Cloud Embed User Guide

April 27, 2022

Contents

Welcome to Akamai Cloud Embed

Akamai Cloud Embed (ACE—formerly Wholesale Delivery) provides the features and APIs necessary for you, acting as our partner, to deliver the benefits of a content delivery network (CDN) to your customers ("subcustomers") at scale.

With a combination of the Property Manager tool in Control Center and the ACE API, you can configure CDN features per domain and give a subcustomer the ability to purchase, configure, and monitor Akamai's CDN services directly through your custom portal.

What is a CDN?

At its most basic level, a content delivery network (CDN) can greatly improve the end-user experience of a website or application by moving cacheable content closer to that end user.

Akamai Cloud Embed (ACE) CDNs use our Edge network servers to cache and deliver this content. There are definite benefits to delivering content through our Edge servers:

- You'll see faster delivery of content to the end user
- You'll accomplish offload of processing from the origin server
- You'll get reduced egress bandwidth from the origin server.

When content is delivered from a server near the end user, the shortened distance reduces latencies. As a result, duplicate requests for the same content (for example, 10,000 end users all requesting the same video of a cat) can be handled at the Edge servers. So, the request load and bandwidth use at your origin server is greatly reduced. This frees the origin server to handle more unique requests, or requests for dynamic content.

How is the CDN put in place?

Put simply, you place the Edge network between the end users and the origin server through a simple reassignment of the domain name for website or application your customer (our "subcustomer) owns. For example, imagine the mapping from the website domain to the origin server IP was direct, like this:

Website hostname	Origin server IP
www.example.com	93.184.216.34

The Edge network can be inserted by generating a CNAME of the website to the Akamai domain, which points the end user request to the Akamai Edge server IP, rather than to the origin server.

Website hostname	Akamai edge hostname	Edge server IP
www.example.com	www.example.com.akamaiedge.net	184.51.102.67

The Akamai Edge server then uses a separate hostname when addressing the origin server:

origin hostname	origin server IP
origin-www.example.com	93.184.216.34

How are CDN behaviors configured?

When an end-user request arrives at the Akamai Edge server, the server understands how to handle the request by consulting a "base configuration" you've set up for your customers' websites or applications, and a "delivery policy" you've set up comprised of settings for a specific subcustomer."

The Edge server looks at the `Host:` header value in the request, and then finds the corresponding property configuration file for that website or application. You set up this configuration using the Property Manager in Control Center (recommended), or via the Property Manager API. You apply general rules for caching, authorization, origin server location, and other services, that apply to all subcustomers.

You also set up a "delivery policy" for each subcustomer, to define match criteria and behaviors to apply to requests that come in for that specific subcustomer's website or application. This is done via the ACE API.

You should be familiar with these terms

Before you get started with Akamai Cloud Embed (ACE), you should be familiar the following concepts and terms.

ACE-specific concepts and terms

These terms and concepts are specific to Akamai Cloud Embed and its usage, and are called out throughout this documentation. Terms are ordered based on their importance.

Concept/Term	Description
Cloud Partner	These are Akamai resellers or partners who provide delivery services to their customers. As a user of ACE, you are a cloud partner. (This documentation is focused toward the cloud partner.)
subcustomer	As a cloud partner, *your customers* are subcustomers. Akamai does not assign individual Content Provider (CP) codes to subcustomers even though their traffic is sent over the Akamai platform. A cloud partner has access to usage detail reports for each subcustomer, based on the subcustomer IDs you provide Akamai.
subcustomer ID	This is a unique ID controlled by you, as our cloud partner, that you establish for each subcustomer within your own billing systems. All traffic for a particular subcustomer is rolled up by this ID for billing purposes.
Base Configuration	This configuration includes all of the common rules used to process end-user requests via your subcustomers. You use the Property Manager in Control Center (or the Property Manager API) to set up this configuration. This is similar to an individual "Property Configuration" that you set up

Concept/Term	Description
	to use any of our other delivery products (AMD, DD or OD). However, with ACE, a base configuration serves as a single configuration that you apply for use with multiple subcustomers. **Note:** The terms "base configuration," "property configuration," "configuration," and "configuration file" are used interchangeably to describe this component.
Subcustomer behavior	In Property Manager, this is the specific behavior you use to control which individual ACE (and optionally ICA) features are available to handle your subcustomers' traffic. You can set up this behavior to provide access to all available features, or you can select a subset of features to define different classes of service.
Policy	Policies determine how Akamai Edge servers handle a given subcustomer's requests. A single policy is bound to a specific property hostname, that has been defined in a specific base configuration. Your policy JSON is made up of rules, which contain both match criteria and behaviors. When an incoming request meets the match criteria in a rule, it triggers the behaviors listed in that rule. Within a policy, rules must be unique: they can't have identical sets of matches. A policy can also contain up to 100 behaviors.
Policy Rules	These rules include both policy match criteria and policy behaviors. When an incoming request meets the match criteria in a rule, it triggers the behaviors listed in that rule. You can use each match type and each behavior once in a policy rule. Also, no one rule can contain both a whitelist and blacklist behavior of a given type. For example, you can add an IP whitelist and a referrer blacklist to a rule, but you can't have both an IP whitelist and IP blacklist in the same rule. Rules are applied from top to bottom and should be listed from least restrictive (top) to most restrictive (bottom). For example, you would list a match on `url-wildcard` value `/*` *first*, because it would apply to all requests, where `/images/*` would only apply to a subset of requests.
Policy Matches	Within a policy rule, a policy match defines which subcustomer requests receive the policy behaviors within the rule. All match conditions for a rule must evaluate to "true" in order for the behaviors to apply. When constructing matches in

Concept/Term	Description
	a rule, the type of data you enter depends on the type of match. For example, if you use the query string match, you have to enter the exact string and keep case sensitivity in mind. If you use the `url-scheme` match, you enter either `HTTP` or `HTTPS`. Within a match, you cannot repeat entries in the value string.
Policy Behaviors	Policy behaviors are used to encapsulate customizable settings. Behaviors are a part of a policy's rules. A rule can have many behaviors or only one. With ACE, you group rules into policies. You can have a maximum of 100 behaviors in a subcustomer policy. Policies exceeding 100 behaviors are rejected upon submission. Within a behavior, you cannot repeat entries in the value string.
Content Characteristics-Dynamic Web Content Behavior	Include this behavior if you've included acceleration via ICA to your ACE instance, and you only want to enable specific ICA optimizations for your subcustomers. (Contact your Account Representative for more information on ICA.)
InstantConfig Behavior	If your subcustomers only send HTTP traffic, you have to add this behavior to your base configuration. This behavior, also known as "Multi-Domain Configuration," lets you associate multiple web assets to a single property without adding each hostname separately. It applies property settings to all incoming hostnames based on a DNS lookup.

Generic Terms

These terms describe general components used throughout Akamai, and also apply to ACE.

Concept/Term	Description
Content Provider (CP) Code	This is an identifier for a particular subset of content on your Origin Server. Some features are applied individually to particular CP codes. CP codes are also used to provide additional granularity in reports and to track billing. You'll have at least one CP code per contract. Also, a CP code is associated with a single contract.
Property Configuration	See *Base Configuration*.
Property Hostname	This is an identifier that is used to determine which base configuration file to use when processing end-user HTTP/HTTPS requests for a subcustomer's resource. Often, the Property Hostname is the fully qualified domain name of the endpoint subcustomers use to access your cloud

Concept/Term	Description
	partner CDN. For instance, `www.mycdn.com` would be your endpoint domain name *and* your Property Hostname. A property hostname is sometimes referred to as a "digital property," too.
DNS Record	This is a record that associates a Domain Name to an IP address.
Edge Hostname	This describes your specific cloud partner subdomain on an Edge network domain. It maps incoming requests to our Edge servers. You generate a "property hostname to Edge hostname association" to determine an Edge hostname. For example, assume the endpoint that subcustomers access is `www.mycdn.com`. This would serve as the property hostname in this association that results in the Edge hostname, `www.mycdn.com.akamaized.net`. An end-user request for a subcustomer's assets is ultimately directed to this Edge hostname. In turn, the Edge hostname resolves the request to individual servers on the Edge network, where the associated base configuration is read to determine how to process the request.
Edge Server	This is a server in the Akamai Edge network that ultimately receives end-user requests from subcustomers. (End user requests to the subcustomer's hostname are redirected to an associated Edge hostname which resolves to the Edge Server.)
Forward HOST Header	The name the Edge server forwards (often to the Origin Server) in the HTTP Host request header, sometimes also referred to as the expected host. Often, this is the same name as the Property Hostname.
Origin Server	The server where subcustomer content is housed to be served to end users (where your site/ application resides). Edge servers read first from the applicable subcustomer's policy for origin information, and second from the Origin Server settings applied in your base configuration.
Origin-server Hostname	The Hostname that maps to your Origin Server from which Edge Servers retrieve your content. The usual syntax of the Origin-server Hostname is the Property Hostname preceded by “origin” and a hyphen.

The Akamai Cloud Embed workflow

Configuring Akamai Cloud Embed (ACE) for your customers involves various steps, that need to be performed in a specific order.

Review the table that follows to get an idea of the tasks required, and order they need to be performed.

Step	Description
Step 1: Enable ACE	Contact your Account Representative to enable ACE for your account.
Step 2: Determine the delivery method	Are subcustomers delivering content via HTTP or HTTPS? • **HTTPS (and HTTP, if applicable) traffic**: Contact your Account Representative to create a secure ACE property and provision the necessary certificates. This is the preferred configuration for Akamai Cloud Embed. • **HTTP traffic, only**: Contact your Account Representative to request a Multi-Domain Edge Hostname and create an InstantConfig property. A Multi-Domain Edge Hostname lets you use a single hostname to represent multiple web assets. **Note:** This step only applies at this phase if you're using the Property Manager API to generate your ACE base configuration. If you're using the Property Manager in Akamai Control Center to generate the configuration, you can set all necessary options to define the delivery method there.
Step 3: Create an origin hostname	If you don't already have a hostname to identify the origin server separate from public hostnames, you need to create one. You do this by creating a DNS record for your origin IP. The origin hostname is used by Akamai edge servers to locate your origin server. For example, you might create the hostname `origin-www.example.com` to point to the IP address of your origin server, if your public hostname is `www.example.com`. **Note:** This isn't necessary if you're using NetStorage as your origin.

Step	Description
Step 4: Create a test hostname	Until you’ve completed configuration and testing of your service, you’ll use a test hostname to access the Akamai network. You do this through CNAME of a test hostname to the applicable Edge Hostname.
Step 5: Create the ACE Base Configuration	Here, you use the Property Manager in Control Center to create a configuration file that controls how all traffic from subcustomers is served via Akamai Edge servers. Most cloud partners only need one base configuration. So, we suggest that you contact your Account Representative for suggestions on the best configuration for your specific use case.
Step 6: Set up use of the ACE API	You need to obtain API credentials for use, and enable use of the API using the Control Center.
Step 7: Add APIs to Your Customer Portal	Integrate the applicable ACEAPIs into your customer portal, so that subcustomers can apply desired settings for delivery of their content.
Step 8: Other recommended tasks	There are tasks that are not strictly required to implement ACE. However, they are associated with your application, or are highly recommended. This includes things such as configuring log delivery and reporting.
Step 9: Testing	Test the end-to-end CDN functionality with subcustomers. You should use a test domain to validate both the functionality and the ACE base configuration.
Step 10: Going Live	For the final step to go live, you need to change the DNS to switch subcustomer websites or applications to the Akamai platform.

The base configuration vs. the delivery policy

In your ACE instance there are two separate "configurations" that need to be generated. Before you get started, you should know the difference between the two.

The base configuration

In the simplest of terms, this configuration defines which subcustomers can access your ACE CDN, and what those subcustomers can and can't do. (All subcustomers associated with a base configuration are subject to its rules and behaviors.) You use our Property Manager application (either in Control Center or via its API) to define these settings.

The delivery policy

This is a configuration that you generate for each specific subcustomer. Using the ACE API, you set up a delivery policy to include subcustomer-specifc rules to supplement the rules defined in the base configuration.

Set up a base configuration

The ACE base configuration refers to the settings and options you set up for use by our Edge network servers. Edge servers communicate with your origin server(s) to fetch, cache or process content, and subsequently serve that content to the requesting client.

We recommend you use the Property Manager in Control Center

You can use either the Property Editor interface in Control Center, or the Property Manager APIs (available via developer.akamai.com) to create this configuration. Typically, this is a one-time process. So, for ease-of-use, we recommend that you use the Property Manager UI in Control Center to set up your base configuration.

With this in mind, this document only covers the use of UI to create the configuration. (Use of this API is not covered here.)

How many configurations should I have?

Typically, as a cloud partner, you should only need one base configuration. However, you *can* have multiple base configurations to manage multiple permutations of your network settings. You can also have multiple base configurations to support the different use cases of your customers.

So, before you go on with this process, you may want to contact your Account Representative for suggestions on the best configuration for your specific use case.

Note: This documentation uses the terms **base configuration**, **configuration** and **configuration file** interchangeably.

Before you begin

There are several things you need to do before you get started. This includes various internal and external tasks, such as gathering relevant information used in creating the ACE property, generating a certificate for HTTPS delivery (if applicable), and other considerations.

Enable ACE for your account

Before you can begin, you need to get Akamai Cloud Embed added to your contract.

Contact your Account Representative to get it added. Work with your rep to determine all relevant needs. Your rep can also give you an estimate on how long it will it will take to fully provision ACE for access.

Do you want to add acceleration features?

If so, let your representative know that you want to add support for the Integrated Cloud Accelerator (ICA) feature.

You need to gather specific information

There are specific pieces of information you'll need while creating a base configuration.

Get your account information

Within five days of your service contract submission, you should receive your account information. If you haven't received it (or you don't have it), you can get it from Akamai Business Services (specialist@akamai.com).. *You need this information to activate your configuration for use with ACE.*

You should receive the following information:

- **Your account name and ID**
- **Content provider (CP) codes**. These are used for reporting and billing.
- **An administrator login**. This is used to access the following:
  - **Akamai Control Center**. *https://control.akamai.com*
  - **Content Storage Servers**. This applies if you've included storage (NetStorage) in your service package, for use as the origin to deliver content.
-

Additional items you'll need

In addition to information provided to you by Akamai, you need some other information to complete set up. (Some of these values are gathered during the processes discussed in this documentation.)

- **Know your Endpoint Domain**: This is your unique hostname, that you use to serve as your Akamai Cloud Embed CDN. (You need to have subcustomers CNAME to this endpoint—End-user requests to each subcustomer domain will be resolved to this endpoint, and ultimately to your Edge server hostname to access the base configuration.) This serves as the "Property Hostname" when generating a "property hostname to edge hostname" association for the base configuration.
- **Know the Origin Server Hostname**: This applies if you or a subcustomer is housing deliverable content in a custom origin. (You are not using NetStorage as your the origin.) This is used for Edge servers to fetch target content for a request.
- **Make note of your Edge Server Hostname(s)**: You generate a "property hostname to Edge hostname" association, that results in your Edge server hostname. This associates your endpoint domain with Akamai edge servers. This is done in the Property Editor, while setting up your base configuration. Your endpoint domain must later be CNAMEd to this Edge server hostname to finalize the process ("Go Live").

Secure delivery (HTTPS)

For secure access (HTTPS), Akamai Edge servers deliver via HTTPS, only if the client request is made over HTTPS.

You can create a custom certificate in Control Center

You don't need to provide an existing SSL certificate. You can create and activate a custom one via the Certificate Provisioning System (CPS) in Akamai Control Center, and then apply it in your ACE base configuration when creating it in Control Center. *This is the recommended practice for this process.*

Create the certificate before the configuration

When using the **Property Manager Editor** to create a Property Hostname for your base configuration, you'll notice a button you can use to create a new certificate. Don't use this. You'll need to wait for the certificate to complete provisioning before you can finish your base configuration. You may need to abort the configuration process if the certificate takes too long to provision.

To streamline the process, create the certificate first using the **Certificate Provisioning System (CPS)**, separately in Control Center.

Note: These instructions don't cover the full use of CPS, just what's required for this process. See the online help for CPS for complete details on its use.

1. Log in to Control Center using an administrator-level **User ID** and **Password**.
2. Select **CONFIGURE** > **Certificate Provisioning System**.
3. Click **Create New Certificate**.
4. In section **1 - Select Validation Type**, choose the desired level of validation. Keep in mind that the more extensive the validation, the longer it takes to provision the certificate.
5. In section **3 - Enter Certificate Information**, input all applicable hostnames ("vanity domains") you use for your website or application, in the **Common Name (CN)** field. (When a request originates from one of these hostnames, an Edge server will deliver content.)
6. Set the desired security level in section **6 - Select Network** settings:
   - **Deployment Network**: Select either **Standard TLS** (Standard: SOX and ISO compliant) or **Enhanced TLS** (Advanced: PCI, SOX, ISO and FedRAMP compliant).
   - **SNI-Only**: Set to "On" if you want to extend upon TLS. Ensure that your environment can support it.
7. Review and confirm creation of the certificate.

How long does it take for the certificate to provision?

The time it takes can vary, based on all of the settings you've applied for the certificate. Typically, a certificate with Domain Validation and Standard TLS applied can take 60 minutes to provision, but an Enhanced TLS certificate can take considerably longer, ranging from three to six hours.

The Control Center user account that created the certificate will receive an email when the certificate has complete provisioning.

Are you using your own SSL certificate?

You can't set up SSL service through the edge network without Akamai first receiving or obtaining your certificate information and setting up for it. You'll need to work with your Account Representative to do so.

You can use our shared certificate

We offer a secure certificate for HTTPS delivery (you don't have to create a custom one). However, this requires that you use our specific "shared certificate hostname" in the URLs that must be provided to end users to access content. You define your own unique property hostname to add to the shared certificate hostname to build this URL. (For example, "mypropertyhostname.akamaized.net.") This is all accomplished via the generation of a Property Hostname in your ACE configuration.

The SSL certificate must exist on all origin servers

If you are using a custom origin (not NetStorage), you need to maintain a valid SSL certificate on *all* origin servers that will be contacted by the Edge network, if you want to maintain HTTPS security throughout the request. (This includes the client to Edge server, then origin to client for delivery.) The SSL certificate name should be the same name as your Property Hostname.

You need these advanced behaviors

Before your Akamai Cloud Embed base configuration is complete and ready for production traffic, you need to add some Advanced Behaviors to it.

Work with your Account Representative to ensure that all of the following have been applied:

- Incorporate the FOSSL hack. This supports the real origin domain in the origin certificate by adding the origin domain as an allowed CN.
- Set the appropriate PARTNER_DOMAIN_SUFFIX variable definition.
- Set the `reporting:log.product-info` to "On."
- Set the `reporting:media-delivery.type` to "wsd."
- Auto-enable the **Subcustomer Enablement** behavior.

Create a new property

If you're setting up a new ACE base configuration, you first need to set up a new "Property" via Akamai Control Center.

1. Select the appropriate Control Center Account. Use the top-right pull-down in the header to select the account.
2. Open the application. Go to ☰ > **CDN** > **Properties**. Click **New Property**.
3. Select the **Wholesale Delivery** radio button from the Product list.

   Note: "Wholesale Delivery" is the legacy name of this product. A future release of this toolset will update this value to "Akamai Cloud Embed."

4. If more than one **Contract** is shown, choose the one that has Cloud Embed on it. You are not sure, contact your Account Representative.

5. In the Property Name field, enter a desired name for the ACE base configuration. This serves two purposes:

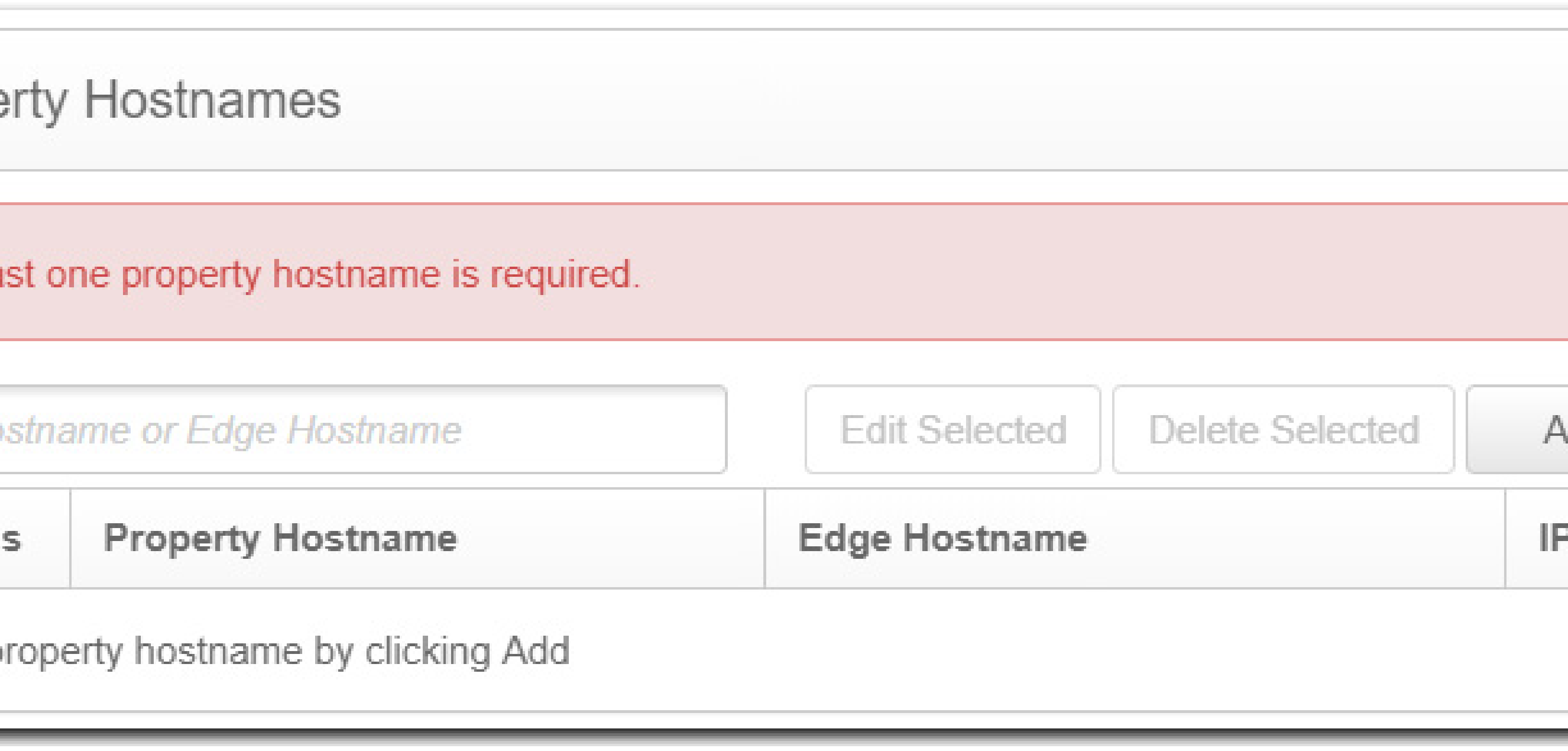
   - The filename for the configuration, and
   - How it will be displayed for access throughout the Property Manager UIs in Control Center.

6. Click the **Create Property** button.

The **Property Manager Editor** window launches showing the default settings for the new property. Use the settings here to define your ACE configuration.

Define property hostnames

You use this content panel to associate your cloud partner endpoint to an Akamai Edge hostname. A "property hostname" association plays a key role in getting subcustomer sites, applications or content out to our Edge servers, for access to requests from subcustomer clients.


erty Hostnames

ist one property hostname is required.

roperty hostname by clicking Add

Once set up, a request from an end user to a subcustomer resource is resolved by the Edge hostname. The request is routed to our Edge servers that then read your ACE base configuration to determine how to serve requests for subcustomer content.

The property hostname

This is an identifier that is used to determine which base configuration file to use when processing end-user HTTP/HTTPS requests for a subcustomer's resource. Often, the Property Hostname is the fully qualified domain name of the endpoint subcustomers use to access your cloud partner CDN. For instance, `www.mycdn.com` would be your endpoint domain name *and* your Property Hostname. A property hostname is sometimes referred to as a "digital property," too.

Wild Card Usage

You can use an asterisk (`*`) to indicate a wild card for a subdomain in a property hostname. On the top level, this causes all hostnames for that domain to use the associated base configuration. For example, if you set up a property hostname as `*.example.com`, both `www.news.example.com` and `www.uk.news.example.com` use the configuration.

Note: You must own the truncated portion of the domain when using a wildcard-enabled property hostname. Using the example above, you must own "example.com."

The edge hostname

This describes your specific cloud partner subdomain (endpoint) on an Edge network domain. It maps incoming requests to our Edge servers. You generate a "property hostname to Edge hostname association" to determine an Edge hostname. For example, assume your endpoint that subcustomers access is `www.mycdn.com`. This would serve as the property hostname in this association that results in the Edge hostname, `www.mycdn.com.akamaized.net`. An end-user request for a subcustomer's assets is ultimately directed to this Edge hostname. In turn, the Edge hostname resolves the request to individual servers on the Edge network, where the associated base configuration is read to determine how to process the request.

Note: Edge Hostnames aren't used to directly serve content (unless leveraging a Shared Certificate for HTTPS delivery—see the table in the section that follows). *Edge Hostnames are used only to resolve your content to the Edge network*.

Edge host naming conventions

The standard format for Edge hostnames varies for non-secure (HTTP) and for secure (HTTPS) web sites and applications. The table that follows shows some Edge hostname examples.

Content type	Property hostname	Edge hostname	Description
Non-secure HTTP (via Instant Config Hostname)	`www.mycdn.com`	`www.mycdn.com.mdc.akamaized.net`	*Instant Config Hostname* is a tool in Property Manager that allows you to quickly create an Edge hostname using a fixed domain —`.mdc.akamaized.net`
Secure HTTPS (via custom certificate)	`secure.mycdn.com`	`secure.mycdn.com.edgekey.net`	When you've applied a custom certificate to a property configuration, the associated Edge hostname

Content type	Property hostname	Edge hostname	Description
			uses the `.edgekey.net` domain.
Secure HTTPS (via Shared Certificate	`mycdn-com.akamaized.net`	(none)	When using our shared certificate, your Property hostname and the Edge hostname are the same value.

How to set up a property hostname

You'll use Property Manager for this, and the process is predominantly the same regardless of product. To maintain consistent instructions, this process is covered in the Property Manager *documentation*.

Property hostname use cases

Here are a few use case examples that address each delivery type, both secure (HTTPS) and non-secure (HTTP).

Each of the use cases begins with the notion that as a Cloud Partner, you operate one or more domains that are directly related to their various PaaS/IaaS offerings, as either of the following:

- The DNS name published to "End-Clients" (browsers, applications, media players, etc.) in a fully-qualified URL.
- The origin hostname to which requests to vanity hostnames may be directed—typically, via a DNS CNAME record, and, especially, when there is no CDN in the request flow.

The various use cases are described with this domain in mind, and they are separated into two primary groups: “HTTP-Only” and “HTTPS.” HTTPS refers to the capability of securing communication between the Akamai platform and the browser, using standard TLS certificates, but that HTTP communication is also supported for these subcustomer without any special requirements. For the purposes of illustration, these examples use the following:

- The hypothetical Cloud Partner domain `cloudplatform.net`.
- When referring to HTTPS delivery, examples use a Cloud Partner subdomain `secure.cloudplatform.net`. (The “secure” record name is not technically required.)

Use case 1: HTTP-only, using a partner-owned DNS name

This use case involves usage of cloud partner-managed DNS hostnames to serve content to end-clients via the Akamai CDN.

For example, `{sub-customer-prefix}.cloudplatform.net`

Value	Example
Partner Domain	`cloudplatform.net`

Value	Example
End-Client-facing Domain CNAME	`{sub-customer-prefix}.cloudplatform.net {TTL} IN CNAME cloudplatform-net.mdc.akamaized.net`

Example End-Client-facing Domains	`static.sub-customer.com.cloudplatform.net`
	`2358c713-a9dad7fd-f838fd35-8c85b331.cloudplatform.net`

Complete End-Client CNAME Chain	`static.sub-customer.com.cloudplatform.net. {TTL} IN CNAME cloudplatform-net.mdc.akamaized.net.`
	
	`cloudplatform-net.mdc.akamaized.net. 21600 IN CNAME a1234.dscw14.akamai.net.`
	
	`a1234.dscw14.akamai.net. 20 IN A 72.246.199.74`

This use case is based on our Multi-Domain Configura-tion (MDC) solution via an "Instant Config Hostname." When using the example domain `cloudplatform.net` in this example, a hypothetical CNAME chain observed by End-Clients might be as shown above. As the Cloud Partner, you can choose the “{TTL}” for the CNAME to the `mdc.akamaized.net` hostname. However, anything less than five (5) minutes is considered too low.

Note: Default TTLs are provided in the example for the remainder of the CNAME chain.

In addition, the Cloud Partner must provide a separate origin DNS hostname or IP address for each Sub-Customer domain. There are two possible mechanisms for deriving the origin DNS name:

1. RECOMMENDED: As the Cloud Partner, you can use the ACE API to create a content delivery policy with an “origin” behavior for each subcustomer. This delivery policy may use the IP address of the origin that would normally have been handed out when the delivery policy is served without Akamai; that is, the A record that would normally be returned by the your DNS for the subcustomer hostname.

   However, we recommend that you use a separate DNS hostname in the “origin” behavior to allow migration of subcustomer origin IPs between your cloud partner data centers without having to update each subcustomer delivery policy.

2. You can configure a static value as a prefix in front of the End-Client-facing hostname. For example, the base configuration can apply a default delivery policy that concatenates a static prefix, for example “origin-” and the End-Client domain. In such cases, the “origin” behavior is not required.

Following the example above, as the Cloud Partner, you need to provision a DNS entry for `origin-static.sub-customer.com.cloudplatform.net`. This might look like the following:

```
origin-static.sub-customer.com.cloudplatform.net. {TTL} IN A 77.76.75.74
```

As the Cloud Partner, you can also optionally do the following:

- **Set the TTL**. Set it to a minimum of one (1) minute, with five (5) minutes (or more) being the recommended value.
- **Use additional intermediate CNAMEs or use DNS-based load balancing**. For example, you could use Akamai's Global Traffic Management solution.

Use case 2: HTTP-only, using a custom/vanity DNS name

This use case refers to using "vanity" hostnames such as "static.{subcustomer}.com" where the *subcustomer* owns the DNS names used to serve content to End-Clients.

Value	Example
Partner Domain	`cloudplatform.net`
End-Client-facing Domain CNAME	`{sub-customer-vanity-domain}. {TTL} IN CNAME cloudplatform-net.mdc.akamaized.net`
Example End-Client-facing Domains	`static.sub-customer.com` `www.other-sub-customer.com`

Complete End-Client CNAME Chain - Example 1: subcustomer hostname direct to Akamai.	`static.sub-customer.com. {TTL} IN CNAME cloudplatform-net.mdc.akamaized.net.`
	
	`cloudplatform-net.mdc.akamaized.net. 21600 IN CNAME a1234.dscw14.akamai.net.`
	
	`a1234.dscw14.akamai.net. 20 IN A 72.246.199.74`

Complete End-Client CNAME Chain — Example 2: Sub-Customer hostname direct to Cloud Partner intermediate hostname to Akamai.	`static.sub-customer.com. {TTL-1} IN CNAME static.sub-customer.com.cloudplatform.net.`
	
	`static.sub-customer.com.cloudplatform.net. {TTL-2} IN CNAME`

Value	Example
	`cloudplatform-net.mdc.akamaized.net.` `cloudplatform-net.mdc.akamaized.net. 21600 IN CNAME a1234.dscw14.akamai.net.` `a1234.dscw14.akamai.net. 20 IN A 72.246.199.74`

This use case is based on our Multi-Domain Configuration (MDC) solution via an "Instant Config Hostname." The subcustomer who owns the DNS zone will do either of the following:

- CNAME the End-Client-facing hostname directly to the Cloud Partner's MDC hostname (as in Example 1).
- CNAME to your Cloud Partner-operated intermediate hostname, that in turn is CNAMEd to the Akamai MDC hostname (as in Example 2).

In both cases, the time to live (TTL) for the vanity hostname CNAME record is determined by the subcustomer In the second example, `{TTL-1}` is also determined by the subcustomer, but as the cloud partner, you determine `{TTL-2}`.

The intermediate hostname is optional. However, to ensure that you (as the Cloud Partner) maintain control of when and how a CDN is implemented, this is strongly encouraged. A common CNAME can be used for all subcustomer. An example might be something like the following, where a more generic `cdn` record replaces the more specific `static.{subcustomer}.com`" record in your Cloud Partner DNS configuration:

```
static.sub-customer.com.cloudplatform.net. {TTL-1} IN CNAME

cdn.cloudplatform.net. cdn.cloudplatform.net. {TTL-2} IN CNAME cloudplatform-
net.mdc.akamaized-net.

cdn.cloudplatform.net.mdc.akamaized.net. 21600 IN CNAME
a1234.dscw14.akamai.net.

a1234.dscw14.akamai.net. 20 IN A 72.246.199.74
```

You can determine the origin domain when the origin you are hosting it, or by the subcustomer when hosted elsewhere. It should be configured for each End-Client-facing hostname; multiple origins are supported based on specific match criteria.

Use case 3: HTTPS using a cloud partner-owned DNS name: wildcard certificate

Some subcustomers may want to serve content securely using one of your Cloud Partner-managed DNS namespaces.

Value	Example
Partner Domain	`secure.cloudplatform.net`

Value	Example
End-Client-facing Domain CNAME	`[sub-customer-prefix].secure.cloudplatform.net {TTL} IN CNAME secure.cloudplatform-net.edgekey.net`

Example End-Client-facing Domains	`secure-sub-customer-com.secure.cloudplatform.net`
	`2358c713-a9dad7fd-f838fd35-8c85b331.secure.cloudplatform.net`

Complete End-Client CNAME Chain	`secure-sub-customer-com.secure.cloudplatform.net. {TTL} IN CNAME secure.cloudplatform.net.edgekey.net.`
	
	`secure.cloudplatform-net.edgekey.net. 21600 IN CNAME a1234.dsce16.akamai.net.`
	
	`a1234.dsce16.akamai.net. 20 IN A 184.24.175.127`

In this example, let's assume that you operate one or more Cloud Partner infrastructure domains where each subcustomer is given a unique prefix, for example:

```
{sub-customer-prefix}.secure.cloudplatform.net
```

To configure secure delivery, we deploy a separate certificate, in which the Common Name (CN) is a wildcard DNS name `*.secure.cloudplatform.net`, and the Akamai Secure Edge Hostname CNAME, `secure.cloudplatform.net.edgekey.net` is provisioned to ensure that the correct certificate is returned to the End-Client.

The Akamai server requires a separate DNS hostname to use as origin. The `origin-` prefix method works, as well as the practice of creating an `origin` behavior for each subcustomer as shown in .

For subcustomers that provision their own certificates (non-recognized CA or even self-signed certificates), the integration requires that you use Akamai Control Center (or the Property Manager API) to configure the origin certificate and trust chain details individually on behalf of the subcustomer.

ⓘ **Note:** This method doesn't use the ACE API. It requires the *Property Manager API* to properly configure the origin security settings.

Use case 4: HTTPS using a custom/vanity DNS name: shared cert

Here, we use a “vanity” hostname for HTTPS traffic, and the subcustomer owns the DNS names used to serve content to End-Clients.

Additionally, the subcustomer doesn't require a vanity TLS certificate. (Other subcustomer hostnames may exist in the list of supported Subject Alternative Name (SAN) entries in a single TLS certificate.)

