Sustainable API Green Score

API Numériquement Responsable

APITHINKING

The API Green Score is a toolkit to help API users, designers and owners to ask themselves questions about the digital impact of their API

This tool is based on 7 different domains in order to create relevant and realistic metrics that stakeholders can use.

The evaluation method is shared with all API Persona
(API owners, API consumers, API developers)

API GREEN SCORE

Excellent	Acceptable	Average	Poor	Very Poor
A	B	C	D	E

7 domains

API Lifecycle

- Decommission an unused API
- Deploy API near consumer
- Reduce number of API versions
- Unify API catalog
- Create consumer referential
- Identify API for single usage
- Urbanization with Data Governance

Data Exchange

- Exchange with Smallest Size
- Following API payload size
- Prefer Opaque Token to JWT
- API Customer Centricity principles
- API Data / Granularity
- Leverage Odata or GraphQL for DB APIs
- Data Management
- Dynamic Content

Data

- Optimize queries to limit returned information
- Collect only required data
- Provide only changed data
- Use cache
- Communicate on Payload size
- API used geolocally close to their consumers

Architecture

- Promote event architecture
- Filter data in payload
- Pagination
- Webhook or Business Notification
- AsyncAPI

Tools

- Define a basis of criteria for rating
- Provide KPIs (Nb of call, payload size, nb of equipments used, ...)
- Evaluate energy consumption for one API
- Know language impact for energy consumption

Infrastructure

- Use adaptive infrastructure
- Use as few cloud suppliers as possible between consumer and backend
- Be near Data Center
- Define which actions are more relevant to do to reduce the impact of API ?

Communication

- Name of API Ecoscore
- Guideline resources
- Sharing criteria of evaluation and methods
- Adapt the communication of each personas

API Lifecycle

API uses: (who, when, what)

Description

Have a consumer referential

What is the impact of this referential on the API Green Score?

Who consumes my API?

What : Which version of API?

When : Which number of asked calls vs number of calls ? Date of last call?

What is the calls volume ?

Governance

API Product Owner

Center of Expertise API

KPI per API

Nb of call per consumer

Nb of consumers per API

Nb of versions per API (US03)

Location of consumers

Documentation quality (US06)

What is the API Footprint?

Tools to measure

Logs API / Operational Reporting

Analytics API Gateway

To influence the Metrics

API Gateway/API Portal

Example

API Order 10000/ call / month

API last Call

Nb of Consumers who used this API

Impact EcoScore

20%

APITHINKING

Data Exchange

API uses: How to exchange information between information systems

Description

Make sure APIs are eco designed

How we exchange data,

Payload size

Message type / integration pattern type

Call frequency / cache

Impact EcoScore

20%

Governance

API Product Owner

API developer

Application Owner

Tools to measure

Api gateway / API portal

E2E observability

KPI per API

Nb of calls per consumer

Calls volume

API Payload average size (DE11)

Integration Layer Payload

Integration pattern

Cache performance (DE01, DE02, DE03)

Examples

Filtering data for calling backend

Orchestration

Consume only relevant data

Data

API uses: Govern Business Object *(naming convention, modelization, pivot format)*

Description

Understand the use case and type of data implicated

Business object

Expose only needed data

Data should only be stored in a single point of truth

Data should be stored in an unique and secured point

Impact EcoScore

20%

Governance

Data Product Owner

Application Owner

Tools to measure

Data Catalog

DataPedia, DataService

KPI per API

Cache management (DE03)

Payload average size

Data format (JSon/XML) (DE01)

Compressed Payload usage

Example

Shared business objects depending on usage

Avoid duplicated data

APITHINKING

Architecture

API uses: (who, how, what)

Description

How I design my integration flow

EDA + API

Which pattern is the most adapted for a use case?

multi-cloud/Hybrid Cloud (private/public/OnPremise) ?

