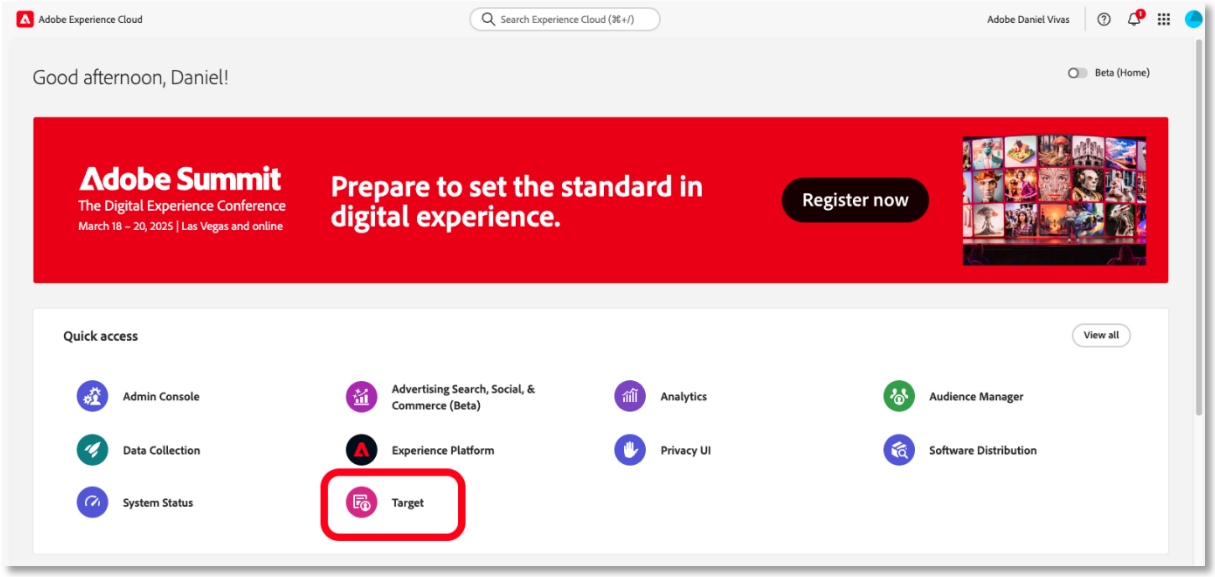


Step 6 [preconfigured on the lab machines]: Include the properties you will be using, in the permissions items by clicking on the "+" button and then **Save**


Edit permissions for CDN Experimentation
Web
Web
Include web properties in your workspace
Available permission items (1)
Search
Add all
CDN experimentation
Included permission items (0)
Search
Remove all
Included permission items will show up here.
Cancel
Save

Exercise 2: Log in to Adobe Target UI

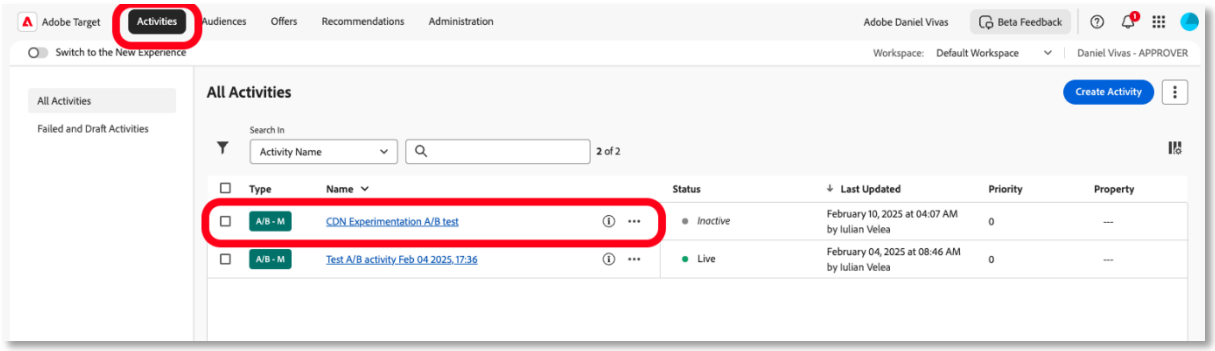
Step 1: Navigate to experience.adobe.com
Step 2: Enter the email address: L537+##@adobeeventlab.com
Step 3: Select "Company or School Account"
Step 4: Enter the password: Adobe4Summit!
Step 5: Select Adobe Target from the list of available solutions



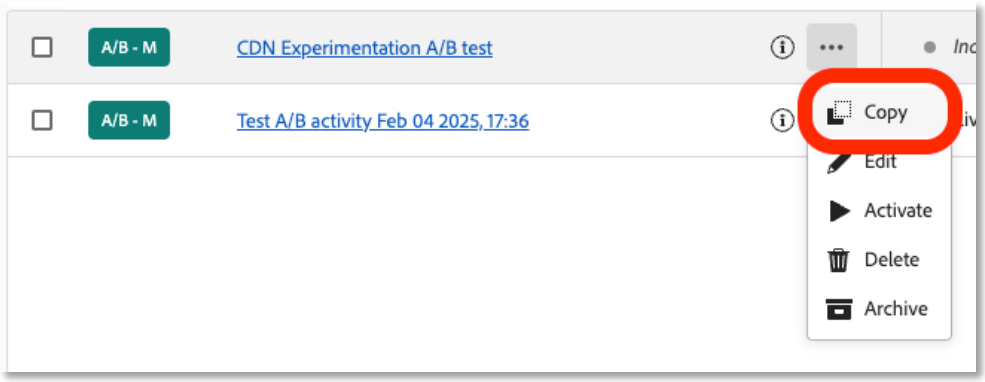
Exercise 3: Create the test A/B activity in the CDN Experimentation workspace

Lab website: https://main--aem-cdn-experimentation-demo--aenascut.aem.live/

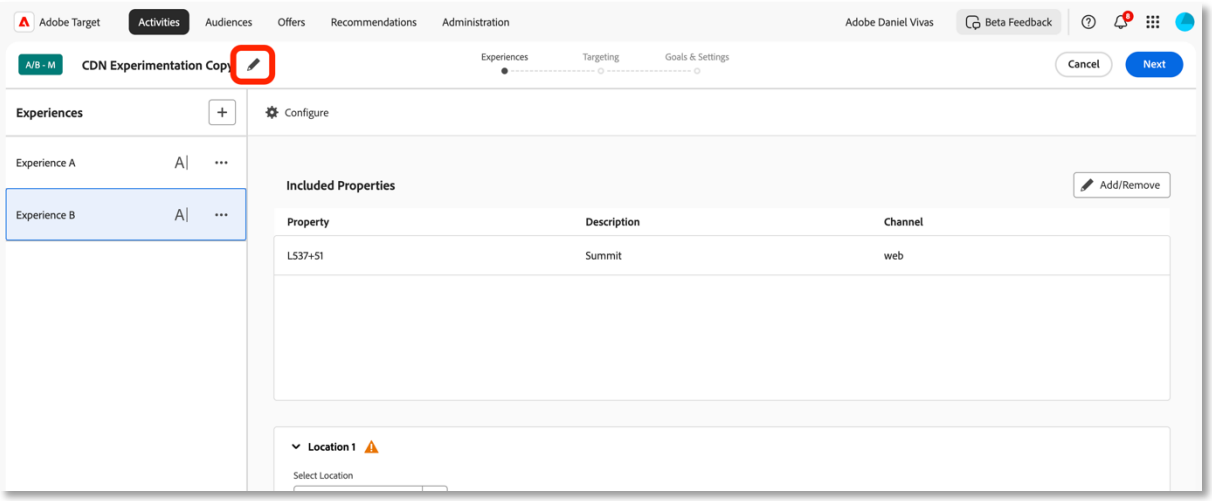
Step 1: Click on the Activities tab in the top horizontal navigation and identify the 'CDN Experimentation' activity that we have created in advance



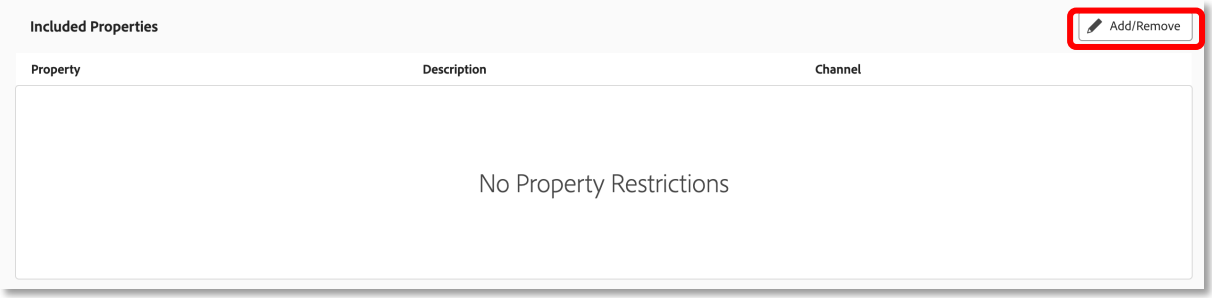
Step 2: Click on the ellipsis menu icon and select 'Copy' to duplicate the test activity

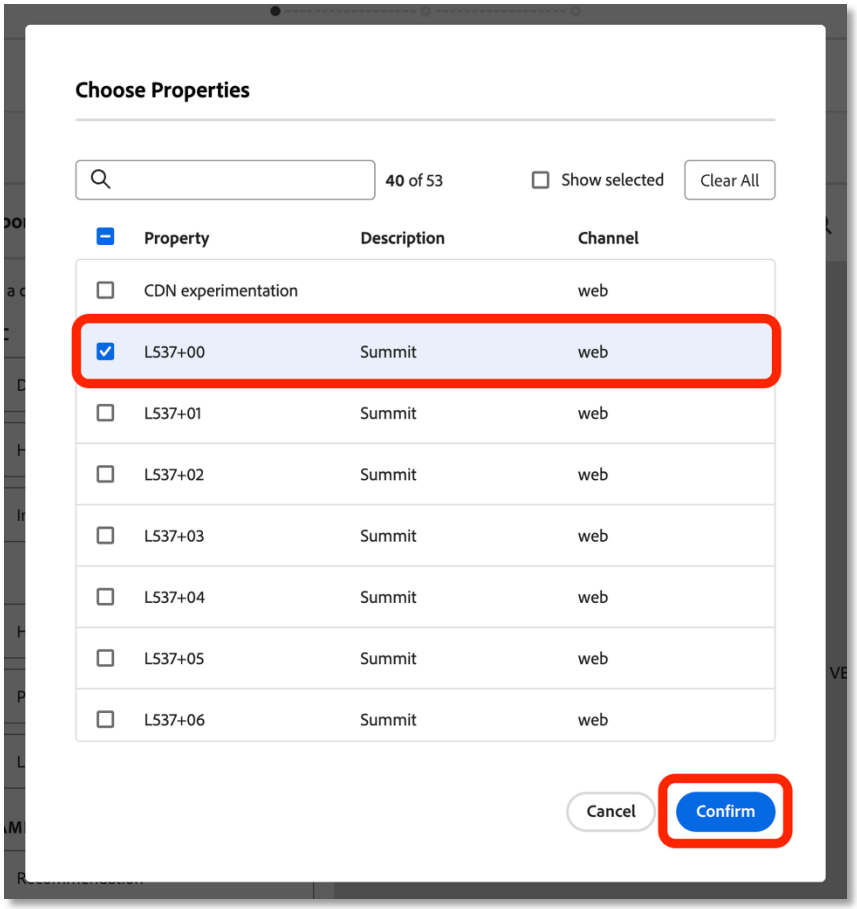


Step 3: Edit the activity name to include your seat number (ex: CDN Experimentation A/B test seat ##) and click save