Value	Example
Partner Domain	`secure.cloudplatform.net`

End-Client-facing Domain CNAME	`{sub-customer-vanity-domain}.`
	`{TTL} IN CNAME`
	`{cert-`
	`identifier}.secure.cloudplatform-`
	`net.edgekey.net`

Example End-Client-facing Domains	`secure.sub-customer.com`
	`shop.other-sub-customer.com`
	`api.another-customer.com`

Complete End-Client CNAME Chain - Example 1: subcustomer hostname direct to Akamai	`secure.sub-customer.com. {TTL} IN`
	`CNAME`
	`san8.secure.cloudplatform.net.edgek`
	`ey.net.`
	
	`san8.secure.cloudplatform.net.edgek`
	`ey.net. 21600 IN CNAME`
	`e1235.dsce16.akamaiedge.net.`
	
	`e1235.dsce16.akamaiedge.net. 20 IN`
	`A 184.24.170.18`

Complete End-Client CNAME Chain — Example 2: subcustomer hostname direct to Cloud Partner	`secure.sub-customer.com. {TTL-1}`
	`IN CNAME`
	`secure.sub-`
	`customer.com.secure.cloudplatform.n`
	`et.`
	
	`secure.sub-`
	`customer.com.secure.cloudplatform.n`
	`et. {TTL-2} IN CNAME`
	`san8.secure.cloudplatform.net.edgek`
	`ey.net.`
	
	`san8.secure.cloudplatform.net.edgek`
	`ey.net.21600 IN CNAME`
	`e1235.dsce16.akamaiedge.net.`
	
	`e1235.dsce16.akamaiedge.net. 20 IN`
	`A 184.24.170.18`

With this example, you (as the Cloud Partner) can support hundreds, or even thousands of these vanity DNS hostnames using shared SAN certificates. Multiple certificates are required to support such large numbers of hostnames, because a single TLS certificate has an overall size limit. Each SAN has a hard limit of 100 alternative name entries, but we recommend that you keep the max closer to 40 entries for best performance.

The Time to Live (TTL) for the vanity hostname CNAME record is determined by each individual subcustomer. (This is the second example `{TTL-1}`, above.) Being the cloud partner, you determine `{TTL-2}`.

In this case, you can adopt a pattern to relate a collection of DNS names to a single TLS SAN certificate. For example:

```
san{n}.secure.cloudplatform.net
```

- `{n}`: This could be a simple index/counter for each SAN certificate you're using. You can create and modify certificates (add or remove hostnames) via the CPS API, via the CPS UI in Control Center, or you can work with Akamai Professional Services (at a cost) to implement them for you.

For example, a value of `san1.secure.cloudplatform.net` might be used for the first SAN certificate. This value is used by the back-end provisioning systems to define the Secure Edge Hostname for all SAN entries in this certificate, and is suffixed with `.edgekey.net`. As an end result, you have `san1.secure.cloudplatform.net.edgekey.net` as the DNS name in this example. It refers to this SAN and would support HTTPS delivery for any of the Subject Alternative Names added to the certificate.

Next, we use the sample subcustomer domain examples from the table above:

- `secure.sub-customer.com`
- `shop.other-sub-customer.com`.

Each subcustomer must CNAME to either:

- Your cloud partner-managed DNS name (that is in turn CNAMEd to the secure Edge Hostname)
- Directly to the secure Edge Hostname

The former lets you maintain control of the delivery and the associated certificate, and we recommend this approach. This might look like the following pair of example CNAME chains to be created by the respective Sub-Customers and in turn by you as the cloud partner:

```
secure.sub-customer.com. {TTL-1} IN CNAME
secure.sub-customer.com.secure.cloudplatform.net.

secure.sub-customer.com.secure.cloudplatform.net. {TTL-2} IN CNAME
san8.secure.cloudplatform.net.edgekey.net.

san8.secure.cloudplatform.net.edgekey.net. 21600 IN CNAME
e1235.dsce16.akamaiedge.net.

e1235.dsce16.akamaiedge.net. 20 IN A 184.24.170.18
```

```
shop.other-sub-customer.com. {TTL-1} IN CNAME
shop.other-sub-customer.com.secure.cloudplatform.net.

shop.other-sub-customer.com.secure.cloudplatform.net. {TTL-2} IN CNAME
san8.secure.cloudplatform.net.edgekey.net.
```

```
san8.secure.cloudplatform.net.edgekey.net. 21600 IN CNAME
e1235.dsce16.akamaiedge.net.

e1235.dsce16.akamaiedge.net. 20 IN A 184.24.170.18
```

The CPS tool in Control Center and the deployed TLS certificate itself, show the list of alternate names in a given certificate. However,as the Cloud Partner, it is your responsibility to maintain the relationship between each vanity subcustomer domain and the Akamai Secure Edge Hostname to which the domain has been associated. It is also your responsibility to ensure that no certificate is overloaded beyond the stated limits.

What about the origin server?

Similar to the other use cases, the Akamai server requires a separate DNS hostname to use as the origin. The `origin-` prefix method works, so does creating an `origin` behavior for each subcustomer (just like what we show in *Use case 1: HTTP-only, using a partner-owned DNS name*, above.)

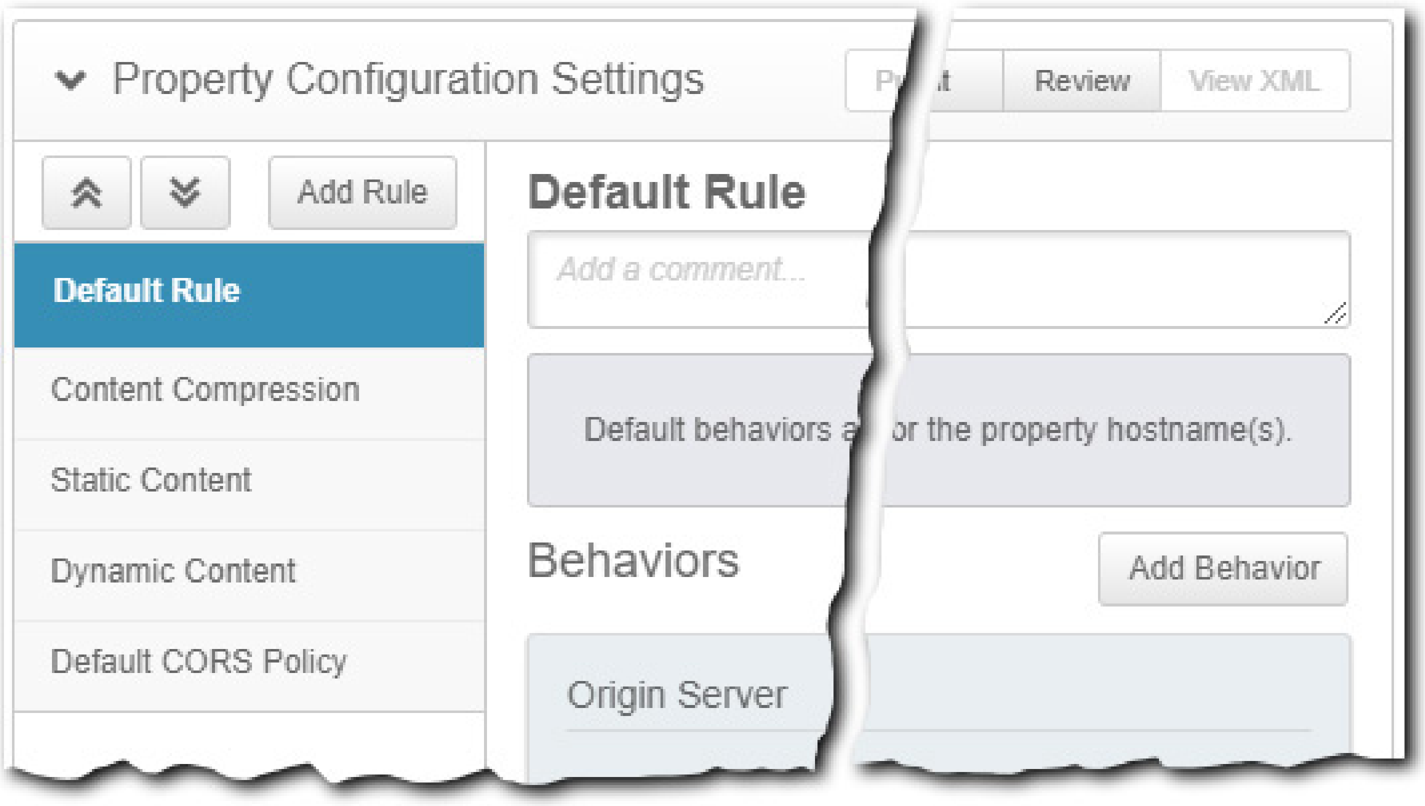
When the intermediate hostname is not the same, the cloud partner DNS hostname that refers to the subcustomer method you use to retrieve content can be configured as the origin domain.

If you have subcustomers who provision their own origin certificates you must use the Property Manager in Control Center (or the Property Manager API) to configure the origin certificate and trust chain details individually on behalf of the subcustomer. When setting up the certificate for this use case, use the **Choose Your Own** option and a **Trust** setting of **Custom Certificate Authority Set**. Once set, you can add new **Certificate Authorities** or **Specific Certificates** for self-signed certificates.

Configure the "Default Rule"

When defining behaviors for a new ACE base configuration, the **Default Rule** is automatically comprised of various necessary and recommended behaviors.

It is a fixed rule, and it must be included in all ACE base configurations. (However, you can edit its individual behaviors, as necessary.) In addition, the Default Rule has *no* Match Criteria, because its behaviors apply to *all requests*.



The Origin Server behavior

This required behavior offers options you use to control how the Edge network contacts an origin server to retrieve website or application content.

Origin Server precedence and ACE

Edge servers read first from the applicable subcustomer's policy for origin server information, and second from the Origin Server behavior settings in your base configuration. *The origin server (and associated information) set in a base configuration is only utilized if an origin server is not defined in the subcustomer's policy*.

Origin Server use cases with ACE

Setting up an origin server for ACE is different than setting one up for a traditional Akamaidelivery product. A traditional product serves from a single origin to deliver your content. With ACE, you'll have multiple (potentially 1,000s) of subcustomers using this configuration, so your origin server needs are different, and this has a bearing on how you set Origin Server options. Consider the following use cases:

- **Typical use case - subcustomers have their own origin servers**: If this is the case, you would set Origin Settings in the base configuration to serve as a "backup," and you would provide specific origin server information in each subcustomer's policy.
- **subcustomers are using your origin server**: If you maintain your own origin server for all of your subcustomers, or are using NetStorage as the origin server, you would provide origin server information in your base configuration, and you'd set the **Origin** slider in the **Subcustomer Enablement** behavior to "**Off**." This way, subcustomer origin server data can't be set in a policy, and the origin you've set in the base configuration will be used, instead.

- **A mix of both**: Here, you'd perform what's discussed in "Typical use case - subcustomers have their own origin servers." However, for the subcustomers you want to use your origin server, you'd need to ensure that origin server information is left out of the subcustomer policy.

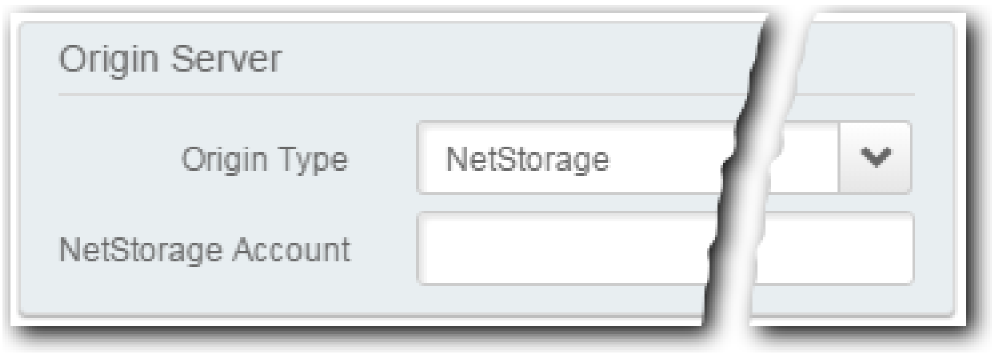
Note: Regardless of your desired use case, Origin Server behavior settings are required in the Default Rule in a base configuration.

Available settings in this behavior vary based on the selected **Origin Type**.

I selected "NetStorage" as my Origin Type

This is the case if you're using NetStorage as your origin to house Subcustomer website or application content.

The following additional options and recommended settings are available when you select NetStorage as your Origin Type:



Origin Type: NetStorage

NetStorage Account:

NetStorage Account

Click this field to select the appropriate NetStorage download domain. This is the name of the domain set up for use with this application on your NetStorage account. Once you assign the download domain, Property Manager automatically assigns the root directory (for example, /R3131), so that content is retrieved from the appropriate location.

You need to set a supported Caching Option

With **NetStorage** set as your origin, you cannot have your Caching behavior set to either "**No Store**" or "**Bypass Cache**." Set the **Caching Option** to the appropriate setting for your instance.

Additional recommendations

Once you set NetStorage as your origin, we have a couple of suggestions:

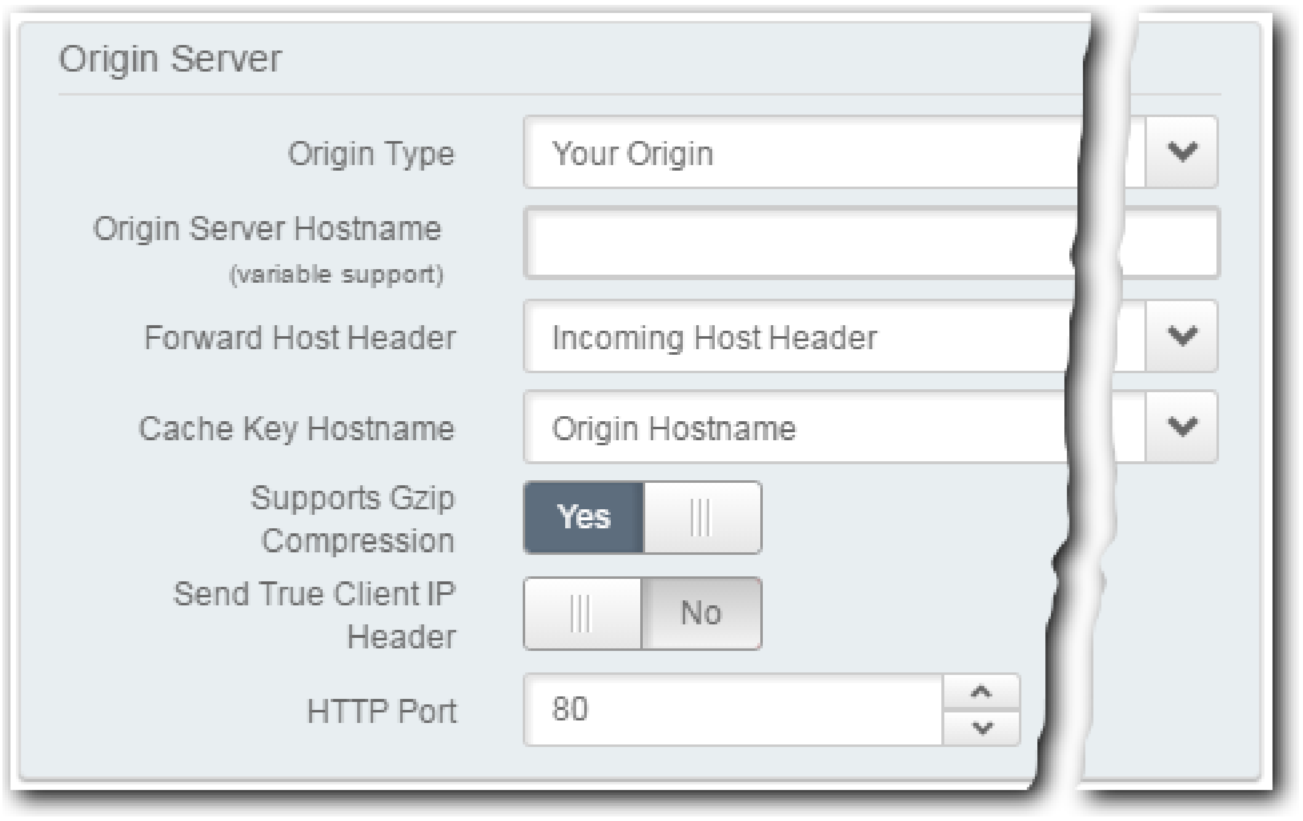
- **Add Cache HTTP Error Responses as an optional behavior**. To do so, click the **Add Behavior** button, select **Cache HTTP Error Responses**, and click **Insert Behavior**. In the newly added behavior, set Max-age to **30**.

- **The Cache Key Query Parameters behavior should be set to "Exclude all parameters."** NetStorage doesn't honor query strings, so you should set this to avoid duplicate cache keys for the same object. Click the **Add Behavior** button, select **Cache Key Query Parameters**, and click **Insert Behavior**. Set the drop-down to **Exclude all parameters**.

I selected "Your Origin" as my Origin Type

This is the case if you're using your own custom origin to house target content.

The following additional options and recommended settings are available when you select **Your Origin** as your Origin Type:


Origin Server

Origin Type: Your Origin

Origin Server Hostname (variable support):

Forward Host Header: Incoming Host Header

Cache Key Hostname: Origin Hostname

Supports Gzip Compression: Yes

Send True Client IP Header: No

HTTP Port: 80

Origin Server Hostname

Input the value that points to the same IP address as the origin domain name. We retrieve content from this address. This has various requirements and options for use:

Tip: Make note of the value input here for later use.

- **There are naming conventions**. An Origin Server Hostname must follow a specific naming convention: `origin-<original origin hostname>`:
  - `origin-`: This is a fixed value: the word `origin`, followed by a hyphen (`-`).
  - `<original origin hostname>`: This is the name that is expected to appear in the host header.
- **The DNS must be edited**. Generation of this configuration does not implement or activate your Origin-server Hostname. Once set, the DNS record for your existing Origin Server needs to be modified using this hostname (either by you or your DNS administrator). You ultimately need your

origin's DNS record to point to the same server IP address as this ACE configuration. For example, if the original DNS record contains:

```
www.mymedia.com. IN A 1.2.3.4
```

The edited record should contain:

```
origin-www.mymedia.com. IN A 1.2.3.4
```

- **Don't use IP addresses as Origin Server Hostnames**. While IPv4 or IPv6 format addresses are supported, they are not recommended, as they can change or be reassigned, which may render your domain unreachable (resulting in a denial of service).
- **This option supports variable expression syntax**. Typing “{{“ in the option field will trigger variable to auto complete. Additional details on this support are available by mousing over this option in the UI and clicking the “Learn more about variable support” link.

Forward Host Header

Select the host header you want the product to pass to your origin server. This is referred to as the "Forward Host Header" because it is the hostname the product “forwards” to the origin server in the HTTP HOST request header. The web server on your origin server uses this value to determine what content to send. Typically, the expected host is the same name as the hostname received in the request, or it can be customized. The following are available:

- **Incoming Host Header (Default)**: When selected, the same name as the hostname received in the request is used. This is a generic option that varies with the hostname received in the request. For example, a request for `www.mymedia.com` sends `www.mymedia.com` the HOST header; while a request for `test-www.mymedia.com` sends `test-www.mymedia.com` as the value.
- **Origin Hostname**: When selected, what you've set as the “Origin Server Hostname” is sent in the request to your Origin Server. Select this option if your Origin Server has been configured to listen for the Origin Server Hostname. For example, if a request for either `www.mymedia.com` or `test-www.mymedia.com` is sent, `origin-www.mymedia.com` is sent in the HOST header in the request to the Origin Server.
- **Custom Value**: Select this option if the hostname is a different name than the one the Origin Server is expecting. For example, an end-user request for `www.mymedia.com.akamaized.net` can set `www.mymedia.com` as the value sent in the HOST header to your Origin Server.

Cache Key Hostname

The cache key is the information the product uses to identify the content in caches. Assuming your application includes at least some cacheable content—the Edge network uses keys based on the entire Origin Server URI path and query string, if there is one. The following selections are available:

- **Origin Hostname**: All objects requested using this Origin Server Hostname and the same path and query string are treated as the same object, including the content served from any other configuration with the same Origin Server Hostname. For example, once cached, these objects would be treated as *the same* object:

```
http://www.mymedia.com/logo.gif
http://www.mymedia.co.uk/logo.gif
```

- **Incoming Host Header** (Virtual Server Option). Objects requested with the same path and query string are given a unique cache key per hostname. Select this option if your origin server is a virtual server. For example, once cached, these objects are treated as *different* objects:

  ```
  http://www.mymedia.com/logo.gif
  http://www.mymedia.co.uk/logo.gif
  ```

Supports Gzip Compression

Compression is important in optimizing performance. You can disable this option only if your Origin Server does not support delivery of content using Gzip compression; or, if for some reason you want to have content served uncompressed. When this feature is enabled, the product sends an `Accept-Encoding: gzip` header in requests to the Origin Server to support Gzip compression.

Send True Client IP Header

When this slider is set to **Yes**, the IP address of the requesting client is passed to the origin. Normally the client IP is passed in the `X-Forwarded-For` header, which is routinely modified by proxies along the way. Once enabled, additional options are offered:

- **True Client IP Header Name**: Input the name of the header that contains the True Client end-user IP address. This is typically the `True-Client-IP` header, which is input here by default.

- **Allow Clients to Set**: Set this slider to "**Yes**" to have the Edge server that receives the request allow the True Client IP Header and pass that value through to the origin, or set it the "**No**" to remove it and set the value itself.

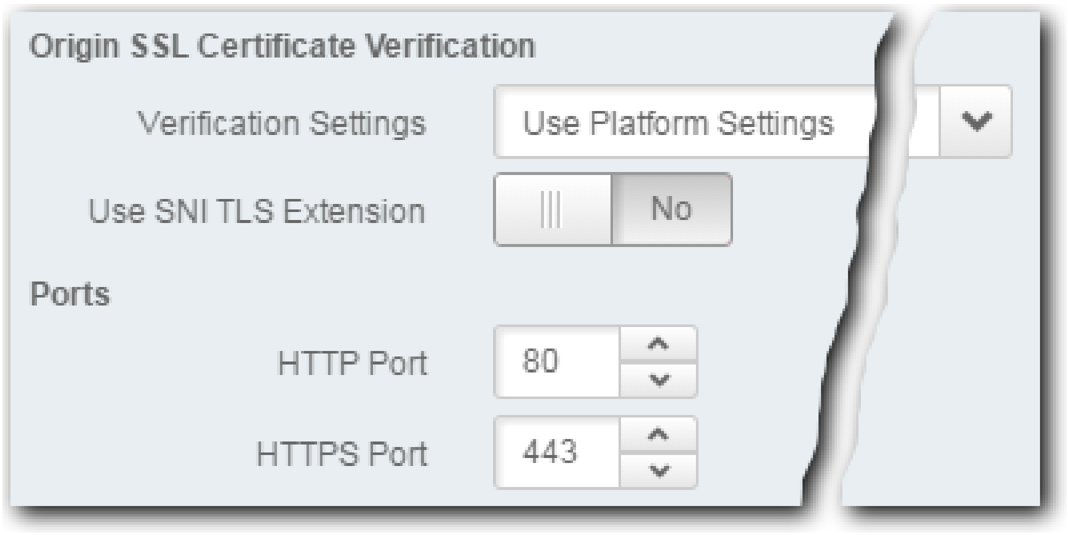
HTTP Port

This exists as a standalone option when your ACE configuration is setup to exclusively deliver non-securely via HTTP. This is the port on your origin server you want our Edge server to connect to for *non-secure HTTP requests*. The standard port is 80. To learn more about ports, mouse-over this option and click the “Learn more” link.

Origin SSL Certificate Verification

These options are revealed if the ACE configuration is set up for secure delivery (HTTPS). They allow you to control how your Origin Server is authenticated. They are intended to prevent 'man-in-the-middle' (MITM) attacks, in which a malicious entity directs end-user traffic to the attacker's server, instead of the expected Origin Server.

When an Edge server routes a request to your Origin Server, it establishes a secure connection through an SSL handshake; your Origin Server provides the Edge server with a certificate which is used to validate it as your Origin Server. If everything is validated, the request goes forward. If the certificate is not valid, the action you set in the ACE configuration for invalid certificates occurs.


Origin SSL Certificate Verification

Verification Settings: Use Platform Settings

Use SNI TLS Extension: No

Ports

HTTP Port: 80

HTTPS Port: 443

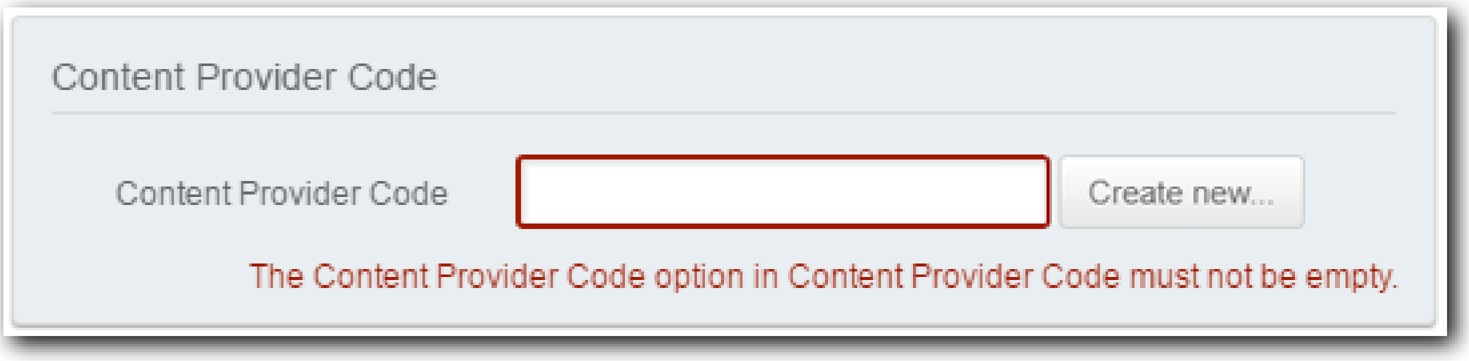
Note: The settings you choose in Origin SSL Certificate Verification override the default settings for your ACE configuration.

- **Verification Settings**: The Secure Network platform has default settings for Origin SSL Certificate Verification that can be overridden by a ACE configuration. The platform, by default, trusts certificates signed by the certificate authorities in the Akamai Certificate Store that also have a CN/SAN that matches the Forward Host Header.
  - **Use Platform Settings**: This allows Edge servers to choose these settings on your behalf, trusting certificates signed by any authority listed in the Akamai Certificate Store. These settings are subject to change at any time.
  - **Choose Your Own (Recommended)**: Select this to maximize security by directly controlling which certificates Akamai Edge servers should trust. (Once selected, additional options are revealed to configure this.)
  - **Use SNI TLS Extension**: Set this slider top "Yes" to have the Edge server send the Server Name Indication (SNI) header in the SSL request to your origin. The SNI header is comprised of the same information contained in the header you have selected as the **Forward Host Header** value.
- **Ports**:
  - **HTTP Port/HTTPS Port**: These are the ports on your origin server you want our Edge server to connect to for non-secure HTTP and secure HTTPS requests, respectively. The standard ports are 80 for HTTP and 443 for HTTPS. To learn more about ports, mouse-over either of these options and click the “Learn more” link.

Tip: To learn more about any of these options, mouse over their names in the UI.

Content Provider Code and ACE

Content Provider (CP) codes are numeric IDs assigned to client requests in your configuration. These codes are used to identify your content for billing, logging, reporting, and cache purging. *This is a required behavior*.



The Content Provider Code option in Content Provider Code must not be empty.

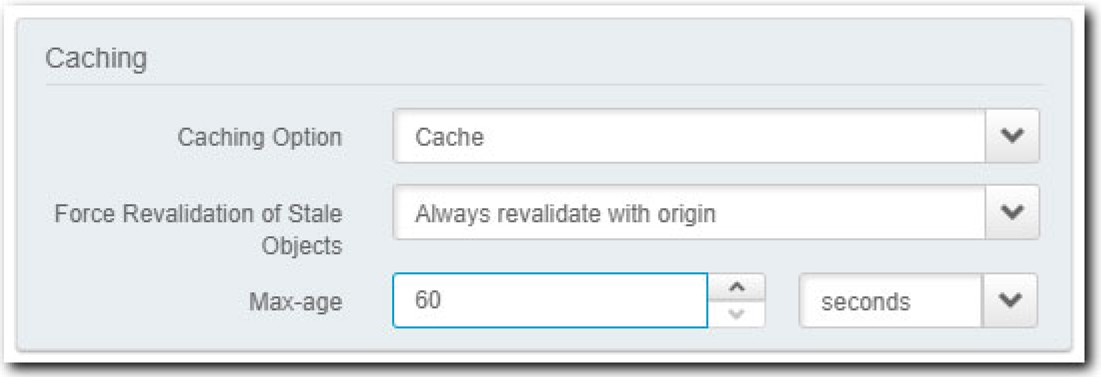
Your Akamai account representative assigns you a CP code for use with ACE, typically during initial setup. To set the CP code for this behavior, perform either of the following:

- **Click the Content Provider Code field**. A drop-down shows a list of CP codes that have already been configured for use with ACE, and are available to the user account that's currently logged in to Control Center.
- **Click Create new...** If your Control Center account has access to do so, you'll see this button in the behavior. Click it to create a new CP code. The default name for the new code is what you've set for your ACE Property Name, but you can change it if you want. A new CP code may take upwards of two hours to propagate to our network. You'll have to wait for it to finish before you can save your ACE property, and go on to test and activate it.

Note: If you get an error when selecting an existing CP code, are told you have no more CP codes available when you click the **Create new...** button, or you don't have access to that button, contact your local Akamai administrator or your Akamai account representative for help creating a new code.

The Caching behavior

This behavior lets you specify basic caching behaviors for our Edge servers. You can define how responses should be cached, for how long, and whether the response can be served stale if the origin server cannot be reached to re-validate the response.


Caching

Caching Option: Cache

Force Revalidation of Stale Objects: Always revalidate with origin

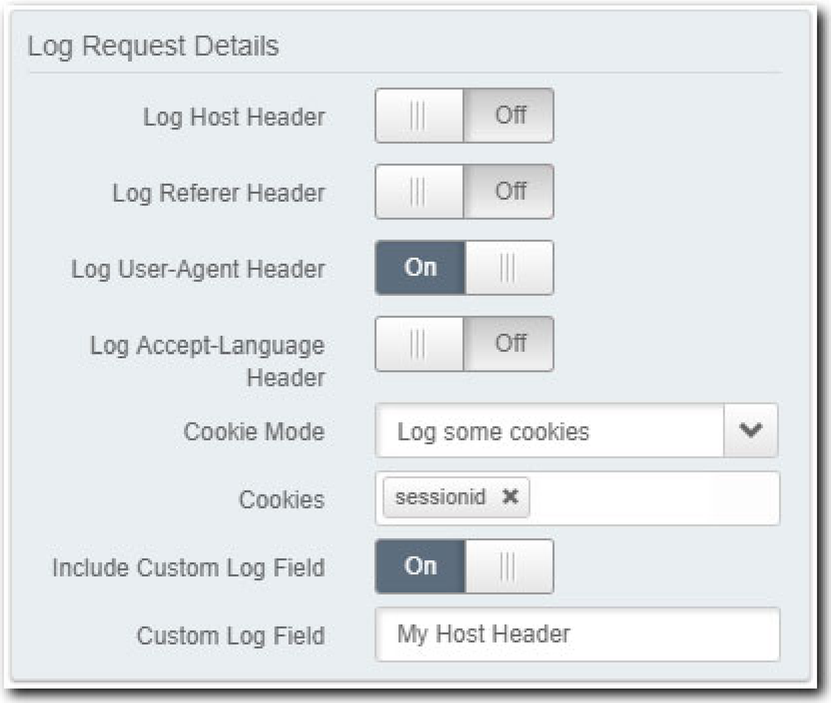
Max-age: 60 seconds

You can select from the following options:

- **No Store** (default): Content is served directly from the specified origin, and any versions in the cache are cleared.
- **Cache**: Content is cached, based on additional settings that are revealed:
  - **Force Revalidation of Stale Objects**: Once content remains in cache for a predetermined amount of time (either what's been defined specifically for the origin, or what's been set in the **Max-age** field), it is considered "stale." Select how you want requests for cached content to be handled once that content is stale:
    - **Serve stale if unable to validate**: The request tries to re-validate with the origin to get the content, but if it can't the "stale" content is served.
    - **Always revalidate with origin**: The request only serves content that has been re-validated from the origin. (The request is retried until revalidation occurs, or it times out.)
  - **Max-age**: Set the amount of time that content will be cached if the origin does not specify one.
- **Bypass cache**: Content is served from the specified origin, but cached versions of the content are *not removed*. This may be useful for certain subcustomers. For example, if using central authorization, bypass the cache for the client request. If end users are authenticated, they are redirected to the cached content. (Otherwise, a login module could be returned.)
- **Honor Origin...**: You can choose to honor the caching settings defined in origin response headers: **Honor Origin Cache Control**, **Honor Origin Expires**), or both. When you use origin response headers, you also set a Default Max-age to use if the relevant origin header is not found or is invalid. You van also optionally enable support for caching settings that may be set in "**Private**" and "**Must-Revalidate**" response headers, by setting the applicable switches to **Yes**.

The Log Request Details behavior

This behavior lets you select the HTTP Request Headers and Cookies you want in included in your Akamai Log Delivery Service (LDS) reports.


Log Request Details

Log Host Header	Off
Log Referer Header	Off
Log User-Agent Header	On
Log Accept-Language Header	Off
Cookie Mode	Log some cookies
Cookies	sessionid
Include Custom Log Field	On
Custom Log Field	My Host Header

- **Log Host Header**: This header specifies the domain name of the server (for virtual hosting), and (optionally) the TCP port number on which the server is listening.
- **Log Referer Header**: This header contains the address of the previous web page from which a link to the currently requested page was followed. The Referer header allows servers to identify from where people are visiting.
- **Log User-Agent Header**: This header contains a characteristic string that allows the network protocol peers to identify the application type, operating system, software vendor or software version of the requesting client.
- **Log Accept-Language Header**: This header advertises which languages the client is able to understand, and which locale variant is preferred.
- **Cookie Mode**: Select the cookies to be logged—**Don't log any cookies**, **Log all cookies** or **Log some cookies**. Selecting the latter reveals a separate field that you click to select the desired cookie format to capture, **"sessionid"**, **"user_local"**. You can also click the field and manually input the name of an applicable cookie format you want to use. (After which, you need to click the <input name> (new item) option in the drop down to add it.)
- **Include Custom Log Field**: Set this switch to "**On**" to reveal the Custom Log Field. Manually input an applicable log format that you want included in the logs.

Note: The **Custom Log Field** is truncated to no more than 40 bytes of data. Also, the information as it appears in the logs is escaped per rfc1738.

The Subcustomer Enablement behavior

This behavior in your Akamai Cloud Embed base configuration allows you to control which individual features can be included and set in a Subcustomer's "policy."

A policy is a set of rules and behaviors that are applied when an end user requests a specific Subcustomer's content. You use the ACE API to create individual Subcustomer policies. (They are defined in a PUT body component included with a specific call from the API.)

This behavior lets you enable or disable various options to limit the use of specific behaviors in Subcustomer policies.

Important: What you set here applies to *all* subcustomers assigned to this base configuration. For example, if you set an option to "Off," any subcustomer assigned to this base configuration *cannot* configure the associated behavior in their policies. If you want the subcustomer to access a behavior, ensure it's set to On."

Dynamic Policy

This switch enables Subcustomer Enablement settings.

- **Set to "On"**: This is typical. It applies if you want to set up individual policies for each Subcustomer. You can enable/disable individual behaviors to meet your needs. (If disabled, that specific behavior can't be modified in an individual Subcustomer policy.)

- **Set to "Off"**: This is uncommon, and only applies to some lesser-used use cases (for example, Subcustomer level billing, when different delivery configurations are not required, or for domain validation for certificates). Talk to your Account Representative for more details on these uses cases.

Partner Domain Suffix

Input the appropriate domain suffix. This is typically the same value you've input for use as your property hostname. We use this value internally to support two things:

- Domains need to be flagged as trustworthy or acceptable ("whitelisted") for access by our Edge servers. Domain information is typically gathered from the Origin Server information defined in a Property Manager configuration. However, since an origin can be set dynamically via a policy in ACE, we whitelist the suffix you input here, and add it internally to all origin domains for this purpose.

- We use VIP-based slot-matching in secure ACE configurations. As a result, URL Purge doesn't work for vanity domains. So, you need to use this suffix to the domain names while making purge requests. For example, to purge `abc.com/somepath` should be requested as `abc.com.<partner domain suffix>/somepath`.

Optional Subcustomer Enablement Settings: Quick Reference

The remaining settings here can be enabled to customize these settings on a per-Subcustomer basis, by generating a policy using the ACE API.