Governance

Enterprise architecture team

API Product owner

KPI per API

Integration pattern used (coverage %)

Nb of cloud providers used

Impact EcoScore

25%

Tools to measure

E2E observability tool (Observability, APM)

Flow cartography

Example

Diagram flow

Tools

API uses: How to measure and evaluate the ecoscore ?

Description

How API gateway/ technology/integration tools could have an impact in the data flow

Governance

Depends on the organization (API Team, infra, etc..)

KPI per API

Tools usage rate

Tools carbon footprint (dev Portal included)

Impact EcoScore

N/A but required

Tools to measure

APM, Observability, EcoIndex, API Gateway Analytics, Carbon Footprint Cloud Provider

Example

Push log to a dedicated tool to analyze results of API calls

Implement rotation logs to aggregate data (rollup)

Infrastructure

API uses: (how, where, what)

Description

How far is the API Data Center from the API consumers/backend?

How many cloud provider, cloud Services, location of DC, between API consumer/API and API backend ?

is a scalable architecture used?

Governance

Enterprise Architecture

KPI per API

API latency

Multi-cloud usage

Impact EcoScore

10%

Tools to measure

End to End observability

API gateway

Cloud Provider reports

PUE : Power Usage Efficiency

Example

Customize footprint dashboard of cloud provider

APITHINKING

Communication, Learning

API uses: (who, how, when)

Description

How to share information around API use cases (CSR team, API Owners, Technical Users), Training

Governance

API COE

Marketing & communication

Sales

KPI per API

Time to live for an API usage

Number of visitors on the API Portal

API consumer number (US06)

Tools to measure

Portal API Gateway

Example

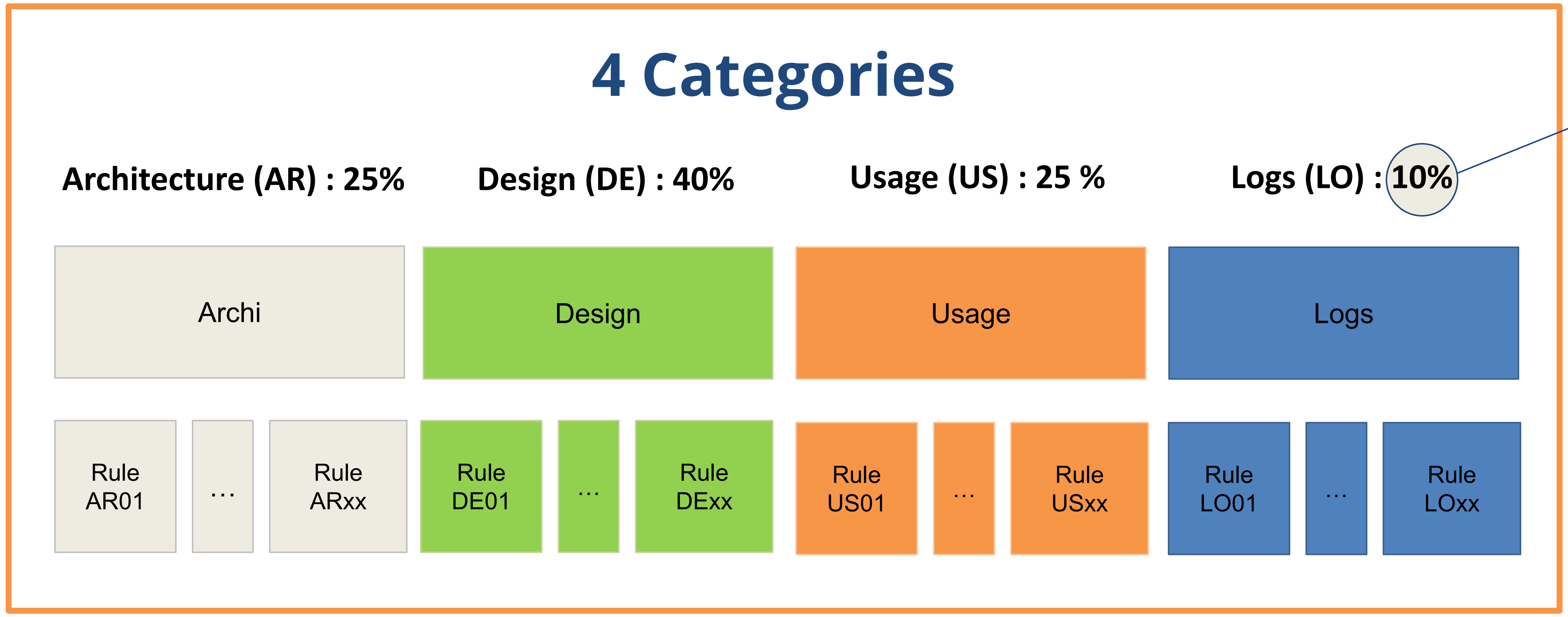
API launch (webinar, portal API, social media...)

