Microsoft

How to secure your precious APIs with Azure API Management

Vladimir Vinogradsky

/in/vladvino

@vladvino

Agenda

API Management 101

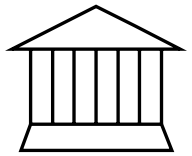
API security

Mitigations

Network security

Wrap-up and Q&A

API management distilled



Façade

Hide backends from frontends

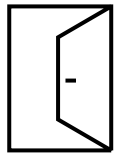
Aggregate or slice

Modernize or normalize

Re-architect

Move

Mock



Front door

Single point of ingress

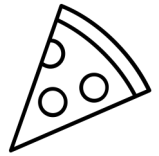
Route

Secure

Throttle

Transform

Observe



Frictionless consumption

Self-service user onboarding

Discover

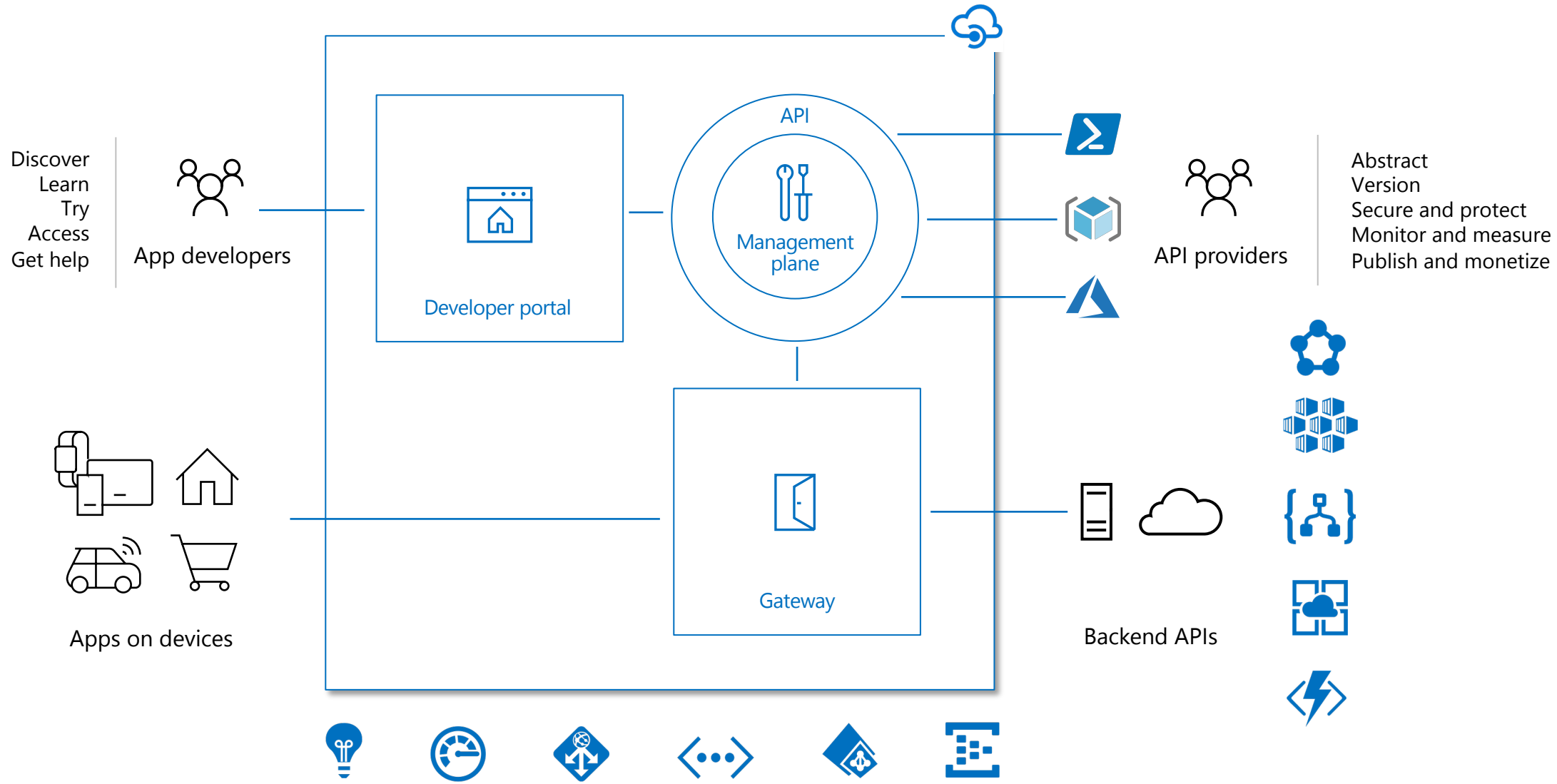
Learn

Try

Request access

Get started

Azure API Management



There is a policy for that

Encapsulate common API management functions

Access control, Protection, Transformation, Caching, …

Mutate request context or change API behavior

E.g. add a header or throttle

Set in the inbound and outbound directions

Applied at a variety of scopes or on error

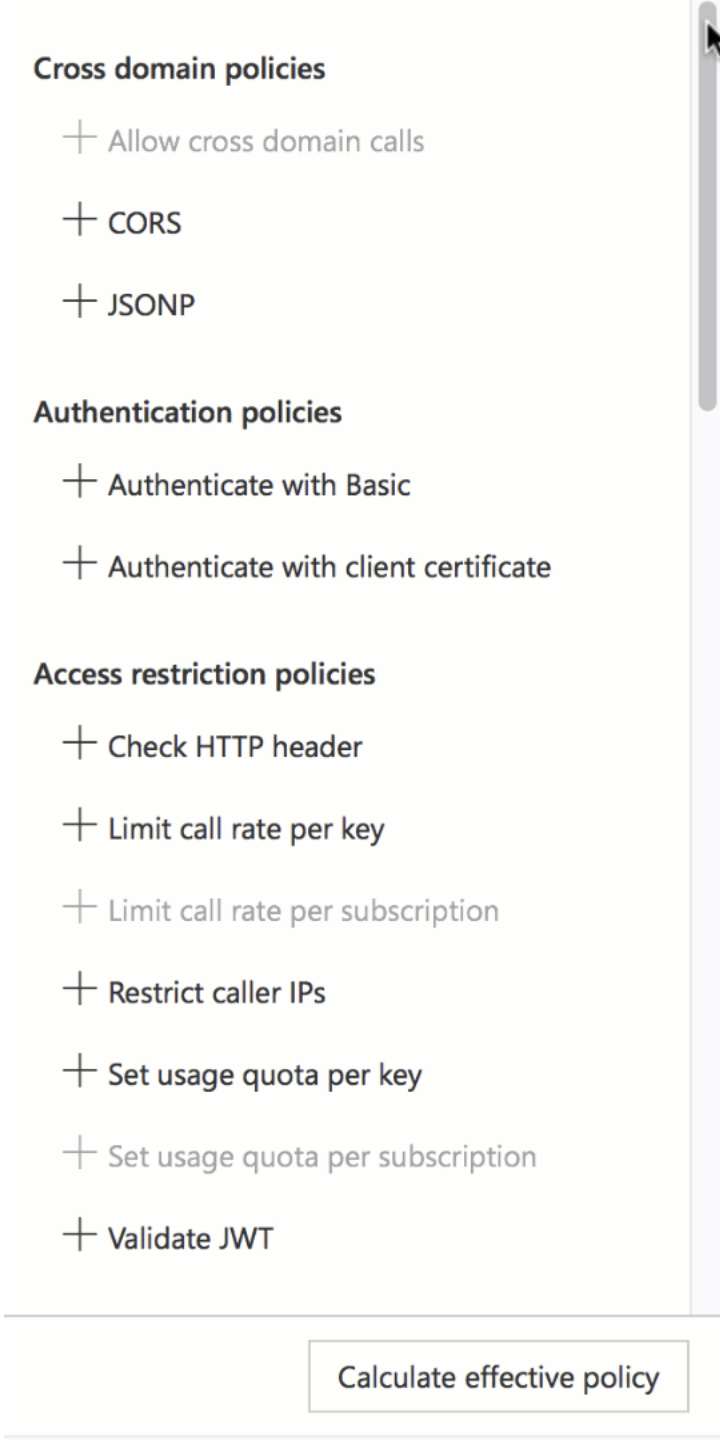
Scope determines which APIs are affected