Setting	Description	Additional Reference
Origin	• **Set to "On"**: An individual Subcustomer's policy can include	

Setting	Description	Additional Reference
	settings to define a custom origin server, with all associated settings. (What's set for a policy's origin overrides what's set for the Origin Server behavior. If no origin settings are applied in a policy, what's set for the Origin Server behavior is used.) • **Set to "Off"**: A Subcustomer policy cannot include origin server information. This applies if you want to use the origin you've set as the Origin Server behavior for all of the Subcustomers that are associated with this base configuration.	
Caching	• **Set to "On"**: You can specify caching settings in an individual Subcustomer's policy. What you set for a policy caching settings will override what you've set for the Caching behavior here in the base configuration. (If caching settings are not applied in a policy, what's set for the Caching behavior is used.) • **Set to "Off"**: You can't change caching settings for a Subcustomer via a policy. What is set here in the base configuration for the Caching behavior will be used for all Subcustomers.	

Setting	Description	Additional Reference
Referrer Allow/Block	Set this to "**On**" to allow dynamic setup of a referrer whitelist (allow referrers) or blacklist (deny referrers) in the policy for each Subcustomer.	
IP Allow/Block	Set this to "**On**" to allow dynamic setup of an IP whitelist (allow specified IP addresses) or blacklist (deny specified IP addresses) in the policy for each Subcustomer. Set to "Off" to block the inclusion of IP whitelists/blacklists in a Subcustomer policy.	
Geo Allow/Block	Set this to "**On**" to allow the inclusion of a geographic regions whitelist (allow specified continents, countries, regions and mesignated market areas—DMAs) or blacklist (deny specified geographic regions) in the policy for each Subcustomer.	
Content Refresh	If you set this to "**On**" you can set up content revalidation schedules for each Subcustomer in the associated policy.	
Modify Forward Path	When set to "**On**", you can dynamically modify the request path to the content server for each Subcustomer in the associated policy.	
Cache Key Query Arguments	When set to "**On**" you can apply various settings in Subcustomers' policies to customize which query string arguments can be included in the Akamai Edge server cache entry (cache-key).	
Token Authentication	Set this to "**On**" to configure whether or not Edge servers can control access to Subcustomer content through the use of tokens. (The token can be transmitted in the client request	

Setting	Description	Additional Reference
	in a cookie, header, or query-parameter.)	
Site Failover	Set this to "**On**" to allow configuration of a unique failover site in each Subcustomers' policy.	
Content Compressor	Set this to "**On**" to configure file compression on a per Subcustomer basis, via their unique policy.	
Access Control	Set to "**On**," you can deny requests to a Subcustomer's content based on certain match conditions.(You set up these match conditions in the Subcustomer's policy.)	
Dynamic Web Content	When set to "**On**" you can apply various settings to Subcustomers' policies to support and streamline the delivery of dynamic web content.	*How to add dynamic web content support for a Subcustomer*
Streaming Video On-demand Delivery	Set this switch to "**On**" to allow addition of streaming on-demand video support via a Subcustomer policy.	*How to add streaming video support for a Subcustomer*
Large File Delivery	When set to "**On**" you can apply various settings in Subcustomers' policies to support and streamline the delivery of large files (up to 1.8 GB in size).	*How to add large file delivery support for a Subcustomer*

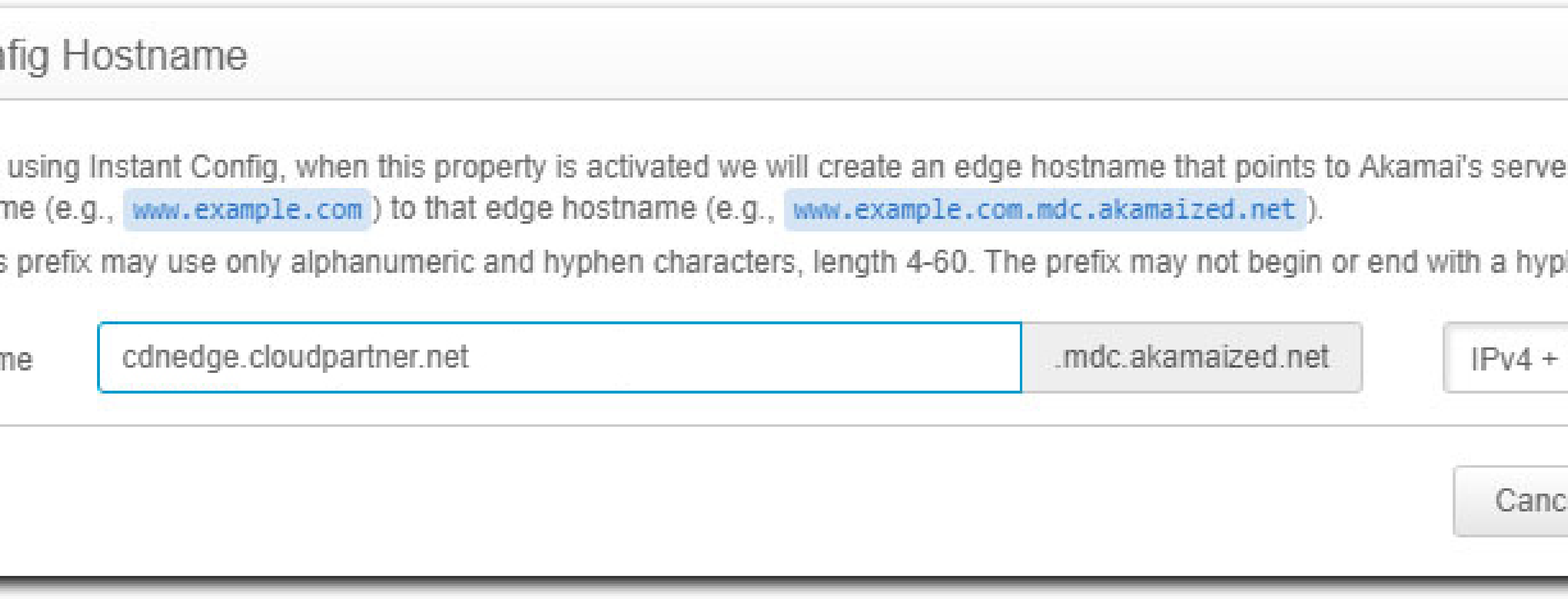
The InstantConfig behavior

Enable this option to apply settings in this base configuration to all incoming requests for subcustomer hostnames, without explicitly adding those hostnames to the base configuration.

Once enabled, you don't need to manually add each applicable hostname as a Property Hostname to this base configuration. However, *it only applies to non-secure (HTTP) hostnames*. So, this base configuration can only process HTTP requests.

How do I set this up?

To start, you need to set up a non-secure HTTP Property Hostname. For example, you could use the **Add Instant Config Hostname** functionality via the **Property Hostnames** content panel.


nfig Hostname

using Instant Config, when this property is activated we will create an edge hostname that points to Akamai's serve
me (e.g., `www.example.com`) to that edge hostname (e.g., `www.example.com.mdc.akamaized.net`).
s prefix may use only alphanumeric and hyphen characters, length 4-60. The prefix may not begin or end with a hyp

ne cdnedge.cloudpartner.net .mdc.akamaized.net

With the base configuration saved, you need to do the following:

1. Create an endpoint for subcustomers, and have them CNAME to this endpoint. For example, a subcustomer "abc.com," would need to CNAME to your endpoint, "`abc.cdnedge.cloudpartner.net`."
2. Create a wildcarded (*) DNS record to CNAME all of your subcustomers to the Edge hostname created with the non-secure HTTP Property Hostname you set up in this base configuration. For example, "`*.cdnedge.cloudpartner.net`" would need to CNAME to "`cdnedge.cloudpartner.net.mdc.akamaized.net`."

You only need to perform both of the steps above a single time, unless you need to change the hostname.

The Allow POST behavior

Use this behavior to set whether POST requests sent to your specified Origin Server should be accepted by an Edge server.

By default, we allow requests to your Origin using HTTP GET or HEAD methods. Set this switch to "On" if you also want to accept POST method requests to your Origin Server.

Tip: If all requests to an Origin Server—either a fixed one you've set here in the base configuration, or an "override" one you define in an individual subcustomer's policy—are non-POST, consider setting this switch to "**Off**" to deny POST requests and potentially improve security.

Allow without Content-Length

If desired, you can allow POST calls without a Content-Length header.

By default, a POST request to an Edge server must contain a Content-Length header. (This is an HTTP RFC standard.) The server will return an "HTTP 411 - Length required" error, even if a POST request *does not* include a body (and therefore, shouldn't require a Content-Length Header).

If you enable this option, the Edge server assumes a POST request without a Content-Length header has no body. So, it adds a Content-Length header with the value of "0" to the forward request, *unless the request is using Transfer Encoding*.

When Transfer Encoding is used, a request can exclude the Content-Length header.

The Allow PUT behavior

Use this behavior to set whether `PUT` requests sent to your specified Origin Server should be accepted by an Edge server.

By default, we allow requests to your Origin using HTTP `GET` or `HEAD` methods. Set this switch to "**On**" if you also want to accept PUT method requests.

Note: If `PUT` is used, content is not cached.

The Allow DELETE behavior

Use this behavior to set whether `DELETE` requests sent to your specified Origin Server should be accepted by an Edge server.

By default, we allow requests to your Origin using HTTP `GET` or `HEAD` methods. Set this switch to "**On**" if you also want to accept `DELETE` method requests.

Note: If `DELETE` is used, content is not cached.

Allow Body

Set this to "**On**" if your setup requires inclusion of a message body with a `DELETE` request. (This is not common.)

Auto Domain Validation

This behavior enables the automatic renewal of Standard TLS Domain Validated certificates.

With Domain Validation (DV), the applicable certificate authority (CA) validates that you have control of the domain. (DV is the lowest level of validation.) The Certificate Provisioning System supports DV certificates issued by Let's Encrypt, an automated, and open CA that is run for public benefit. Certificate expiration is typically as follows:

- Akamai-managed DV certificates expire in 90 days.
- Renewals for Akamai-managed DV certificates start 16 days prior to expiration.
- A third party, customer-supplied, DV certificate can expire whenever the applicable certificate authority determines it expires; this behavior is not necessary for customer-supplied DV certificates.

When should I include this behavior?

If you are using Standard TLS DV certificates for the hostnames in this property, you should include this behavior to enable automatic renewal of the certificate. If you leave this behavior out, the certificate could expire, and HTTPS traffic will be served with certificate errors.

This behavior is not required for any Enhanced TLS certificates.

How is this behavior supported?

You can include this behavior in your property in multiple ways:

- **You can include it in the Default Rule**. In this case, it is applied to all requests for all resources associated with this property.
- **You can include it in a supplemental rule**. This allows you to set up a custom rule that only applies to specific requests for resources associated with this property. This rule must use only the "Hostname" condition match criteria.
- **It can be applied in multiple rules**. Rule priority applies, with rules lower in the order taking precedence.
- **There might be an issue if an incoming request matches another "redirect" behavior**. Assume that the incoming request matches another behavior you have in your property that results in a redirect operation similar to what applies with this behavior. If so, the operation that takes precedence depends on where the behavior is in the property.
  - If you are using a similar behavior, ensure that behavior exists in a rule that is higher in ordering.
  - You should test on your configuration on staging by making a request to `www.yourdomain.com/well-known/acme-challenge/some_random_token`.

You can configure optional behaviors

These behaviors depend on what's generally available for use with ACE, as well as any optional behaviors you may have purchased separately.

Note: Some of the additional features are simple, while others require specialized knowledge. It is beyond the scope of this document to list and discuss all available features and options. Behaviors in the Property Editor in Control Center offer "**?**" buttons you can use to obtain more information, or you can contact your Account Representative for guidance.

Content Characteristics - Dynamic Web Content

This behavior and its options allow support for Integrated Cloud Acceleration (ICA). ICA allows you to add more support for Dynamic Web Content for your customers ("Subcustomers") via your Akamai Cloud Embed CDN.

Prerequisites

- **You need ICA added to your contract**. Talk to your account representative about adding this to your contract to use this support.
- **You need "Dynamic Web Content" enabled**. To include the Content Characteristics - Dynamic Web Content behavior, you must *enable* the **Dynamic Web Content** option in the Sub-Customer Enablement behavior for an ACE base configuration. Also, the Content Characteristics - Dynamic Web Content behavior must be added to the *same rule* that contains the Sub-Customer Enablement behavior, in which you've enabled the Dynamic Web Content option.

What do these options do?

If you enable any of these options, you can configure them for individual Subcustomers. For example, if you enable SureRoute, you can granularly apply settings for it—enable it, disable it or apply other settings—for each Subcustomer. This configuration is done by generating a policy for that Subcustomer using the ACE API, and defining settings via the `content-characteristics` behavior.

If an option is left at "**Off**," you can't configure settings for it via a Subcustomer policy.

See *Add dynamic web content support for a subcustomer* on page 76 for full details on use.

Content Characteristics - Streaming Video On-demand

This behavior allows you to set various optimizations for the delivery of on demand video for your Akamai Cloud Embed (ACE) CDN customers (our "Subcustomers").

Note: This behavior is used exclusively in base configurations for the Akamai Cloud Embed (ACE) product.

Prerequisites

- **You need "Streaming Video On-demand Delivery" enabled**. To include the Content Characteristics - Streaming Video On-demand behavior, you must *enable* the **Streaming Video On-demand Delivery** option in the Sub-Customer Enablement behavior for an ACE base configuration. Also, the Content Characteristics - Streaming Video On-demand behavior must be added to the *same rule* that contains the Sub-Customer Enablement behavior, in which you've enabled the Streaming Video On-demand Delivery option.

What do these options do?

Use the settings here to optimize video on-demand streaming for *all* Subcustomers registered with this ACE base configuration.

You can override any of the settings on a per-Subcustomer basis by generating a policy for that Subcustomer using the ACE API, and defining settings via the `content-characteristics` behavior.

Note: If a media format—**Enable HLS**, **Enable DASH**, etc.—is set to "**Off**" in this behavior, it is *disabled for all Subcustomers registered with this base configuration*. (You cannot configure settings for it via a Subcustomer policy with the ACE API.)

See *Add on demand video support for a subcustomer* on page 86 for full details on use.

Content Characteristics - Large File

This behavior allows you to optimize the delivery of large files for your Akamai Cloud Embed (ACE) CDN customers (our "Subcustomers").

Note: This behavior is used exclusively in base configurations for the Akamai Cloud Embed (ACE) product.

Prerequisites

- **You need "Large File Delivery" enabled**. To include the Content Characteristics - Large File behavior, you must *enable* the **Large File Delivery** option in the Sub-Customer Enablement behavior for an ACE base configuration. Also, the Content Characteristics - Large File behavior must be added to the *same rule* that contains the Sub-Customer Enablement behavior, in which you've enabled the Large File Delivery option.

What do these options do?

Use the settings here to optimize delivery of large files for *all* Subcustomers registered with this ACE base configuration.

You can override the **Origin Object Size** setting you define in this behavior on a per-Subcustomer basis by generating a policy for that Subcustomer using the ACE API, and using the appropriate settings in the `content-characteristics` behavior.

See *Add large file delivery support for a subcustomer* on page 98 for full details on use.

Real-time Reporting

Once added to your base configuration and enabled, this behavior gathers reporting data for Akamai Cloud Embed (ACE) at near real-time latencies—typically, less than 15 minutes. This lets you generate related reports at an aggregation interval of one minute.

Get this provisioned for use

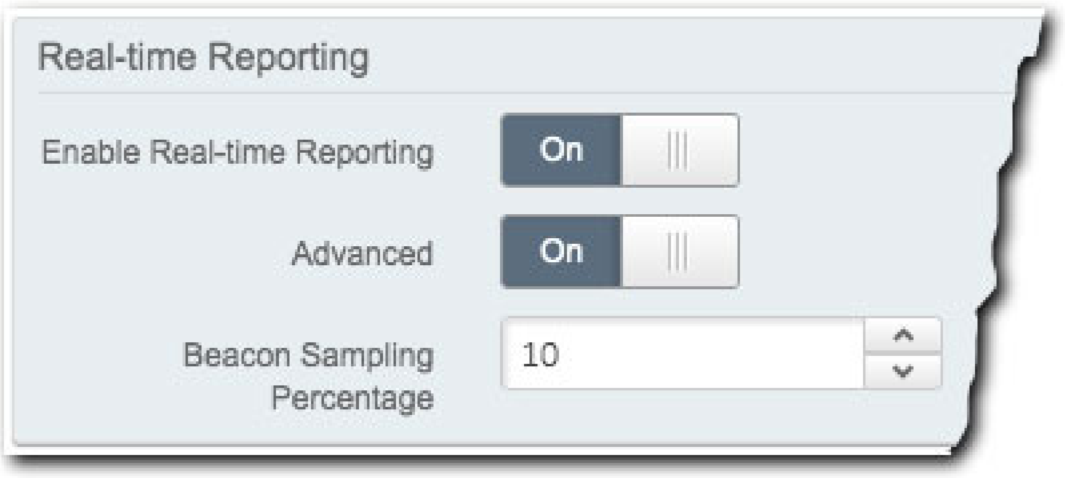
This must be provisioned and added by your account representative ("rep"). Your account rep works with you to determine the applicable ACE base configuration, or create a new one to add this support. Once provisioned, you can view it as a behavior in that base configuration. (You can't edit these settings or remove this behavior. Only your account representative can do this.)

The settings apply as follows:

- **Enable Real-time Reporting**. When enabled, you have access to Real-time reports for ACE, via Media delivery reports in Control Center or the Media Delivery Reports API. Reports data is available at a latency of under 15 minutes.

- **Advanced**. Offers access to additional options. (It must be enabled to get access to the **Beacon Sampling Percentage** option.)

- **Beacon Sampling Percentage**. How much of your overall ACE traffic should be polled to collect data. Your account rep can access and set this option to best address the capacity and limitations of the downstream systems that receive and process data for your real-time reports. (A capacity is set in an attempt to reliably deliver and process data at the expected 15 minute latency, for your environment.) If this is not specifically set, the default is 10%.


Real-time Reporting

Enable Real-time Reporting: On

Advanced: On

Beacon Sampling Percentage: 10

What does the behavior let me do?

With provisioning completed and the behavior added, you can access real-time reports data for that base configuration from either the Akamai Control Center or the Media Delivery Reports API.

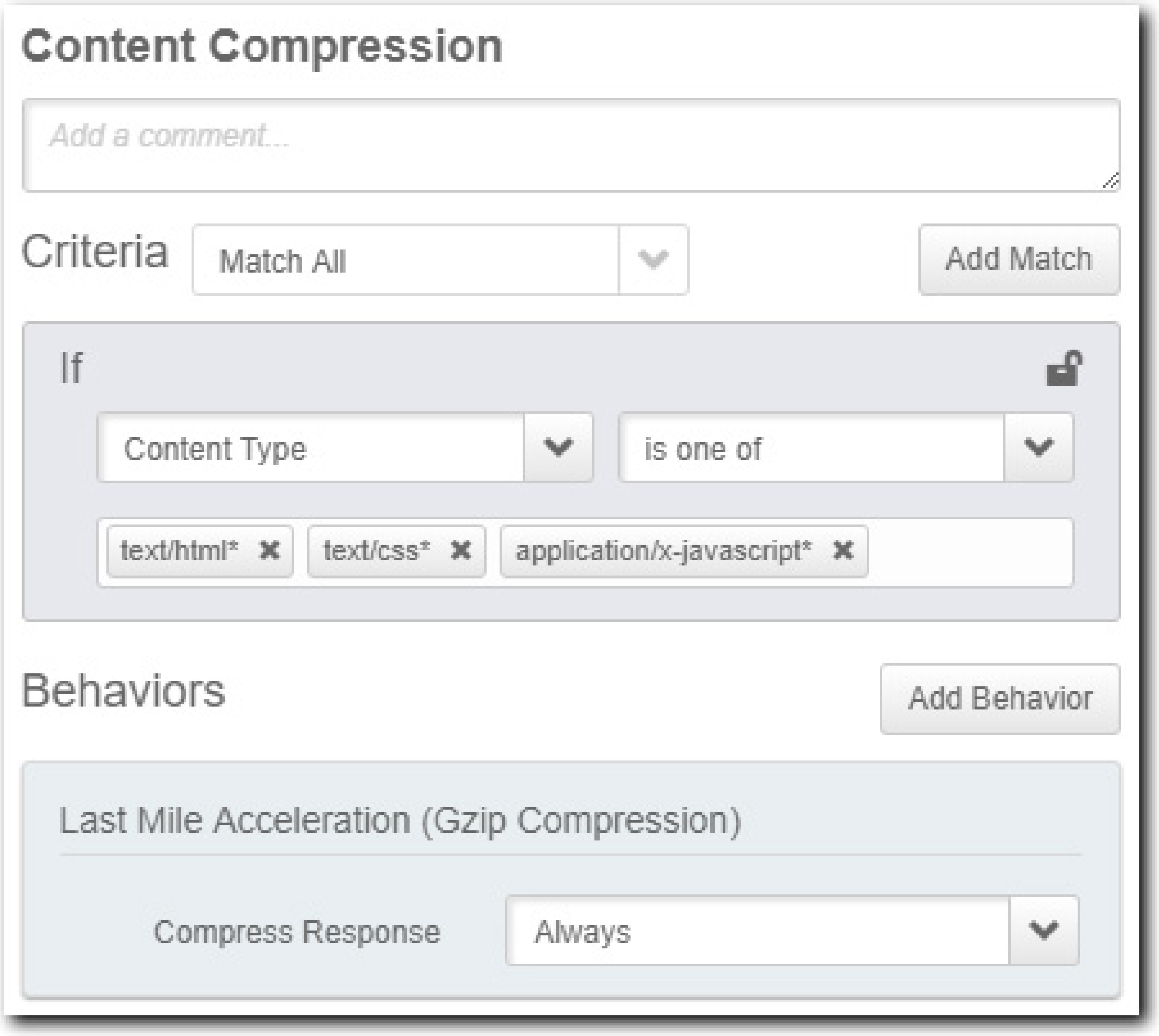
- **Control Center**. Go to ☰ > **MEDIA** > **Media delivery reports**, and then select **Akamai Cloud Embed** > **Realtime** from the menu.
- **Media Delivery Reports API.** This gives you access to the "*Get Akamai Cloud Embed Real-time data*" operation in the API.

Review the other rules

Once you've reviewed and configured the behaviors in the Default Rule, it's important to review each of the additional rule-based behaviors that are also automatically included in your base configuration. For each of these, you should check that the behavior is appropriate, and that the match criteria is properly scoped.

The Content Compression rule

This rule checks values in the `Content-Type` header an origin response and enables compression for the specified content.


Content Compression

Add a comment...

Criteria: Match All

If Content Type is one of text/html* text/css* application/x-javascript*

Behaviors

Last Mile Acceleration (Gzip Compression)

Compress Response: Always

Criteria

You should review the list of Content Types in the match and update the list as needed. For example, you may prefer not to compress JavaScript content, so you would remove the **application/x-javascript** content type from the listing. If you click in the blank area of the listing, you'll see that text/xml and text/plain are already suggested additions to the match values. You can click the blank area of the listing and type any `Content-Type` string to add it to the list. (It appears in the drop down, accompanied by "(new item)." Click the entry to add it.) Use the wildcard (*) at the end of your string to ensure that additional characters in the `Content-Type` definition don't prevent the match from succeeding.

The Last Mile Acceleration (Gzip Compression) behavior

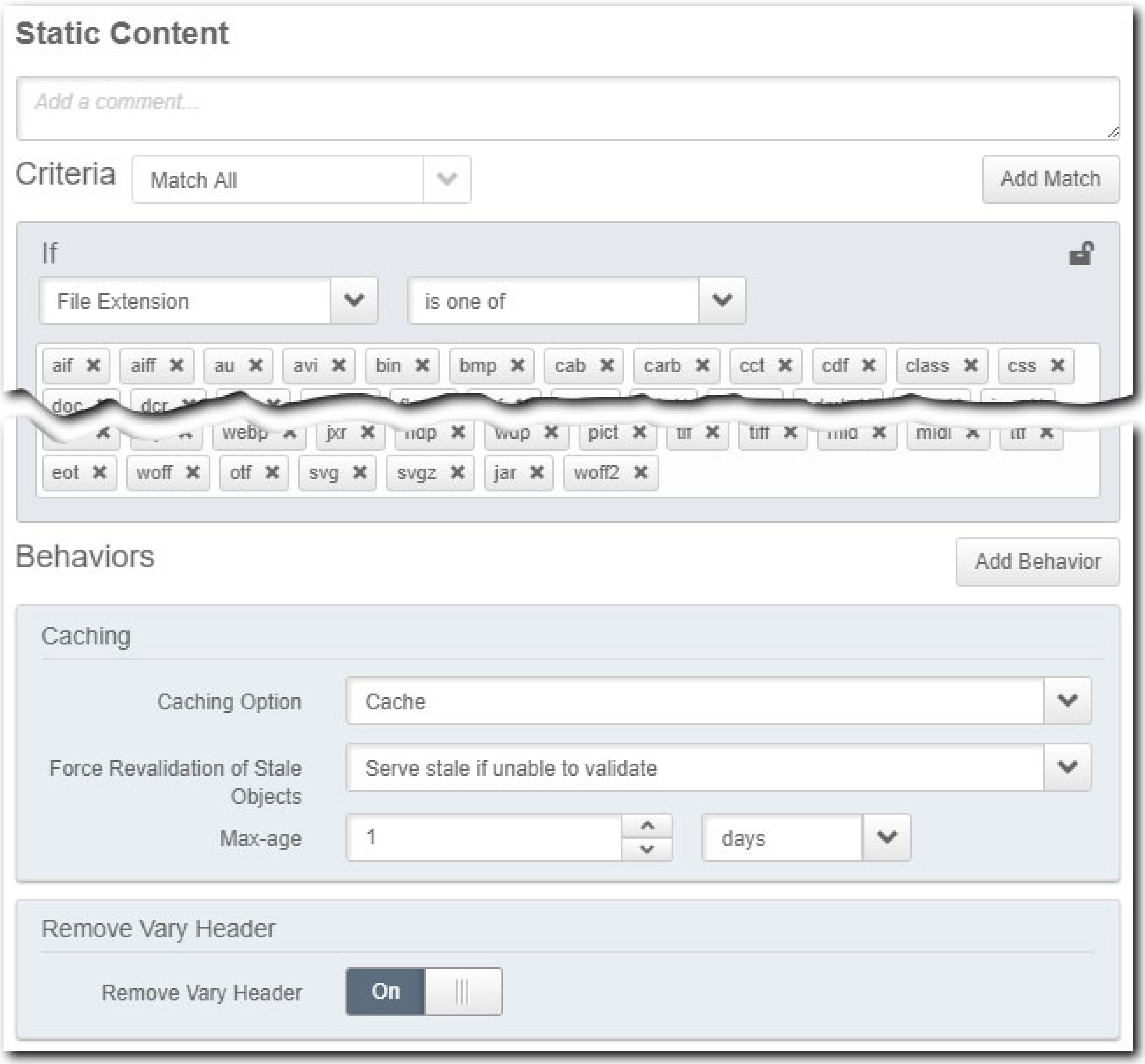
The Compress Response option is set to **Always** by default. The specified Content Types are compressed when the requesting end-user agent (browser or device type) supports Gzip unzipping. This can improve transfer times to clients with slow connections. Consider the following if you want to enable Last Mile Acceleration (LMA):

- **LMA is only useful for certain Content Types**. This includes HTML, JavaScript, or UTF-8 character encoded, and then when the content is larger than roughly 10 KB.

- **LMA should be avoided with other Content Types (outside of what's listed above)**. For example, don't use it to compress images that are already compressed, or to compress small files that require more time to compress, send, and unzip, than would actually be saved in transmission time.
- **You can also choose to use what's set in your origin response headers for compression**. If your origin headers already explicitly describe how compression shoiuld be handled for your content, set this to **Same as origin response**.

The Static Content rule

This rule adjusts the cacheability settings based on the file extension of requested content.



Static Content

Add a comment...

Criteria: Match All

If: File Extension — is one of

aif, aiff, au, avi, bin, bmp, cab, carb, cct, cdf, class, css, doc, dcr, …, webp, jxr, hdp, wdp, pict, tif, tiff, mid, midi, ttf, eot, woff, otf, svg, svgz, jar, woff2

Behaviors

Caching

Caching Option: Cache

Force Revalidation of Stale Objects: Serve stale if unable to validate

Max-age: 1 days

Remove Vary Header

Remove Vary Header: On

Criteria

You should carefully review the list of file extensions in the match criteria to make sure they represent cacheable content for subcustomers' sites. Add or remove new extensions as necessary. *Only the extensions set here will have the Behaviors in this rule applied in regards to caching.*

The Caching behavior

If appropriate, you should also adjust the settings in this rule's **Caching** behavior. (Settings are identical to the **Caching** behavior set in the Default Rule.)

Important: The settings in this rule override whatever you may have set for the **Caching** behavior in the Default Rule.

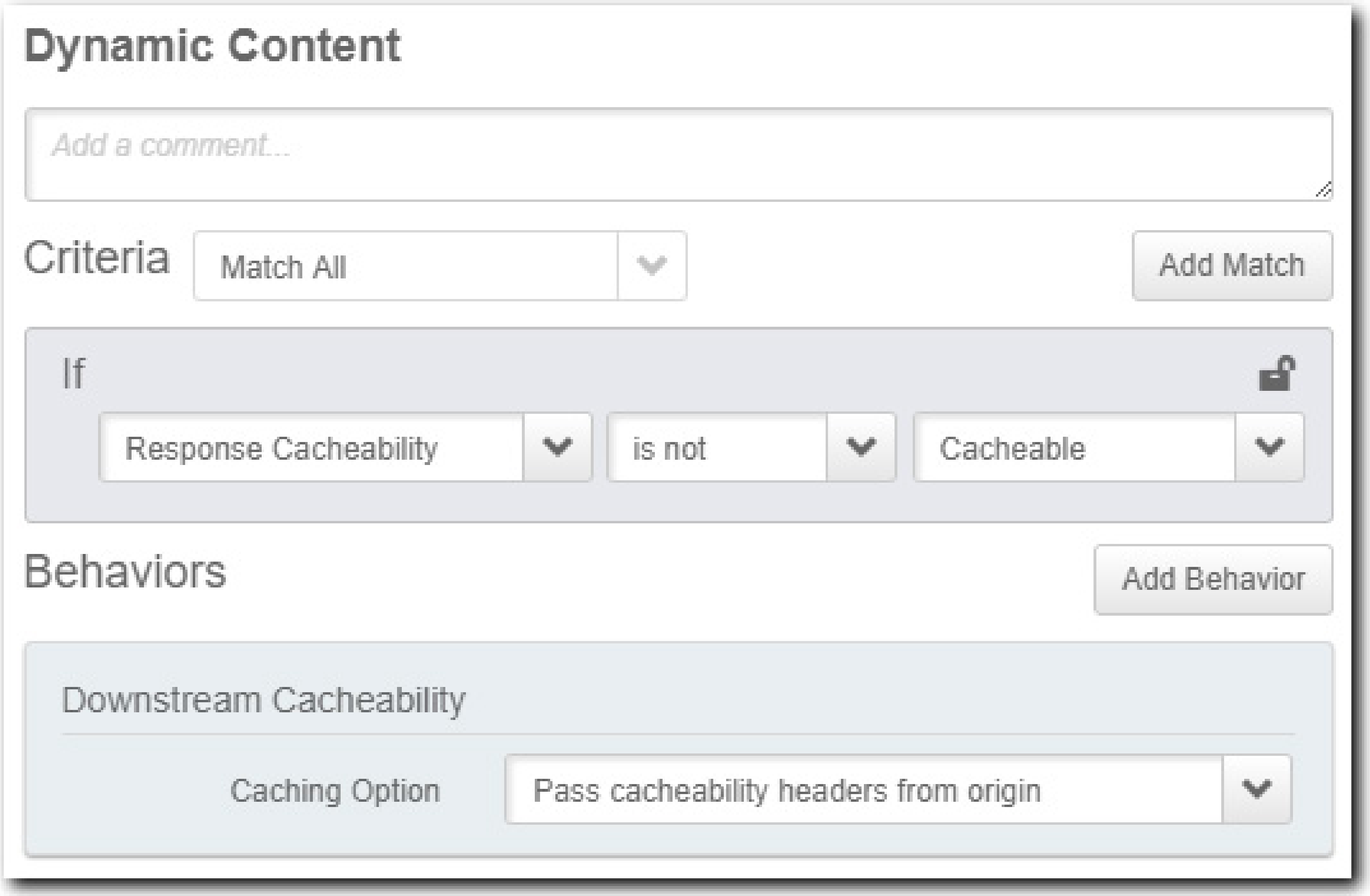
The Remove Vary Header behavior

By default, an Edge server fielding a request from a subcustomer site will not cache an object with an HTTP Vary header in the response, unless that response is a compressed object, and the response headers include both `Vary: Accept-Encoding` and `Content-Encoding: gzip` headers. If this is always the case with subcustomers, set this switch to **Off**.

However, some applications add the HTTP Vary header *even when the content does not vary*. In that case, set this switch to **On** to have the Edge server remove the header so that the content may be cached.

The Dynamic Content rule

This rule lets you adjust how cacheability headers (`Cache-Control` and `Cache-Expires`) should be sent.


Dynamic Content

Add a comment...

Criteria: Match All

If Response Cacheability is not Cacheable

Behaviors

Downstream Cacheability

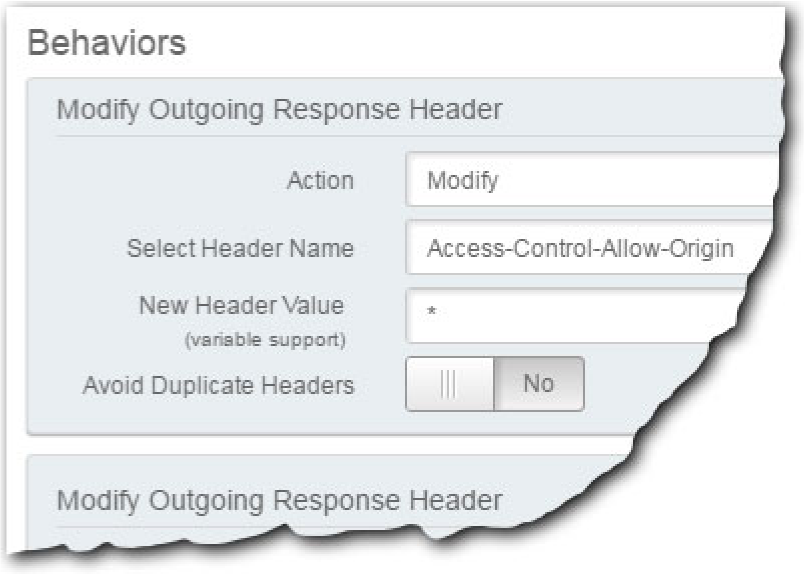
Caching Option: Pass cacheability headers from origin

By default, options are set so that any response that is *not cacheable* on the Edge server is delivered to the client using the what a subcustomers has configured in the `Cache-Control` and `Cache-Expires` headers sent by the origin server.

The Default CORS Policy rule

This rule includes multiple instances of the **Modify Outgoing Response Header** behavior, predefined with recommended values. (They are applied as "recommended behaviors.")

Each individual header is populated with recommended values as follows:


Behaviors
Modify Outgoing Response Header

Action	Modify
Select Header Name	Access-Control-Allow-Origin
New Header Value (variable support)	*
Avoid Duplicate Headers	No

Modify Outgoing Response Header

- **Access-Control-Allow-Origin**: This is populated with the value “*” to indicate "all."
- **Access-Control-Allow-Methods**: This is populated with the following request methods, GET, POST and OPTIONS.
- **Access-Control-Allow-Headers**: This is populated with the values, “origin”, “range”, “hdntl”, and “hdnts.”
- **Access-Control-Expose-Headers**: This is populated with the values, “Server”, “range”, “hdntl”, and “hdnts.”
- **Access-Control-Allow-Headers**: This is set to “true.”.
- **Access-Control-Max-Age**: This is set to “86400” seconds (or 24 hours).

This rule and its Behaviors are not mandatory, and can be removed (via the “**X**” icon in the Rule itself, or in each Behavior), but it is recommended that they be left as is.

Set up subcustomers via the ACE API

Once you have your ACE base configuration set, you need to perform several operations via the ACE API to configure subcustomers.

The ACE API version

The current version of the ACE API is 2.0. All services discussed in this documentation use this version of the API.

Note: Other functions and services offered for use with ACE may use legacy or newer versions of the API. They are covered in separate documents, where the required version is specifically called out.

Before you begin with the API

To get up and running with the ACE API, you need to do a few things.

