

API Property: Best Practices

Table of Contents

Akamai API Gateway.....2

Standalone API.....3

What to Configure.....3

Caching.....3

GraphQL Caching.....3

Zone Apex Mapping.....4

Compression.....4

Monitor.....5

Disable Response Buffer.....5

SureRoute.....6

Cloud Origin.....7

What Not to Configure.....8

Real User Monitoring.....8

Adaptive Acceleration.....9

Content Prefetching.....9

Front-End Optimization.....10

Image Optimization.....10

API as a Resource.....11

Scenario: One Domain.....11

Scenario: Multi-Domain.....11

What to Configure.....12

HTTP/2.....12

Image Optimization.....12

Prefetching.....12

Other.....13

API Property: Best Practices

This white paper will not explain how to set up a Property Manager configuration ([this can be found here](#)), but will provide some best practices to follow. We cover things that should and should not be done when working with an API (from an Akamai perspective).

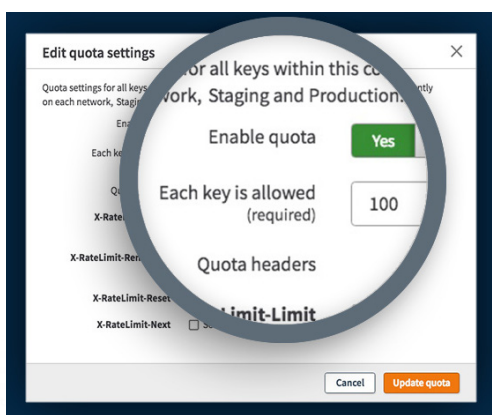
Going forward, we will be referring to two use cases:

- **Standalone API** (text-based API, like [Akamai API](#))
- **API as a resource** of websites

WHAT TO CONFIGURE	WHAT NOT TO CONFIGURE
<ul style="list-style-type: none"> • Caching • Zone Apex Mapping • Compression • SureRoute • Monitor (DataStream/Cloud Monitor/DLR) • Disable Response Buffer • Cloud Origin 	<ul style="list-style-type: none"> • Adaptive Acceleration • Image Optimization • Akamai RUM Injection (mPulse/Legacy) • Content Prefetch • Front-End Optimization

Akamai API Gateway

When using the [Akamai API Gateway](#), some of the recommendations will overlap with the features that are managed by it.



- Caching
- GraphQL Caching
- CORS
- Compression
- Error Response Customization
- Privacy Setting
- Quota Management
- JSON Web Token

Any feature in the list has to be managed by API Gateway since it will override anything set up within the Property Manager. All other recommendations mentioned are still applicable and should be taken into consideration.

Standalone API

What to Configure

Caching

Akamai can store a copy of your content so that the edge servers don't have to continually call the application/origin servers to deliver content to your site visitors. This increases performance for your users, and reduces load and bandwidth on your origin web server.

Things to keep in mind:

- Always set the default caching setting to "No-Store" to avoid unnecessary caching and avoid caching pollution
- Only cache specific and known scenarios
- CacheKey: By default, we include query strings (QS) in the cache key used to store the object

Order matters: The order of how we store the QS in the cache matters. If the application does not care about the order of the QS (meaning the content will not change if the order changes), then ordering the QS will allow us to maximize offload by reducing the possible variants of the cache key for the same object.

Include only the QS you need: This ensures the uniqueness of the object. By removing QS that aren't relevant to the object returned, objects can be served from cache for more scenarios.

Include headers/cookies: Like QS, this will ensure that we maximize offload while reducing the risk of cache pollution by adding headers or cookies that affect the object returned by the application/origin.

GraphQL Caching

We can detect GraphQL errors and uncacheable content (mutations and subscriptions). It also provides an increased cache hit rate and performance for GraphQL-based open APIs.

If the application uses GraphQL, we can add the GraphQL caching behavior to a property to control how Akamai handles caching.

Property Manager UI

API JSON Snippet

```
{
  "name": "graphqlCaching",
  "options": {
    "enabled": true,
    "cacheResponsesWithErrors": false,
    "advanced": "",
    "postRequestProcessingErrorHandling": "NO_STORE",
    "operationsUriQueryParameterName": "query",
    "operationsJsonBodyParameterName": "query"
  }
}
```

API Property: Best Practices

Zone Apex Mapping

Zone apex mapping uses the mapping data available on the Akamai platform to reduce DNS lookup times for your websites.

