



# NETFOUNDRY™

SPIN UP YOUR NETWORK

**Improving API reliability & security**

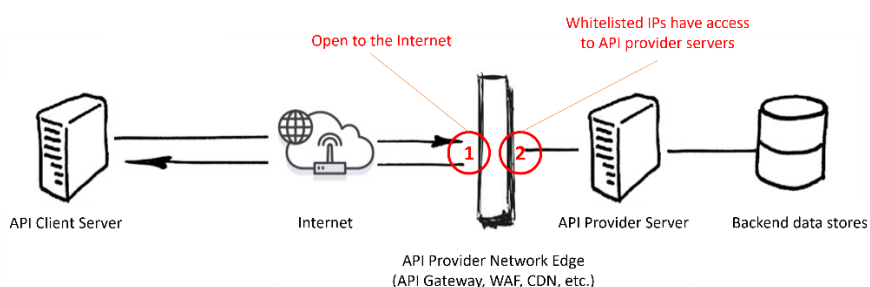
## Optimizing API reliability and security

API providers have been stuck between a rock and a hard place:

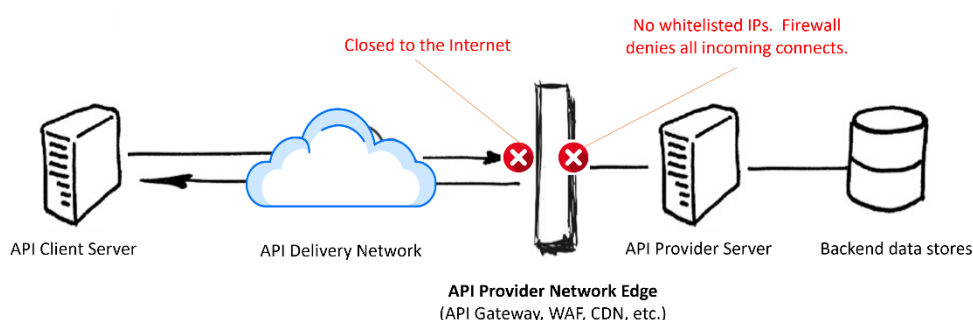
1. **Rock.** APIs are now the [#1 threat vector](#) so providers want to stop exposing their API edges to the Internet (API edges include API server, CDN, WAF, LB and API gateway). Private APIs (not exposed to the Internet) would be more secure than Public APIs.
2. **Hard place.** To **stop** exposing their API edges to the Internet, these providers would need VPN, MPLS and/or whitelisted IPs to make Private APIs available to API clients. Those techniques are used in single clouds and private WANs, but are often infeasible for Internet distributed API clients.

NetFoundry's API Delivery Network gives API providers a new option: make APIs edge inaccessible from the Internet, **while the API clients still use public Internet access**. Yes, Private APIs which are accessed from API clients on the Internet! The solution is similar to the Private API solutions which some cloud providers now offer. A key difference is the API Delivery Network enables API providers to serve **all** API clients – in **any** cloud, private DC or IoT environment – rather than just for APIs which stay in a single cloud or network.

**Without** the API Delivery Network, Public APIs are exposed to the Internet:



**With** the API Delivery Network, Providers seal their edge from the Internet – moving their API servers or edges behind a firewall with a an inbound firewall rule of **deny-all**:



This is done **without** VPNs, MPLS or IP whitelisting, **so API clients continue to consume the APIs from the Internet**.

This paper details the solution. However, API developers, cloud teams and ops teams can spin up the solution, right now, for free. For example, follow these recipes from [SPS](#), a leading MSP, to [secure Kubernetes APIs](#), enable zero trust access [to a private API gateway](#) deployed in a public cloud, or enable [multicloud, zero trust API access to a MuleSoft API](#).

## How to get on the API Delivery Network without VPNs or whitelisted IPs

Careful readers now have a burning question: how do I grant authorized API clients (customers or internal users) access to the private API Delivery Network from anywhere on the Internet **without** huge numbers of VPNs or whitelisted IPs to manage?

The short answer is to add code to the existing API code. Here's how simple it can be – this is the code for an app on Digital Ocean to access a private API on Oracle Cloud. The result of this code is no public IP addresses, no open inbound firewall ports, no Internet exposure for a multicloud, private API:

```
import ziti from '@openziti/ziti-sdk-nodejs';

const zitiIdentityFile = process.env.ZITI_IDENTITY_FILE;
// Authenticate ourselves onto the Ziti network
await ziti.init( zitiIdentityFile ).catch(( err ) => { /* probably exit */ });

const on_resp_data = ( obj ) => {
  console.log(`response is: ${obj.body.toString('utf8')}`);
};

// Perform an HTTP GET request to a dark OpenZiti web service
ziti.httpRequest(
  undefined,
  'http://darkside.api.com/', // Dark Service HTTP URL
  'GET',
  '/api/people/1',           // path part of the URL including query params
  ['Accept: application/json'], // headers
  undefined,                 // optional on_req cb
  undefined,                 // optional on_req_data cb
  on_resp_data               // optional on_resp_data cb
);
```

Adding that code is one of three options which developer and operations teams can use to secure API edges right now with [CloudZiti](#) (hosted NaaS; free for up to 10 endpoints, or [OpenZiti](#) (open source zero trust networking platform):