- To use this API and its core delivery, download, and video streaming capabilities, you need to add ACE to your contract. To use the acceleration features, you have to sign up for Integrated Cloud Accelerator (ICA).
- If your subcustomers support *both* HTTP and HTTPS traffic, contact Akamai to create a secure ACE base configuration and provision the necessary certificates. Akamai recommends this configuration for ACE. You can create this configuration yourself, using Property Manager in *Akamai Control Center*.
- If your subcustomers only support HTTP traffic, contact Akamai to request a multi-domain edge hostname and create an InstantConfig property. A multi-domain edge hostname lets you use a single hostname to represent multiple web assets.
- Create an origin hostname. You need to create a hostname to identify a default origin server that is separate from your subcustomer cloud origins.
- Create an error page. Once you create a hostname to identify a default origin server, configure this default origin to serve an error page when subcustomers don't configure their domain correctly.
- Determine the method to use to create subcustomer IDs. Each customer in your billing system who uses your ACE implementation needs this ID. You can use any method to generate IDs as long as the IDs are unique. Akamai rolls all traffic for a particular subcustomer into this ID for billing.
- Add the *Subcustomer Enablement behavior* to your base configuration via Property Manager in Akamai Control Center. Once added, you need to decide which ACE features you want to offer to your subcustomers, and enable them accordingly. Your decisions determine the structure of your policy JSON going forward. (Your policy JSON is made up of rules that include both match criteria and behaviors for use with that specific subcustomer.)
- You need your `propertyId`. This is a unique value that is automatically generated for a base configuration after you create it. This value is required in all ACE API operations. Contact your Account Representative to get this value.

- If you're using ICA, make sure you enable the *Dynamic Web Content* option in the Sub-Customer Enablement behavior.
- If you're using a non-secure (HTTP), Multi-Domain Edge Hostname, enable the *InstantConfig* behavior in your base configuration via Property Manager. InstantConfig lets you associate multiple web assets to a single property without adding each hostname to the base configuration.
- See the *Akamai Developer documentation* for details on how to set up and apply authentication credentials and choose a supported developer interface to access the ACE API. To do this, you'll use the Identity and Access Manager interface in Akamai Control Center. During the process, you need to:
  - **Verify you have the API service named Wholesale Delivery**. "Wholesale Delivery" is the legacy name for Akamai Cloud Embed.
  - **Set the API access to READ-WRITE**. This gives you access to all of the operations in this API.
  - **Note your Group name**. Click **Show additional details** and review the Groups table to see a list of groups. Make note of the Group name for your applicable group.
- We recommend that your review the *API concepts* in *The Akamai Cloud Embed API v2* on page 105topic to familiarize yourself with some of the common terms used in the API.

Register subcustomers

You must generate and register an ID for each subcustomer you wish to add to your base configuration. A subcustomer ID is a mandatory element in various ACE API calls.

Before you begin: You need subcustomer IDs

When registering a subcustomer, you need to provide a unique ID—a "subcustomer ID". As the cloud partner, it's up to you to determine a method to create these IDs and associate them with your subcustomers.

Each subcustomer in your billing system who uses your Akamai Cloud Embed implementation needs an ID. All traffic for a particular subcustomer is rolled up by this ID for billing.

subcustomer IDs have the following requirements and limitations:

- Each subcustomer must have a unique ID.
- The ID cannot be more than 50 characters.
- The ID can only consist of alphanumeric characters (a-z, A-Z, 0-9), dashes (" - ") or periods (" . ")

You should define subcustomer scope carefully, because it helps a great deal with billing and reporting.

How do I register a subcustomer ID?

This is accomplished by running the "**Add a new subcustomer**" operation via the ACE API. You'll need the following values to run this operation:

- `propertyId`: This is a unique ID value that is generated and associated with a base configuration. Contact your Account Representative for this value.

- `sub-customerId`: The ID you created for the subcustomer you wish to register with the named base configuration (`propertyId`).

See the ACE API docs

Call formatting and other specifics are covered in the ACE *API documentation*.

List all of your subcustomers

You can view basic details for all of the subcustomers you've registered for a specific base configuration. This includes the geographic region ("geo") and the digital properties associated with the subcustomer.

This is accomplished by running the "**List all subcustomers**" operation via the ACE API. You'll need the following values to run this operation:

- `propertyId`: This is the unique ID value associated with the base configuration that houses the subcustomers you want to list. Contact your Account Representative for this value.
- `network`: This is the network version of subcustomers you want to list: production (live) or staging (testing).

Call formatting and other specifics are covered in the ACE *API documentation*.

View a specific subcustomer

You can implement an API operation to view the geographic region ("geo") associated with a specific subcustomer.

This is accomplished by running the "**Get a subcustomer**" operation via the ACE API. You'll need the following values to run this operation:

- `propertyId`: This is the unique ID value associated with the base configuration that houses the target subcustomer. Contact your Account Representative for this value.
- `sub-customerId`: The ID associated with the subcustomer you want to view.
- `network`: You need to decide which network version of the subcustomer you want: production (live) or staging (testing).

See the ACE API docs
Call formatting and other specifics are covered in the ACE *API documentation*.

Remove a subcustomer

If you no longer need a subcustomer you've previously registered, you can use an API operation to remove it.

This is accomplished by running the "**Remove a subcustomer**" operation via the ACE API. You'll need the following values to run this operation:

Note: Take caution when removing a subcustomer from the production network. This will affect any live access to that subcustomer.

- `propertyId`: This is the unique ID value associated with the base configuration that houses the target subcustomer. Contact your Account Representative for this value.

- `sub-customerId`: The ID associated with the subcustomer you want to remove.
- `network`: You need to decide which network version of the subcustomer you want to remove: production (live) or staging (testing).

Call formatting and specifics are covered in the *API documentation*.

List all sub-properties for a base configuration

A sub-property (`subPropertyID`) is another name for a domain associated with a specific subcustomer. You can use the API to request a list of all of the sub-properties for all subcustomers you've registered with a base configuration.

This is accomplished by running the "**List all sub-properties for a base configuration**" operation via the ACE API. You'll need the following values to run this operation:

- `network`: You need to decide which network version of the base configuration you want: production (live) or staging (testing).
- `propertyId`: This is the unique ID value associated with the target base configuration. Contact your Account Representative for this value.

Call formatting and specifics are covered in the *API documentation*.

List all domains for a subcustomer

If you've already registered a subcustomer with a base configuration, you can call it to list all of the domains (`subPropertyID`) associated with it.

This is accomplished by running the "**List all subcustomer domains**" operation via the ACE API. You'll need the following values to run this operation:

- `network`: You need to decide which network version of the base configuration you want: production (live) or staging (testing).
- `propertyId`: This is the unique ID value associated with the base configuration that houses the target subcustomer. Contact your Account Representative for this value.
- `sub-customerId`: The ID associated with the subcustomer you want to target.

Call formatting and specifics are covered in the *API documentation*.

Create a delivery policy for each subcustomer

Each subcustomer you register for use with your base configuration must have its own delivery policy, to define various settings specific to that subcustomer.

What is a delivery policy?

A delivery policy is a set of rules that consist of match criteria and behaviors that supplement those you've set in the base configuration, but they only apply to the specific subcustomer named in the operation to create the delivery policy.

A delivery policy is laid out in a request body component, and it should include at least the following:

- **An “origin” behavior**. This requires that you include the subcustomer's hostname (`digitalProperty`) as well as the origin hostname (`originDomain`) where the subcustomer's content is located. (If left out, what you've set for the Origin Server behavior in the base configuration will serve as this subcustomer's origin.)
- **A time to live setting (TTL) for content**: Some sites may require this.
- **A geographic restriction**: Include this to prevent serving content into some countries. Some sites may require this.

How do I create a Policy?

This is accomplished by running the "**Create or update a policy**" operation via the ACE API. You'll need the following to complete this operation:

- `propertyId`: This is included in the request syntax. This is the unique ID value associated with the base configuration that houses the subcustomer for this delivery policy. Contact your Account Representative for this value.
- `domainName`: This is included in the request syntax. This is the hostname (domain) associated with the subcustomer that's using this delivery policy.
- `network`: This is included in the request syntax. You need to decide the network where the delivery policy is applied: production (live) or staging (testing). We recommend that you use the staging network, so you can test the delivery policy. You can later update the delivery policy to production to "go live" with it.
- **A properly formatted request body**: You also need to include a JSON-formatted document of rules comprised of match criteria and behaviors to apply to requests for this subcustomer's content—If a request adheres to the defined match criteria, the behaviors set in the delivery policy are applied. A host of match criteria ("matches") and behaviors are available for use.

Note: You can disable or enable behaviors for all policies associated with the applicable base configuration, using the *Subcustomer Enablement* behavior. For example, if you set Token Authentication to "Off" in your base configuration, subcustomers cannot configure this behavior in a delivery policy.

Instructions on the "create or update a policy" operation

We offer some topics here in this guide that give you usage guidelines as well as some examples, and we recommend that you review these topics:

- *Example request body for a policy* on page 54
- *How to structure a rule* on page 55
- *View a specific policy* on page 69
- *Delete a policy* on page 70
- *Review a policy's history* on page 70
- *Map expansion and ACE API operations* on page 70
- *A simple test of a policy* on page 71

However, all call formatting specifics are covered in the ACE *API documentation*.

Example request body for a policy

Each "create or update a policy" operation via the ACE API requires a properly formatted request body.

This example assumes you're creating a policy that does the following for a new subcustomer whose hostname is `www.example.com`:

- **It defines an origin domain**. The subcustomer origin hostname is `c1234567.cloudprovider.com`. Also, we want to ensure that the incoming request `Host` header is sent to the origin.
- **It sets a default caching policy**. This is set to `no-store`, which presumes HTML pages and other content not defined as cacheable elsewhere in the policy is dynamic in nature.
- **It applies a custom one day TTL for certain content**. This is applied for all PNG, GIF and JPG images in the `/static/*` directory.
- **It applies another separate one hour TTL**. This is applied for all CSS and JS files in the `/static/*` directory
- **It blocks clients in North Korea**.

If so, the JSON you must include with this operation might look like the following:

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-wildcard",
                    "value" : "/*"
                }
            ],
            "behaviors" : [
                {
                    "name" : "origin",
                    "value" : "-",
                    "params" :
                        {
                            "digitalProperty" : "www.example.com",
                            "originDomain" : "c1234567.cloudprovider.com",
                            "cacheKeyType" : "origin",
                            "hostHeaderType" : "digital_property",
                            "hostHeaderValue" : "-",
                            "cacheKeyValue" : "-"
                        }
                    },
                {
                    "name" : "caching",
                    "type" : "no-store",
                    "value" : "-"
                },
                {
                    "name" : "geo-blacklist",
                    "type" : "country",
                    "value" : "KP"
                }
            ]
        },
        {
            "matches" : [
                {
                    "name" : "url-wildcard",
                    "value" : "/static/*"
                },
                {
                    "name" : "url-extension",
                    "value" : "png gif jpg"
                }
            ],
            "behaviors" : [
                {
                    "name" : "caching",
                    "type" : "fixed",
                    "value" : "86400s"
                }
            ]
        },
        {
            "matches" : [
                {
                    "name" : "url-wildcard",
                    "value" : "/static/*"
                },
                {
                    "name" : "url-extension",
                    "value" : "css js"
                }
            ],
            "behaviors" : [
                {
                    "name" : "caching",
                    "type" : "fixed",
                    "value" : "3600s"
                }
            ]
        }
    ]
}
```

The response, if successful, will repeat the JSON structure representing the complete (unformatted) policy.

How to structure a rule

A policy takes the form of `rules` composed of `matches` and applicable `behaviors`. The match criteria must be set first, followed by the desired behaviors to apply. You need to know how to properly format a rule.

Once the criteria set in a match is met, associated behaviors are applied to a request. You can nest matches in an individual rule, and you can have multiple behaviors in a single rule. You can also include multiple rules —match and behavior combinations—in a single policy.

The format for an individual rule in a policy follows a specific format:

```
{
    "rules" : [
        "matches" : [
            {
                <match criteria>
            }
         ]
         "behaviors" : [
             {
                 <behaviors>
             }
         ]
    ]
}
```

Example rule

This policy contains a single rule set, that denies access to all .jpg, .gif, and .png files requested from any IP address other than 198.18.48.211.

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg gif png"
                }
            ],
            "behaviors" : [
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.211"
                }
            ]
        }
    ]
}
```

You can negate match conditions

All matches can be negated (inverting from “must match” to “must *not* match”) by adding a `"negated": true` attribute to the match condition.

The `"negated":` element is available with each match condition. It is optional, and its default setting is `false`. So, you don't need to include `“negated”: false` in your matches.

For example, if you want to apply behaviors to all paths that are not in either `/path1/*` or `/path2/*` then you could use the following negated match condition:

```
"matches" : [
    {
        "name" : "url-path",
        "value" : "path1 path2",
          "negated" : true
     }
]
```

Rule application and precedence

Rules are applied from the top down. The more precise the match criteria, the more important it is to ensure that subsequent rules do not alter the behavior in unexpected ways. Therefore, the more specific the criteria, the farther down in the list of rules it should be placed.

For example, let's assume you have a caching match/behavior combination that exists at the top of your rules listing that does the following:

- **Match criteria**: It matches on a "`/static/*`" URL wildcard *and* on URL extensions (for example "`png gif jpg`)
- **Behavior**: A caching TTL is set for this content of one day (`1d`).

If you were to add a separate caching match/behavior combination *farther down* in the JSON, that uses the same "`/static/*`" URL wildcard pattern, and set a caching TTL of one hour (`1h`), then the first combination would never be applied, because this second combination is a more generic set of match conditions, and would take precedence.

Nest matches

One level of match nesting is supported. (You can include two match criteria in a single rule.)

Example of nesting match criteria

With this example code segment, if the URL begins with `/path1` or `/path2`, and the request method is `GET`, then the `"behaviors" : [ ]` that are set in the policy are applied.

```
"matches" : [
     {
          "name" : "url-path",
          "value" : "path1 path2"
     },
     {
          "name" : "http-method",
          "value" : "GET"
     }
],
"behaviors" : [ {any behaviors you apply in the policy} ]
```

Include multiple behaviors

Along with your match criteria, you need to include behaviors that should be applied to requests. You can include multiple behaviors in a single delivery policy.

A behavior needs to be enabled for use in a delivery policy

A behavior's associated option in the Subcustomer Enablement behavior must be set to "**On**" in the base configuration associated with the applicable subcustomer.

There is a maximum number of behaviors

Currently the system supports a maximum of 100 behaviors in a single delivery policy—either within a single rule, or as the grand total for multiple rules. If the delivery policy exceeds this maximum, it is rejected upon submission.

An example of multiple behaviors

This is a simple example in which multiple rules are defined in a single rule. With it, the following apply:

- If a `GET` request comes from the IP address, `198.18.48.211` for content in the directories, `path1` or `path2`, it is denied access.
- The content in the directories, `path1` or `path2` is refreshed if it was cached by the Edge server *before* the epoch time `1533081600` (Wed, 01 Aug 2018 00:00:00 GMT).

```
"matches" : [
    {
        "name" : "url-path",
        "value" : "path1 path2"
    },
    {
        "name" : "http-method",
        "value" : "GET"
    }
],
"behaviors" : [
    {
        "name" : "ip-blacklist",
        "value" : "198.18.48.211"
    },
    {
        "name" : "content-refresh",
        "type" : "epoch",
        "value" : "1533081600"
    }
]
```

Supported matches and behaviors

Several match criteria ("matches") and behaviors are supported for use in a policy rule.

Matches

Click the "Name" entry for a match to access the ACE API documentation to review requirements and access a schema example.

Requirements and schema example	Description
client-ip	Include this to match using the IP address assigned to the requesting client. You can specify individual IP addresses, or CIDR blocks (that express a range of addresses).
cookie	Include this match to define specific cookie names for use when matching on an incoming request.

Requirements and schema example	Description
geography	Use this match to test the requesting client's location, either by continent, country, region, or designated market area (DMA). Each subcustomer policy can include up to ten geography matches.
header	Associated behaviors are applied if a header or header value you specify in this match criteria are included with a request.
host-name	Include this to match on hostnames listed in the incoming request's Host header.
http-method	Include this to match on a set of HTTP methods.
url-extension	Include this to match on the extension in the incoming request. This match criteria has no effect on URL paths that do not include a file extension.
url-filename	Include this to match on the extension in the incoming request. This match criteria has no effect on URL paths that do not include a file extension.
url-path	Include this to match on the first path component in the incoming request. The first path component is the section directly after the base URL.
url-querystring	Include this to match on the protocol or scheme (HTTP or HTTPS) of an incoming request.
url-scheme	Include this to match on the protocol or scheme (HTTP or HTTPS) of an incoming request.
url-wildcard	Include this to use wildcards when matching on the incoming request path, minus any query strings. This match type only supports the `*` wildcard.

Behaviors

Click the "Name" entry for a behavior to access the ACE API documentation to review requirements and usage, and access a schema example.

Name	Description
access-control	Include this to deny client requests based on the selected match conditions. ⓘ **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Access Control set to "On" in the *Subcustomer Enablement* behavior.
cachekey-query-args	Include this to specify how to handle query-string arguments in incoming requests. ⓘ **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Cache Key Query Arguments set to "On" in the *Subcustomer Enablement* behavior.
caching	Include this to provide time-to-live (TTL) cache settings for subcustomers.

Name	Description
	Note: To set this in a policy, the base configuration a subcustomer is assigned to must have Caching set to "On" in the *Subcustomer Enablement* behavior.
content-char-dynamic-web	If you're using Integrated Cloud Acceleration, this uses SureRoute to optimize the forward path to the origin server. It controls embedded object prefetching, and situational image compression. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Dynamic Web Content set to "On" in the *Subcustomer Enablement* behavior. (By default, this is set to "Off.")
content-char-large-file	Include this to optimize the delivery of large file downloads of up to 1.8 GB. This behavior uses partial object caching with pre-fetched object data. As a best practice, only use this behavior if you serve large files. Otherwise, the Akamai platform may send additional requests to your origin. When using Large File Optimization, if an object doesn't meet the minimum size criterion of 10 MB, the platform requests the entire object from the origin. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Large File Delivery set to "On" in the *Subcustomer Enablement* behavior. (By default, this is set to "Off.")
content-characteristics-on-demand-streaming	Include this to optimize cache and network timeout conditions for on-demand video content. The Akamai platform examines the URI file extension and path for the media format then automatically optimizes: cache efficiency, time-to-live, automated failover, downstream `Content-Type` headers, and network timeout settings. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Streaming Video On-demand Delivery set to "On" in the *Subcustomer Enablement* behavior. (By default, this is set to "Off.")
content-characteristics-live-streaming	Include this to optimize caching and network timeout conditions for live video content. The Akamai platform examines the URI file extension and path for the media format. It then automatically optimizes cache efficiency, time-to-live, automated failover, downstream `Content-Type` headers, and network timeout settings. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Streaming Video Live Delivery set to "On" in the *Subcustomer Enablement* behavior. (By default, this is set to "Off.")

Name	Description
content-compression	Include this in your policy to provide compression settings. You can enable gzip compression, decompress objects before delivering them to the client, or maintain the origin's compression settings. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Content Compressor set to "On" in the *Subcustomer Enablement* behavior.
content-refresh	Include this to invalidate CDN cache at an explicit date and time. This behavior uses epoch time to denote when a request should receive a new copy of the object or a revalidated one. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Content Refresh set to "On" in the *Subcustomer Enablement* behavior.
downstream-caching	Include this to control downstream caching of alternate content. Only use this behavior if site failover is enabled for the alternate hostname property. If you do not include this behavior, the subcustomer policy uses the downstream caching settings specified in the alternate hostname property. To enable site failover, use the *Subcustomer Enablement* behavior in Property Manager.
geo-blacklist	nclude this to *block* access to content based on the continent, country, region/state, or designated marketing area (DMA) of the requesting IP address. All other geographic areas are allowed. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Geo Allow/Block set to "On" in the *Subcustomer Enablement* behavior.
geo-whitelist	Include this to *allow* access to content based on the continent, country, region/state, or designated marketing area (DMA) of the requesting IP address. All other geographic areas are denied. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Geo Allow/Block set to "On" in the *Subcustomer Enablement* behavior.
ip-blacklist	Include this to *block* access based on the requesting IP address. All specified IP addresses are blocked. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have IP Allow/Block set to "On" in the *Subcustomer Enablement* behavior.
ip-whitelist	Include this to *allow* access based on the requesting IP address. Only the IP addresses listed are allowed access.

Name	Description
	ⓘ **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have IP Allow/Block set to "On" in the *Subcustomer Enablement* behavior.
modify-outgoing-request-header	Include this to modify the outgoing request headers sent from Akamai to an origin. This also works on request headers sent from a client if the request is sent back to the origin, but not a cache hit.
modify-outgoing-request-path	Include this to provide options for altering the request URL before it is sent to origin. ⓘ **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Modify Forward Path set to "On" in the *Subcustomer Enablement* behavior.
modify-outgoing-response-header	Include this to modify the outgoing response headers sent from the Edge server back to the client.
origin	Inlcode this to provide origin settings for the specific subcustomer. You need to include the origin DNS hostname, forward `host` header, and cache key. Optional settings include the origin base path and ports. ⓘ **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Origin set to "On" in the *Subcustomer Enablement* behavior.
origin-characteristics	Include this if you have Integrated Cloud Acceleration (ICA), to select the type of origin supporting your ACE implementation. Use the `origin` behavior to configure origin settings for subcustomers at the policy level.
origin-failover	This feature identifies primary origin connection failures based on a type you specify and marks that origin as “bad” after connections to all its IPs fail repeatedly. Rather than issuing a redirect to the end user, requests are failed over to a backup origin you call out. This improves response times, because the end user doesn’t have to wait several seconds for a connect-timeout on the forward request. Additionally, you specify a duration of time the primary origin is marked as bad. During this time, all requests are failed over to your backup origin. This relieves pressure on the primary by reducing the number of connection attempts, at a time when it appears to be having difficulties.
referer-blacklist	Include this to *block* access based on the `Referer` request header. This behavior helps verify that the client is a browser that supports *RFC 2616*, section 14.36, and that the referring HTML page is served from a domain trusted by the content owner. ⓘ **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Referrer Allow/Block set to "On" in the *Subcustomer Enablement* behavior.

Name	Description
referer-whitelist	Include this to *allow* access based on the `Referer` request header. This behavior helps verify that the client is a browser that supports *RFC 2616*, section 14.36, and that the referring HTML page is served from a domain trusted by the content owner. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Referrer Allow/Block set to "On" in the *Subcustomer Enablement* behavior.
site-failover	Include this to define the alternate hostname and path to use when the Edge server can't contact the origin server. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Site Failover set to "On" in the *Subcustomer Enablement* behavior.
token-auth	Include this to use tokens to control access to content. You can choose to transmit the token in a cookie, header, or query parameter. **Note:** To set this in a policy, the base configuration a subcustomer is assigned to must have Token Authentication set to "On" in the *Subcustomer Enablement* behavior.
url-redirect	Include this behavior to configure redirect responses for specific client requests, and stop them from contacting the origin.

Examples of match and behavior combinations

Here, you'll find examples of the use of rules in a policy to define matches that must be met, and the behaviors that will be applied. Additionally, we offer examples of conflicting and overlapping behaviors.

One rule, two matches, one behavior

In this example, we show a working base configuration that incorporates a single rule that has been set up with two match criteria that can be met to apply a single behavior. The following are set:

- **Match Criteria**: The URI extension must be “jpg,” “gjf,” or “png” and the request method must be GET.
- **Behavior**: If the match criteria is met, an IP whitelist is applied, denying all IPs except 198.18.48.211 and 198.18.48.212 access to these objects using the GET method.

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg gif png"
                },
                {
                    "name" : "http-method",
                    "value" : "GET"
                }
            ],
            "behaviors" : [
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.211 198.18.48.212"
                }
            ]
        }
    ]
}
```

Two rules, two matches, two behaviors

In this example we shows two rules, each with two match conditions, that when met will apply two behaviors.

In this first rule, the following apply:

- **Match Criteria**: The URI extension must be “jpg,” “gjf,” or “png” and the request method must be GET.
- **Behaviors**: Both an IP whitelist and a referrer blacklist are defined. Only requests from 198.18.48.211 and 198.18.48.212 are allowed, unless they also have a Referer header value containing “abcd.com.”

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg gif png"
                },
                {
                    "name" : "http-method",
                    "value" : "GET"
                }
            ],
            "behaviors" : [
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.211 198.18.48.212"
                },
                {
                    "name" : "referer-blacklist",
                    "value" : "*abcd.com*"
                }
            ]
        },
        {
            "matches" : [
                {
                    "name" : "host-name",
                    "value" : "www.example.com"
                },
                {
                    "name" : "http-method",
                    "value" : "GET"
                }
            ],
            "behaviors" : [
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.215 198.18.48.216"
                },
                {
                    "name" : "referer-blacklist",
                    "value" : "*www.abc.com*"
                }
            ]
        }
    ]
}
```

In the second rule, the following apply: matches all GET requests for the hostname “www.example.com” and applies both an IP whitelist and a referrer blacklist; only requests from 198.18.48.215 and 198.18.48.216 are allowed, unless they also have a Referer header value containing “www.abc.com.”

- **Match criteria**: All GET requests for the hostname, “www.example.com.”
- **Behaviors**: Both an IP whitelist and a referrer blacklist are applied. Only requests from 198.18.48.215 and 198.18.48.216 are allowed, unless they also have a Referer header value containing “www.abc.com.”

```
{
    "rules" :
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg gif png"
                },
                {
                    "name" : "http-method",
                    "value" : "GET"
                }
            ],
            "behaviors" : [
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.211 198.18.48.212"
                },
                {
                    "name" : "referer-blacklist",
                    "value" : "*abcd.com*"
                }
            ]
        },
        {
            "matches" : [
                {
                    "name" : "host-name",
                    "value" : "www.example.com"
                },
                {
                    "name" : "http-method",
                    "value" : "GET"
                }
            ],
            "behaviors" : [
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.215 198.18.48.216"
                },
                {
                    "name" : "referer-blacklist",
                    "value" : "*www.abc.com*"
                }
            ]
        }
    ]
}
```

A conflict that results in an undefined behavior

In this example, we show how setting up conflicting behaviors can result in an "undefined behavior," and offer a suggestion to avoid this.

In this incorrect rule, the following apply:

- **Match Criteria**: The URI extension must be “jpg,” “gjf,” or “png” and the request method must be GET.
- **Behaviors**: Both an IP whitelist, including a single IP, and an IP blacklist also including a single IP are defined as behaviors. The IP whitelist takes precedence, and all IPs *except* 198.18.48.212 are denied access to these objects. Since 198.18.48.214 is outside the white list, it is already denied. So, the IP blacklist is unnecessary, making it an "undefined behavior."

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg gif png"
                },
                {
                    "name" : "http-method",
                    "value" : "GET"
                }
            ],
            "behaviors" : [
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.211 198.18.48.212"
                },
                {
                    "name" : "ip-blacklist",
                    "value" : "198.18.48.213 198.18.48.214"
                }
            ]
        }
    ]
}
```

What to do to fix this

A more practical use case might be to implement the following: match on URI extension “jpg,” “gjf” or “png,” request method GET and apply a geographic restriction, but where an IP whitelist allows otherwise blocked clients to access the content. The following example will deny all clients inside the United States, except IPs 198.18.48.211 and 198.18.48.212 have been explicitly granted access to these objects.

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg gif png"
                },
                {
                    "name" : "http-method",
                    "value" : "GET"
                }
            ],
            "behaviors" : [
                {
                    "name" : "geo-blacklist",
                    "type" : "country",
                    "value" : "US"
                },
                {
                    "name" : "ip-whitelist",
                    "value" : "198.18.48.211 198.18.48.212"
                }
            ]
        }
    ]
}
```

Be careful with behaviors that overlap

Rule order is important because of the possibility of creating overlapping rules. Rule order is most important when the behaviors overlap.

As the partner, you are responsible for managing rules to prevent unexpected behavior. Essentially, this means that you need to do both of the following:

- Keep track of the match conditions within a given “rules” collection.
- Remember that, the more restrictive a set of match conditions is, the later it should appear in the collection.

For example, if there is a generic match for a “jpg” file extension that defines a time to live (TTL) of seven days, and another match that includes both a path of “`/images/*`” and a file extension match for “jpg” that defines a TTL of one day, *the latter rule should exist after the first in the configuration*. Putting the less complex rule later in the list would override all previous matches and the related behaviors. In this example, if the order were reversed the Akamai server would use a seven day TTL, regardless of whether or not the “jpg” object was requested using the “`/images/*`” path.

Below is an example of a properly-constructed rules collection for the above scenario:

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg"
                }
            ],
            "behaviors" : [
                {
                    "name" : "caching",
                    "type" : "fixed"
                    "string" : "7d"
                }
            ]
        },
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg"
                },
                {
                    "name" : "url-wildcard",
                    "value" : "/images/*"
                }
            ],
            "behaviors" : [
                {
                    "name" : "caching",
                    "type" : "fixed",
                    "string" : "1d"
                }
            ]
        }
    ]
}
```

ⓘ **Note:** If there is no overlap in behavior, then the order is irrelevant.

Consider the case where the "`/images/*`" path and "jpg" file extension matches are used to define a TTL, but another rule has a match for "jpg," that applies a geographic blacklist behavior for, say, North Korea. These two rules do not share behaviors. So, the less restrictive rule is safe to place later in the list.

Below is an example in which the rule order does not matter, because there is no overlap in behavior:

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg"
                }
            ],
            "behaviors" : [
                {
                    "name" : "geo-blacklist",
                    "type" : "country",
                    "value" : "KP"
                }
            ]
        },
        {
            "matches" : [
                {
                    "name" : "url-extension",
                    "value" : "jpg"
                },
                {
                    "name" : "url-wildcard",
                    "value" : "/images/*"
                }
            ],
            "behaviors" : [
                {
                    "name" : "caching",
                    "type" : "fixed"
                    "string" : "1d"
                }
            ]
        }
    ]
}
```

As the partner, you also need to consider the case when multiple behaviors are applied with overlapping *matches*. When this is the case, it is generally safe to put the most restrictive matches *last* in the rules collection, regardless of whether there are overlapping behaviors.

View a specific policy

You can GET a specific subcustomer's associated policy to view its rules and behaviors. This can help if you need to validate policy settings or to prepare updates to an existing policy.

This is accomplished by running the "**Get a policy**" operation via the ACE API. You'll need the following values to run this operation:

- `propertyId`: This is the unique ID value associated with the base configuration that houses the subcustomers you want to list. Contact your Account Representative for this value.
- `domainName`: This is the digital property (domain) for the subcustomer associated with the target policy.
- `network`: You need to decide which network version of the policy you want to view: production (live) or staging (testing).

Call formatting and other specifics are covered in the ACE *API documentation*.

Delete a policy

If you no longer need a specific policy file, you can delete it.

This is accomplished by running the "**Delete a policy**" operation via the ACE API. You'll need the following values to run this operation:

- `propertyId`: This is the unique ID value associated with the base configuration that houses the subcustomers you want to list. Contact your Account Representative for this value.
- `domainName`: This is the digital property (domain) associated with the subcustomer for the target policy.
- `network`: You need to decide which network version of the policy you want to delete: production (live) or staging (testing).

Note: Take caution when deleting a policy in the production network. Ensure that it is not currently serving a live subcustomer.

Call formatting and other specifics are covered in the ACE *API documentation*.

Review a policy's history

The API offers an operation that allows you to view change information for an existing policy.

This is accomplished by running the "**Get policy history**" operation via the ACE API. You need the following values to run this operation:

- `propertyId`: This is the unique ID value associated with the base configuration that houses the subcustomers you want to list. Contact your Account Representative for this value.
- `domainName`: This is hostname (domain) associated with the subcustomer for the target policy.
- `network`: You need to decide which network version of the policy you want to view change information: production (live) or staging (testing).

Call formatting and other specifics are covered in the ACE *API documentation*.

Map expansion and ACE API operations

We've enhanced the ACE API in support of our recent map expansion improvements.

What is Map Expansion?

Previously with ACE, the regional network maps used were limited to address safety concerns, to prevent risk to the entire network. So, ACE wasn't operating as well as it could. To improve performance, we've made large improvements that allowed us to expand to 100% of the regional maps in the Akamai network.

As a result, propagation of a new policy requires more time. Early testing has shown this has increased from one minute to eight minutes. But, the overall performance and safety benefits definitely make up for this.

There's a propagation status for each policy

To accommodate the added propagation time that map expansion introduces, we've made changes to the ACE API that allow you to check on your policies.

- **You can check the status of a policy**. The *Get a Policy* operation now includes the `activationStatus` value in the response:
  - `ACTIVE`. The policy has been applied and is active in the applicable network (staging vs. production).
  - `PROPAGATING`. The policy was just created or edited and is still being propagated to the applicable network.
  - `ACTIVATION_FAILED`. Propagation of the policy failed. Check its configuration and resubmit it.
- **You can view the last active version of a policy**. Use the *Get latest active policy* operation to view the last `ACTIVE` status version of a policy. You can review all of the current settings applied for it, to determine if an update is required. (And, skip the new propagation time delay if the update isn't necessary.)

A simple test of a policy

When a request is made against a subcustomer hostname that has policy rules set up, the rules take effect when the requested content is delivered to the end-user.

Let's assume your subcustomer's hostname is `www.example.com`, and an end-user request is issued for the following URL:

```
http://www.example.com/us-content/uploads/2014/10/file1.png
```

You've setup a policy for `www.example.com`, that includes the following rules:

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-wildcard",
                    "value" : "/us-content/*",
                }
            ],
            "behaviors" : [
                {
                    "name" : "geo-whitelist",
                    "type" : "country",
                    "value" : "US"
                }
            ]
        }
    ]
}
```

With `"value" : "US"` included, requests from outside the ‘United States’ are denied. (A 403 response is issued, and the content is not served.)

To test this, you could issue a request from a non US-based client. However, using standard command line tools and adding the Akamai debug “Pragma” values can help identify the reason for the 403 response. (Contact your Account Representative for information on debug "Pragma" values.)

Other recommended tasks

This includes some highly recommended tasks such as setting up test objects, activating log delivery and alerts, scheduling reports, and other tasks.

These tasks deal with monitoring and maintaining applications. They are recommended, but you don't have to complete them to implement your ACE property. We recommend you review these tasks and determine which ones best suit your needs.

Important: Log delivery must be activated prior to delivering any content from the Edge network, as logs cannot be collected retroactively. It takes 24-48 hours for the log delivery to start from the time when you activate it.

You should activate log delivery

Edge network logs can help you troubleshoot content and configuration problems. You can receive Edge server logs of all requests for your content in a standard log file format. These logs are aggregated from data from all Edge servers, sorted by time, and can be delivered with specified frequency to a specific email address, or sent to your FTP servers.

Log delivery is not enabled by default. If you need Edge network logs, you need to activate it in Akamai Control Center.

1. Log in to Control Center using a **User ID** and **Password** that have been configured for access to ACE.
2. Select **Configure** > **Log Delivery**.
3. Locate the applicable **Object ID** (CP code) associated with your instance of ACE. (It is listed in the **Products** column.)
4. Click the **Action** (gear icon) drop-down and select **Begin Log Delivery** > **New**.
5. From the **Delivery** tab, set all required options (marked with an asterisk).
6. Click the **Next** button.
7. Set contact information for the new log configuration and click **Finish**. Logs will be delivered based on your defined settings.

Tip: Detailed information on the Log Delivery tool can be accessed via its dedicated Online Help (the "?" button revealed in the UI in Control Center).

NetStorage

If your Edge network services include having logs delivered via FTP to your NetStorage account, specify the NetStorage FTP upload account in the Log Delivery settings. The NetStorage account information is included in your service activation package. You can also look it up in Control Center.

NetStorage	
	⚠ **Caution:** When you use NetStorage for log delivery, it is important to manage the content. If you don't remove excess log files, you may see large overage charges as the files take up more and more space. You can remove the log files after you download them, or you can set up automatic purging rules through the NetStorage interface on Control Center.

You can test log delivery

Edge network logs are delivered in standard log formats, such as W3C Extended log format. However, if you are currently using your own server logs for reporting or data analysis, you should verify that Edge network logs are fully compatible with your log processing application. Test the first log files you receive from the Edge network through your regular log processing application or workflow, and adjust them if necessary.

You can schedule recurring reports

In addition to viewing online reports in Control Center, you can activate recurring reports that are automatically sent to you.

1. Log in to Control Center using a **User ID** and **Password** that have been configured for access to ACE.
2. Select **Configure** > **Alerts**.
3. Select **Cloud Embed** and click the **Continue** button.
4. Click the **All Configured Alerts** link.
5. From the **Choose Alert Type** interface, you can select from a host of HTTP Downloads-related alerts-Click the **Add** link for the desired one.
6. Input a desired **Alert Name**.
7. Select the applicable **CP Code(s)** for which the alert should be issued.
8. Scroll down and set alert parameters as necessary.
9. Define **Notification** options to indicate who will receive the alert, and how it will be sent.
10. Click the **Save** button.

You should create additional Administrator accounts

An administrator account for Control Center is provided to you in your account information after you initially provision ACE. As an administrator, you can create additional accounts with this, or lesser levels of access.

Creating additional administrator-level accounts allows multiple people to log on and perform administrator-level tasks.

1. Log in to Control Center using a **User ID** and **Password** that have been configured for access to ACE.
2. Select **Configure** > **Organization** > **Manage Users and Groups**.

Tip: Click the “?” button in the upper right of the UI for assistance with this tool.

Go live

The Akamai platform requires that the subcustomer resource be CNAMEd *to* Akamai. So, we recommend that each subcustomer use a test domain to validate end-to-end CDN functionality via Akamai.

Let's assume the goal is to enable `www.example.com` via Akamai.

1. Create a test hostname—such as test-www.example.com
2. CNAME the test hostname to the same endpoint (that you provide as the cloud partner) for `www.example.com`
3. Create a copy of the policy via the ACE API, so that it uses the same origin. If required, the policy can override the `Host:` header sent to the origin to use the "fixed" value, `www.example.com` as shown below:

Note: All other behaviors, such as time to live (TTL) settings should match what is set for `www.example.com` policy that was copied.

