Application Modernization Guidance   
API First Architecture Patterns for Public Cloud PaaS

Version 1.1

**Table of Contents**

[Overview 1](#_Toc53751880)

[Patterns Built from Collaborations 3](#_Toc53751881)

[Policy References 4](#_Toc53751882)

[Security Architecture References 5](#_Toc53751883)

[APIs Delivered with Platform-as-a-Service 6](#_Toc53751884)

[Architecture Patterns and Anti-Patterns 7](#_Toc53751885)

[Anti-Pattern 1a: Batch Transfers 7](#_Toc53751886)

[Pattern 1b: Application Program Interfaces (APIs) 8](#_Toc53751887)

[Anti-pattern 2a: Direct Client-to-service Connection 9](#_Toc53751888)

[Pattern 2b: API Gateway (Reverse Proxy) 10](#_Toc53751889)

[Anti-pattern 3a: API Gateway and Network Firewall 13](#_Toc53751890)

[Pattern 3b: API Gateway with WAF 14](#_Toc53751891)

[Anti-Pattern 4a: Gated Operating Model for API Platform Services 16](#_Toc53751892)

[Pattern 4b: Operating Model for API Platform Services 17](#_Toc53751893)

[Alignment with GC Connectivity Scenarios 19](#_Toc53751894)

[Use Cases 20](#_Toc53751895)

[Use Case 1: SaaS to GC Cloud Hosted Application 20](#_Toc53751896)

[Use Case 2: GC Data Centre Hosted Application to GC Cloud Hosted Application 21](#_Toc53751897)

[Cloud Provider Solution Architectures 22](#_Toc53751898)

[Amazon Web Services 22](#_Toc53751899)

[Microsoft 24](#_Toc53751900)

[Oracle 24](#_Toc53751901)

[References 25](#_Toc53751902)

# Overview

In April 2020, the Government of Canada updated the Directive on Service and Digital to include mandatory procedures for Application Programming Interfaces (APIs). The focus on APIs is a natural one; a platform economy has grown out of externalizing application methods so that other platforms and automation can access those methods across the ubiquitous and widely supported http/s protocol. This provides developer with powerful architectures for services to communicate with each other using standardized technologies.

Just as the Graphical User Interface (GUI) is the method by which human users access applications, APIs are how other systems (services) access applications. APIs make automation possible.

The GC’s application architecture is shifting. Traditionally, our sensitive data and our systems were deployed within a perimeter that, for the, most part, was owned and operated by the GG and its contractors. System integration was often limited to a data centre or across the GC’s network.



Figure : Traditional view of application deployment where sensitive data is kept within the GC perimeter

The GC’s application architecture increasingly spans the internet. The introduction of cloud services and web-based data repositories has shifted the GC towards an architecture where services exchange data using the https protocol over the internet front-ended with API endpoints. The push towards digital services is causing the GC to adopt architectures that bring those services closer to where our users access them; the internet.



Figure : The GC's sensitive data resides on internet services

This guidance document is meant to provide developers and operations with a view of architectures that can be used to enable APIs. The focus is on OSI layer 7 technologies and the internet. A ubiquitous toolbox of services and capabilities have been made available by cloud service providers (as well as traditional IT providers) that allow APIs to orchestrate business flows without being concerned with deeper portions of the technology stack. Cloud service providers manage these services and offer them to consumers as a platform as a service model. These cloud-native platforms not only secure our data, but due to their scalability, add resilience and availability to our services while also providing a service model that allows architects and developers to focus on configuration choices and remain abstracted from underlying infrastructure engineering activities.

# Why an API-First Architecture?

An API-first approach is about focusing on the application layer for integrating applications and services across the internet and GC networks.

If we take a typical citizen interaction with a GC online service such as Employment Insurance thru a Service Canada portal, a user accesses the portal through a web browser over the internet. The connection between the client browser and the portal is secured via an encrypted https session and the user is authenticated. Sensitive information is accessed and updated over the internet.

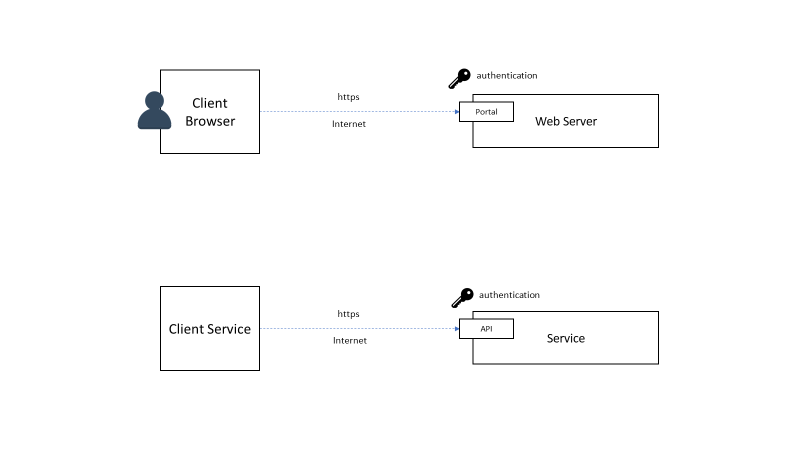


Figure : User-based access and update of GC data across the internet

APIs are endpoints that provide a similar experience for accessing and updating information in application. A client application or service takes the places of a user with a web browser. Instead of accessing a graphical interface through a portal, an API endpoint exposes functions for accessing and updating data. This is why an API-first approach to connecting cloud services over the internet is the preferred approach. This approach keeps the technology required to exchange data between services at the application layer using ubiquitous technologies such as the internet, https, and identity services.