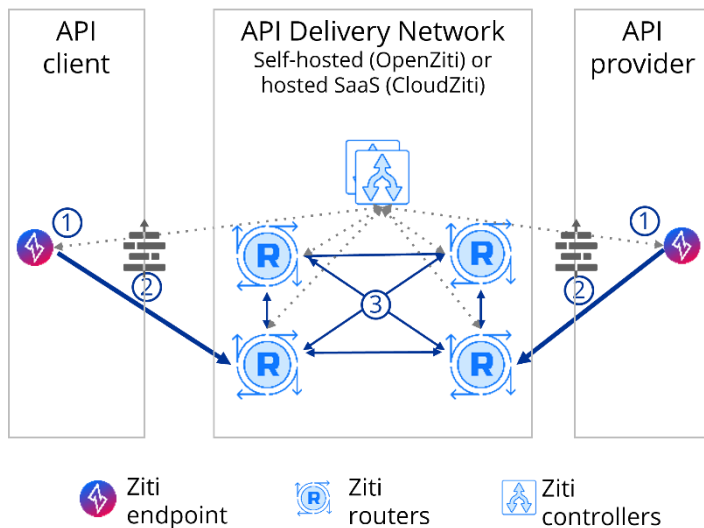
1. As shown above, option #1 is to use Ziti SDKs to add blocks of code to existing API endpoint code (more examples: [Golang](#), [Python](#)).
2. Option #2 is to add the Ziti code directly to API gateways (example: [NGINX](#)).
3. Option #3 is to use Ziti software agents.

Options can be combined – e.g. one option used on the API client side, and a different option used on the API provider side. Options #2 and #3 are similar to VPNs except they are built to only “intercept” the APIs, and to leverage full mesh, performance-optimized overlay network, rather than a single point to point tunnel for all data. For example, if the API gateway also talks to management systems, then the API provider can choose whether to use the zero trust overlay for that communication or not – it doesn't all get forced on to a VPN.

In summary, the API provider uses these 3 options to ‘shift left’ to build zero trust networking into the API itself (and the API client adds the code to their existing API query code, as demonstrated above). This is made simple for the API providers – they spin up private API Delivery Networks in minutes - because the API Delivery Network:

1. Provides all the zero trust and networking primitives.
2. Can be consumed as hosted, turnkey NaaS (“CloudZiti”), or as self-hosted open source (“OpenZiti”).
3. Is a cloud native network, and doesn’t require any hardware deployment, so the Ziti platform enables API producers to spin up private API Delivery Networks in minutes.

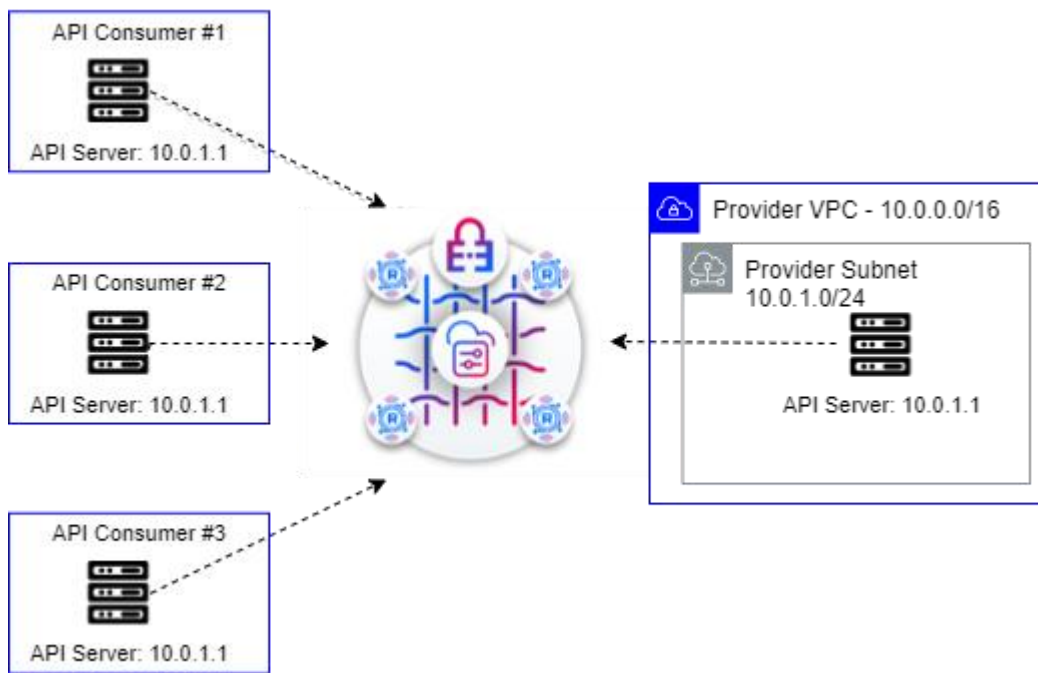
The API Delivery Network is built on the Ziti Zero Trust Networking Platform (which is used for any app or API, and is detailed in the appendix). In addition to the security benefits, the API Delivery Network optimizes performance by providing Smart Routing across multiple tier one networks, choosing the best paths for each session:



The API client and provider endpoints shown above include X.509 certificates with an integrated PKI solution for strong authorization. This enables API Providers to change their inbound firewall rule to deny-all, reducing their attack surface:

- Since the policy enforcement point is moved to the originator, the API Provider edge no longer listens to the Internet **and then** determines if requests are authorized.
- Instead, the API provider’s API edge solely communicates with its private API Delivery Network (and this communication is outbound only from the API server or API gateway towards the API Delivery Network).