Can define custom scopes in addition to four available b default

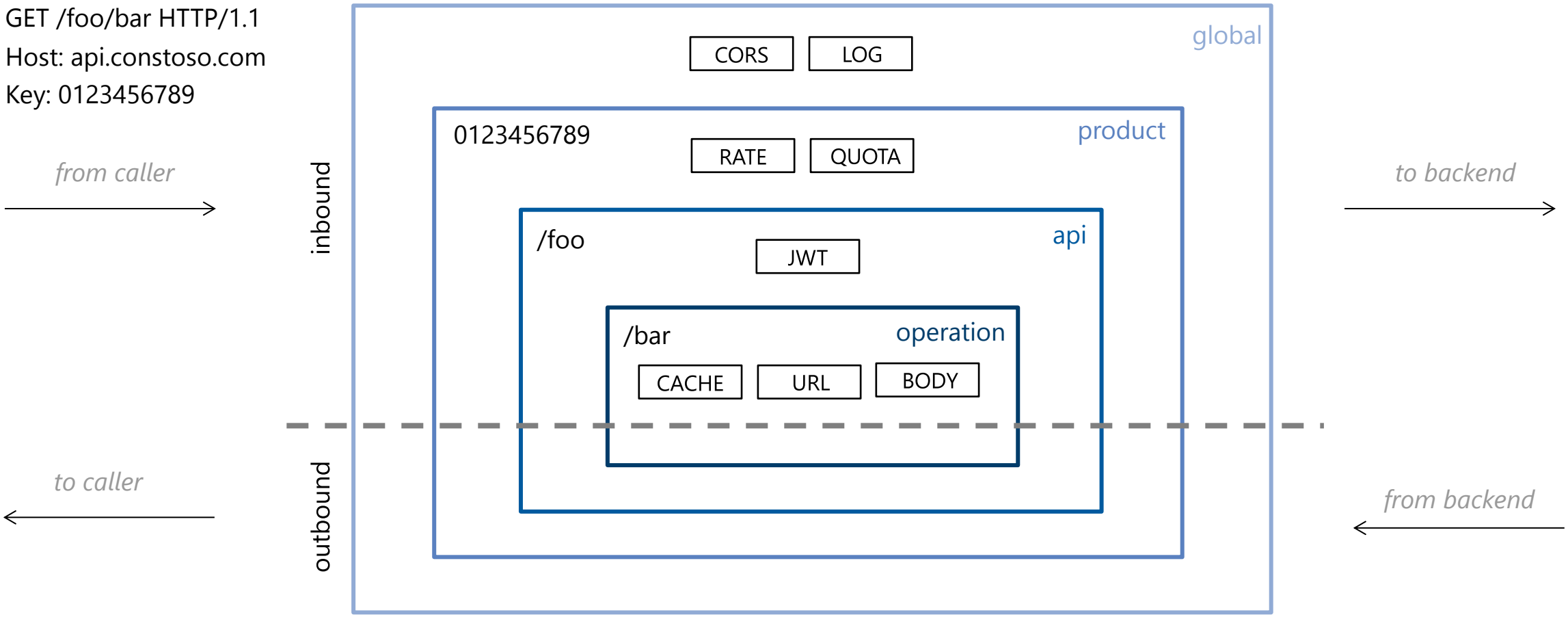
Composed into a pipeline from effective scopes

Degree of control over inheritance of scopes, i.e. <base/> element

Don’t delete <base/> inadvertently



Policy scopes



Policy expressions

Are C# “snippets” used with policies
Have read-only access to the request context
Can use only whitelisted .NET types
Are used to configure and conditionally execute policies

Named values

Are scoped to an APIM service instance
Keep secrets and “magic” strings out of policies
Provide environment-specific values
Add semantics, if named well
Enable a single point of change

```
...
<inbound>
    <base />
    <set-variable name="content-length" value="@(context.Request.Headers["Content-Length"][0])" />
    <choose>
        <when condition="@(int.Parse(context.Variables.GetValueOrDefault<string>("content-length")) > {{max-content-length}})">
            <rewrite-uri template="{{alternate-path-and-query}}" />
            <set-backend-service base-url="{{alternate-host}}" />
        </when>
    </choose>
</inbound>
...
```

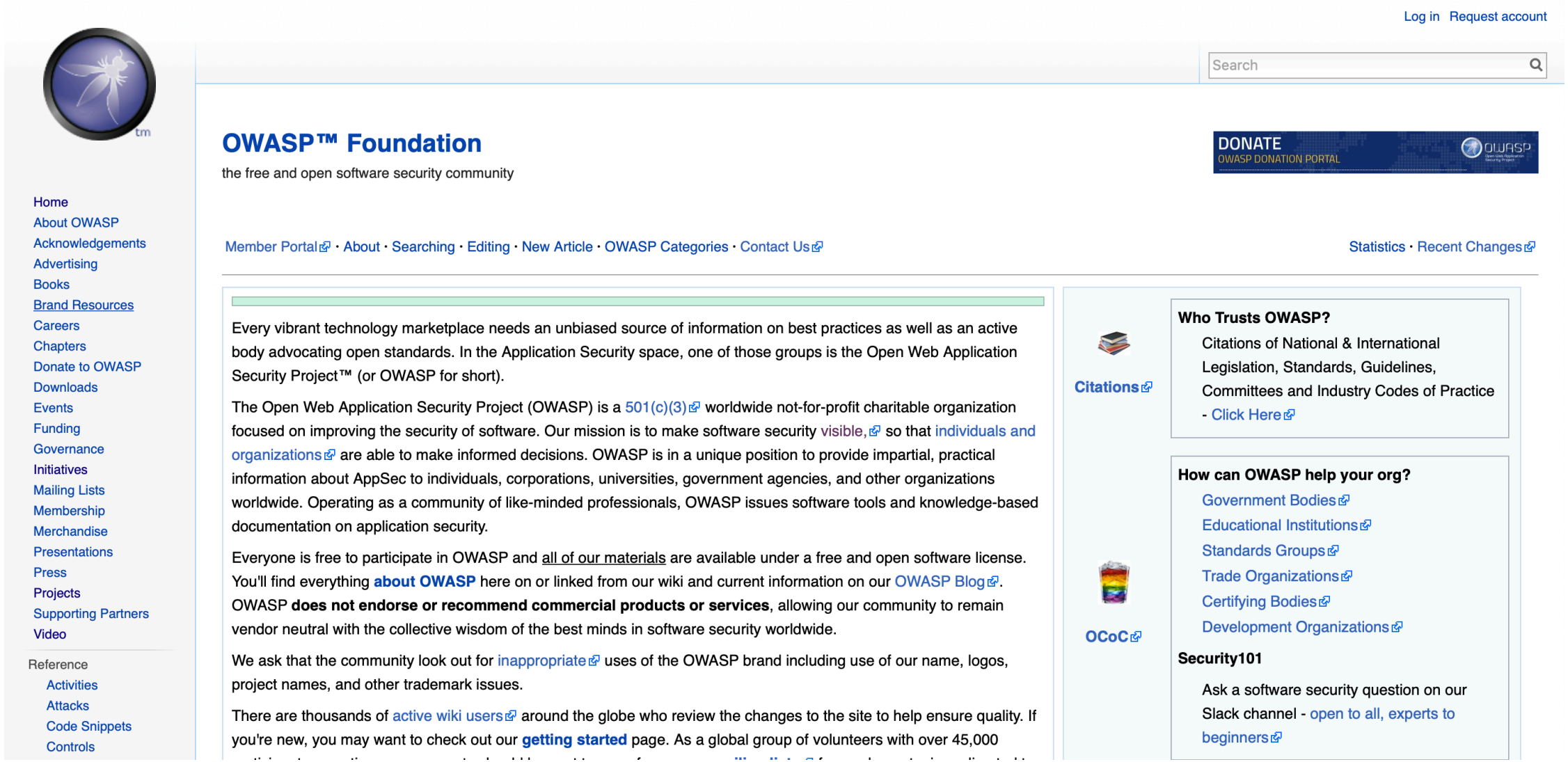
"By 2021, **90%** of web-enabled applications will have **more** surface area for attack in the form of exposed **APIs** rather than the UI, up from 40% in 2019."

