8 Ways to Secure Your Microservices Architecture

Okta Inc.
100 First Street
San Francisco, CA 94105

info@okta.com
1-888-722-7871

Table of Contents

You've heard the saying "less is more", but for developers and software architects, the opposite can be true—especially as customer and business demands mean they have to be more agile. The ongoing transition from monolithic architectures to microservices is a perfect example.

Compared to monolithic applications, which house all code in a single system, microservices are small, autonomous units that address individual functions and work with others to help an application function. This makes them a good choice for developers who need to deliver large, complex applications rapidly, frequently, and reliably. Operating with these distributed components brings several benefits, but also its own unique set of security requirements. Instead of a single entry point, microservices offer dozens or even hundreds of potential points of vulnerability—and as such each one needs to be effectively secured in order for an application to operate effectively.

4 benefits of microservices

Improves productivity and speed, as multiple teams can work simultaneously on different elements of a distributed system

Allows developer teams to selectively deploy specific components of their architecture—without causing major disruptions

The infrastructure is easy to use with containers (e.g., Docker)

There's room to choose the best tool for the problem

While this may seem like a daunting task, securing your microservices comes down to implementing a series of best practices that make security an integral component to how your developer teams work—without compromising productivity.

Here are eight steps your teams can take to protect the integrity of your microservices architecture.

1 Make your microservices architecture secure by design

Much like construction workers need to strategically layer rebar and concrete to build strong foundations for skyscrapers, developers must embed layers of security in applications to protect the data they hold. Within microservices architecture, this means being "secure by design"—keeping security top of mind at every stage of production, from design to build to deployment.

When it comes to writing your code, this means implementing a form of continuous stress testing on your architecture. In part, this means testing your continuous integration (CI) and continuous delivery (CD) pipelines. This can be done by simultaneously implementing security unit tests such as static analysis security testing (SAST) and dynamic analysis security testing (DAST):

- SAST will detect vulnerabilities in your code, as well as the libraries you import. It works from the inside and therefore requires a scanner that's compatible with your programming language.

- DAST works from the outside, mimicking malicious attacks, to identify vulnerabilities. Unlike SAST, it doesn't depend on a specific language.

These unit tests can be built into your delivery pipeline to help minimize the manual security checks that might burden your developers. The Open Web Application Security Project (OWASP) also offers a series of resources and analysis tools to help your team implement best practices as they build out software.

2 Scan for dependencies

Many libraries used to develop software depend in turn on other libraries, which means much of the code deployed to production consists of third-party dependencies. This makes security an even bigger concern, as these relationships could create large chains of dependencies, which might pose vulnerabilities for your systems.

These points of failure can be eliminated by regularly and thoroughly scanning an application's source code repository, new code contributions, and deployment pipeline for vulnerable dependencies (including updated release versions).

While the majority of applications supply release notes, only 75% report security issues—and just 10% report common vulnerabilities and exposures. Knowing your dependencies helps ensure there are no vulnerabilities due to new pull requests and that your code is up to date at the time of deployment. In addition to enabling security alerts in your repository, tools like Github's Dependabot can help automate updates via pull requests.

3 Use HTTPS everywhere

This may feel like a basic principle, but it's important to implement consistently as a foundational element to your internal and external operations.

While common attacks like phishing and credential stuffing are top of mind for most IT professionals when it comes to implementing security infrastructure, mitigating attacks that could originate within your network is also important. This can be done in part by implementing HTTPS across your microservices architecture.

Officially known as Transport Layer Security (TLS), HTTPS ensures privacy and data integrity by encrypting communication over HTTP. Think of it this way: much like driving requires a license that proves you are who you say you are and grants permission to operate vehicles, HTTPS requires a certificate to authenticate your identity and provide access to encrypted communications via Public Key Infrastructure. Once you've acquired your certificates, you can continue enhancing your security posture by automating certificate generation and renewals—keeping bad actors that might want to compromise your architecture at bay.

Every aspect of your microservices architecture, from Maven repositories to XSDs, can refer to these secure URLs. You can also use the HTTP Strict-Transport-Security response header to instruct browsers to only access your endpoints using HTTPS.

