NetFoundry™

SPIN UP YOUR NETWORK

Simple **and** secure API delivery

API Cloud Private, powered by Ziti

Private APIs, but accessed from anywhere

Private APIs are secure – only reachable from your WAN. However, that's also the problem – not all of your API clients are on your WAN. For example, some of your APIs may be a part of mobile and web apps, accessed from anywhere.

Now you can have the security of private APIs, unreachable from the Internet…and yet authorized API clients access your APIs from anywhere. For example, adding a few lines of code to your mobile app (or an agent, if you prefer) will establish private, mTLS connections to your API server, while your API server will be unreachable from the Internet. Here is an example in which an industry leader used the Ziti SDK for Swift to securely connect iPhone apps to Azure AI APIs.

API Cloud Private is built on NetFoundry's Ziti Platform. Ziti delivers billions of secure sessions per year for leaders such as Microsoft, LiveView and Intrusion.

Cancel the race against network-based API attacks

Public APIs are incredibly difficult to secure. This is because we need to continually find and fix all API vulnerabilities before any attacker on the Internet can exploit them - we are racing against the entire Internet. The statistics show the growing problem:

Gartner

Gartner 2017:
"By 2022, API abuses will be the most frequent attack vector".

Gartner 2021:
"The 2017 prediction could be counted as missed - because we *underestimated* the rise in API attacks".

Day two solutions – like WAFs, API gateways, API security platforms and IPS/IDS - are helpful but day two solutions can mainly only block threats which have already been identified. However, because APIs are so unique and rapidly evolving, many API attacks are effectively zero day attacks, unknown to the day two security solutions until after it is too late. On the other hand, API Cloud Private enables you to **cancel** the race against the Internet by making your APIs private – **not** exposed to the Internet.

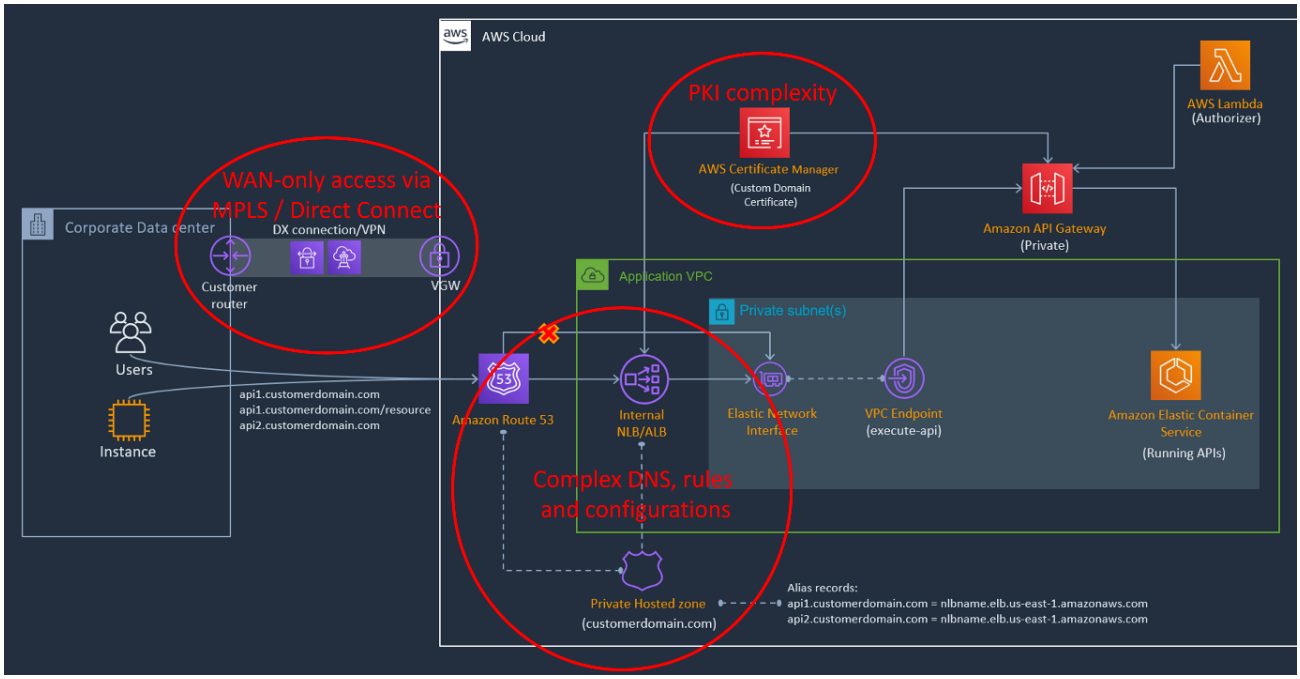
In summary, API Cloud Private:

1. Minimizes the threat surface – from the entire Internet to strongly identified, authenticated and authorized API clients (users, devices or servers).
2. Helps prevent data exfiltration and lateral movement if an attack does land.
3. Enables day two API security layers to be more effective (they can now focus on API requests from authorized users – they don't need to try to police the Internet).

Let's dive in!