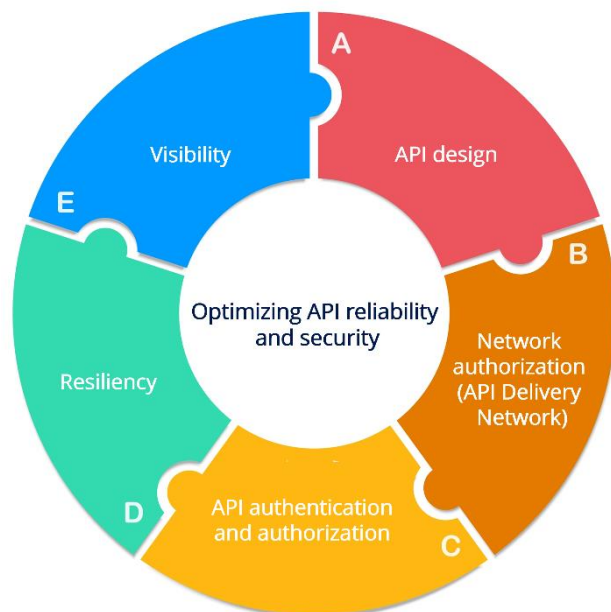
The diagram below shows this in a bit more detail. With the API Delivery Network, each API server (client side and provider side) uses private IPs, and still communicates with other API servers over the Internet, even if the IP addresses overlap (the API Delivery Network is the icon in the middle):



### Helping API providers mitigate against the top 10 OWASP API threats

The following section details how your API Delivery Network helps mitigate against the top OWASP threats as a key part of a mature API delivery approach designed to optimize uptime and security. While no model is 100% airtight from a security perspective, these concepts decrease the risk of breach-related downtime:

- A. **API design** includes data models so each client has access to the minimal amount of data they require, with data validation of all queries. It includes built-in instrumentation and management for simple and strong monitoring, auditing and logging.
- B. **Network authorization** is the focus of the API Delivery Network. It protects API Providers from Internet-based attackers exploiting vulnerabilities and misconfigurations such as the ones described in the OWASP top 10.
- C. **API authentication and authorization** enables providers to authorize each API request. This is usually done by API servers, API gateways or WAFs. It is a critical function, and is effective other than in cases like the vulnerabilities described in the OWASP Top 10 (which is why those vulnerabilities are in the top 10). However,



when the API Delivery Network is authorizing at the network layer (mitigating against the threats outlined by OWASP), and other solutions are authorizing at the application layer, then API providers have two independent and complementary layers of security.

- D. **Resiliency** includes load balancing, high availability and rate limiting. It is critical at scale, and needs to be built on the philosophy of assuming the system needs to survive the failures of any individual components. This is an area in which API gateways, CDNs and load balancers can be helpful. The fabric of the API Delivery Network also provides this function.
- E. **Visibility** includes operational and stakeholder visibility, logging, auditing and diagnostics. Visibility is critical during the ‘good’ and the ‘bad’, as visibility ‘shines a light’ on trends before they result in customer impact. The API Delivery Network is helpful in providing network-level visibility, as well as eliminating noise from unauthorized queries (since only authorized queries can now reach the API edge).

With that context, let’s look at the top OWASP threats in more detail. This is a summary chart, and [the details behind it are here](#).

This chart focuses on mitigating attacks from unauthorized endpoints. Authorized endpoints also need to be considered to be an attack surface, but that attack surface is orders of magnitude narrower than the Internet attack surface. The API Delivery Network primarily mitigates the Internet attacker risk, but in some cases (see the detailed analysis linked above) is also helpful for mitigating attacks from authorized users:

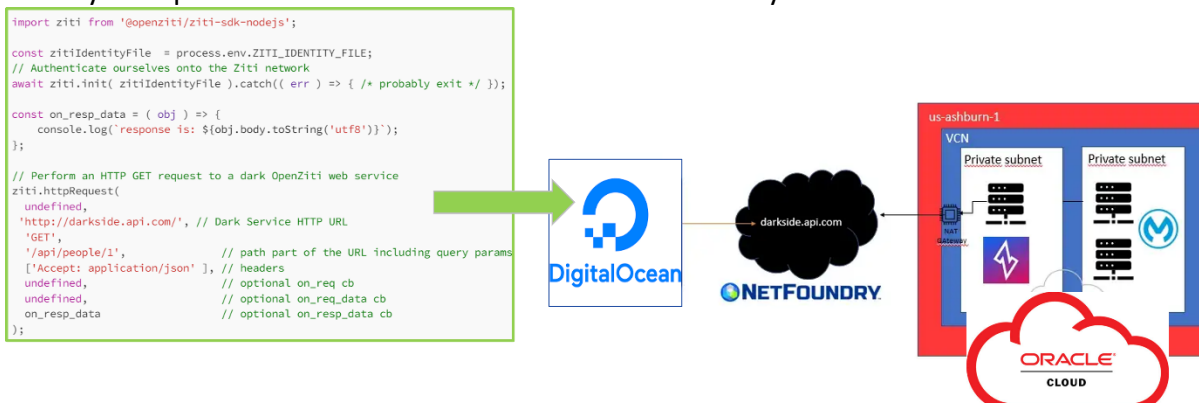
Mitigating risk from unauthorized API clients without the API Delivery Network	Mitigating risk from unauthorized API clients with the API Delivery Network
<b>OWASP #1 Severity API Threat: Broken Object Level Authorization</b> <b>OWASP #2 Severity API Threat: Broken User Authentication</b>	
<ul style="list-style-type: none"> <li>• Network FWs can only prevent attacks from known attackers, and only after their IPs are added to the ACL.</li> <li>• WAFs are excellent if the auth problem is previously known and widespread enough to be added to the WAF, and there are no bugs or misconfigs.</li> <li>• API GWs which provide mTLS user authentication and basic OpenAPI schema validations can be helpful for cases in which there are no bugs or misconfigs.</li> </ul>	<ul style="list-style-type: none"> <li>• The vulnerabilities can’t easily be exploited from unauthorized users via Internet because the API Delivery Network takes providers’ API edge off the Internet. In order to exploit the bug or misconfigs, an attacker also needs to break into your API Delivery Network.</li> </ul>

