

FORUM ON TAX ADMINISTRATION

Unlocking the Digital Economy - A Guide to Implementing Application Programming Interfaces in Government



Unlocking the digital economy

A guide to implementing application programming
interfaces in government

This document, as well as any data and any map included herein, are without prejudice to the status of or sovereignty over any territory, to the delimitation of international frontiers and boundaries and to the name of any territory, city or area.

Please cite this publication as:

OECD (2019), Unlocking the Digital Economy - A guide to implementing application programme interfaces in Government, OECD, Paris.

www.oecd.org/tax/forum-on-tax-administration/publications-and-products/unlocking-the-digital-economy-guide-to-implementing-application-programming-interfaces-in-government.htm

Photo credits: Cover © CoreDESIGN – Shutterstock.com

© OECD 2019

You can copy, download or print OECD content for your own use, and you can include excerpts from OECD publications, databases and multimedia products in your own documents, presentations, blogs, websites and teaching materials, provided that suitable acknowledgment of the source and copyright owner(s) is given. All requests for public or commercial use and translation rights should be submitted to rights@oecd.org. Requests for permission to photocopy portions of this material for public or commercial use shall be addressed directly to the Copyright Clearance Center (CCC) at info@copyright.com or the Centre français d'exploitation du droit de copie (CFC) at contact@cfcopies.com.

Preface

As tax administrators we must continue to evolve in order to meet client expectations for seamless service delivery whilst also protecting the integrity of revenue systems. As public servants and community leaders we also have a broader opportunity to make positive contributions to Government and community ecosystems that may extend beyond our traditional tax administration roles. The need for us to contribute to the broader ecosystem is reinforced by the fact that our clients operate across the boundaries of different government services, across public and private sectors and across government jurisdictions.

The digitisation of our economies allows us to now imagine a world where “tax just happens” as a by-product of people going about their normal business. It allows us to move beyond a siloed, period based and retrospectively audited reactive world. We can now begin to craft a new reality where services are seamlessly integrated and where the integrity of the broader system is assured within these integrated services.

Our ability to create a world where “tax just happens” is increasingly dependent on “machine to machine” Application Programming Interfaces (APIs). These APIs allow our revenue systems to digitally interact with other Digital Service Providers (DSPs) including banks, accounting software providers and other government agencies. We can use APIs to send and receive information, to validate activities, to facilitate transactions and to impose behavioural nudges close to real time.

This paper briefly explores the business context for APIs before exploring a broad range of implementation issues associated with APIs. Where possible, this paper has drawn on some of the most commonly implemented industry standards at the time of publication and we hope that any agency, regardless of where they are on their API implementation journey, can benefit from the concepts explored within this paper.

I would like to acknowledge the contributions of participating jurisdictions for providing insights and implementation examples of APIs. The ability to build on experience and application across a wide range of use cases has enabled us to develop a robust guidance paper with extensive relevance to agencies across the globe.

Finally, I would like to recognise the efforts of my staff at the Australian Taxation Office. There has been a great deal of collaboration between the ATO and the hundreds of organisations that interact with our APIs. This paper is a reflection of the many learnings that have come from that journey.

Chris Jordan

Commissioner of Taxation, Australian Taxation Office

Table of contents

Abbreviations and acronyms	7
Executive summary	9
Chapter 1. Introduction	11
1.1. Why APIs?	11
Chapter 2. Technical concepts	17
2.1. A brief history of APIs	17
2.2. API Management.	18
2.3. Architectural approaches	21
2.4. API security practices and controls.	37
2.5. API consumer experience	44
2.6. API measurement and reporting	46
2.7. API delivery techniques and toolsets.	47
Chapter 3. Future applications of APIs	51
3.1. A transport management company executes a payroll event	51
3.2. A transport management company makes a contract payment	52
3.3. A transport management company attempts a contract payment.	53
Annex A. ATO Lessons Learned	55
Annex B. Glossary of terms	59
References	65
 Figures	
Figure 1.1 The difference between partially connected and fully connected ecosystems	11
Figure 1.2 Attributes of paper, electronic and digital ecosystems	12
Figure 1.3 The growth of digital “event based” reporting	13
Figure 1.4 ATO transition from paper to digital (from the reinventing the ATO blueprint)	15
Figure 2.1 Connections across the ecosystem	17
Figure 2.2 Extract from the MuleSoft End to End Lifecycle for Microservices	20
Figure 2.3 Peer-to-peer model	21
Figure 2.4 The four-corner model	22
Figure 2.5 Client-server model.	23
Figure 2.6 Monoliths and microservices	24
Figure 2.7 The API gateway pattern	27

Figure 2.8	Event-Driven Architecture	28
Figure 2.9	Liability vs Return On Investment (ROI) Analysis	37
Figure 2.10	OpenID Connect Protocol Suite	39
Figure 3.1	A transport management company executes a payroll event	52
Figure 3.2	A transport management company makes a contract payment	53
Figure 3.3	A transport management company attempts a contract payment.	54

Tables

Table 1.1	Comparison of machine to machine (M2M) and human processing	14
Table 2.1	Peer-to-peer application technologies and standards	22
Table 2.2	Commonly used HTTP methods, purpose and classification.	31
Table 2.3	Fundamental technology standards	32
Table 2.4	Messaging standards.	32
Table 2.5	WS-* standards	34
Table 2.6	Primary data standards.	35
Table 2.7	Authentication and authorisation standards	38
Table B.1	Definitions for key terms used throughout the document	63

Box

Box 2.1	Example of exchanging SMS data	23
---------	--	----

Abbreviations and acronyms

ABN	Australian Business Number
ABR	Australian Business Register
API	Application programming interface
ASF	Apache Software Foundation
CC	Creative Commons
COTS	Commercial off the shelf
DOS	Denial of service attack
DSP	Digital Service Providers
ESB	Enterprise service bus
EVTE	External vendor test environment
HATEOAS	Hypermedia as the Engine of Application State)
HTTP	HyperText Transfer Protocol
IAAS	Infrastructure as a service
ISO	International Organisation for Standardisation
IETF	Internet Engineering Task Force
JSON	JavaScript Object Notation
JWA	JSON Web Algorithms
JWE	JSON Web Encryption
JWK	JSON Web Key
JWS	JSON Web Signatures
JWT	JSON Web
MAC	Message authentication code
MIT	Massachusetts Institute of Technology
OASIS	Organisation for the Advancement of Structured Information Standards
OAS	Open API Specification
PEPPOL	Pan-European Public Procurement On-Line
OWASP	Open Web Application Security Project
PAAS	Platform as a Service

PII	Personally identifiable information
PKI	Public Key Infrastructure
RFC	Request for Comments
SAAS	Software as a service
SAML	Security Assertion Markup Language
SSL	Secure Socket Layer
SLA	Service Level Agreement
SOA	Service-oriented architecture
SOAP	Simple object access protocol
SDK	Software development kit
TLS	Transport layer security
W3C	World Wide Web Consortium
WSDL	Web services description language
XBRL	eXtensible Business Reporting Language
XML	eXtensible Markup Language
XSD	XML Schema Definition

Executive summary

The advent of the World Wide Web has revolutionised the way business is being conducted. Organisations can now leverage off the data that flows through their “natural systems” and connect to other services such as government infrastructure. Application Programming Interfaces (APIs) allow this connectivity between systems, people and things without facilitating direct access.

There are many opportunities to be realised through the use of APIs. Information can be collected as a primary function of an API or as a by-product. This collection of data is often rules based. When an event occurs, it triggers the push or pull of information from a data source. If calculations are required, this can be built into the API functionality. The data can be validated through the API before being posted to a back end system.

With the correct rules and validations in place, seamless interaction between multiple touch points simultaneously allows “tax and other government obligations” to just happen.

Benefits of creating a digital ecosystem through APIs include:

- fostering a self-regulating economy, making it harder for people to operate outside the system
- streamlining and automating government interactions
- moving past use of electronic forms to true digital interactions.

Good API management varies by organisation, however there are two factors these organisations will have in common. First, ensuring the design focus has the user at the centre. Understanding the need and operation of the end user helps when designing the functionality of an API. Second, take a product management approach. This includes understanding the product lifecycle and what success or failure looks like, providing a sandbox environment for appropriate testing to take place and, establishing a valid business case to justify the approach. It needs to be noted that success of government APIs is not always guaranteed and the one size fits all approach isn’t necessarily always the best way.

There are two conceptual architectural models information exchange is based on, peer-to-peer and client-server. In a peer-to-peer model, communications can occur between any of the peers, while a client-server model only allows client to server and/or server to client communication.

API implementation patterns have seen a shift from monolithic architecture to microservices architecture. Microservices provide the ability to scale and distribute services across an organisation’s servers according to need. Microservices need not be as small as possible, however the focus should be to manage a single specific business capability. Another architecture pattern is the event-driven pattern. Event-driven architecture manages asynchronous activities between event producers and event consumers. Both the microservices and event-drive architecture implementation patterns have their own strengths and weaknesses which need to be considered before adoption.

The two key API implementation styles used are Representational State Transfer (ReST) and Remote Procedure Call (RPC). Although RPC was commonly used early in the web API implementation journey, it has been surpassed by ReST in recent years.

Having the right API security practices and controls in place is essential when exposing information externally to an organisation. The solution should be the right balance of cost vs risk.

Authentication and authorisation standards should be used to provide a layer of protection against unintentional exposure and control access to APIs. Encrypted connections are essential for communication, further security can be achieved by avoiding the use of predictable resource locators.

When determining requirements, consider whether to align them to an established international standard such as ISO/IEC 27001. Other controls an organisation can implement to heighten security in their API environment should include:

- whitelisting of software identifiers, names and source locations
- throttling or rate limiting
- establishing visibility and integrity through the supply chain
- geoblocking of specific locations
- implementation of intrusion detection and intrusion prevention systems and
- use of real-time fraud analytics.

API risk management involves understanding the risk matrix and what types of operational security problems can arise whether they be accidental or deliberate.

Developers building these services need to be given the proper tools and information to build, deliver and maintain APIs beneficial to end users. This can be done by providing developers the ability to self-serve through portals, ensuring there are documentation standards and providing information in developer kits to assist developers in building APIs.

When implemented correctly, API measurement and reporting insights are useful to support the management of the API suite. Monitoring is not only useful for security, it can also be used to track availability and real time use of APIs. Organisations can monitor system health and use effective alerting where critical warning thresholds are reached and may impact API user experience.

Delivery of APIs should be done through an agile methodology which will allow for organisations to respond to changing demands and deliver incrementally. Automated processes such as continuous integration and continuous delivery allow for a resource shift away from operational tasks into delivery.

With the widespread use of open source or royalty free code, it is recommended that organisations incorporate licence management into their practices. Where there are dependencies on third-party software, steps should be taken to understand their risk profile.

Chapter 1

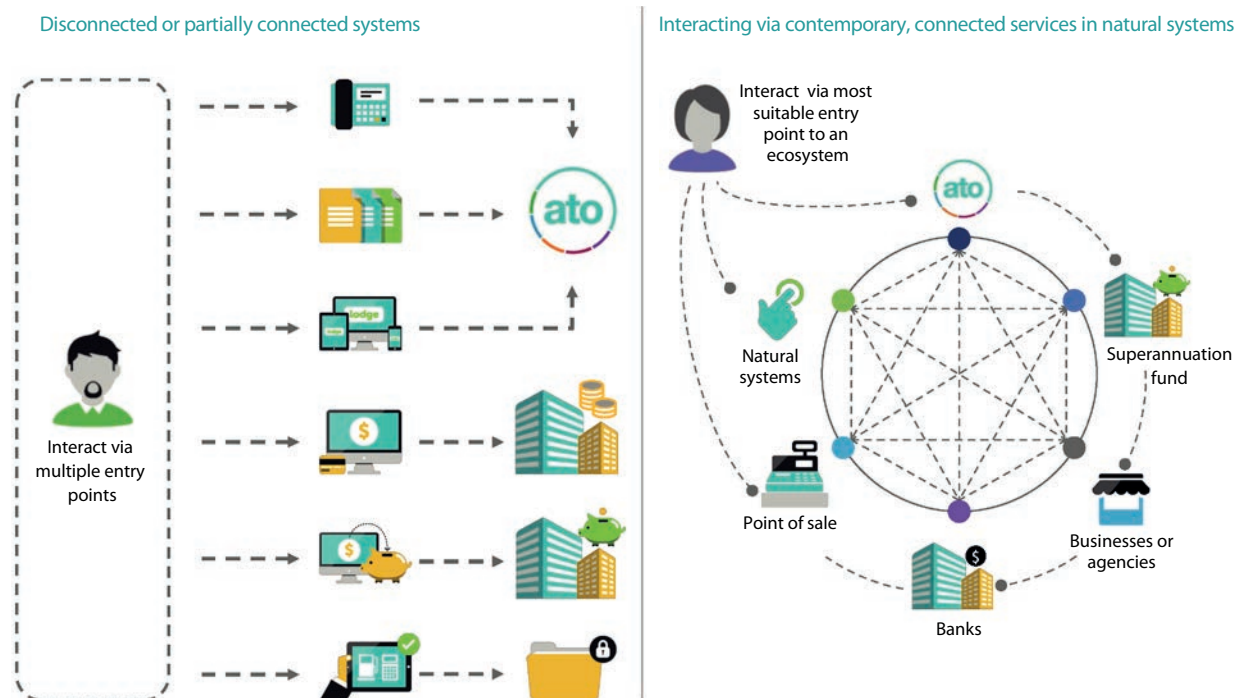
Introduction

1.1. Why APIs?

1. The advent of the internet has led to a fundamental shift in the way people interact across society. It has revolutionised the way people do business. The World Wide Web and web 2.0 have led to shifts in the way governments interact with the community to deliver services. Many of those initial forays were nothing more than digitising paper and providing web based forms through online portals.

2. The introduction of the smartphone led to a shift and expansion to delivering services through applications (apps). These apps were underpinned by small lightweight data calls which drove an increase of web based services. The next wave is not being driven through web pages but now through web services called application programming interfaces (APIs).

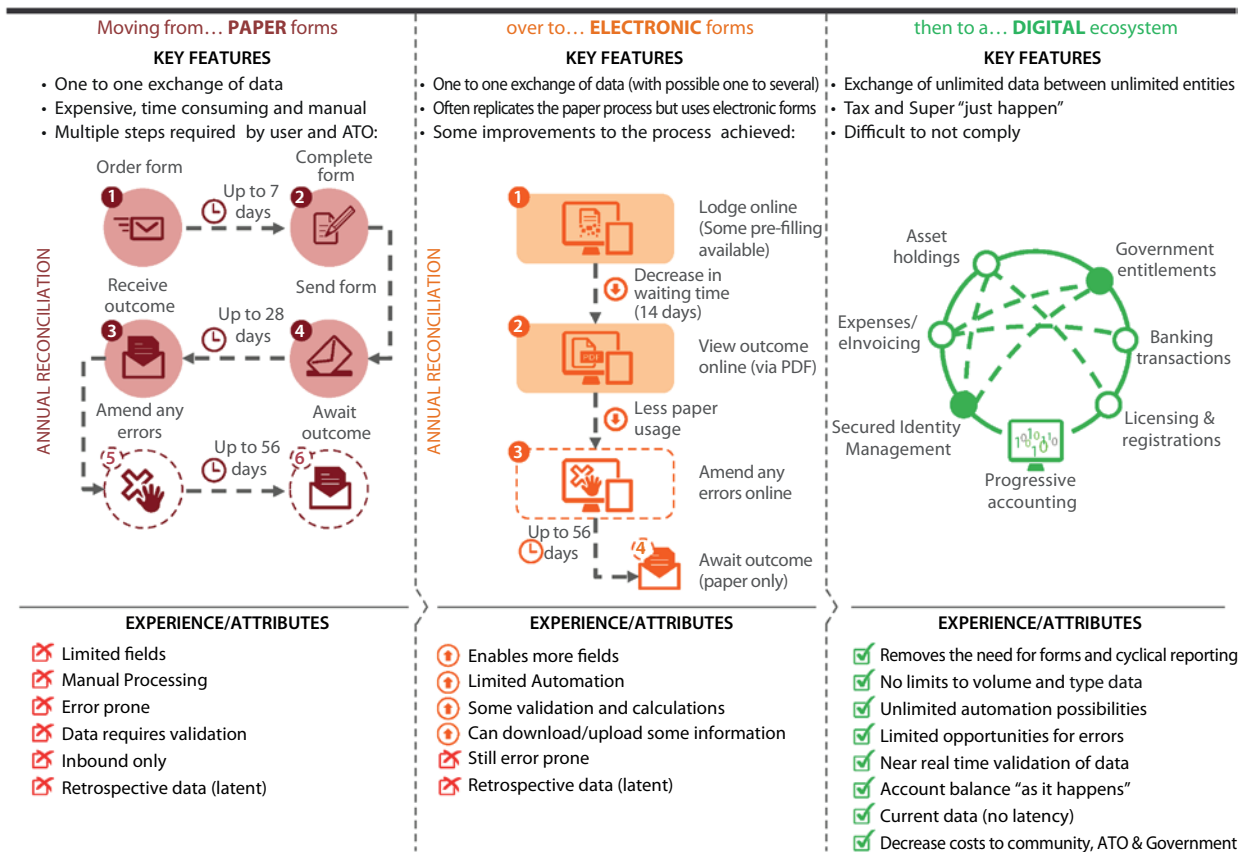
Figure 1.1. The difference between partially connected and fully connected ecosystems



Source: Australia – Australian Tax Office (2018).