With zone apex mapping, name servers resolve DNS lookup requests with the IP address of an optimal edge server.

FastDNS zone apex benefits:

- Eliminate the CNAME chains inherent with CDN solutions
- Reduce DNS lookup times for any website on the Akamai platform
- Deploy Akamai acceleration solutions for records at the zone apex for which a CNAME cannot otherwise be used

Compression

As content travels from the edge server to users, the final leg of content delivery tends to be slower, particularly for users who are browsing on mobile devices.

Enabling compression saves time in content delivery transit, which helps with performance and the total volume of data transmitted, as well as associated costs. Ensuring that we have it enabled – and that it covers all MIME/content types that the application uses – is crucial for the success of the feature.

Features to enable:

- Last Mile Acceleration
- Brotli from Origin (if origin supports it)

Property Manager UI

Content Compression

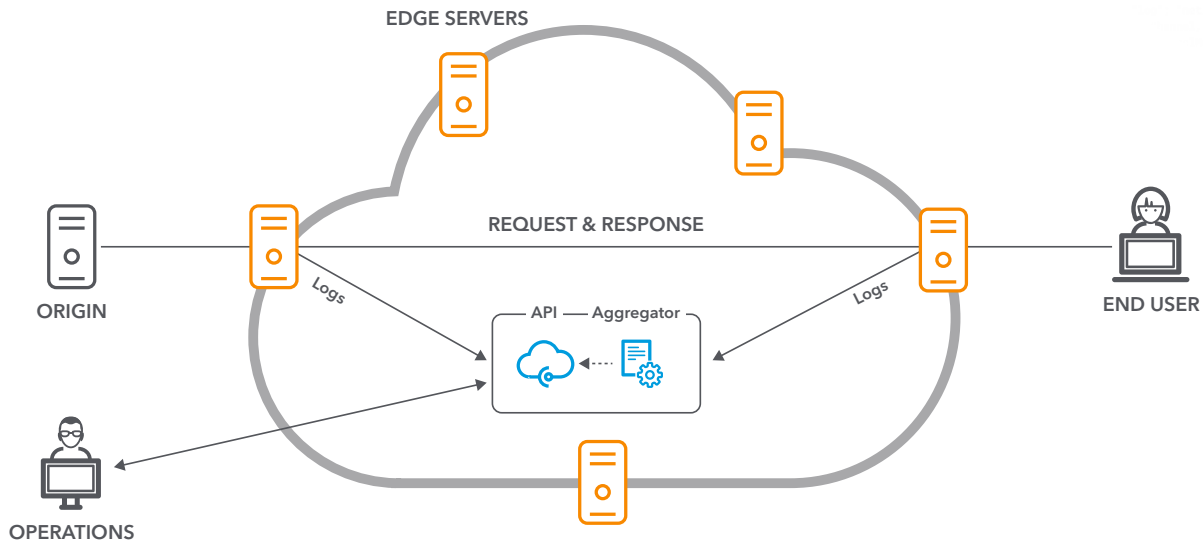
The screenshot shows the Akamai Property Manager interface for configuring Content Compression. At the top, there's a text input field labeled "Add a comment...". Below it, the "Criteria" section has a dropdown menu set to "Match All" and an "Add Match" button. The "Behaviors" section includes an "Add Behavior" button and a configuration box for "Last Mile Acceleration (Gzip Compression)". Inside this box, the "Compress Response" dropdown is set to "Always".

API JSON Snippet

```
"behaviors": [
  {
    "name": "gzipResponse",
    "options": {
      "behavior": "ALWAYS"
    }
  }
],
"criteria": [
  {
    "name": "contentType",
    "options": {
      "matchCaseSensitive": false,
      "matchOperator": "IS_ONE_OF",
      "matchWildcard": true,
      "values": [
        "text/html*",
        "text/css*",
        "application/x-javascript*"
      ]
    }
  }
]
```

Monitor

Enabling features like `DataStream` or `Cloud Monitor` allows us to capture platform interactions between end users, Akamai edge servers, and the origin servers as logs and aggregated metrics on performance, events, and errors.



These logs are then aggregated and made available to your operations teams through push or pull APIs. This is of great importance and value for applications teams, as this capability allows the teams to monitor their performance and health of the API, especially when enabling caching at the edge (since these requests will not reach their infrastructure).