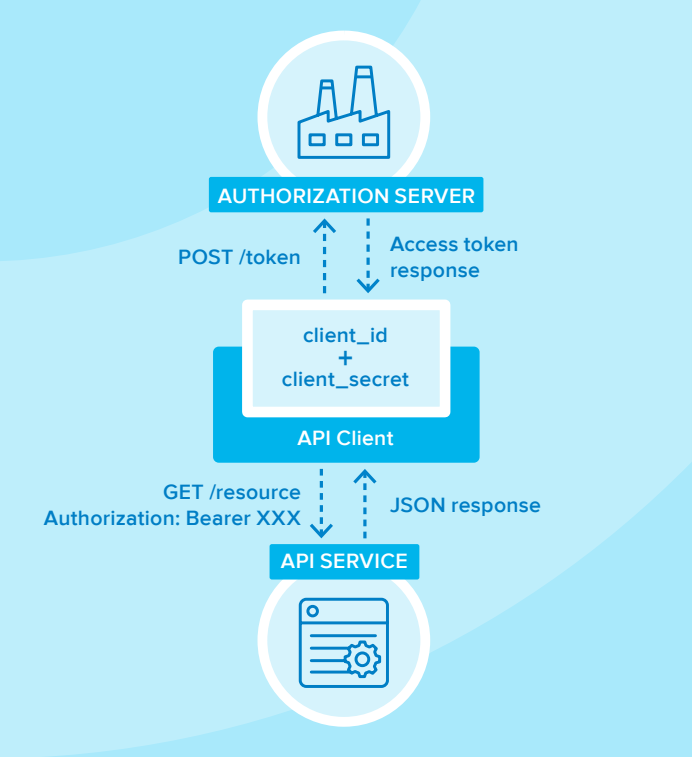
As you build your microservices and APIs, HTTPS will be vital for securing the data that's transmitted within your systems. And once they're deployed, it will serve to offer secure connections for external users, protecting your data—and your reputation.

4 Use access and identity tokens

A microservices architecture can encompass everything from the backend services that provide data, to the middleware code that talks to the data stores, to the UI that serves up the data in a user-friendly way. And putting the right tools and protocols in place is crucial for delivering secure and effective authentication and authorization across those microservices.

OAuth 2.0, for example, offers an industry-standard protocol for authorizing users across distributed systems. Within the context of microservices, OAuth 2.0's client credential flow allows for secure server-to-server communication between an API client and an API server.

In 2014, OpenID Connect (OIDC) extended OAuth, adding federated identity to delegated authorization. Together, these two layers offer a standard specification that developers can write code against in a way that will work across multiple identity providers.



OIDC and OAuth 2.0 also enable you to look up a user's identity by sending an access token to a user information endpoint. The path for this endpoint can be determined by using OIDC discovery. As a result, OAuth 2.0 reduces the burden on your developers as they don't have to build their own authentication mechanisms into each microservice.

OAuth 2.0 + OIDC can:

- Discover OpenID provider metadata
- Perform OAuth flows to obtain an ID token and/or access token
- Obtain JSON Web Key Sets (JWKS) for signature keys
- Validate identity tokens (e.g. JSON web tokens)
- Receive additional user attributes with access tokens from userinfo endpoints

Authorization servers: many-to-one or one-to-one?

Using the OAuth 2.0 standard, authorization servers play a key role as they are responsible for authenticating resource owners, issuing access tokens, and enabling user authorizations. Typically, authorization servers are set up in a many-to-one relationship where many microservices talk to a single authorization server. However, there are both benefits and challenges to this approach.

Pros of a many-to-one relationship:

- Services use access tokens to securely talk to other internal services
- Puts scope and permission definitions in one place
- Improves management for developers and security people
- Increases speed (because it's less chatty)

Cons of a many-to-one relationship:

- Opens applications up to rogue services, which can cause problems with tokens
- Puts all services at risk if one token is compromised
- Creates vague service boundaries because they're all communicating with a single identity engine

Another—more secure—option is to adopt an architecture where each microservice is bound to its own authorization server, ensuring that communications only occur within trusted relationships.