3. In parallel the development of software applications, often referred to as the “natural systems” people work in from day to day, has increased. Key examples of these are accounting and business management software. These systems are able to leverage off the growing range of API’s available in the market to gain access to data to support real time updates and transactions.
4. This has allowed a shift away from the previous “digitising paper” and form based approaches to a model where the required data, and only the required data, needed to support a specific transaction is transmitted.
5. As a result APIs are becoming the foundation of digital business. They allow connectivity between systems, people and things without providing direct access. This limits the risk of compromise to the system as opposed to if someone was allowed direct access to the system and the underpinning data stores.
6. When APIs are used by a taxation administration, they allow tax and other government obligations to happen as a by-product of a natural system. Natural systems such as accounting software products can integrate with point of sale systems to give organisations consolidated data across their business.
7. Complete connected environments with the right rules and validations in place enable seamless interactions, which can be facilitated by software. APIs provide benefit where there are multiple touch points and complex chains of interactions. They allow for these transactions to execute one after the other or simultaneously.

Figure 1.2. Attributes of paper, electronic and digital ecosystems



Source: Australia – Australian Tax Office (2018).

APIs can support the move beyond “electronic” forms to real digital interactions

8. True digital requires a move beyond electronic forms and electronic lodgements. Full digital end-to-end transactions require little user involvement beyond the initial request. These transactions will usually stay in channel, and where a response is required, it will be provided to the originating point. There may be many touch points in a digital end-to-end transaction, but on face value, the interaction will be seamless and almost instantaneous. The data goes everywhere it needs to go and there is connectivity throughout the entire ecosystem.

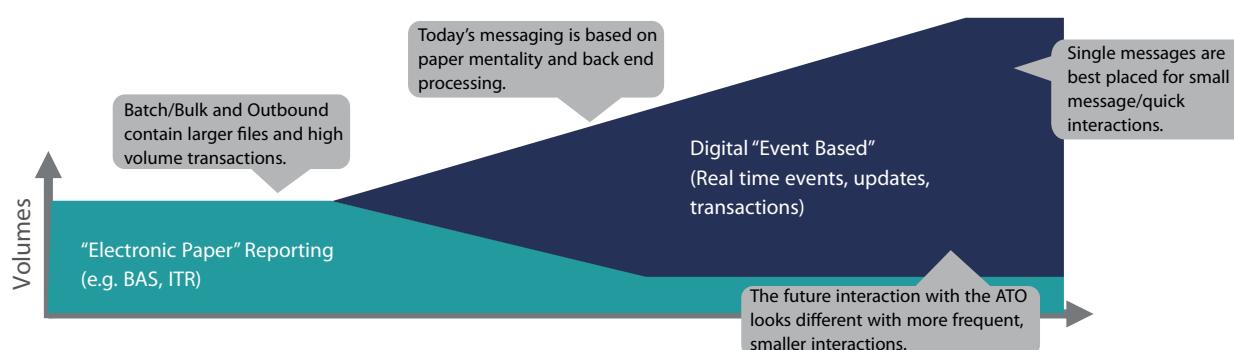
9. There is the opportunity for tax administrations using only paper forms to move straight from paper to digital interactions and bypass building electronic forms. This may provide efficiencies in the longer term, providing greater service connectivity to consumers from the outset and not investing in the build and maintenance of temporary platforms and infrastructure.

10. Many administrations will have investments in electronic forms. As a result a deliberate but phased shift will be required to shift to a digital ecosystem.

APIs can support a system that auto regulates

11. Connectivity through the ecosystem makes it easier for citizens to comply with their obligations while making it harder not to. APIs facilitate conditionality, making sure a transaction can only progress when certain conditions are met, such as a valid business registration. Data can flow through to multiple end points ensuring integrity and consistency across the system. Through this integration, participants are forced to operate within the system. This is discussed further in Chapter 3, “Future applications of APIs”.

Figure 1.3. The growth of digital “event based” reporting



Source: Australia – Australian Tax Office (2018).

APIs can support multi-directional information

12. Basic electronic forms provide functionality for the inward transmission of information, but there is little opportunity for data to be returned through the same channel. APIs allow for data and information to be exchanged in multiple directions. Information can be sent in and out between multiple parties in a digital ecosystem.

APIs can support the application of rules or conditions

13. In machine to machine (M2M) transactions rules can be built in to perform several functions. Validation rules can be used to improve data quality and reduce the occurrence of errors. Upfront validation rules can include checking the right alpha numerical values are included in each field. Upfront validation can also be applied to electronic forms. Where APIs offer additional functionality is through interactive service validation. This goes beyond the upfront validation and includes checking, for example, that the identifying data for a transaction matches. This can reduce the number of orphan transmissions which cannot be processed due to an inability to match to correct record.

14. Conditional rules mentioned above are also important in a M2M environment as the systems need to know when and how certain transactions need to take place. Building conditional rules into an API also has advantages for business processes as they allow for certain transactions to be off limits in certain circumstances. Rules should support and not hinder the efficient operation of natural systems.

15. Access controls can be built in to the API environment to restrict who can access a service. Other levels of control can include when a user can access a service, the level of detail they can see and the modifications they can make within the environment.

APIs can increase Machine to Machine (non-Human) interactions

16. Ecosystems can connect at an M2M level with little to no human interaction. They function on rules which are built into the APIs and determine what the machine can or can't do with the information they collect or distribute.

Table 1.1. Comparison of machine to machine (M2M) and human processing

M2M	Human
Anytime, anywhere	Working hours, physical locations
Rules based	Process/procedure based open to interpretation
Uses previously validated data	Data entry open to errors
Simultaneous touch points	Linear touch points

APIs can support the consistency and correctness of content

17. APIs facilitate the exchange of data, real time payments and other information between systems. Data exchange can be as a result of a transaction, or data from a transaction can be extracted and used to fulfil other reporting obligations. APIs reduce the need for double handling of data and re-entry out of channel as they facilitate the connectivity to various different data repositories. Content is less exposed to errors while transposing as the need for human data entry is limited.

APIs can support appropriate timing

18. Through the use of APIs, reporting can be event based or posted close to real time instead of annual or quarterly. Events can be reported simultaneously to different end points or can be reported in a series of transactions dictated by rules built into the ecosystem, only having the next transaction take place on the successful execution of the previous transaction.




APIs can support a new and improved client experience

19. Organisations have moved beyond simply providing a service to their customers. Customers now expect service experience and the private sector is delivering on this expectation. Typically restrictive organisations such as banks have been setting the standard for a service experience. Strict 9am to 4pm, Monday to Friday operating hours have been replaced with online banking and banking apps, allowing customers access and control over their funds 24 hours a day 7 days a week. It has come to be an expectation of our modern society that customers are able to access services 24/7.

20. The Australian Taxation Office (ATO) has been exploring the theme of customer (or client) experience through its reinvention programme. As part of this, a comparison has been drawn against what the ATO looked like in a purely paper based world, in an electronic world and what it could look like in a fully digital world.

Figure 1.4. ATO transition from paper to digital (from the reinventing the ATO blueprint)

EVOLVING THE TAX AND SUPER SYSTEMS RIGHT APPROACH FOR THE RIGHT TIME

	 Pre-1986: ATO assessment era	 1986-2014: Self-assessment era	 Towards 2020 and beyond: Streamlined self-assessment
AUSTRALIAN ENVIRONMENT	No personal computers or internet, reliance on TV and print media	Emergence of personal computers, the internet and mobile devices, and changing community expectations	Globalisation, digital economy, social media, rapidly changing community expectations, digital by design
TAX ADMINISTRATION PARADIGM	ATO assessment Based on full and complete disclosure	Self-assessment Based on true and correct statements	Streamlined self-assessment Based on integrated digital solutions and stronger relationships
CLIENT SERVICES	Front counter, mail telephone	Call centres, field services, emerging digital services	Right services at the right time
ATTITUDE TO ADMINISTRATION	We set the standards and protect the revenue	Agreed mutual obligations and protect the revenue	We align to community standards and expectations and ensure taxpayers pay the right amount
STYLE OF WORKPLACE	Focus on individual, rigid procedures and rules	Increased focus on teams and multiskilling, rigid procedures and rules	Agile and empowered networks of individuals and teams
STRUCTURAL DIVISIONS	Regional divisions	Market-based divisions	Whole-of-client, whole-of-government, whole-of-ATO
PRIMARY MEDIUM	Paper records	Paper moving to electronic records	Integrated digital solutions
BUSINESS DESIGN FOCUS	Internally focused	Listening to the community	Foster willing participation by making it easy to get things right and hard not to
RISK CULTURE	No risk tolerance	Risk aversion	Sensible risk management
RISK APPROACH	No risk differentiation, random selection	Compliance to model and risk differentiation framework	Tailored engagement based on risk
COMPLIANCE FOCUS	100% assessment	Risk-based reviews and audits	Increasing voluntary compliance

Source: Australia – Australian Tax Office (2018).

APIs can support both retail and wholesale

21. Organisations providing digital services through retail channels (e.g. external websites, portals) should consider the benefits of providing similar services in the wholesale market (i.e. machine-to-machine interactions). Often, third party service providers are able to provide richer functionality when a wholesale service is available. It often comes with reduced ongoing costs to the organisation as there is less maintenance due to there being no user interface to upkeep. There are still build requirements on the organisation, and there is a need to know who the end consumer will be and what their requirements for a service will be.

Chapter 2

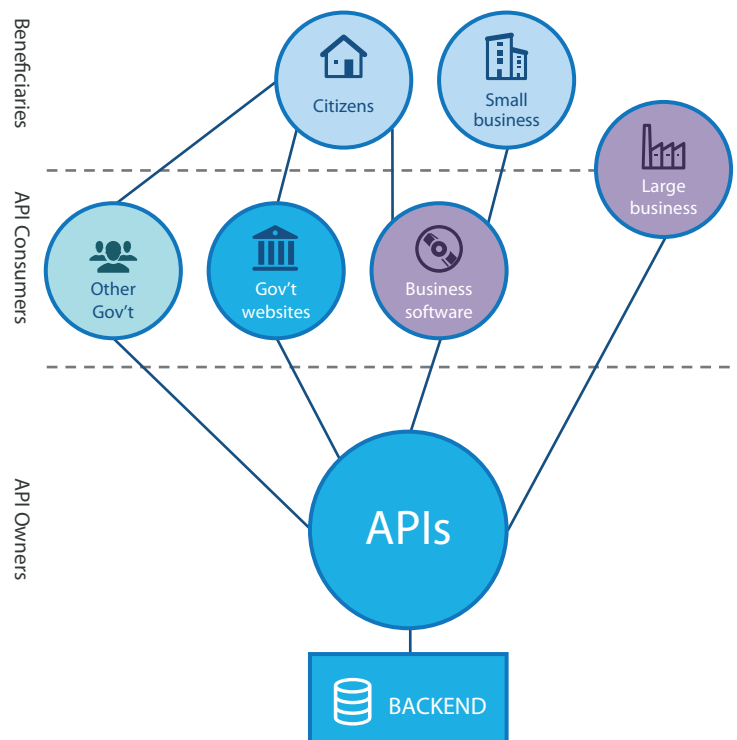
Technical concepts

2.1. A brief history of APIs

22. Application Programming Interfaces (APIs) are defined as: “A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service” (OxfordDictionaries.com, 2018).

23. The concept of an Application Programming Interface is quite simple, it connects to systems via an agreed protocol and set of functions. In most cases, an API is separate from the backend systems that store the information to be shared. Essentially, this is a similar architecture to websites that provide access to organisational information.

Figure 2.1. Connections across the ecosystem



Source: Australia – Australian Tax Office (2018).

24. Modern web scale APIs use the same protocol as between the web browser and web server. Instead of returning data and code that is interpreted by the web browser and rendered as a human readable web page, the API returns machine readable data.

2.2. API Management

What is API Management?

25. API Management has become an increasing focus for delivery teams and executives as organisations seek to participate in the API economy and rapidly grow their catalogue of external facing APIs. The scope and approach to API Management varies from organisation to organisation, but it is generally accepted to include:

- the process of publishing, promoting, and overseeing APIs in a secure, scalable environment
- ensuring that developers and partners are productive consumers of your APIs
- managing, securing, and mediating your API traffic
- allowing an organisation to grow their API programme to meet increasing demands
- enabling the monetisation of APIs
- where technology, business, organisation, and integration concerns intersect.

26. API Management is the operational element that underpins a successful API strategy.

What is API Governance?

27. API Governance is a sub-set of API Management and is a policy driven approach to enforcement throughout the design, delivery, and operation of an API. API Governance concerns frequently include:

- version management – version numbering, prior version support arrangements
- API lifecycle – API status and deployment assurance
- security monitoring and policy enforcement – service accreditation, authentication, authorisation, and audit policies
- risk parameters for managing API environments – visibility, subscription management, service standards and consumer accreditation.

28. There are many commercial product offerings available in the market that claim to automate or simplify API management and governance concerns. These products are many and varied, and organisations should assess each product against their common and unique requirements. This area is further complicated for government agencies and regulators due to legislative constraints and minimum service standards.

Contemporary API Management practices

29. There are some key practices that have emerged which may assist organisations to embed effective API management regardless of specific technology implementations. Many approaches and practices exist but two consistent themes continue to stand out:

- user at the centre
- product-based approach.

Applying user-centred design

30. User-centred design (UCD) practices with IT projects have come of age in the past five years, particularly for projects delivering Web Applications or Mobile Apps. The benefits of UCD are well established across the industrial design segment. Car manufacturers determined long ago that the sound a car door makes when it closes will influence a potential buyer's perception of the quality and robustness of a new vehicle, and ultimately value for money.

31. Application of UCD practices is slightly more complex when considering APIs and wholesale web services. The API producer has little control over the ultimate end-user experience, only the capabilities that are exposed to the next layer in the digital supply chain.

32. Despite the limited ability to directly execute a change to the end-user's experience, good user research can be highly valuable. It can uncover challenges and pain points that may be the manifestation of poor Web API design. User research can also uncover previously unnoticed opportunities to foster innovation through secure access to useful organisational information via APIs.

33. Information and insight gathered through user research should be shared openly with the registered members of the digital supply chain. This may foster competition, prompt innovation, and generate demand for new API services. Ultimately, it will result in a product which better meets the needs of your end consumer.

34. It is critical to establish shared understanding with an organisation's primary API consumers. This is often achieved through close collaboration and co-design. An organisation must provide a clear mechanism for API consumers to provide input and feedback and ensure that they close the loop and respond to input in a meaningful and transparent manner. Providers of regulatory or compliance related wholesale APIs should acknowledge their API consumers as critical partners in the digital supply chain.

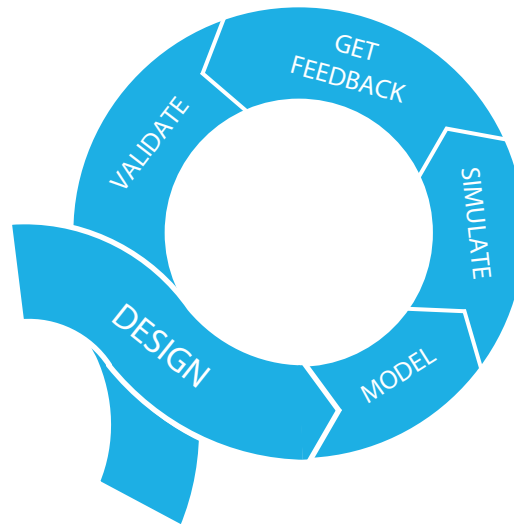
35. Organisations may find that establishing an agreed set of business and technical patterns can assist in creating a shared understanding of common API capabilities and expected service levels.

36. When introducing new services or major enhancements, an API provider should undertake a collaborative user-centred design process with API consumers. A practical example of a co-design process that is applicable to the development of APIs is depicted in Figure 2.2.

37. This process recognises the value of early and meaningful co-design engagement with API consumers as reflected by the following steps:

1. **Model** – Identify process and business requirements, create a logical data model and translate into API groupings.
2. **Simulate** – Model the API resources, operations and methods. Model Payloads and error codes.
3. **Get Feedback** – Publish API mock-ups to an interactive developer portal and manage feedback.
4. **Validate** – Modify the API design as appropriate and continue validation.

Figure 2.2. Extract from the MuleSoft End to End Lifecycle for Microservices



Source: MuleSoft (2017), *Best Practices for Microservices: Implementing a foundation for continuous innovation*.

Applying a product management approach

38. Many organisations are realising benefits and competitive advantage through applying a product management approach to API delivery. APIs are the new client face of the organisation and should be viewed as a product in their own right rather than just a technology solution.

39. This approach has several implications:

1. **Understand the product lifecycle** – Organisations should understand the lifecycle of their products, with emphasis on understanding what success looks like and the appropriate time to withdraw or encourage users to migrate off a specific product or product version.
2. **Provide a sandbox** – Organisations should seek to provide API consumers with an appropriate sandbox environment to undertake “real” market testing. External vendor testing environments should facilitate API consumers in reproducing realistic test scenarios to support a consistent, high-quality end-user experience.
3. **Establish a business case** – All API investment should be subject to existence of a positive business case. Product managers should understand the anticipated usage of the API and the associated cost-benefit of an API implementation project. Business cases need not be overzealous, but rather a clear and concise case for change.

40. Convergence of user and product is not a new concept and has been at the core of most industries for generations. However, for many internal IT teams this is a new way of working. Some key considerations specific to the delivery of APIs that should be acknowledged include:

- the success of government API services is not guaranteed, despite the monopoly on the market
- one size doesn’t always fit all. Organisations should be prepared to tailor APIs to meet various usage scenarios

- API versioning should be actively managed to ensure that project support remains sustainable, whilst balanced against customer expectations
- must have consumers or a known need that the market will consume or fill.

2.3. Architectural approaches

Architecture principles

41. As with all architectural endeavours, an organisation will be faced with competing options and will need to make trade-off decisions. To maximise the likelihood of an API project succeeding and minimise design delays, the organisation should establish a set of guiding principles to address architectural preferences and delivery approaches.