Wait…private APIs are not an option

The problem with private APIs is complexity and access. Forcing API clients to access APIs through some the complexity of WAN extensions like AWS Direct Connect is not always a realistic option:



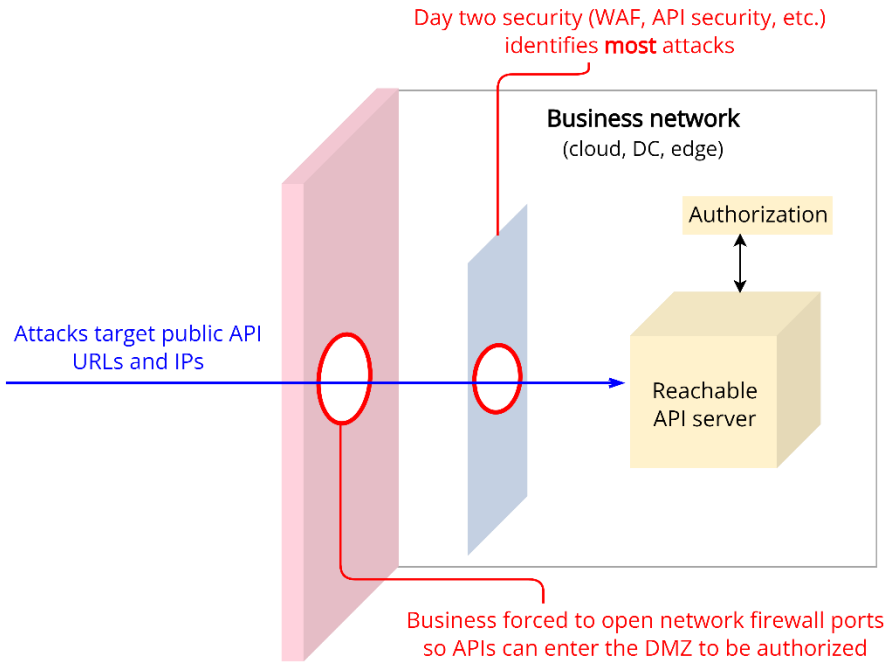
This is why the Ziti-powered API Cloud Private solution enables private APIs, but **without** the WAN, VPN, MPLS, bastion or permitted IP address complexity.

This paper details API Cloud Private, powered by OpenZiti, the open source zero trust networking platform. You can:

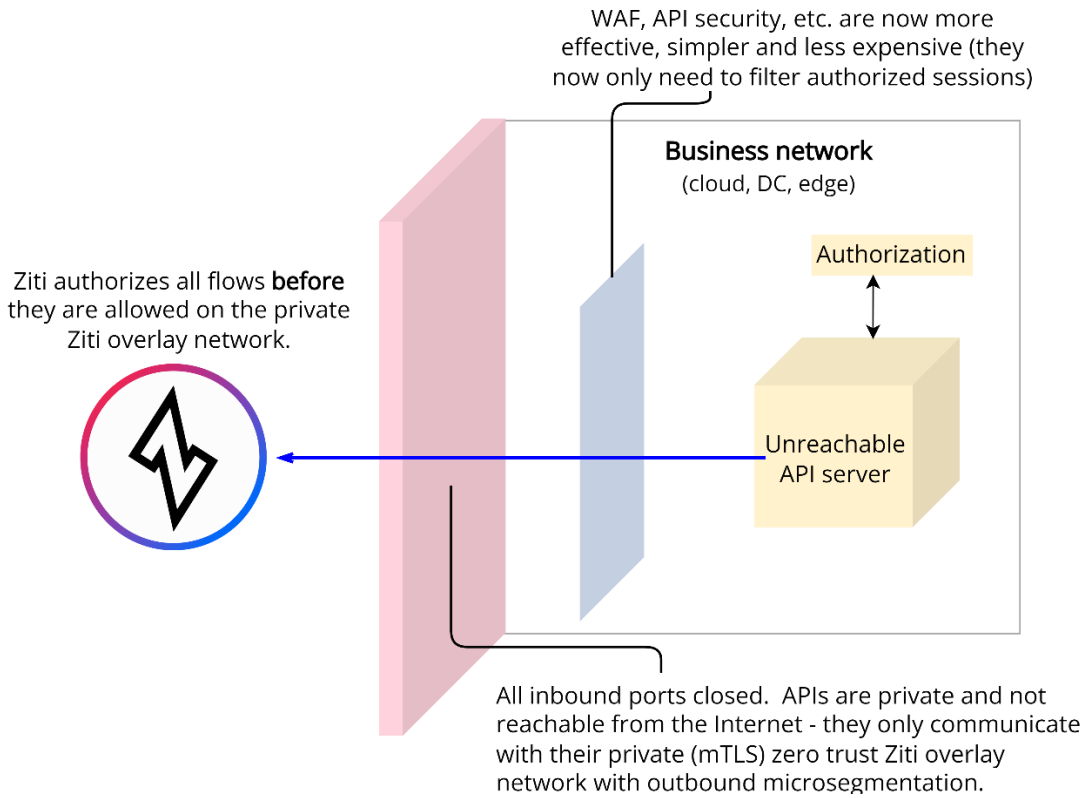
- Skip the paper and try it yourself – it is all software so you can spin up a free sandbox in minutes via CloudZiti (hosted OpenZiti, as SaaS)
- Try solution recipes like these from SPS, a leading MSP, to secure Kubernetes APIs, enable Internet access to a private API gateway, or enable multicloud, zero trust APIs.
- Dive right into the open source.

If you want to read the paper first, or are coming back to it now that your sandbox is up and running, then let's start by looking at how both public APIs and API Cloud Private APIs can be reached from authorized Internet-based API clients, but the API Cloud Private APIs can't be reached by Internet-based attackers.

Public APIs. We are often forced to make our API servers (or gateways) public, so that our API clients can access our APIs. The result is the attack surface is the Internet, and we rely on day two security to police the Internet, after the attacks are inside our DMZs:

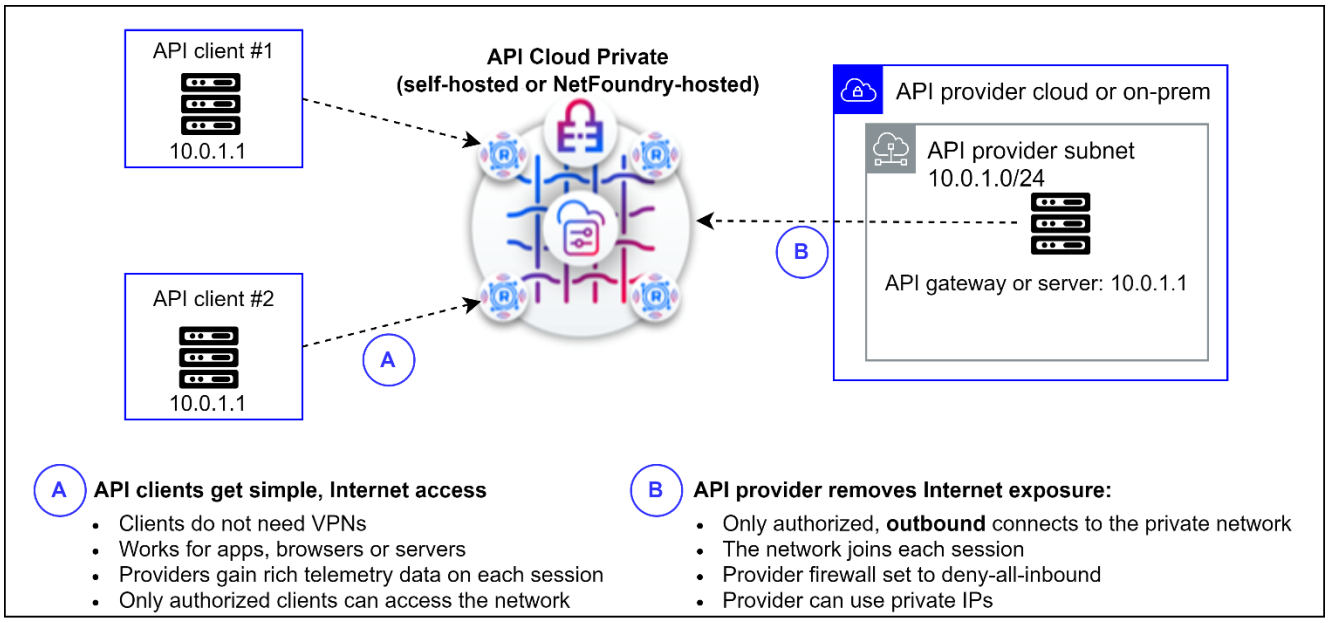


API Cloud Private. Now, you can use private API servers and gateways (literally…private IP addresses or domains), but yet authorized API clients can reach your APIs the same way they do today, without VPNs:



API Private Cloud architecture

How does API Private Cloud enable you to pull off that magic trick – to enable your API clients to reach your APIs at a private 10.10.x.y address, or a non-reachable domain like yourAPI.unreachable? Here's the overall picture:



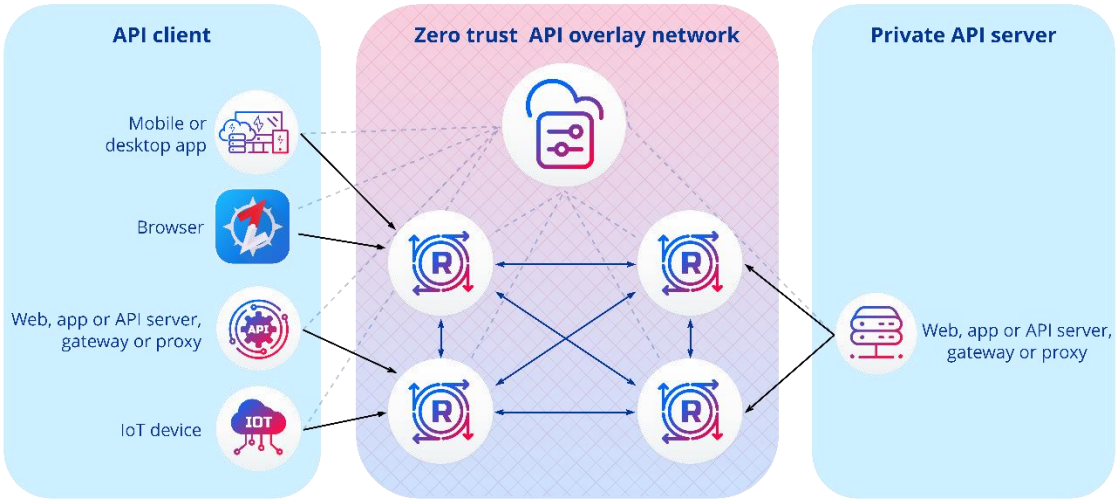
NetFoundry provides two overall options for API providers:

1. **Use CloudZiti SaaS**. Each API Cloud Private is still private, and NetFoundry's CloudZiti service provide the hosted global network, as software-only SaaS.
2. **Use OpenZiti open source**. OpenZiti enables API providers to deploy self-hosted, open source API Cloud Private instances.

We'll detail the architecture later in this paper, but in either option, you gain:

- Security. Close all inbound ports in front of your API servers. Use private, unreachable addresses or domains.
- Simplicity. Software-only. No MPLS, VPN, etc. dependencies. No impact on existing architecture. Spin up the networks in minutes, as software.
- Extensibility. Open source based, API-first, software-only, cloud-native architecture.
- Centralized visibility controls, policies and identities
- Global, mutual TLS (mTLS) API delivery
- Built-in identification and authentication
- Least privileged access authorization with data exfiltration mitigation
- Rich telemetry data
- Performance optimization with dynamic routing
- Encryption
- Standardization across all underlying networks and clouds

This architecture works for any type of API client:



Often, the best way to do this is agentless - using the Ziti SDKs to add a few lines of code to your existing app or API code, with the Ziti SDK enabling the zero trust networking. This example is how a Digital Ocean app accesses private APIs on Oracle Cloud:

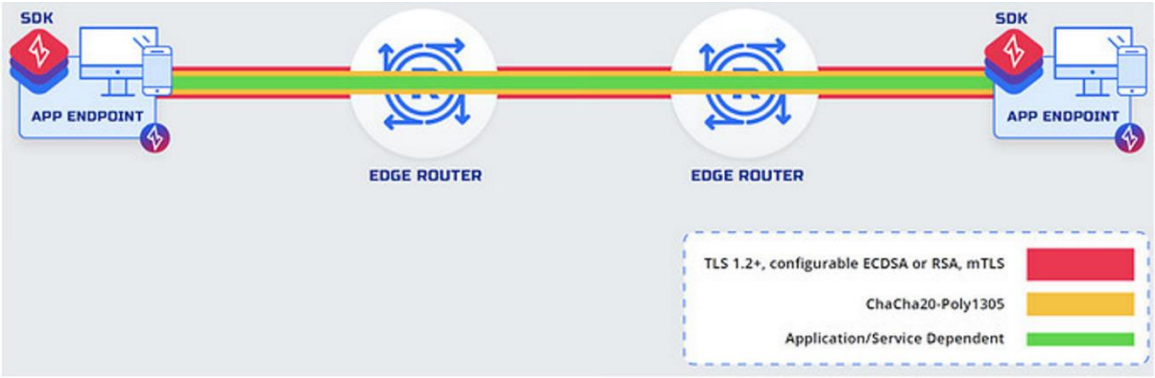
```
import ziti from '@openziti/ziti-sdk-nodejs';

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
);
```

If you do want to take an agent-based approach, then, unlike VPNs, Ziti software agents can be used, because Ziti agents:

- Only “intercept” the APIs – other data isn’t touched. This is because the client side app or API server is also talking to other systems, so can’t use a single VPN tunnel.
- Provide identity, authentication and least privileged access microsegmentation.
- Are managed as software – open source, cloud native, API-first software, made to integrate with the other layers of your API solution.
- Are integrated with the private, multi-tenant overlay network fabrics.

Ziti agents are deployed on any OS, as containers, sidecars or VMs, and at the edges of any network in a ‘gateway’ type model. The result is full security, including mTLS everywhere,

with Ziti taking care of enrollment and PKI. Note how even a compromised host wouldn't be able to access the API sessions!



Summary

API Cloud Private is a key layer in a defense-in-depth solution to enable secure, reliable API delivery, including mitigating against the top 10 OWASP API threats (see appendix).

The choice is straightforward: try to reactively defend against the entire Internet, or proactively don't allow unauthorized Internet sessions to enter your DMZ. API Cloud Private enables you to take the proactive approach, which also dramatically improves the effectiveness of the other layers. In summary, API Cloud Private:

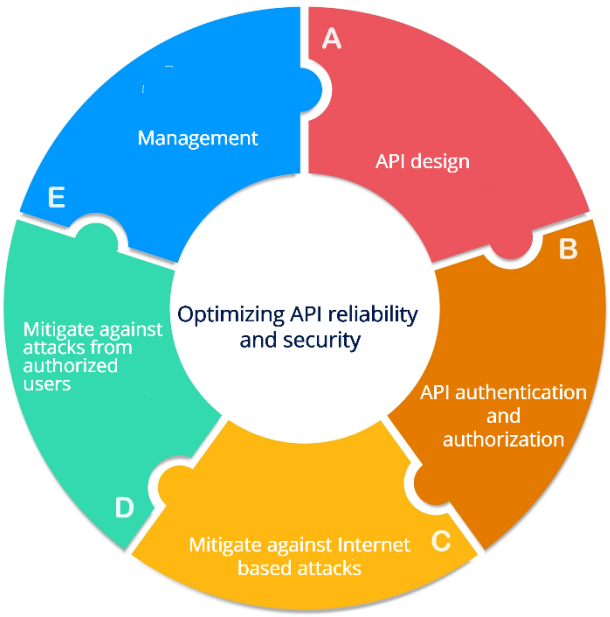
- **Innovation and speed**. The unique agentless approach enables developers to embed zero trust overlays in API client endpoints, as code. As an example, a mobile app can query private APIs, even from unmanaged mobile devices.
- **Private API servers**. All inbound is denied - this means that even if there is a zero day, misconfiguration, authorization bug, or business logic gap in the API provider network, unauthorized API clients cannot exploit the gap from the Internet.
- **Prevent data exfiltration**. In the outbound direction, the only permitted sessions are microsegmented API connections between authorized endpoints and private Ziti routers. Nothing else is permitted. For example, an attempt towards the home servers of a ransomware attacker is denied.
- **Better visibility**. The platform provides app-level network visibility. Meanwhile, ops doesn't need to parse through unauthorized requests from the Internet.
- **Better performance**. Optimization and direct paths for each API (no backhaul) minimizes latency, dynamically routing across tier one ISPs, selecting the best routes.
- **Strong security**. Endpoints need a private key verified X.509 certificate to enroll to each API Cloud Private instance. This prevents attacks such as stolen credentials, phishing, spoofing and compromised IAM. 3rd party CAs and HSMs can also be used.

Start now:

- CloudZiti (hosted SaaS), free for up to 10 endpoints…be up and running in minutes
- OpenZiti open source zero trust networking platform

Appendix one: mitigating against the OWASP Top Ten

It is very difficult to provide secure and reliable APIs, as illustrated by the OWASP Top 10. While no model is 100% airtight, organizations can take a multi-layered approach to decrease the risk of breaches and breach-related downtime:



A. **API design** includes comprehensive data modelling, so each API 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. **API authentication and authorization** enables providers to authenticate and authorize each API request. This is a critical function, and is very effective, other than in dealing with most of the vulnerabilities described in the OWASP Top 10 (which is why those vulnerabilities are in the top 10). However, when used with the other layers, this helps form a strong solution.
C. **Mitigate against Internet-based attacks**. This is done by the API Private Cloud, as described in this paper. This function makes the other functions far simpler and stronger, because they no longer need to try to police the Internet.
D. **Mitigate against attacks from authorized users.** With API Private Cloud taking the APIs off the Internet, the focus becomes authorized users – for example users which have been compromised. This is where WAFs and API Security Platforms (Salt etc.) are very helpful, at least for attacks which have been previously identified.
E. **Management** 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 also 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. This is an area in which API gateways, CDNs and load balancers are helpful. API Private Cloud is helpful in providing network-level visibility, eliminating noise from unauthorized queries and providing centralized identities, policies and controls.