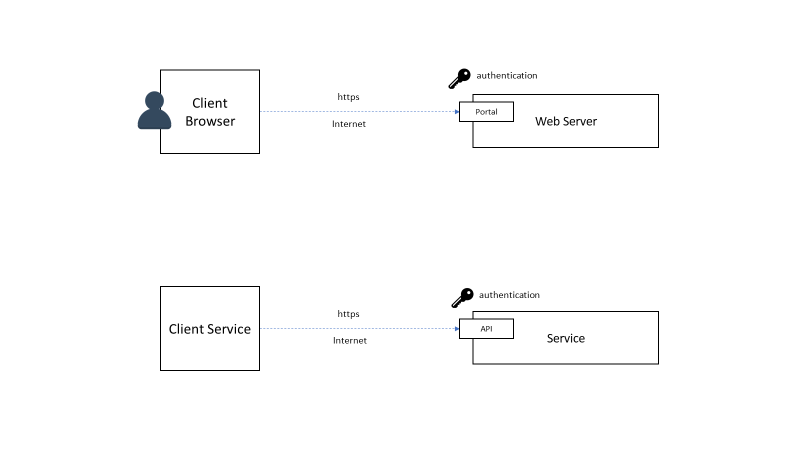


Figure : Service/application-based access and update of GC data across the internet

# Patterns Built from Collaboration

This document was built as a collaboration of the GC Cloud Technical Working Group in response to the interest shown by its members in establishing best practices for deploying API endpoints as well as clarifying, what, if any policy requirements would impact how those endpoints are deployed.

In the late Spring of 2020, a first draft was created by the Working Group. A draft of the document was posted on the Working Group’s GCCollaboration forum. By early summer, the document was distributed to Cloud Service Providers qualified on the Shared Services Canada Cloud Services Framework Agreement. Those providers were invited to present at a series of future Cloud Technical Working group Meetings where each provider would be given an hour to:

1. Provide their feedback on the architectural patterns for any correction and clarifications.
2. Provide a solution architecture using their services that align to the patterns provided in this document

Oracle, Microsoft, and AWS accepted the Working Group’s invitation to present. In late September and October 2020, those providers presented their feedback and solution architectures. A summary of those solution architectures are provided as an annex. These are not comprehensive architectures and should not be universally applied for every use case. Instead, you should further consult the providers’ documentation for a more comprehensive coverage of the subject.



In December 2020, the Cloud and Computing Network of Expertise provided and endorsement for this document. In late January 2021 the Enterprise Architecture Community of Practice endorsed this document. Final publishing was February 2021.

# Policy References

The Government of Canada has published mandatory procedures for Application Programming Interfaces (APIs). Appendix D of the Directive on Service and Digital include [Mandatory Procedures for Application Programming Interfaces](https://www.tbs-sct.gc.ca/pol/doc-eng.aspx?id=32604). These mandatory procedures focus on how APIs should be developed and delivered.

|  |
| --- |
| *B.2.2.2 Build APIs following the RESTful model by default. Representational State Transfer (REST) is effectively the standard for integration with cloud services and is also the standard set by the majority of other Governments with mature API programs.* |

# Security Architecture References

The document [Security Design Patterns for SaaS-based Information Systems](https://www.gcpedia.gc.ca/gcwiki/images/f/f4/GC_ESA_Security_Design_Patterns_for_SaaS-based_Solutions.pdf) includes a security overlay for APIs. The document focuses Software-as-a-Service, however the portions of the document that focuses on APIs can be applied to any service model that follows the same pattern. In effect, limiting service interactions to APIs, makes PaaS and IaaS services, outwardly facing, the same as SaaS-based API services. Note: the Secure Gateway in the diagram below are the Enterprise Internet Gateways hosted within the GC’s network.

A screenshot of a cell phone

Description automatically generated

Additionally, the document, [Micro-services and Container Security Guidelines](https://github.com/canada-ca/platform-security_securite-de-plateforme/blob/master/en/5_Microservice_Security.md#api-security) provides security requirements for protecting API end-points. Some of those same protections will be found throughout the patterns discussed in this document, however the guideline provides a consolidated and more complete list.

# API Endpoints Delivered with Platform-as-a-Service

This document focuses on API-first patterns using Platform-as-a-Services (PaaS). PaaS is the focus because, like SaaS, the network layers are not managed by the GC, but the cloud provider. It is the cloud provider who provides management and protection of the network layers of the service. Because the underlying network, compute, and storage is managed by the cloud service provider, the GC’s focus is on secure the applications and with it, the API endpoints.

This does not mean; however, IaaS does not factor into these patterns. PaaS can be used to provide a façade for IaaS-deployed applications. When a PaaS façade exists, the patterns and anti-patterns presented in this document still apply.

Figure : PaaS Providing a Facade for IaaS

Cloud Service

PaaS

IaaS

Private Network

API endpoint

Internet

# Architecture Patterns and Anti-Patterns

This document contains a collection of patterns and anti-patterns. As patterns can promote an optimal architecture, anti-patterns are equally useful as they help eliminate certain architectural choices and help illustrate weaknesses in those choices.

Each pattern and anti-pattern are presented with policy references and an architecture that can be applied to any cloud service provider that has the platform-as-a-service capabilities described in this document. Solution architectures are provided specific to each cloud service provider are provided at the end of the document. These solution architectures, however, are not comprehensive. The reader should always refer back to the cloud service provider’s own technical references for a more complete coverage.

## Anti-Pattern 1a: Batch Transfers

In this anti-pattern, information is aggregated in bulk and transferred in a single asynchronous transfer of transactions.



Figure : Ant-pattern - batch transfer

Why this is an anti-pattern?

Batch transfers do not provide a standardized method for accessing data in applications. While they have worked for decades, they are not optimized for an API-driven environment. They do not scale for an internet architecture. Authorization and authentication are often not standardized. Extract, transform, and load process are often needed and do not provide real-time data.

For these reasons, batch transfers are considered an anti-pattern.