42. These principles should reflect and link to the organisational objectives, priorities, and immovable boundaries. The principles should have a defined order of precedence to allow designers and architects to determine which principle is applied if two principles are perceived to be in conflict. Additionally, a clear escalation path should be established to support a risk-based decision if required.

Conceptual architectural models

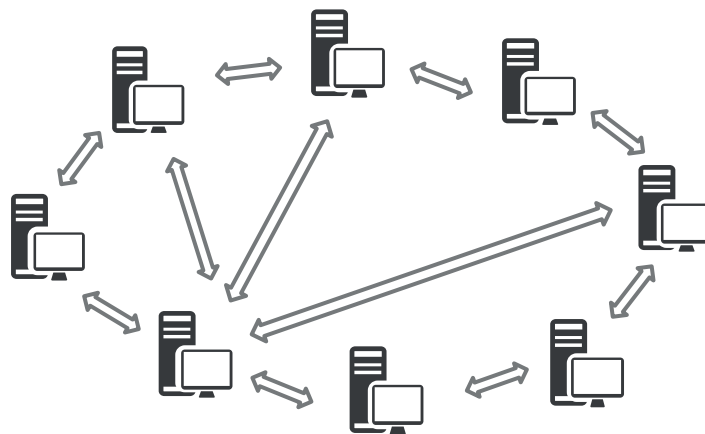
43. Each exchange of information between parties or computer systems is predominately based on one of two conceptual models:

- peer-to-peer
- client-server.

Peer-to-peer

44. Solutions based on the peer-to-peer model involve direct communications between two or more peer computer systems to exchange information in a mutually agreed manner. Each peer is able to communicate with any other peer on the business network using an agreed set of messaging protocols.

Figure 2.3. Peer-to-peer model



Source: Australia – Australian Tax Office (2018).

45. Some readily identifiable peer-to-peer application technologies and standards are included in Table 2.1.

Table 2.1. **Peer-to-peer application technologies and standards**

Technology	Description
Blockchain and Distributed Ledger Technologies	“blockchain is an open, distributed ledger that can record transactions between two [or more] parties efficiently and in a verifiable and permanent way. The ledger itself can also be programmed to trigger transactions automatically” (Lakhani and lansiti, 2017).
Applicability Statement (AS-n) Specifications	The Applicability Statement Specifications: AS1; AS2; & AS3; were defined in the early to mid-2000s to allow support for the secure exchange of business data using various emerging internet protocols. E.g. FTP; HTTP and SMTP (Email). AS4 was first standardised in 2013 and is a sub-set (or profile) of the ebMS3 standard.
EDIFACT	In 1987 the United Nations Economic Commission for Europe established arguably the first global standards for peer-to-peer message exchange.
Health Level 7 (HL7)	HL7 is a “framework (and related standards) for the exchange, integration, sharing, and retrieval of electronic health information” (Health Level Seven International, 2018).

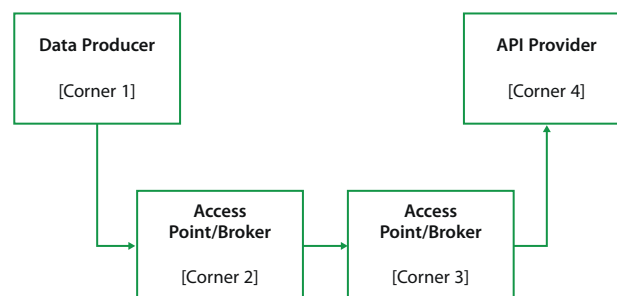
46. One of the major challenges in a peer-to-peer environment is ensuring that all peers can consistently send and receive information that can be readily understood by any other peer. This is known as interoperability.

47. Several standardised models and messaging standards have been developed to improve inter-peer interoperability and are heavily used in the banking, healthcare, retail, and utility sectors. The complexity of the problem space often dictates the approach to standardisation, the implementation cost, and the level of participation in a peer-to-peer messaging network. Some key elements that assist in classifying a use case for peer-to-peer include:

- the regulatory framework – where participants are required by law to exchange data
- the need for participants to conduct the same type of business transaction with multiple parties
- the multi-directional flow of information between transaction participants
- the number of consumers or maintainers of a shared information asset.

48. To alleviate some of the cost and complexity that results from conforming with agreed standards for peer-to-peer communication, brokerage models have emerged. The most recognisable model is the logical four-corner model.

Figure 2.4. **The four-corner model**



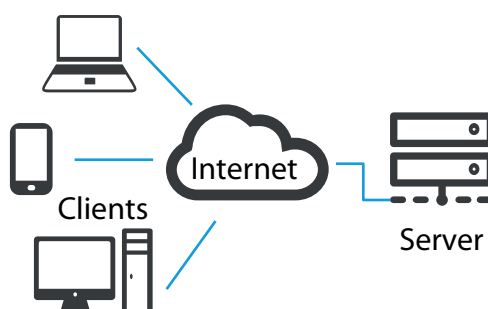
Source: Australia – Australian Tax Office.

49. In the four-corner model, Corners 2 and 3 provide the standardised exchange between parties, whilst offering customised integrations, or commercial transformations for their respective customers, Corners 1 and 4.

Client-Server

50. The Client-Server model is the foundation of the World Wide Web, where solutions generally consist of multiple clients exchanging information with a common party (the Server), but not each other. In these arrangements the Server usually dictates the nature of the information exchange to reflect their internal system representation. Figure 2.5. provides a conceptual representation of the Client-Server model.

Figure 2.5. Client-server model



Source: Australia – Australian Tax Office.

51. With the rapid expansion of the API economy, the Client-Server model has seen a resurgence for Web APIs as companies seek to monetise their data and services in new ways. This has led to a trend away from ratified standards with the industry, to adopting architectural styles and open frameworks in their place. It has also led to a divergence in implementation techniques and API definitions for consistent use cases as organisations seek to maximise their internal advantages.

52. There has been some standardisation initiatives in recent years, but market forces have largely determined the preferred techniques and approaches, particularly around data representation and syntax.

53. Further information on contemporary specifications and standards is included in the Technology sub-section below.

Box 2.1. Example of exchanging SMS data

Telstra and Optus provide APIs to allow a registered user to send an SMS message to a mobile phone number (consistent use case).

Despite having a common industry standard to exchange SMS data between network providers, both providers have distinctly different customer-facing API implementations, defining different mandatory data, different data formats, and different error responses.

Sources: Telstra, 2018; ModicaGroup, 2018.

Contemporary API implementation patterns

54. There are multiple prominent and complementary API solution architecture patterns that have emerged in recent years. They are well supported by vendor tools and products.

Microservices architecture

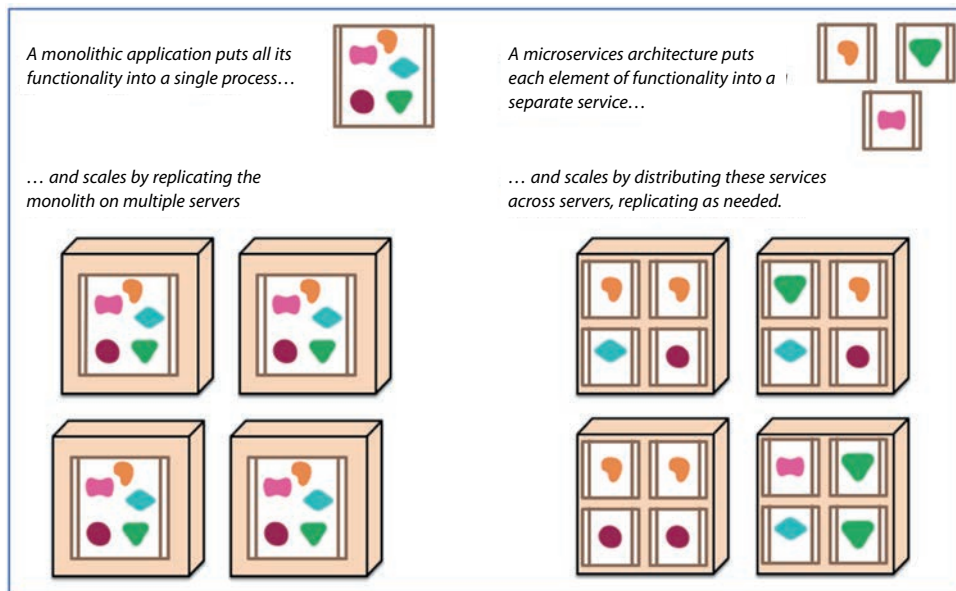
55. There is no clear definition of the microservices architectural style, but it is generally accepted as an “approach to developing a single application as a suite of small services, each running its own process and communicating with lightweight mechanisms” (Fowler and Lewis, 2014).

56. The microservices architecture espouses the value of separating the scope of responsibility for a concept into a single [micro] service rather than building monolithic architectures. It is really a refinement of the Service-Oriented Architecture, with a resource centric focus. Microservices “are built around business capabilities and independently deployable by fully automated deployment machinery” (Fowler and Lewis, 2014).

57. The intent is that all actions related to a specified capability or resource are encapsulated into a single microservice, that can be designed, built, and scaled to meet requirements independent of other business capabilities and largely dependency free.

58. Figure 2.6. illustrates the conceptual difference between a monolithic application and a microservices application.

Figure 2.6. **Monoliths and microservices**



Source: Fowler, M. and J. Lewis (2014), “Microservices: A definition of the new architectural term”.

Key changes from traditional patterns

Determining size, scope, and capabilities

59. One key challenge organisations face when adopting a microservices architecture is how to size and scope these “micro” services. Many incorrectly assume that microservices should be as small as possible or that SOA-based Web Services can just be renamed.

60. Instead, designers should ensure that each microservice embodies the Single Responsibility Principle. Where the focus of the microservice should be limited to a specific, low-level business capability and only manage the required information for that domain context.

It is recommended to start with relatively broad service boundaries and refine over time, as specific business requirements highlight the need for more granular services.

Decentralised data management

61. Decentralisation of data management often appears counter-intuitive to enterprise IT organisations, raising questions like:

- Why would we split our relational database into multiple databases?
- Doesn't it increase the maintenance overhead?
- But our DBA team is centralised...?

62. Decentralised data management is an extension of the concept of limited and isolated scope where a given microservice can only access its dedicated database, but not the databases of other microservices. A microservices API is the only entry point to access a specified data resource to ensure consistency. As a result, its data can be managed independently of other data domains. It can then be hosted, tuned, and scaled to meet the specific needs of the microservice.

63. However, the resulting challenge is to ensure that common data models exist across the organisation, despite their independent data storage arrangements (Fowler and Lewis, 2014). It is critical to understand the requirement to support eventual data consistency. This concept is described in the following sub-section.

Managing complexity

64. With the proliferation of many microservices with discreet responsibilities and independent data stores, the complexity of the operating environment will also increase substantially. As such, to successfully establish and maintain a microservices ecosystem API automation and governance are critical.

65. Additionally, the introduction of decentralised data management and fully self-contained services may require a business transaction to leverage multiple microservices to achieve the business outcome. As such, transactional logic and microservices orchestration is required. This is either achieved on the client-side or via an intermediary microservice or gateway pattern.

To manage the failure of a distributed transaction and eventual data consistency, each microservice must implement compensating transactions.

66. If a dependent microservice transaction fails, then the entire microservice fails and all upstream operations have to be undone by invoking the respective compensating operation of those microservices (Indrasiri, 2016).

Microservices strengths and weaknesses

67. The microservices architectural style has several strengths and weaknesses that need to be considered prior to organisational adoption:

- Strengths:
 - supporting technology optimisation (Gartner, 2018) – enabling each microservice to operate on the most suitable technology platform and at the appropriate scale (Microsoft Corporation, 2017)
 - supporting incremental adoption, evolution and co-existence (Gartner, 2018)
 - enables focused delivery teams to have full responsibility for the capability (Microsoft Corporation, 2017)
 - improved fault isolation (Microsoft Corporation, 2017)
 - can enable higher release velocity, faster innovation, and a more resilient architecture (Microsoft Corporation, n.d.)
- Weaknesses:
 - increased complexity for enterprise environments – complexity is moved as services become more discreet and orchestration requirements are moved to other areas of the architecture (Microsoft Corporation, 2017)
 - cultural change is required (Gartner, 2018) and reconfiguration of IT delivery teams is also required
 - aspiration vs rationale – many organisations want microservices without understanding the rationale or the implications (Gartner, 2018).

More granular microservice implementation patterns

Command Query Responsibility Segregation (CQRS)

68. CQRS is a design pattern that separates command functions (create, update, delete) from query (read) functions through the introduction of separate interfaces and data models. This means that each model can be designed and optimised for a specific function. This pattern can be useful where there are very different functional or quality of service requirements between command and query use cases.

69. An example where this pattern may be useful is the ATO's Australian Business Register (ABR). The ABR's ABN Lookup service is considered a high-volume service with several orders of magnitude difference in transaction volume compared to the ABR's ABN record update services. CQRS would enable those APIs to be independently scaled and data models/platforms tuned for their specific needs.

70. CQRS has the potential to tailor APIs and data models for specific purposes but it also involves some complex issues including the cognitive complexity for designers, and the challenge of ensuring eventually consistent data, across all data models and stores.

CQRS should be a targeted pattern applied to specific sub-systems and services, not a broad-scale solution pattern.

Designing for failure

71. Organisations should treat failure of individual IT system components as an inevitable reality. Even the most robustly designed component can fail due to unforeseen circumstances. As such, organisation should take proactive steps to understand the behaviour of APIs in various failure scenarios. This information should be used to inform and design fault tolerant API services that produce anticipated results even in a failure scenario.

72. Organisations should:

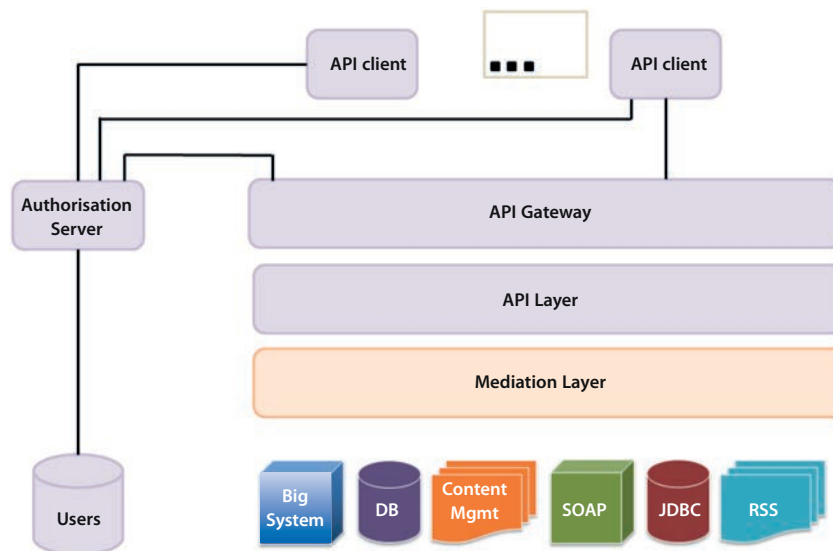
- map and understand external dependencies that underpin APIs (e.g. 3rd Party Authentication Services)
- implement High Availability and Load Balancing
- design services to be replaceable and implement self-healing actions
- plan for portability, including platform substitution
- plan for error handling and plan monitoring as part of design.

API Gateway pattern

73. Organisations who provide more than one API will immediately recognise they have common functions and features that must be performed for all services. These functions often include authentication and authorisation of users, audit logging and quality of service management.

74. Under a strict microservices architecture these functions would be re-implemented or recoded in each microservices. This may result in some negative implications including duplication of code and code maintenance, and the potential for eventual functional divergence.

Figure 2.7. The API gateway pattern



Source: Australia – Australian Taxation Office.

75. The API Gateway pattern has been designed to abstract common functions into a single, lightweight messaging gateway. The API Gateway acts as a single-entry point for all customers, including other microservices and ensures the consistent application of API policies. The API Gateway is effectively the API Governance enforcement layer.

76. As a note of caution, many commercial API Management products offer API Gateway capabilities. Some vendors have true lightweight product offerings, but many have expanded product offerings that blur the line between an API Gateway and an Enterprise Service Bus. Architects should ensure that the selected product is fit for the intended patterns and purpose and utilised in accordance with that purpose.

77. This pattern has the added benefits, that when combined with the microservices architecture it enables teams to focus purely on the microservice scope with policy responsibility sitting in the Gateway Layer.

API Gateway strengths and weaknesses

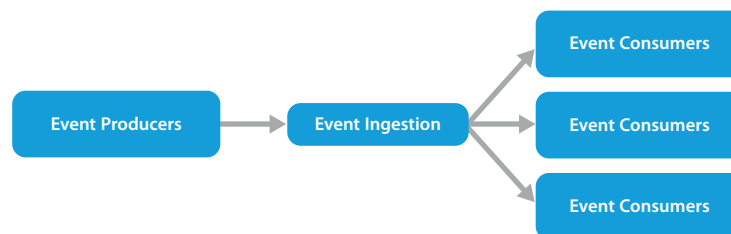
78. The API Gateway pattern has several strengths and weaknesses that need to be considered prior to organisational adoption.

- Strengths:
 - supports the consistent application of API Governance policies
 - supports the consistent collection of API Monitoring and Measurement information (see later section)
 - enables the offloading of microservices cross-cutting concerns to a common layer, supporting simplified scope management.
- Weaknesses:
 - Some API Gateway deployment models introduce a centralised component, which is a potential single point of failure.

Event-Driven pattern

79. The Event-Driven architecture is a tried and tested pattern enabling the effective management of asynchronous activities between event producers and event consumers, as shown in Figure 2.8. In this pattern the event producer need not have any knowledge of the number or identity of event consumers.

Figure 2.8. **Event-Driven Architecture**



Source: Microsoft Corporation (n.d.), “Architecture styles”.