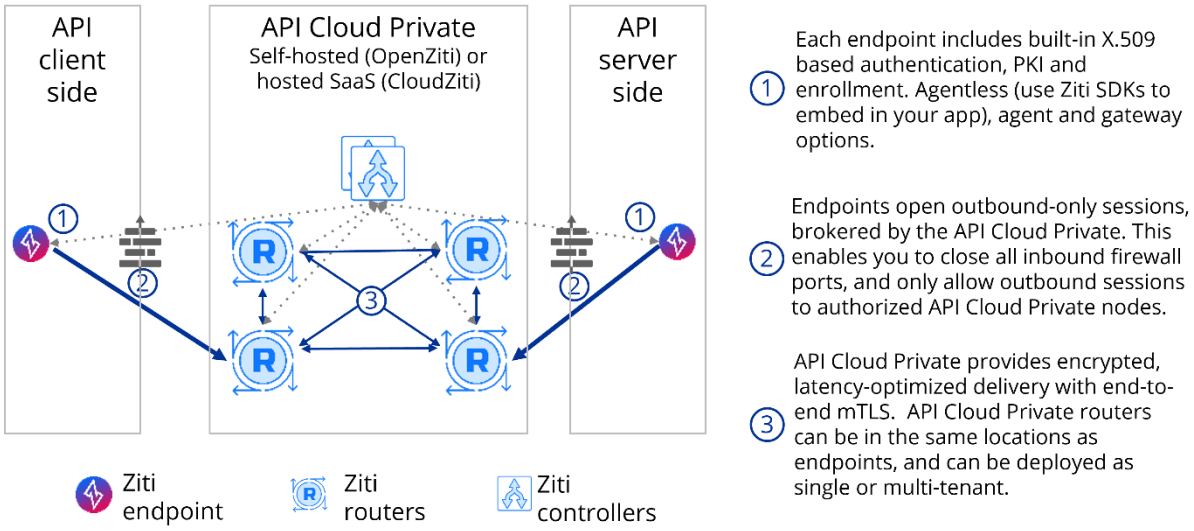
With that context, let's look at the top OWASP threats in more detail. This is a summary chart, and the details behind it are here. This chart focuses on mitigating attacks from

unauthorized endpoints. Authorized endpoints are also an attack surface, but that attack surface is orders of magnitude narrower than the Internet attack surface.

Mitigating risk from unauthorized API clients without private API extranet	**Mitigating risk from unauthorized API clients with private API extranets**
OWASP #1 Severity API Threat: Broken Object Level Authorization **OWASP #2 Severity API Threat: Broken User Authentication**	
Network FWs don't do auth. WAFs can help if the auth problem is previously known and widespread enough to be added to the WAF. API GWs which provide mTLS user authentication and basic OpenAPI schema validations can be helpful, but are vulnerable to attack (public).	The extranet means only authorized API clients get network access, drastically limiting the attack surface, and data exfiltration is blocked.
OWASP #3 Severity API Threat: Excessive Data Exposure	
When the API developer needs to expose many object properties, or rely on the client to filter, API edge solutions are more vulnerable to network attacks (beyond basic detection of sensitive data or schema deviances).	The extranet means only authorized API clients get network access, drastically limiting the attack surface, and data exfiltration is blocked.
OWASP #4 Severity API Threat: Lack of Resources & Rate Limiting	
Most WAFs, network firewalls and API gateways are effective in helping to mitigate against these threats.	The extranet means only authorized API clients get network access, drastically limiting the attack surface, and data exfiltration is blocked.
OWASP #5 Severity API Threat: Broken Function Level Authorization **OWASP #6 Severity API Threat: Mass Assignment**	
Network firewalls do not help. WAFs and API gateways can do schema validation in some cases (e.g. OpenAPI). From a WAF perspective, the API user is executing permitted functions.	The extranet means only authorized API clients get network access, drastically limiting the attack surface, and data exfiltration is blocked.
OWASP #7 Severity API Threat: Security Misconfiguration **OWASP #8 Severity API Threat: Injection**	
Network FWs are irrelevant. WAF or API GW may detect common misconfigured HTTP headers, methods, or permissive (CORS), but can't detect every vulnerability, so are still vulnerable to network-based attacks. WAFs are now very good at detecting common injection flaws, such as SQL, NoSQL, and Command Injection.	The extranet means only authorized API clients get network access, drastically limiting the attack surface, and data exfiltration is blocked.
OWASP #9 Severity API threat: Improper Assets Management	
This category generally needs to be addressed by the API developer. Further details are in the detailed analysis.	
OWASP #10 Severity API Threat: Logging and Monitoring	
This category generally can be addressed by a variety of solutions, but Ziti does provide independent logging and monitoring.	

Appendix two: API Cloud Private, architecture

Here is a quick visual for context before we discuss the main components, and then run through a sample flow:



Component #1: extend anywhere

The Ziti-powered API Cloud Private solution extends zero trust overlays anywhere apps and APIs go, including:

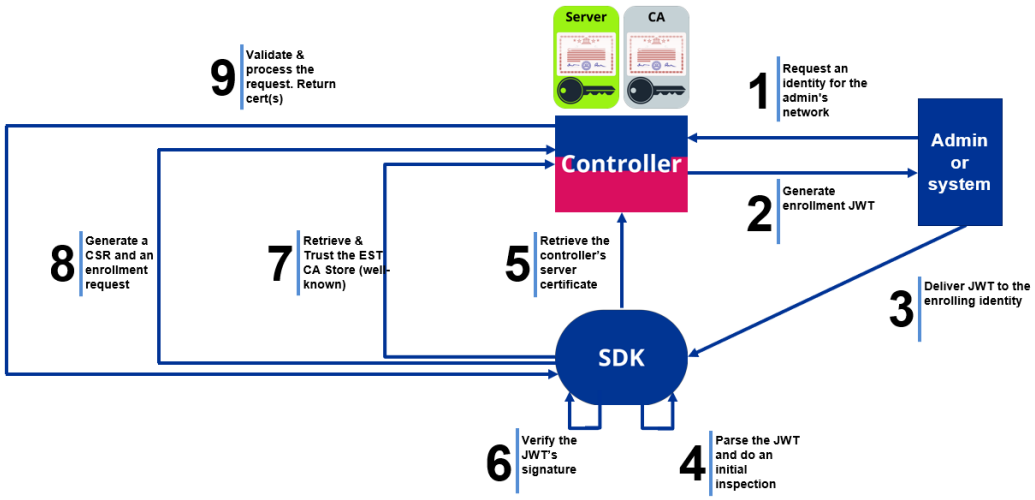
- Inside an application or solution (agentless) via Ziti SDKs. With this option, a provider adds the Ziti software to their existing software, such that there is no additional agent or gateway required. As detailed in the appendix, this provides the strongest security, and enables zero trust for enviros in which it is not possible or practical to deploy agents such as unmanaged devices.
- As host based agents for every device type, OS and cloud.

Appendix three details the agentless and agent options.

In all of the Ziti endpoint options, an X.509 certificate solution replaces IP address dependencies for strong identity, authentication and authorization. **Other sessions can't get access to API Cloud Private without enrolling with these X.509 certificates.** See next section for details.

Component #2: strong identity based authentication

In all of the endpoint options, the Ziti platform bootstraps a secure trust environment for secure enrollment. Each side of each session needs a bi-directional, private key (X.509) authenticated identity in order to earn a network connection. This is essentially a modern version of Network Access Control (NAC) in which the network is defined by software instead of sites, and private key / public key cryptography ensures simplicity and security. The Ziti controllers are spun up in containers or VMs on any hardware:



This strong identity is core to the authenticate **before** connect paradigm, which in turn enables all inbound firewall ports to be closed. This means you are not dependent on IP addresses, VPNs, SAML and DNS.

The SaaS solution includes the PKI bootstrapping shown above, and federates with third-party CAs (RFC 7030), and supporting solutions like YubiKeys (PKCS #11). With the open source option (OpenZiti), you have ultimate PKI flexibility.

Only apps and devices which have been securely identified, authenticated and authorized can use the overlay network connection, and they still don't get network level access. The endpoint will only get access to the specific services it needs – not an entire network/VPN or bastion. **This least privileged access microsegmentation minimizes the blast radius if something is compromised.**

Component #3: private network overlay connections

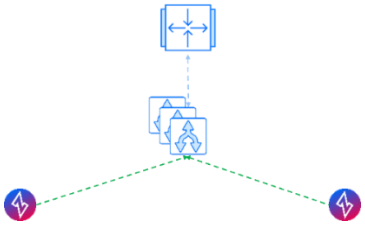
Now that we see how the architecture enables 'go anywhere' identification, authentication, authorization, least privileged access, isolation and microsegmentation for every use case, we have one main problem left to solve. We need to close all the inbound firewall ports. Enter the private overlay network fabric part of the API Cloud private solution.

1. Private fabric routers are software only so they can be deployed anywhere. For example, as a container or VM, or as a module on an existing vehicle network proxy, router or firewall. The routers broker all sessions, enabling each side to open outbound only connections to the private fabrics. Bidirectional data can then flow over these connections.
2. In the CloudZiti SaaS option NetFoundry or partners manage the private fabric routers. With OpenZiti open source, you manage them.
3. The routers are extensible, programmable and ephemeral – spin them up and down in minutes.
4. Your endpoints and routers form a private, full mesh data plane – with resiliency and dynamic routing across the mesh. The routers use the paths with the lowest latency, constantly measuring each path, and switching paths as conditions change. Each flow is routed directly and independently – it doesn't need to backhaul.

Sample Ziti secured session

Now that we have reviewed the components, let's look at an example:

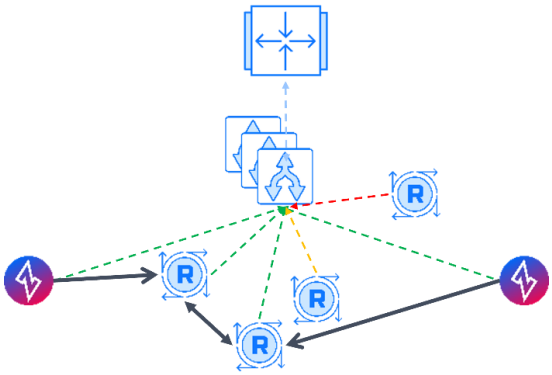
Step one:



The API servers are unreachable from the underlay networks because they have shut down all their inbound firewall ports. Each side of the API queries a Ziti controller to request a network overlay connection for a specific session. In this example, both requests are authorized, as represented by the green dashed lines. This means:

- Bi-direction identity verification and authentication was successful, using the X.509 certificate solution.
- All posture and MFA (optional) checks were passed.
- Policies authorized the connection of the involved endpoints and routers, for the specific requested service or application (least privileged access microsegmentation).

Step two:



This step allocates eligible overlay network routers, optimizes the path, and provides network resiliency:

- Depending on policies, such as geofencing, costs, latencies, etc., a set of routers will match for each specific app session. Ziti enables you to apply these policies to individual routers, or to groups of routers.
- In this case, the router connected with the red line doesn't match the policy, so will never be included. The router connected with the orange line does match the policies, but is not currently available.
- Real-time routing algorithms select the routers from the authorized set which results in the best path for a specific session, based on your metrics. The metrics default to minimize latency. In this example, two routers were selected.