## Pattern 1b: Application Program Interfaces (APIs)



Figure : Pattern - Application Program Interface

APIs provide a modern architecture for accessing methods and data within applications, services, and platforms. They are ubiquitous for internet scale services. Cloud providers provide a suite of platform services and architectures to support an API architecture. The Government of Canada has published mandatory procedures for API integration. Appendix D of the Directive on Service and Digital include [Mandatory Procedures for Application Programming Interfaces](https://www.tbs-sct.gc.ca/pol/doc-eng.aspx?id=32604)

|  |
| --- |
| *D.2.2.2 Build APIs following the RESTful model by default. Representational State Transfer (REST) is effectively the standard for integration with cloud services and is also the standard set by the majority of other Governments with mature API programs.* |

## Anti-pattern 2a: Direct Client-to-service Connection

In this anti-pattern, the client communicates directly with each service.



Figure : Anti-pattern - direct client to service

Why this is an anti-pattern:

In this anti-pattern, the client communicates with each service individually. Each service may require one or more of the following capabilities; authentication, authorization, billing, et cetera. In the anti-pattern above, these common capabilities have been integrated into the individual services. As a best practice, those common capabilities should be decoupled from business logic. Those capabilities are not part of the business logic being executed by the service, but instead is required for managing the interactions between the service and the client. The client may also be required to access multiple services with multiple unique resource locators (URLs). Providing a layer of decoupling between the client and services can allow greater flexibility such as adding backwards compatibility as API versions change.

## Pattern 2b: API Gateway (Reverse Proxy)

Common capabilities are aggregated into a single reverse proxy which is a facade for all services. The client does not interact directly with each service, but instead an API Gateway that is a reverse proxy for all services. Cloud Service Providers deliver API Gateway capabilities as a Platform-as-a-Service model where the underlying infrastructure including compute, storage, and networking, is managed by the cloud service providers. The pattern below assumes a PaaS deliver model for the API Gateway. Delivery as an IaaS service model is not recommended as it requires a larger operational burden including managing patching, scaling, and security of infrastructure components.

Appendix -B Mandatory Procedures for Application Programming Interfaces states:

|  |
| --- |
| *B.2.2.5.1Enforce secure communications by ensuring that sensitive data is never sent over an insecure or unencrypted connection, and where possible non-sensitive data should also be sent over a secure connection. Enable TLS 1.2 or subsequent versions, in accordance with CSE guidance.*  *B.2.2.5.6 Use gateways and proxies instead of whitelists when exposing APIs to the internet. Use a secure gateway layer to provide a security control point instead of simply whitelisting inbound Internet Protocol addresses (IPs). The API Store's gateway functionality may be used. When consuming external APIs, route flows through a forward (egress) proxy instead of using IP address whitelisting on the outbound firewall.*  *B.2.2.5.4 Protect access to APIs by implementing an access control scheme that protects APIs from being improperly invoked, including unauthorized function and data references. Always authenticate and authorize before any operation to ensure access to APIs are restricted to permitted individuals and/or systems. Use open standards such as OpenID Connect and Open Authorization 2.0 (OAuth 2.0) for RESTful APIs, and Security Assertion Markup Language 2.0 (SAML 2.0) for SOAP APIs. Ensure that the API key/secret is adequately protected. APIs must be secured with an API key to allow for usage tracking and provide the ability to identify and prevent potential malicious use. Open data APIs must be secured with an API key to allow for usage tracking and provide the ability to identify and prevent potential malicious use.*  *B.2.2.5.9 Log and monitor for performance and activity by tracking usage and monitoring for suspicious activity including abnormal access patterns such as after-hours requests, large data requests, etc. Use logging standards (e.g. common event format) and integrate logs centrally. Identify dependencies and monitor for vulnerabilities, especially those for uploaded run-times that work as part of the API. Suspicious events must be sent to the appropriate security operations capability or authority in compliance with Government of Canada Cyber Security policies and Government of Canada Cyber Security Event Management Plan.* |



Figure : Pattern - Reverse Proxy with API Gateway

API Gateways provide the following capabilities:

**Authentication** is the control of the connection attempt. The client provides credentials that are verified before access is granted. These credentials can be plain text (not recommended) or encrypted. OAuth 2.0 and SAML as both suggested in the mandatory procedures for APIs.

**Authorization** is the granting of access to service. Authentication verifies the client’s connection is allowed, authorization verifies which services and methods are allowed by the client.

**SSL termination** is the process of converting encrypted information from the client to plain text for processing.

**Routing** of client connection to different services based upon the URL provided.

For example, <https://domain.com/payment> will route client interactions to the payment service, while the <http://domain.com/cart> will route client interactions to the cart service. These services may be deployed using completely different architectures, but the client those complexities are abstracted as the API Gateway provides the routing required.

**Aggregation** of multiple service calls into one service call. This allows the amount of chatter between the client and services to be reduced.

**Caching** of commonly invoked methods can reduce the processing required by back-end services while improving response for clients. Cache expiry can be set to force a refresh of the cache.

**Load balancing** of service requests when the performance of a service is not within a specified window can be routed between multiple instances of that service. Load balancing is also an effective way of achieving blue/green deployments and canary deployments.

**Protocol conversion** allows clients to use a protocol, for example HTTP v1 to make a request even though the service only accepted HTTP v2.

**Monitoring telemetry** is a critical capability for monitoring an alerting based upon security, financial, and performance measures.

**Billing** for API usage or attributing costs back to users based upon usage is a common business model.

## Anti-pattern 3a: API Gateway and Network Firewall

In this anti-pattern all traffic is routed through a next generation firewall (deployed in an IaaS environment hosted on VMs) prior to being consumed by an API end-point on the API gateway.