"By 2022, **API** abuses will move from an infrequent to the **most-frequent attack vector**, resulting in data breaches for enterprise web applications."

Source: Gartner, Inc., API Security: What You Need to Do to Protect Your APIs,
Mark O'Neill, Dionisio Zumerle, Jeremy D'Hoinne,
28 August 2019

"The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted"

Learn more at https://www.owasp.org



API vs. web app security

#	OWASP Top 10 (2017)	OWASP API Top 10 (2019)
1	Injection	Broken Object Level Authorization
2	Broken Authentication	Broken Authentication
3	Sensitive data exposure	Excessive Data Exposure
4	XML External Entities (XXE)	Lack of Resources & Rate Limiting
5	Broken Access control	Broken Function Level Authorization
6	Security misconfigurations	Mass assignment
7	Cross Site Scripting (XSS)	Security misconfigurations
8	Insecure Deserialization	Injection
9	Using Components with known vulnerabilities	Improper Assets Management
10	Insufficient logging and monitoring	Insufficient logging and monitoring

API Management to the rescue

#	OWASP API Top 10 (2019)	Mitigations
1	Broken Object Level Authorization	*Area of investment*
2	Broken Authentication	Throttling; Request transformation; Token validation
3	Excessive Data Exposure	Filtering or masking sensitive data
4	Lack of Resources & Rate Limiting	Throttling; Limiting backend concurrency
5	Broken Function Level Authorization	Key/token/certificate-based authorization; Custom authorization
6	Mass assignment	*Area of investment*
7	Security misconfigurations	Restricting attack surface; TLS enforcement and configuration; CORS; Sanitization of response headers and error messages;
8	Injection	*Area of investment*
9	Improper Assets Management	Up-to-date API catalog; API lifecycle management
10	Insufficient logging and monitoring	Enforceable, customizable logging and tracing.