Post launch API (social media, portal API, e-mail)

Impact EcoScore

5%

Evaluation Grid: Some precisions before starting



Each rule (U01, L01, L02…) has a score and its category has a weighting

Each category are rules based on 7 domains (Architecture, Data, API Lifecycle, Data Exchange, Tools, Infrastructure, Communication)

We have 2 ways to evaluate :

- Boolean : true/false
- Rate : calculation should be shared with persona, to avoid any misunderstanding

If some rules are not relevant - ex GraphQL (DE09), you can remove them from your referential and adapt with your own weighting

If some rules are not explicit enough, it is important to share them with all personas

Evaluation Grid: Results

When you fill the grid, a calculation will be done based on your response and weighting

A global note will be provided, should be matched with the range of each letter

Keep in mind to share the calculation in case of change between 2 periods

ex : eco-score

Présentation - Eco-score (score-environnemental.com)

API : Green Score Grid

Section	RuleID	Items analysed	Description	Weight		Score Evaluation				
						Points	Total Weight	Eval	Score	Comment
Architecture	AR01	Use Event Driven Architecture to avoid polling madness and inform subscribers of an update	Use Event Driven Architecture to avoid polling madness.	25%	25,0%	375	6,25%		0	
	AR02	API runtime close to the Consumer	Deploy the API near the consumer		25,0%	375	6,25%		0	
	AR03	Ensure the same API does not exist *	Ensure only one API fit the same need		25,0%	375	6,25%		0	
	AR04	Use scalable infrastructure to avoid over-provisioning	Use scalable infrastructure to avoid over-provisioning		25,0%	375	6,25%		0	
Design	DE01	Choose an exchange format with the smallest size (JSON is smallest than XML)	Prefer an exchange format with the smallest size (JSON is smaller than XML).	40%	25,0%	600	10,00%		0	
	DE02	new API --> cache usage	Use cache to avoid useless requests and preserve compute resources.		15,0%	360	6,00%		0	
	DE03	Existing API --> cache usage efficiency	Use the cache efficiently to avoid useless resources consumtion.		20,0%	480	8,00%		0	
	DE04	Opaque token usage	Prefer opaque token usage prior to JWT		2,0%	48	0,80%		0	
	DE05	Align the cache refresh with the datasource **	Align cache refresh strategy with the data source		4,0%	96	1,60%		0	
	DE06	Allow part refresh of cache	Allow a part cache refresh		4,0%	96	1,60%		0	
	DE07	Is System, Business or cx API ?	Use Business & Cx APIs closer to the business need		10,0%	240	4,00%		0	
	DE08	Possibility to filter results	Implement filtering mechanism to limit the payload size		2,5%	60	1,00%		0	
	DE09	Leverage OData or GraphQL for your databases APIs	Leverage OData or GraphQL when relevant		10,0%	240	4,00%		0	
	DE10	Redundant data information in the same API	Avoid redundant data information in the same API		5,0%	120	2,00%			
	DE11	Possibility to fitler pagination results	Implement pagination mechanism to limit the payload size		2,5%	60	1,00%		0	
Usage	US01	Use query parameters for GET Methods	Implement filters to limit which data are returned by the API (send just the data the consumer need).	25%	5,0%	75	1,25%		0	
	US02	Decomission end of life or not used APIs	Decomission end of life or not used APIs		10,0%	150	2,50%		0	
	US03	Number of API version <=2	Compute resources saved & Network impact reduced		10,0%	150	2,50%		0	
	US04	Usage of Pagination of results available	Optimize queries to limit the information returned to what is strictly necessary.		10,0%	150	2,50%		0	
	US05	Choosing relevant data representation (user don't need to do multiple calls) is Cx API ?	Choose the correct API based on use case to avoid requests on multiple systems or large number of requests. Refer to the data catalog to validate the data source.		20,0%	300	5,00%		0	
	US06	Number of Consumers	Deploy an API well designed and documented to increase the reuse rate. Rate based on number of different consumers		25,0%	375	6,25%	0%	0	this a rate evaluation
	US07	Error rate	Monitor and decrease the error rate to avoid over processing		20,0%	300	5,00%	0%	0	this a rate evaluation
Logs	LO01	Logs retention	Align log retention period to the business need (ops and Legal)	10%	100,0%	600	10,00%		0	
				100%		*6000*	*100%*		*Rank* 0 E	