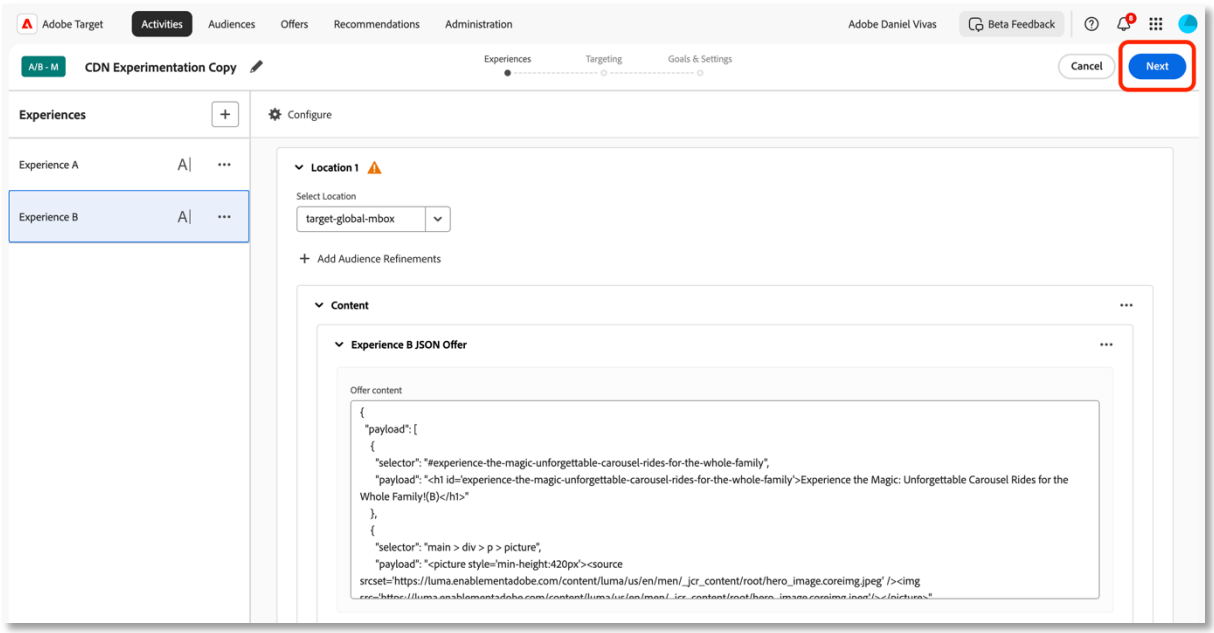


Step 4: From the activity editing page, in the Included Properties section click “Add/Remove” to add the corresponding property

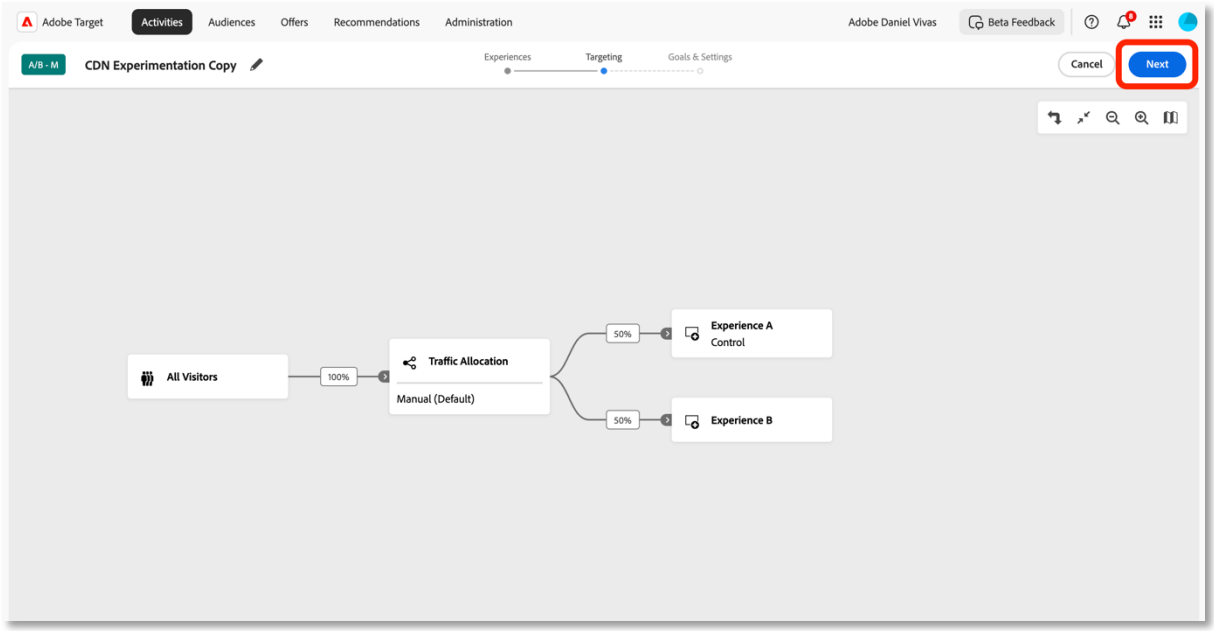




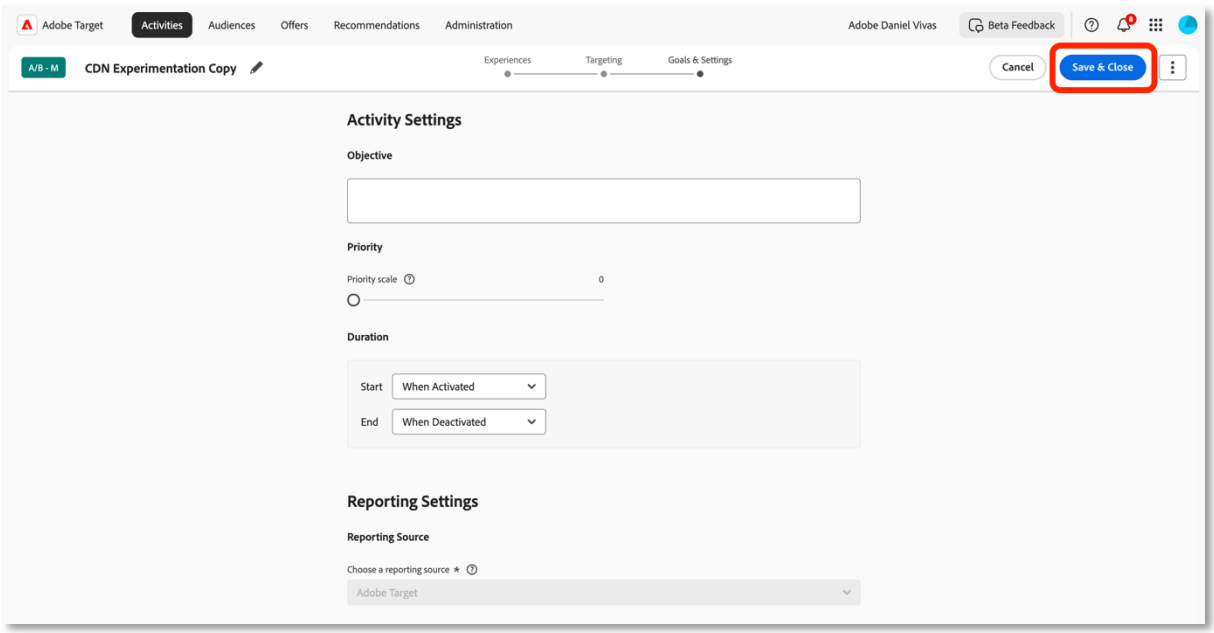
Step 5: Examine the changes being made to **Experience B** and click **Next**.
Bonus step: Add additional changes to the experience that you would want to see in your final result (e.g. change the **h1** element text and/or text color)



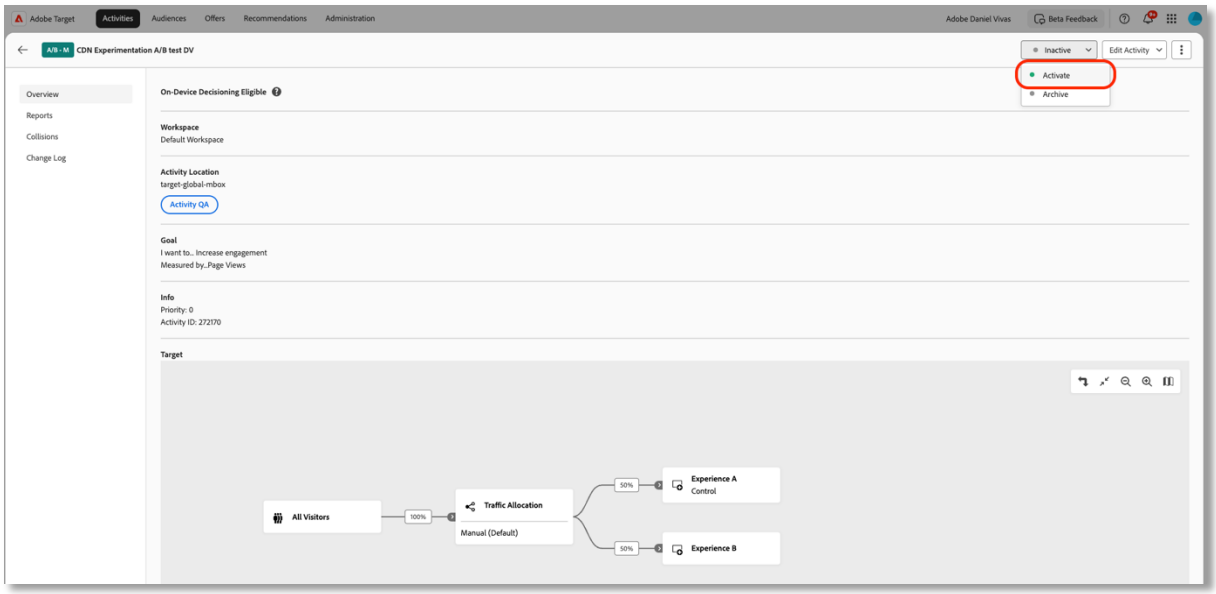
Step 6: Click **Next** on the “Targeting page”



Step 7: Click **Save & Close** to finish the editing



Step 8: Activate your Target activity


Adobe Target
Activities
Audiences
Offers
Recommendations
Administration
Adobe Daniel Vivas
Beta Feedback
A/B - M
CDN Experimentation A/B test DV
Inactive
Edit Activity
Activate
Archive
Overview
Reports
Collisions
Change Log
On-Device Decisioning Eligible
Workspace
Default Workspace
Activity Location
target-global-mbox
Activity QA
Goal
I want to.. Increase engagement
Measured by..Page Views
Info
Priority: 0
Activity ID: 272170
Target
All Visitors
100%
Traffic Allocation
Manual (Default)
50%
Experience A
Control
50%
Experience B

Part 2: CDN edge-worker CLI and CDN Property configuration

Intro: CDN overview

Content Delivery Networks benefits:

- Distributed servers delivering web content based on user location.
- Reduces load times and improves performance by caching content globally.