API Management to the rescue

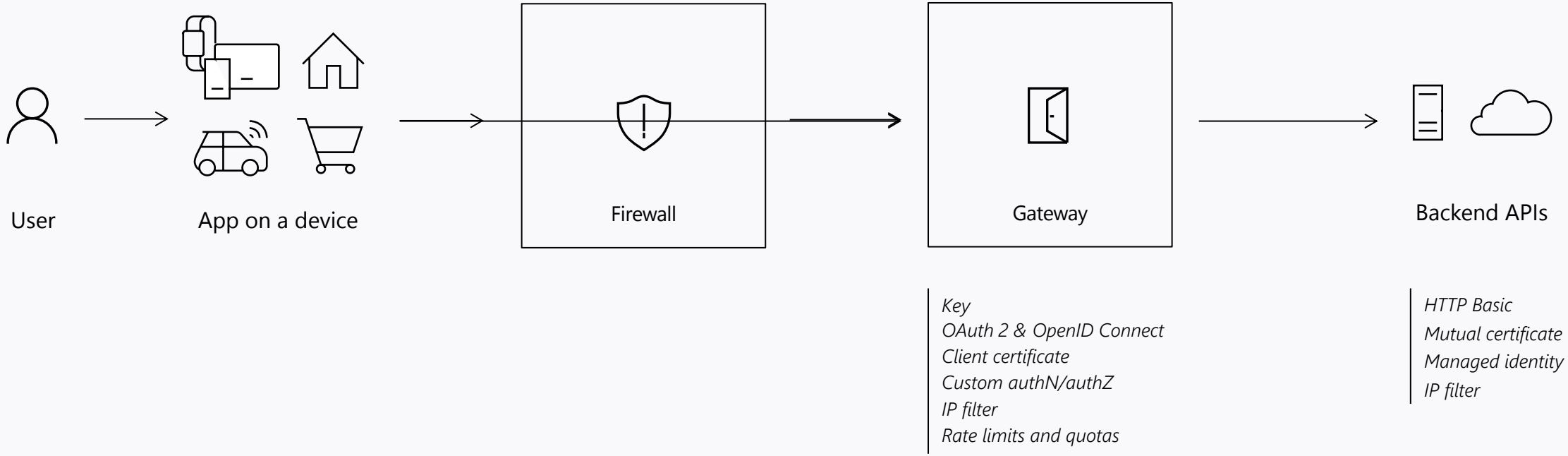
#	OWASP API Top 10 (2019)	Mitigations
1	Broken Object Level Authorization	*Area of investment*
2	Broken Authentication	**Throttling**; Request transformation; **Token validation**
3	Excessive Data Exposure	**Filtering or masking sensitive data**
4	Lack of Resources & Rate Limiting	**Throttling**; Limiting backend concurrency
5	Broken Function Level Authorization	**Key/token/certificate-based authorization; Custom authentication and authorization**.
6	Mass assignment	*Area of investment*
7	Security misconfigurations	**Restricting attack surface; TLS enforcement and configuration**; **CORS**; **Sanitization of response headers and error messages**
8	Injection	*Area of investment*
9	Improper assets management	Up-to-date API catalog; API lifecycle management
10	Insufficient logging and monitoring	Enforceable, customizable logging and tracing

Layered defense

Defense in depth options

Comprehensive authN/authZ in every layer

Separation of concerns between the layers



Disclaimers

Using API Management doesn’t automatically guarantee security

I am not a security expert

Façade

Expose specific backend resources

Limit HTTP methods on exposed resources

Enforce HTTPS and TLS configuration

CORS

Response sanitization

Filter or mask confidential data

Standardize error messages

Remove sensitive headers

Throttling

Rate limit

- Approximate
- Per region
- Key expression defines throttling semantics
- Can count requests with specific status code
- Variable increment count

Quota

- Calls and data transfer
- Approximate
- Per service
- Key expression defines throttling semantics
- Can count requests with specific status code
- Variable increment count

Concurrency limit

- Precise
- Per node

Keys

On by default – prevent access to APIs

UUIDs by default – hard to guess

Used to identify and authorize apps

Security-wise roughly equivalent to Basic

JWT

Signed (JWS) and encrypted (JWE)

Validate via policy and expressions

Enforce claims

Require signatures and expiration time

Provide keys inline or via a discovery endpoint

Key vs. JWT

	Key	JWT
Credential type	Bearer	Bearer
Token type	Reference	Self-contained
Size	Immaterial	Material*
Protection	Transport	Transport/Signature/Encryption
Scope	All-or-nothing	Fine-grained
Expiration	External	Built in
Revocation	Yes	Refresh tokens only*
Subject	App	App or User