<b>OWASP #3 Severity API Threat: Excessive Data Exposure</b>	
When the API developer needs to expose many object properties, or rely on the client to do all the filtering, API edge solutions are more vulnerable to network attacks (beyond basic detection of sensitive data or schema deviances).	The API Delivery Network means only authorized API clients have network access, drastically limiting the attack surface.
<b>OWASP #4 Severity API Threat: Lack of Resources &amp; Rate Limiting</b>	
Most WAFs, network firewalls and API gateways are effective in helping to mitigate against these threats.	The API Delivery Network means only authorized API clients have network access, drastically limiting the attack surface.
<b>OWASP #5 Severity API Threat: Broken Function Level Authorization</b>	
<b>OWASP #6 Severity API Threat: Mass Assignment</b>	
Network firewalls do not help with these issues, and WAFs and API gateways can only do schema validation in some cases (e.g. OpenAPI). From the perspective of the WAF, the API user is simply executing permitted functions.	The API Delivery Network means only authorized API clients have network access, drastically limiting the attack surface.
<b>OWASP #7 Severity API Threat: Security Misconfiguration</b>	
<b>OWASP #8 Severity API Threat: Injection</b>	
<ul style="list-style-type: none"> <li>• Network FWs are irrelevant in dealing with these misconfigurations.</li> <li>• WAFs or API GWs may detect commonly misconfigured HTTP headers, methods, or permissive Cross-Origin resource sharing (CORS), but can't detect every vulnerability, so are still vulnerable to network-based attacks.</li> <li>• WAFs are now very good at detecting common injection flaws, such as SQL, NoSQL, and Command Injection.</li> </ul>	The API Delivery Network means only authorized API clients have network access, drastically limiting the attack surface.
<b>OWASP #9 Severity API threat: Improper Assets Management</b>	
This category generally needs to be addressed by the API developer. Further details are in the <a href="#">detailed analysis</a> .	
<b>OWASP #10 Severity API Threat: Logging and Monitoring</b>	
This category generally can be addressed by a variety of solutions, but Ziti does provide independent logging and monitoring.	

## Case studies and use cases

Common patterns, use cases and case study links:

1. **API server to API server, B2B, multicloud and hybrid cloud.** An API provider needs to make APIs available to multiple businesses (or different internal orgs), and often across different clouds and private data centers. [Here is a case study](#) of consuming a MuleSoft API in the Oracle Cloud from a Digital Ocean cloud. All over a private overlay with private IP addresses thanks to the bit of code you see to the left:



The capability to overcome this problem is embedding Ziti into the API or app, as code, as you see above (more examples: [a Java app](#), [AWS Lambda webhook](#)), so that the entities don't need VPN clients. However, the Ziti agents can also be used because they are API and app specific – meaning, they will only deliver the specific APIs or apps. So for example, if an Alice Company webserver is using the Ziti agent to consume an API from Bob Company API endpoint, then the Ziti agent will only carry the API over the zero trust overlay (for example, unless Alice Company chooses to configure it differently, the web server will maintain its current communication methods with all other entities). Other examples of this pattern:

[Private APIs](#) (Oracle)

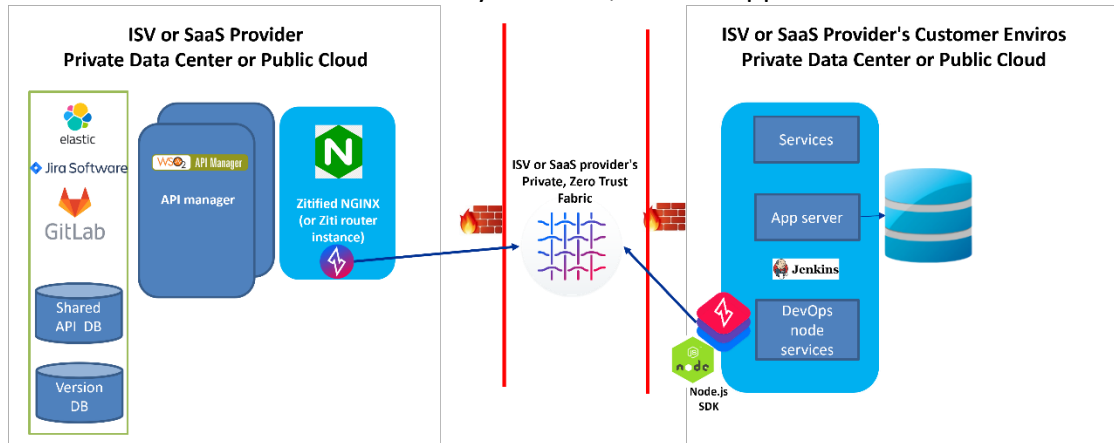
[Multicloud](#) (IBM)