Pros of a one-to-one relationship:

- More clearly defined security boundaries

Cons of a one-to-one relationship:

- Slower
- Harder to manage

As they offer a more complex avenue towards secure communications, one-to-one relationships need effective planning and documentation before they can be adopted across your microservices architecture.

Okta's role as an identity provider

When it comes to providing secure access to your APIs and microservices, Okta plays the role of the authorization server, authenticating the user and issuing an ID or access token. Your gateway and downstream app can validate the token.

Meanwhile, our API Access Management product offers custom authorization servers, which allow you to adjust for audience parameters, custom scopes, and access policies. Implementing API Access Management helps developer teams save up to two weeks of time every year, reduces time spent writing code for authorization policies for each app or API pair by two days, lowers the chance of a breach via exposed APIs, and standardizes the tools, libraries, and training because everything speaks OAuth tokens.

By managing user and resource access to your APIs with fine-grained controls, Okta can support your API gateway in making allow/deny decisions. Operating with API gateway partners, Okta's solutions help you rapidly build, deploy, and secure new services; lower your cost of ownership; and easily provision and deprovision users across all your APIs.

Customer story

Pitney Bowes, a global technology company, has nearly a decade of experience developing APIs to help provide e-commerce solutions. Powering billions of transactions for millions of clients around the world—including 90% of the Fortune 500—it wasn't long before the company realized that it's on-premises infrastructure was both costly and time consuming to maintain.

"Our infrastructure was a big, iron ship—agility and speed weren't its strong point," says Kenn Bryant, Director of Architecture and SaaS Services at Pitney Bowes. And even though the company was using OAuth for web service authorization prior to implementing Okta, it faced challenges with integrations.

As part of its digital transformation efforts, Pitney Bowes decided to migrate to the cloud and integrate its API gateway into its overall identity and access management (IAM) strategy, purchasing Okta's API Access Management solution to facilitate the integration with its API partner.

This solution not only contributes to a reliable, scalable cloud infrastructure, but also offers integrations with more than 6,000 vendors and protects against data breaches for 100% of the company's APIs.

With Okta at the helm, Pitney Bowes' small development team can focus its efforts on user experience, rather than writing code and securing enterprise data. "Before Okta, it would take us a few days to integrate and expose APIs. Now it only takes a few hours," says Henry Rogando, Principal Software Architect. The company can easily administer APIs and manage user access from a centralized location to deliver a secure, unified set of shared services for developers.

5 Encrypt and protect secrets

When you develop microservices that talk to authorization servers and other services, the microservices likely have secrets that they use for communication—these might be API keys, client secrets, or credentials for basic authentication.

These secrets should not be checked into your source control management system. Even if you develop code in a private repository, it's likely to cause trouble while your team is working on production code. Rather, the first step in securing the secrets in your microservices is to store them in environment variables. Better yet, developers should encrypt secrets using tools like HashiCorp Vault, Microsoft Azure Key Vault, or Amazon KMS. With Amazon KMS, for example, you can generate a master key, encrypt your data with unique data keys that are then encrypted by the master key, creating an encrypted message that's stored as a file or in a database. This approach ensures the keys needed to decrypt any data are always unique and safe, which means your team can spend less time implementing and managing other safeguards.