80. The Publish-Subscribe approach is the most traditional implementation of the Event-Driven Pattern, where the event producer publishes the event to messaging infrastructure, for example a message queue or topic. The messaging infrastructure then sends the message to any event consumers currently subscribed. Event consumers that subscribe after an event is published will not receive a copy of the event.

81. The Event Streaming approach is a more contemporary implementation of the Event-Driven Pattern that has come to prominence with the advent of the Internet of Things and extremely high-volume APIs. Event Streaming enables all activities to be published to an ordered log that can be processed in part, or in full, by one or more consumers. The log is persistent, so late consumers are able to process prior history.

82. The Event-Driven Pattern is a complementary pattern that can be used in combination with the Microservices and API Gateway patterns to support extremely high-volume processing or real-time risk analytics and pattern matching.

Publish-Subscribe pattern – strengths and weaknesses

83. The Publish-Subscribe pattern has several strengths and weaknesses that need to be considered prior to organisational adoption.

- Strengths:
 - decoupling of the producer and consumer enables a highly scalable architecture
 - no point-to-point integrations
 - new consumers can be easily added.
- Weaknesses:
 - It has the potential to introduce data synchronisation delays between systems. Implementers should be careful to ensure that solutions and data stores are eventually consistent.
 - It may be difficult to ensure that events are processed in order or only processed once when deployed in a multi-instance scenario.

API implementation styles

84. There are two key styles of API implementation:

- Representational State Transfer (ReST) and
- Remote Procedure Call (RPC).

85. The RPC API implementation style was popular during the first wave of Web APIs in the early 2000s but in recent years its market prevalence has been surpassed by ReST. ReST now accounts for over 80% of API implementations (Cloud Elements, 2018).

86. Due to its prevalence during the first wave, the RPC style underpins many of the API messaging standards that have been ratified by international standards organisations. Many standards processes take several years from conception to ratification, which may be unable to keep pace with the rate of innovation and change currently occurring in the Web API segment.

87. An overview of each style and its existing standards landscape has been provided below.

Representational State Transfer (ReST) Style

88. The ReST style was first described in Fielding’s dissertation (2000) but has been rapidly growing in popularity since its 2006 adoption by Twitter and Facebook. The ReST style relies heavily on the well-established and standardised features of the underlying HTTP protocol and only describes a small set of additional constraints which define a “ReSTful service”.

ReST design constraints

89. The Rest style is defined by a set of design constraints. An API must adhere to all design constraints to be considered ReSTful. The constraints are:

- **Client-Server** – ReST APIs follow the Client-Server architectural model, separating the user interface from the data storage concerns, thus allowing components to evolve and scale independent of each other.
- **Stateless** – All requests between the client and the server must contain all information required to understand the request. It must not rely on any stored context on the server-side. This constraint increases visibility, reliability and scalability.
- **Cache** – To improve efficiency, scalability and user-perceived performance all response data is labelled as cacheable or non-cacheable. This enables a client following an initial request to determine if a subsequent request to the server is required to service the users need.
- **Uniform Interface** – the ReST style has a strong emphasis on a uniform interface between components:
 - **Identification of resources** – a unique resource identifier (or URI) is used to identify the unique resource involved in an API interaction.
 - **Manipulation of resources through representations** – ReST components perform actions on a resource by using a representation to capture the current or intended state of that resource and transferring that representation between components. Common representations formats include JSON and XML. Multiple APIs may be provided to enable a client to access different representations of a single resource.
 - **Self-descriptive messages** – each message contains all the information necessary to complete the task and the requested HTTP Method (see below) describes the expected processing action to be performed by the server.
 - **HATEOS** – (see the Linked Data section below)
- **Layered System** – An API client cannot see and does not need to understand the complexity behind the immediate layer they are interacting with.

HTTP Methods

90. HTTP Methods are at the core of all ReSTful APIs as they instruct the server on the expected processing action that is taken with the provided representation data.

91. Some HTTP methods are “Safe Methods” that should not modify the state of the resource. HTTP methods may also be classified as idempotent, meaning they will return a consistent result no matter how many times the server executes an identical request.

92. Table 2.2 describes the commonly used HTTP methods, their purpose and classification.

Table 2.2. Commonly used HTTP methods, purpose and classification

Method	Purpose	Safe	Idempotent
GET	The GET method means retrieve whatever information (in the form of a representation) is identified by the Request-URI	Y	Y
HEAD	The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response	Y	Y
POST	The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line	N	N
PUT	The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server	N	Y
PATCH	The PATCH method requests that a set of changes described in the request entity be applied to the resource identified by the Request-URI. The difference between the PUT and PATCH requests is reflected in the way the server processes the enclosed entity to modify the resource identified by the Request-URI. With PATCH the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version.	N	N
DELETE	The DELETE method requests that the origin server delete the resource identified by the Request-URI.	N	Y

93. Implementers should note that ReSTful APIs need not implement or externally expose all HTTP Methods for each resource.

94. ReSTful APIs leverage the underlying HTTP Error codes to provide high-level information on the processing outcome of a transaction. If required, additional error information can be included in the response payload.

Remote Procedure Call Style (RPC) (incl. SOAP)

95. RPC is an architectural style for distributed systems that has been in widespread use since the 1980s but saw a substantial re-vitalisation with the advent of the World Wide Web. The central concept of RPC is the procedure, where a specified action can be taken to retrieve or modify the status of one or more data objects. Effectively the procedure can run on any remote system if the information exchange format between the two systems can be understood.

96. The complexity of managing the information exchange led to standardised practices to manage metadata related to the exchange. Simple Object Access Protocol (SOAP) is the most widely used RPC standard for Web APIs, with SOAP version 1.2 becoming a W3C recommendation in April 2007.

97. The standard describes SOAP as “a lightweight protocol intended for exchanging structured information in a decentralised, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.” (W3C, 2007a)

98. SOAP implementations are generally implemented using the HTTP POST method. Error handling is implemented using the SOAP fault concept that provides for highly granular and customised error reporting.

99. Critics of the RPC style would argue that SOAP and its derivations are no longer considered lightweight due to its use of XML and complex relationship with the WS-* Web Service Quality of Service standards.

Technology standards

100. APIs that conform to the architectural models and patterns previously identified can be implemented using a variety of technology standards and pseudo-standards.

101. For tax administrations, the ultimate choice of technology should be driven through consultation and co-design with end-users and digital intermediaries. There are however, some contemporary options that should serve as the starting point for these conversations.

The fundamentals

102. There are some fundamental technology standards that should underpin any contemporary Web API implementation (see Table 2.3).

Table 2.3. **Fundamental technology standards**

Standard ^a	Standards body	Description
HTTP (v1.1 or v2.0)	IETF	The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems. (IETF, 2014) HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. It was first standardised in 1999.
TLS (v1.2)	IETF	The Transport Layer Security protocol provides communications security over the Internet. The protocol allows client-server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery (IETF, 2008)..

Notes: Prior versions of TLS and SSL have been proven to be insecure and should not be used.

- a. Recommended at time of writing. Readers should refer to the responsible standards organisation for the current status of each standard, any relevant addendums, or superseding standards.

Messaging standards

103. As previously noted, many of the ratified standards have been produced based on the RPC style and were ratified by their respective standards bodies in the mid-2000s. Many of these standards have remained stable, with little innovation and limited refinement since that time.

Table 2.4. **Messaging standards**

Standard ^a	Standards body	Description
SOAP (v1.2)	W3C	SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralised, distributed environment (W3C, 2007a). Organisations should avoid establishing proprietary SOAP message structures where ReST or an alternative is fit for purpose.
ebMS3 Core Features (1 Oct 2007)	OASIS	This specification defines a communications-protocol neutral method for exchanging electronic business messages. It defines specific Web Services-based enveloping constructs supporting reliable, secure delivery of business information. Furthermore, the specification defines a flexible enveloping technique, permitting messages to contain payloads of any format type (OASIS, 2007a). This specification became an OASIS Standard on 1 Oct 2007.

Table 2.4. **Messaging standards** (continued)

Standard ^a	Standards body	Description
ebMS3 Advanced Features (19 May 2011)	OASIS	This specification complements the ebMS 3.0 Core Specification by specifying advanced messaging functionality for message service configuration, message bundling, messaging across intermediaries and transfer of messages as series of smaller message fragments (OASIS, 2011). This specification is currently a Committee Specification 01, not an OASIS Standard. Many vendors are yet to implement these features or subject them to widespread interoperability testing.
AS4 v1.0	OASIS	The AS4 profile of the ebMS 3.0 specification has been developed in order to bring continuity to the principles and simplicity that made AS2 successful, while adding better compliance to Web Services standards, and features such as message pulling capability and a built-in receipt mechanism. Using ebMS 3.0 as a base, a subset of functionality is defined along with implementation guidelines adopted based on the “just-enough” design principles (OASIS, 2013). This specification became an OASIS Standard on 23 Jan 2013.
ReST	-	As previously mentioned ReST is a style rather than a standard. It has become an industry pseudo-standard but does not require any specific profiling beyond what is described in the underlying HTTP Protocol.

Note: a. Recommended at time of writing. Readers should refer to the responsible standards organisation for the current status of each standard, any relevant addendums, or superseding standards.

104. SOAP-based messaging standards have seen a significant decline in popularity over the past decade. However, some specifications are seeing ongoing usage for niche or large-scale peer-to-peer networks. For example, the AS4 specification is regulated for use by the Australian Superannuation industry (Australian Taxation Office, 2016) and is currently being adopted across the EU for public procurement under the OpenPEPPOL initiative (Pagh-Rasmussen and Fieten, 2017).

105. The benefits of specifications like AS4 for large peer-to-peer networks result from their comprehensive support for fault-tolerance and quality of service management. These features are critical to ensuring a highly functional peer-to-peer ecosystem, but they come with increased cost and complexity. XML technologies are also heavily used to support these standards, this adds to the perception that these standards are heavyweight and complex.

Organisations should seek to implement all APIs using the ReSTful style.

For specific, niche use cases an organisation may consider a proven and well adopted RPC-style messaging standard (e.g. AS4). This is likely to be for complex peer-to-peer transactions and organisations should seek to use the minimum possible feature set to avoid complexity.

Quality of service standards

106. Underlying each messaging standard is a requirement to effectively manage quality of service attributes like security and reliability. In parallel to the development of SOAP-based specifications the WS-* Standards suite was also created. The WS-* standards suite is also based on XML representation of messaging attributes and consists of the standards described in Table 2.5.

Table 2.5. WS-* standards

Standard ^a	Standards body	Description
WS-Addressing (v1.0)	W3C	Web Services Addressing provides transport-neutral mechanisms to address Web services and messages (W3C, 2006a).
WS-Context (v1.0)	OASIS	When multiple Web services are used in combination, the ability to structure execution related data called context becomes important. This information is typically communicated via SOAP Headers. WS-Context provides a definition, a structuring mechanism, and service definitions for organising and sharing context across multiple execution endpoints (OASIS, 2007b).
WS-Discovery (v1.1)	OASIS	This specification defines a discovery protocol to locate services (OASIS, 2009a).
WS-Policy (v1.5)	W3C	The Web Services Policy 1.5 – Framework provides a general-purpose model and corresponding syntax to describe the policies of entities in a Web services-based system (W3C, 2007b).
(WS-Security) (v1.1.1)	OASIS	This specification describes enhancements to SOAP messaging to provide message integrity and confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies (OASIS, 2012a).
WS-Trust (v1.4)	OASIS	Web Services – Trust 1.4 defines extensions that build on [WS-Security] to provide a framework for requesting and issuing security tokens, and to broker trust relationships (OASIS, 2012b).
WS-Federation (v1.2)	OASIS	This specification defines mechanisms to allow different security realms to federate, such that authorised access to resources managed in one realm can be provided to security principals whose identities and attributes are managed in other realms. This includes mechanisms for brokering of identity, attribute, authentication and authorisation assertions between realms, and privacy of federated claims (OASIS, 2009b).
WS-SecureConversation (v1.4)	OASIS	This specification defines extensions that build on [WS-Security] to provide a framework for requesting and issuing security tokens, and to broker trust relationships (OASIS, 2009c).
WS-Coordination (v1.2)	OASIS	The Web Services – Coordination specification describes an extensible framework for providing protocols that co-ordinate the actions of distributed applications (OASIS, 2009d).
WS-ReliableMessaging (v1.1)	OASIS	This specification (WS-ReliableMessaging) describes a protocol that allows messages to be transferred reliably between nodes implementing this protocol in the presence of software component, system, or network failures (OASIS, 2008).
WS-Reliability (v1.1)	OASIS	Web Services Reliability (WS-Reliability) is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering (OASIS, 2004).

Note: a. Recommended at time of writing. Readers should refer to the responsible standards organisation for the current status of each standard, any relevant addendums, or superseding standards.

107. The WS-* standards are highly complex and have varying degrees of implementation. The most popular standards are the WS-Security; WS-Trust; and WS-Federation. The remaining standards have less common implementations but may be referenced or underpin other OASIS standards.

108. Due to their close relationship with XML, these standards are rarely used for ReST/JSON implementations.

Data representation

109. The primary intent of Web APIs is to exchange or modify data between systems. As such, the data representation, or format, is critical to any API. There are two primary data formats used by most Web APIs, each with sub-variants for specific niche purposes.

110. The two primary data standards are shown in Table 2.6.

Table 2.6. **Primary data standards**

Standard ^a	Standards body	Description
XML (v1.1)	W3C	eXtensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879) (W3C, 2006b). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. XML 1.1 (2 nd Ed) became a W3C recommendation in August 2006.
JSON (2nd Ed.)	ECMA International & IETF	JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent syntax for defining data interchange formats (ECMA International, 2017). JavaScript Object Notation Data Interchange Syntax second edition was standardised as ECMA-404 in December 2017 and concurrently published by IETF as RFC8259.

Note: a. Recommended at time of writing. Readers should refer to the responsible standards organisation for the current status of each standard, any relevant addendums, or superseding standards.

111. Many contemporary Web APIs are now designed and delivered using JSON as the primary data representation because it is perceived as lightweight in comparison to XML. Ultimately the choice of data format should be driven by factors including:

- the existing practices within the ecosystem – awareness and utilisation
- the complexity of the data domain and
- the capability of the device used for data input – some mobile device frameworks do not support XML natively.

It is recommended that organisations consider the use of JSON first, and provide XML capabilities if required to support the existing user community. There are numerous API Gateway products and open source libraries that support execution time conversion between the two data formats.

Niche data representation variants

112. XBRL is a niche data representation standard that was adopted by the ATO and the Standard Business Reporting Program in 2010. XBRL has adoption for securities filings, particularly reporting financial performance and compliance information across 50 countries (XBRL International, 2018). XBRL is a highly specialised specification built on top of XML.

Whilst well suited to financial reporting and cross-entity comparison, the XBRL standard is complex to implement and access to domestic skills is limited. It is not suitable for simple transactional data exchanges like change of address or status of account transactions.

Emerging trends

113. Several major industry segments have relied heavily on XML-based data representations for many years. Two of the most significant XML-based industry standards have recently taken steps towards supporting JSON-based representations. These are:

- **ISO20022:** This international standard is the global contemporary standard for inter-bank funds transfer and the basis for Australian New Payments Platform. ISO20022 recently published a paper describing how to define ISO20022 messages as ReST/JSON APIs (ISO20022 Registration Management Group, 2018).
- **HL7:** Health Level Seven is the international standard for the interoperability of healthcare data. The most recent draft of HL7's Fast Healthcare Interoperability Resources Specification (FHIR3) contains a fully documented JSON representation as a peer to XML (HL7 Community, 2017).

Linked data

114. One of the key elements of ReSTful Web APIs is support for Hypermedia As The Engine Of Application State (HATEOS). An API that implements HATEOS will provide an API client with additional information (links) in its response that enables the client to understand (and navigate) the other actions that are currently available based on the state of the requested resource.

115. A popular example used to demonstrate HATEOS principles is the bank account scenario. Consider an API that allows a client to retrieve the balance of a bank account:

- If the account balance is positive, the API will return links to APIs that enable the client to:
 - deposit funds
 - withdraw funds
 - transfer fund; or
 - close the account.
- If the account is overdrawn, the API will only return a link that enables the client to:
 - deposit funds.

116. HATEOS is defined as one of the original style constraints for a ReSTful API, but it is largely accepted as a requirement for the highest level of API Maturity (Fowler, 2010). There are a variety of formats and techniques for representing linked data. The following common practices and standards are in active use:

- **JSON-LD** (W3C, 2014) – a JSON-based format to serialise linked data. The syntax is designed to easily integrate into deployed systems that already use JSON, and provides a smooth upgrade path from JSON to JSON-LD.
- **Hypertext Application Language (HAL)** (Kelly, 2018) – HAL provides a set of conventions for expressing hyperlinks in either JSON or XML.
- **Collection+JSON** (Amundsen, 2013) – Collection+JSON is a JSON-based read/write hypermedia-type designed to support management and querying of simple collections.

Whilst there is still significant debate about the best approach to describing linked data in your API, JSON-LD is the leading option as it is reasonably well supported from a tools perspective and is a recognised W3C Standard.

JSON-LD also provides a backwards compatible approach to implementing linked data on your existing APIs, as you progress up the API maturity scale.

Schema definitions

117. A key element of a secure data exchange is the ability to validate the data that you have received, to ensure it is fit for purpose, and free from malicious content. Each data representation has its own approach to the definition of an acceptable data schema, and the validation of received data against that schema.

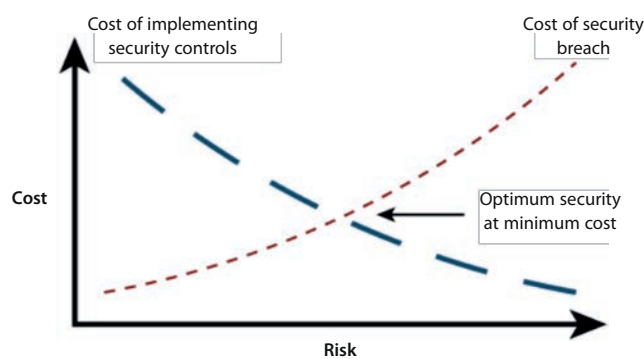
The two preferred respective methods for the JSON and XML data representations are JSON-Schema and XML Schema Definition (XSD).