[Hybrid cloud](#) (CERM)

[Kubernetes](#) (SPS)

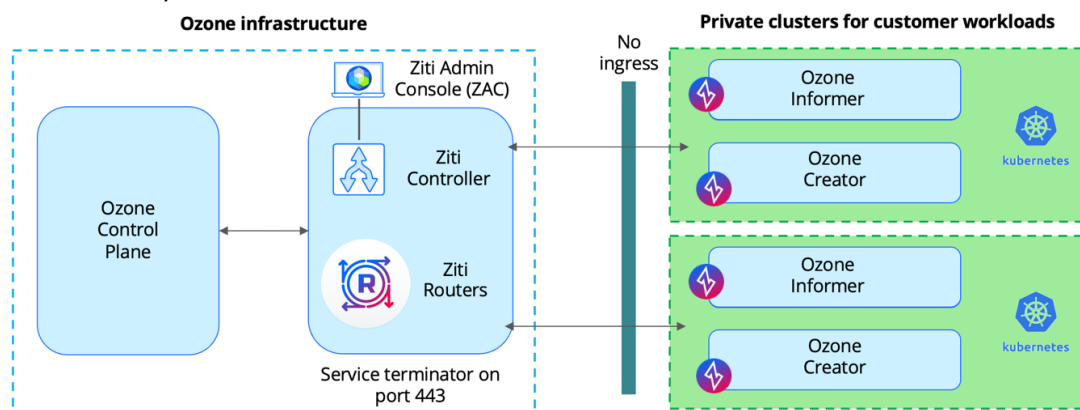
2. **ISVs and SaaS providers managing software deployed in their customer's clouds or networks.** Managing this software often involves API, SSH and or SFTP access. This access is usually managed via whitelisted IP addresses or VPNs. In this case, the Ziti

Fabric is used as both an API Delivery Network, and to support other data flows:



### 3. Securing DevOps, GitOps and management APIs, such as Kubernetes Kubectl.

Management APIs are often the only vulnerability of privately deployed software, at the very least requiring whitelisted IPs, and often being impractical to do over VPNs. Instead, leaders [like Ozone](#), use the API Delivery Network to transform these into zero trust APIs, meaning that their entire software deployment (or their customer's environment) can now be secured from the Internet:



4. **IoT APIs and IoT management.** Especially because IoT devices and gateways often have private IP addresses or constantly changing IPs, management via methods like APIs, RDP and SSH becomes a complex set of port forwarding, NAT, overlapping IP address management, whitelisted IPs and backhauling data through centralized intermediate points. This often impairs performance as well. [Private 5G](#) (Microsoft) and [IoT analytics](#) (TOOQ) are representative case studies. In the Microsoft case, a Ziti agent is used on the edge servers. TOOQ uses an agent on Nvidia Jetson and Raspberry Pi devices.

5. **Distributed apps and APIs.** Distributed API queries are multiplying as app and API servers are moving closer to end users, while datastores remain more centralized. Suddenly, these queries are traversing Internet instead of racks in the same data center. Meanwhile, self-hosted software increasingly has APIs and webhooks to remote services, creating holes in the firewall (whitelisted IPs), which can then be exploited. The API Delivery Network fixes this (for example, NetFoundry's customers running self-hosted Confluence servers [were protected](#)). MSPs and MSSPs have the

same problem because their API access (and RDP, SSH, etc) to customer environments also creates firewall holes which can be exploited. The MSP or MSSP can be used as ransomware conduit (see the infamous Kaseya attack). Examples:

[Zero trust user and admin access](#) (Ramco)

[Multicloud SAP remote management](#) (Novis)

[Zero trust MSSP services](#) (Ohka)

## Summary

Businesses use solutions such as MPLS-WAN, SD-WAN, VPNs and whitelisted IP addresses to reduce Internet exposure. Cloud providers offer similar services in their clouds (e.g. AWS PrivateLink, Azure Private Link) for the same purposes of eliminating Internet exposure.

However, most APIs are not in a single cloud or WAN, and need API clients to use Internet access. APIs are therefore still exposed to the Internet, which is why they are so vulnerable.

The API Delivery Network gives API Providers a new alternative: make API edges unreachable from the Internet, while enabling API clients to still use Internet access. Private APIs, consumed from Internet endpoints!

Secure your API edge (server, gateway, WAF, CDN) now:

- [CloudZiti](#) (hosted NaaS), free for up to 10 endpoints...be up and running in minutes
- [OpenZiti](#) open source zero trust networking platform