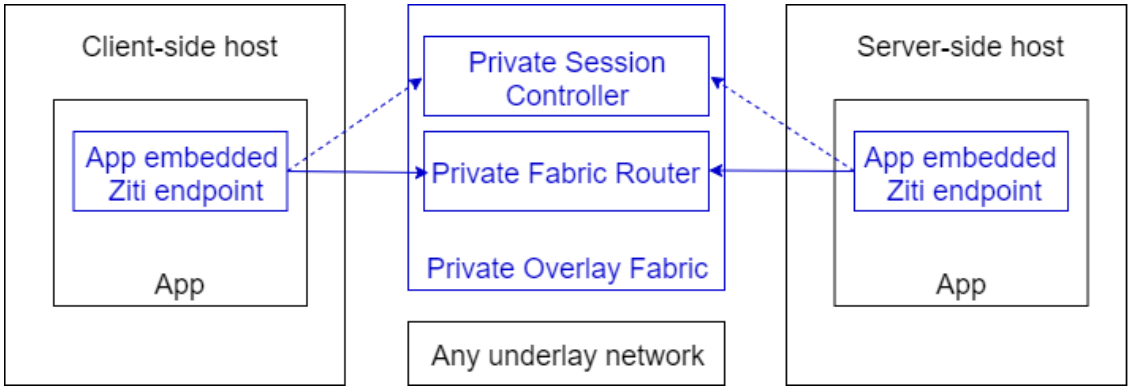
Appendix 3: endpoint details

Agentless options: **App, API, browser or JDBC-embedded**

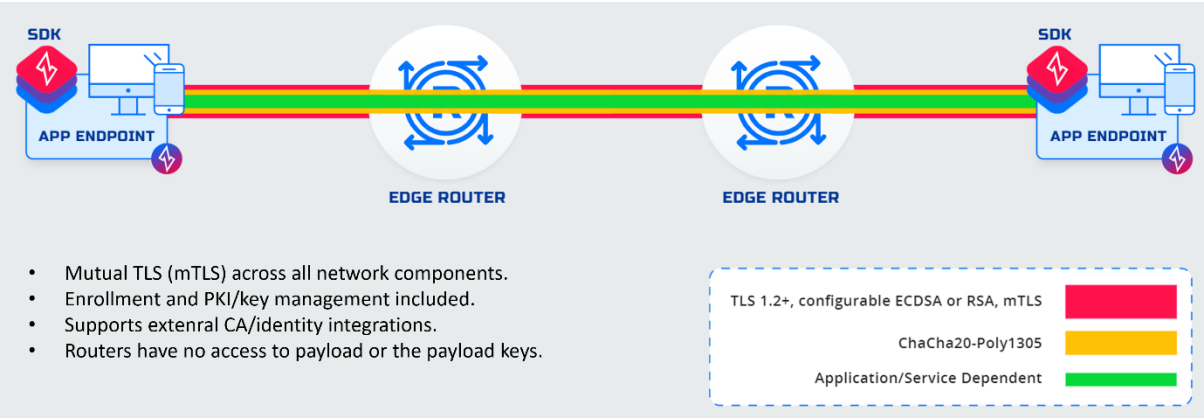
Agentless solves these two problems:

1. Your app or API requires the strongest security available.
2. It isn't practical to deploy agents, e.g. unmanaged devices and/or large numbers of endpoints.

It looks like this:



As you see above, nothing is trusted in the app-embedded option – not even the host device. Your app or API is enabled to establish an end-to-end, app-specific, encrypted, private network overlay connection, which is identified, authenticated and authorized before it is given network access. Even the server side has no listening ports on the underlay network! The result:



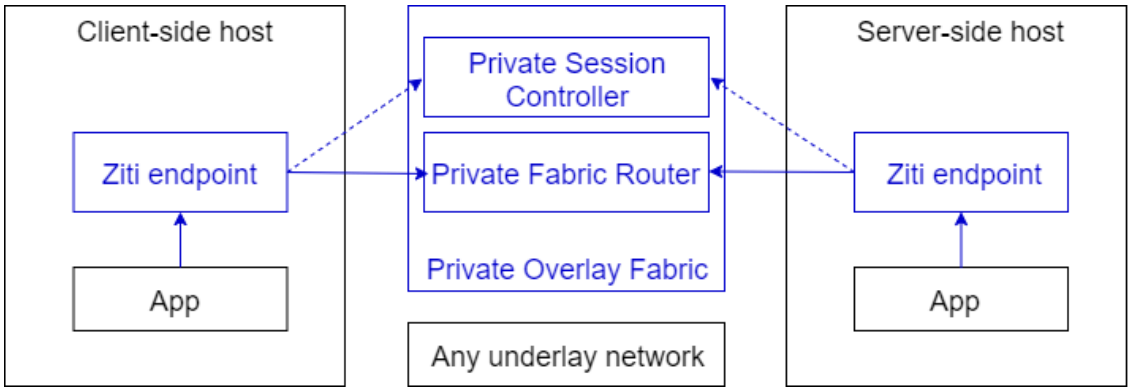
The Ziti SDKs enable programmers to add the Ziti code in the programming language which their app is written in. The Ziti SDKs essentially provide a shim around the network interfaces which each language uses, such that the shim handles all the zero trust networking, while the developer uses the same techniques, patterns, and idioms they already know. The developer is then able to focus on the desired experience, without worrying about IP addresses, NAT, VPNs, DNS, etc. For example:

- Golang provides a net package that include net.Conn and net.Listener interfaces. Ziti's SDK for Golang provides implementations of net.Conn and net.Listener which send data over Ziti when built into the Go application. There are some good write-ups and examples here.
- C-language apps often use socket descriptors to handle networking. Therefore the Ziti SDK for C provides socket descriptors that can be used with the existing networking calls.
- Ziti's Python SDK supports monkey patching the Python socket module, allowing a Python developer to embed Ziti with just a few lines of code.

There are also SDKs for Java, NodeJS, etc. The newest Ziti endpoint type, BrowZer, enables apps or APIs to be consumed from the browser, without any plugins, and without the developer needing to use a Ziti SDK.