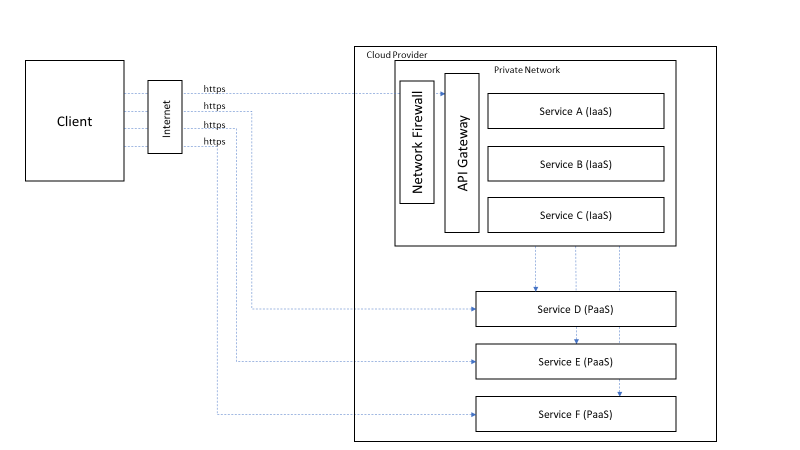


Figure : Anti-pattern - API Gateway with Network Firewall

Why this is an anti-pattern?

Network firewalls are designed to protect infrastructure such as servers, but not necessarily the applications or the clients. Network firewalls focus on OSI layer 3 and 4 traffic inspection and lack knowledge of HTTP/S and application specific threats such as SQL injection and Cross Site Scripting. Network firewalls support numerous protocols, however for APIs, the only protocol used is HTTP/S. DDoS protection is limited to protecting the network, not necessarily protecting the application’s availability. The capabilities of a next generation firewall are redundant with the services provided by a cloud service provider when an PaaS API gateway is deployed. The cloud service provider protects the network layers of the platform.

In a cloud environment, network firewalls are deployed as VMs in an IaaS service model. Those VMs are contained within a private subnet and are limited to a region unless multiple VMs are deployed across multiple regions. WAFs and API gateways are PaaS and can scale to be almost unlimited. This provides a higher level of resilience and availability than an architecture that includes firewall appliances. In a PaaS model, the cloud service provider provides the security protections needed at the network level. The GC’s focus is on securing the API endpoint and building applications that account for common threats such as SQL injection.

## Pattern 3b: API Gateway with WAF

The addition of a Web Application Firewall (WAF) provides services with higher availability while protecting those services and the clients from common threats. This pattern builds upon the authentication and authorization capabilities in an API Gateway with additional protections for the applications and services for which the gateway is a reverse proxy. Cloud Service Providers deliver WAF capabilities as a Platform-as-a-Service model where the underlying infrastructure including compute, storage, and networking, is managed by the cloud service providers. The pattern below would not apply when WAF capabilities are delivered as IaaS as the underlying virtual private network management remains with the GC.

Appendix B - Mandatory Procedures for Application Programming Interfaces states:

|  |
| --- |
| *B.2.2.5.2 Design APIs to be resistant to attacks by ensuring that all APIs are designed and implemented to be resistant to common API attacks such as buffer overflows and SQL injection. Treat all submitted data as untrusted and validate before processing. Leverage schema and data models for ensuring correct data validation.* |

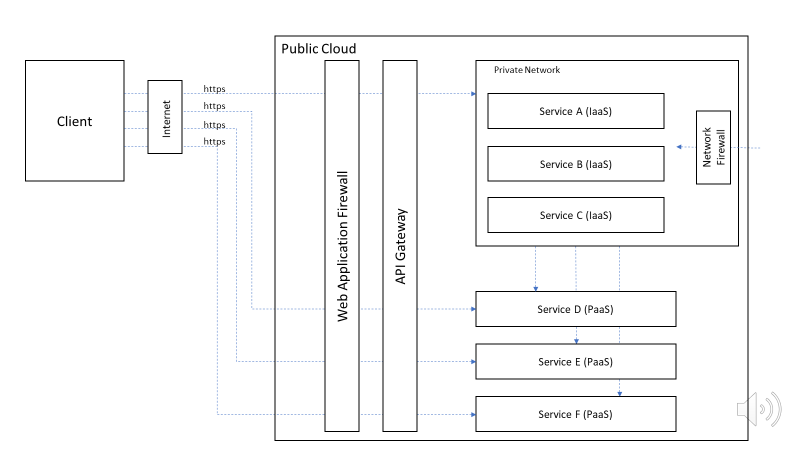


Figure : Pattern - Services behind a WAF and API Gateway

Capabilities of Web Application Firewall as focused on protecting against many of the [OWASP top 10 web security risks](https://owasp.org/www-project-top-ten/). A WAF can augment the protections of the API Gateway by providing a temporary method of resolving deficiencies found in application code. For example, if a legacy application is vulnerable to SQL injections, the WAF can be used to provide those protections until legacy applications code can be refactored. For that reason, a WAF is most effective in the hands of a developer who has knowledge of how the application works. Some would argue that with proper code testing a WAF is not needed as the vulnerabilities will be found and remediated with proper testing. WAFs, when delivered as a cloud PaaS, providers developers with the following capabilities:

* Protects against SQL injection attack.
* Protects against Cross-site scripting.
* Protects against session or parameter tampering.
* Allows developers to add inspections that are specific to an application
* Inspects SSL (HTTPS) encrypted traffic.
* Protects against application-layer denial and distributed denial of service (DDoS) attacks.
* Provides XML schema validation
* Blocks malicious XML content

## Anti-Pattern 4a: Gated Operating Model for API Platform Services

Operational models that introduce gates and hand-off of work between teams should be avoided. Traditionally, IT is organized into functional silos focused on specialization with hand-off between those teams to deploy a complete solution. Applying that traditional model to API delivery across API Gateway, WAF, and CDN platforms will slow the pace of delivery, complex service management processes, and people intensive delivery. As these capabilities are delivered through PaaS, concerns about sprawling instances of these capabilities is not an issue. Instead, the focus should be on managing the configurations of these platform capabilities.