```
{
    "rules" : [
        {
            "matches" : [
                {
                    "name" : "url-wildcard",
                    "value" : "/*",
                }
            ],
            "behaviors" : [
                {
                    "name" : "origin",
                    "value" : "-",
                    "params" : {
                        "digitalProperty" : "test-www.example.com",
                        "originDomain" : "c1234567.cloudprovider.com",
                        "cacheKeyType" : "origin",
                        "cacheKeyValue" : "-"
                        "hostHeaderType" : "fixed",
                        "hostHeaderValue" : "www.example.com",
                    }
                }
            ]
        }
    ]
}
```

In the short term, this means fully-qualified URLs in the resulting HTML are not updated to reflect the test hostname, so browser-based testing will not suffice. Automated testing can be used to functionally test delivery of content via the Akamai platform.

Add dynamic web content support for a subcustomer

To configure support for dynamic web content, you need to include the `content-characteristics` behavior in the JSON body when creating or updating a subcustomer's delivery policy.

What is dynamic web content?

This describes web content (such as an individual web page) that displays different each time it's viewed. For example, the content may change with the time of day, the user that accesses it, or the type of user interaction.

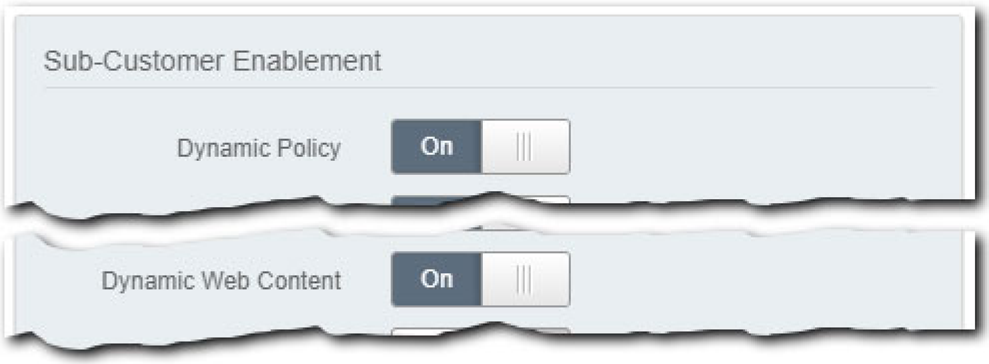
Prerequisites for dynamic web content

Before you update or create a policy to support dynamic web content for a subcustomer, there are various tasks you need to complete.

You need to enable Dynamic Web Content in your base configuration

Just like any instance of ACE, you need to create a base configuration to define feature and support permissions for use with subcustomers. To support dynamic web content, you need to apply some specific settings, using the **Subcustomer Enablement** behavior in the base configuration.

Note: Only the options required to enable support for dynamic web content are discussed here. Other options can be enabled (or disabled) as necessary for your base configuration.



Sub-Customer Enablement

Dynamic Policy: On

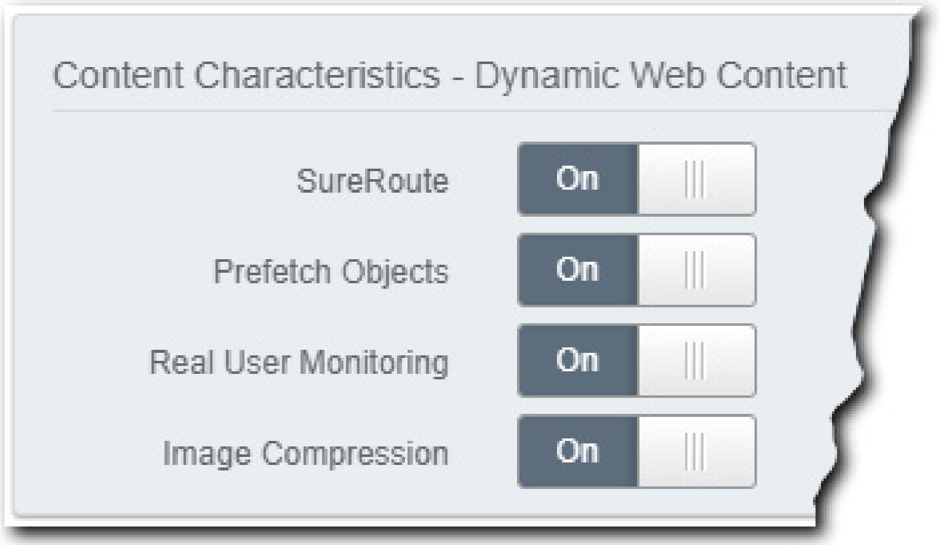
Dynamic Web Content: On

- **Dynamic Policy**: This switch needs to be set to **On**. This allows interaction with policies for subcustomers that have been registered with this base configuration. (This must be enabled to modify or create a policy to support dynamic web content.)
- **Dynamic Web Content**: This switch must be set to **On** to allow configuration of dynamic web content support in a subcustomer policy.

You can add Integrated Cloud Acceleration

Integrated Cloud Acceleration (ICA) is a collection of features that you can use to add more support for Dynamic Web Content for subcustomers.

- **You need ICA added to your contract**. Talk to your account representative about adding this to your contract to use this support.
- **You need "Dynamic Web Content" enabled**. This option needs to be enabled in the **Sub-Customer Enablement** behavior.
- **You need to add the "Content Characteristics - Dynamic Web Content" behavior**. With **Dynamic Web Content** enabled, you can add this behavior to the *same rule*. In that rule, perform the following:
    1. Click the **Add Behavior** button.
    2. In the **Search available behaviors** field, input "Content" and select the **Content Characteristics - Dynamic Web Content** behavior from the list.
    3. Click the **Insert Behavior** button.
    4. Enable any of the options in the behavior to allow their use when configuring a subcustomer policy via the ACE API. (Mouse-over each option for a detailed description.)


Content Characteristics - Dynamic Web Content

SureRoute On

Prefetch Objects On

Real User Monitoring On

Image Compression On

You need to register subcustomers

To offer a subcustomer access to this support, it must be registered with the base configuration that has the appropriate **Subcustomer Enablement** behavior options applied.

- **This is a new subcustomer**. This is no different than registering a subcustomer with any other ACE base configuration—you use the "**Add a new subcustomer**" operation via the ACE API.
- **This subcustomer is already registered**: You don't need to re-register an existing subcustomer to apply this support.

1. Ensure that the base configuration that the subcustomer is associated with has the appropriate **Subcustomer Enablement** options applied.
2. Update the existing delivery policy settings for that subcustomer to apply the settings covered in this document.

Set up dynamic web content

You need to run the "Create or update a policy" operation via the ACE API, and set the `type` element value to `dynamic-web-content` in the `content-characteristics` behavior.

Create or update a policy

General instructions on this operation can be found in the *Akamai Cloud Embed API v2* documentation. See *Create or update a policy*.

Define the appropriate match criteria in the delivery policy

Like all other rules in a delivery policy, you must set a match criteria (`matches`) that must be met to apply the associated `behaviors`. In the example presented here, the `url-wildcard` match is used. This match compares the incoming request path (excluding query string) to what is defined as the `value` for the match "`/*`" is set here, which indicates all incoming request paths.

Include the "content-characteristics" behavior and set required members

To set up this support, you need to include the `content-characteristics` behavior, and the following members must be included:

- `type`: This must be set to `dynamic-web-content`.
- `value`: This must be included and always set to a hyphen (`-`).

Optional parameters

You have access to additional parameters ("`params`") with the `content-characteristics` behavior, when configuring support for dynamic web content:

Note: To configure these parameters in a subcustomer's policy, that subcustomer must be registered with a base configuration that has had the parameters enabled. They are enabled via the **Content Characteristics - Dynamic Web Content** behavior. (This behavior must be added to the base configuration, and the parameters must be set to "On.") If the behavior is not added, or a parameter is set to "Off," they cannot be included.

Name	Type	Description
`sureRouteTestObjectPath`	String (optional)	Include this to enable AkamaiSureRoute for the policy. The associated value must be a valid URI path that points to an object on your origin that meets the standard criteria for a valid SureRoute Test Object. (It can't have auth, and it must be compressible—between 4KB –12KB in size after compression.)

Name	Type	Description
`prefetchEnabled`	Boolean	Include this and set the value to "Yes" to enable the embedded object pre-fetching feature
`mobileImageCompressionEnabled`	Boolean	Include this and set the value to "Yes" to enable compression of JPEG images for requests over certain mobile network conditions.

What is SureRoute?

The Akamai SureRoute feature provides the optimal route between an Edge server and the origin server, at any given point in time. If the `sureRouteTestObjectPath` parameter is left out of a subcustomer policy, SureRoute is *disabled*.

What is Prefetching?

Prefetching positions objects at the Edge to ready them for requests, to reduce the overall time to deliver these objects. This behavior relies on the "origin-assisted" prefetch model—it expects your origin to send details on the objects that need to be prefetched, in its response. When `prefetchEnabled` is set to "true," Edge servers "prefetch" objects that have the following file extensions:

.aif	.gcf	.ppc
.aiff	.gff	.pws
.au	.gif	.swa
.avi	.grv	.swf
.bin	.hdml	.txt
.bmp	.hqx	.vbs
.cab	.ico	.w32
.carb	.ini	.wav
.cct	.jpeg	.wbmp
.cdf	.jpg	.wml
.class	.js	.wmlc
.css	.mov	.wmls
.doc	.mp3	.wmlsc
.dcr	.nc	.xsd
.dtd	.pct	.zip
.exe	.pdf	
.flv	.png	

What is Mobile Image Compression?

When `mobileImageCompressionEnabled` is set to "true," Akamai Edge servers increase the compression of JPEG images (.jpg, .jpeg, .jpe, .jif, .jfif, and .jfi extensions), when slower network speeds are detected for a requesting mobile client. Serving compressed images reduces the amount of data required to load a page, which can compensate for suboptimal network performance. You don't need to change any your HTML content when using or configuring mobile image compression.

When this compression is triggered, the Akamai network does the following:

1. **The application segment metadata is stripped**. This does not affect how the image is displayed or decoded.

2. **The JPEG Comment (COM) segment is preserved**. This segment may contain copyright information (which may need to be preserved).

JSON Examples

- **Example 1**: This example shows how an endpoint is optimized for dynamic web content.

```
{
    "rules":[
        {
            "matches": [
                {
                    "name": "url-wildcard",
                    "value": "/*"
                }
            ],
            "behaviors": [
                {
                    "name": "origin",
                    "value": "-",
                    "params": {
                        "cacheKeyValue": "-",
                        "digitalProperty": 
"sitecustomer.partnerdomain.net",
                        "cacheKeyType": "origin",
                        "hostHeaderValue": "-",
                        "originDomain": 
"dynamicwebcustomer.origin.mediaservices.partnerdomain.net",
                        "hostHeaderType": "origin"
                    },
                },
                {
                    "name": "content-characteristics",
                    "type": "dynamic-web-content",
                    "value": "-",
                    "params": {
                        "sureRouteTestObjectPath": "/akamai/akamai-
sureroute-test-object.htm",
                        "realUserMonitoring": true,
                        "prefetchEnabled": true,
                        "mobileImageCompressionEnabled": true
                    }
                }
            ]
        }
    ]
}
```

- **Example 2**: In this example the endpoint is optimized for dynamic web content, but SureRoute is *disabled*. (The `sureRouteTestObjectPath` member is left out.)

```
{
    "rules":[
        {
            "matches": [
                {
                    "name": "url-wildcard",
                    "value": "/*"
                }
            ],
            "behaviors": [
                {
                    "name": "origin",
                    "value": "-",
                    "params": {
                        "cacheKeyValue": "-",
                        "digitalProperty":
"sitecustomer.partnerdomain.net",
                        "cacheKeyType": "origin",
                        "hostHeaderValue": "-",
                        "originDomain":
"dynamicwebcustomer.origin.mediaservices.partnerdomain.net",
                        "hostHeaderType": "origin"
                    },
                },
                {
                  "name": "content-characteristics",
                    "type": "dynamic-web-content",
                    "value": "-",
                    "params": {
                        "realUserMonitoring": true,
                        "prefetchEnabled": true,
                        "mobileImageCompressionEnabled": true
                    }
                }
            ]
        }
    ]
}
```

Add live video support for a subcustomer

Live video delivery support for ACE offers several optimizations for live content delivery for subcustomers.

You don't need to enable it in your base configuration

Live video support is enabled here by default. You don't need to update your base configuration to enable it or to apply any additional settings.

But, you need to enable it in a subcustomer's policy

You need to run the "Create or update a policy" operation via the ACE API, and set the `type` element value to `streaming-video-live` in the `content-characteristics` behavior in a subcustomer's policy.

Set up live video in a policy

Here, we cover the specifics required to update a subcustomer policy to add support for live video.

Create or update a policy

General instructions on this operation can be found in the *Akamai Cloud Embed API v2* documentation. See *Create or update a policy*.

Define the appropriate match criteria in the delivery policy

Like all other rules in a delivery policy, you must set a match criteria (`matches`) that must be met to apply the associated `behaviors`. In the example presented here, the `url-wildcard` match is used. This match compares the incoming request path (excluding query string) to what is defined as the `value` for the match "`/*`" is set here, which indicates all incoming request paths.

Include the "content-characteristics" behavior in the policy

To set up this support, you need to include the `content-characteristics` behavior, and the following members must be included:

- `type`: This must be set to `streaming-video-live`.
- `value`: This must be included and always be set to a hyphen (`-`).

With just these members set, live video is enabled for the subcustomer, using default settings. Additional specifics on the use of this behavior in a policy can be found in *The "content-characteristics-live-streaming" behavior* on page 149.

You can set optional parameters

You have access to additional parameters ("`params`") with the `content-characteristics` behavior, when configuring support for live streaming video. They allow you to define specific settings for this subcustomer.

If you apply a `segmentDuration<media format>` parameter, what you set overrides the Akamai default duration values for this specific subcustomer.

Name	Type	Description	Default
segmentDurationHLS	Number	This is the duration, in seconds for HLS media segments. The supported range is 0.1 to 10.0. See *Apple Technical Note 2224* for their recommended best practices.	10.0 sec.
segmentDurationHDS	Number	This is the duration in seconds for HDS media segments. The supported range is 0.1 to 10.0.	6.0 sec.
segmentDurationDASH	Number	This is the duration in seconds for DASH media segments. The supported range is 0.1 to 10.0.	6.0 sec.
segmentDurationSmooth	Number	This is the duration in seconds for Smooth media segments. The supported range is 0.1 to 10.0.	2.0 sec.
overrideDefaultManifestTTL	Boolean	By default, Akamai sets the time to live (TTL) for the manifest file based on the segment size specified for that format. However, you can also set a TTL using the Caching behavior in your base configuration. You can include this parameter to specifically define which TTL to use. • `true`: The TTL set in the Caching behavior is used. • `false`: Akamai defines the TTL based on the segment size.	false
prefetch	Object	Use this object to enable and configure origin-assisted segment prefetching for live video for this subcustomer.	fals

Name	Type	Description	Default
		Complete details and requirements are covered in *Add prefetching support for video* on page 91. **Note:** HDS format video is currently not supported for use with prefetching.	e*

** Individual members are included in this object to define the segment prefetch method to use. Each member is boolean and defaults to “false.”*

The Akamai platform identifies the media format by examining the URI file extension or the structure of the URI path, and automatically optimizes various settings relating to cache efficiency, cache time-to-live, automated failover/retry, downstream “Content-Type” headers, and network timeout conditions for on-demand video content.

JSON Examples

- **Example 1**: This example omits all of the optional parameters. (The `params` object is included, but left empty.) Default segment/fragment durations for all supported media formats will be used.

```
{
    "rules":[
        {
            "matches": [
                {
                    "name": "url-wildcard",
                    "value": "/*"
                }
            ],
            "behaviors": [
                {
                    "name": "content-characteristics",
                    "type": "streaming-video-live",
                    "value": "-",
                    "params": {}
                }
            ]
        }
    ]
}
```

- **Example 2**: This example shows how to include live video streaming. It also shows how to set custom segment/fragment durations for a subcustomer, and also enable prefetching.

```
{
    "rules":[
        {
            "matches": [
                {
                    "name": "url-wildcard",
                    "value": "/*"
                }
            ],
            "behaviors": [
                {
                    "name": "content-characteristics",
                    "type": "streaming-video-live",
                    "value": "-",
                    "params": {
                        "segmentDurationHLS": 6,
                        "segmentDurationHDS": 5.5,
                        "segmentDurationDASH": 2.5,
                        "segmentDurationSmooth": 2.5,
                        "overrideDefaultManifestTTL": true,
                        "prefetch": {
                            "originAssist": true
                        }
                    }
                }
            ]
        }
    ]
}
```

Add on demand video support for a subcustomer

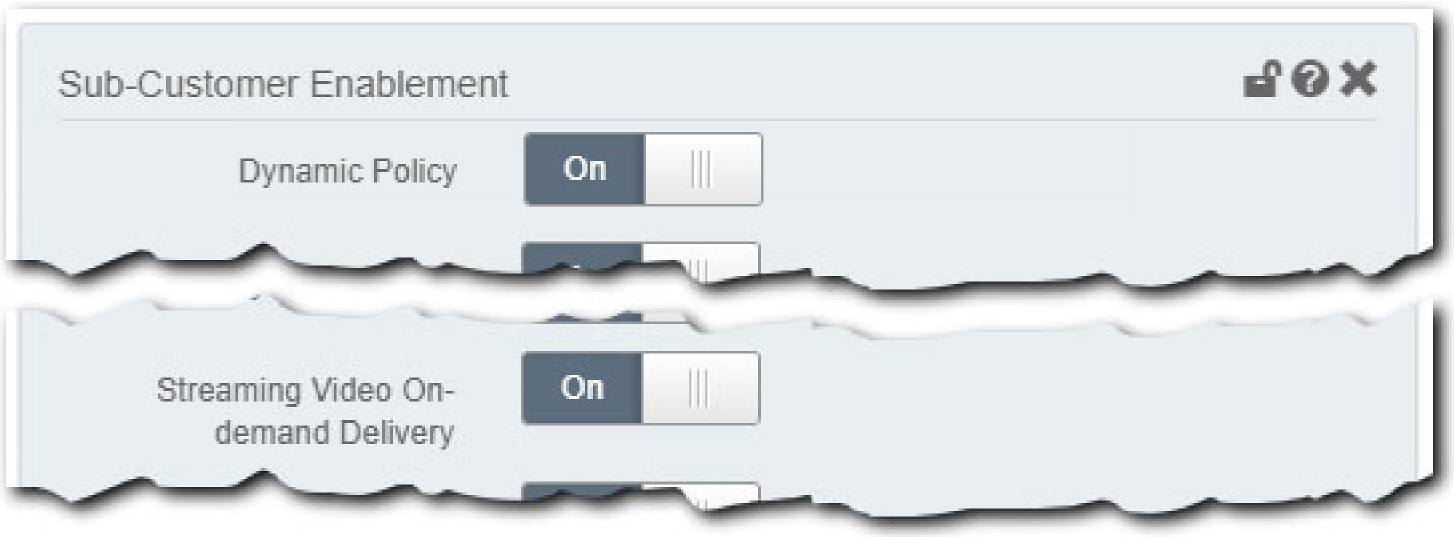
To configure subcustomer support for streaming on demand video, you need to start by configuring a specific behavior in your base configuration. Then, you need to use the ACE API to configure the `content-characteristics` behavior in the JSON request body when creating or updating a subcustomer's policy.

Enable on demand video in your base config.

To start, two specific options must be enabled in your ACE base configuration, and you need to perform some additional tasks.

Apply settings in the Subcustomer Enablement behavior

Just like any instance of ACE, you need a base configuration to define feature and support permissions for all subcustomers. To support on demand streaming video, enable these option in the Subcustomer Enablement behavior.


Sub-Customer Enablement

Dynamic Policy: On

Streaming Video On-demand Delivery: On

Note: Only the options required to enable on demand streaming video support are discussed here. Other options in the Subcustomer Enablement behavior can be enabled (or disabled) as necessary for your base configuration. See *Set up a base configuration* on page 11 for complete details.

- **Dynamic Policy**: This switch needs to be set to **On**. This allows interaction with policies for subcustomers that have been registered with this base configuration. (This must be enabled to modify or create a delivery policy to support streaming video.)
- **Streaming Video On-demand Delivery**: This switch must be set to "**On**" to allow configuration of on demand streaming video in a subcustomer delivery policy. (This is "**On**" by default.)

There are other optional settings

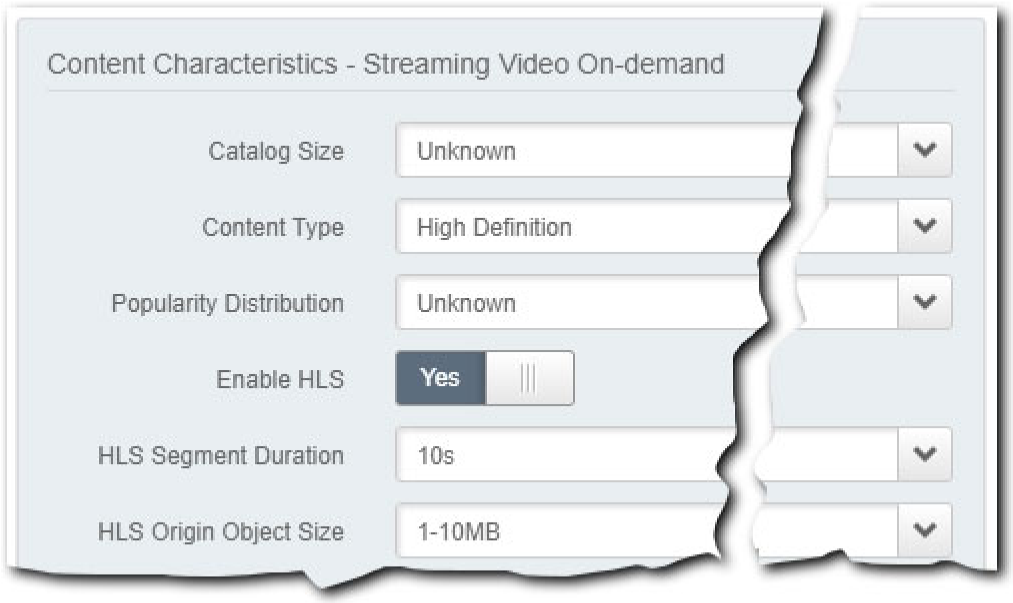
You can add a separate behavior to configure various optimizations for on demand video delivery. These settings are applied by default to all subcustomers that are registered with this base configuration. (You can override these settings on a per-subcustomer basis, by applying the appropriate settings in a delivery policy via the ACE API.)

- **Add the "Content Characteristics - Streaming Video On-demand" behavior**. You need to add this behavior to the *same rule* that contains the **Subcustomer Enablement** behavior:

  1. Click **Add Behavior**.
  2. In the Search available behaviors field, input "**Content**" and select the **Content Characteristics - Streaming Video On-demand** behavior from the list.
  3. Click **Insert Behavior**.
  4. Set the options in the behavior to define various optimizations for streaming on-demand video delivery. (Mouse-over each option for a detailed description.)

Note: Consider the following when setting options here:

- If you want to offer support for a specific media format (HLS, HDS, DASH, or Smooth) for your subcustomers, its Enable <Media format> slider must be set to "**Yes**." These all default to "**Yes**." So, if you left this behavior out of your base configuration, all of these formats default to enabled.
- If you're unsure of what to set for a specific option, leave it set to "**Unknown**."



Content Characteristics - Streaming Video On-demand

Catalog Size	Unknown
Content Type	High Definition
Popularity Distribution	Unknown
Enable HLS	Yes
HLS Segment Duration	10s
HLS Origin Object Size	1-10MB

You need to register subcustomers

To offer a subcustomer access to this support, it must be registered with the base configuration that has the appropriate **Subcustomer Enablement** behavior options applied.

- **This is a new subcustomer**. This is no different than registering a subcustomer with any other ACE base configuration—you use the "**Add a new subcustomer**" operation via the ACE API.
- **This subcustomer is already registered**: You don't need to re-register an existing subcustomer to apply this support.
    1. Ensure that the base configuration that the subcustomer is associated with has the appropriate **Subcustomer Enablement** options applied.
    2. Update the existing delivery policy settings for that subcustomer to apply the settings covered in this document.

Set up on demand video

You need to run the "Create or update a policy" operation via the ACE API, and set the `type` element value to `streaming-video-on-demand` in the `content-characteristics` behavior in a subcustomer's delivery policy.

Create or update a policy

General instructions on this operation can be found in the *Akamai Cloud Embed API v2* documentation. See *Create or update a policy*.

Define the appropriate match criteria in the delivery policy

Like all other rules in a delivery policy, you must set a match criteria (`matches`) that must be met to apply the associated `behaviors`. In the example presented here, the `url-wildcard` match is used. This match compares the incoming request path (excluding query string) to what is defined as the `value` for the match "`/*`" is set here, which indicates all incoming request paths.

Include the "content-characteristics" behavior in the delivery policy

To set up this support, you need to include the `content-characteristics` behavior, and the following members must be included:

- `type`: This must be set to `streaming-video-on-demand`.
- `value`: This must be included and always be set to a hyphen (`-`).

With just these members set, on demand video is enabled for the subcustomer, using default settings.

You can set optional parameters

You have access to additional parameters ("`params`") with the `content-characteristics` behavior, when configuring support for on demand streaming video. Each allows you to define the segment duration for various supported media formats:

Note: To configure these parameters in a subcustomer's delivery policy, that subcustomer must be registered with a base configuration that has had the parameters enabled. They are enabled via the **Content Characteristics - Streaming Video On-demand** behavior. (This behavior must be added to the base configuration, and the parameters must be set to "**On**.") If the behavior is not added, or a parameter is set to "**Off**," they cannot be included.

Name	Type	Description *
`segmentDurationHLS`	Float (optional)	This is the duration, in seconds for HLS media segments. The supported range is 0.1 to 10.0. The Akamai default value is 10.0 seconds. See *Apple Technical Note 2224* for their recommended best practices.
`segmentDurationHDS`	Float (optional)	This is the duration in seconds for HDS media segments. The supported range is 0.1 to 10.0. The Akamai default value is 6.0 seconds.
`segmentDurationDASH`	Float (optional)	This is the duration in seconds for DASH media segments. The supported range is 0.1 to 10.0. The Akamai default value is 6.0 seconds.
`segmentDurationSmooth`	Float (optional)	This is the duration in seconds for Smooth media segments. The supported range is 0.1 to 10.0. The Akamai default value is 2.0 seconds.
`prefetch`	Object	Use this object to enable and configure origin-assisted segment prefetching for on demand video for this subcustomer. This is disabled (set to "`false`") by default. Complete details and requirements are covered in *Add prefetching support for video* on page 91. **Note:** HDS format video is currently not supported for use with prefetching.

** The default durations here only apply if you did not apply a custom value via the Content Characteristics - Streaming Video On Demand behavior in the associated base configuration.*

The Akamai platform identifies the media format by examining the URI file extension or the structure of the URI path, and automatically optimizes various settings relating to cache efficiency, cache time-to-live, automated failover/retry, downstream “Content-Type” headers, and network timeout conditions for on-demand video content.

JSON Examples

- **Example 1**: This example shows how an endpoint is optimized for on demand video streaming.

```
{
    "rules":[
        {
            "matches": [
                {
                    "name": "url-wildcard",
                    "value": "/*"
                }
            ],
            "behaviors": [
                {
                    "name": "origin",
                    "value": "-",
                    "params": {
                        "cacheKeyValue": "-",
                        "digitalProperty":
"videocustomer.partnerdomain.net",
                        "cacheKeyType": "origin",
                        "hostHeaderValue": "-",
                        "originDomain":
"videocustomer.origin.mediaservices.partnerdomain.net",
                        "hostHeaderType": "origin"
                    },
                },
                {
                    "name": "content-characteristics",
                    "type": "streaming-video-on-demand",
                    "value": "-",
                    "params": {
                        "segmentDurationHLS": 10,
                        "segmentDurationHDS": 5.5,
                        "segmentDurationDASH": 3,
                        "segmentDurationSmooth": 4.5,
                        "prefetch": {
                            "originAssist": true
                        }
                    }
                }
            ]
        }
    ]
}
```

- **Example 2**: You can also omit some or all of the segment duration `params`, as well as the additional parameters. If you do, the applicable default duration setting is applied for each media format:

  – **I included the Content Characteristics - Streaming Video On Demand behavior in my base configuration**. If you've set a different value for segment duration for a specific media format, *that value is used*.

  – **I didn't include the Content Characteristics - Streaming Video On Demand behavior in my base configuration**. The default durations discussed in the table above are applied for each media format.

  The snippet below shows the `content-characteristics` behavior with an empty `params` object to use default durations.

```
{
    "name": "content-characteristics",
    "type": "streaming-video-on-demand",
    "value": "-",
    "params":{}
}
```

Add prefetching support for video

Prefetching positions target media content at the Edge in anticipation of requests by end users. This reduces the time to deliver that content.

Prefetching extracts data from the current request or response and determines the next object that should be requested. (For example, when a specific segment in a media clip is requested by a player, you can prefetch the next segment to the Edge to expedite its delivery.)

A high level flow of Prefetching

1. The media player makes a request for `{object-1}`.

2. Akamai handles the request by forwarding it to your origin.

3. You origin returns the requested object—`{object-1}`—back to Akamai.

    - The origin response also includes headers to prefetch `{object-2}`. The value of this header is either an absolute URL path, or a relative URL path (relative to `{object-1}`).

4. Akamai returns `{object-1}` to player and this completes the request/response flow for `{object-1}`.

5. Since the response from your origin also included prefetch response headers for `{object-2}`, Akamai triggers prefetching.

    - Akamai creates an HTTP/S request for `{object-2}` and forwards it to your origin. This is called a "prefetch request." Akamai includes a preconfigured request header in it, that tells your origin that it's a prefetch request.

6. Your origin returns `{object-2}` to Akamai.

    - *The origin response could include a response header to prefetch `{object-3}` to continue the prefetching process.*

7. Akamai caches `{object-2}`.

    - *Akamai doesn't trigger prefetching for `{object-3}`*. This is because `{object-2}` was already prefetched from your origin (and you don't want a never-ending recursive prefetch cycle).

8. The player makes a request for `{object-2}`.

9. Akamai fetches `{object-2}` from it's cache and returns it in the response to the player.

    - If the origin response for `{object-2}` included prefetch response headers for `{object-3}`, Akamai triggers prefetching of `{object-3}`, and the process can continue.

What media formats are supported?

Streaming Formats	• Apple HTTP Live Streaming (HLS)

	• Dynamic Adaptive Streaming over HTTP (DASH) • Microsoft Smooth Streaming (MSS)
Live or Video on Demand (VoD)	Both

Add prefetching for a subcustomer

You add this support to a subcustomer's policy via the ACE API v2. It's incorporated within the `content-characteristics` behavior for either the `streaming-video-on-demand` or `streaming-video-live` types:

```
{
    "rules":[
        {
            "matches": [
                {
                    "name": "url-wildcard",
                    "value": "/*"
                }
            ],
            "behaviors": [
                {
                    "name": "content-characteristics",
                    "type": "streaming-video-live",
                    "value": "-",
                    "params": {
                        "prefetch": {
                            "originAssist": true
                        }
                    }
                }
            ]
        }
    ]
}
```

See the following pages for complete details on how to incorporate the `content-characteristics` behavior, for your selected delivery method:

- **Live**: *Set up live video in a policy* on page 82
- **On demand**: *Set up on demand video* on page 88

What can be prefetched

An adaptive bit rate stream, such as HLS and DASH consists of multiple object types that are requested by a player in a specific order. You can use this behavior to prefetch these objects to speed up delivery to end users.

What's the typical flow for an HLS stream?

The table that follows discusses the object types you can usually find in an HLS stream, and the order they're presented in a request.

Object	What is triggered next in a Prefetch	Additional details
Master Playlist (.m3u8)	One or more variant Playlists (.m3u8).	N/A
Variant Playlist (.m3u8)	One or more segments belonging to the current variant playlist. The segment to prefetch depends on the play position of the player, and this can be difficult to determine for a VoD stream. For Live streams, this is *typically* one of the last three segments in the variant playlist.	For the typical HLS stream, segments have the following extensions: • .ts for video, video + audio • .aac, .ac3, .ec3 for audio-only • .vtt, .webvtt, .mp4, etc. for subtitles For the newer HLS streams with CMAF segments, the extensions are: • .mp4, .m4v, etc. for video • .mp4, .m4a, etc. for audio • .vtt, .webvtt, .mp4 for subtitles These are not definitive, complete lists. There are a lot of supported extensions, and some are non-standard. This is to support various one-off workflows you may have in place.
Variant Playlist (.m3u8)	Zero or one "`init`" segment (described via the `#EXT-X-MAP` tag), that belongs to the current variant playlist.	The `init` segments are present *almost* exclusively for CMAF segments, and the typical extensions are:

Object	What is triggered next in a Prefetch	Additional details
		• .mp4 • .m4v • .m4a
Segment	The segment that follows next in the video/audio/subtitle presentation time.	N/A

What's the typical flow for a DASH stream?

The table that follows discusses the object types you can usually find in a DASH stream, and the order they're presented in a request.

Object	What is triggered next in a Prefetch	Additional details
Manifest File (.mpd)	`init` segments for audio and video followed by actual audio and video segments. There are no bit rate specific playlists and content is always demuxed. So, right after fetching the MPD file, the player typically requests two segments: one for video and another for audio.	N/A
Init segment	One or more content segments.	N/A
Segment	The segment that follows next in the video/audio/subtitle presentation time.	N/A

Use the origin-assist scheme

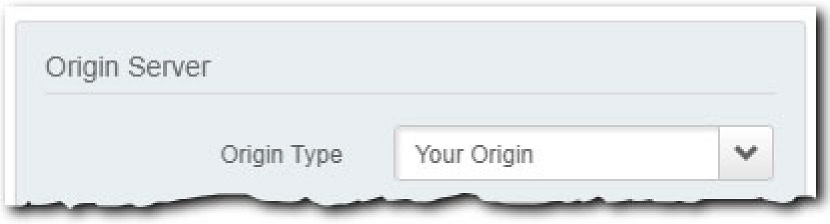
With this prefetching scheme, when Akamai fetches an object from an origin, the response includes a new header that lists the next object in the sequence. Akamai can read this information and prefetch this object.

Basically, Akamai relies on assistance from your "intelligent" origin to trigger prefetching.

What Origin Type can I use?

When defining your origin server for your base configuration, only certain origin types can be used with prefetching.

When setting up your base configuration in Property Manager, you need to set the **Origin Type** in the *Origin Server* behavior to either of the following:



- **Your Origin**: To use your own custom origin.
- **Media Services Live**: If you're using Media Services Live as your origin.

Note: Currently, you can't use NetStorage as your origin if you want to use prefetching.

How an origin triggers prefetching

Akamai sends specific information to your origin to initiate prefetching. Your origin must include a properly formatted response header after this request, to trigger prefetching.

Phase 1: Akamai tells your origin Prefetching is enabled

Akamai handles a player request for content and sends a *prefetch request header* to your origin, telling it that Prefetching is enabled:

```
CDN-Origin-Assist-Prefetch-Enabled: 1
```

Phase 2: Your origin responds to Akamai with what to prefetch

Your origin needs to use the Akamai request header to trigger prefetching. Your origin must send a *prefetch response header* that tells Akamai what should be prefetched. This response header must be comprised of the absolute or relative path to the object to be prefetched, followed by the `Content-Length` header stating the length of the complete prefetch response header.