Disable Response Buffer

If the client does not accept chunking (that is, the request is HTTP/1.0), and if the response from the origin server does not contain a content-length header, then the edge server will buffer the response until it reaches 32 kb of the response body.

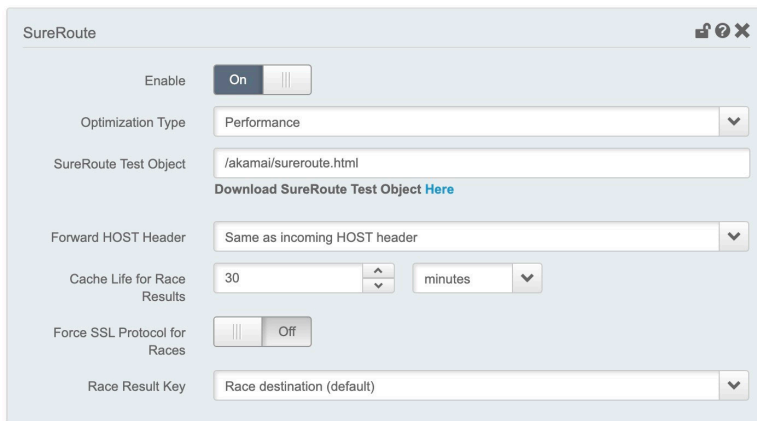
```
<network:http:buffer-response>off</network:http:buffer-response>
<network:http:buffer-response-v2>off</network:http:buffer-response-v2>
```

By setting this tag to "off," you can cause the server to deliver content to the client without buffering the response body.

SureRoute

The goal of SureRoute is to find the fastest path to the origin. Because most API requests are dynamic in nature, SureRoute should be configured to ensure the use of optimal routes to application origin.

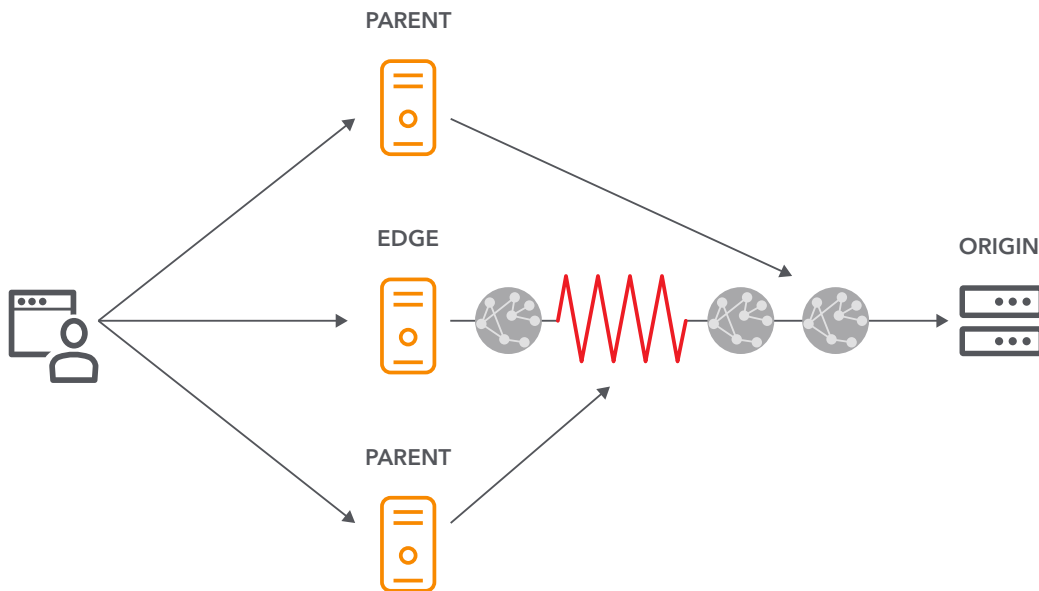
Property Manager UI



API JSON Snippet

```
{
  "name": "sureRoute",
  "options": {
    "enabled": true,
    "forceSslForward": false,
    "raceStatTtl": "30m",
    "testObjectUrl": "/srto.html",
    "toHostStatus": "INCOMING_HH",
    "type": "PERFORMANCE",
    "enableCustomKey": false,
    "srDownloadLinkTitle": ""
  }
}
```

Note: If the site is served over HTTPS, make sure to add "Force SSL Protocol Races" (see <https://developer.akamai.com/legacy/learn/Optimization/SureRoute.html>).