A gated, waterfall, delivery model should be avoided.



## Pattern 4b: Operating Model for API Platform Services

The operating model for delivering API enabling platforms is critical to ensure developers can deliver new APIs and changes with speed while not compromising security, reliability, or quality.

The Mandatory Procedures for Application Programming Interfaces states:

|  |
| --- |
| *D.2.2.1.2 Empower staff to deliver better services by:*  *D.2.2.1.2.1 Ensuring the necessary tooling, training, and processes are in place to support a robust and agile API development and lifecycle management process.*  *D.2.2.1.2.2 Adopt Continuous Integration and Delivery (CI/CD) and Test Driven Development (TDD) practices supported by automation tools and integrated security testing. This provides the basis for DevOps adoption as maturity improves.* |

In a cloud environment, API gateway and WAF as platform services. The platform itself is maintained by the cloud service provider. Developers deploy configurations to the platform to support their applications, typically through a pipeline. Operations works with governance to deploy the policies required for all developers to follow as well as monitoring for high risk scenarios. These policies may remove high-risk configuration options. Examples of policies that may be enforced could include:

* All APIs need to be registered in an API gateway
* All API gateways need to be connected to a WAF
* All APIs require authentication (perhaps a specific authentication technology)
* All APIs must use the https protocol
* All WAFs must include a key set of rules

The depiction below should not be thought of as ‘instances’ of common capabilities for each application, but instead a common set of platforms. Each application may require a unique configuration of the platform tuned for its purpose. Additionally, operations may implement a common base configuration across all platforms.

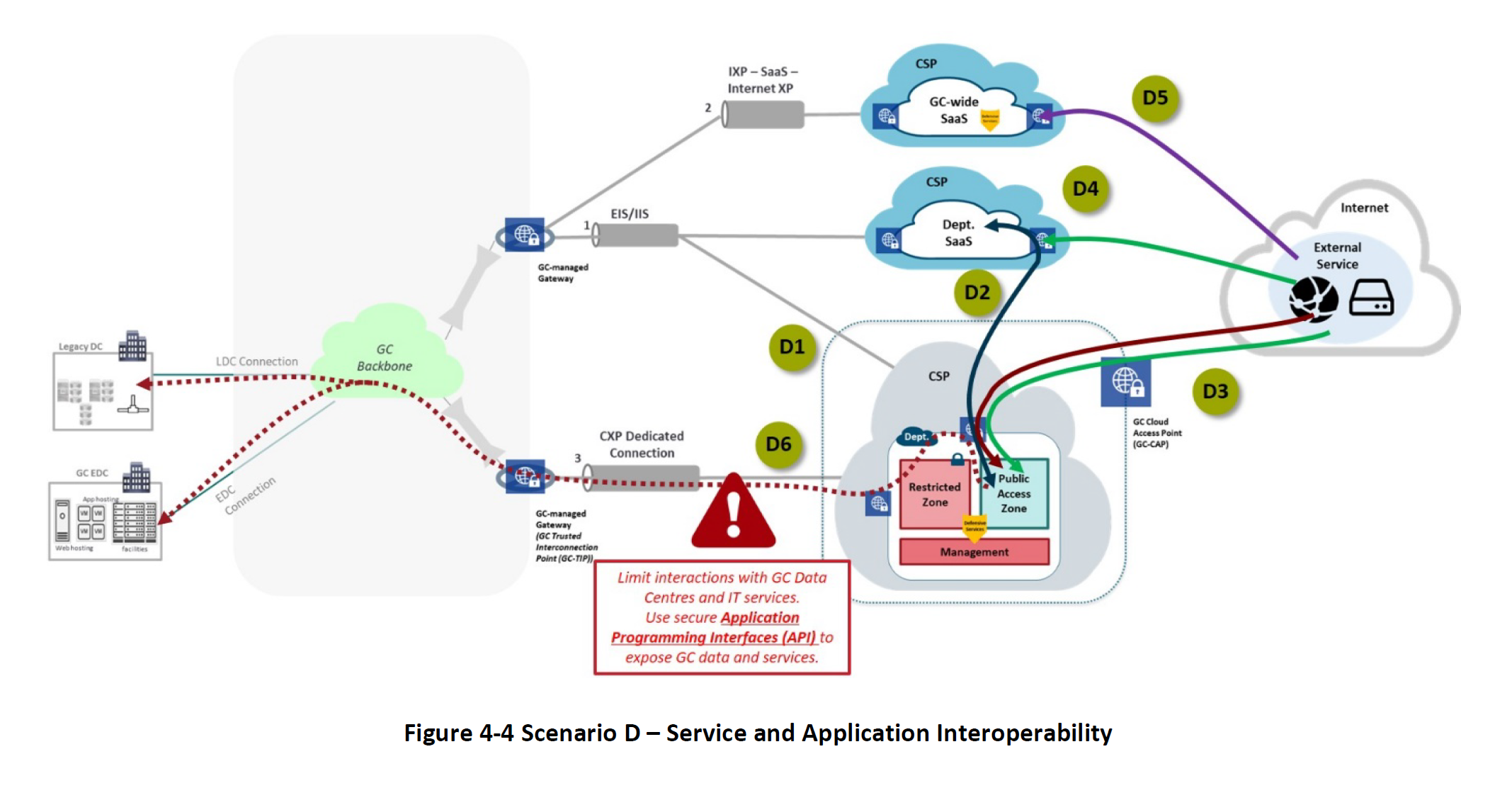


# Alignment with GC Connectivity Scenarios

To help GC departments describe their connectivity patterns, GC EARB approved a set of [connectivity scenarios](https://www.gcpedia.gc.ca/gcwiki/images/e/e7/GC_Secure_Cloud_Connectivity_Requirements.pdf). Hybrid IT application or service integration (cloud & on-prem) falls under scenario D. For those orchestrating API interactions over the internet, between public facing, but secured, APIs on-prem and those in the cloud, D1 is applicable as all traffic traverses the GC EIS. For those orchestrating the same scenario over the CXP, scenario D6 applies.