Edge Computing aspects:

- Brings computation and data storage closer to data sources and users.
- Reduces latency, conserves bandwidth, and enhances user experience.

CDNs + Edge Computing key benefits:

- Real-time data processing and lower latency.
- Improved performance for dynamic content.

Exercise 1: CDN CLI and CDN credentials

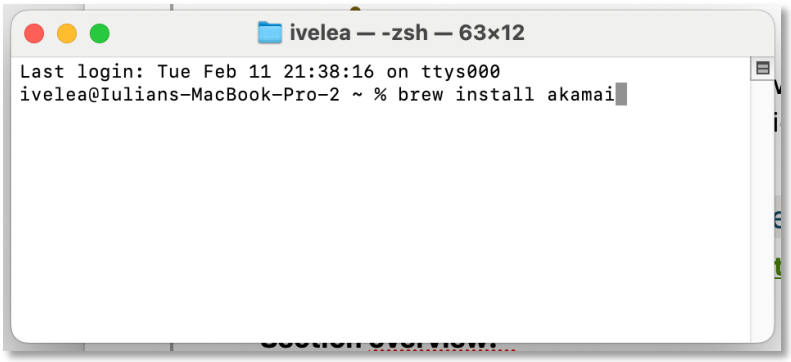
This section contains steps related to configurations already available on the lab machines.

Section overview: To be able to work with Akamai EdgeWorkers there are a few things that are required:

- Access to Akamai Control Center: https://control.akamai.com/
- Access to your organization group in Akamai
- A configured Property in Akamai
- Akamai CLI installed on development machine
- Akamai CLI for EdgeWorkers installed on development machine
- API Client credentials

Step 1 [preconfigured on the lab machines]: Install Akamai CLI

1. Open your terminal.



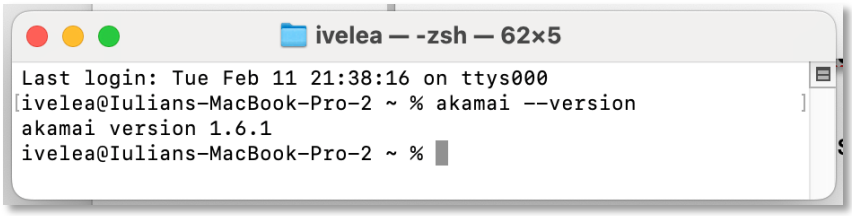
2. Run the following command to install Akamai CLI using Homebrew:

```
brew install akamai
```

Step 2: Verify Akamai CLI Installation - To ensure Akamai CLI is installed properly, run:

```
akamai --version
```

This should print the Akamai CLI version, confirming the installation.



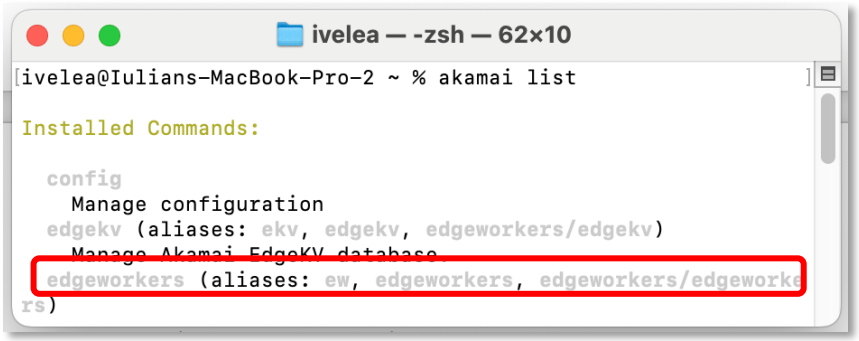
Step 3 [preconfigured on the lab machines]: Install Akamai CLI for EdgeWorkers Assuming Akamai CLI is installed, run the following command to install EdgeWorkers:

```
akamai install edgeworkers
```

Step 4: Verify EdgeWorkers Installation **-** To ensure EdgeWorkers is installed properly, run:

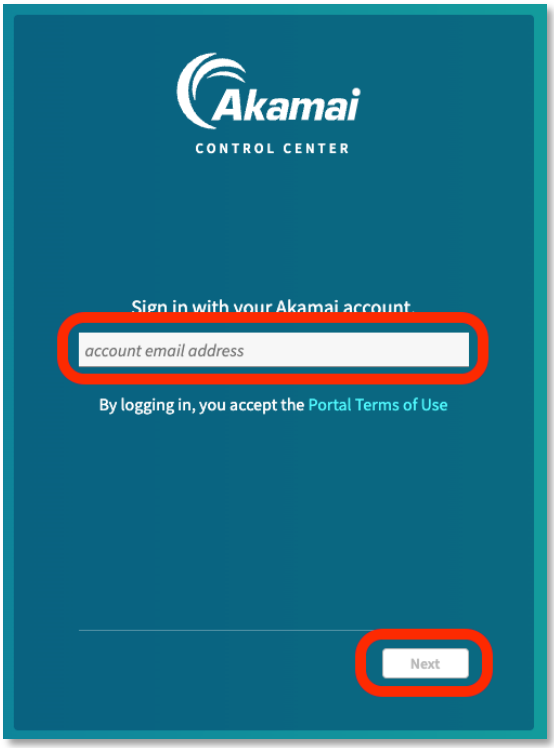
```
akamai list
```

This should print a list of available Akamai CLI commands, including **"edgeworkers"**.

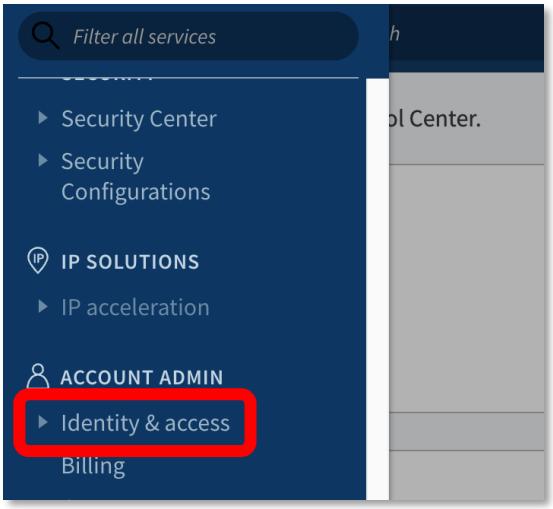


Step 5 [preconfigured on the lab machines]: Create an API Client

1. Go to **control.akamai.com** and use the registered email address to sign in by clicking on the **“Next”** button.

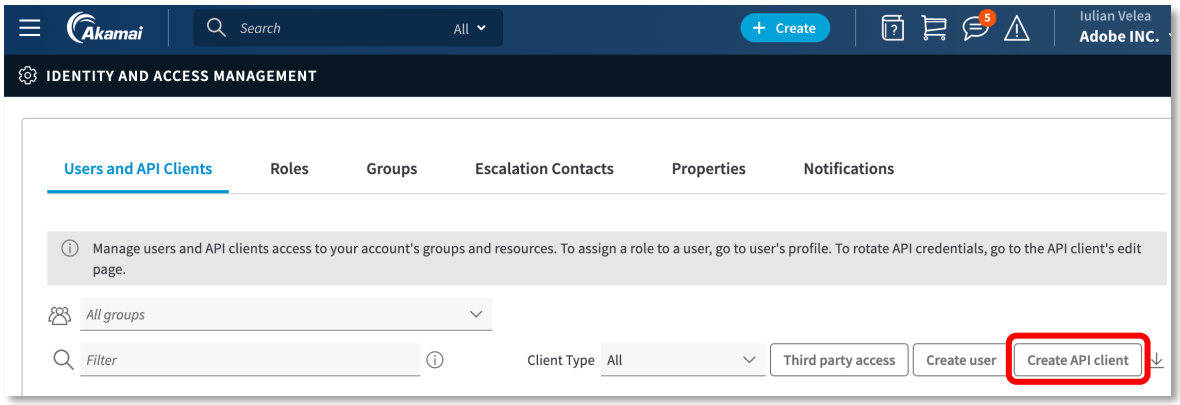


2. From the left navigation bar, go to **"Account Admin"**. Select **“Identity and Access”**.
3.

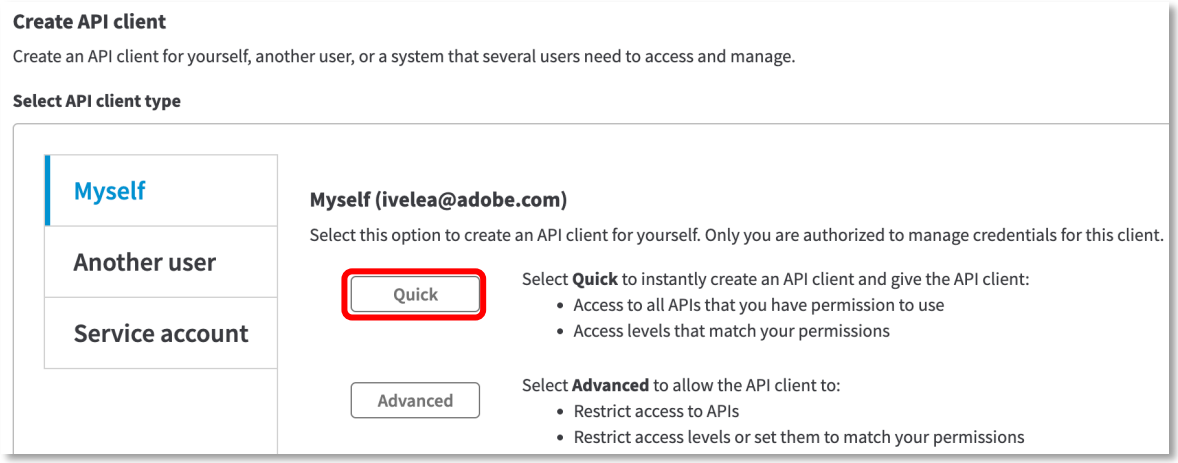


You should be redirected to the **"Users and API Clients"** tab.