2.4. API security practices and controls

118. Securing information exposed by internet facing applications is met with numerous challenges, both for information producers as well as consumers. Exposing information, traditionally only available inside the organisation, to external clients introduces a security risk which must be taken into consideration when designing digital services. Security breaches are a common occurrence and high-profile cases (Armerding, 2018) (Bell, 2018) happen on a regular basis.

119. However, security is not a “one size fits all solution” (Sandoval, 2015). A balance between security and ease of use is not easy to achieve, although an important decision when considering solution complexity and cost (Gruman, 2007).

Figure 2.9. Liability vs Return On Investment (ROI) Analysis



Analysis of cost vs. risk
Cost of implementing security vs. cost of security breach

Source: Patronus Laboratories Corporation (2019)

120. With the rapid growth of the API economy, app developers are getting accustomed to a set of standards and best practices when relying on APIs. Global standards improve interoperability between a wide range of programming platforms (Patronus Laboratories

Corporation, 2019) The IT industry changes fast and best practices from several years ago may no longer be recommended (Greenberg, 2016). This also applies to the use of security standards. The industry is additionally converging on a few best practices regarding API security mechanics. These are outlined in the sub-sections below.

Authentication and authorisation

121. Authentication and authorisation of API consumers is critical to ensure that an organisation’s data is appropriately protected from unintentional exposure or modification.

122. There are numerous standards that have emerged to support Authentication and Authorisation of Web APIs (see Table 2.7).

Table 2.7. **Authentication and authorisation standards**

Standard ^a	Standards body	Description
SAML (v2.0)	OASIS	Security Assertion Markup Language (SAML) enables Cross Domain Single Sign On by allowing one computer to perform security functions (authentication and/or authorisation) on behalf of another relying party. SAML is based on XML.
OAuth (v2.0)	IETF	OAuth is an authorisation framework enabling a third party application limited access to an HTTP service (IETF, 2012a). OAuth2 is now suffering from the same problems as the WS-* at the time around web service standards. The proliferation of extensions to the protocol has made it harder for developers to implement consistently. Note OAuth1.0 has been superseded.
JWT	IETF	JSON Web Tokens (JWT) is a standard for passing authenticated identity information between an identity provider and service provider similar to SAML but optimised for JSON. JWT relies on other standards: JWS, JWE (IETF, 2015a).
JWS	IETF	JSON Web Signatures support a data structure representing a digitally signed message (IETF, 2015b).
JWE	IETF	JSON Web Encryption provides support for encrypted content using JSON-based data structures (IETF, 2015c).
JWK	IETF	JSON Web Key provides a data structure that represents a cryptographic key (IETF, 2015d).
JWA	IETF	JSON Web Algorithms provide cryptographic algorithms and identifiers to be used with the JSON Web Signature, JSON Web Encryption and JSON Web Key (IETF, 2015e).
OpenID Connect	OpenID Foundation	OIDC provides an identity layer on top of the OAuth2.0 protocol (OpenID Foundation, 2014a). Uses RFC 6750 OAuth 2.0 Bearer Token Usage Specifications (IETF, 2012b).
SCIM (v2.0)		System for Cross Domain Identity Management (SCIM) simplifies the management of user identities across multiple domains.

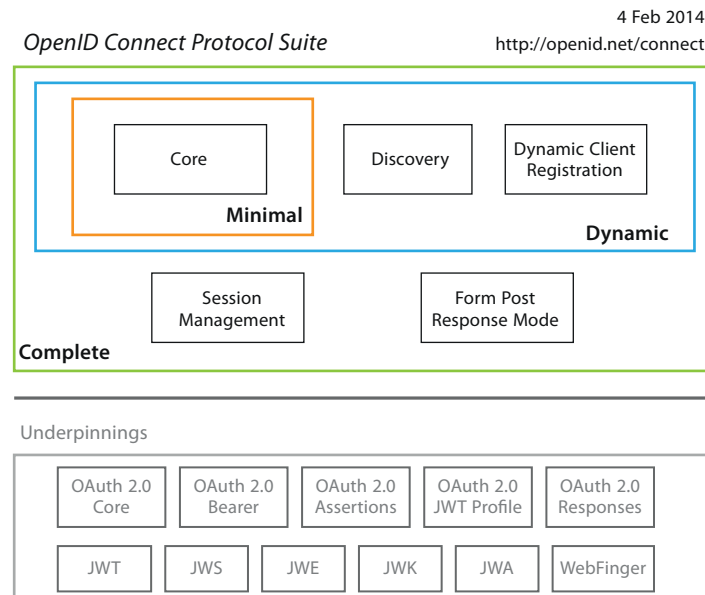
Note: a. Recommended at time of writing. Readers should refer to the responsible standards organisation for the current status of each standard, any relevant addendums, or superseding standards.

The OpenID Connect Protocol Suite

123. Since OAuth2.0 does not provide authentication, an additional layer is needed to achieve this. A popular standard (OpenID Connect) is provided by the OpenID Foundation and builds on the OAuth2.0 set of standards.

124. OpenID Connect “allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner” (OpenID Foundation, 2014a).

Figure 2.10. OpenID Connect Protocol Suite



Source: OpenID Foundation (2014b), “Welcome to OpenID Connect”.

125. OpenID Connect allows clients to request information about authenticated sessions and end-users. OpenID Connect is one of the most prevalent approaches being deployed to authenticate RESTful Web APIs.

SAML and WS-Security Suite

126. The recommended approach to authenticating SOAP-based Web APIs is using the WS-Security suite of standards, in combination with the SAML Standards. These provide a comprehensive approach to assuring the request or response has been securely delivered, and that the credential presented by the initiator has been assured.

127. SAML is often used in enterprise scenarios but is becoming less popular in web-scale API solutions.

Proactive vulnerability management

Always use HTTPS for APIs

128. There is no valid reason not to use an encrypted connection to APIs. Computing power provides fast enough performance where the end user will not notice a difference in the response times because of encryption overhead.

129. Domain validation certificates, which ensure the client is connecting to the right API, are available for free. Acquiring a server certificate for a new API can be integrated in the software development workflow using DevOps processes.

130. APIs should only be made available over a secure connection; there is no need to offer non-secure connections. It is therefore recommended to disable non-secure access to the API altogether. HTTP Strict Transport Security (HSTS) will also prevent HTTP fallback vulnerabilities which may be possible in some circumstances.

131. OWASP recommends mutual authenticated client-side certificates for highly privileged web services (Cloud Elements, 2018). Although more complex to integrate and manage, when dealing with highly privileged services, the security benefits will outweigh the cost of the overhead.

Prevent predictable resource locations

132. Be careful of predictable identifiers when designing API interfaces. Predictable identifiers enable the enumeration of resources or allow a client to guess the location of information that is restricted.

133. Mitigation strategies include performing strict authorisation checks where not only the authentication status is verified but also if the current user is allowed to access a certain resource location. Instead of using predictable identifiers, the API could use long random strings such as Universally Unique Identifiers.

Understand your dependencies

134. Most organisations recognise the importance of undertaking vulnerability identification and verification activities as part of their standard systems delivery lifecycle. However, there is a growing area of concern that extends beyond the API code that has been written directly by the organisation. With the increasing use of Open Source libraries and packages, developers are often unaware of the quality of, or vulnerabilities contained within the code with which they rely upon (Forrester, 2017).

It is highly recommended that organisations leverage static code analysis tools in their automated delivery pipeline that analyses all dependencies.

In addition, organisations should consider leveraging one of the handful of tools available that map open-source dependencies and support crowdsourced vulnerability tracking and reporting.

Security certification

135. Organisations should establish their requirements to certify their API platform and services. For the ATO this means compliance with the Australian Signals Directorate published Information Security Manual (ISM) and the Information Security Registered Assessors Program (IRAP).

Additionally, organisations should consider if certification is required to allow external consumers to access your APIs. This should be a risk-based assessment of the API and API consumption model.

Organisations should seek to leverage internationally recognised certification programmes that their API consumers may already hold, prior to implementing their own certification programme for example, ISO/IEC 27001:2013.

Whitelisting and throttling

Whitelisting

136. Whitelisting is a common technique to restrict execution rights to pre-approved software or a pre-approved request source. Traditional API whitelisting may consist of a pre-configured list of software identifiers, names, source locations or hashes that are allowed to execute.

137. With the advent of the API economy, providers are seeking to quickly support developers to consume the APIs through self-service catalogues and developer portals. This self-service adoption objective is at odds with traditional whitelisting processes. Manual whitelisting processes are often not responsive enough to support some commercial API models that seek to reduce barriers to adoption.

138. The contemporary approach to Web API whitelisting is to issue each API consumer or API Subscription with a unique API key that provides access to the API, rather than registering consumer provided information. The API consumer is responsible for the management and protection of their unique API key. API keys can be vulnerable to discovery by a third party if not appropriately protected when stored on client systems or during transmission if the API is poorly designed or not protected by TLS. As such, organisations should provide a mechanism to revoke and reissue replacement API keys.

139. Organisations should ensure they do not use API keys as the only method of authentication for APIs that provide sensitive or Personally Identifiable Information (PII).

140. Organisations that use API keys to conduct whitelisting should ensure that:

- APIs are protected using TLS 1.2;
- API keys are not provided in the request URI;

141. Organisations should also consider developing appropriate blacklisting techniques to protect medium to high risk APIs. If an organisation considers that malicious or accidental misuse of an API is likely to result in material consequences, then it may wish to implement blacklisting. Blacklisting is the explicit blocking of specific products, users, or connection from source IPs or regions.

Throttling

142. It is recommended not to allow unlimited access to an API in terms of the volume of requests that can be made in a given period of time. Unlimited access leaves a service vulnerable to denial of service attacks where the service is slowed down by inundating the API with millions of requests. Unlimited access also allows malicious clients to retry authentication requests by brute forcing access codes.

143. Unlimited access to API's can be mitigated by rate limiting (throttling) strategies. Examples of rate limiting strategies are:

1. limit per connection property (IP address)
2. limit per user (account/access token/API key)
3. limit per application property (user account/resource type)
4. limit per context (region/type of app).

144. Rate limiting strategies can also be used when monetising the use of APIs. Based on the client identifier or user, the API provider determines the number of requests made in a certain period.

145. The HTTP standard provides a standard response code when the limit is reached and a client makes more requests than allowed, for example, 429 Too many requests.

Security monitoring and intervention

Supply chain visibility/integrity

146. In a modern, digital information supply chain, many parties may transport, transform or enrich data that is provided to your Web API, or between your API response and the ultimate end user. Organisations have very little direct influence over the usage and manipulation of API inputs and outputs beyond their direct interface boundary.

147. In a truly client-server scenario, the API definition and transport level security such as TLS is usually sufficient to enable integrity. However, more complex peer-to-peer, or multi-hop, scenarios may introduce additional risk to each transaction.

148. Organisations provisioning medium to high risk APIs should proactively work with their API consumers to understand the digital supply chain and implement visibility and integrity measures. These may include, but are not limited to, digital signatures, token chains, and notary capabilities.

GeoBlocking

149. As previously mentioned in the whitelisting sub-section, the ability to blacklist an IP address range or a geographic location can provide partial mitigation from some forms of Distributed Denial of Service attack. If an API provider determines that they are unlikely to receive any legitimate business transactions, or detects an unusual spike in regional transaction ratios it may choose to proactively block, or rate limit the suspect traffic. This technique is known as GeoBlocking.

150. Organisations should be careful to understand their anticipated and actual usage geographic consumer scenarios as it is often not straightforward. The popularity of public cloud services, crowd sourcing and off-shore development labs may result in consumers connecting to your APIs from highly diverse locations. The geographic traffic profile for your external testing environments may also vary from your production environments.

Intrusion Detection/Prevention Systems

151. An Intrusion Detection System (IDS) can be either a software or hardware appliance that monitors network traffic to detect potential threats or malicious activity indicators. An IDS will analyse traffic and identify patterns that may indicate a cyber-attack, but is a passive solution that simply detects, captures intelligence and alerts when a pattern is detected.

152. An Intrusion Prevention System (IPS) builds on top of the IDS concept to actively block potentially harmful traffic. Both IDS and IPS solutions can be subject to false positives and some organisations are reluctant to implement active blocking capabilities.

Implementation of IDS and/or IPS solutions is considered good practice.

Real-time fraud analytics

153. Real-time fraud or behavioural analytics is an emerging capability that can provide additional protection to an organisation. Many organisations are collating substantial amounts of data about the typical usage patterns of their consumers. This information can support real-time streaming analytics processes to detect a potentially fraudulent transaction as it is in flight. These processes can subsequently trigger additional actions and risk treatment, or suspend the transaction pending further validation. Banks have been using this style of processing to determine unusual credit card transactions for some time.

Risk management

154. The exposure of your organisational data to the outside world through APIs is not without risk, but these risks are often manageable. Organisations should seek to fully understand their API risk profile and technology leaders should engage in overt and meaningful discussions with their line of business counterparts about the risk profile, key mitigations and incident response plans.

API risk matrix

155. Each new API should be assessed against an agreed risk matrix that considers the impact of exposure of API data to an unauthorised or malicious user, including:

- breach of privacy (through explicit exposure or implicit confirmation of facts)
- financial loss or penalty
- identity theft
- reputational damage and loss of public trust
- misuse of APIs to gather unfair market advantage.

156. Organisations should ensure they do not apply overzealous security controls to the wrong APIs, effectively limiting their potential. For example, if an organisation currently provides some information on their public website to anonymous users, it should not design an API using authentication and whitelisting to provide the same information.

Operational security

157. There are two types of operational security problems:

- **Accidental misconfigurations:** These are inadvertent by nature and are by far the most frequent type of operational issues.
- **Deliberate misconfigurations:** These are deliberate in nature but vary in their degree of maliciousness. For example, violation of the security policy to allow an operator's home system access through the corporate firewall is not as likely to be as severe as acts of sabotage by a disgruntled employee.

158. The typical reaction when looking for a solution to a security problem is to look for features to configure. It is important to understand that operational problems cannot be fully solved by features, because the person making the misconfiguration may also remove the feature that is meant to protect against such misconfigurations. Operational problems require operational solutions, and operational competence of the organisation.

159. Operational solutions include:

- **Operational security policy:** there should be clear guidelines on what operators can do and what they are not allowed to do.
- **Change management process:** every organisation should create precise processes that define and control how changes to their APIs and underlying infrastructure occur.
- **Access control:** it is good practice to restrict access to API Management Interfaces. Access restrictions are traditionally implemented via AAA authentication.
- **Authorisation:** the access an operator has should be restricted to the minimum access needed for the operator to do their job.
- **Secure and verify:** all of the above measures are active attempts to detect a change in the API, such as a configuration change. It is also possible to detect policy violations by analysing the traffic on the API. For example, intrusion detection systems can create alerts when flows are seen on the network that does not correspond to the policy. There are many other ways to monitor for traffic anomalies.
- **Automation:** It is generally recommended to automate processes and procedures, specifically recurring verification processes, because humans tend to overlook details in log files and similar processes. Automated processes are also less likely to make mistakes, although if a mistake does happen, it is often systematic and therefore easily detectable.

2.5. API consumer experience

Developer portals – self service

160. One of the major changes that has led to the widespread adoption of Web APIs is the ability for developers to discover and explore an organisation's APIs.

161. At a minimum, most organisations now provide developers with self-service access to comprehensive API documentation and the ability to register/create an account to access testing services.

162. More advanced developer portal solutions allow developers to manage paid subscriptions, service level packages, API security keys and limited support capabilities. Many portals also provide a web interface to support simple testing of APIs with user configured inputs.

163. Publishing to the API developer portal should be an automated by-product of continuous integration processes publishing a new API version to an integrated API gateway service. Consumers can opt-in to be automatically notified of an update to any APIs that they have subscribed to consume.

164. API Providers should consider the privacy requirements associated with subscribed data collection ensuring that they adhere to emerging global practices (European Commission, 2018).

API documentation standards

165. Significant change has occurred in API documentation practices over the past five years. Traditionally, Web Services were documented as text documents with a few supporting machine consumable descriptions like XSD schemas and WSDL files.

166. With the increasing popularity of Web APIs several initiatives were commenced to improve the approach to API definitions and associated documentation. The most popular specification is the OpenAPI Specification (OAS), formally known as Swagger. OAS provides a comprehensive, user friendly way to describe and share your API prior to implementing the associated business logic. OAS specifications are well supported by several API gateway products. API Providers can import the OAS definitions to automatically create their gateway configuration.

167. A complementary (originally alternative) API modelling language known as ReSTful API Modelling Language (RAML) was developed in parallel to OAS. The two projects have since converged and key partners are collaborating to leverage the best of both approaches (Sarid, 2017).

Organisations should apply a contemporary approach to API definition, leveraging the Open API Specification.

Organisations should use OAS Specifications to expose their API design to their consumers early and often in the API lifecycle. This reinforces the co-design feedback loop and reduces the cost of rework when compared to traditional lifecycles.

Organisations should ensure the API lifecycle status is clearly defined and visible to the consumer (e.g. Consultation Draft vs Live).

Developer kits

168. Traditionally organisations may have produced a Software Developer Kit (SDKs) to assist their API consumers overcome implementation complexity and reduce the time required to develop API consuming software.

169. Often organisations would need to support multiple programming languages and regularly update their SDKs to keep pace with leading edge developers, but also need to support an extensive number of prior versions for late adopters.