The Cloud Access Point (CAP) is built for IaaS scenarios as PaaS is out of scope for the Secure Cloud Enablement and Defense (SCED) initiative. As API endpoints rely on API gateway capabilities such as IP filtering, but more importantly, identity to authenticate and authorize access to the end point, routing traffic through the CAP is not necessary. Using Platform-as-a-Service deployment models put the network layer of the API gateway and WAF under the management of the Cloud Service Provider. At the time of writing this document, the following minimum services are recommended:

* Access control
* Logging & monitoring
* Auditing
* Cyber defense services (CCCS CBS/HBS)
* *Additional services may be required based on risk profile (e.g. if Internet flows are enabled)*

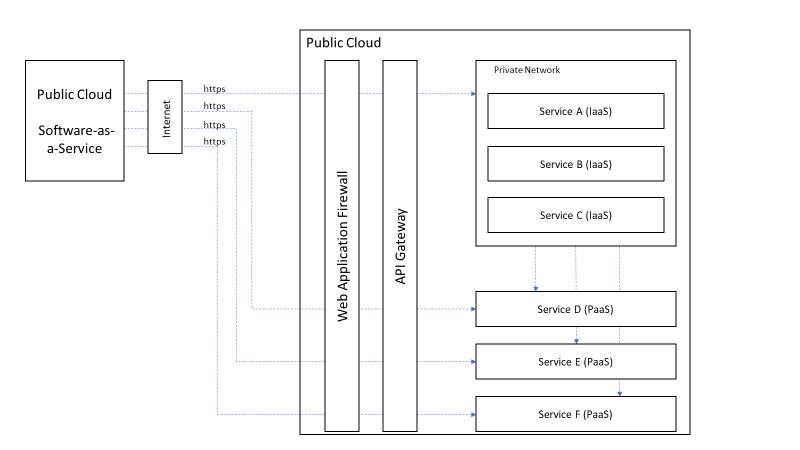


# Use Cases

## Use Case 1: SaaS to GC Cloud Hosted Application

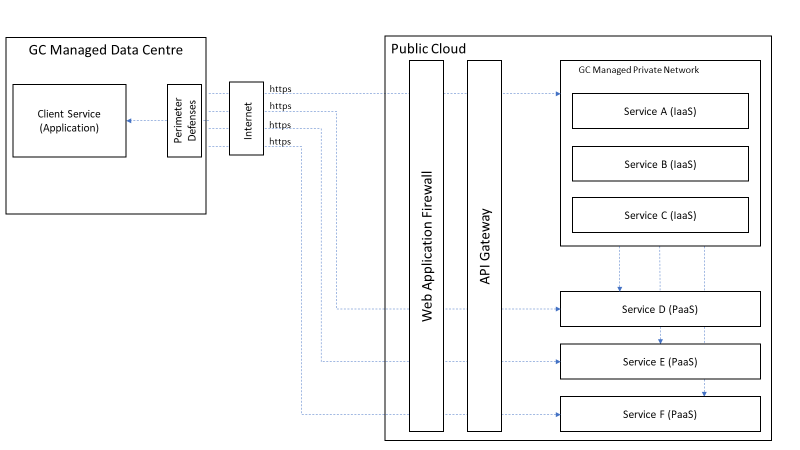
Public cloud software-as-a-service methods can be invoked externally through APIs. Similarly, the SaaS may need to invoke external APIs to complete processing. For example, in a workflow, information from another system of record may be required to complete processing.

In this scenario, the SaaS is invoking methods across multiple services hosted by the GC in a public cloud environment. Those services can be served up through IaaS solutions, for example VMs running a COTs product, or using PaaS services such as a microservices application architected using a function-as-a-service platform.



## Use Case 2: GC Data Centre Hosted Application to GC Cloud Hosted Application

In this scenario an application hosted in a GC-managed data centre and needs to invoke methods of an application/service hosted in a public cloud and vice versa. This is a Hybrid IT use case where the only connectivity between applications is API-based over https. An API gateway and a WAF are required both in the data centre and the public cloud.



# Cloud Provider Solution Architectures

## Amazon Web Services

Amazon Web Services (AWS) presented their generic architecture for API-first architecture patterns. Then further provided an example pattern and anti-pattern. AWS suggests the following [whitepaper](https://d1.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf) to further explain how AWS’ service can enable API-first patterns.