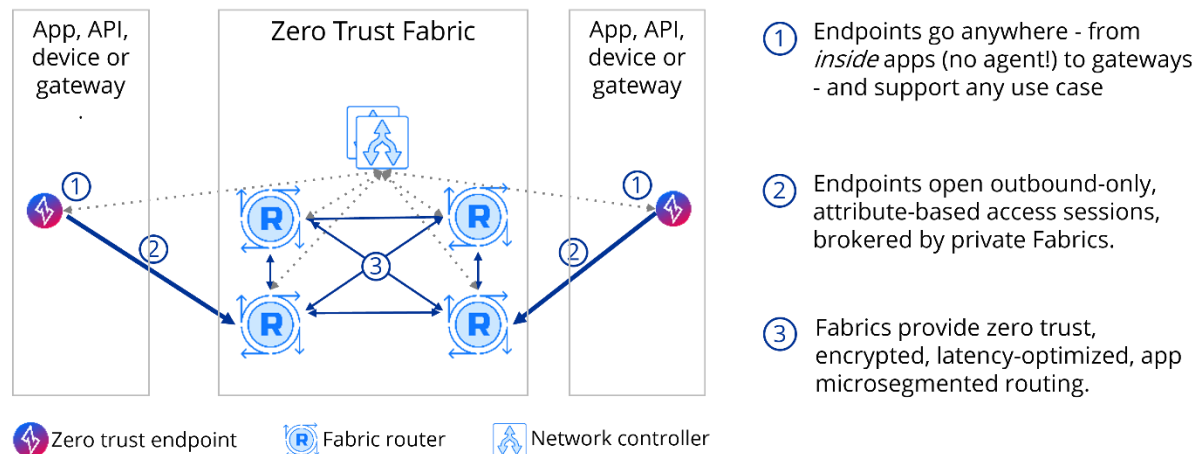
Results:

- **Improve availability and uptime.** The API Delivery Network is a critical piece of a complete architecture to maximize uptime and security
- **Simplify API security.** Replace firewall ACLs with one inbound rule: deny-all.
- **Reduce solution sprawl.** The platform secures the APIs, and secures remote management of the API infrastructure (admin access and CI/CD system access).
- **Simplify API operations.** Ops no longer needs to parse through thousands of unauthorized requests from the Internet.
- **Innovate.** The unique agentless approach enables developers to embed private, zero trust overlays in API client endpoints, as code.
- **Strengthen API security.** Includes mTLS (with enrollment, PKI infrastructure, key management, and standards-based options to integrate 3<sup>rd</sup> party CAs and identity solutions); integrated X.509 certificate identities; encrypted data plane.
- **Improve visible.** The Ziti platform provides app-level network visibility to all API producer internal groups, and even to the end customers (API clients), if desired.
- **Improve performance.** The Ziti Fabric includes real-time routing algorithms which minimize latency, dynamically routing APIs across multiple tier one backbone paths, selecting the best performing ones. API clients connect to the nearest Fabric location, minimizing Internet on-ramps, and maximizing middle-mile backbones.

## Appendix 1: architecture details

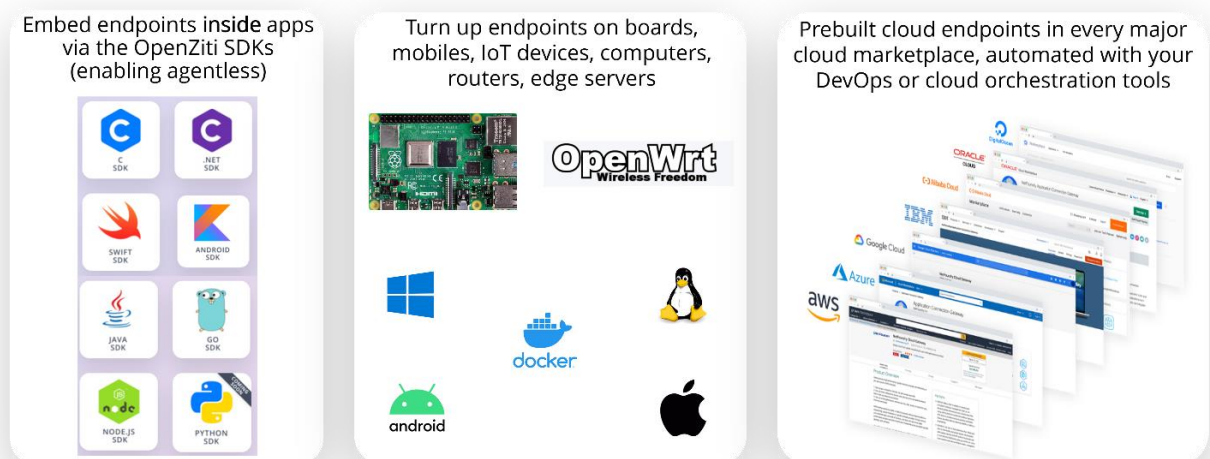
Although this overview focuses on APIs, NetFoundry's Ziti Platform combines zero trust functions with SD-WAN type functions to enable businesses to apply zero trust networking for any use case. This appendix shows this at a platform level.

The overall result looks like this:



### Ziti endpoints

You can use three main types of endpoints. App-embedded, host-based (including IoT) and cloud (or private data center) based:

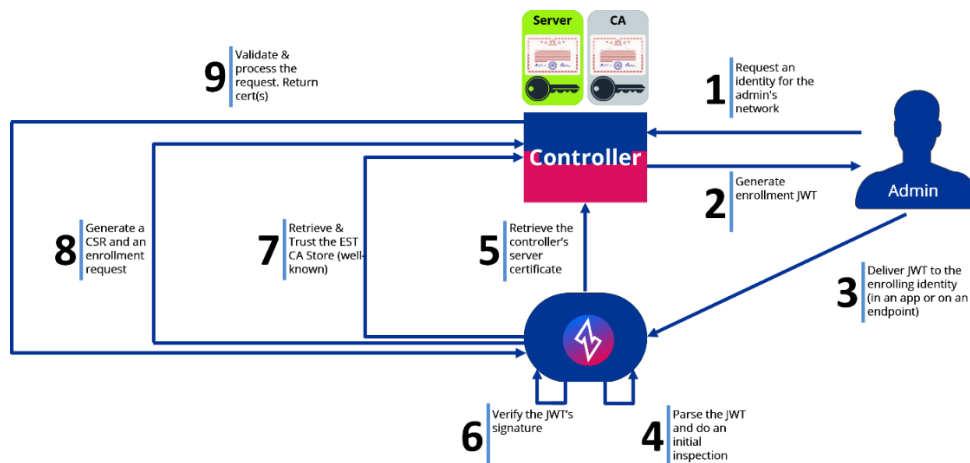


All Ziti endpoints:

1. Contain X.509 identities.
2. Enable MFA and posture checks.
3. Secure the app, data and device (make it unreachable from the networks).
4. Enable extensions into security and networking tools and solutions.
5. Provide networking capabilities such as routing, visibility and control.