170. With the improvements to API documentation standards, simplified API styles and focused service scopes, the need to provide comprehensive SDKs is dissipating. In addition, many API portals that leverage the Open API Specification can automatically generate client and service-side code packages based on the API definition. These code generation tools are supported by an active open source community that adds new generator packages when another programming language becomes popular.

171. Most importantly, when a new API is released, or a version update occurs the organisation has no requirement to manually update an SDK.

Organisations should seek to provide code generation capabilities in their developer portal and should eliminate (or minimise the scope of) any existing SDKs.

2.6. API measurement and reporting

172. Organisations have been seeking collection of information and insights from their software applications for many years. APIs are no exception and organisations have the ability to capture real-time information to support the ongoing operation of their APIs, but also to support product management and monetisation outcomes.

Operations management

173. Organisations implement Operations Management techniques such as API Monitoring to achieve resilience objectives including High Availability. Monitoring enables organisations to minimise the Time to Detect and Time to Mitigate a production issue (Guckenheimer, 2017a). The objective is to use monitoring to detect an emerging issue before it impacts or is reported by a customer.

Monitoring techniques

174. Organisations often implement one or more of the following monitoring techniques:

- **Telemetry** – the process for collection of data for API monitoring purposes. Organisations can implement telemetry collection through any combination of mechanisms including installation of monitoring agents, flags in source code, and application or infrastructure logging.
- **Synthetic Monitoring** – the process of executing a repeatable set of API transactions to measure the current performance of the API compared to previous history or defined threshold parameters.
- **Real-User Monitoring** – provides insights into the end user experience replicating conditions such as network lag or mobile device performance.

Effective alerting

175. Collecting real-time telemetry provides little value if it is not easily interpreted or does not provide actionable insights. Organisations should enable alerting to ensure that operational staff are aware of a change to system health and are able to proactively investigate the cause.

176. Organisations should determine initial monitoring thresholds for key application metrics and establish a cyclical review and tuning programme. Organisations should ensure that they have identified both warning and alert thresholds.

177. It is also important to ensure that the alert distribution method is understood and suitable for the target audience. There is a tendency for organisations to send alerts broadly and to aggregate up red or amber alerts to the highest levels of the organisation dashboard. In many circumstances, this can create unnecessary unrest and a negative perception of an application's performance. For example, a system administrator may wish to set an alert threshold that triggers when a storage device reaches 80% capacity. However, this alert might have no immediate business impact and should not be displayed on a business facing dashboard.

178. Organisations should consider the criticality of the alert and determine if it should be delivered via SMS, email or a messaging or social channel, for example Slack.

Product management

179. API monitoring can also be useful to support product management decisions. Transaction volumetrics may provide useful insights including trend analysis of product usage, failure and error rates etc.

180. Additionally, API developer portal and subscription information can provide product managers with insights on the attractiveness of product offerings, the conversion and implementation rates and the general popularity of each service.

181. Finally, monitoring in combination with techniques such as alpha and beta testing can inform product managers about the optimal configuration of product features and functions.

Benchmarking

182. The consumers of Government provided APIs are often operating in a competitive market and providing innovative solution to end users. API providers may be able to stimulate improvements to product quality and API adoption through the provision of consumer-facing metrics and benchmarking tools.

183. Some organisations have benefited from publishing business error benchmarks to API consumers, which have triggered substantial quality improvements. API providers should ensure they consider the privacy and reputational impacts of publishing benchmarking data and take appropriate steps to de-identify or aggregate data as appropriate.

2.7. API delivery techniques and toolsets

Agile

184. There is a multitude of Agile, and agile at scale, frameworks and practices available. The consistent theme throughout all agile methods is maximising value creation. Often this is using just-in-time practices for concept elaboration and implementation, that allow organisations to rapidly respond to changing demands and priorities.

185. The Australian Government’s *Digital Transformation Agency* (DTA) and the UK Government’s *Government Digital Service* has some excellent resources to support organisations to gain an understanding of agile approaches and the agile mindset (Digital Transformation Agency, 2016). The DTA’s Digital Transformation Culture Posters can provide an effective reminder to practitioners on how to focus their effort for positive results (Digital Transformation Agency, 2017).

Automation pipelines

186. As an organisation’s API portfolio grows or consumption of its APIs rapidly increases, the importance of automation becomes critical. Automation allows an organisation to quickly respond to changing business needs, delivering consistent and reliable outcomes.

Organisations should consider scope for automation during service design and consider automation in the context of the entire service lifecycle, not just development.

Continuous integration

187. Automation of key software development steps is a generally established practice that has evolved over the last decade. Automation often involves build, code packaging and various levels of test automation, including regression testing. This automation is commonly known as Continuous Integration (CI), where code is checked-in by the developer and automatically built and tested.

188. In recent years the use of automation has progressed beyond CI to support continuous delivery and cloud elasticity.

Organisations should seek to ensure they have an automation profile that is appropriate to the risk profile of their services and the supporting operational model.

Continuous delivery

189. Continuous Delivery (CD) is the natural extension of CI. CD automates the process of deployment, enabling deployments to occur at the push of a button. This approach still ensures the business or product owners have full control and can deploy an application version on demand (ThoughtWorks, 2018). CD has the ability to substantially reduce the time and effort involved in deployment of APIs and their dependencies, allowing for more frequent releases and rapid feedback cycles (Atlassian, 2018).

190. Organisations that have mature CD processes may wish to further automate the deployment decision and implement Continuous Deployment, which automatically deploys new software to production if it has passed all automated tests.

191. Some organisations leverage the concept of deployment rings, or progressive release of a deployment to various groups of subscriptions stakeholders (Guckenheimer, 2017b). This approach enables organisations to monitor the success and stability of a release with each deployment group prior to release to the next “broader” ring of stakeholders. This approach is commonly applied by organisations such as Microsoft and Apple, which enable users to opt into early adoption of new features, the inner deployment rings.

It is recommended that organisations incrementally progress through the Continuous Integration, Continuous Delivery and Continuous Deployment maturity levels, only progressing when technology teams, product management and business owners are all comfortable with the risk profile.

Cloud elasticity

192. One of the key benefits of cloud technology is the ability to dynamically scale up and down to meet the demand for API services. To enable elasticity each application component must be able to scale in a repeatable manner, within a set of predefined thresholds. API and microservice implementers will need to apply technologically appropriate techniques to create lightweight deployment blueprints for each component or dependency.

193. Some common techniques include the use of containerisation solutions based on Docker or Kubernetes. Containerisation techniques allow for consistent and rapid environment provision and tear-down and are highly suited to stateless API solutions.

Resilience of dependencies

194. The use of open source or royalty-free code libraries is nothing new in the software development world and has been occurring for decades. However, the legal system has caught up. Many technology organisations are regularly finding themselves in court to argue their ownership or appropriate use of intellectual property manifested as application source code.

195. As such, organisations are becoming increasingly aware of the need to understand the licensing that applied to all the proprietary and open source code on which they rely. Several tools have emerged to help identify, track, and reconcile issues with source library licensing. Many tools also support developers to discover libraries that both meet their functional needs and their preferred licensing model.

Organisations should incorporate licence management into their library selection and usage processes. Where practical, they should also include licensing checking processing into their automated build pipeline.

196. As previously identified in the API Security Practices and Controls sub-section, developers should analyse and understand the risk profile associated with all third-party software dependencies, noting the risk profile can change very rapidly.

Chapter 3

Future applications of APIs

197. In an environment fully connected via APIs the system can auto regulate. Machine to machine transactions based on pre-established rules create potentially limitless opportunities for data sharing, real time reporting and real time payments.

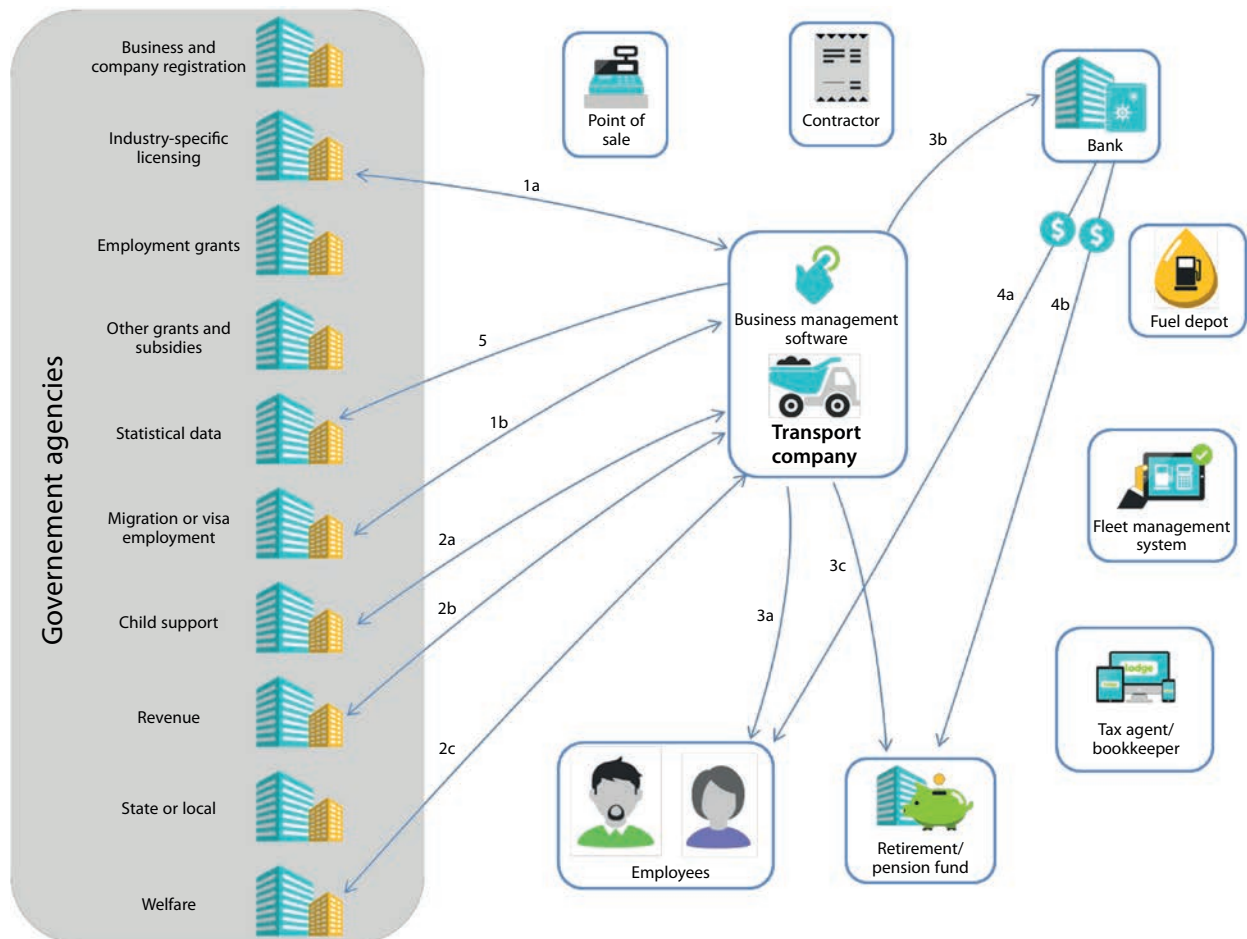
198. The following examples are conceptual use cases for the potential interconnectivity within the environment through the widespread implementation and use of APIs.

3.1. A transport management company executes a payroll event

199. From their business management software (BMS), the company runs a payroll event. All checks must pass for the transaction to be completed. The following interactions will happen almost instantaneously and there is the potential for several of the interactions to occur simultaneously:

- **1a.** API made available from the agency responsible for industry specific licensing. A response will be provided indicating whether requested staff members hold the appropriate type of licence if required for the work they are doing.
- **1b.** API made available from the agency responsible for migration or visa employment. A response will be provided indicating whether requested staff members hold the right visa for the type of work they are doing (where applicable for non-residents).
- **2a.** BMS sends pay details to the child support agency via their API. A response is provided confirming whether an amount is to be taken out of the pay to meet child support obligations.
- **2b.** BMS sends pay details to the revenue agency via their API. A response confirms whether the employee has any outstanding obligations.
- **2c.** BMS sends pay details to the welfare agency via their API. A response is provided confirming whether an amount is to be taken out of the pay to meet outstanding welfare obligations.
- **3a.** BMS issues a payslip to the employee with the final amount to be paid.
- **3b.** BMS notifies bank of payments that need to be made.
- **3c.** BMS notifies retirement of pension fund of amounts to be paid.
- **4a.** Bank issues payment to the employee.
- **4b.** Bank issues payment to the Superannuation fund.
- **5.** BMS issues non sensitive data to the agency responsible for collection of statistical data.

Figure 3.1. A transport management company executes a payroll event



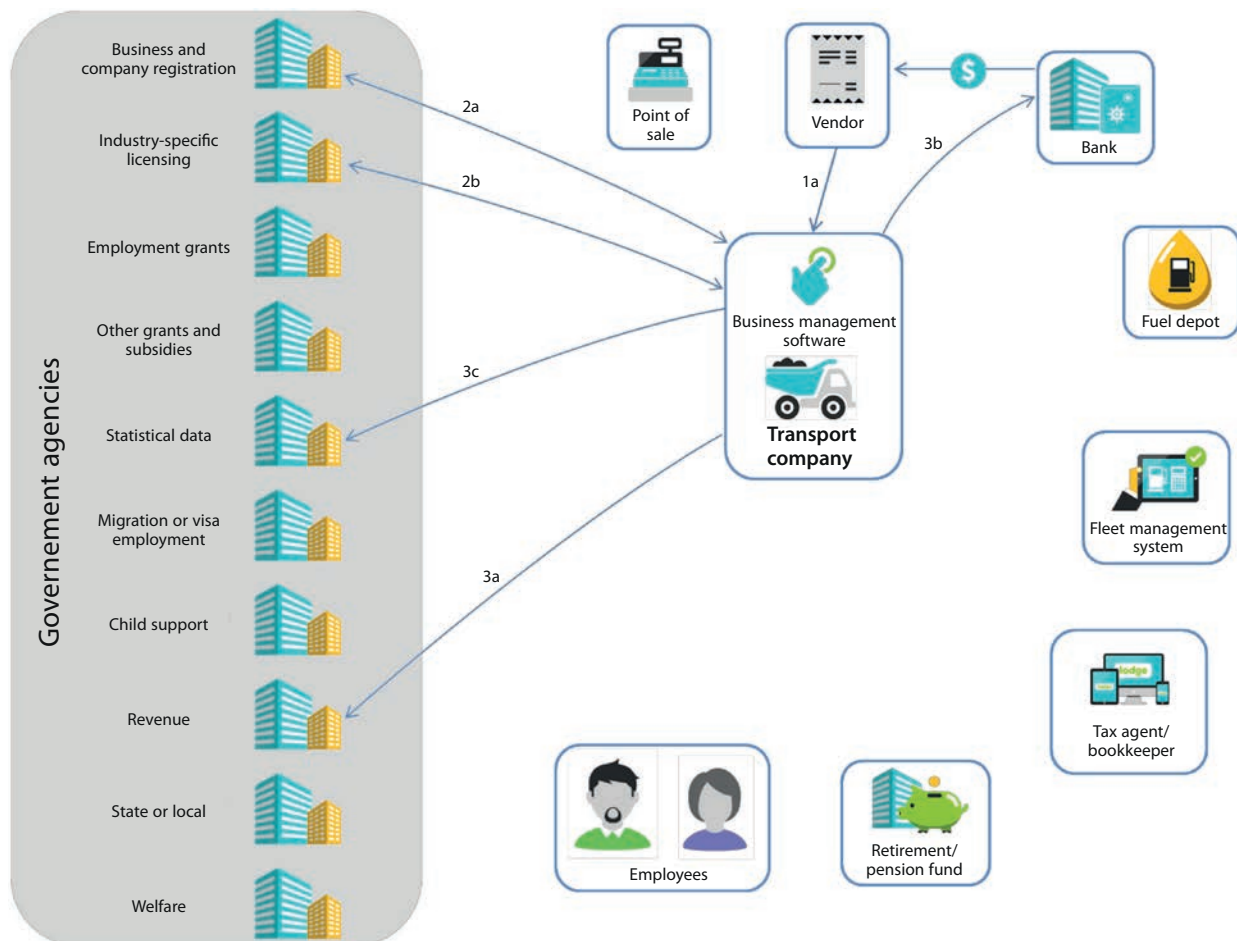
Source: Australia – Australian Tax Office (2018).

3.2. A transport management company makes a contract payment

200. From their business management software (BMS), the company makes a contract payment. All checks must pass for the transaction to be completed. The following interactions will happen almost instantaneously and there is the potential for several of the interactions to occur simultaneously:

- **1a.** A vendor sends an invoice to the BMS of the transport company
- **2a.** BMS checks vendor details are valid with the Agency responsible for Business and Company Registration.
- **2b.** BMS checks vendor details are valid with the Agency responsible for Industry Specific Licensing.
- **3a.** BMS sends contract payment details to the Revenue Agency.
- **3b.** BMS sends contract payment details to the Bank.
- **3c.** BMS issues non sensitive data to the agency responsible for collection of statistical data.
- **4.** Bank issues payment to the vendor.

Figure 3.2. A transport management company makes a contract payment



Source: Australia – Australian Tax Office (2018).