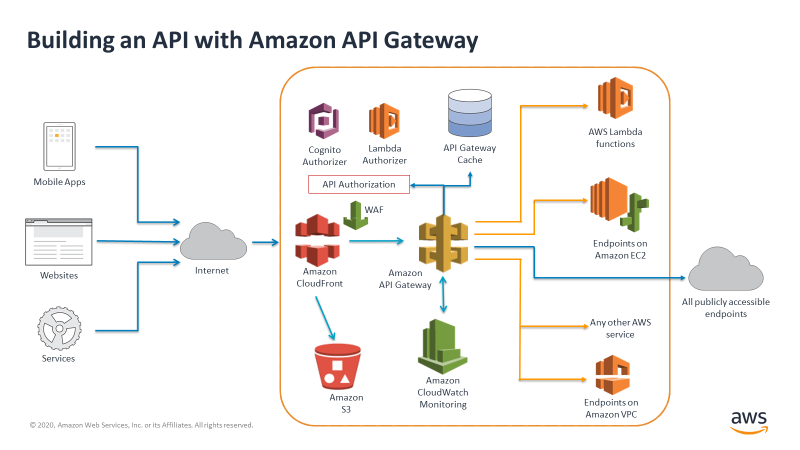


Figure : AWS 'generic' approach to API-first architectures

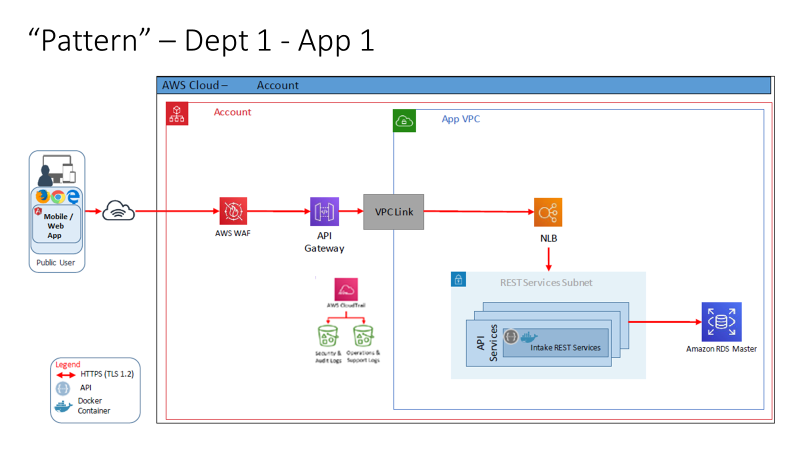


Figure : AWS example of an API-first pattern

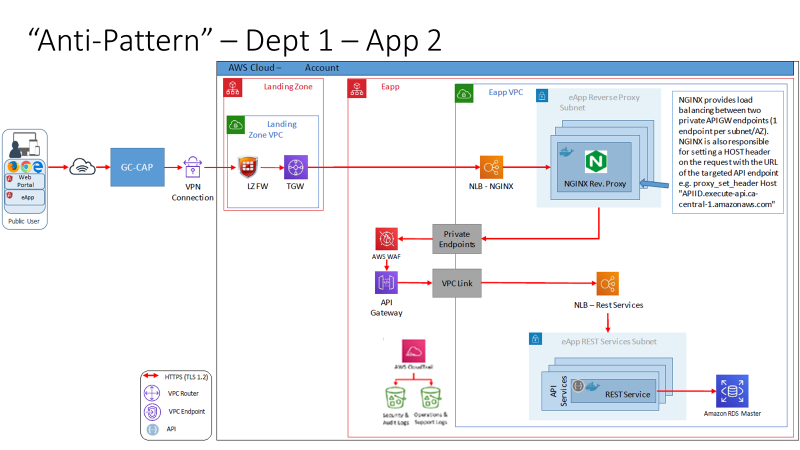
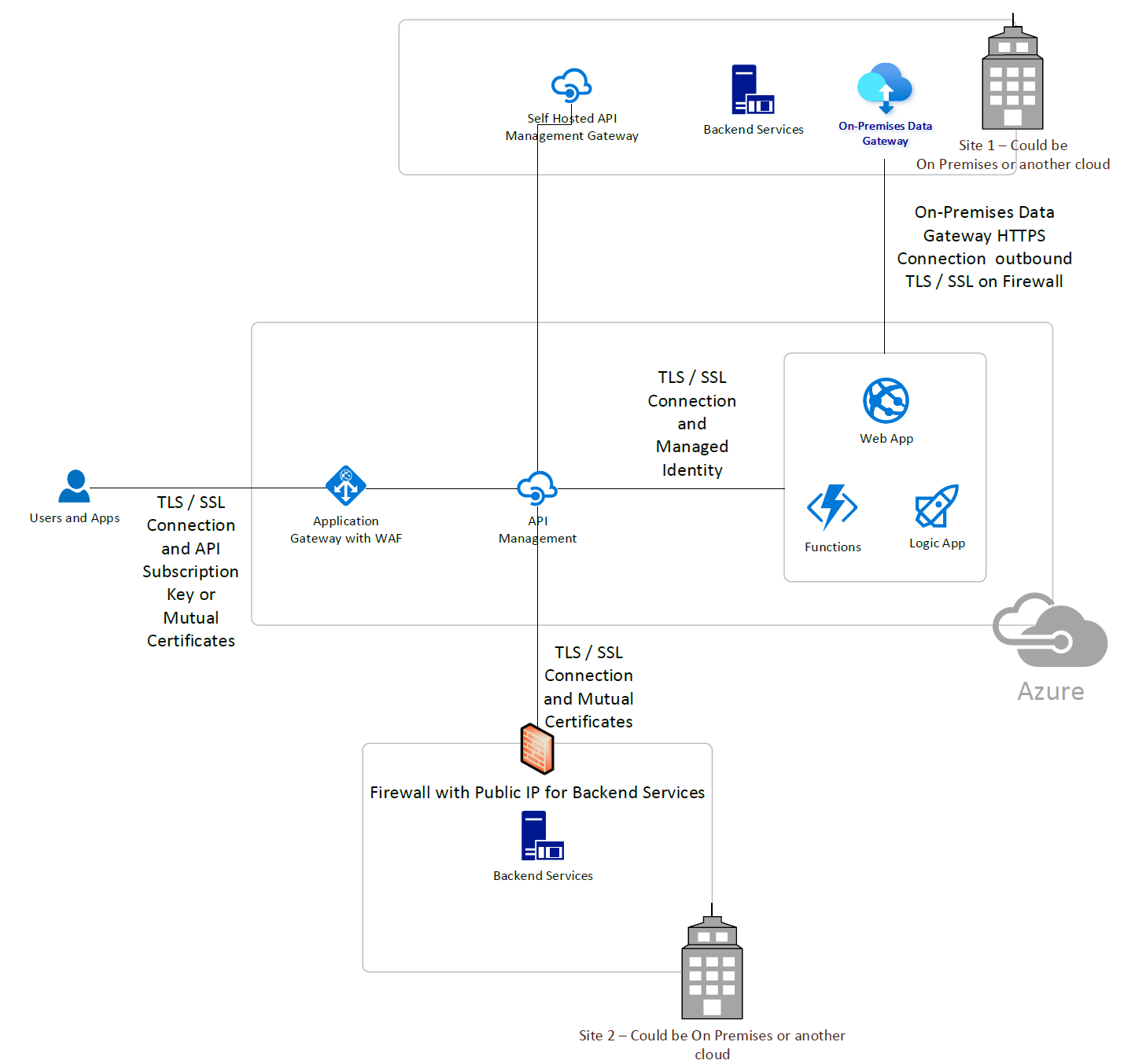