API Property: Best Practices

Cloud Origin

When working with IaaS providers (e.g., AWS, GCP, Azure), we have to ensure that we configure our edge servers to expect their dynamic behaviors. This essentially means that IPs will vary frequently and affect features like SureRoute.

We recommend the following best practices:

SureRoute: The default settings are to use either the incoming hostname or the origin hostname for the statistics gathered when calculating the best routes.

```
<forward:cache-parent.sureroute2>
  <force-origin-ip-from-edge>on</force-origin-ip-from-edge>
  <stat-key>
    <host>origin-ip</host>
  </stat-key>
</forward:cache-parent.sureroute2>
```

Forward Origin SSL (FOSSL): Configure the origin server behavior with “Third Party Certificate Store” as the certificate verification method and do not pin the leaf certificate. This allows Akamai to not only exercise control over the certificate to be trusted, but also maintain the certificate as up to date without having to update the configuration every time the IaaS updates it.

Origin SSL Certificate Verification

Verification Settings Third Party Settings

API JSON Snippet

```
{
  "name": "origin",
  "options": {
    "cacheKeyHostname": "REQUEST_HOST_HEADER",
    "compress": true,
    "enableTrueClientIp": false,
    "forwardHostHeader": "REQUEST_HOST_HEADER",
    "httpPort": 80,
    "httpsPort": 443,
    "originSni": true,
    "originType": "CUSTOMER",
    "verificationMode": "THIRD_PARTY",
    "hostname": "roymartinezblanco.github.io",
    "originCertificate": "",
    "ports": ""
  }
}
```


What Not to Configure

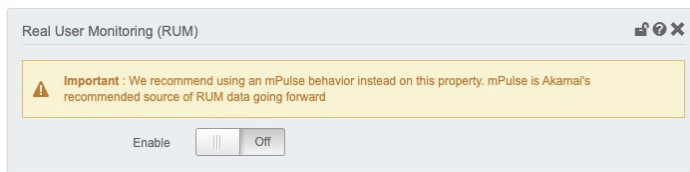
Because of the nature of an API, there is no need to enable some of our features.

Real User Monitoring

Real User Monitoring (RUM) solutions monitor end-user browser performance that, for an API, is not relevant. A website might use an API as part of the assets to dynamically pull some of the content shown to users – for this use case, if the API and the website are on the same domain, then RUM injection should be disabled for the API endpoint URLs. This is because an API will generally respond with a JSON body and an HTML body. If we have a RUM solution enabled, they generally insert themselves on most HTML responses, which are unnecessary for an API response.

Legacy RUM

Property Manager UI

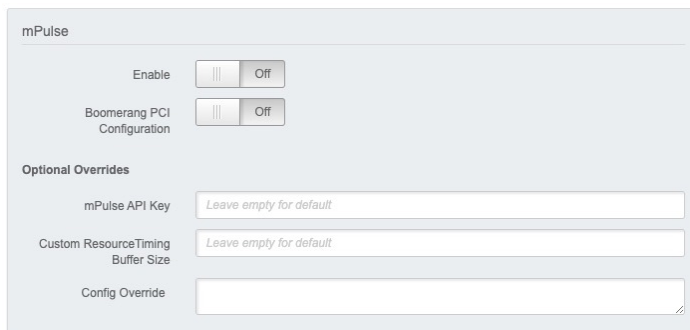


API JSON Snippet

```
{
  "name": "realUserMonitoring",
  "options": {
    "enabled": false
  }
}
```

mPulse

Property Manager UI



API JSON Snippet

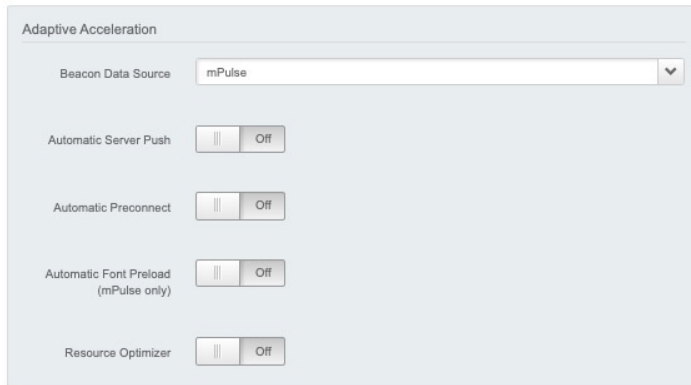
```
{
  "name": "mPulse",
  "options": {
    "enabled": false,
    "apiKey": "",
    "bufferSize": ""
  }
}
```