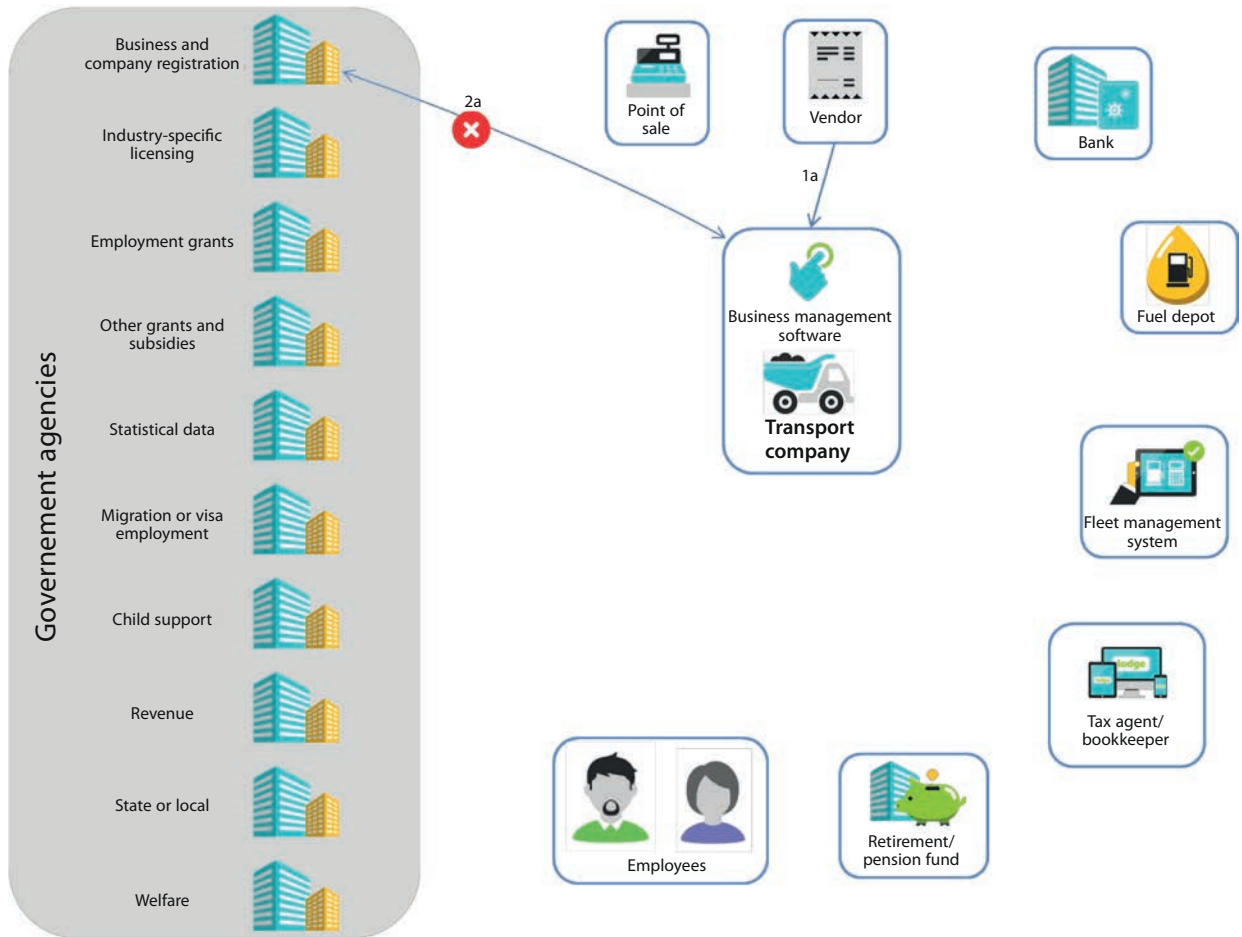
3.3. A transport management company attempts a contract payment

201. From their business management software (BMS), the company makes a contract payment. All checks must pass for the transaction to be completed. The following interactions will happen almost instantaneously and there is the potential for several of the interactions to occur simultaneously:

- **1a.** A vendor sends an invoice to the BMS of the transport company
- **2a.** BMS checks vendor details are valid with the Agency responsible for Business and Company Registration. It is identified that the vendor details are not valid and the transaction stops ensuring payment is not made to a potential unregistered business.

202. This example highlights how the system self-regulates where it identifies someone without a valid business registration. Payment is not made and in order to receive payment, the vendor is forced to participate in the system.

Figure 3.3. A transport management company attempts a contract payment



Source: Australia – Australian Tax Office (2018).

Annex A

ATO Lessons Learned

Engage with your digital supply chain

203. The Australian Taxation Office (ATO) has been delivering APIs using various techniques and technologies for over a decade. Throughout that time the ATO has observed a dramatic increase in technology maturity across its digital supply chain. For some early initiatives the ATO effectively dictated the technology solution and approach to its service consumers.

204. However, in recent years the organisation has recognised the benefits of close engagement and active collaboration with its digital supply chain. Early and ongoing engagement, and service co-design ensures that each API can be effectively and efficiently consumed in an appropriate, scalable and secure manner.

205. The ATO has learnt that poor technology choices and a “build it and they will come” mentality can materially impact on the successful adoption of an API offering. Instead, greater market tractions can be achieved through understanding the natural data lifecycle, building fit-for-purpose APIs and ensuring low barriers to entry. The only way to uncover the true value of the data held across the supply chain is to engage in a joint discovery exercise with that supply chain.

206. There continues to be a growing understanding across the ATO that engaging and enabling Digital Service Providers (DSPs) is critical to ensuring that ATO APIs are consumed effectively. This ensures value is surfaced for the ultimate end users, the participants in the Australian Taxation and Superannuation systems.

Market test your products early

207. For much of the last decade the ATO has followed a traditional enterprise-scale waterfall software delivery methodology. As with all waterfall methods, heavy investment in design, build and test occurs before a product or service is ready to expose to the market.

208. In an API world this can have significant impacts on the overall time to consumer value. On various occasions third-party API consumers have waited several months to access documentation and machine consumable API end-points only to identify design flaws and need to wait for a subsequent cycle to correct the bug(s).

209. The ATO is continuing to focus on improvement in this area through the implementation of agile practices, early exposure of API documentation, and collaborative co-design. However, there is further work to do in this area and the ATO are actively investigating opportunities presented by OpenAPI specifications and DevOps enabled agile.

210. Organisations should seek to expose API products early, while listening to and actively addressing feedback.

API Governance – DSP Operational Framework

211. The ATO has developed its DSP Operational Framework as part of its recognition and response to risks posed by API based (wholesale) digital services. It specifically identifies the need to improve:

- enrolment, identification, authentication and authorisation technologies available to clients
- registration and certification of DSPs who are allowed to use the APIs being made available by the ATO
- the maturity of their monitoring environments.

212. The ATO has recognised that APIs are creating new opportunities for Taxation and Superannuation, but also need new frameworks and techniques to ensure trust and security across the ecosystem. The ATO has identified the following compounding risk and opportunity factors:

- increased community connectivity and demand for availability (more access “anytime”)
- increased range in types of APIs available (including data IN and data OUT)
- exponential increases in volumes and payloads of API transactions
- significant growth in the range, number and complexity of digital service providers
- increased digital enrolments and consumption (e.g. by businesses, agents, others)
- increased throughput velocity in ATO (i.e. more transactions are processing in real time)
- increased automation in community and by service providers (enables “mass” volumes).

213. To address the increased risk of cybercrime, the ATO DSP Operational Framework determines if access to APIs (and ultimately organisation data) is permitted based on three dimensions:

- API Risk Category – see Risk Management section above
- DSP Trust Rating – determined by registration and certification processes, and prior performance
- API Monitoring Maturity – how mature the ATO’s monitoring and behavioural analytics capability is in relation to the specific API information domain.

214. The ATO has further identified a suite of additional controls that must be implemented for medium to high risk APIs and DSPs with a large customer base to increase a DSP’s Trust Rating or to mitigate real-time transactional and data risks. These mitigations include:

- multi-factor authentication in the client-facing system
- appropriate onshore/offshore application and data hosting arrangements
- enhanced supply chain visibility
- encryption of data in transit
- encryption of data at rest
- recognised IT security certification and/or accreditation.

215. The ATO’s critical learning is to ensure that controls are fit-for-purpose and appropriately risk-based. Additionally, transition in timelines should seek to balance the need for security with the commercial impact on the DSP market.

Cyber-security is critical

216. Government provided APIs and digital services will always be an attractive target for malicious actors. As such, API providers should ensure they plan for end-to-end API security, applying fit-for-purpose security controls that align to the risk associated with the API and the API platform. The ATO actively works with the Australian Signals Directorate to implement best practices and government specified security controls. Often implementation of controls to address emerging threats can be challenging. The implementation of continuous delivery and containerisation techniques can aid organisations to rapidly tear-down and patch environments.

217. Organisations should also take steps to ensure their systems integrators are appropriately secure and reliable.

Judicious use of standards

218. Some of the ATO’s early API endeavours and indeed business services were hampered by technology selection and the use of immature standards. The ATO has often been at the forefront of digital technologies and found itself in a position where it needs to select a technology stack or a technology standard with minimal global application or experience relevant to the Taxation or Superannuation domains.

219. The ATO recognises that some of its historical technology choices have subsequently suffered from lack of broad adoption and resulted in difficulty accessing skills and expertise. Additionally, an early standard will often contain many features that the authors have identified as potentially useful. However, some features will never make it into widespread commercially available implementations.

220. The ATO continues to recognise the potential value of leveraging international standards but has adjusted its approach to the adoption of standards to consider the following factors:

- only use well adopted standards or be prepared to foster the market to support a new standard;
- standards will change over time, be ready for change; immature standards may change very rapidly or prove difficult to implement;
- standards may contain a high-degree of optionality; and
- standards may contain ambiguity that will only be resolved through implementation and interoperability testing.

Provide the right enablers

221. Most successful API strategies and implementations have a common success factor, the API publisher has provided the right enablers to support API consumers. Enablers will vary from project to project, and API to API but the ATO has learnt that missing enablers

or poor-quality enablers is guaranteed to impact on adoption and generation of value. For example:

- Poor implementation guidance material – increases the support overhead, introduces time to value delays and may result in a poor user experience or data quality.
- SDKs are not generally core business for an organisation – if they can't be generated then they become a maintenance burden and a barrier to innovation. Organisations should only create SDKs if they generate unique market value or the current market maturity requires the organisation to simplify API consumption and lower barriers to entry.
- The API testing experience needs to be as realistic as possible – artificial stubs or contrived test cases don't reflect the real world, making it difficult for API consumers to ensure a predictable experience for end users.

222. Organisations should invest to provide the right enablers and expose them to API consumers early in the delivery lifecycle.

Annex B

Glossary of terms

Table B.1. Definitions for key terms used throughout the document

Term	Synonyms/ abbreviation	Definition
Apache 2.0 License		The Apache License is a free and open source software (FOSS) licensing agreement from the Apache Software Foundation (ASF). The agreement stipulates terms for use, reproduction, modification and distribution of any software that is released under the Apache License.
Application Programming Interface	API	A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service.
API Gateway		An API gateway is programming that sits in front of an application programming interface (API) and filters traffic or performs other cross-cutting concerns.
AS4 profile of the ebMS3	AS4	The AS4 profile of the ebMS 3.0 specification has been developed in order to bring continuity to the principles and simplicity that made AS2 successful, while adding better compliance to Web Services standards, and features such as message pulling capability and a built-in receipt mechanism.
Australian Business Number	ABN	An Australian Business Number (ABN) is a unique 11 digit number that identifies your business to the government and community.
Australian Business Register	ABR	The Australian Business Register (ABR) stores details about businesses and organisations when they register for an Australian business number (ABN). The Register is recognised nationally as a valuable asset and is used by the community and government daily to identify and verify business information.
Authentication		The process of verifying that someone or something is the actual entity that they claim to be.
Authorisation		Authorisation is the process of determining whether an authenticated subject (a user) can see, change, delete or take other actions upon data.
Blacklist		When performing input validation, the set of items that – if matched – result in the input being considered invalid. If no invalid items are found, the result is valid.
Brute Force Attack		An attack on an encryption algorithm where the encryption key for a ciphertext is determined by trying to decrypt with every key until valid plaintext is obtained.
Claim		Piece of information asserted about an Entity
Code Signing		Signing executable code to establish that it comes from a trustworthy vendor. The signature must be validated using a trusted third party in order to establish identity
Commercial off the Shelf	COTS	Commercially available specialised software designed for specific applications that can be used with little or no modification.
Confidentiality		Confidentiality is roughly equivalent to privacy. Measures undertaken to ensure confidentiality are designed to prevent sensitive information from reaching the wrong people, while making sure that the right people can in fact get it:
Conformance		Fulfilment of an implementation of all requirements specified; adherence of an implementation to the requirements of one or more specific specifications or standards

Term	Synonyms/ abbreviation	Definition
Creative Commons		A Creative Commons (CC) licence is one of several public copyright licences that enable the free distribution of an otherwise copyrighted work. A CC licence is used when an author wants to give people the right to share, use, and build upon a work that they have created.
Credential		Data presented as evidence of the right to use an identity or other resources.
Default deny		A paradigm for access control and input validation where an action must explicitly be allowed. The idea behind this paradigm is that one should limit the possibilities for unexpected behaviour by being strict, instead of lenient, with rules.
Denial of Service Attack	DoS	Any attack that affects the availability of a service. Reliability bugs that cause a service to crash or go into some sort of vegetative state are usually potential denial-of-service problems.
DevOps		DevOps is the combination of cultural philosophies, practices, and tools that increases an organisation's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organisations using traditional software development and infrastructure management processes.
Digital Service Provider	DSP	Digital Service Providers (DSP) are critical enablers in the delivery of services to the community/end-user. They offer a range of digital products to the community (e.g. accounting and payroll software, super services, banking services, business registration and licensing services etc). They use software to enable their services and they may choose to connect their software to government services via APIs.
Digital Signature		Data that proves that a document (or other piece of data) was not modified since being processed by a particular entity.
Digital Supply Chain		The digital supply chain is a process of networking and applications between individuals and organisations involved in a transaction that is initiated in a paperless environment, using web-enabled capabilities.
ebXML Messaging Services Version 3	EBMS3	This specification defines a communications-protocol neutral method for exchanging electronic business messages. It defines specific Web Services-based enveloping constructs supporting reliable, secure delivery of business information. (OASIS, 2007a)
Enterprise Service Bus	ESB	An enterprise service bus (ESB) implements a communication system between mutually interacting software applications in a <u>service-oriented architecture</u> (SOA).
External Vendor Test Environment	EVTE	An environment used by DSPs to test their API client software packages before deploying to production.
Gateway		Gateways securely manage data flows between connected networks from different security domains.
GNU General Public License	GNU GPL	The GNU General Public License is a free, copyleft licence for software and other kinds of works.
HATEOS		HATEOAS (Hypermedia as the Engine of Application State) is a constraint of the REST application architecture. A hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses.
HTTP		HTTP means HyperText Transfer Protocol. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.
HTTP Header		HTTP headers are the name or value pairs that are displayed in the request and response messages of message headers for Hypertext Transfer Protocol (HTTP).
HTTP Method		HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred as HTTP verbs
HyperMedia		The concept of providing links to other resources. Hypermedia is one of the key principles to a REST architecture
IaaS		Infrastructure as a service (IaaS) is a form of cloud computing that provides virtualised computing resources over the internet.

Term	Synonyms/ abbreviation	Definition
Idempotency		From a RESTful service standpoint, for an operation (or service call) to be idempotent, clients can make that same call repeatedly while producing the same result. In other words, making multiple identical requests has the same effect as making a single request. Note that while idempotent operations produce the same result on the server (no side effects), the response itself may not be the same (e.g. a resource's state may change between requests).
Identity		A person's identity is not a fixed concept; it is highly dependent on context. It is some combination of characteristics or attributes that allow a person to be uniquely distinguished from others within a specific context
Identity Proofing		An identity proofing process tests the veracity of claims an individual makes regarding their identity
Integrity		Integrity involves maintaining the consistency, accuracy, and trustworthiness of data over its entire life cycle
Integrity Checking		The act of checking whether a message has been modified either maliciously or by accident.
International Organisation for Standardization	ISO	ISO is an independent, non-governmental international organisation with a membership of 160 national standards bodies. Through its members, it brings together experts to share knowledge and develop voluntary, consensus-based, market relevant International Standards that support innovation and provide solutions to global challenges.
Internet Engineering Task Force	IETF	The IETF is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.
JavaScript Object Notation	JSON	JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition, December 1999.
JSON Web Algorithms	JWA	JSON Web Algorithms provides cryptographic algorithms and identifiers to be used with the JSON Web Signature, JSON Web Encryption and JSON Web Key. (IETF, 2015e)
JSON Web Encryption	JWE	JSON Web Encryption provides support for encrypted content using JSON-based data structures. (IETF, 2015c)
JSON Web Key	JWK	JSON Web Key provides a data structure that represents a cryptographic key. (IETF, 2015d)
JSON Web Signatures	JWS	JSON Web Signatures support a data structure representing a digitally signed message. (IETF, 2015b)
JSON Web Tokens	JWT	JSON Web Tokens (JWT) is a standard for passing authenticated identity information between an identity provider and service provider similar to SAML but optimised for JSON. JWT Relies on other standards: JWS, JWE. (IETF, 2015a)
Message Authentication Code	MAC	A function that takes a message and a secret key (and possibly a nonce) and produces an output that cannot, in practice, be forged without possessing the secret key.
Message integrity		A message has integrity if it maintains the value it is supposed to maintain, as opposed to being modified on accident or as part of an attack.
MIT License		The MIT License is a permissive free software licence originating at the Massachusetts Institute of Technology (MIT). As a permissive licence, it puts only very limited restriction on reuse and has, therefore, an excellent licence compatibility
Non-repudiation		The capability of establishing that a message was signed by a particular entity. That is, a message is said to be non-repudiable when a user sends it, and one can prove that the user sent it. In practice, cryptography can demonstrate that only particular key material was used to produce a message. There are always legal defences such as stolen credentials or duress.
OAUTH		OAuth is an authorisation framework enabling a third-party application limited access to an HTTP service.
Open ID Connect	OIDC	OIDC provides an identity layer on top of the OAuth2.0 protocol. (OpenID Foundation, 2014a)