4. On the **"Users and API Clients"** page, click **"Create API Client"**.
5.

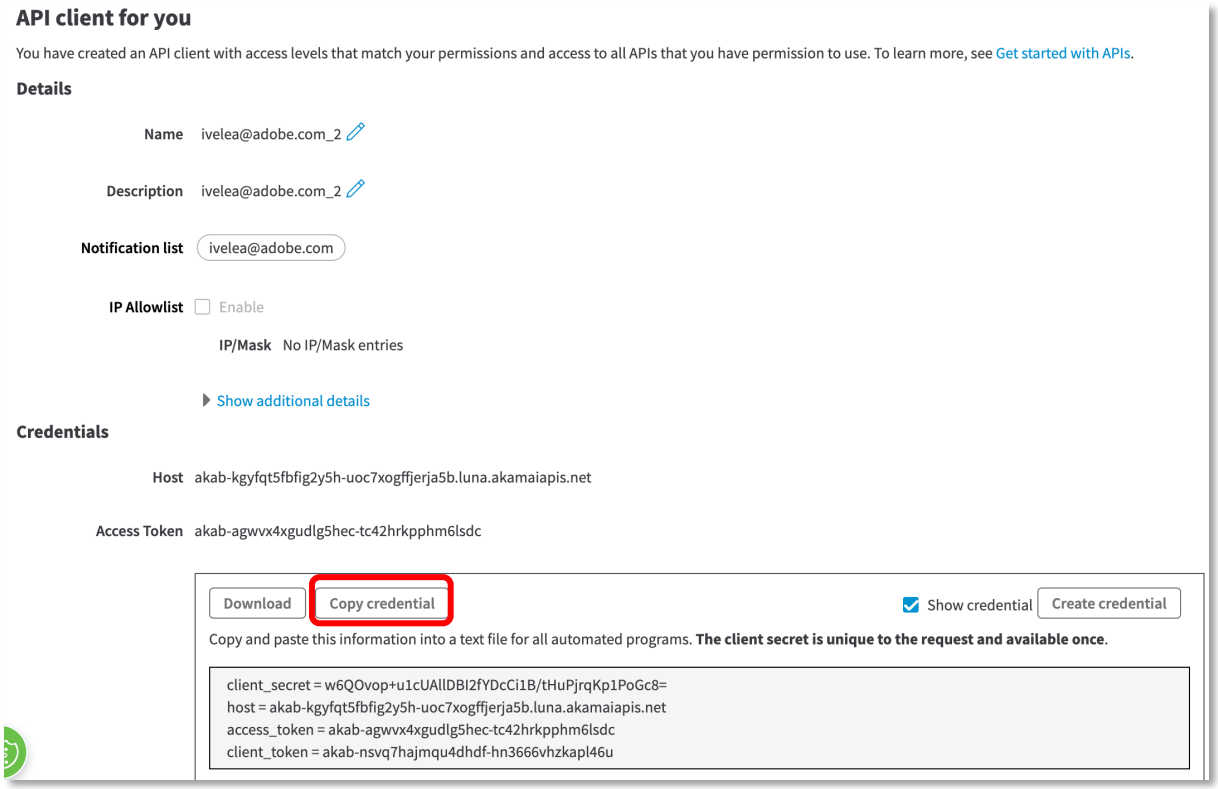


6. On the **"Create API Client"** page, select "**Myself"** and then click the **"Quick"** button. This will create an API Client with all your credentials. For a more restricted version, select **"Advanced"** and grant only the necessary permissions.
7.



A new page named **"API Client for you"** will open.

8. On the **"API Client for you"** page, go to the bottom table and copy your credentials or create a new one.



Make sure to copy the credentials and store them.

Step 6: Check Credentials

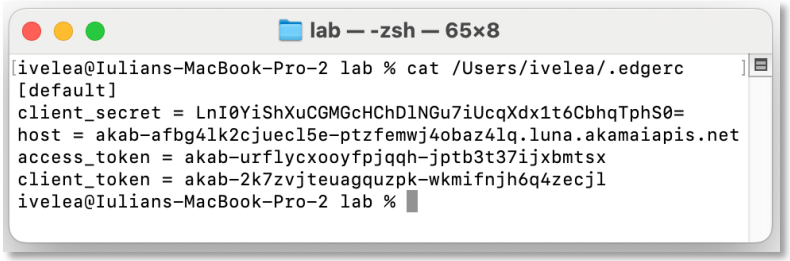
Akamai CLI needs credentials to execute operations that might change the Akamai account state. These credentials should be stored in a special file named `.edgerc`.

On Mac OS X, the `.edgerc` file should be located at:

```
/Users/{username}/.edgerc
```

Check your local file by running the following command in your terminal:

```
cat /Users/ivelea/.edgerc
```



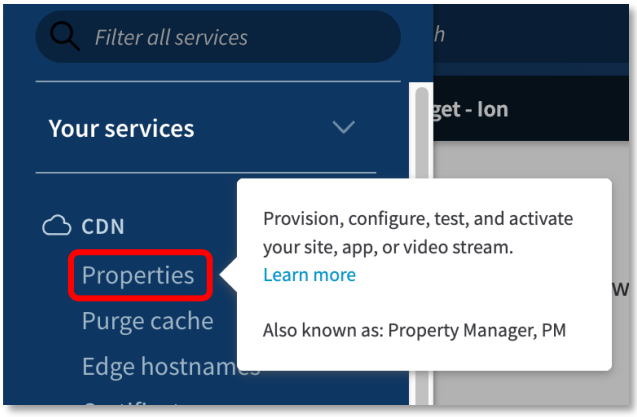
Exercise 2: CDN Property configuration (preconfigured for the lab)

Step 1: Log in to Akamai Control Center

Open your web browser and go to the Akamai Control Center: **control.akamai.com**
Enter your credentials to log in.

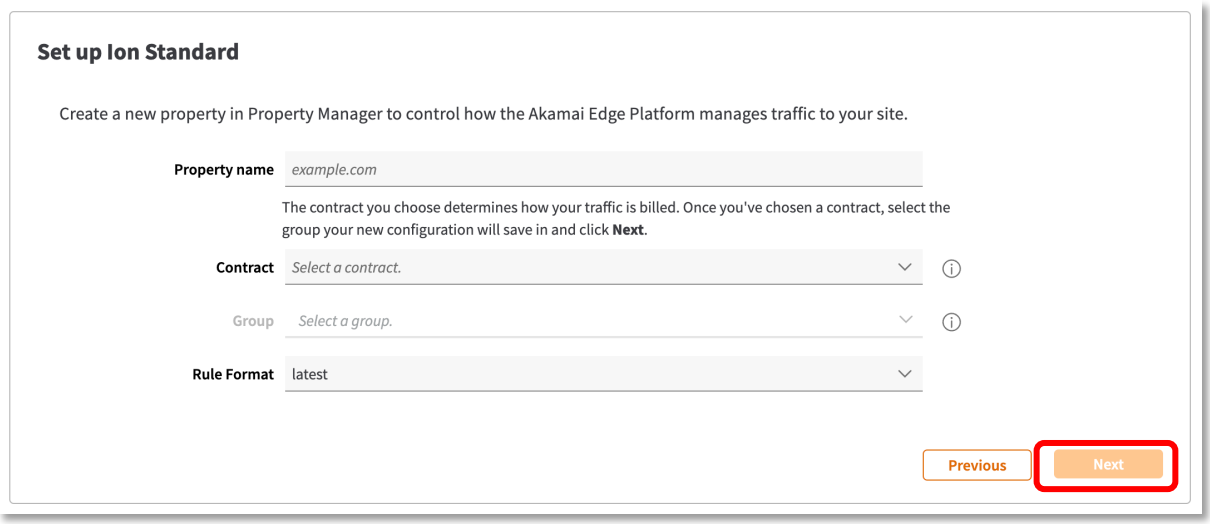
Step 2: Navigate to Property Manager

From the main dashboard, go to the left menu section. Click on **"Properties"** under the **“CDN”** section.



Step 3: Create a New Property

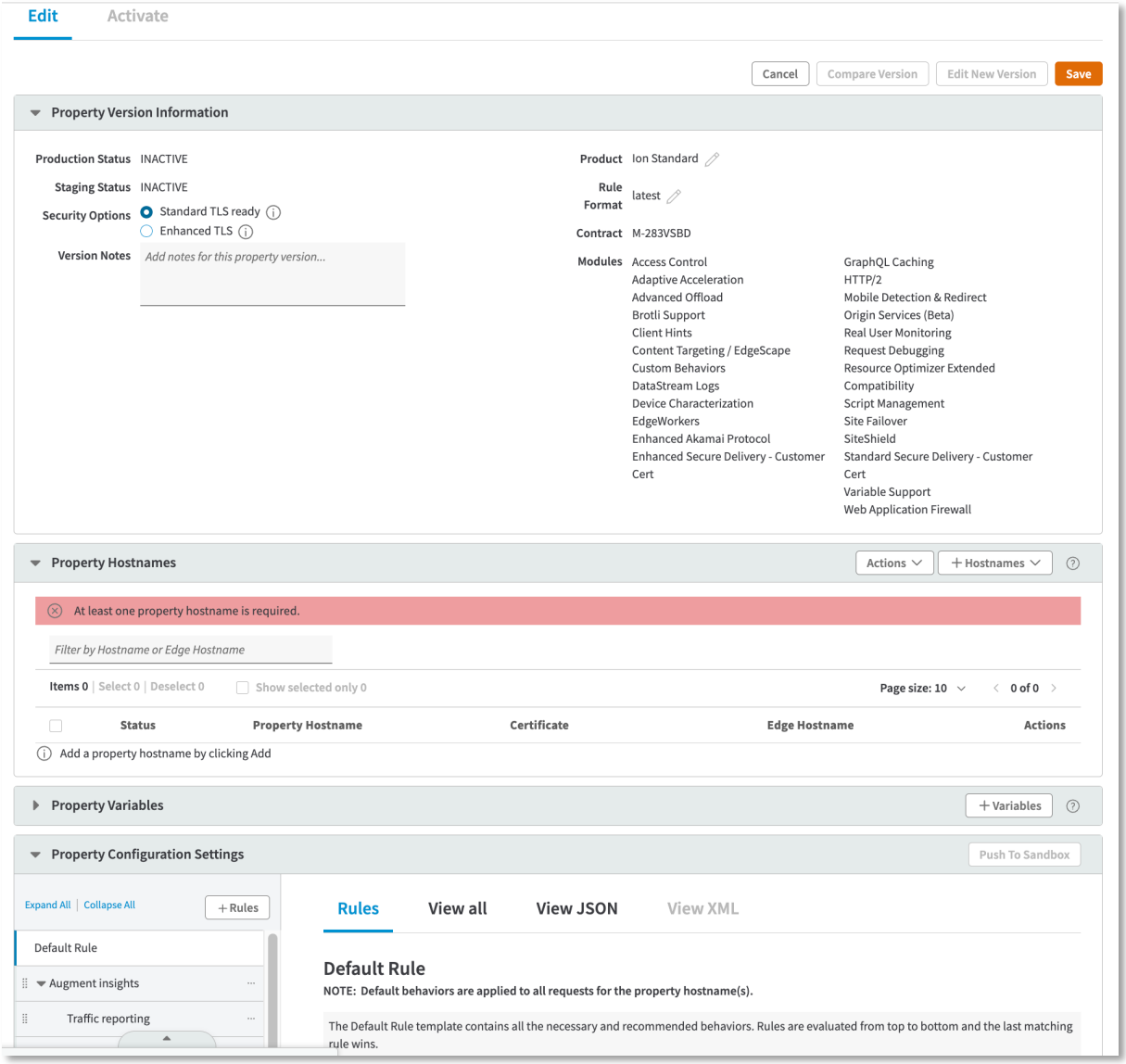
- In the Property Manager, click on the **"+New Property**" button.
- Select **“Ion Standard”** from the product list and click **“Create Property”**.
- The **“Set up Ion Standard”** page should be displayed