Figure : AWS Example of an Anti-pattern due to complexity and lower availability

## Microsoft

Microsoft offered its solution architecture for API-first patterns. Microsoft offers a product to be installed on-prem called ‘Data Gateway’ that connects directly with Web Apps. Alternatively, the more standardized pattern of deploying an API Gateway on-prem can also be used for hybrid IT models.



## Oracle

Oracle presented their generic solution architecture for API first patterns. They have also create a whitepaper [Oracle Cloud Infrastructure Serverless Multi-Tier Architectures](https://056gc-my.sharepoint.com/personal/slevac_tbs-sct_gc_ca/Documents/Oracle%20Cloud%20Infrastructure%20Serverless%20Multi-Tier%20Architectures) (https://bit.ly/2TmHhku)

There is also a lab provided (https://bit.ly/3moNpoy) that provides a step by step guide on implementing the architecture described in the document.

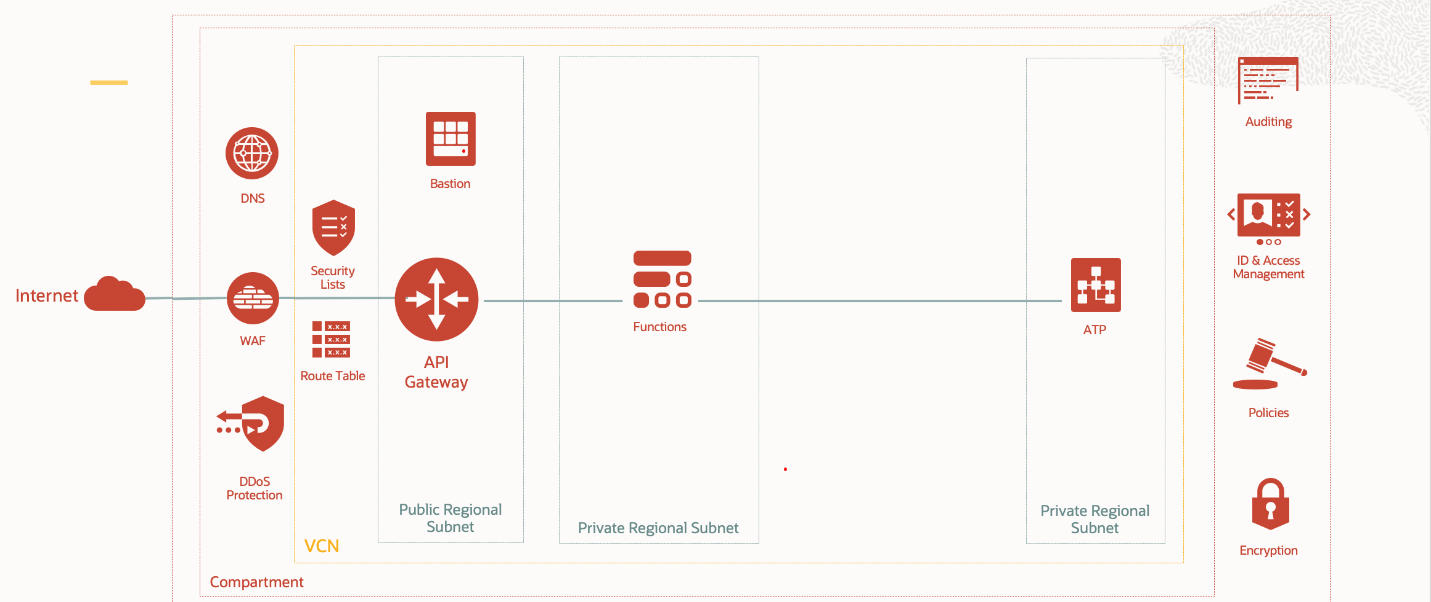


Figure : Oracle OCI API First solution architecture

# References

[The API Gateway Pattern](https://manningbooks.medium.com/the-api-gateway-pattern-cd8af792084)

[Mandatory Procedures for Application Programming Interfaces](https://www.tbs-sct.gc.ca/pol/doc-eng.aspx?id=15249&section=procedure&p=D)

[GC Secure Cloud Connectivity Requirements](https://www.gcpedia.gc.ca/gcwiki/images/e/e7/GC_Secure_Cloud_Connectivity_Requirements.pdf)

[Security Design Patterns for SaaS-based Information Systems](https://www.gcpedia.gc.ca/gcwiki/images/f/f4/GC_ESA_Security_Design_Patterns_for_SaaS-based_Solutions.pdf)

[Micro-services and Container Security Guidelines](https://github.com/canada-ca/platform-security_securite-de-plateforme/blob/master/en/5_Microservice_Security.md#api-security)

Application Modernisation Guidance: [Building Modern Applications Using Platform Services](https://wiki.gccollab.ca/images/b/bc/01_Building_Modern_Applications_Using_Platform_Services_EN.pdf)