Term	Synonyms/ abbreviation	Definition
Open Source		Denoting software for which the original source code is made freely available and may be redistributed and modified.
Organisation for the Advancement of Structured Information Standards	OASIS	OASIS is a non-profit consortium that drives the development, convergence and adoption of open standards for the global information society.
OpenAPI Specification	OAS Swagger	The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.
OpenPEPPOL	PEPPOL	OpenPEPPOL is a non-profit international association and consists of both public sector and private members. The purpose of OpenPEPPOL is to enable European businesses to easily deal electronically with any European public-sector buyers in their procurement processes, thereby increasing opportunities for greater competition for government contracts and providing better value for tax payers' money.
Open Web Application Security Project	OWASP	The Open Web Application Security Project (OWASP) is a 501(c)(3) worldwide not-for-profit charitable organisation focused on improving the security of software.
PaaS		Platform as a Service (PaaS) or application platform as a Service (aPaaS) or platform base service is a category of <u>cloud computing services</u> that provides a <u>platform</u> allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure
Personally Identifiable Information	PII	Information that: a. can be used to identify the natural person to whom such information relates, or b. is or might be directly or indirectly linked to a natural person to whom such information relates.
Private key		In a public key cryptosystem, key material that is bound tightly to an individual entity that must remain secret in order for there to be secure communication.
Proxy (server)		A proxy server is a dedicated computer or a software system running on a computer that acts as an intermediary between an endpoint device, such as a computer, and another server from which a user or client is requesting a service.
Public key		In a public key cryptosystem, the key material that can be published publicly without compromising the security of the system. Generally, this material must be published; its authenticity must be determined definitively.
Public Key Infrastructure	PKI	A system that provides a means for establishing trust as to what identity is associated with a public key.
Request For Comments	RFC	Each distinct version of an Internet standards-related specification is published as part of the "Request for Comments" (RFC) document series. RFCs cover a wide range of topics in addition to Internet Standards, from early discussion of new research concepts to status memos about the Internet.
SaaS		Software as a service is a software <u>licensing</u> and <u>delivery</u> model in which <u>software</u> is licensed on a <u>subscription</u> basis and is centrally <u>hosted</u> .
SAML		Security Assertion Markup Language (SAML) enables Cross Domain Single Sign On by allowing one computer to perform security functions (authentication and/or authorisation) on behalf of another relying party. SAML is based on XML.
SAML Provider		A SAML Provider is an entity that undertakes authentication functions on behalf of a Relying Party.
SAML Assertion		A SAML Assertion is the XML document exchanged between the SAML provider and the Relying Party. The SAML Assertion may contain one or more Claims.
Scalable		The ability of the system to increase/decrease the capacity of specific system or solution components to meet current processing requirements
Secure Socket Layer	SSL	A popular protocol for establishing secure channels over a reliable transport, utilising a standard X.509 Public Key Infrastructure for authenticating machines. This protocol has evolved into the TLS protocol, but the term SSL is often used to generically refer to both. See TLS

Term	Synonyms/ abbreviation	Definition
Service Level Agreement	SLA	A Service Level Agreement (SLA) is an agreement between an IT Service Provider and a Customer that is outside of IT. An SLA describes the service being delivered, documents service level targets and specifies the responsibilities of the IT Service Provider and the Customer. An SLA describes expectations of the Customer and a commitment by the Service Provider to meet those expectations.
Slack		A set of cloud based tools and services. Slack is an acronym which stands for Searchable Log of All Conversation and Knowledge.
SOAP		SOAP Version 1.2 is a lightweight protocol intended for exchanging structured information in a decentralised, distributed environment. (W3C, 2007a)
Software Developer Kit	SDK	A software development kit (SDK or devkit) is typically a set of software development tools that allows the creation of applications for a certain software package, software framework or hardware platform.
Specification		Specifications are often published when the subject under question is still under development or when insufficient consensus for approval of an International Standard is available. Specifications approach International Standards in terms of detail and completeness, but have not yet passed through all approval stages either because consensus has not been reached or because standardisation is seen to be premature
Standard		A standard is a document, established by consensus and approved by a recognised body, that provides, for common and repeated use, rules, guidelines or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context.
Swagger Specification	OpenAPI Specification	The Swagger project was the original foundation of the OAS, consisting of the Swagger Specification and the Swagger tools. The Swagger brand has been retained for the tooling, whilst the specification has been transitioned to OAS.
Tincup		Tincup is a microservice implemented by UBER to provide currency and exchange rate services.
Transport Layer Security	TLS SSL	The successor to SSL, a protocol for establishing secure channels over a reliable transport, using a standard X.509 Public Key Infrastructure for authenticating machines. The protocol is standardised by the IETF.
Validation		The act of determining that data is sound. In security, generally used in the context of validating input.
W3C		The World Wide Web Consortium (W3C) is an international community where Member organisations , a full-time staff , and the public work together to develop Web standards .
Web API	API	Modern web scale APIs that use the same underlying protocol as between the web browser and web server. Instead of returning code that is interpreted by the web browser and rendered as a human readable web page, the API returns machine readable code
Web Service		The term Web services describes a standardised way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards
Web Services Description Language	WSDL	An XML language for describing Web services.
Whitelist		When performing input validation, the set of items that, if matched, results in the input being accepted as valid. If there is no match to the whitelist, then the input is considered invalid. That is, a whitelist uses a "default deny" policy.
WS-*		WS-* is a prefix used to indicate specifications associated with web services and there exist many WS* standards, as outlined in the Quality of Service section of this document.
X.509 Certificate		A digital certificate that complies with the X.509 standard (produced by ANSI).
XBRL		XBRL (eXtensible Business Reporting Language) is a freely available and global standard for exchanging business information. XBRL allows the expression of semantic meaning commonly required in business reporting
XML		XML stands for eXtensible Markup Language. XML was designed to store and transport data. XML was designed to be both human- and machine-readable.
XSD		XSD (XML Schema Definition) is a World Wide Web Consortium (W3C) recommendation that specifies how to formally describe the elements in an Extensible Mark-up Language (XML) document.

References

- Amundsen, M. (2013), *Collection+JSON – Document Format*, <http://amundsen.com/media-types/collection/format/> (accessed on 11 December 2018).
- Armerding, T. (2018), *The 17 biggest data breaches of the 21st century*, www.csoonline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html (accessed on 11 December 2018).
- Atlassian (2018), *Continuous integration vs. continuous delivery vs. continuous deployment*, www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd (accessed on 11 December 2018).
- Australian Taxation Office (2016), *SuperStream legislation, standards and schedules*, www.ato.gov.au/Super/SuperStream/In-detail/Legal-framework/Legislative-instrument/SuperStream-legislation,-standards-and-schedules/ (accessed on 11 December 2018).
- Bell, T. (2018), *5 myths of API security*, <https://www.cso.com.au/article/635902/5-myths-api-security/> (accessed on 11 December 2018).
- Cloud Elements (2018), *State of API Integration 2018 Report*, <https://offers.cloud-elements.com/hubfs/cld-2018-soai-final-2018.pdf?t=1526679859334> (accessed on 11 December 2018).
- Digital Transformation Agency (2017), *Digital Transformation Culture Posters*, <https://dta-www-drupal-20180130215411153400000001.s3.ap-southeast-2.amazonaws.com/s3fs-public/files/digital-service-standard/dta-culture-posters-wcag.pdf> (accessed on 11 December 2018).
- Digital Transformation Agency (2016), *Digital Service Standard: 3. Agile and User-Centred Process*, <https://guides.service.gov.au/digital-service-standard/3-agile-and-user-centred/> (accessed on 11 December 2018).
- ECMA International (2017), *The JSON Data Interchange Syntax*, www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (accessed on 11 December 2018).
- European Commission (2018), *Data Protection: Rules for the protection of personal data inside and outside the EU*, https://ec.europa.eu/info/law/law-topic/data-protection_en (accessed on 11 December 2018).
- Forrester (2017), *The Forrester Wave™: Software Composition Analysis, Q1 2017*, Forrester Research, Inc.
- Fowler, M. (2010), *Richardson Maturity Model*, <https://martinfowler.com/articles/richardsonMaturityModel.html> (accessed on 11 December 2018).
- Fowler, M. and J. Lewis (2014), *Microservices: A definition of this new architectural term*, <https://martinfowler.com/articles/microservices.html> (accessed on 11 December 2018).

- Gartner (2018), *Assessing Microservices for Agile Application Architecture and Delivery*, Gartner.
- Greenberg, A. (2016), *So Hey You Should Stop Using Texts For Two-Factor Authentication*, www.wired.com/2016/06/hey-stop-using-texts-two-factor-authentication/ (accessed on 11 December 2018).
- Gruman, G. (2007), *Strategies for Dealing With IT Complexity*, www.cio.com/article/2437606/it-organization/strategies-for-dealing-with-it-complexity.html (accessed on 11 December 2018).
- Guckenheimer, S. (2017b), *What is continuous delivery?*, <https://docs.microsoft.com/en-us/azure/devops/what-is-continuous-delivery> (accessed on 11 December 2018).
- Guckenheimer, S. (2017a), *What is Monitoring?*, <https://docs.microsoft.com/en-us/azure/devops/what-is-monitoring> (accessed on 11 December 2018).
- Health Level Seven International (2018), *Introduction to HL7 Standards*, www.hl7.org/implement/standards/ (accessed on 11 December 2018).
- HL7 Community (2017), *HL7 FHIR Release 3 (STU) Resource Formats*, www.hl7.org/fhir/formats.html (accessed on 11 December 2018).
- IETF (2014), *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, <https://tools.ietf.org/html/rfc7230> (accessed on 11 December 2018).
- IETF (2008), *The Transport Layer Security (TLS) Protocol: Version 1.2*, <https://tools.ietf.org/html/rfc5246> (accessed on 14 June 2018).
- IETF (2015e), *JSON Web Algorithms (JWA)*, <https://tools.ietf.org/html/rfc7518> (accessed on 11 December 2018).
- IETF (2015c), *JSON Web Encryption (JWE)*, <https://tools.ietf.org/html/rfc7516> (accessed on 11 December 2018).
- IETF (2015d), *JSON Web Key (JWK)*, <https://tools.ietf.org/html/rfc7517> (accessed on 11 December 2018).
- IETF (2015b), *JSON Web Signature (JWS)*, <https://tools.ietf.org/html/rfc7515> (accessed on 11 December 2018).
- IETF (2015a), *JSON Web Token (JWT)*, <https://tools.ietf.org/html/rfc7519> (accessed on 11 December 2018).
- IETF (2012a), *The OAuth 2.0 Authorization Framework*, <https://tools.ietf.org/html/rfc6749> (accessed on 11 December 2018).
- IETF (2012b), *The OAuth 2.0 Authorization Framework: Bearer Token Usage*, www.ietf.org/rfc/rfc6750.txt (accessed on 11 December 2018).
- Indrasiri, K. (2016), *Microservice in Practice – Key Architectural Concepts of an MSA*, <https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/> (accessed on 11 December 2018).
- ISO20022 Registration Management Group (2018), *ISO 20022 and JSON: An implementation Best Practices Whitepaper*, www.iso20022.org/sites/default/files/documents/general/ISO20022_API_JSON_Whitepaper_Final_20180129.pdf (accessed on 11 December 2018).

- Kelly, M. (2018), *HAL – Hypertext Application Language*, <http://stateless.co/hal-specification.html> (accessed on 11 December 2018).
- Lakhani, M. and K. Iansiti (2017), *The Truth About Blockchain*, <https://hbr.org/2017/01/the-truth-about-blockchain> (accessed on 11 December 2018).
- Microsoft Corporation (2017), *Microservices architecture style*, <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices> (accessed on 11 December 2018).
- Microsoft Corporation (n.d.), *Architecture Styles*, <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/> (accessed on 11 December 2018).
- ModicaGroup (2018), *Optus SMS Suite Mobile Gateway REST API*, <https://confluence.modicagroup.com/display/OPTUS/Optus+SMS+Suite+Mobile+Gateway+REST+API> (accessed on 11 December 2018).
- MuleSoft (2017), *Best practices for microservices: Implementing a foundation for continuous innovation*.
- OASIS (2013), *AS4 Profile of ebMS 3.0 Version 1.0*, <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/AS4-profile/v1.0/os/AS4-profile-v1.0-os.html> (accessed on 11 December 2018).
- OASIS (2011), *OASIS ebXML Messaging Services Version 3.0: Part 2, Advanced Features*, <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/part2/201004/ebms-v3-part2.html> (accessed on 11 December 2018).
- OASIS (2008), *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1*, <http://docs.oasis-open.org/ws-rx/wsrn/200702/wsrn-1.1-spec-os-01-e1.html> (accessed on 11 December 2018).
- OASIS (2004), *Web Services Reliable Messaging TC – WS-Reliability 1.1*, http://docs.oasis-open.org/wsrn/ws-reliability/v1.1/wsrn-ws_reliability-1.1-spec-os.pdf (accessed on 11 December 2018).
- OASIS (2007a), *OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features*, http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.html (accessed on 11 December 2018).
- OASIS (2007b), *Web Services Context Specification (WS-Context) Version 1.0*, <http://docs.oasis-open.org/ws-caf/ws-context/v1.0/OS/wsctx.html> (accessed on 11 December 2018).
- OASIS (2009d), *Web Services Coordination (WS-Coordination) Version 1.2*, <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os/wstx-wscoor-1.2-spec-os.html> (accessed on 11 December 2018).
- OASIS (2009a), *Web Services Dynamic Discovery (WS-Discovery) Version 1.1*, <http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html> (accessed on 11 December 2018).
- OASIS (2009b), *Web Services Federation Language (WS-Federation) Version 1.2*, <http://docs.oasis-open.org/wsfed/federation/v1.2/os/ws-federation-1.2-spec-os.html> (accessed on 28 December 2018).
- OASIS (2012a), *Web Services Security: SOAP Message Security Version 1.1.1*, <http://docs.oasis-open.org/wss-m/wss/v1.1.1/os/wss-SOAPMessageSecurity-v1.1.1-os.html> (accessed on 11 December 2018).

- OASIS (2009c), *WS-SecureConversation 1.4*, <http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.html> (accessed on 11 December 2018).
- OASIS (2012b), *WS-Trust 1.4*, <http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/errata01/ws-trust-1.4-errata01-complete.html> (accessed on 11 December 2018).
- OpenID Foundation (2014a), *OpenID Connect Core 1.0 incorporating errata set 1*, http://openid.net/specs/openid-connect-core-1_0.html (accessed on 11 December 2018).
- OpenID Foundation (2014b), *Welcome to OpenID Connect*, <https://openid.net/connect/> (accessed on 11 December 2018).
- OxfordDictionaries.com (2018), *API*, <https://en.oxforddictionaries.com/definition/api> (accessed on 11 December 2018).
- Pagh-Rasmussen, N. and S. Fieten (2017), *OpenPEPPOL CC F2F meetings – AS4 Transition*, <https://peppol.eu/wp-content/uploads/2017/05/AS4-transition-Vienna-CC-Sander-and-Niels-0905-2017.pdf> (accessed on 11 December 2018).
- Patronus Laboratories Corporation (2019), *Patronuslabs*, <https://www.patronuslabs.com/products/liability-vs-roi-analysis> (accessed on 6 February 2019).
- Sandoval, K. (2015), *API Security: The 4 Defenses of The API Stronghold*, <https://nordicapis.com/api-security-the-4-defenses-of-the-api-stronghold/> (accessed on 11 December 2018).
- Sarid, U. (2017), *Open API and RAML: Better Together*, <https://blogs.mulesoft.com/dev/api-dev/open-api-raml-better-together/> (accessed on 11 December 2018).
- Telstra (2018), *Telstra Messaging API*, <https://dev.telstra.com/content/messaging-api#> (accessed on 11 December 2018).
- ThoughtWorks (2018), *Continuous Delivery*, www.thoughtworks.com/continuous-delivery (accessed on 11 December 2018).
- W3C (2014), *JSON-LD 1.0: A JSON-based Serialization for Linked Data*, www.w3.org/TR/json-ld/ (accessed on 11 December 2018).
- W3C (2007a), *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, www.w3.org/TR/soap12/ (accessed on 11 December 2018).
- W3C (2007b), *Web Services Policy 1.5 – Framework*, www.w3.org/TR/2007/REC-ws-policy-20070904/ (accessed on 11 December 2018).
- W3C (2006a), *Web Services Addressing 1.0 – Core*, www.w3.org/TR/ws-addr-core/ (accessed on 11 December 2018).
- W3C (2006b), *Extensible Markup Language (XML) 1.1 (Second Edition)*, www.w3.org/TR/xml11/ (accessed on 11 December 2018).
- XBRL International (2018), *An Introduction to XBRL*, www.xbrl.org/the-standard/what-an-introduction-to-xbrl/ (accessed on 11 December 2018).

Unlocking the Digital Economy - A Guide to Implementing Application Programming Interfaces in Government

New digital technologies are reshaping the economy, leading to the development of new products, services and business models and creating new ways for citizens and businesses to interact in their daily lives. They are also allowing tax administrations to be more data and service driven, with increasing use of proactive tools for engaging with taxpayers, greater use of third party data and increasing use of advanced analytics to better target interventions. This in turn offers opportunities to make tax a more seamless process, with easier self-service, reductions in burdens and enhanced compliance.

A key enabler of these changes is the use of Application Programming Interfaces (APIs). This is the functionality that connects systems, people and things without facilitating direct access, an invisible process that people already use every day on mobile phones and via the internet.

This report, which is aimed at the more specialist reader within tax administrations, provides an overview of the practices, techniques and standards used to deliver contemporary and effective digital services for taxpayers. It is intended to provide practical assistance to tax administrations, and other parts of government, which are seeking to implement or further develop their API strategy.