```
CDN-Origin-Assist-Prefetch-Path: <absolute|relative path of prefetch-able
object's URL>
Content-Length: x
```

Akamai creates the full URL for the object to be prefetched by using:

- The player-requested URL of the current object, and
- The relative or absolute path listed in the prefetch response header.

Absolute path examples

Specify an absolute path by *prefacing it with forward slash*. This tells Akamai to use the exact path specified.

URL of player that initiated the request	Value of the response header from the origin (used to trigger prefetch)	URL of a request prefetched from Akamai
`https://property-hostname/some/1234/video-100k/pl.m3u8`	`/hls/live/1234/video-100k/seg1.ts`	`https://property-hostname-from-player-request/hls/live/1234/video-100k/seg1.ts`
`https://property-hostname/thing/1234/video-100k/seg1.ts`	`/hls/live/1234/video-100k/seg2.ts`	`https://property-hostname-from-player-request/hls/live/1234/video-100k/seg2.ts`

Note: In these examples, both `property-hostname` and `property-hostname-from-player` are the HTTP/1.1 "Host" header as seen by Akamai.

Relative path examples

Specify a relative path by *leaving out the forward slash from the beginning of the path*. Akamai uses the forward path of the current request, minus the filename as the base path when creating the prefetch URL path.

URL of player that initiated the request	Value of the response header from the origin (used to trigger prefetch)	URL of a request prefetched from Akamai
`https://property-hostname/some/1234/video-100k/pl.m3u8`	`video-100k/seg1.ts`	`https://property-hostname/some/1234/video-100k/seg1.ts`
`https://property-hostname/thing/1234/video-100k/seg1.ts`	`seg2.ts`	`https://property-hostname/some/1234/video-100k/seg2.ts`

Note: In these examples, `property-hostname` is the HTTP/1.1 "Host" header as seen by Akamai.

Example 1: Include a single "prefetchable" path per response header

The origin uses individual instances of the prefetch response header to include a single object to be prefetched.

```
CDN-Origin-Assist-Prefetch-Path: /hls/live-streaming/fifa/france-croatia/
video-1000k/pl.m3u8
Content-Length: x
```

If multiple objects are to be prefetched, then one response header per prefetchable URL can be returned by origin. Ensure that all response headers use the same name. The order of these headers determines the order Akamai follows to request each object.

```
CDN-Origin-Assist-Prefetch-Path: /hls/live-streaming/fifa/france-croatia/
video-1000k/pl.m3u8
```

```
CDN-Origin-Assist-Prefetch-Path: /hls/live-streaming/fifa/france-croatia/
audio/pl.m3u8
Content-Length: x
```

Note: Middle layer proxies may not preserve this ordering of response headers, and content may not be delivered in the desired order.

Example 2: Include multiple "prefetchable" paths in a single response header

The origin uses a single instance of the prefetch response header to include a comma-separated list of multiple objects to be prefetched.

```
CDN-Origin-Assist-Prefetch-Path: /hls/live-streaming/fifa/france-croatia/
video-1000k/pl.m3u8,/hls/live-streaming/fifa/france-croatia/audio/pl.m3u8
Content-Length: x
```

The order of the paths listed determines what is prefetched first. (Since this is only a single response header, the issue with middle layer proxies doesn't apply.)

Phase 3: Akamai requests the prefetched object for the player

With a "prefetch object" determined by the response header from your origin, Akamai requests it, as if it was requested by the player. To allow your origin to differentiate a prefetch request from a regular request all prefetch requests from Akamai use the same request header:

```
CDN-Origin-Assist-Prefetch-Request: 1
```

Once Akamai receives the prefetch object, it caches it until the player requires it.

Along with sending the prefetch object to Akamai, you can also have your origin send a new response header to prefetch the next object in the queue, if applicable.

Add large file delivery support for a subcustomer

To configure support for large file delivery, you need to include the `content-characteristics` behavior in the JSON body when creating or updating a subcustomer's delivery policy.

What constitutes a large file?

A large file is anything in excess of 10 MB, up to a maximum of 1.8 GB in size.

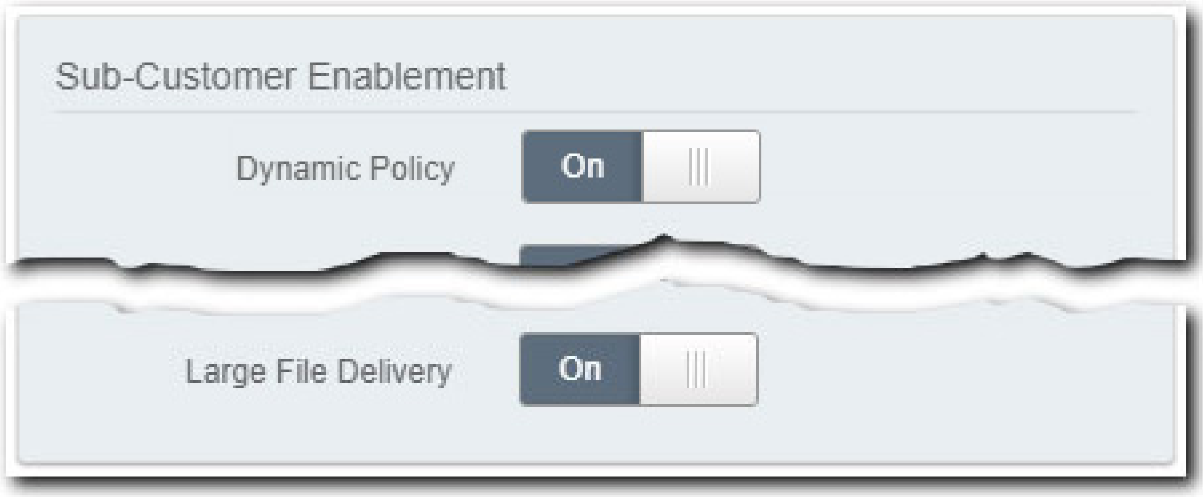
Prerequisites

Before you update or create a delivery policy to support large file delivery for a Subcustomer, there are various tasks you need to complete.

You need to enable Large File Delivery in your base configuration

Just like any instance of ACE, you need to create a base configuration to define feature and support permissions for subcustomers. To support large file delivery, you need to apply some specific settings, using the **Subcustomer Enablement** behavior in the base configuration.

Note: Only the options required to enable large file delivery are discussed here. Other options can be enabled (or disabled) as necessary for your base configuration.



Sub-Customer Enablement

Dynamic Policy: On

Large File Delivery: On

- **Dynamic Policy**: This switch needs to be set to **On**. This allows interaction with policies for subcustomers that have been registered with this base configuration. (This must be enabled to modify or create a delivery policy to support large file delivery.)

- **Large File Delivery**: This switch must be set to **On** to allow configuration of large file delivery in a subcustomer delivery policy.

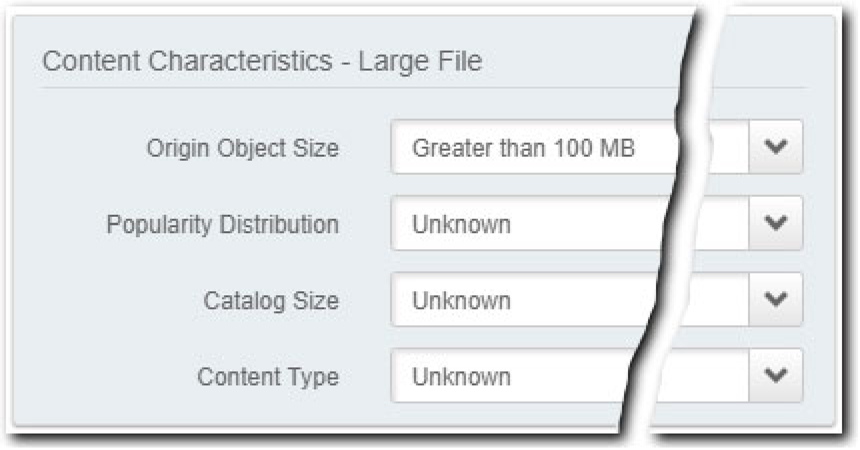
You can add other Large File optimizations

You can add a separate behavior that allows you to set various optimizations for use in large file delivery. These settings will be applied to all subcustomers that are registered with this base configuration. (You can override these settings on a per-subcustomer, by applying the appropriate "params" in a delivery policy via the ACE API.)

- **Add the "Content Characteristics - Large File" behavior**. Make sure that **Large File Delivery** is enabled in the Subcustomer Enablement behavior, and add this behavior to the *same rule*:

    1. Click the **Add Behavior** button.
    2. In the **Search available behaviors** field, input "Content" and select the **Content Characteristics - Large File** behavior from the list.
    3. Click the **Insert Behavior** button.
    4. Set the options in the behavior to define various optimizations for large file delivery. (Mouse-over each option for a detailed description.)

Note: If you're unsure of what to set for a specific option, leave it set to "Unknown."



Content Characteristics - Large File

Origin Object Size: Greater than 100 MB

Popularity Distribution: Unknown

Catalog Size: Unknown

Content Type: Unknown

You need to register subcustomers

To offer a subcustomer access to this support, it must be registered with the base configuration that has the appropriate **Subcustomer Enablement** behavior options applied.

- **This is a new subcustomer**. This is no different than registering a subcustomer with any other ACE base configuration—you use the "**Add a new subcustomer**" operation via the ACE API.

- **This subcustomer is already registered**: You don't need to re-register an existing subcustomer to apply this support.

    1. Ensure that the base configuration that the subcustomer is associated with has the appropriate **Subcustomer Enablement** options applied.
    2. Update the existing delivery policy settings for that subcustomer to apply the settings covered in this document.

Support for large file delivery

You need to run the "Create or update a policy" operation via the ACE API, and set the `type` element value to `large-files` in the `content-characteristics` behavior.

Create or update a policy

General instructions on this operation can be found in the *Akamai Cloud Embed API v2* documentation. See *Create or update a policy*.

Define the appropriate match criteria in the delivery policy

Like all other rules in a delivery policy, you must set a match criteria (`matches`) that must be met to apply the associated `behaviors`. In the example presented here, the `url-wildcard` match is used. This match compares the incoming request path (excluding query string) to what is defined as the `value` for the match "`/*`" is set here, which indicates all incoming request paths.

Include the "content-characteristics" behavior and set required members

To set up this support, you need to include the `content-characteristics` behavior, and the following members must be included:

- `type`: This must be set to `large-files`.
- `value`: This must be included and always set to a hyphen (`-`).

Optional parameters

A single parameter ("`params`") is available for this use case.

Name	Type	Description
`objectSize`	String (optional)	Apply any of the following values: • `lt1mb`: Specifiy this for files less than 1 MB in size. (This is *not* considered a "large file" use case.) • `1mbto10mb`: Specify this for files from 1 MB to 10 MB in size. (This is *not* considered a "large file" use case.) • `10mbto100mb`: (Default) Specify this for files from 10 MB to 100 MB in size. (This is considered a "large file" use case.) • `gt100mb`: Specify this for files that are 100 MB in size and larger. (This is considered a "large file" use case.)

Note: If you have added the **Content Characteristics - Large File** behavior to thesubcustomer's base configuration, and set an **Origin Object Size**, what you set via the `objectSize` parameter in a policy will override that setting for this specific subcustomer.

You should know subcustomer file size requirements

The `params` settings, `10mbto100mb` and `gt100mb` are considered "large file" settings. The other settings represent much smaller object sizes, and they actually prevent serving large objects, because the “partial object caching” (POC) feature is *disabled* with these cases.

POC increases origin server offload by caching in chunks, instead of as a single, composite object. POC is required for the delivery of "large files"—those in excess of 10 MB in size. (It is automatically enabled if you set `10mbto100mb` or `gt100mb` in a policy.) If you set too small a file size, POC will not be enabled for these larger files, and 403 errors will be returned for requests.

Also, if you specify a "large file" setting, but your origin never actually serves files in that size range, the Akamai platform makes additional requests to the origin. (This can impact overall performance and access.) This happens because the Edge server requests the first byte of each object to determine if the minimum object size criteria (at least 10MB) has been met. Once that check fails, the server simply re-requests the entire object from the origin.

We recommend that you verify the size of your delivery objects on your subcustomer's origin, and select the appropriate size to ensure that POC is enabled or disabled, as necessary.

JSON Examples

- **Example 1**: This example shows how an endpoint is optimized for large files.

```
{
    "rules":[
        {
            "matches": [
                {
                    "name": "url-wildcard",
                    "value": "/*"
                }
            ],
            "behaviors": [
                {
                    "name": "origin",
                    "value": "-",
                    "params": {
                        "cacheKeyValue": "-",
                        "digitalProperty":
"downloadcustomer.partnerdomain.net",
                        "cacheKeyType": "origin",
                        "hostHeaderValue": "-",
                        "originDomain":
"downloadcustomer.origin.mediaservices.partnerdomain.net",
                        "hostHeaderType": "origin"
                    },
                },
                {
                    "name": "content-characteristics",
                    "type": "large-files",
                    "value": "-",
                    "params": {
                        "objectSize": "gt100mb",
                    }
                }
            ]
        }
    ]
}
```

- **Example 2**: The snippet below shows the `content-characteristics` behavior with an empty `params` object. In this case, the `10mbto100mb` is automatically used as the default. (Objects will be fetched in byte-range specific chunks, using the 10 MB to 100 MB size setting.)

  ```
  {
      "name": "content-characteristics",
      "type": "large-files",
      "value": "-",
      "params":{}
  }
  ```

Upgrade a request from HTTP to HTTPS

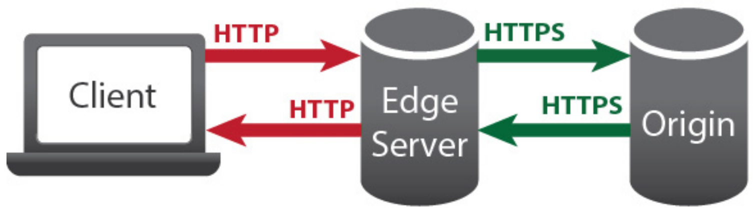
Add the HTTP to HTTPS Upgrade behavior to your property if you want to convert HTTP (non-secure) requests from your clients to use secure HTTPS between the Akamai edge and your origin server.

PS Upgrade

ior to upgrade an HTTP request received at the edge to HTTPS for the remainder of th
havior, you need to properly set up 'Origin SSL Certificate Verification' in the 'Origin Se
y. You also need to ensure that your origin supports HTTPS.

A complete request flow involves three total entities:

- The client making the request
- The Akamai Edge server, where your property is read, and target content may be cached
- The origin where the target content is actually hosted



With this behavior added, all requests in the flow *between the Akamai edge and your origin* are converted to HTTPS to secure them. Since TCP is stateful, an HTTP request from a client must be answered with an HTTP response. If you require a complete HTTPS connection end-to-end, consider implementing a redirect from the original HTTP URL to an HTTPS one.

How do I get access to HTTP to HTTPS Upgrade?

You need to have this added to your contract to access the appropriate behavior in Property Manager. Contact your Account Representative to add this functionality.

Add HTTP to HTTPS Upgrade

Once you have it added to your contract, you can add this behavior to your ACE property by performing the following:

1. Create a new ACE configuration, or edit an existing one using Property Manager.
2. In the Property Configuration Settings options, click **Add Behavior**.
3. In the *Search available behaviors* field, input **Add HTTP** to filter the listed behaviors, and select **Add HTTP to HTTPS Upgrade** from the list.

You also need to set up your origin to support HTTPS

You don't need to do anything to actually configure this behavior. Just adding it to your property enables the conversion. However, if you've selected "**Your Origin**" as your **Origin Server** in your property, you also need to:

- Configure the additional settings that are revealed, with *"Your origin" selected as your Origin Server*.
- Properly configure your origin for HTTPS transfer

Note: This does not apply if you're using NetStorage as your origin. Akamai sets origin security automatically for NetStorage when you add the HTTP to HTTPS Upgrade behavior.

Additional considerations

- The upgrade is to Standard TLS (HTTPS L1). To use Enhanced TLS if you're transferring personally identifiable information (PII), you need to create and provision an Enhanced TLS certificate, edit your property to set Security Options and define a Property hostname to Edge hostname association.
- This behavior uses 443 as the forward port for all products other than AMD, DD, and OD.

The Akamai Cloud Embed API v2

As an Akamai cloud partner, you can use the Akamai Cloud Embed (ACE) to provide Content Delivery Network (CDN) features to your cloud customers, known as "subcustomers." Use the ACE API v2 API to add customers to access your unique ACE CDN instance, and manage policies of rules and behaviors specific to each customer.

You can configure CDN features per domain and give subcustomers the ability to buy, configure, and monitor Akamai's CDN services directly through your portal. In most cases, once you create one or more base configurations to support your business needs, you can use this API to support hundreds of thousands of subcustomers per base configuration.

Before you begin

There are several prerequisites you need to meet. See *Before you begin with the API* on page 49 and ensure that you've addressed all of the points there.

API concepts

Familiarize yourself with some of the common terms used in this API:

- **Cloud partners.** Akamai resellers or partners who provide delivery services to their customers. You, as a user of this API, are a cloud partner, also referred to simply as "partner" throughout this documentation.
- **Subcustomers.** A cloud partner's customers are subcustomers. Akamai does not assign individual Content Provider (CP) codes to subcustomers even though their traffic is sent over the Akamai platform. As a Cloud Partner you have access to usage detail reports for each of your subcustomers, based on the subcustomer IDs you provide Akamai.
- **Subcustomer ID.** This is a unique ID for each subcustomer within your own billing systems. All traffic for a particular subcustomer is rolled up by this ID for billing purposes.
- **Base configuration.** An Akamai delivery configuration that includes all of the common rules for processing end-user requests. You use the Property Manager API or GUI application to configure properties used for both ACE and ICA.
- **Sub-Customer Enablement behavior.** In Property Manager, the *subCustomer behavior* controls which individual ACE and ICA features you can use to handle your subcustomers' traffic. You can set up this behavior to provide access to all available features, or you can select a subset of features to define different classes of service.
- **Content Characteristics-Dynamic Web Content behavior.** Include this behavior if you want to enable specific ICA optimizations for your subcustomers.
- **InstantConfig behavior.** If your subcustomers specifically send HTTP traffic, you have to add the *instantConfig* behavior to your property. This behavior, also known as Multi-Domain Configuration, lets you to associate multiple web assets to a single property without adding each hostname separately. It applies property settings to all incoming hostnames based on a DNS lookup.
- **Policy.** Policies determine how Akamai edge servers handle a given subcustomer's requests. A single policy is bound to a hostname. Your policy JSON is made up of rules, which contain both match criteria and behaviors. When an incoming request meets the match criteria in a rule, it triggers

the behaviors listed in that rule. Make rules unique within a policy: they can't have identical sets of matches. A policy can also contain up to 100 behaviors.

- **Policy rules.** Rules include both match criteria and behaviors. When an incoming request meets the match criteria in a rule, it triggers the behaviors listed in that rule. Within a rule you can use each match type and each behavior once. Also, no one rule can contain both a whitelist and blacklist behavior of a given type. For example, you can add an IP whitelist and a referrer blacklist to a rule, but you can't have both an IP whitelist and an IP blacklist in the same rule. ACE applies rules from top to bottom, so ensure you list them from least restrictive to most restrictive. For example, you would list a match on `url-wildcard` value `/*` first because it would apply to all requests, where `/images/*` would only apply to a subset of requests.

- **Policy matches.** Within a rule, a match defines which subcustomer requests receive the behaviors within the rule. If a match condition for a rule is met, ACE applies the behaviors set in a rule. When constructing the `matches` array in a rule, the type of data you enter depends on the type of match. For example, if you use the query string match, you have to enter the exact string and keep case sensitivity in mind. If you use the `url-scheme` match, you enter either `HTTP` or `HTTPS`. Within a match, you cannot repeat entries in the value string.

- **Policy behaviors.** Like Property Manager, the ACE API uses behaviors to encapsulate settings to customize a configuration. Behaviors are a part of a policy's rules. A rule can have many behaviors or only one. For the ACE API, you group rules into policies. You can have a maximum of 100 behaviors in a subcustomer policy. ACE rejects policies exceeding 100 behaviors upon submission. Within a behavior, you cannot repeat entries in the value string.

There are other APIs you can use

You can optionally use other Akamai APIs to perform other aspects of the configuration:

- *Property Manager API*. You can use this API to create and update your base configurations for ACE. This may be excessive, because you typically only need to interact with this configuration a single time. So we recommend you use the Property Manager in Control Center, instead.

- *Media Delivery Reports*. This API offers operations specific to ACE that let you retrieve usage and quality metrics, but it doesn't include billing data. Currently, it combines ACE and ICA data: If a given endpoint served both types of traffic during the selected time period, the report does not differentiate between the two. In addition, this API also reports metrics for Adaptive Media Delivery, Download Delivery, Object Delivery and RTMP Media Delivery.

- *Billing Center*. This API provides access to CSV-based contract usage data for accounts you can access. We provide separate reports for ACE and ICA per billing period. The ACE report lists subcustomer usage (hits and bytes) by geography. The ICA report contains billing data for subcustomers that used ICA, without reference to geography.

The ACE API workflow

Here's a basic workflow that shows how to create a subcustomer ID and policy.

1. If not completed already, *Set up a base configuration* on page 11 and ensure that you configure the **Sub-Customer Enablement** behavior to meet your needs. This configuration and behavior determine baseline settings for use. It serves as a "limiter." It states what settings can and can't be used in a policy for all subcustomers assigned to the configuration.

2. Complete the prerequisites in *Before you begin with the API* on page 49.

3. Run the *Add a new subcustomer* operation to apply and register a “subcustomer ID” for each of your customers.

4. Run the *Create or update a policy* operation for each subcustomer ID you need to add to your base configuration. Set up the policy with individual rules and behaviors that you want applied when ACE receives a request for that subcustomer’s content.

5. Test the end-to-end CDN functionality with each subcustomer. You should use a test domain to verify both the functionality and the configuration.

6. For the final step to go live, change applicable DNS to switch Cloud Partner websites to the Akamai platform.

Resources

This section provides details on the API’s various operations.

Operation	Method	Endpoint
Subcustomer		
List all subcustomers	GET	`/partner-api/v2/network/{network}/properties/{propertyId}/customers`
Get a subcustomer	GET	`/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`
Add a new subcustomer	PUT	`/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`
Remove a subcustomer	DELETE	`/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`
Property		
List all sub-properties for a base configuration	GET	`/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties`
List all subcustomer domains	GET	`/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}/sub-properties`
Policy		
Get a policy	GET	`/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`
Create or update a policy	PUT	`/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`
Delete a policy	DELETE	`/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`

Operation	Method	Endpoint
Get policy history	GET	`/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy/history`
Get latest active policy	GET	`/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy/active`

List all subcustomers

Returns a list of all subcustomers assigned to the selected base configuration (`propertyId`).

Request

GET `/partner-api/v2/network/{network}/properties/{propertyId}/customers`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/customers`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID after you create the configuration.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Make a GET request to `/partner-api/v2/network/{network}/properties/{propertyId}/customers`.

4. Review the list of subcustomers and their geographies returned in the response.

Response

Status 200 `application/json`

Object type: *SubCustomer*

Response body:

```
[
    {
        "customerID": "abc-123",
        "geo": "FR",
        "subPropertyIDs": [
            "www.example.com",
            "www.example.biz",
            "www.example.net",
            "www.example.co.uk"
        ]
    }
]
```

Get a subcustomer

Returns policy information for the selected subcustomer and activation network.

Request

GET `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/customers/SC12345`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID after you create the configuration.
`subcustomerId`	String	`SC12345`	The unique ID of the subcustomer. Your organization assigns this value.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).
2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.
3. Run the *List all subcustomers* operation and store the `customer` value, which corresponds to the `subcustomerId` URL parameter.
4. Make a GET request to `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`.

Response

Status 200 `application/json`

Object type: *SubCustomer*

Response body:

```
{
    "geo": "FR"
}
```

Add a new subcustomer

Add a new subcustomer ID to the selected base configuration (`propertyId`).

Request

PUT `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/customers/SC12345`

Content-Type: `application/json`

Request body:

```
{
    "geo": "FR"
}
```

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network,

Parameter	Type	Sample	Description
			or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID after you create the configuration.
`subcustomerId`	String	`SC12345`	The unique ID of the subcustomer. Your organization assigns this value.

Steps

1. Determine the `network` for the subcustomer. You can choose `staging` (testing) or `production` (live).
2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.
3. You need to provide a unique ID for the subcustomer (`subcustomerId`). As the cloud partner, it's up to you to determine a method to create these IDs and associate them with each subcustomer. The ID can contain up to 50 alphanumeric, dot or dash characters.
4. Make a PUT request to `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`. You also need to include the subcustomer's `geo` (geographic region) in a PUT request body. Use the valid two letter country designation (for example, `US` for United States or `BR` for Brazil).
5. Verify that the `geo` value returned in the response is accurate for the subcustomer you added.

Response

Status 200 `application/json`

Object type: *SubCustomer*

Response body:

```
{
    "geo": "FR"
}
```

Remove a subcustomer

Remove a specific subcustomer using its unique `subcustomerId`.

Request

DELETE `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/customers/SC12345`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID after you create the configuration.
`subcustomerId`	String	`SC12345`	The unique ID of the subcustomer. Your organization assigns this value.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Run the *List all subcustomer domains* operation and store the `customerID` value, that applies to the applicable subcustomer. Use this as the `subcustomerId` URL parameter for this operation.

4. Make a DELETE request to `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}`.

5. Review the `message` in the response to verify the delete was successful.

Response

Status 200 `application/json`

Response body:

```
{
    "description": "The subcustomer for property_id '251922' and
subcustomer_id '0004-propertyfolks' was successfully deleted.",
    "message": "Successfully deleted"
}
```

List all sub-properties for a base configuration

Returns all of the domains (`subPropertyID` values) for subcustomers assigned to a selected base configuration (`propertyId`) and activation network.

Request

GET `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/sub-properties`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID when the configuration is created.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Make a GET request to `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties`.

Response

Status 200 `application/json`

Object type: *SubProperty*

Response body:

```
[
    {
        "subPropertyID": "www.example.com",
        "customerID": "abc-123"
    },
    {
        "subPropertyID": "www.example.biz",
        "customerID": "abc-123"
    },
    {
        "subPropertyID": "www.example.com.net",
        "customerID": "def-456"
    },
    {
        "subPropertyID": "www.example.com.co.uk",
        "customerID": "def-456"
    }
]
```

List all subcustomer domains

Returns domain-specific information for the selected subcustomer domain (`subPropertyID`) and activation network.

Request

GET `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}/sub-properties`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/customers/SC12345/sub-properties`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID when the configuration is created.
`subcustomerId`	String	`SC12345`	The unique ID of the subcustomer. Your organization assigns this value.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Run the *List all subcustomers* operation and store the `customer` value, which corresponds to the `subcustomerId` URL parameter.

4. Make a GET request to `/partner-api/v2/network/{network}/properties/{propertyId}/customers/{subcustomerId}/sub-properties`.

Response

Status 200 `application/json`

Object type: *SubProperty*

Response body:

```
[
    {
        "customerID": "abc-123",
        "geo": "FR",
        "subPropertyIDs": [
            "www.example.com",
            "www.example.biz",
            "www.example.net",
            "www.example.co.uk"
        ]
    }
]
```

Get a policy

Returns the policy for the selected subcustomer domain, base configuration (`propertyId`), and activation network.

Request

GET `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/sub-properties/www.example.com/policy`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network,

Parameter	Type	Sample	Description
			or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID when the configuration is created.
`domainName`	String	`www.example.com`	The name of the subcustomer's domain.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Run the *List all sub-properties for a base configuration* operation and store the value from the `subPropertyIDs` array, that corresponds to the appropriate subcustomer's `customerID`. Use this as the `domainName` URL parameter for this operation.

4. Make a GET request to `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`.

5. Review the rule information returned in the response.

Response

Status 200 `application/json`

Object type: *Policy*

Response body:

```
{
    "rules": [
        {
            "matches": [
                {
                    "name": "http-method",
                    "value": "GET"
                }
            ],
            "behaviors": [
                {
                    "params": {
                        "originBasePath": "/",
                        "cacheKeyValue": "-",
                        "digitalProperty": "www.example.com",
                        "cacheKeyType": "origin",
                        "httpPort": 80,
                        "hostHeaderValue": "-",
                        "originDomain": "www.example.com",
                        "hostHeaderType": "origin"
                    },
                    "name": "origin",
                    "type": "-",
                    "value": "-"
                },
                {
                    "name": "content-refresh",
                    "type": "fixed",
                    "value": "1m"
                }
            ]
        }
    ],
    "activationStatus": "ACTIVE"
}
```

Create or update a policy

Create a new subcustomer policy or update an existing one.

Request

PUT `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/sub-properties/www.example.com/policy`

Headers:

```
X-Customer-ID: 100-example.com
X-Custom-Metadata: 100
```

Content-Type: `application/json`

Object type: *Policy*

Request body:

```
{
    "rules": [
        {
            "matches": [
                {
                    "name": "http-method",
                    "value": "GET"
                }
            ],
            "behaviors": [
                {
                    "params": {
                        "originBasePath": "/",
                        "cacheKeyValue": "-",
                        "digitalProperty": "www.example.com",
                        "cacheKeyType": "origin",
                        "httpPort": 80,
                        "hostHeaderValue": "-",
                        "originDomain": "www.example.com",
                        "hostHeaderType": "origin"
                    },
                    "name": "origin",
                    "type": "-",
                    "value": "-"
                },
                {
                    "name": "content-refresh",
                    "type": "fixed",
                    "value": "1m"
                }
            ]
        }
    ],
    "activationStatus": "ACTIVE"
}
```

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID when the configuration is created.
`domainName`	String	`www.example.com`	The name of the subcustomer's domain.

Steps

1. Determine the Akamai `network` for the policy. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Run the *List all sub-properties for a base configuration* operation and store the value from the `subPropertyIDs` array, that corresponds to the appropriate subcustomer's `customerID`. Use this as the `domainName` URL parameter for this operation.

4. Include a valid, registered subcustomer ID as the `X-Customer-ID` header. You can also optionally specify a string up to 50 characters in length to serve as the `X-Custom-Metadata` header, for use in billing and reporting.

5. Make a PUT request to `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`. Include a properly formatted request body with rules that consist of proper *match criteria and the behaviors* that you want applied when a request for the subcustomer's content meets that criteria.

6. Review the information returned in the response.

Response

Status 200 `application/json`

Object type: *Policy*

Response body:

```
{
    "rules": [
        {
            "matches": [
                {
                    "name": "http-method",
                    "value": "GET"
                }
            ],
            "behaviors": [
                {
                    "params": {
                        "originBasePath": "/",
                        "cacheKeyValue": "-",
                        "digitalProperty": "www.example.com",
                        "cacheKeyType": "origin",
                        "httpPort": 80,
                        "hostHeaderValue": "-",
                        "originDomain": "www.example.com",
                        "hostHeaderType": "origin"
                    },
                    "name": "origin",
                    "type": "-",
                    "value": "-"
                },
                {
                    "name": "content-refresh",
                    "type": "fixed",
                    "value": "1m"
                }
            ]
        }
    ],
    "activationStatus": "ACTIVE"
}
```

Delete a policy

Deletes the currently active policy file for the selected subcustomer domain and activation network.

Request

DELETE `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/sub-properties/www.example.com/policy`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID when the configuration is created.
`domainName`	String	`www.example.com`	The name of the subcustomer's domain.

Steps

1. Determine which Akamai `network` holds the policy you want to delete. Is it on the `staging` (testing) or `production` (live) network?

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Run the *List all sub-properties for a base configuration* operation and store the value from the `subPropertyIDs` array, that corresponds to the appropriate subcustomer's `customerID`. Use this as the `domainName` URL parameter for this operation.

4. Make a DELETE request to `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy`.

5. Review the `message` in the response to verify the delete was successful.

Response

Status 200 `application/json`

Response body:

```
{
    "message": "Successfully deleted",
    "description": "The policy was successfully deleted.",
    "successInstanceId": "a123bcedfg45"
}
```

Get policy history

Returns policy change information for the selected subcustomer domain, base configuration (`propertyId`), and activation network.

Request

GET `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy/history`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/sub-properties/www.example.com/policy/history`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID when the configuration is created.
`domainName`	String	`www.example.com`	The name of the subcustomer's domain.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Run the *List all sub-properties for a base configuration* on page 113 operation and store the appropriate value from the `subPropertyIDs` array, which corresponds to the `domainName` URL parameter.

4. Make a GET request to `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy/history`.

5. Review the rule information returned in the response.

The response includes up to 100 of the most recent versions, not including the current version. If there are more than 100 versions of a policy, ACE discards the oldest policy. (For example, version 101 replaces version 1.)

Response

Status 200 `application/json`

Object type: *Policy*

Response body:

```
{
    "policy": {
        "rules": [
            {
                "behaviors": [
                    {
                        "name": "origin",
                        "params": {
                            "cacheKeyType": "digital_property",
                            "cacheKeyValue": "-",
                            "digitalProperty": 
"sportsguys.wsdtest.akalab.com",
                            "hostHeaderType": "digital_property",
                            "hostHeaderValue": "-",
                            "originDomain": "sportsguys.wsdtest.akalab.com.s3-
website-us-west-1.amazonaws.com"
                        },
                        "value": "-"
                    }
                ],
                "matches": [
                    {
                        "name": "http-method",
                        "value": "GET"
                    }
                ]
            },
            {
                "behaviors": [
                    {
                        "name": "caching",
                        "type": "fixed",
                        "value": "1h"
                    }
                ],
                "matches": [
                    {
                        "name": "url-extension",
                        "value": "jpg"
                    }
                ]
            },
            {
                "behaviors": [
                    {
                        "name": "caching",
                        "type": "fixed",
                        "value": "1d"
                    }
                ],
                "matches": [
                    {
                        "name": "url-extension",
                        "value": "mp4"
                    }
                ]
            }
        ]
    },
    "update_timestamp": 1440722458,
    "version": 1
}
```

Get latest active policy

Returns the latest activated version of the policy, one with an `activationStatus` of `ACTIVE`. If no active version exists, the API returns a 404 error.

Request

GET `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy/active`

Sample: `/partner-api/v2/network/production/properties/a123bcdefg45/sub-properties/www.example.com/policy/active`

Parameters

Parameter	Type	Sample	Description
URL path parameters			
`network`	Enumeration	`production`	The network for the policy, either `staging` for the testing network, or `production` for the live network.
`propertyId`	String	`a123bcdefg45`	The ID number of the base configuration file. Akamai Cloud Embed (ACE) automatically generates this ID when the configuration is created.
`domainName`	String	`www.example.com`	The name of the subcustomer's domain.

Steps

1. Determine which `network` you want to gather information from. You can choose `staging` (testing) or `production` (live).

2. Contact your Akamai account representative for the `propertyId` assigned to the applicable base configuration.

3. Run the *List all sub-properties for a base configuration* on page 113 operation and store the appropriate value from the `subPropertyIDs` array, which corresponds to the `domainName` URL parameter.

4. Make a GET request to `/partner-api/v2/network/{network}/properties/{propertyId}/sub-properties/{domainName}/policy/active`.

5. Review the rule information returned in the response.

Response

Status 200 `application/json`

Object type: *Policy*

Response body:

```
{
    "rules": [
        {
            "matches": [
                {
                    "name": "http-method",
                    "value": "GET"
                }
            ],
            "behaviors": [
                {
                    "params": {
                        "originBasePath": "/",
                        "cacheKeyValue": "-",
                        "digitalProperty": "www.example.com",
                        "cacheKeyType": "origin",
                        "httpPort": 80,
                        "hostHeaderValue": "-",
                        "originDomain": "www.example.com",
                        "hostHeaderType": "origin"
                    },
                    "name": "origin",
                    "type": "-",
                    "value": "-"
                },
                {
                    "name": "content-refresh",
                    "type": "fixed",
                    "value": "1m"
                }
            ]
        }
    ],
    "activationStatus": "ACTIVE"
}
```

Data

This section shows you the data model for the Akamai Cloud Embed API.

The main “{Object}” sections reveal an example of the JSON returned for a GET call. The “{Object} Members” sub-sections illustrate elements that are required or optional, based on the call method. For example, “required” indicates the element *must be used* in the request body for a PUT method call, and it is *always included* in the response for a GET call.

The data schema tables below list membership requirements as follows:

✓	Member is required in requests, or always present in responses, even if its value is empty or `null`.
○	Member is optional, and may be omitted in some cases.
✗	Member is out of scope, and irrelevant to the specified interaction context. If you include the member in that context, it either triggers an error, or is ignored.

SubCustomer

Allows you to add or update a subcustomer.

Sample GET response:

```
[
    {
        "customerID": "abc-123",
        "geo": "FR",
        "subPropertyIDs": [
            "www.example.com",
            "www.example.biz",
            "www.example.net",
            "www.example.co.uk"
        ]
    }
]
```

SubCustomer members

Member	Type	Required	Description
`customerID`	String	✓	The ID of the customer assigned to the subproperty.
`geo`	String	✓	The geographic location of the subcustomer.
`subPropertyIDs`	Array	✓	Individual, unique strings that represent

Member	Type	Required	Description
			each subcustomer’s domain.

SubProperty

Lists ID, domain, and geographic information for a subproperty, which is the subcustomer’s domain.

Sample GET response:

```
[
    {
        "subPropertyID": "www.example.com",
        "customerID": "abc-123"
    },
    {
        "subPropertyID": "www.example.biz",
        "customerID": "abc-123"
    },
    {
        "subPropertyID": "www.example.com.net",
        "customerID": "def-456"
    },
    {
        "subPropertyID": "www.example.com.co.uk",
        "customerID": "def-456"
    }
]
```

SubProperty members

Member	Type	Required	Description
`SubProperty`: Lists ID, domain, and geographic information for a subproperty, which is the subcustomer’s domain.			
`customerID`	String	○	The ID of the customer assigned to the subproperty.
`geo`	String	○	The geographic location of the subcustomer.
`subPropertyID`	String	○	A unique string representing the subcustomer’s domain.

Policy

Encapsulates rules consisting of match criteria and resulting behavior settings Akamai Cloud Embed (ACE) applies for a specific subcustomer.

Sample GET response:

```
{
    "rules": [
        {
            "matches": [
                {
                    "name": "http-method",
                    "value": "GET"
                }
            ],
            "behaviors": [
                {
                    "params": {
                        "originBasePath": "/",
                        "cacheKeyValue": "-",
                        "digitalProperty": "www.example.com",
                        "cacheKeyType": "origin",
                        "httpPort": 80,
                        "hostHeaderValue": "-",
                        "originDomain": "www.example.com",
                        "hostHeaderType": "origin"
                    },
                    "name": "origin",
                    "type": "-",
                    "value": "-"
                },
                {
                    "name": "content-refresh",
                    "type": "fixed",
                    "value": "1m"
                }
            ]
        }
    ],
    "activationStatus": "ACTIVE"
}
```

Sample of a GET on an older policy:

```
{
    "policy": {
        "rules": [
            {
                "behaviors": [
                    {
                        "name": "origin",
                        "params": {
                            "cacheKeyType": "digital_property",
                            "cacheKeyValue": "-",
                            "digitalProperty": 
"sportsguys.wsdtest.akalab.com",
                            "hostHeaderType": "digital_property",
                            "hostHeaderValue": "-",
                            "originDomain": "sportsguys.wsdtest.akalab.com.s3-
website-us-west-1.amazonaws.com"
                        },
                        "value": "-"
                    }
                ],
                "matches": [
                    {
                        "name": "http-method",
                        "value": "GET"
                    }
                ]
            },
            {
                "behaviors": [
                    {
                        "name": "caching",
                        "type": "fixed",
                        "value": "1h"
                    }
                ],
                "matches": [
                    {
                        "name": "url-extension",
                        "value": "jpg"
                    }
                ]
            },
            {
                "behaviors": [
                    {
                        "name": "caching",
                        "type": "fixed",
                        "value": "1d"
                    }
                ],
                "matches": [
                    {
                        "name": "url-extension",
                        "value": "mp4"
                    }
                ]
            }
        ]
    },
    "update_timestamp": 1440722458,
    "version": 1
}
```

Sample of a GET on the latest active policy:

```
{
    "rules": [
        {
            "matches": [
                {
                    "name": "http-method",
                    "value": "GET"
                }
            ],
            "behaviors": [
                {
                    "params": {
                        "originBasePath": "/",
                        "cacheKeyValue": "-",
                        "digitalProperty": "www.example.com",
                        "cacheKeyType": "origin",
                        "httpPort": 80,
                        "hostHeaderValue": "-",
                        "originDomain": "www.example.com",
                        "hostHeaderType": "origin"
                    },
                    "name": "origin",
                    "type": "-",
                    "value": "-"
                },
                {
                    "name": "content-refresh",
                    "type": "fixed",
                    "value": "1m"
                }
            ]
        }
    ],
    "activationStatus": "ACTIVE"
}
```

Policy members

Member	Type	PUT	Description
`Policy`: Encapsulates rules consisting of match criteria and resulting behavior settings Akamai Cloud Embed (ACE) applies for a specific subcustomer.			
`activationStatus`	Enumeration	○	The current propagation status of the policy. This can be `PROPAGATING` for a policy that is still being processed, `ACTIVE` for a policy that has successfully completed propagation, or `ACTIVATION_FAILED` for a policy that encountered an error during propagation. *Get the policy* to check its configuration and *update the policy* to resubmit it for propagation.
`rules`	Policy.rules[]	✓	The set of matches and behaviors for this subcustomer policy.
`Policy.rules[]`: The set of matches and behaviors for this subcustomer policy.			
`behaviors`	Policy.rules[].behaviors[]	✓	The behaviors to apply to requests that meet the match criteria set in this policy. See *Behaviors* below to

Member	Type	PUT	Description
			access more details on the supported behaviors.
`matches`	Policy.rules[].matches[]	✓	The criteria that identify which requests to process. When a request matches the criteria, ACE applies the behaviors to the request.
`Policy.rules[].behaviors[]`: The behaviors to apply to requests that meet the match criteria set in this policy.			
`name`	Enumeration	✓	The specific behavior you want to apply to incoming requests that meet the associated match criteria.
`params`	Object	o	The set of options that determine how the behavior operates.
`type`	String	✓	The format to use for the `value` element.
`value`	String	✓	The values to use for the selected behavior.
`Policy.rules[].matches[]`: The criteria that identify which requests to process. When a request matches the criteria, ACE applies the behaviors to the request.			
`name`	Enumeration	✓	The specific match criteria that must be met to apply the `behaviors` you define. See *Matches* below to access more details on the supported matches.
`negated`	Boolean	o	If set to `true`, ACE applies the behaviors set in this rule when an incoming request does not meet the match criteria.
`value`	String	✓	The value to match.

Matches

A match defines the criteria to be met in a request for subcustomer content in order to apply the behaviors set in a rule.

- *client-ip*
- *cookie*
- *geography*
- *header*
- *host-name*
- *http-method*
- *url-extension*
- *url-filename*
- *url-path*
- *url-querystring*
- *url-scheme*
- *url-wildcard*

Behaviors

Behaviors determine the settings ACE applies to the delivery of subcustomer content. Akamai Cloud Embed (ACE) applies a behavior's settings when a request that meets the rule's match criteria is received for a subcustomer's content.

- *access-control*
- *cachekey-query-args*
- *caching*
- *content-characteristics (for dynamic web content)*
- *content-characteristics (for large files)*
- *content-characteristics (for on-demand streaming)*
- *content-characteristics (for live streaming)*
- *content-compression*
- *content-refresh*
- *downstream-caching*
- *geo-blacklist*
- *geo-whitelist*
- *ip-blacklist*
- *ip-whitelist*
- *modify-outgoing-request-header*
- *modify-outgoing-request-path*
- *modify-outgoing-response-header*
- *origin*
- *origin-characteristics*
- *origin-failover*
- *referer-blacklist*
- *referer-whitelist*
- *site-failover*
- *token-auth*
- *url-redirect*

The "client-ip" match

Include this to match using the IP address assigned to the requesting client. You can specify individual IP addresses, or CIDR blocks (that express a range of addresses).

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "client-ip",
            "value": "-",
            "params": {
                "ipOrCidrList": [
                    "1.2.3.4",
                    "5.6.7.8",
                    "192.168.100.14/24",
                    "2001:db8:abcd:8000::/50"
                ],
                "source": "both",
                "ipFromHeader": "all"
            },
            "negated": true
        }
    ],
    "behaviors": []
}
```

client-ip members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `client-ip` to match based on the requesting client’s IP address.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`params`	client-ip.params[]	✓	Specifies each set of addresses along with criteria to match them.
`value`	String	✓	All match criteria require the `value` member, but `client-ip` does not use one. Specify a dash for this member (`-`).
`client-ip.params[]`: Specifies each set of addresses along with criteria to match them.			
`ipFromHeader`	Enumeratio n	○	When matching the `X-Forwarded-For` header, specifies which IP address to match in the list, either the `first` IP address, or `all` to match any.
`ipOrCidrList`	Array	✓	Enter one or more specific IP addresses or CIDR blocks of IP addresses. If a requesting client is assigned to one of these IP addresses, the behaviors set in this policy are applied. For

Member	Type	Required	Description
			example, `["1.2.3.4", "5.6.7.8", "192.168.100.14/24", "2001:db8:abcd:8000::/50"]`.
`source`	Enumeration	○	Specifies where to find the IP address, either `connectingIp` for the requesting client's IP address, `headers` for the `X-Forwarded-For` header, or `both` to match either location.

The "cookie" match

Include this match to define specific cookie names for use when matching on an incoming request.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "cookie",
            "value": "cookie-name cookie-value1 cookie-value2",
            "negated": true
        }
    ],
    "behaviors": []
}
```

cookie members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `cookie` to use cookie names when matching on the incoming request.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Specifies the case-insensitive cookie name to match on. You can follow the cookie name with a space-delimited list of cookie values to match against, all of which are case-sensitive. Cookie name and cookie values do not support white spaces. Currently, the `+ ! ( ) [ ] { }` characters are not supported, even though they are normally supported for use in cookies. For example, `cookiename cookievalue1 cookievalue2`.

The "geography" match

Use this match to test the requesting client’s location, either by continent, country, region, or designated market area (DMA). Each subcustomer policy can include up to ten `geography` matches.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "geography",
            "value": "-",
            "params": {
                "continent": [
                    "AS",
                    "EU"
                ],
                "country": [
                    "IN",
                    "CA"
                ],
                "region": [
                    "US:CA",
                    "US:TX"
                ],
                "dma": [
                    500,
                    501,
                    502
                ],
                "source": "both",
                "ipFromHeader": "first"
            }
        }
    ],
    "behaviors": []
}
```

geography members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `geography` to match based on the requesting client’s geographic location.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`params`	See the `[geography.params]` array.	✓	Specifies any geographic area to match. You need to include at least one `continent`, `country`, `region`, or `dma` type, and at least one value for each type, if specified. Any specified value matches, so if you include `NA` (North America) as

Member	Type	Required	Description
			a `continent`, you don't need to include `US` as a `country`.
`value`	String	✓	All match criteria require the `value` member, but `geography` does not use one. Specify a dash for this member (`-`).
`geography.params`: Specifies any geographic area to match. You need to include at least one `continent`, `country`, `region`, or `dma` type, and at least one value for each type, if specified. Any specified value matches, so if you include `NA` (North America) as a `continent`, you don't need to include `US` as a `country`.			
`continent`	Array	○	Specify a list of two-letter continents to match: `AF` for Africa, `AS` for Asia, `EU` for Europe, `NA` for North America, `OC` for Oceania, `SA` for South America, and `OT` for all others.
`country`	Array	○	Specify a list of two-letter countries to match. For example, `US` for United States and `IN` for India. A list of supported Country Codes is available on *Akamai Control Center*. Open the **Data Codes** entry and click **country_codes.csv** to download the list.
`dma`	Array	○	This only applies to the United States. Specify a list of numeric designations as string values to serve as designated market area (DMA) codes. A DMA is a specific Nielsen Media Research area in which the population can receive the same, or similar Internet media offerings. A list of supported DMA codes is available on *Akamai Control Center*. Open the **Data Codes** entry and click **dma_list.txt** to download the list.
`ipFromHeader`	Enumeration	○	When matching the `X-Forwarded-For` header for the `source`, specifies which geographic client IP information to match, either the `first` client IP listed for a specified geographic location, or `all` to match any. The default is `first`.
`region`	Array	○	Specify a list of regions to match. Use the two letter designation for a specific region (state or territory), prefaced with the applicable `country` and a colon. For example, `US:CA` for California, United States or `CA:BC` for British Columbia, Canada. A list of supported Country Codes is available on *Akamai Control Center*. Open the **Data Codes** entry and click country_codes.csv to download the list.
`source`	Enumeration	○	Specifies where to find the geographic information, either `connectingIp` to use the requesting client's IP address, `headers` to use the `X-Forwarded-For` header, or `both` to match either location. The default is `both`.

The "header" match

Associated behaviors are applied if a header or header value you specify in this match criteria are included with a request.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "header",
            "value": "header-name header-value1 header-value2",
            "negated": false
        }
    ],
    "behaviors": []
}
```

header members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `header` to match on an incoming request header or header value.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Enter a header name, and optionally include an associated header value, for the match (for example, `Some-Header-Name` or `Some-Header-Name "some string value" "another value"` or). Separate multiple header value entries with spaces. Header values are also case sensitive.

The "host-name" match

Include this to match on hostnames listed in the incoming request's Host header.

Sample

Here's a ample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "host-name",
            "value": "host-name1 host-name2 host-name3",
            "negated": false
        }
    ],
    "behaviors": []
}
```

host-name members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `host-name` to match on hostnames listed in the incoming request's Host header.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Enter a space-separated list of hostnames to match. You can use the `?` and `*` wildcards with this match value.

The "http-method" match

Include this to match on a set of HTTP methods included in a client request.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "http-method",
            "value": "PUT POST",
            "negated": true
        }
    ],
    "behaviors": []
}
```

http-method members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `http-method` to match based on HTTP method.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Enter the HTTP methods to match on, separated by spaces. The following values are supported: `GET`, `HEAD`, `POST`, `PUT`, `DELETE`, `OPTIONS`, `TRACE`, and `CONNECT`.

The "url-extension" match

Include this to match on the extension in the incoming request. This match criteria has no effect on URL paths that do not include a file extension.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "url-extension",
            "value": "jpg png gif",
            "negated": false
        }
    ],
    "behaviors": []
}
```

url-extension members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `url-extension` to match on the extension in the incoming request.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Enter the filename extensions to match on, separated by spaces. This string cannot be empty and entries are case sensitive.

The "url-filename" match

Include this to match on the filename and extension included in the incoming request.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "url-filename",
            "value": "filename.ext myfile.ext",
            "negated": false
        }
    ],
    "behaviors": []
}
```

url-filename members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `url-filename` to match on the filename and extension included in the incoming request.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Enter a space-separated list of filenames to match. Wildcards are not supported. (Include the full name of the target file.) The filename can be in any subdirectory or URL path. For example, `filename.ext` matches on both `/filename.ext` and `/path/to/filename.ext`.

The "url-path" match

Include this to match on the first path component in the incoming request. The first path component is the section directly after the base URL.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "url-path",
            "value": "new-files old-files",
            "negated": false
        }
    ],
    "behaviors": []
}
```

url-path members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `url-path` to match on the first path component in the incoming request.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Enter a space-separated list of path names to match on. Do not include slashes with your entry. This match is case sensitive and does not support wildcard characters. For example, entering `new-files old-files` matches on all URI paths that begin with `/new-files/` or `/old-files/`, like `/new-files/dir1/dir2/filename.ext`.

The "url-querystring" match

Include this to match on a combination of query string parameters and their values.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "url-querystring",
            "params": [
                {
                    "key": "test*"
                },
                {
                    "key": "version*",
                    "values": "beta* *test*"
                },
                {
                    "key": "mode",
                    "values": "bypass"
                }
            ],
            "negated": false
        }
    ],
    "behaviors": []
}
```

url-querystring members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `url-querystring` to match on a combination of query string parameters and their values.
`params`	See the `url-querystring.params[]` array.	○	The parameters that define how you want to handle query strings for your implementation.
`url-querystring.params[]`: The parameters that define how you want to handle query strings for your implementation.			
`key`	String	○	Enter the query string to match on. You can use the `?` and `*` wildcards with your entry.
`value`	String	○	Enter the query string values to match on, separated by spaces. You can use the `?` and `*` wildcards with this match value.

The "url-scheme" match

Include this to match on the protocol or scheme (HTTP or HTTPS) of an incoming request.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "url-scheme",
            "value": "HTTP",
            "negated": true
        }
    ],
    "behaviors": []
}
```

url-scheme members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `url-scheme` to match on the protocol or scheme (HTTP or HTTPS) of an incoming request.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	Enumeration	✓	Enter either `HTTP` or `HTTPS` depending on the protocol you want to match on.

The "url-wildcard" match

Include this match to use wildcards when matching on the incoming request path, minus any query strings. This match type only supports the `*` wildcard.

Sample

Here's a sample inclusion for this match in a PUT policy operation:

```
{
    "matches": [
        {
            "name": "url-wildcard",
            "value": "/styles/* /images/baseball.png",
            "negated": false
        }
    ],
    "behaviors": []
}
```

url-wildcard members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `url-wildcard` to use wildcards when matching on the incoming request path, minus query strings.
`negated`	Boolean	○	If set to `true`, Akamai Cloud Embed (ACE) applies the behaviors set in the subcustomer policy when an incoming request does not meet the match criteria.
`value`	String	✓	Enter the path to match on. If you do not include a wildcard, the incoming request must match the value exactly as entered.

The "access-control" behavior

Include this behavior to allow or deny client requests based on what's specified as the `matches` criteria.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "access-control",
            "type": "denied",
            "value": "deny-protocol"
        }
    ]
}
```

access-control members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `access-control` to deny client requests based on any of the available match conditions.
`type`	Enumeration	✓	Either `deny` or `allow` an incoming request that matches on this rule.
`value`	String	✓	Enter strings of up to 64 characters to annotate logs with the approval or denial reason. Alphanumeric characters, dashes, and underscores are valid. For a `type` of `allow`, you can also use the `*` wildcard character. Separate strings with spaces.

The "cachekey-query-args" behavior

Include this behavior to determine how query string arguments within an incoming request should be handled.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "cachekey-query-args",
            "type": "include",
            "value": "product=1"
        }
    ]
}
```

cachekey-query-args members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `cachekey-query-args` to define how query-string arguments are handled when creating the cache key for the Akamai edge server.
`type`	Enumeration	✓	Options for handling query-string arguments. Use `include-all` to include all query-string values in the cache key. Use `include` to only append the query arguments listed in the cache key's `value` attribute. Use `ignore-all` to remove all query-string arguments from the incoming request. Use `ignore` to not include the query arguments listed in the `value` attribute in the cache key.
`value`	String	✓	If using `include` or `ignore` as the caching type, list the query strings to match on. The match is case sensitive. Use spaces to separate entries. Add an equal sign to the value to match the query string argument exactly and also extend the value of the query argument. For example, entering `product=1` matches both `product=1` and `product=123` but not `product=234`. Not required for the `include-all` and `ignore-all` type settings.

The "caching" behavior

Include this behavior to define time-to-live (TTL) cache settings for subcustomers.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "caching",
            "type": "fixed",
            "value": "86400s"
        }
    ]
}
```

caching members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `caching` to configure the time-to-live (TTL) settings for a subcustomer based on specific match conditions.
`type`	Enumeration	✓	Select the type of cache response to use. Choose `no-store` to never cache the response and evict any existing cache entry; `bypass-cache` to never cache the response and retain the existing cache entry; `fixed` to cache the response for the time period defined in `value`; `honor` to cache based on the values in the `cache-control` and `expires` headers; `honor-cc` to cache based on the values in the `cache-control` header; or `honor-expires` to cache based on the values in the `expires` header.For the honor-based options, objects currently in cache may be used with the current request. Also, if the response doesn’t include the Cache-Control or Expires header, the object is cached based on the time listed in the `value` parameter. For the `honor` and `honor-cc` options, if the Cache-Control header includes a `no-store` or `no-cache` directive, the Akamai server doesn’t cache the response.No caching occurs if the Expires header has an *RFC 2616* time string in the past or includes `-1`, which helps prevent downstream caching.
`value`	String	✓	Ignored for the `no-store` and `bypass-cache` types. For all other type settings, enter a simple duration string, which is an integer followed by a specifier to represent seconds (`s`), minutes (`m`), hours (`h`), or days (`d`). For example: `86400s`, or

Member	Type	Required	Description
			`1440m`, or `24h`, or `1d` all represent a TTL setting of one day.

The "content-characteristics-dynamic-web-content" behavior

Include the `content-characteristics` behavior and set the `type` to `dynamic-web-content` if you're using Integrated Cloud Acceleration, to use SureRoute to optimize the forward path to the origin server. It controls embedded object prefetching and situational image compression.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "content-characteristics",
            "type": "dynamic-web-content",
            "value": "-",
            "params": {
                "mobileImageCompressionEnabled": true,
                "prefetchEnabled": false,
                "realUserMonitoringEnabled": true,
                "sureRouteTestObjectPath": "/sureroute"
            }
        }
    ]
}
```

content-characteristics-dynamic-web-content members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `content-characteristics`, the behavior that includes this optimization.
`params`	See the `[content-characteristics-dynamic-web-content.params]` array	○	Optimization settings for this behavior.
`type`	Enumeration	✓	Enter `dynamic-web-content` to use Akamai's SureRoute feature to optimize the forward path to the origin server.
`value`	String	✓	Ignored for this behavior. For consistency, enter a dash (`-`).
`content-characteristics-dynamic-web-content.params`: Optimization settings for this behavior.			

Member	Type	Required	Description
mobileImageCompressionEnabled	Boolean	○	Enable or disable JPEG compression based on mobile network conditions.
prefetchEnabled	Boolean	○	Inspects HTML responses and prefetches embedded objects in HTML files. Prefetching works on any page that includes `<img>`, `<script>`, or `<link>` tags that specify relative paths. It also works when the resource hostname matches the request domain in the HTML file, and it is part of a fully qualified URI. When set to `true`, edge servers prefetch objects with the following file extensions: `aif`, `aiff`, `au`, `avi`, `bin`, `bmp`, `cab`, `carb`, `cct`, `cdf`, `class`, `css`, `doc`, `dcr`, `dtd`, `exe`, `flv`, `gcf`, `gff`, `gif`, `grv`, `hdml`, `hqx`, `ico`, `ini`, `jpeg`, `jpg`, `js`, `mov`, `mp3`, `nc`, `pct`, `pdf`, `png`, `ppc`, `pws`, `swa`, `swf`, `txt`, `vbs`, `w32`, `wav`, `wbmp`, `wml`, `wmlc`, `wmls`, `wmlsc`, `xsd`, and `zip`.
sureRouteTestObjectPath	String	○	Enable SureRoute by entering a valid path to the test object on your origin. A valid test object is between 4 KB to 12 KB compressed and requires no authorization. SureRoute looks for the optimal route between an edge server and an origin server.

The "content-characteristics-large-files" behavior

Include the `content-characteristics` behavior and set the `type` to `large-files` to optimize the delivery of large file downloads of up to 1.8 GB. This behavior uses partial object caching with prefetched object data.

As a best practice, only use this behavior if you serve large files. Otherwise, the Akamai platform may send additional requests to your origin. When using Large File Optimization, if an object doesn't meet the minimum size criterion of 10 MB, the platform requests the entire object from the origin.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "content-characteristics",
            "type": "large-files",
            "value": "-",
            "params": {
                "objectSize": "10mbto100mb"
            }
        }
    ]
}
```

content-characteristics-large-files members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `content-characteristics`, the behavior that includes this optimization.
`params`	See the `[content-characteristics-large-files.params]` array	○	The parameters that define how you want to handle file optimization for your implementation.
`type`	Enumeration	✓	Enter `large-files` to optimize the delivery of large file downloads of up to 1.8 GB.
`value`	String	✓	Ignored for this behavior. For consistency, enter a dash (`-`).
`content-characteristics-large-files.params`: The parameters that define how you want to handle file optimization for your implementation.			
`objectSize`	Enumeration	○	Options include: `lt1mb` for objects less than 1 MB, `1mbto10mb` for objects 1 MB to 10 MB, `10mbto100mb` for objects 10 MB to 100 MB, and `gt100mb` for objects 100 MB and larger. Both `10mbto100mb` and `gt100mb` enable Large File Optimization. Using `lt1mb` or `1mbto10mb` disables partial object caching and prevents the platform from serving large objects.

The "content-characteristics-on-demand-streaming" behavior

Include the `content-characteristics` behavior and set the `type` to `streaming-video-on-demand` to optimize caching and network timeout conditions for on-demand video content.

The Akamai platform examines the URI file extension and path for the media format. It then automatically optimizes cache efficiency, time-to-live, automated failover, downstream `Content-Type` headers, and network timeout settings.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "content-characteristics",
            "type": "streaming-video-on-demand",
            "value": "-",
            "params": {
                "segmentDurationDASH": "3.0",
                "segmentDurationHDS": "4.0",
                "segmentDurationHLS": "7.0",
                "segmentDurationSmooth": "3.0",
                "prefetch": {
                    "originAssist": true
                }
            }
        }
    ]
}
```

content-characteristics-on-demand-streaming members

Member	Type	Required	Description
`name`	Enumeration	✓	Specify `content-characteristics`, the behavior that includes this optimization.
`params`	See the `[content-characteristics-on-demand-streaming.params]` array	○	The parameters that define how you want to handle on-demand video streaming.
`type`	Enumeration	✓	Specify `streaming-video-on-demand` to optimize cache and network timeout conditions for on-demand video content.
`value`	String	✓	Ignored for this behavior. For consistency, enter a dash (`-`).
`content-characteristics-on-demand-streaming.params`: The parameters that define how you want to handle on-demand video streaming.			
`prefetch`	See the `[content-characteristics-on-demand-streaming.params.prefetch]` array	○	To reduce delivery time, segment prefetching places target media content at the edge to anticipate end-user requests. *Note:* You can't use `prefetch` with HDS format media.
`segmentDurationDASH`	Number	○	The duration in seconds for Dynamic Adaptive Streaming over HTTP (DASH) media segments for this subcustomer. The supported range is `0` to `15`, with only one decimal place allowed. If you omit this member, ACE uses the duration set for **DASH Segment Duration** in the Content Characteristics - Streaming Video On Demand behavior in the assigned base configuration. Otherwise, ACE uses the default segment duration for DASH (6.0 seconds).
`segmentDurationHDS`	Number	○	The duration in seconds for HTTP Dynamic Streaming (HDS) media fragments for this

Member	Type	Required	Description
			subcustomer. The supported range is `0` to `15`, with only one decimal place allowed. If you omit this member, ACE uses the duration set for **HDS Fragment Duration** in the Content Characteristics - Streaming Video On Demand behavior in the assigned base configuration. Otherwise, ACE uses the default fragment duration for HDS (6.0 seconds).
`segmentDurationHLS`	Number	`o`	The duration in seconds for HTTP Live Streaming (HLS) media segments. The supported range is `0` to `15`, with only one decimal place allowed. For recommended best practices, see *Apple Technical Note 2224*. If you omit this member, ACE uses the duration set for **HLS Segment Duration** in the Content Characteristics - Streaming Video On Demand behavior in the assigned base configuration. Otherwise, ACE uses the default segment duration for HLS (10.0 seconds).
`segmentDurationSmooth`	Number	`o`	The duration in seconds for Microsoft Smooth Streaming media fragments. The supported range is `0` to `15`, with only one decimal place allowed. If you omit this member, ACE uses the duration set for **Smooth Fragment Duration** in the Content Characteristics - Streaming Video On Demand behavior in the assigned base configuration. Otherwise, ACE uses the default fragment duration for Smooth (2.0 seconds).
`content-characteristics-on-demand-streaming.params.prefetch`: To reduce delivery time, segment prefetching places target media content at the edge to anticipate end-user requests. *Note:* You can't use `prefetch` with HDS format media.			
`originAssist`	Boolean	`o`	This enables prefetching using the *origin-assist scheme*. When Akamai fetches an object from an origin, the response needs to include a new header that lists the next object in the sequence. Akamai can read this information and prefetch this object. This scheme relies on assistance from your "intelligent" origin to trigger prefetching. Requirements and setup of an origin are explained in *Add prefetching support for video* on page 91.

The "content-characteristics-live-streaming" behavior

Include the `content-characteristics` behavior and set the `type` to `streaming-video-live` to optimize caching and network timeout conditions for live video content.

The Akamai platform examines the URI file extension and path for the media format. It then automatically optimizes cache efficiency, time-to-live, automated failover, downstream `Content-Type` headers, and network timeout settings.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "content-characteristics",
            "type": "streaming-video-live",
            "value": "-",
            "params": {
                "segmentDurationDASH": "3.0",
                "segmentDurationHDS": "4.0",
                "segmentDurationHLS": "7.0",
                "segmentDurationSmooth": "3.0",
                "prefetch": {
                    "originAssist": true
                }
            }
        }
    ]
}
```

content-characteristics-live-streaming members

Member	Type	Required	Description
`name`	Enumeration	✓	Specify `content-characteristics`, the behavior that includes this optimization.
`params`	See the `[content-characteristics-live-streaming.params]` array	○	The parameters that define how you want to handle live video streaming.
`type`	Enumeration	✓	Specify `streaming-video-live` to optimize delivery for live video content.
`value`	String	✓	Ignored for this behavior. For consistency, enter a dash (`-`).
`content-characteristics-live-streaming.params`: The parameters that define how you want to handle live video streaming.			
`prefetch`	See the `[content-characteristics-live-streaming.params.prefetch]` array	○	To reduce delivery time, segment prefetching places target media content at the edge to anticipate end-user requests. *Note:* You can't use `prefetch` with HDS format media.

Member	Type	Required	Description
`segmentDurationDASH`	Number	○	The duration in seconds for Dynamic Adaptive Streaming over HTTP (DASH) media segments. The supported range is 0 to 10, with only one decimal place allowed. If you omit this member, ACE uses the default segment duration for DASH (6.0 seconds).
`segmentDurationHDS`	Number	○	The duration in seconds for HTTP Dynamic Streaming (HDS) media fragments. The supported range is 0 to 10, with only one decimal place allowed. If you omit this member, ACE uses the default fragment duration for HDS (6.0 seconds).
`segmentDurationHLS`	Number	○	The duration in seconds for HTTP Live Streaming (HLS) media segments. The supported range is 0 to 10, with only one decimal place allowed. For recommended best practices, see *Apple Technical Note 2224*. If you omit this member, ACE uses the default segment duration for HLS (10.0 seconds).
`segmentDurationSmooth`	Number	○	Duration in seconds for Microsoft Smooth Streaming media fragments. The supported range is 0 to 10, with only one decimal place allowed. If you omit this member, ACE uses the default fragment duration for Smooth (2.0 seconds).
`content-characteristics-live-streaming.params.prefetch`: To reduce delivery time, segment prefetching places target media content at the edge to anticipate end-user requests. *Note:* You can't use `prefetch` with HDS format media.			
`originAssist`	Boolean	○	This enables prefetching using the *origin-assist scheme*. When Akamai fetches an object from an origin, the response needs to include a new header that lists the next object in the sequence. Akamai can read this information and prefetch this object. This scheme relies on assistance from your "intelligent" origin to trigger prefetching. Requirements and setup of an origin are explained in *Add prefetching support for video* on page 91.

The "content-compression" behavior

Include this behavior to provide compression settings. You can enable gzip compression, decompress objects before delivering them to the client, or maintain the origin's compression settings.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "content-refresh",
            "type": "date-time",
            "value": "2018-12-31T11:59:59Z",
            "params": {
                "mustRevalidate": true
            }
        }
    ]
}
```

content-compression members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `content-compression` to enable this behavior.
`type`	Enumeratio n	✓	Select the appropriate compression setting for this behavior. Use `always` to compress all objects served to clients that send an `Accept-Encoding:*gzip*` header, `never` to decompress objects served to the client, and `follow-origin` to retain the origin's compression settings.
`value`	String	✓	Enter the MIME types to compress, separated by spaces. The maximum string length is 1024 characters, and wildcards can be used (`*`). Enter – if you do not want to apply compression to any MIME type.

The "content-refresh" behavior

Include this behavior to invalidate the CDN cache at an explicit date and time. This behavior uses epoch time to denote when a request should receive a new copy of the object or a newly validated one.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "content-refresh",
            "type": "epoch",
            "value": "1543273814",
            "params": {
                "mustRevalidate": true
            }
        }
    ]
}
```

content-refresh members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `content-refresh` to invalidate CDN cache at an explicit date/time.

Member	Type	Required	Description
`params`	See the `[content-refresh.params]` array	○	Settings used by this behavior to reapply validation to content.
`type`	Enumeration	✓	Declares the format of the `value` element. Enter `epoch` to use epoch time; `date-time` to use an ISO 8601 time value (`YYYY-MM-DDThh:mm:ssZ`); `date` to use an ISO 8601 date value (`YYYY-MM-DD`), which invalidates cache at midnight GMT on the specified date; and `natural` to invalidate content immediately upon publication of the content policy.
`value`	String	✓	The time after which the invalidation of objects in cache should occur.
`content-refresh.params`: Settings used by this behavior to reapply validation to content.			
`mustRevalidate`	Boolean	○	If `true`, the edge server only serves content from cache if it has been validated again after the given invalidation time. If `false`, the edge server may serve content from cache if an attempt to validate the content again fails to receive a response from the origin server.

The "downstream-caching" behavior

Include this behavior to control downstream caching of alternate content.

Only use this behavior if site failover is enabled for the alternate hostname property. If you do not include this behavior, your subcustomer policy uses the downstream caching settings specified in the alternate hostname property. To enable site failover, use the Subcustomer Enablement behavior in Property Manager.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "downstream-caching",
            "value": "no-store"
        }
    ]
}
```

downstream-caching members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `downstream-caching` to control downstream caching of alternate content.

Member	Type	Required	Description
`value`	Enumeration	✓	Enter `no-store` to serve the alternate response with an HTTP `Cache-Control: no-store` header. Enter `no-cache` to serve the alternate response with an HTTP `Cache-Control: no-cache` header.

The "geo-blacklist" behavior

Include this behavior to block access to content based on the continent, country, region/state, or designated marketing area (DMA) of the requesting IP address. (DMA only applies in the United States.) All other geographic areas are allowed.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "geo-blacklist",
            "type": "region",
            "value": "US:CA US:OR US:WA US:ID US:AZ US:NV"
        }
    ]
}
```

geo-blacklist members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `geo-blacklist` to block access to content based on the geographic location of the requesting IP address.
`type`	Enumeration	✓	Declares the type of geographies to blacklist. Valid types are `continent`, `country`, `region`, and `dma`.
`value`	String	✓	Enter a space-separated list of geographic areas to blacklist. Proper values for these areas are maintained in the *EdgeScape Data Codes* lists on Akamai Control Center. Each `continent` and `country` follows a two-digit format. If matching on `region` values, these follow a `country:region` format like `US:AK` (Alaska in the United States). If matching on `dma`, enter the integer that represents the Designated Market Area (DMA). Larger areas are include smaller ones, so if you include NA (North America) as a continent, you don't need to include US as a country.

The "geo-whitelist" behavior

Include this behavior to allow access to content based on the continent, country, region/state, or designated marketing area (DMA) of the requesting IP address. (DMA only applies in the United States.) All other geographic areas are denied.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "geo-whitelist",
            "type": "country",
            "value": "US:CA US:OR US:WA US:ID US:AZ US:NV"
        }
    ]
}
```

geo-whitelist members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `geo-whitelist` to allow access to content based on the geographic location of the requesting IP address.
`type`	Enumeration	✓	Declares the type of geographies to whitelist. Valid types include: `continent`, `country`, `region`, and `dma`.
`value`	String	✓	Enter a space-separated list of geographic areas to whitelist. Proper values for these areas are maintained in the *EdgeScape Data Codes* lists on Control Center. Each `continent` and `country` follows a two-digit format. If matching on `region` values, these follow a `country:region` format like `US:AK` (Alaska in the United States). If matching on `dma`, enter the integer that represents the Designated Market Area (DMA). Larger areas are inclusive of smaller ones, so if you include NA (North America) as a continent, you don't need to include US as a country.