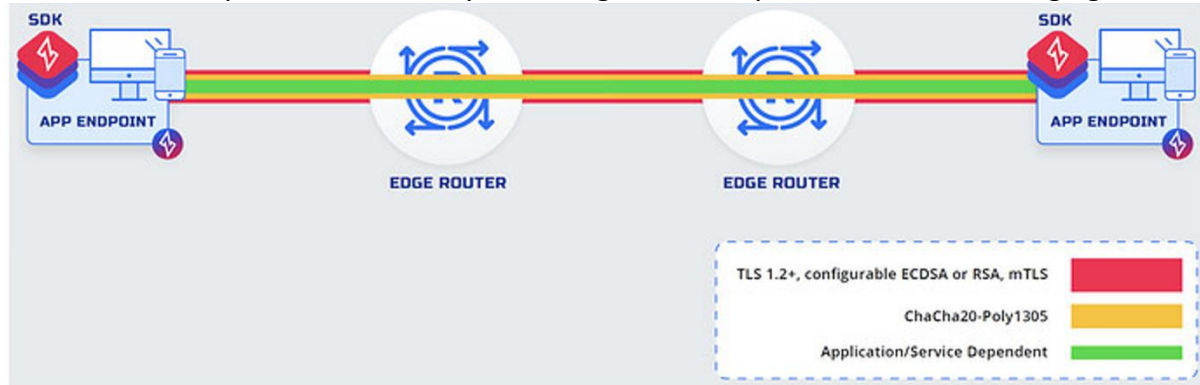
## Bootstrapped identification and authentication

Here is the basic diagram of the bootstrapped identity and authentication (details [are here](#)):



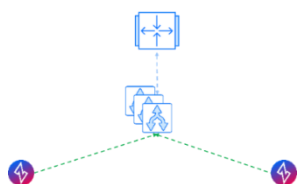
Endpoints need to be authenticated and authorized via their X.509 identities in order to access your private overlay. The bootstrapping and Certificate Authority (CA) are included, and you can [add your own CA](#) (via RFC 7030). The platform supports PKCS #11 - enables you to store certificates in secure ways (e.g. HSMs).

The result is comprehensive security, including mTLS everywhere, without managing PKI:



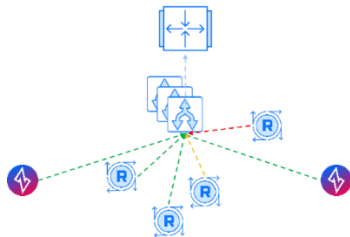
## Least privileged access via distributed controllers

Your authorization and access policies are enforced via your private, NetFoundry-hosted (CloudZiti) or self-hosted (OpenZiti) controllers



Least privilege access can be paired with your IDP or SSO type solutions such that even if your IDP or SSO solution is compromised, there is still no network connection (layer 3). Similarly, if Ziti is compromised, the attacker also often needs to thwart your IDP.

### Private overlay fabrics

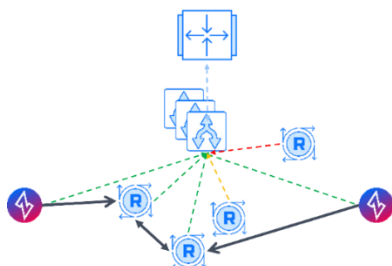


Your Ziti Routers function as combined routers/firewalls, but operate in the opposite model: instead of delivering packets unless told differently (IP based firewall rules), Ziti Routers deny all packets unless told differently (cryptography authorized flows). Ziti Routers are self-hosted (OpenZiti) or hosted by NetFoundry (CloudZiti).

The Routers are in every cloud marketplace and can be deployed as VMs anywhere. The Routers are governed by your policies (geofencing etc.) and are ephemeral – spin them up and down, programmatically. Your Routers form mesh overlay fabrics to:

1. Provide you with end-to-end control, across the overlay.
2. Enable you to close all your inbound ports. Instead, authorized Ziti endpoints will open outbound-only connections to authorized Routers. The Routers bridge the connections, enabling bi-directional data across the private overlay.
3. Provide you with optimized routing. This is detailed in the next section.
4. Enforce least privilege access on your overlay. This enables you to only grant permissions which are absolutely required for a given workload or session – no network level access. This applies to systems as well, for example a CI/CD system can only access certain ports on databases, and the connection will be ephemeral (only available during a code push).