Client certificates

Trusted and untrusted

Validate via expressions

Require certificate on per host basis

Check or ignore revocation lists

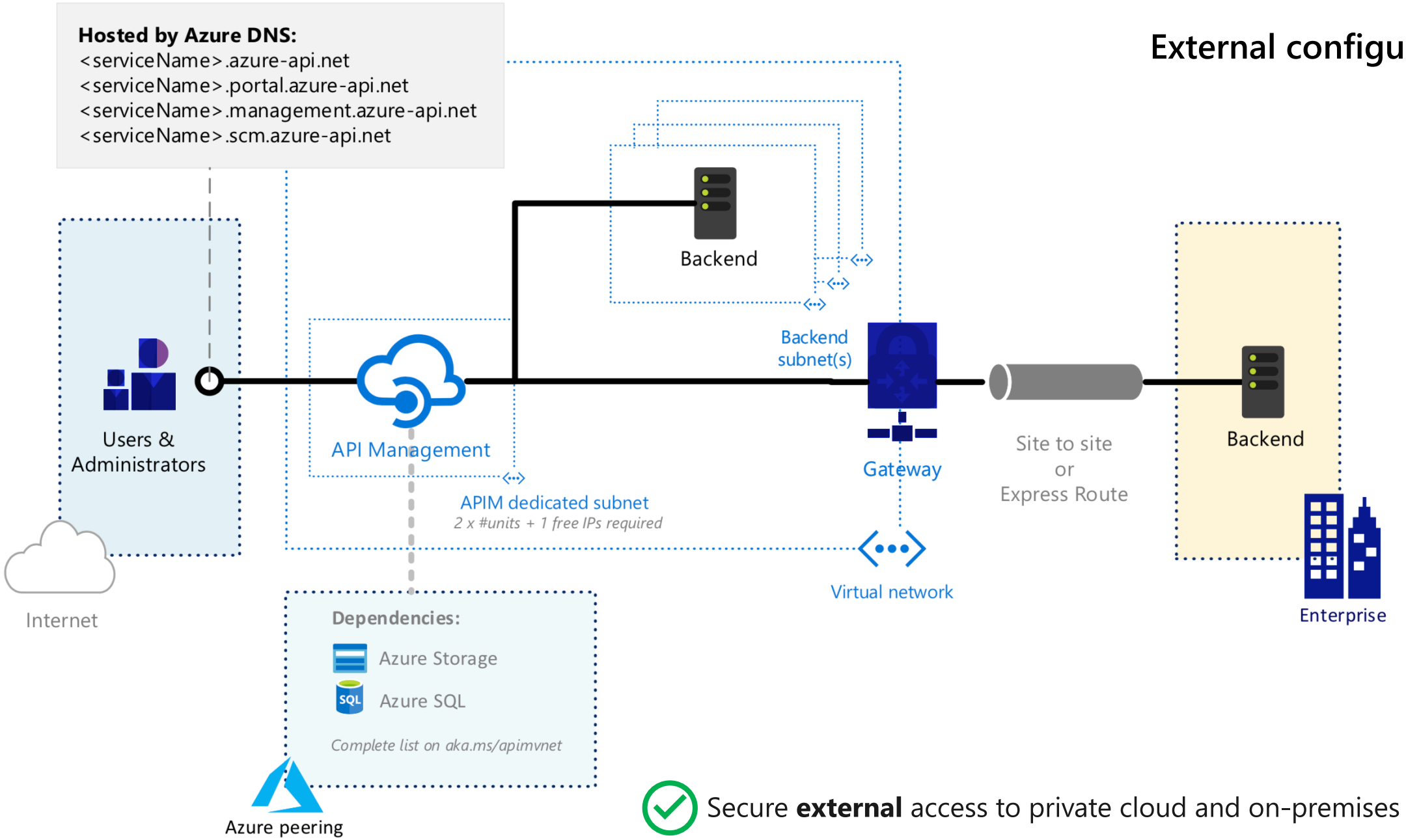
Custom authentication and authorization

Integrate with a bespoke or not supported identity or authorization system

Call out to an external HTTPS endpoint

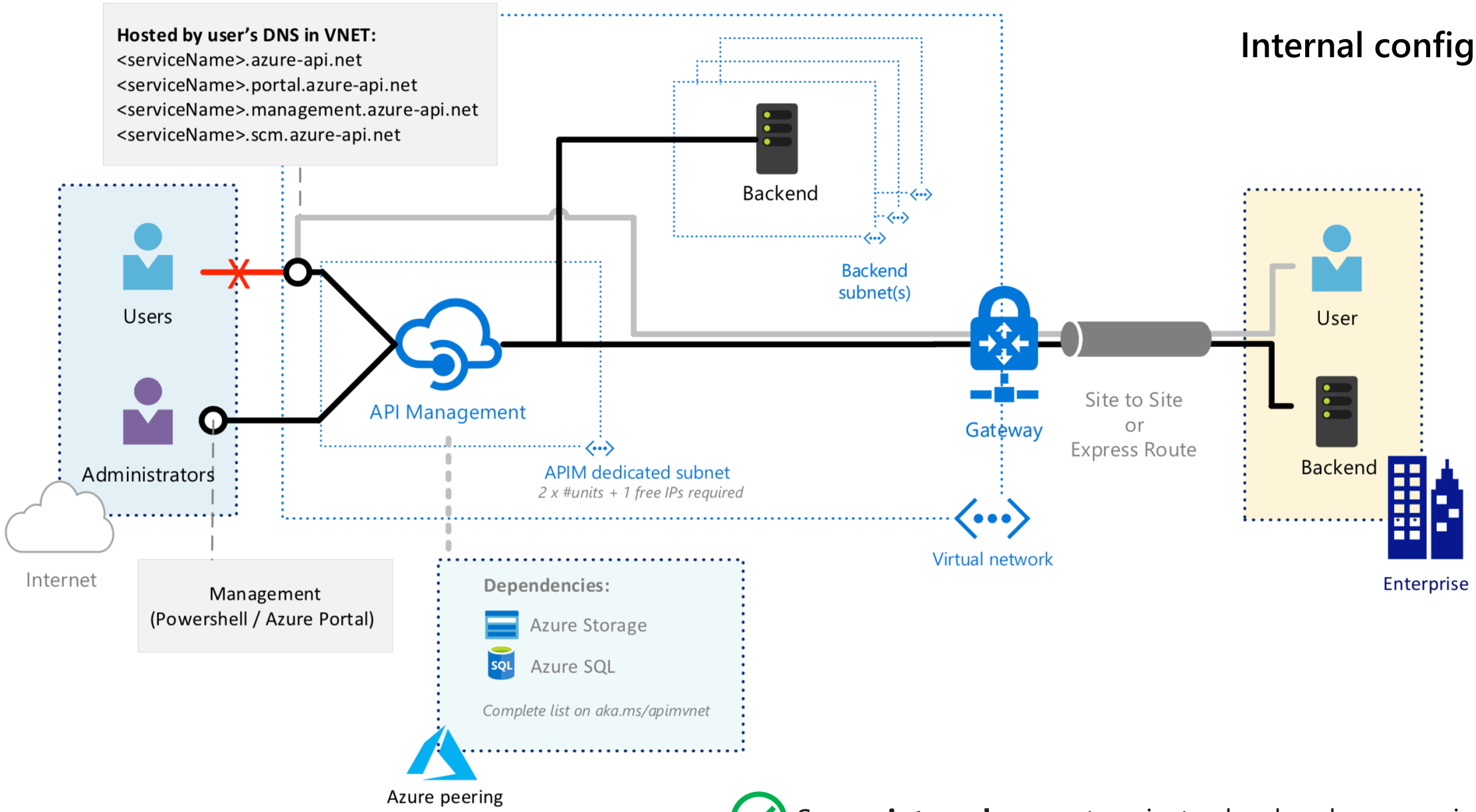
Cache the result for efficiency

External configuration