Legend :

* > 70% redundant fields with on other API

** cache refresh must be equal to the data update frequency on the source system

Excellent Response	Acceptable Response	Average Response	Poor Response	Showstopper	Not evaluted
>=6000	6000<>=3000	3000<>=2000	2000<>=1000	<1000	N.C
A	B	C	D	E	N

Rules distribution in 7 categories

API Lifecycle

- AR03 : Ensure Only One API fits same need
- US02 : Decommission EOL or unused APIs
- US03 : Limit the number of API versions
- US05 : Choose the correct API based on use case
- US06 : API well designed and documented to increase reuse rate
- US07 : Monitor Error Rate

Data Exchange

- DE01 : Prefer smallest format for exchange (JSON instead of XML)
- DE02 : Use Cache
- DE03 : Use the cache efficiently to avoid useless resources consumption
- DE05 : Align Cache refresh strategy to data source
  DE07 : Is system, Business or CX API?
- DE08 : Implemented filtering mechanism to limit payload size
- DE11 : Availability of pagination
- US01 : Use query parameters for GET Methods

Data

- DE02 : Use Cache
- DE03 : Use the cache efficiently to avoid useless resources consumption
- Compressed Payload
- DE06 : Allow a part cache refresh and align it on data refresh
- DE09 : Leverage OData or GraphQL when relevant
- US04 : Optimize queries to limit the information returned to what is strictly necessary
- US05 : Choose the correct API based on use case

Architecture

- AR01 : Use Event Driven Architecture
- AR02 : API Runtime close to the consumer
- AR03 : Ensure Only One API fits same need

Tools

- LO01 : Define log Retention Period (ops and legal)

Insfrastructure

- AR05 Footprint dashboard of Cloud Provider
- AR04 : Use Scalable infra to avoid over-provisioning

Communication

- US06 : API well designed and documented to increase reuse rate

APITHINKING

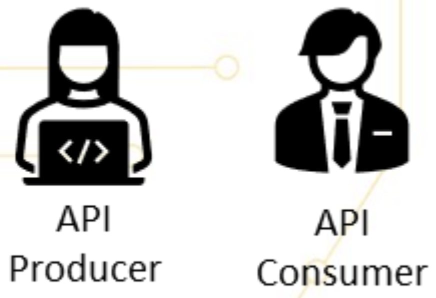
Architecture

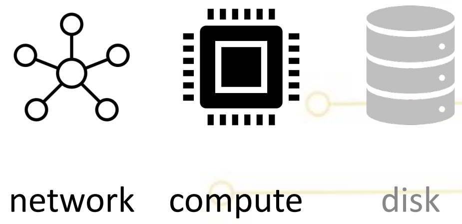
AR01 : Use Event Driven Architecture to avoid polling madness.