### Optimized, multipoint routing



Real-time routing algorithms select the best path across your mesh, based on your metrics. In the example above, two of the four Routers are selected at first. This will change,

automatically, as conditions change. Your Routers form your private, programmable Fabric, which can be used to:

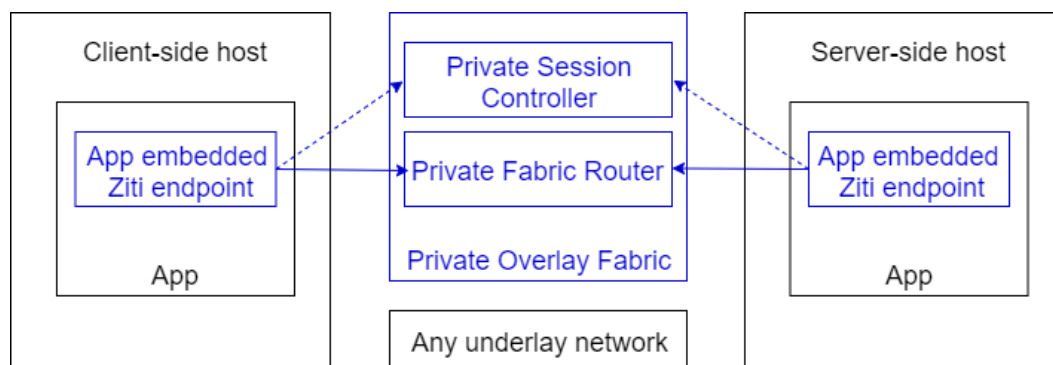
1. Enable your assets to deny all inbound connections, making your data unreachable from the networks
2. Enforce geofencing and similar policies. In the example above, the Router with the red connection was not eligible due to its location.
3. Use specific clouds for specific sessions.
4. Enable resiliency and improve quality (e.g. leverage Routers across multiple edges and clouds). With Routers across multiple clouds and networks, there are many potential routes. When “Internet weather” makes certain routes perform better than others, or if certain routes have outages, the algorithms will automatically (per programmable policy) select the lowest latency routes.
5. Create ephemeral data planes (periodically spin up new sets of routers, creating a moving target).

## Appendix 2: app-embedded, host-integrated, edge-integrated options

### App embedded option (app, API, browser, proxy, etc. embedded)

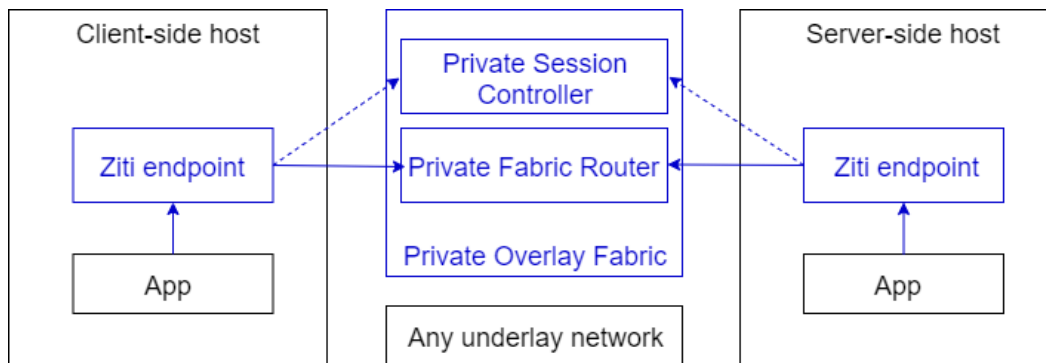
App-embedded (agentless) solves these problems:

1. Security and control. Use cases which require the strongest security and control can now securely connect without even trusting the hosts.
2. Topology. Use cases in which it is difficult to deploy agents, including B2B APIs, third party endpoints, IoT and unikernal environments. By embedding in the app, data or database, secure networking goes anywhere your app goes, eliminating DNS, VPN, etc. problems ([this video](#) shows how to use the SDKs).
3. Integrations. You embed the software directly into the app, browser, proxy, API, database driver, etc ( [the video in this blog post](#) shows a Postgres driver).



### Host integrated option (app specific agents for IT and IoT devices)

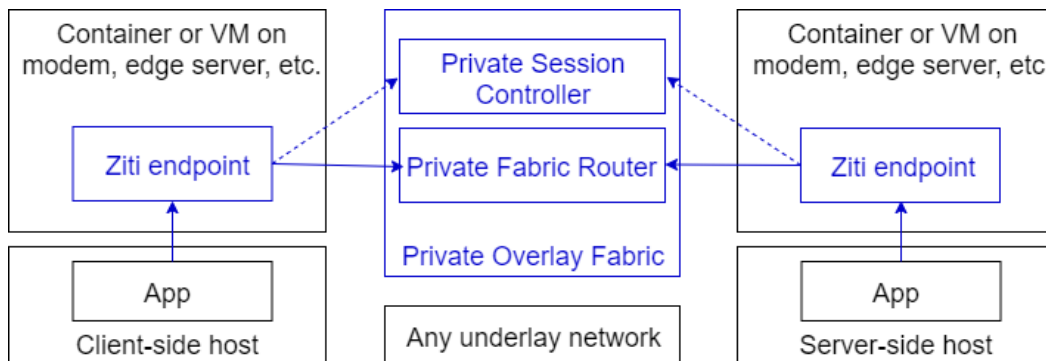
This option leverages Ziti endpoints, which [support every device type, OS and cloud](#), and are built on the Ziti SDKs described above:



Your Ziti endpoint is ‘moved’ from the app or browser in option #1 to the device hosting the app in this option.

### Edge-integrated option (containers or VMs):

In this option, Ziti endpoints are deployed as containers or VMs on modems, DMZ routers, edge servers and cloud instances:



This option places your endpoints on aggregation points in places like your DMZ or the edges of cloud networks.