API Property: Best Practices

Adaptive Acceleration

Adaptive Acceleration (A2) is a feature set that automatically and continuously applies performance optimizations to a website using Akamai's machine learning to determine optimizations. With A2, we gain the ability to (i) prioritize which content is visible to the user, (ii) reduce the size of critical cached resources, (iii) manage third-party content, and (iv) discover, identify, and defer unresponsive scripts on your page. This is an excellent feature for web content, but it is not necessary for API endpoints (especially for Resource Optimizer).

Property Manager UI



API JSON Snippet

```
{
  "name": "adaptiveAcceleration",
  "options": {
    "source": "mPulse",
    "titleHttp2ServerPush": "",
    "enablePush": false,
    "titlePreconnect": "",
    "enablePreconnect": false,
    "titlePreload": "",
    "preloadEnable": false,
    "titleRo": "",
    "enableRo": false
  }
}
```

Content Prefetching

Prefetching is the mechanism by which Akamai can read the HTML response from an origin and start fetching embedded content like JavaScript, style sheets, and images before the browser starts to make the request. Since these are the resources required for loading the page in the browser, we try to reduce the round-trip times by having the object ready in the edge cache. When the browser parses HTML and requests JavaScript, style sheets, and images, we serve them without the latency of a round trip to the origin. Thus, by avoiding the processing time for the browser, we are able to load the resources faster.

This is a key feature for websites but not for APIs, since they will most likely be (at most) a resource within a site. When we have prefetching enabled and the API responds with content type "Text/HTML," it will parse the body of the API response.

Property Manager UI



API JSON Snippet

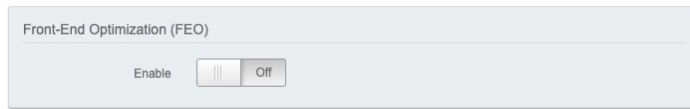
```
{
  "name": "prefetch",
  "options": {
    "enabled": false
  }
},
{
  "name": "prefetchable",
  "options": {
    "enabled": false
  }
}
```

API Property: Best Practices

Front-End Optimization

Front-End Optimization (FEO) employs several techniques that are similar to Prefetching and Resource Optimizer, in the sense that it analyzes and manipulates the response content. These types of optimizations generally aren't needed for APIs.

Property Manager UI



API JSON Snippet

```
{
  "name": "frontEndOptimization",
  "options": {
    "enabled": false
  }
}
```

Image Optimization

Some API endpoints, like analytic pixels, hide behind image URLs that could be treated by our edge servers as a normal image. To avoid this, we need to ensure that our customers' endpoints are excluded from Image Optimization if present in the same Property Manager configuration.

If your API is used to serve image content, then enabling Image Optimization is very important. It's also key to enable it only for the API endpoints that will serve this type of content.

A screenshot of a web interface titled "Image Manager". It contains three sections: "Image Optimization Settings" with an "Enable" toggle (set to "Off") and a "Region" dropdown menu (set to "United States"); "Traffic Settings" with two rows for "Pristine Images CP Code" and "Derivative Images CP Code", each with a text input field and a "Create new..." button; and "Policy Set (API Key)" with a "Policy Set Type" dropdown (set to "Standard") and a "Policy Set Name (API Key)" text input field containing "default-6900126".A screenshot of a web interface titled "Adaptive Image Compression". It contains two sections: "Adaption Settings for Mobile Network Requests" with an "Enable" toggle (set to "Off"); and "Adaption Settings for Non Mobile Network Requests" with an "Enable" toggle (set to "Off").

API Property: Best Practices

API as a Resource

In many scenarios, APIs are used as just another way to load resources such as images. For this use case and depending on the specific scenario, multiple features mentioned in the [What Not to Configure](#) section of this document can be added. The [What to Configure](#) and [What Not to Configure](#) sections from the [Standalone API](#) also apply here, but with some differences mentioned below.

NOTE: You need to be very specific with the match criteria (i.e., ensure that you have features enabled for the endpoints you want and not globally for the API). If we enable globally, it could – depending on the feature – have a negative impact.