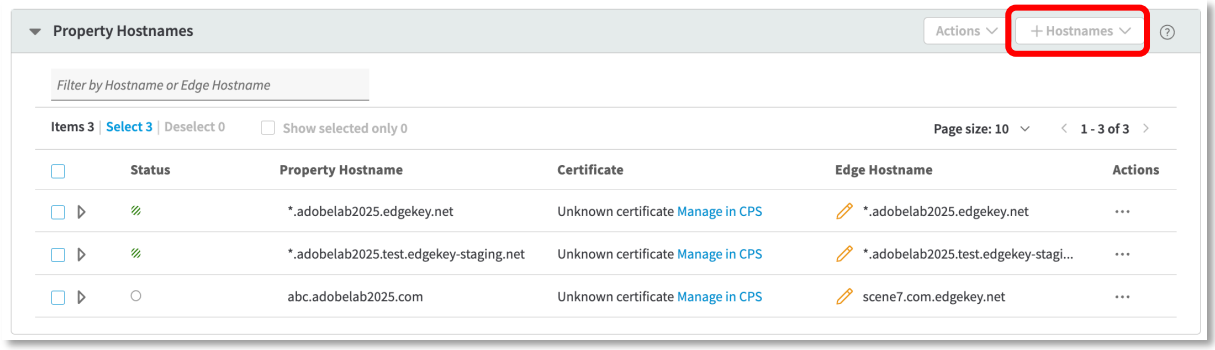
- In this page set a **“Property name”**, select the **“Contract”** for your organization and click on the **“Next”** button

Step 4: Configure Property Settings

- In the **Edit** property configuration page, you will see various settings and options.

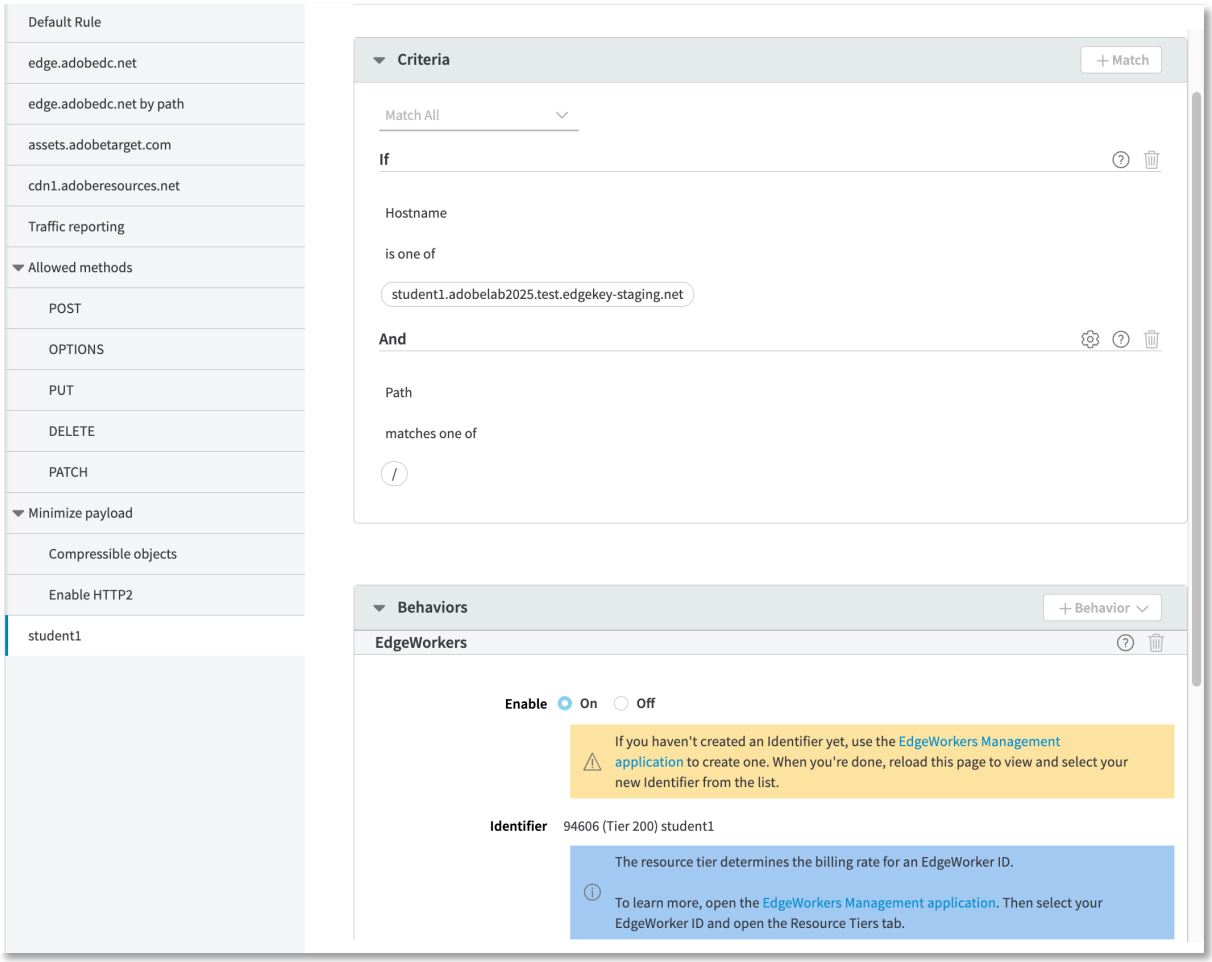


Step 5: Set up Hostnames



Hostnames section define the origin **Hostnames** URLs on which the Akamai Property applies, the associated **Certificates** and the **Edge Hostnames** corresponding to the origin hostnames. When defining hostnames wildcards are accepted.
You can add new Hostnames by clicking on the **"+Hostnames"** button.

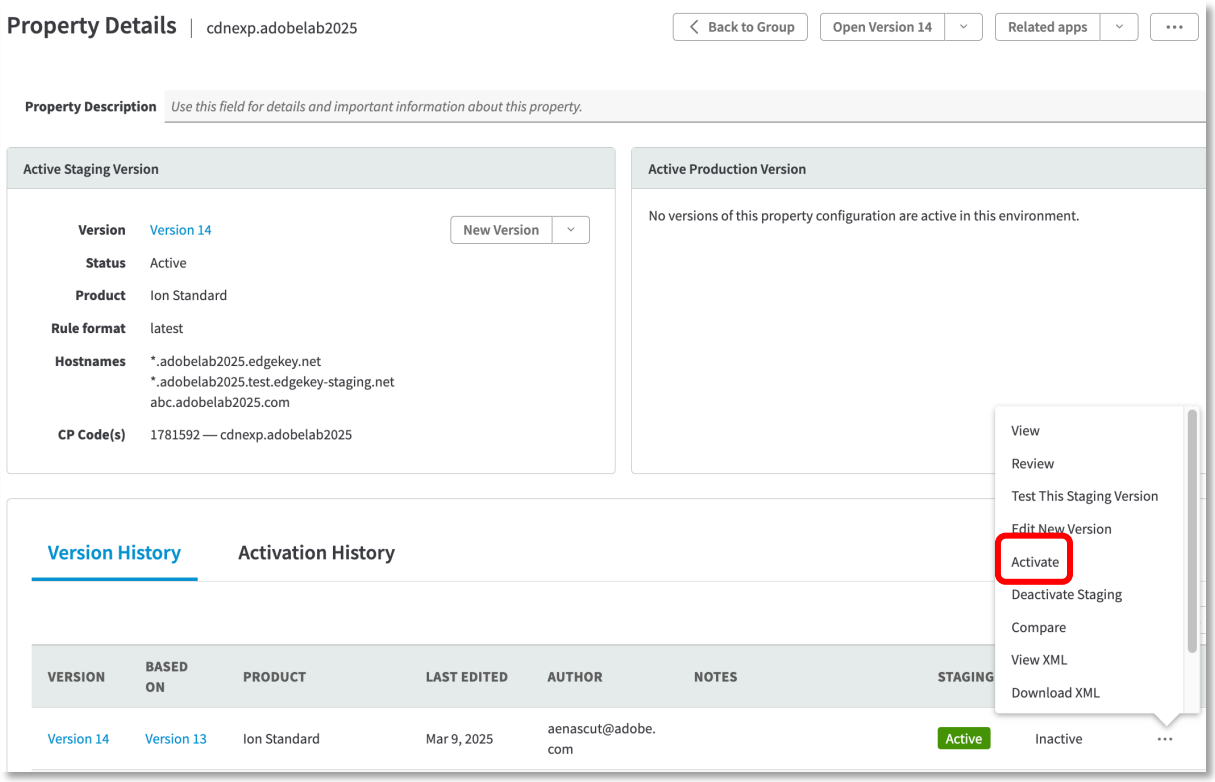
Step 6: Set Up Rules and Behaviors



- In the Property configuration page, go to the "**Property Configuration**" section. In here you can define the Behavior of the property by adding **Rules**, configuring **Criteria** and **Behaviors** for each rule.
- When a request is received by the CDN Edge nodes, the rules are evaluated sequentially, and the most specific rule will be the one applied.
- If you want to add a new **Rule** you need to click on the **"+Rule"** button from the left panel.
- In the **Criteria** section you can define the conditions which are evaluated for each **Rule**. The **Condition** is a Boolean expression.
- In the **Behaviors** section you can define the behavior when a Rule has been selected to be applied.
- In the above example when the requested URL hostname matches the value student1.adobelab2025.test.edgekey-staging.net and the path is "/" then the **"student1"** rule will be selected. In this case the Behavior is to route the request to the EdgeWorker having the ID equal with **94606**.
- Click **"Save"** to persist the configuration of the **Property**.

Step 7: Activate the Property

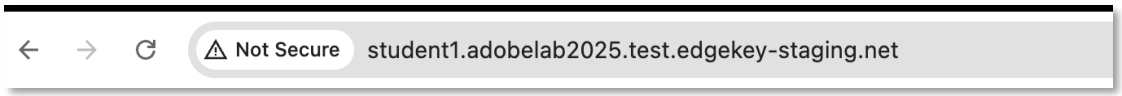
- Once you have configured all the necessary settings, when you get back to the **Property Details** page you will see a new version being available. Click on the **"Activate"** button from the contextual menu.



- Select the environment (e.g., **Staging** or **Production**) where you want to activate the property.
- Review the configuration and click "**Activate vxx on Production"** to deploy the new Property version.

Step 8: Test the Configuration

- After activation, test the configuration by accessing the external URL through your Akamai property.