Host-based options

Ziti software endpoints support every device type, OS and cloud, and are built on the same Ziti SDKs as described above. They enable you to deploy on hosts such as IoT devices, user devices, API gateways, and cloud hosts.

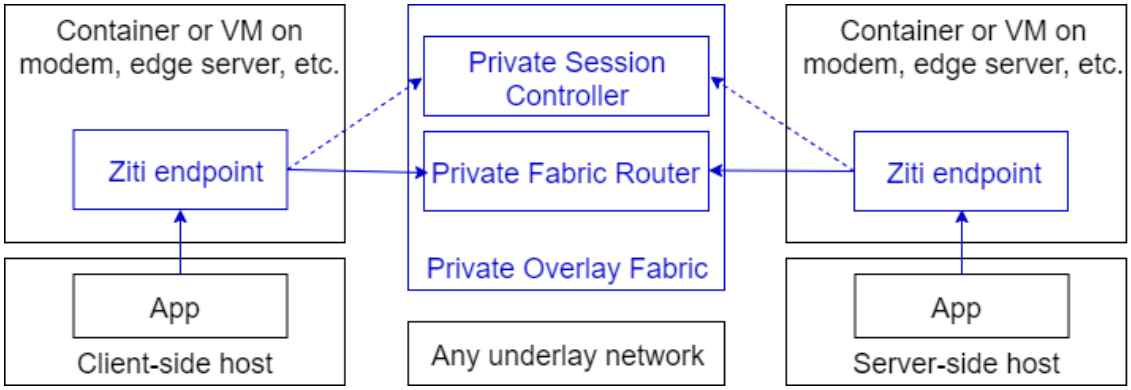


The Ziti endpoint is moved from inside the app, browser or driver to the device hosting the app. All inbound firewall ports are still closed, and the Ziti endpoint will initiate outbound, app-specific, microsegmented, private fabric network overlay brokered connections.

Host agents are app specific, meaning they **don't** backhaul data to a central location. Different apps and APIs are directly routed to their destinations, improving quality, decreasing latency and saving cloud egress costs.

Aggregation options

Ziti software endpoints are deployed as containers or VMs on aggregation points such as modems, DMZ routers, edge servers and cloud routers:



As in every other option, all inbound firewall ports are closed, and the Ziti endpoint will only initiate authorized outbound, app-specific sessions.

In all three of the Ziti endpoint options:

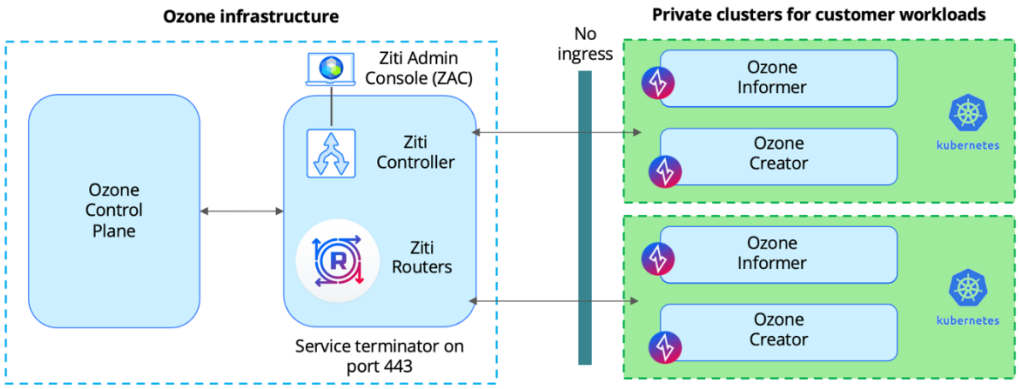
1. You can self-host (OpenZiti open source) or use CloudZiti SaaS (hosted by NetFoundry and NF partners, including the overlay network fabrics).
2. Prebuilt integrations are included and can be extended further.
3. Basic posture check capabilities are included.
4. X.509 certificate authorization is used, includes federating with third-party CAs (via RFC 7030), and supporting solutions like smart cards and YubiKeys (via PKCS #11).

Appendix four: case studies and use cases

API Cloud Private can be used for any API in which the end users are known, e.g. the user needs a token or claim to access your API (if your API needs to be available to unknown users, then it needs to be fully public). There are three dominant patterns:

1. **Extending Private APIs outside the WAN.** Make private API servers or gateways available to API clients (and apps), **without** backhauling via corporate WAN, and forcing the APIs on to circuit/VPN between the corporate WAN and a specific cloud region. For example, this enables mobile and web apps to access the APIs from any Internet connection, while the API server or gateway remains private. Here is a case study of consuming a MuleSoft API in the Oracle Cloud from a Digital Ocean cloud.
2. **Managing software inside customers' networks.** Using APIs to manage your software when it is inside your customers' networks, without needing customers to provide inbound network access, VPN, bastion or permitted IP addresses. Managing your software often involves APIs, SSH and RDP. There are two main problems:
    a. The customer simply refuses to give access. This often means the provider can't sell SaaS services, so needs to continue to support an on-premises model. It also means the provider can't easily update their software, causing customer support problems, and an inability to sell new features.
    b. The customer agrees to give access, but it is via whitelisted IP addresses or VPNs. So it is complex, operationally expensive, insecure and fragile (it is one audit, infosec change, or CISO directive way from being outlawed by IT).

    API Cloud Private solves both problems. Because the customer's network is no longer in danger (no open inbound ports), the customer infosec team permits the provider to manage their software. Because API Cloud Private doesn't require whitelisted IPs or VPNs, it is sturdy, simple, extensible and scalable. This is why leaders like Ozone, use the private API extranets for use cases like this:



3. **Securing management APIs, such as Kubernetes Kubectl.** Management APIs are often the only vulnerability of privately deployed software. By adding the Ziti code to APIs like the Kubernetes API (see example here), K8s APIs are made unreachable. Or, this video shows how Delta Secure provides secure SIEM/SOAR for SOC services.