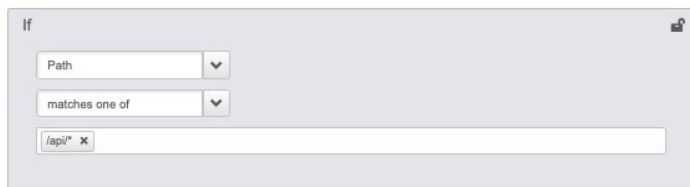
Scenario: One Domain

In this scenario, the API is used by a site to load resources, and it is either hosted under the same domain as the site or both domains are Akamaized.

Example: **www.example.com** is our website/application, and our API endpoint will be **www.example.com/api/getImage**.

For this use case, everything will live in one Akamai delivery configuration, and we'll have rules that will be applied for our API.

Property Manager UI



API JSON Snippet

```
{
  "name": "path",
  "options": {
    "matchOperator":
      "MATCHES_ONE_OF",
    "values": [
      "/api/*"
    ],
    "matchCaseSensitive": false
  }
}
```

Scenario: Multi-Domain

In this scenario, the API is used by a site to load resources, and both domains are Akamaized.

Example: **www.example.com** uses **api.example.com** to lazy load images or load any type of resource for the site.

What to Configure

Within this rule set, we should enable the following:

HTTP/2

HTTP/2 is a major revision of the HTTP protocol focused on performance and efficiency. Enabling HTTP/2 support on the end user will allow the browser to use HTTP/2 features like:

- Multiplexing and concurrency
- Stream dependencies
- Header compression
- Server push

Note: For HTTP/2 to work correctly, the domain has to live on the same certificate as the parent site.

Image Optimization


If we are loading images with this API, then we can and should optimize them where possible. Enabling Image Manager or Adaptive Image Compression will help us to not only improve the end-user experience, but also offload the application.

Prefetching

Because we are loading resources from our API, we can help the user by prefetching these assets.

Prefetching works on URLs with or without extensions (extensions and/or URL paths should be added to the match criteria, if any); in our use case, our endpoint has the path /api/getImage (no extension).

Property Manager UI



The screenshot shows a web interface for a Property Manager. It features a search bar with the text 'getImage' entered. Below the search bar, there is a dropdown menu with the text 'matches one of'. The search bar has a small 'x' icon to clear the text. The background is a light gray.

API JSON Snippet

```
{
  "name": "filename",
  "options": {
    "matchOperator": "IS_ONE_OF",
    "values": [
      "getImage"
    ],
    "matchCaseSensitive": true
  }
}
```

However, if our endpoint URL is something like /api/getImages/(directory), our criteria will change to path + EMPTY_STRING for the filename.

API Property: Best Practices

Property Manager UI

Path

▼

matches one of

▼

/api/getImage/*

✕

AND

Filename

▼

matches one of

▼

EMPTY_STRING

✕

API JSON Snippet

```
{
  "name": "path",
  "options": {
    "matchOperator":
      "MATCHES_ONE_OF",
    "values": [
      "/api/getImages/*"
    ],
    "matchCaseSensitive": false
  }
},
{
  "name": "filename",
  "options": {
    "matchOperator": "IS_ONE_OF",
    "values": [
      "EMPTY_STRING"
    ],
    "matchCaseSensitive": true
  }
}
```

Note: When prefetching is turned on and we are using multiple domains, such as **www.example.com** and **api.example.com**, it is possible to allow prefetching of the objects across domains. However, we need to ensure that all domains use the same edge hostname or edge map and enable the following snippet.

```
<edgeservices:prefetch.fetch>
  <allow-same-map>on</allow-same-map>
  <serial-must-match>off</serial-must-match>
</edgeservices:prefetch.fetch>
```

Other

On the default rule, we should have A2, RUM, and FEO, but these features should be explicitly disabled for the API endpoint. Generally, APIs are hosted separately from the web app (different origin). So remember to also add SureRoute for the alternate origins.



About the Author

Roy Martinez is a photography enthusiast, but in business hours he is an enterprise architect with 10 years of industry experience. He has a strong background in full-stack web development, DevOps, web performance, cloud computing, architecture changes, and advanced edge logic implementations, which allows him to provide consulting and support for customers.



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 akamai.com, blogs.akamai.com, or [@Akamai](https://twitter.com/Akamai) on Twitter. You can find our global contact information at akamai.com/locations. Published 06/20.