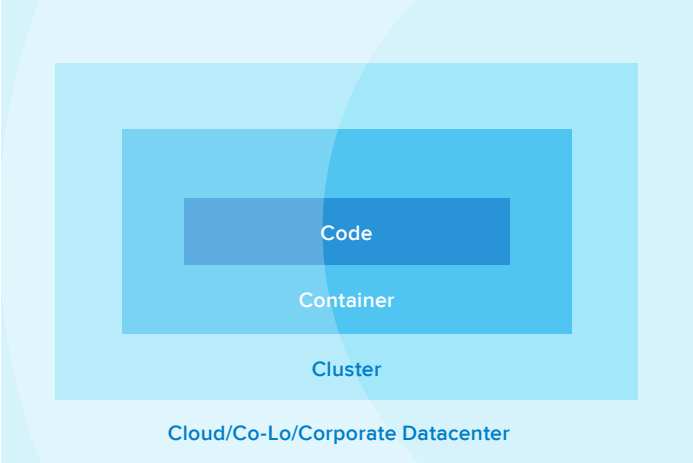
6 Slow down attackers

If someone tries to attack your API with hundreds of username and password combinations, it will likely take a while for them to authenticate successfully. Slowing down this process even more is another way to protect your various endpoints. Using an approach like rate limiting, for instance, means that attackers can only make one or two attempts per second, which may deter them from continuing credential stuffing attacks on your systems. Rate limiting can be implemented in your code—with an open-source library—or within an API gateway. Okta's API rate limits and email rate limits, for example, are particularly helpful in preventing denial-of-service attacks.

7 Know your cloud and cluster security

If your team is managing its own production clusters and clouds, they should be aware of the four Cs of cloud native security: code, container, cluster, and cloud/co-lo/corporate datacenter.

Each one of the four Cs depends on the security of the squares in which they fit. It is nearly impossible to safeguard microservices if you are only addressing security at one level. However, adding the appropriate security to the code, container, cluster, and cloud augments an already strong base.



We've talked about how you can best design secure code and how to operate within a secure cloud environment. When it comes to container security, meanwhile, ensuring the integrity of your containers and their base images is crucial. You can do this by:

- Using signed images with Docker Content Trust
- Building an internal registry
- Using secure container versioning
- Scanning container images for vulnerabilities and bugs
- Keeping configurations secret
- Preventing container breakouts by operating on least-privilege principles
- Implementing authentication features (this is true across all four Cs)
- Keeping up-to-date with container security best-practices—remember, what's secure today might not be secure tomorrow

If you're looking to secure your clusters and increase their resiliency in the face of an attack, these are the actions you should consider:

- Implement Okta Advanced Server Access to govern access to server accounts and apply authorization policies across any cloud
- Employ TLS everywhere (as we discussed above)
- Enable role-based access control with least privilege
- Disable attribute-based access control and use audit logging
- Use a third-party authentication provider like Google, GitHub, or Okta
- Rotate your encryption keys to mitigate vulnerabilities
- Use network policies to limit traffic between pods
- Run a service mesh

8 Cover your security bases

In addition to the strategies already discussed, there are three tactical ways your team can enhance your microservices security:

1. **Use rootless mode:** Docker 19.03+ has a rootless mode that was designed to reduce the security footprint of the Docker daemon and expose Docker capabilities to systems where users cannot gain root privileges. If you are running Docker daemons in production, this feature adds an extra layer of security.

2. **Implement time-based security:** The idea behind time-based security is that your system is never fully secure. As such, you don't only have to prevent intruders from accessing your application infrastructure, but also be able to detect anomalies and react quickly. Products like Okta's Advanced Server Access (ASA) slow down intruders by authenticating server login requests via Single Sign-On and multi-factor authentication and authorizing based on customizable access policies. ASA additionally generates rich request events that are consumable via API, or can be directly streamed to a SIEM like Splunk to help your team quickly detect security threats.

3. **Scan Docker and Kubernetes configuration for vulnerabilities:** Docker containers are often used in microservice architectures—so they need to be secured properly. This means following the steps we've outlined above for protecting images and containers alike.

Embarking on a path towards secure microservices

The move to microservice architecture is redefining how developer and DevOps teams operate. Stepping in for what has commonly been a monolithic approach to building systems and applications, microservices allow teams to be more agile, cost-effective, and better enabled to scale their systems. And as this approach to architecture becomes more prevalent within your organization, it's crucial that you enable your team with the right tools and resources to secure these disparate features.

At Okta, we've developed a suite of identity and access management products that can be easily integrated into your applications, minimizing the need for your team to develop internal security solutions and allowing them to focus on building highly-scalable applications for your business.

To learn more about how Okta can support you with your microservice security, get in touch.