The "ip-blacklist" behavior

Include this behavior to block access based on the requesting IP address. All other requesting IP addresses are allowed access.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "ip-blacklist",
            "value": "10.10.1.100 10.10.2.100 10.10.3.100"
        }
    ]
}
```

ip-blacklist members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `ip-blacklist` to block access based on the requesting IP address.
`value`	String	✓	Enter a space-separated list of IP addresses or CIDR blocks to deny. All other IP addresses are allowed.

The "ip-whitelist" behavior

Include this behavior to allow access based on the requesting IP address. Only the IP addresses listed are allowed access.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "ip-whitelist",
            "value": "10.10.1.100 10.10.2.100 10.10.3.100"
        }
    ]
}
```

ip-whitelist members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `ip-whitelist` to allow access based on the requesting IP address.
`value`	String	✓	Enter a space-separated list of IP addresses or CIDR blocks to allow. All other IP addresses are blocked.

The "modify-outgoing-request-header" behavior

Include this behavior to modify the outgoing request headers sent from Akamai to an origin. This also works on request headers sent from a client if the request is sent back to the origin, but not a cache hit.

Samples

Here's a sample of this behavior in a policy to append the header values, `add_this_value` and `also_add_this_value` to the outgoing request header, `header1`; append `add_this_value` to the outgoing request header, `header2`; and set a comma plus a space (`,`) as the delimiter:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "modify-outgoing-request-header",
            "type": "append",
            "value": "-",
            "params": {
                "headerList": [
                    {
                        "header1": "add_this_value, also_add_this_value"
                    },
                    {
                        "header2": "add_this_value"
                    }
                ],
                "delimiter": ", "
            }
        }
    ]
}
```

This is a sample of this behavior in a policy to delete the header values, `this_is_an_old_value` and `this_is_an_older_value` from the outgoing request header, `header1`; delete the empty header, `header2`; and set the semicolon and a space (`;`) set as a delimiter:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "modify-outgoing-request-header",
            "type": "delete",
            "value": "-",
            "params": {
                "headerList": [
                    {
                        "header1": "this_is_an_old_value;
this_is_an_older_value"
                    },
                    {
                        "header2": ""
                    }
                ],
                "delimiter": "; "
            }
        }
    ]
}
```

Finally, here's a sample of this behavior in a policy to match the header, `header1` and overwrite its header value with `replace_with_this_new_header_value` (with no `delimiter`, because an `overwrite` doesn't use one):

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "modify-outgoing-request-header",
            "type": "overwrite",
            "value": "-",
            "params": {
                "headerList": [
                    {
                        "header1": "replace_with_this_new_header_value"
                    }
                ]
            }
        }
    ]
}
```

modify-outgoing-request-header members

Member	Type	Required	Description
`name`	Enumeration	✓	Set to `modify-outgoing-request-header` to modify an outgoing request header.
`params`	See the `[modify-outgoing-request-header.params]` array	✓	Use these members to provide attributes to modify outgoing request headers.
`type`	Enumeration	✓	Specifies the type of modification to perform if the request meets the criteria set in the rule's match. Set this to `append` to add a given header value to a header name set in the `headerList`. Set this to `delete` to remove a given header value from a header name set in the `headerList`. Set this to `overwrite` to match on a specified header name and replace its existing header value with a new one you specify. Caveats apply to each `type`. . See *Usage caveats*, below.
`value`	String	✓	This is ignored for this behavior, but required for consistency in the API. Set the value to a dash (`-`).
`modify-outgoing-request-header.params`: Use these members to provide attributes to modify outgoing request headers.			

Member	Type	Required	Description
`delimiter`	Enumeratio n	o	Specifies the delimiter to be used when indicating multiple values for a header. Set this to a `<space>`, `,` (comma), `;` (semicolon), `,<space>` (comma and space), or `;<space>` (semicolon and space). Delimiters are only supported for use with `append` and `delete`. Caveats apply to `delimiter` use. This defaults to a `,` if nothing is set. See *Usage caveats*, below.
`headerList`	Array	✓	A collection of key value pairs that specify the headers and associated values to be modified. The key sets the header name to be modified and the value is the header value to be appended, deleted or overwritten. Various caveats apply to header names and values. See *Usage caveats*, below.

Usage caveats

Consider the following usage caveats before applying this behavior.

- You can modify a maximum of 15 request headers with this behavior, in a single policy.
- You can use each behavior `type` (`append`, `delete`, and `overwrite`) *once* in a single rule.
- You can't modify the same header more than once in a single policy.
- You can use multiple `modify-outgoing-request-header` behaviors in the same rule. Ensure they don't conflict, based on other caveats in this list. (This only applies to this behavior.)
- You can't use a CRLF or a colon in header names or header values.
- The `append` and `overwrite` behavior types create a non-existent header and apply the specified header value.
- The `append` behavior type won't append a header value if it already exists in the header. If duplicate instances of a header value are identified, they are left as is because the `append` type only adds content, it does not remove anything. Use the `overwrite` type in this scenario to completely replace the header content, and remove duplicate values.
- If multiple occurrences of a single header are included in a request, an `append` or `delete` behavior type only applies to the first occurrence. That header is modified and sent to the origin, and all other instances of the same header in the request are ignored. If you don't include the `modify-outgoing-request-header`, *all* occurrences of a request header are sent to the origin.
- The `delete` behavior type can be used to remove an empty header. (If it has no header value.) Just include the header name, and leave the header value empty (for example, `"my_header": ""`).
- The `delete` behavior type only removes a value from a header if that value is found. If the value isn't found, it does nothing.
- If the `delete` behavior type removes the only header value, older origin servers may experience issues. (This can result in a "null pointer exception" on an older origin server not built to handle empty headers.)

- When delimiter-separating multiple header values in a `delete`, values are deleted only if they exist in the order specified. For example, if the policy is set to match and delete `"header1": "value1,value3"`, and the actual request header is `"header1: value1,value2,value3"`, nothing is deleted, because the exact match, `"value1,value3"` wasn't found. However, if the actual request header is `"header1: value2,value1,value3"`, `value1` and `value3` are deleted because they match the order set.
- The `overwrite` behavior type overwrites *all* header value information for a header name it matches.
- If you don't include the `delimiter` object, it defaults to a comma.
- If you use a delimiter to include a list of header values, it must match what's set for the `delimiter` member (or a comma if you didn't set one.) Otherwise, the API matches on exactly what's set.
- If you include a `delimiter` when using the `overwrite` behavior type, Akamai Cloud Embed ignores it.
- Header names are *case-insensitive* when matching eligible headers, for all behavior types.
- Header values are *case-sensitive* when matching eligible headers for `append` and `delete`. (Header values are *replaced* for `overwrite`, so no matching is performed.)
- Since header names are case-insensitive, the header name you set in a policy is what's used as the header name once modified.

Blacklisted request headers

These headers can't be modified. If the `type` criteria set matches one of these headers, an error is revealed. *Both* header names and values are case-insensitive when matching on blacklisted headers. The `*` represents a wildcard.

- `Connection`
- `Content-Length`
- `Forwarded`
- `Host`
- `TE`
- `Upgrade`
- `x-akamai*`
- `x-cache*`

Blacklisted header values

You can't modify these items in a header value.

- `akamai-x-*`

Known issue: appending multiple header values and duplicate, existing values

When you use the `append` behavior type and include multiple header values, if a header value already exists, this behavior duplicates it in the resulting header.

For example, assume you have the existing header, `"header1": "value1;value2"` and you set this behavior to append: `"header1": "value1;value3;value4"`. This results in `"header1: value1;value2;value1;value3;value4"`. This is because the API is trying to match on the *full value* you've specified, `"value1;value3;value4"` when checking for duplicates.

As a best practice, you should only include multiple header values for an `append` if you're sure one of those values doesn't already exist.

The "modify-outgoing-request-path" behavior

Include this behavior to provide options for altering the request URL before it is sent to origin.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "modify-outgoing-request-path",
            "type": "replace",
            "value": "/dir1/dir2/###/dir4/"
        }
    ]
}
```

modify-outgoing-request-path members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `modify-outgoing-request-path` to alter the request URL before it is sent to origin.
`type`	Enumeration	✓	Set the type of request path change to use. Enter `remove` to remove the first occurrence of the `value` from the outgoing request URL; `replace-all` to replace the original request path before the filename with the path specified in `value`; or `replace` to search for a portion of the original request URL and replace it with a value you define.
`value`	String	✓	If using `remove`, enter the string of characters to remove from the forward request. The behavior doesn't remove leading and trailing slashes. If using `replace-all`, enter the URL path to replace the original incoming URL path. If using `replace`, enter a value in this format: `/find/path/###/replace/ path/`, where `###` separates the string to find and the string to replace. For example, if your base URL is `www.example.com/dir1/dir2/dir3/index.html` and you enter `/dir1/dir2/###/dir4/` as the `value`, the resulting URL is `www.example.com/dir4/dir3/index.html`.

The "modify-outgoing-response-header" behavior

Include this behavior to modify the outgoing response headers sent from the Edge server back to the client.

Samples

Here's a sample of this behavior in a policy to append the header value, `new_value1` to the outgoing response header, `header1`; append the values `newer_value1` and `newest_value1` to the outgoing response header, `header2`; and set the semicolon (`;`) as the `delimiter`:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "modify-outgoing-response-header",
            "type": "append",
            "value": "-",
            "params": {
                "headerList": [
                    {
                        "header1": "new_value1"
                    },
                    {
                        "header2": "newer_value1;newest_value1"
                    }
                ],
                "delimiter": ";"
            }
        }
    ]
}
```

This is a sample of this behavior in a policy to delete the empty header, `header1`; and delete the header values, `old_value`, `older_value`, and `oldest_value` from the outgoing response header, `header2` (The behavior only deletes these values if they exist in `header2` in the order specified. See *Usage caveats*, below):

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "modify-outgoing-response-header",
            "type": "delete",
            "value": "-",
            "params": {
                "headerList": [
                    {
                        "header1": ""
                    },
                    {
                        "header2": "old_value,older_value,oldest_value"
                    }
                ],
                "delimiter": ","
            }
        }
    ]
}
```

Finally, here's a sample of this behavior in a policy to match the header, `header1` and overwrite its header value with `use_this_instead` (with no `delimiter`, because an `overwrite` doesn't use one):

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "modify-outgoing-response-header",
            "type": "overwrite",
            "value": "-",
            "params": {
                "headerList": [
                    {
                        "header1": "use_this_instead"
                    }
                ]
            }
        }
    ]
}
```

modify-outgoing-response-header members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Set to `modify-outgoing-response-header` to modify an outgoing response header.
`params`	See the `[modify-outgoing-response-header.params]` array	✓	Use these members to provide attributes to modify outgoing response headers.
`type`	Enumeratio n	✓	Specifies the type of modification to perform if the response meets the criteria set in the rule's match. Set this to `append` to add a given header value to a header name set in the `headerList`. Set this to `delete` to remove a given header value from a header name set in the `headerList`. Set this to `overwrite` to match on a specified header name and replace its existing header value with a new one you specify. Caveats apply to each `type`. See *Usage caveats*, below.
`value`	String	✓	This is ignored for this behavior, but required for consistency in the API. Set the value to a dash (`-`).
`modify-outgoing-response-header.params`: Use these members to provide attributes to modify outgoing response headers.			
`delimiter`	Enumeratio n	○	Specifies the delimiter to be used when indicating multiple values for a header. Set this to a

Member	Type	Required	Description
			`<space>`, `,` (comma), `;` (semicolon), `,<space>` (comma and space), or `;<space>` (semicolon and space). Delimiters are only supported for use with `append` and `delete`. Caveats apply to `delimiter` use. This defaults to a `,` if nothing is set. See *Usage caveats*, below.
`headerList`	Array	✓	A collection of key value pairs that specify the headers and associated values to be modified. The key sets the header name to be modified and the value is the header value to be appended, deleted or overwritten. Various caveats apply to header names and values. See *Usage caveats*, below.

Usage caveats

Consider the following usage caveats before applying this behavior.

- You can modify a maximum of 15 response headers with this behavior, in a single policy.
- You can use each behavior `type` (`append`, `delete`, and `overwrite`) *once* in a single rule.
- You can't modify the same header more than once in a single policy.
- You can't use a CRLF or a colon in header names or header values.
- The `append` and `overwrite` behavior types create a non-existent header and apply the specified header value.
- The `append` behavior type won't append a header value if it already exists in the header. If duplicate instances of a header value are identified, they are left as is because the `append` type only adds content, it does not remove anything. Use the `overwrite` type in this scenario to completely replace the header content, and remove duplicate values.
- If multiple occurrences of a single header are included in a request, an `append` or `delete` behavior type only applies to the first occurrence. That header is modified and sent to the origin, and all other instances of the same header in the request are ignored. If you don't include the `modify-outgoing-response-header`, *all* occurrences of a request header are sent to the origin.
- The `delete` behavior type can be used to remove an empty header. (If it has no header value.) Just include the header name, and leave the header value empty (for example, `"my_header": ""`)
- The `delete` behavior type only removes a value from a header if that value is found. If the value isn't found, it does nothing.
- If the `delete` behavior type removes the only header value, older origin servers may experience issues. (This can result in a "null pointer exception" on an older origin server not built to handle empty headers.)
- When delimiter-separating multiple header values in a `delete`, values are deleted only if they exist in the order specified. For example, if the policy is set to match and delete `"header1": "value1,value3"`, and the actual request header is `"header1: value1,value2,value3"`, nothing is deleted, because the exact match, `"value1,value3"` wasn't found. However, if the

actual request header is `"header1: value2,value1,value3"`, `value1` and `value3` are deleted because they match the order set.

- The `overwrite` behavior overwrites *all* header value information for a header name it matches.
- If you don't include the `delimiter` object, it defaults to a comma.
- If you use a delimiter to include a list of header values, it must match what's set for the `delimiter` member (or you must use commas if you didn't set one.) Otherwise, the API matches on exactly what's set.
- If you include a `delimiter` when using the `overwrite` behavior type, Akamai Cloud Embed ignores it.
- Header names are *case-insensitive* when matching eligible headers, for all behavior types.
- Header values are *case-sensitive* when matching eligible headers for `append` and `delete`. (Header values are *replaced* for `overwrite`, so no matching is performed.)
- Since header names are case-insensitive, the header name you set in a policy is what's used as the header name once modified.

Blacklisted response headers

These headers can't be modified. If the `type` criteria set matches one of these headers, an error is revealed. *Both* header names and values are case-insensitive when matching on blacklisted headers. The `*` represents a wildcard.

- `Age`
- `Alt-Svc`
- `Connection`
- `Content-Encoding`
- `Content-Length`
- `Content-Range`
- `Transfer-Encoding`
- `Vary`
- `x-akamai*`
- `x-cache*`

Blacklisted header values

You can't modify the following items when they occur in a header value.

- `akamai-x-*`

Known issue: appending multiple header values and duplicate, existing values

When you use the `append` behavior type and include multiple header values, if a header value already exists, this behavior duplicates it in the resulting header.

For example, assume you have the existing header, `"header1": "value1;value2"` and you set this behavior to append: `"header1": "value1;value3;value4"`. This results in `"header1: value1;value2;value1;value3;value4"`. This is because the API is trying to match on the *full value* you've specified, `"value1;value3;value4"` when checking for duplicates.

As a best practice, you should only include multiple header values for an `append` if you're sure one of those values doesn't already exist.

The "origin" behavior

Include this behavior to provide origin settings for the specific subcustomer.

You need to include: origin DNS hostname, forward `host` header, and cache key. Optional settings include the origin base path and ports.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "origin",
            "value": "-",
            "params": {
                "digitalProperty": "www.mysite.com",
                "originDomain": "c1234567.cloudprovider.com",
                "cacheKeyType": "origin",
                "cacheKeyValue": "-",
                "hostHeaderType": "digital_property",
                "hostHeaderValue": "-"
            }
        }
    ]
}
```

origin members

Member	Type	Required	Description
`name`	Enumeratio n	○	Enter `origin` to set up the origin behavior. Akamai Cloud Embed requires an origin in every content policy.
`params`	See the `[origin.p arams]` array	○	Origin settings for the policy.
`value`	String	○	Enter a dash (-) for this parameter.
`origin.params`: Origin settings for the policy.			

Member	Type	Required	Description
`cacheKeyType`	Enumeration	○	Enter the method to use when constructing the cache key. Use `digital_property` if the response is different for each property, `origin` if the origin response is the same regardless of property, or `fixed` to set a specific cache key value.
`cacheKeyValue`	String	○	If `cacheKeyType` is `fixed`, enter a valid domain name for the hostname portion of the cache key. Use a dash (`-`) to indicate no value.
`digitalProperty`	String	○	Enter the hostname used by the client to access the site or application. Use a valid domain name like `www.example.com` or `blogs.example.com`.
`hostHeaderType`	Enumeration	○	For requests sent to this origin, enter the `host` header value to generate. Use `digital_property` to have the `host` header match the digital property, `origin` to use the `originDomain` value as the `host` header, and `fixed` to use the `hostHeaderValue` as the `host` header.
`hostHeaderValue`	String	○	If `hostHeaderType` is `fixed` enter a valid domain name to use in the `host` header. Use a dash (`-`) to indicate no value.
`originDomain`	String	○	Enter the origin hostname you provide to subcustomers when you provision services. Your entry can either be a valid domain name or an IP address.

The "origin-characteristics" behavior

Include this behavior if you have Integrated Cloud Acceleration (ICA), to select the type of origin supporting your Akamai Cloud Embed implementation.

Use the *origin* behavior to configure origin settings for your subcustomers, at the policy level.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "origin-characteristics",
            "value": "azure"
        }
    ]
}
```

origin-characteristics members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Use the `origin-characteristics` behavior to select the type of origin you are using for your Akamai Cloud Embed implementation.
`type`	Enumeratio n	✓	The type of origin supporting your implementation. Enter `azure` if you use an Azure Media Services live origin. Use `unknown` for all other origins.

The "origin-failover" behavior

Origin failover identifies primary origin connection failures based on a type you specify and marks that origin as “bad” after connections to all its IPs fail repeatedly. Rather than issuing a redirect to the end user, requests are failed over to a backup origin you call out.

This improves response times, because the end user doesn’t have to wait several seconds for a connect-timeout on the forward request. Additionally, you specify a duration of time the primary origin is marked as bad. During this time, all requests are failed over to your backup origin. This relieves pressure on the primary by reducing the number of connection attempts, at a time when it appears to be having difficulties.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "origin-failover",
            "type": "backupOrigin",
            "value": "-",
            "params": {
                "errorType": "timeout",
                "originConnectionTimeout": "20s",
                "errorCountBeforeFailover": 10,
                "timeoutErrorCacheDuration": "2h",
                "backupOrigin": "backup.myorigin.com",
                "customForwardHost": "somebackuporigin.com"
            }
        }
    ]
}
```

origin-failover members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Set to `origin-failover` to add this to your policy to configure offload of failover detection and recovery.
`params`	See the `[origin-failover.`	✓	These parameters are used to define your backup origin as well as what’s used to mark an origin as bad, in order to failover requests to that backup origin.

Member	Type	Required	Description
	params] array		
type	String	✓	Specifies the type of failover to use if the request meets the criteria set in the rule's match. Set this to backupOrigin to failover to a backup origin.
value	String	✓	This is ignored for this behavior, but required for consistency in the API. Set the value to a dash (-).
origin-failover.params: These parameters are used to define your backup origin as well as what's used to mark an origin as bad, in order to failover requests to that backup origin.			
backupOrigin	String	✓	The domain name (the complete path and hostname) or IP address of the origin serving as your backup for failover.
customForwardHost	String	○	Include this to customize what's included in the X-Forwarded-Host header, when a failover directs the request to a backupOrigin. (This header typically identifies the original host requested by the client in the Host HTTP request header.)
errorCountBeforeFailover	Integer	○	This tells Akamai how many errors of the selected errorType, timeout to recognize before an origin is considered bad, and failover is applied. As a best practice, set this to a value higher than 1. An origin shouldn't be considered bad after only a single failure. Note that the default for this is 0. So, if you don't set a value, all error requests are failed over and the first failed request marks the origin as bad. There are additional caveats that apply to this member, specifically when a request is actually failed over. See *Usage caveats*, below.
errorType	Enumeration	✓	This defines the type of error that occurs to mark the connection as a failure. Set this to timeout to indicate a failure once an origin connection request times out after a specific amount of time. Caveats apply to the use of timeout. (Currently, only timeout is supported. You can expect additional errorType values with a future release.) See *Usage caveats*, below.
originConnectionTimeout	String	○	The amount of time that constitutes an origin connection timeout and results in a failure. If you leave this parameter out, the connection timeout is default set to five seconds.
timeoutErrorCacheDuration	String	○	Set an amount of time that needs to pass before a request retries an origin IP that was flagged as bad, as a result of a timeout error.

Usage caveats

Consider the following usage caveats before applying this behavior.

- Failover to your specified `backupOrigin` happens under two conditions: when a `timeout` request error occurs; or when an origin receives a request after ACE has marked it as bad,because the `errorCountBeforeFailover` count has been reached. (Each `timeout` request error counts toward this total.) Marking an origin as bad stops connection attempts to it while it may be having trouble.
- This behavior has two primary functions: To failover error requests to a `backupOrigin` you specify, and to mark an origin as "bad" after a certain number of these error requests occur, to stop further requests to a potentially origin when it may be having trouble.
- The Akamai server tries a failed connection one more time before counting it towards your `errorCountBeforeFailover`.
- The `errorCountBeforeFailover` member marks an origin as bad for the request that occurs *after* the quantity you've specified, and enacts the `timeoutErrorCacheDuration` to stop requests from targeting the origin for that length of time. Each subsequent request is then failed over to the `backupOrigin`. For example, if you set this to six, the seventh failed request triggers the `timeoutErrorCacheDuration` and the eighth and subsequent requests are failed over to your `backupOrigin`.
- When used with the *origin behavior* in a policy, set its `cacheKeyType` member to `origin`. This keeps cache key creation for your primary origin consistent with the Origin Failover behavior cache key creation for your backup origin.
- During failover to your backup origin, the cache key is modified by adding a "`b-`" in the path. This avoids cache sharing and negative Time to Live (TTL) settings that may use what is in cache instead of accessing your specified backup origin. For example, assume a request is sent to this origin that has been as marked as bad: `/prodtest-ff.qa.akamai.com.partnerdomain.net/OD/bird.jpg` The request is rerouted to this backup origin and `b-` is added to the origin hostname: `/b-mde-origin.qa.akamai.com.partnerdomain.net/OD/bird.jpg`
- Duration values allow for settings in days (d), hours (h), minutes (m), seconds (s), and milliseconds (ms). All default values are applied in seconds.
- If you've enabled Tiered Distribution in your base configuration, origin failover is only valid for the top-tier level. (This represents the "last hop" to your origin to request content.)

The "referer-blacklist" behavior

Include this behavior to *block* access based on the `Referer` request header.

This behavior helps verify that the client is a browser that supports *RFC 2616*, section 14.36, and that the referring HTML page is served by Akamai Cloud Embed from a domain trusted by the content owner.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "referer-blacklist",
            "value": "/dir1/dir2/###/dir4/"
        }
    ]
}
```

referer-blacklist members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `referer-blacklist` to block access based on the Referer request header.
`value`	String	✓	Enter a space-separated list of Referer header values to disallow. Use the `*` wildcard to match on a substring within the header value.

The "referer-whitelist" behavior

Include this behavior to *allow* access based on the `Referer` request header.

This behavior helps verify that the client is a browser that supports *RFC 2616*, section 14.36, and that the referring HTML page is served from a domain trusted by the content owner.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "referer-whitelist",
            "value": "www.mysite.com www.myothersite.com*"
        }
    ]
}
```

referer-whitelist members

Member	Type	Required	Description
`name`	Enumeratio n	✓	Enter `referer-whitelist` to allow access based on the Referer request header.
`value`	String	✓	Enter a space-separated list of Referer header values to allow. Use the `*` wildcard to match on a substring within the header value.

The "site-failover" behavior

Include this behavior to define the alternate hostname and path to use when an Akamai edge server can't contact your origin server.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "site-failover",
            "type": "serve-301",
            "params": {
                "httpResponseStatus": "404 500:504",
                "alternateHostname": "www.mysite_failover.com",
                "alternatePath": "/failover/mysite",
                "preserveQueryString": true
            }
        }
    ]
}
```

site-failover members

Member	Type	Required	Description
`name`	Enumeration	✓	Enter `site-failover` to define an alternate response to serve when the edge server can't contact the origin server.
`params`	See the `[site-failover.params]` array	○	These are failover settings.
`type`	Enumeration	✓	Select the failover action to use. Enter `serve-301` for 301 redirects, `serve-302` for 302 redirects, or `serve-alternate` to send a request to an alternate hostname and path.
`site-failover.params`: These are your failover settings.			
`alternateHostname`	String	○	Enter the domain of the hostname to failover to. Enter a dash (`-`) to use the original hostname when constructing the new URL.
`alternatePath`	String	○	Enter a valid URI path to failover to. Always include the initial slash. Include the closing slash to change the URL path but keep the original filename. Enter a dash (`-`) to use the original path when constructing the new URL.
`httpResponseStatus`	String	○	Enter a space-separated list of the HTTP status codes served to the client when `site-failover` is not in effect. You can use integer ranges (for example, `404`, `500:504`).
`preserveQueryString`	Boolean	○	Enter `true` to retain the query string from the original request URL, or enter `false` to remove the query string.

The "token-auth" behavior

Include this behavior to use tokens to control access to content. You can choose to transmit the token in a cookie, header, or query parameter.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "token-auth",
            "value": "-",
            "type": "serve-301",
            "params": {
                "tokenName": "__mytoken__",
                "tokenDelimiter": "~",
                "aclDelimiter": "!",
                "hmacAlgorithm": "SHA256",
                "escapeTokenInputs": true,
                "ignoreQueryString": false,
                "key": "adf123",
                "transitionKey": "bce987",
                "salt": "worldnews175892"
            }
        }
    ]
}
```

token-auth members

Member	Type	Required	Description
`name`	Enumeration	✓	Set to `token-auth` to use tokens to control access to content.
`params`	See the `[token-auth.params]` array	○	Use these parameters to define how tokens are used by this behavior.
`value`	String	○	This is ignored for this match type, but required for consistency in the API. Set the value to a dash (`-`).
`token-auth.params`: Use these parameters to define how tokens are used by this behavior.			
`aclDelimiter`	String	○	Specifies a single character to separate access control list (ACL) fields. You can't use alphanumeric characters, or any of the following character class: [.&`=/:%]. If you don't specify a value, `!` is used as the delimiter.
`escapeTokenInputs`	Boolean	○	Set to `true` input values are escaped before adding them to the token.

Member	Type	Required	Description
hmacAlgorithm	Enumeration	○	Specifies the algorithm to use for the token's hash-based message authentication code (HMAC) field. Valid entries are SHA256, SHA1, or MD5.
ignoreQueryString	Boolean	○	Set to true query strings are removed from a URL when computing the token's HMAC algorithm.
key	String	○	Specifies an even number of hex digits for the token key. An entry can be up to 64 characters in length.
salt	String	○	Specifies a salt for use in the token. This can be a maximum of 250 characters, and it can't use characters from the following class: `[~@#%^&=/|\]`.
tokenDelimiter	String	○	Specifies a single character to separate the individual token fields. You can't use characters from the following class: `[a-zA-Z0-9=&/\:%]`. If you don't specify a value, ~ is used as the delimiter.
tokenName	String	○	Specifies the name of the token. Match the string on the following regular expression: `^([a-zA-Z_][a-zA-Z0-9-_]*)$`.
transitionKey	String	○	Specifies an even number of hex digits for the token transition key. An entry can be up to 64 characters in length.

The "url-redirect" behavior

Include this behavior to configure redirect responses for specific client requests, and stop them from contacting the origin.

Sample

Here's a sample inclusion for this behavior in a PUT policy operation:

```
{
    "matches": [],
    "behaviors": [
        {
            "name": "url-redirect",
            "type": "301",
            "value": "-",
            "params": {
                "protocol": "request",
                "hostname": "request",
                "staticHostname": "www.somedomain.com",
                "subDomain": "music",
                "path": "request",
                "staticPath": "/abc",
                "pathPrefix": "/pre",
                "pathSuffix": "/su",
                "includeQueryString": true
            }
        }
    ]
}
```

url-redirect members

Member	Type	Required	Description
`name`	Enumeration	✓	Set `url-redirect` to define the URL used for a redirect client request.
`params`	See the `[url-redirect.params]` array	✓	These members are used to customize the URL used for the redirect.
`type`	Integer	✓	Specifies the type of redirect sent to the client if the request meets the criteria set in the rule's match. You can choose from `301` moved permanently, `302` found, `303` see other, or `307` temporary redirect.
`value`	String	✓	This is ignored for this behavior, but required for consistency in the API. Set the value to a dash (`-`).
`url-redirect.params`: These members are used to customize the URL used for the redirect.			
`hostname`	Enumeration	✓	This defines the destination hostname that should be used in the redirect URL. Allowed values include: `static` to apply a unique `staticPath`, `addSubDomain` to preface the hostname with a `subDomain` in the redirect request, `replaceSubDomain` to replace an existing `subDomain` value with one you specify, or `request` to use a subdomain already set in the client request.
`includeQueryString`	Boolean	○	When enabled, retains the query string on the redirect request. Defaults to `true`.
`path`	Enumeration	✓	This defines how the redirect URL is constructed. Allowed values include: `static` to apply a fixed `staticPath`, `addPrefix` to preface the path in the client request with a `pathPrefix`, `addSuffix` to append a `pathSuffix` to the path in the client request, or `request` to only use the path provided in the client request.
`pathIncludesFilename`	Boolean	○	Include if `path` is set to `staticPath`. Indicates whether the `path` also includes the filename and extension for the redirect. Defaults to `false`.
`pathPrefix`	String	○	Include if `path` is set to `addPrefix`. Specifies the prefix to add to the path in a redirect request. For example, to redirect from the path `/example.html` to `/baseball/example.html`,

Member	Type	Required	Description
			set this to `/baseball`. You can't use a single `/` to specify this path. A valid path is a `/` followed by at least one character.
`pathSuffix`	String	○	Include if `path` is set to `addSuffix`. The suffix to be added to the path in a redirect request. For example, to redirect from the path `/example/index.html` to `/example/football/index.html`, set this to `/football`. You can't use a single `/` to specify this path. A valid path is a `/` followed by at least one character.
`protocol`	Enumeration	✓	This defines the destination protocol to be used for the redirect URL. Allowed values include: `HTTP` to set the request to non-secure, `HTTPS` to set it to secure, or `request` to use the protocol set in the client request.
`staticHostname`	String	○	Include if `hostname` is set to `static`. Set this to the desired hostname for use in the redirect URL.
`staticPath`	String	○	Include if `path` is set to `staticPath`. Set this to a static path that should be used in the redirect URL. A valid path is a `/` followed by at least one character.
`subDomain`	String	○	Include if `hostname` is set to `addSubDomain` or `replaceSubDomain`. If set to `addSubDomain`, specify the subdomain to preface the existing domain. For example, to redirect from `example.com` to `www.m.example.com` set this value to `www`. If set to `replaceSubDomain`, set a value to replace an existing subdomain. Include a subdomain in the client request hostname, and only the first subdomain is replaced. For example, to change a redirect URL subdomain from `domain.example.com` or www.example.com to `newdomain.example.com`, set the value to `newdomain`.

Errors

This section shows you how to handle various kinds of errors this API generates, and lists the range of HTTP status codes along with their likely causes.

Error responses

The following example shows an error response for a request processed by this API:

```
{
    "errorCode" : 403,
    "message" : "Authorization failed",
    "description" : "The user is not authorized by Akamai Cloud Embed to
access the policies or subcustomers assigned to the given property id.",
    "helpLink" : "https://developer.akamai.com/api/delivery-policies/
errors.html#403",
    "errorInstanceId" : "31f1a7532f",
}
```

You would expect to see a message like this if the property ID isn't valid or is not configured correctly.

HTTP status codes

This section lists the full range of response codes the API may generate.

Code	Description
200	The operation was successful.
201	Resource successfully created.
202	Resource successfully accepted.
400	Bad Request.
401	Authentication failure.
403	Access is forbidden.
404	Resource not found.
408	Request timeout.
500	Internal server error.
503	Too many requests. Service is temporarily unavailable.

What's new with Cloud Embed

Below are details on the various releases for Cloud Embed and what we've added.

2019-11-07

New Features:

- **Akamai Cloud Embed (ACE) map expansion**. This lets us use the Akamai edge map without restrictions and we expect it will improve performance for subcustomers. ACE policy activations now take longer to complete (eight minutes versus one minute), but this update provides marked benefits for safety and performance (improvement in throughput, latency, and time to first byte (TTFB)).
- **Live delivery optimizations**. This feature provides optimizations for live streaming to ACE partners and their customers. This is available as a new option, “streaming-video-live” in the existing “content-characteristics” behavior.
- **Prefetching of streaming content**. Prefetching positions target media content at the Edge in anticipation of requests by end users. This reduces the time to deliver that content. The origin housing content needs to be configured to support the origin-assist model, and prefetching is supported for use with DASH, HLS, and Smooth streaming formats.
- **Origin failover**. This feature identifies primary origin connection failures based on a type you specify and marks that origin as “bad” after connections to all its IPs fail repeatedly. Rather than issuing a redirect to the end user, requests are failed over to a backup origin you call out.

Enhancements

- **HTTPS orchestration**. We’ve made improvements to certification selection logic.

2019-05-24

New Features:

We have added support for new behaviors in a subcustomer policy:

- **Request Header Rewrite**: Use this behavior to modify the outgoing request headers sent from Akamai to an origin. This also works on request headers sent from a client if the request is sent back to the origin, but not a cache hit.
- **Response Header Rewrite**: Use this behavior to modify the outgoing response headers sent from the origin or CDN edge back to the client.

2019-03-30

New Features:

We have added support for a new behavior in a subcustomer policy:

- **URL Redirect**: You can include this behavior to configure redirect responses for specific client requests, and stop them from contacting the origin.

2019-03-13

New features:

You can associate a Vanity Domain with a different Partner Domain. To accomplish this you need to do the following:

1. Disable HTTPS for the Vanity Domain. This ensures that the vanity domain is successfully de-linked from the current or old partner domain.

2. Enable HTTPS for the Vanity Domain with the new Partner domain.

A request to enable HTTPS on a Vanity Domain is rejected with a status “HTTP 409 - Request Conflict” in the following scenarios:

- If the Vanity Domain is HTTPS enabled with the old Partner Domain.

- If there are pending changes against the old Partner Domain.

Bug fixes:

- PUT requests for `/secure-delivery` that have a non-empty endpoint domain are rejected with a status “HTTP 400” and an associated message.

2019-01-17

New Features:

We have added support for new rule match criteria in a subcustomer policy:

- **Geography**: You can test the requesting client’s location, either by continent, country, region, or designated market area (DMA).

2018-11-23

New Features

We have added support for new rule match criteria in a subcustomer policy:

- **Client IP match**: You can specify an individual or a range of IP addresses. If a request originates from a specified address, the behavior applied in a policy rule is applied.

- **Cookie match**: You can specify a specific cookie name, and optionally include associated cookie values. If a request for content uses the named cookie (and optionally, the cookie values), the behavior applied in a policy rule is applied.

Notice

Akamai secures and delivers digital experiences for the world’s largest companies. Akamai’s Intelligent Edge Platform surrounds everything, from the enterprise to the cloud, so customers and their businesses can be fast, smart, and secure. Top brands globally rely on Akamai to help them realize competitive advantage through agile solutions that extend the power of their multi-cloud architectures. Akamai keeps decisions, apps, and experiences closer to users than anyone — and attacks and threats far away. Akamai’s portfolio of edge security, web and mobile performance, enterprise access, and video delivery solutions is supported by unmatched customer service, analytics, and 24/7/365 monitoring. To learn why the world’s top brands trust Akamai, visit *www.akamai.com*, *blogs.akamai.com*, or *@Akamai* on Twitter. You can find our global contact information at *www.akamai.com/locations*.

Akamai is headquartered in Cambridge, Massachusetts in the United States with operations in more than 57 offices around the world. Our services and renowned customer care are designed to enable businesses to provide an unparalleled Internet experience for their customers worldwide. Addresses, phone numbers, and contact information for all locations are listed on *www.akamai.com/locations*.

Published 11/2021