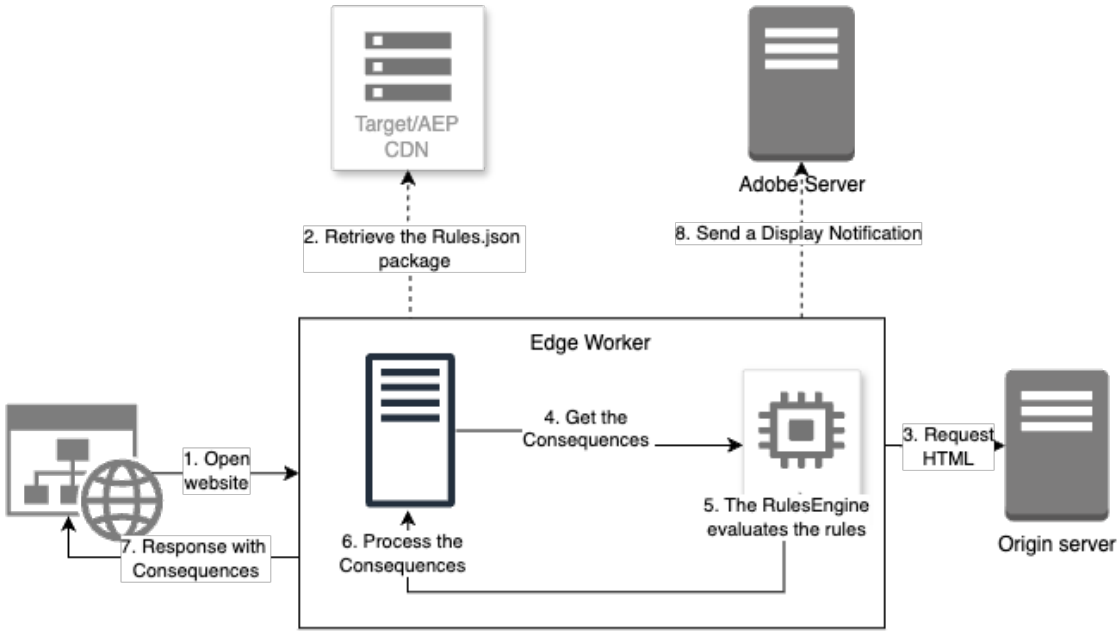


- Ensure that the requests are properly routed, and that the external URL is accessible.

Part 3: Coding with CDN Experimentation SDK

Overview of CDN Experimentation Solution:

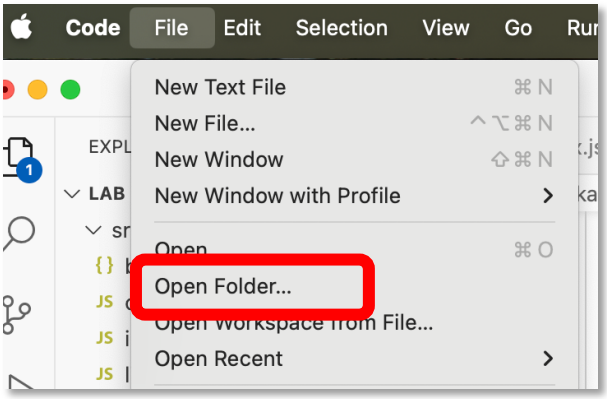
1. The visitor accesses the website, initiating a request to the CDN edge.
2. The rules package is either fetched or supplied.
3. The HTML page is retrieved from the origin.
4. The rules are assessed to determine the outcomes.
5. Consequences are applied.
6. Personalized page is displayed
7. Event reports are dispatched.



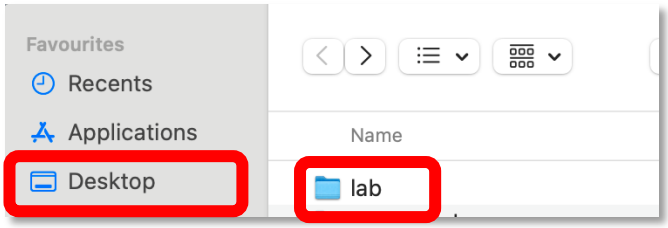
Exercise 1: Open the Lab Project in the IDE

Step 1: Open the **Visual Studio Code** IDE

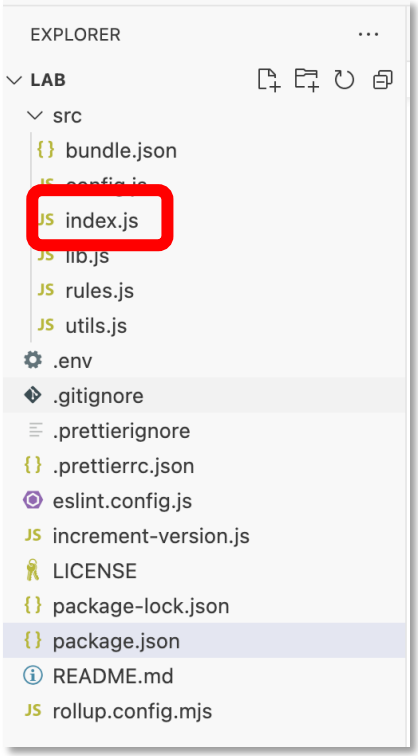
Step 2: From the Menu bar select **File/Open Folder** action



Step 3: From the **Desktop** folder select the **lab** subfolder containing the lab code



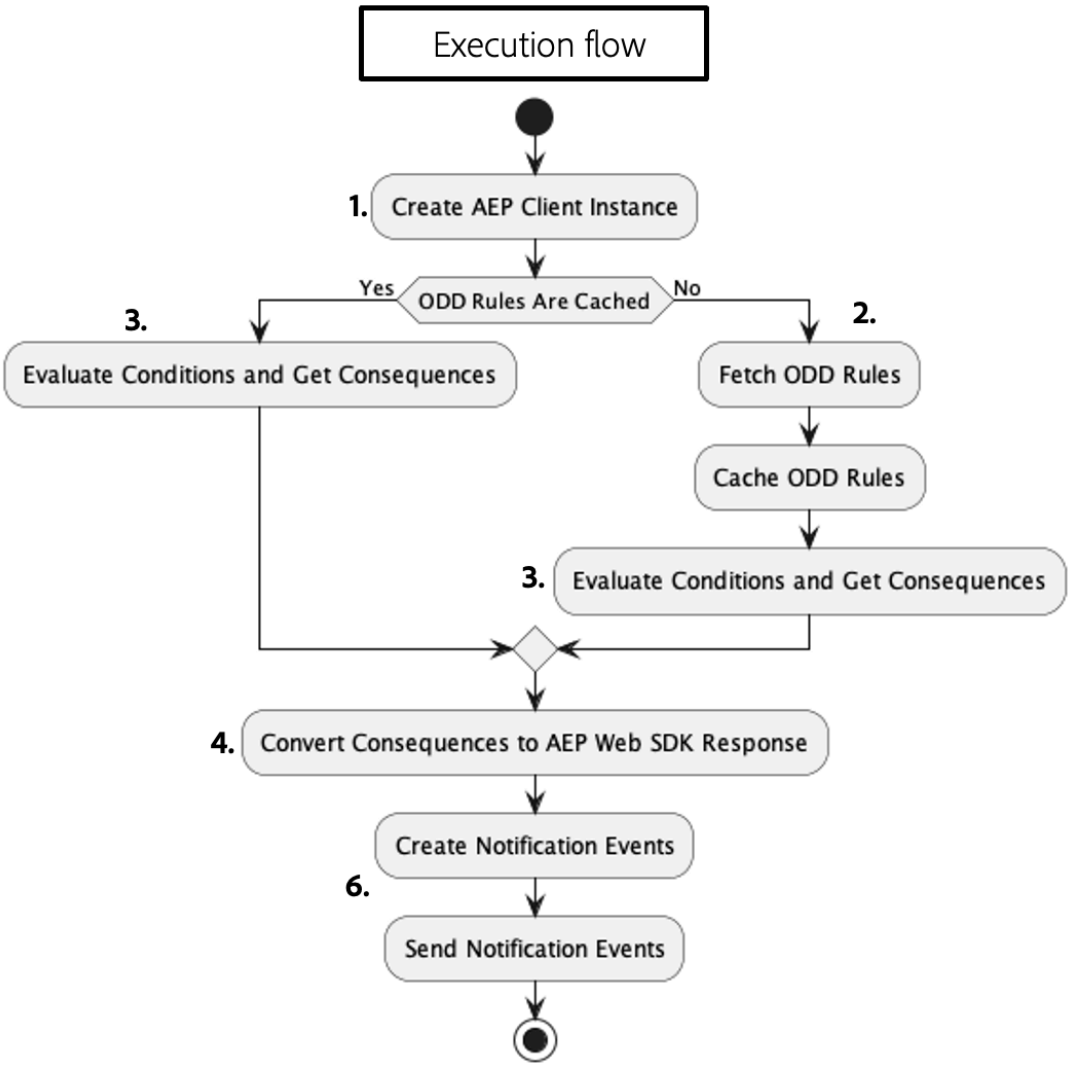
You should see the following project structure in VS Code **Explorer** window:



Step 4: Click on the **index.js** file to open it in the Editor pane.

EdgeWorker execution process:

1. The SDK Client is set up and configured (for example, with the rules bundle CDN address).
2. It fetches and stores the rules package.
3. The **rules** are then assessed, and the applicable **consequences** are determined.
4. These consequences are transformed into a response compatible with AEP Web SDK.
5. Based on this response, the SDK user can decide whether to apply it (using *html-rewriter*), integrate it into the HTML page, or utilize it for other personalization needs.
6. The SDK user has the option to notify the Adobe Edge Network once the response has been ”consumed" (to ensure accurate metric tracking).



Exercise 2: Creating the CDN Experimentation SDK Client Instance

The following snippet demonstrates the instantiation of the CDN Experimentation client with the specified arguments:

```
import { Client } from "@adobe/cdn-experimentation-akamai-sdk";
import { rules } from "./rules.js";
import { PROPERTY_TOKEN } from "./config.js";

export async function responseProvider(request) {
  try {
    const client = await Client({
      datastreamId: DATASTREAM_ID,
      orgId: ORG_ID,
      propertyToken: PROPERTY_TOKEN,
      oddEnabled: true,
      rules: rules,
    });

```

Step 1: Into the **index.js** file add the following lines of code, inside the `responseProvider` function, to create and configure the SDK client:

```
const client = await Client({
      orgId: "82C94E025B2385B40A495E2C@AdobeOrg",
      datastreamId: "bdb5cb8a-4496-4abd-8afc-e9396c1b1c27",
      propertyToken: "",
      oddEnabled: true,
      edgeDomain: "abc.adobelab2025.com",
    });
```

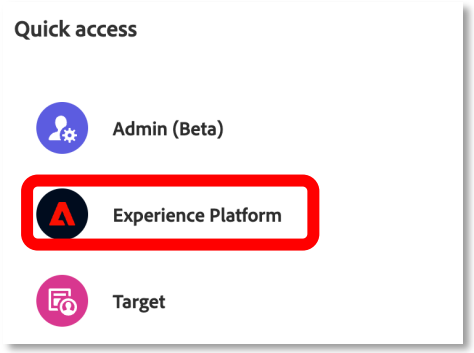
Step 2: orgId: it represents the IMS Organization Id.