Secure **external** access to private cloud and on-premises endpoints

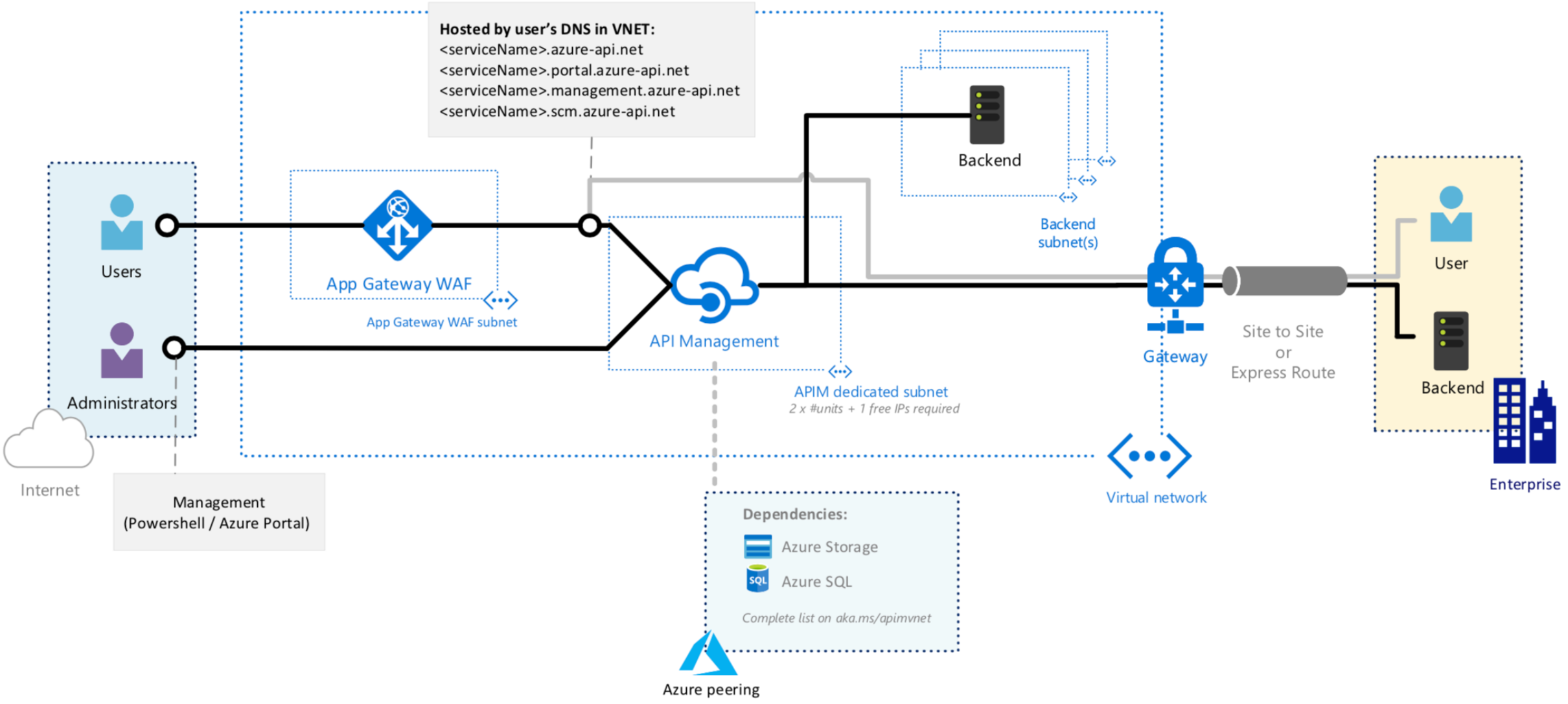
Internal configuration

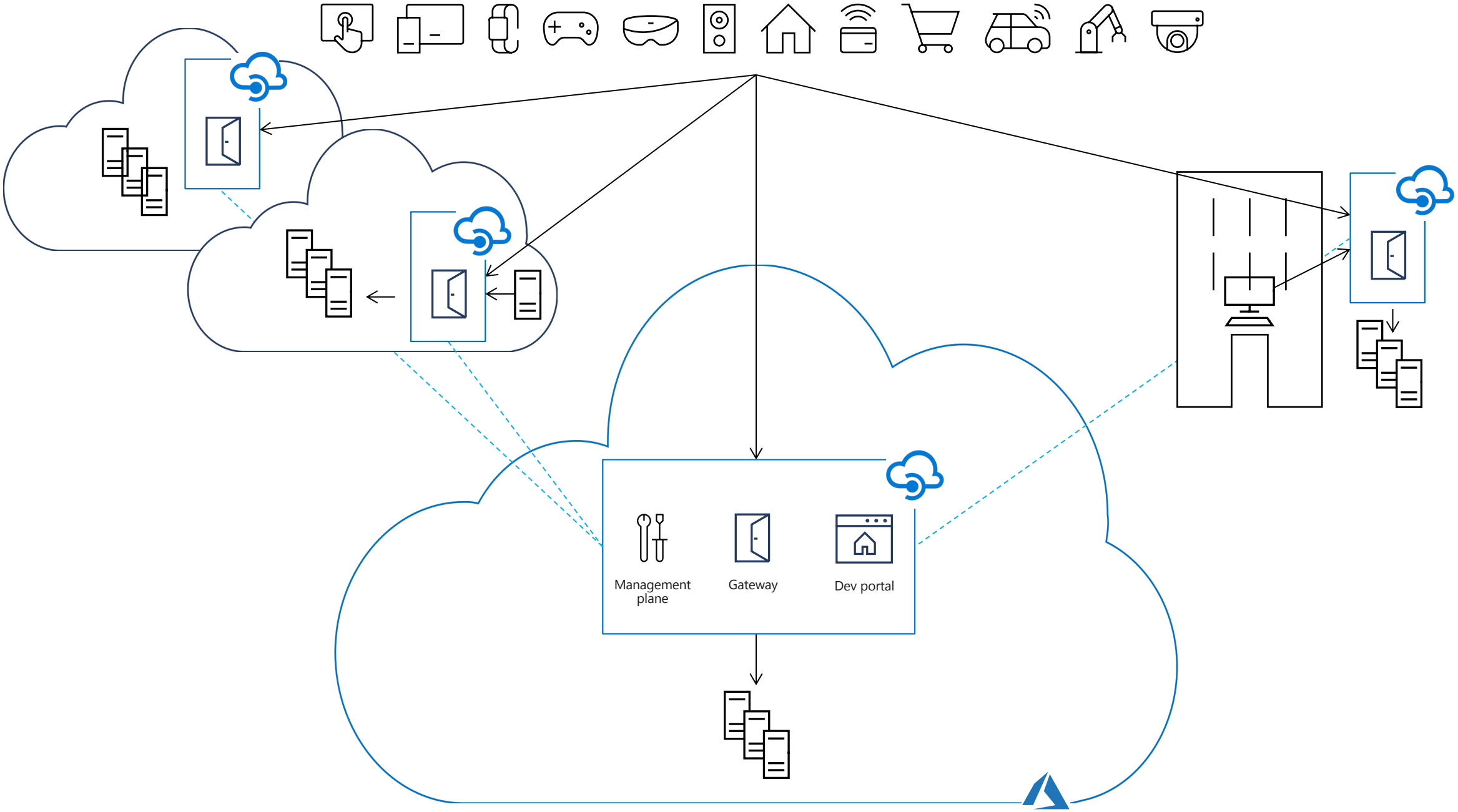


Secure **internal** access to private cloud and on-premises endpoints

Internal configuration with WAF

- More secure **external** access to private and on-premises endpoints
- Secure **internal** access to private cloud and on-premises endpoints




Management plane
Gateway
Dev portal

Self-hosted API Management gateway PREVIEW

Hosted on premises or in a cloud

Packaged as a Linux-based Docker container image

Functionally equivalent to the managed gateway

Federated with an API Management service

Gateway pulls down configuration and pushes up telemetry

Gateway requires only outgoing connectivity to Azure

Deployed to Docker or Kubernetes

K8s simplifies deployment, scaling, updates, availability

Other container orchestrators can be used as well

Information

https://aka.ms/apimlove