To retrieve the IMS Organisation Id you need to navigate to the Adobe Target – **Administration / Implementation** section, **Account details,** select and copy the value for the org Id:

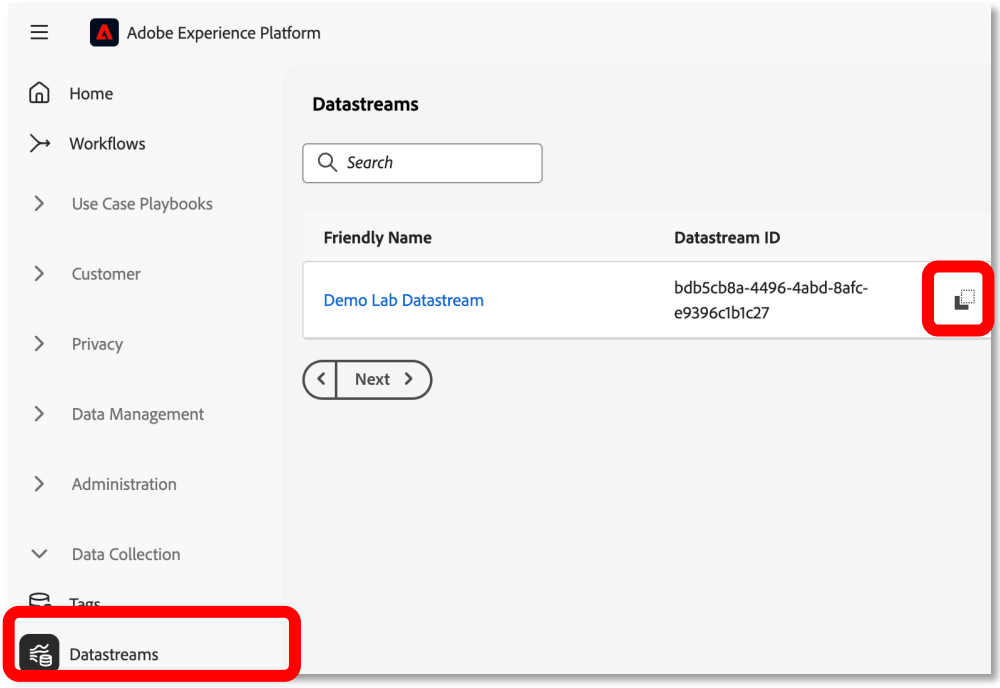
The copied value should be set as the value for the orgId parameter in the previous code snippet. In the case of the lab exercise the orgId is: 82C94E025B2385B40A495E2C@AdobeOrg

Step 3: datastreamId: the datastreamId from the Adobe Experience Platform – Datastream section. connected to Adobe Target (e.g. `bdb5cb8a-4496-4abd-8afc-e9396c1b1c27`).

Step 3.1: Navigate back to the **Adobe Experience Cloud Home** page and select **Experience Platform** solution from the **Quick Access** panel:



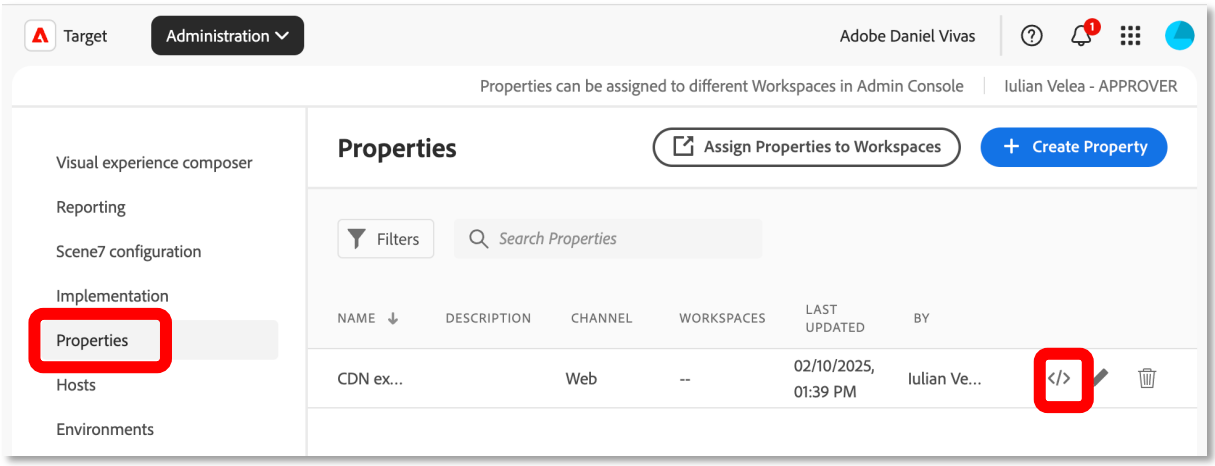
Step 3.2: Select the **Datastreams** section from the left Panel and then, in the right **Datastreams** list click on the **"Copy to clipboard"** button associated with the datastream we want to use, to copy its **datastreamId**:



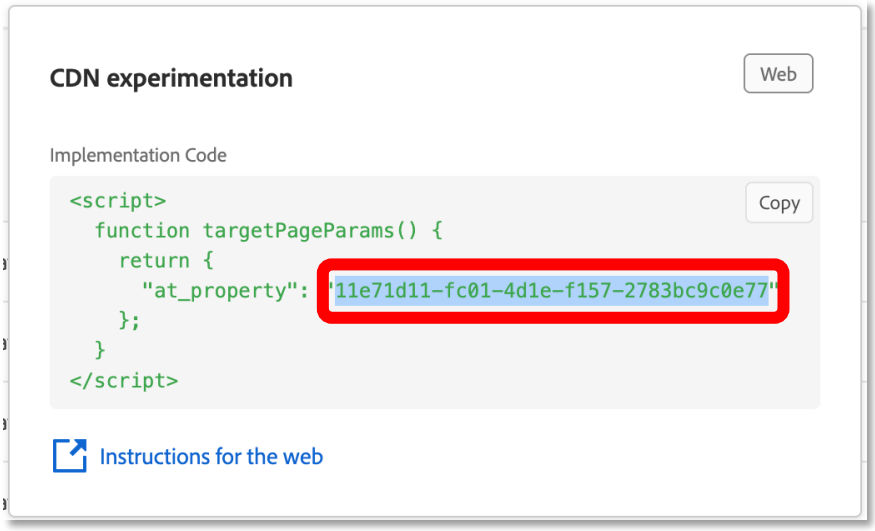
The copied value should be set as the value for the datastreamId parameter in the previous code snippet.

Step 4: propertyToken: Adobe Target property token associated with the activity used to reduce the bundle size by selecting only the activities associated with it.

Step 4.1: To retrieve the **propertyToken** you need to navigate to the **Adobe Target Administration** page, **Properties** section.
Locate your property associated with the seat number (e.g. L537-seat01) and click on the **Code button** (</> icon):



Step 4.2: From the **Code** dialog select and copy the property token value: **Todo**



The copied value should be set as the value for the propertyToken parameter in the previous code snippet.

Step 5: oddEnabled: true/false;

- **true** – will do the decisioning inside the edge worker.
- **false –** the library will act as a proxy to the Adobe API and call the */interact* endpoints for each decision request. No local decisioning is made.

For the lab exercise oddEnabled parameter value needs to be set to true.

Step 5 [not required]: rules: object containing the rules retrieved from the Adobe servers in AEP format. The rules can be retrieved by the developer and passed to the SDK client

as a configuration parameter. If the rules are not provided, the SDK Client will retrieve them from the Adobe servers.
For the lab exercise the rules bundle will be automatically fetched.

Evaluation and preparation of the rules

The **SDK Client** will return an object that contains the rules ready for evaluation and execution. If there is any problem the **Client** will throw an error.
The types of errors that can appear can be related to

- the library not being able to fetch the rules (might be a configuration problem in Akamai)
- the rules are not in the correct format
- the configuration Id's are missing or incorrect
- ...other problems

Exercise 3: Retrieving the consequences

Upon receiving a client request, the SDK will assess the request and determine the implications. The **Event** must be built on the EdgeWorker in the XDM format.
In the code sequence below, we are constructing an event that enables decision-making based on the visitor allocation to the available Experiences, using the ECID cookie value, and the URL, by passing in the address.

```
12
13      const address = `${request.scheme}://${request.host}${request.url}`;
14      const event = {
15        "type": "decisioning.propositionFetch",
16        "personalization": {
17          "sendDisplayEvent": false
18        },
19        "xdm": {
20          "web": {
21            "webPageDetails": {
22                    "URL": address
23            }
24          }
25        }
26      }
27      const consequences = await client.sendEvent(event);
28
```

Step 1: The **client.sendEvent** function will receive the **event** and respond with the consequences.
In the **index.js** file go to the section where the event is created and add the following line of code to retrieve the consequences:

```
const consequences = await client.sendEvent(event);
```

Exercise 4: Sending the Display event

Step 1: To track displayed events, we need to send a notification to Adobe servers. Typically, this can be done in a fire-and-forget manner, as the result is not needed.
Set **"sendDisplayEvent: true"** in the **event** passed to the **sendEvent** function to let the library automatically send the event.

Exercise 5: Working with the consequences

The consequences provided by the **client.sendEvent** function can be applied in the Edgeworkers code for A/B testing the origin website.
In the following code example, we are using the **HtmlRewritingStream**, extracting the consequence items, and applying the content of the consequence on the Origin response stream.

```
37
38    const consequenceItems = consequences.handle
39    .filter((payload) => payload.type === "personalization:decisions")
40    .map((consequence) => consequence.payload.map((payload) => payload.items))
41    .flat(2);
42
43    consequenceItems.forEach((item) => {
44      switch (item.schema) {
45        case "https://ns.adobe.com/personalization/json-content-item":
46         const { content } =  item.data;
47          // match the DOM element based on the CSS selector
48          streamRewriter.onElement(content.selector, (el) => {
49            // example of working with the consequence
50            el.replaceWith(content.payload);
51        });
52          break;
53      }
54    });
55
```

Step1: Go to the line `// Exercise 5. Apply the consequences to the origin stream` and add the following lines of code to instantiate the **HTMLRewritingStream**

```
const streamRewriter = new HtmlRewritingStream();
```

Step 2: Go to the **forEach** loop in the **index.js** file and, after the line containing the comment `// match the DOM element based on the CSS selector`, add the following code sequence:

```
streamRewriter.onElement(proposition.selector, (el) => {
            el.replaceWith(proposition.payload);
        });
```

Exercise 6: Returning the processed response

After creating the HTML rewriter processing stream, we need to return the Response to the client and the EdgeWorker code will apply the changes on the returned page.

```
61
62      return createResponse(
63        200,
64        {
65          "Powered-By": ["Akamai EdgeWorkers" + Date.now()],
66        },
67        originResponse.body.pipeThrough(streamRewriter),
68      );
69
70
```

Step 1: In the **index.js** file go before the line marking the end of the **catch** block and add the following code snippet:

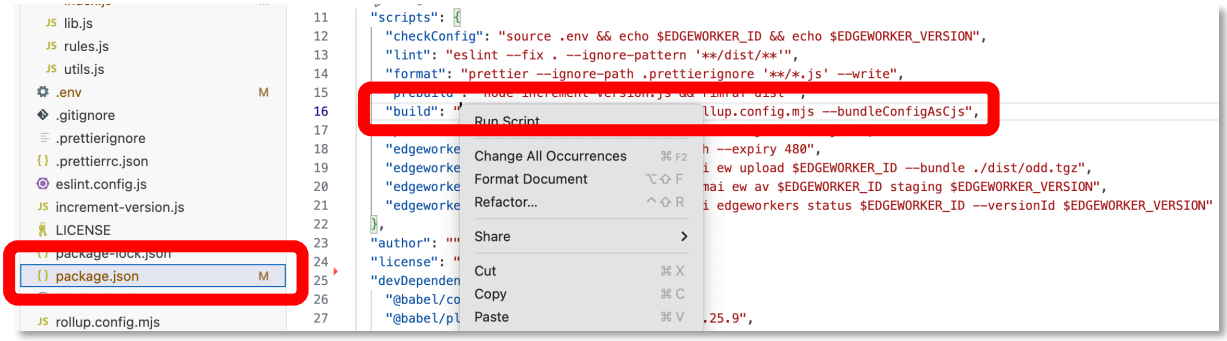
```
      return createResponse(
            200,
            {
              "Powered-By": ["Akamai EdgeWorkers " + new Date().toString()],
            },
            originResponse.body.pipeThrough(streamRewriter),
          );
```

Exercise 7: Deploy your code to the EdgeWorker

Step 1: Set the EdgeWorker ID
Open the **.env** file in the IDE and add the **EdgeWorker ID** provided for your seating number (e.g. 94606) as the value for the `EDGEWORKER_ID` variable:

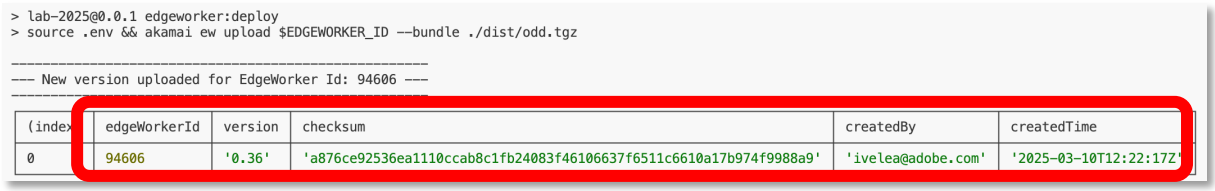
Leave the `EDGEWORKER_VERSION` unchanged as this will automatically be update during the **prebuild** step.

Step 2: Build your project
In the Visual Studio Code left navigation panel select the **package.json** file, go to the **scripts** section, and run the **“build”** script:

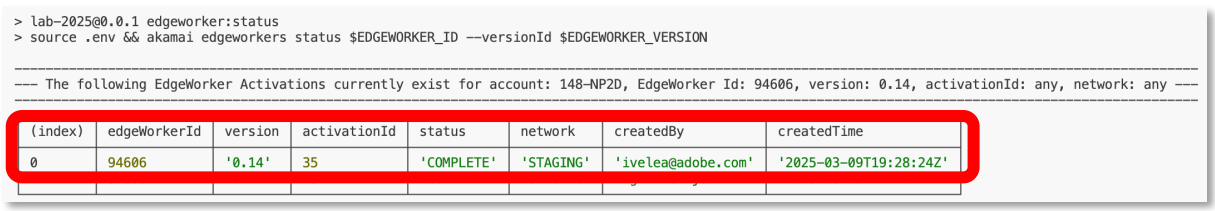


There should be no errors reported in the terminal:

Step 3: Deploy your code
From the **package.json** file go to the line with the **“edgeworker:deploy”** script and run it. If the deploy is succesful you should see the following message in the terminal window:



Step 4: Activate your deployment
From the **package.json** file go to the line with the **"edgeworker:activate"** script and run it in order to activate the new package deployed on the EdgeWorker.
For checking the status of the deployment you can run the **“edgeworker:status”** script. After the activation has been completed, this command should display in the terminal panel the **Complete** value in the **status** column. This usually takes a couple of minutes depending on the size of the new build package:



Exercise 8: Test your new deployment

Step 1: Test the newly activated code package
Open a new Browser Tab and enter the ***seatXX***.adobelab2025.test.edgekey-staging.net value in the address bar, depending on your seat number.
The page should be loaded and the result of the EdgeWorker processing should be embedded in the displayed version. Depending on the performed user allocation, you should be served with the result of applying Experience A or Experience B defined in the Target Activity.
To get the result of the other Experience you can open a new Incognito browser window and access the same URL.

Additional steps:
Exercise 9: Persisting the ECID cookie value

Adobe Target identifies unique visitors based on the ECID value that it's generated in the server. The ECID value provides a stable Experience selection and reporting.
The ECID value need to be persisted in the browser for each visitor in Cookies.

When the requests arrive from the browser we will see if the ECID cookie is present and add it to the Event object used for deciding the Experience.

```
// Exercise 10. Extract the ECID from the request cookie and if it's present add it to the request
const cookies = new Cookies(request.getHeader( name: "Cookie"));
const ecid = cookies.get("X-ADOBE-ECID");
if (ecid) {
  event.xdm.identityMap = {
    ECID: [
      {
        id: ecid,
        authenticatedState: "ambiguous",
        primary: true,
      },
    ],
  }
}
```

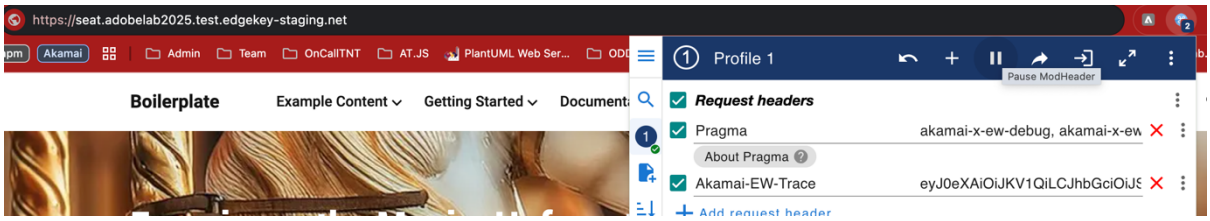
When returning the response, we make sure that the ECID cookie is set so that we persist it across the sessions.

```
25    // Exercise 10. Persist the ECID in a cookie in the browser
26    const ecidNamepace : T  = consequences.handle.find(
27        (payload : T ) =>
28            payload.type === "identity:result" &&
29            payload.payload[0].namespace.code === "ECID",
30    );
31    const ecidValue = ecidNamepace.payload[0].id;
32    streamRewriter.onElement("body", (el) : void  => {
33      el.append(
34          `<script> document.cookie = "X-ADOBE-ECID=${ecidValue};SameSite=None; Secure"; </script>`,
35      );
36    });
37
```

Additional steps:
Exercise 10: Debugging your new deployment

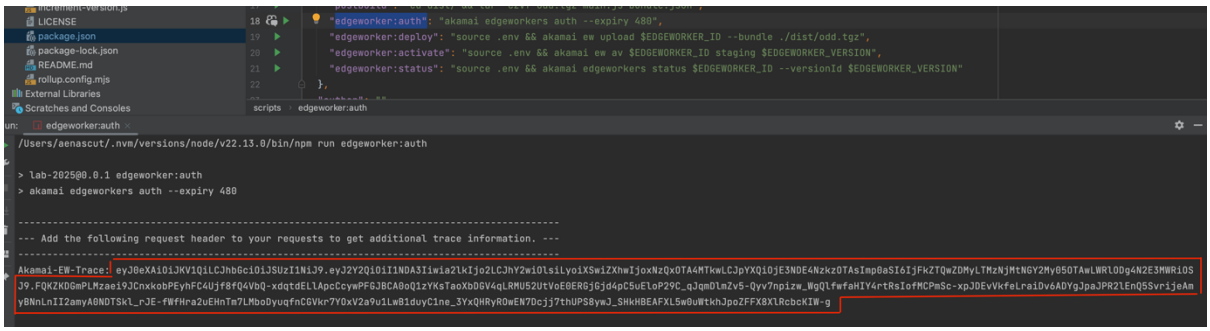
Akamai Edgeworkers communicate the log messages through the response headers attached to the response. To activate the debug functionality, we need to set two headers to be sent along with the webpage request inside the ModHeader chrome extension.
Step 1) Inside Google Chrome find the ModHeader extension

Step 2) The **Pragma** header specifies the type of logs and debug information that will be returned. For the Lab set the header name Pragma and the value the following list: *akamai-x-ew-debug, akamai-x-ew-debug-rp, akamai-x-ew-debug-subs, akamai-x-ew-subworkers*
For a reference of values please see [the Akamai documentation]



Step 3) Go to the proejct and run the command "**edgeworker:auth**" from the **package.json** file. This will provide you with an authentication token. Copy paste just the token value highlighted in the screenshot below.
Step 4) Go to Google Chrome and open the ModHeader extension. Set the Header name **Akamai-EW-Trace** and the value copied from the console in step 3. This will allow the Edgeworker to output the log in the response headers.

Copy paste the token the red border inside the ModHeader chrome extension.



Appendix

Edge worker runtime environment limitations

- https://techdocs.akamai.com/edgeworkers/docs/resource-tier-limitations

responseProvider	Maximum memory usage for responseProvider	2 MB	4 MB	6 MB
	Maximum CPU time for responseProvider	100 milliseconds	200 milliseconds	300 milliseconds
	Maximum wall time for responseProvider	4 seconds	8 seconds	10 seconds
	Maximum number of HTTP sub-requests allowed for responseProvider	50	50	50
	Maximum number of HTTP sub-requests allowed in parallel for responseProvider	5	5	10
	Maximum wall time per HTTP sub-request during the execution of the responseProvider event handler	4 seconds	8 seconds	10 seconds
	Maximum response size per HTTP sub-request during the execution of the responseProvider event handler	5 MB	5 MB	8 MB

- https://techdocs.akamai.com/edgeworkers/docs/limitations

Compressed size for a code bundle	512 KB
Uncompressed size for a code bundle	1 MB
Maximum response header size for HTTP sub-requests	8192 bytes
Maximum body size for responses created using the `respondWith()` method	2048 characters
Maximum body size for responses from origin to an EdgeWorkers function, using `httpRequest()`	128 KB when using `json()` or `text()` to buffer the entire response. 5 MB when using `body` to read the response as a stream.
Maximum body size for responses from an EdgeWorkers function to a browser when the response is passed through as a string	100,000 bytes if you pass a string to `createResponse` in `responseProvider` No direct limit if you pass a stream to `createResponse` in `responseProvider`.
Maximum body size for responses from an EdgeWorkers function to a client browser	2048 characters if you use `request.respondWith()` in the `onClientRequest` or `onClientResponse` event handler.
Maximum body size when `request.text()` or `request.json()` is called in the `responseProvider` event handler	16 KB
Maximum streaming request body size when `request.body` is called in the `responseProvider` event handler	1 MB
Maximum size of the response status and header for responses generated using `createResponse`	8 KB

Workflow for the rules package:

1. A user of Adobe Target creates **Activities**.
2. The rules package is uploaded to an AWS S3 bucket.
3. Content from the S3 bucket is delivered through a CDN.
4. The rules package is accessible for download via the adobe.io API.