We often notice that applications, in order to refresh their data, make very frequent requests to APIs.
This causes an important workload and we increase the computing resources to absorb this load in order not to penalize the other users.

The best practice is to use an event-driven architecture in order to receive a notification when a piece of information is modified to avoid making regular useless requests. But the data contained in the event must be precise to be sure to avoid a system making a request to retrieve an unused data.

Expected gain: Compute resources saved & Network impact reduced





AR02 : Deploy the API near the consumer.

It is not uncommon to find that APIs are deployed in locations that are not selected in relation to their consumers.

This results in not only a degraded user experience in some cases, but also a greater demand on the network to route requests sometimes to a region on the other side of the world.

Good architecture practices therefore recommend deploying APIs, and services in general, as close as possible to the consumers. Also, if possible, prefer a deployment in several locations with geo routing (aka. position based routing) to the closest instance to improve response times and reduce the number of kilometers traveled by requests.

Expected gain: Inter-regions network traffic reduced

API Producer   API Consumer

network   compute   disk

AR03 : Ensure only one API fits the same need.

It is often noticed, especially in large information systems, that an API with the same purpose and objective can exist several times.

These duplicate APIs, in addition to creating confusion in the minds of users, consume additional resources instead of pooling them for a unique API.

It is recommended to use the data catalog to make sure that the API you want to develop does not already exist. If an existing API covers part of the functional scope, it may be worthwhile to contact the producer as it may be possible to plan an evolution of the existing system rather than creating a duplicate.

Expected gain: Compute resources saved

API Producer

API Consumer

network compute disk

APITHINKING

Architecture

AR04 : Use scalable infrastructure to avoid over-provisioning

Depending of your infrastructure, used scalable runtime fit to your activity

Example : Docker EE, Kubernetes as a best way to scale up or scale Down depending of season activities

Expected gain: Network, compute

API Producer

API Consumer

network compute disk

Architecture

AR05 : Carbon Footprint Dashboard

Some cloud providers produce carbon footprint dashboards. You can implement your own or adapt it based on your infrastructure to be close to your usage.

This is not a rule to evaluate API Green score, but it is important to be able to measure the impact on infrastructure

Example : evaluation of the impact of computing, network and disk divided by the number of calls of the evaluated API.

Expected gain: Network, compute

API Producer

API Consumer

network compute disk

Data Exchange

DE01 : Prefer an exchange format with the smallest size (JSON is smaller than XML).

One of the structuring questions when designing an API is the selection of the exchange format to use. If the choice is often made by technical constraints or personal affinities, the durability aspect is also to be taken into account.

Indeed, there are exchange formats that are heavier than others. For example, JSON is smaller than XML. The second format will therefore have a stronger impact on the network, the computing and the storage.

In the interest of sustainability, we recommend to use a lighter exchange format to reduce the bandwidth consumed for the requests, the compute and storage resources consumption used to process and store the payloads.

Expected gain: Network, compute and storage impact reduced

API Producer

API Consumer

network compute disk

APITHINKING

Data | Data Exchange

DE02/DE03/DE05 : Use cache to avoid useless requests and preserve compute resources.

The use of a cache has become common in computer architectures to store frequently used information on a fast storage.

In addition to improving the response time of APIs, and therefore the consumer's experience of the service, it also saves computational resources by avoiding executing the same query on the same data multiple times.

It is recommended to place a cache in front of each brick of an architecture returning data (API, database, frontend application, ...) and close to the users to preserve compute resources and improve performances of the API.

Expected gain: Compute resources saved & Network impact reduced

API Producer

API Consumer

network

compute

disk

APITHINKING

Data Exchange

DE04 : Prefer opaque token prior to JWT

One of the structuring questions when designing an API is the selection of the token type to use. If the choice is often made by technical constraints or personal affinities, the durability aspect is also to be taken into account.

We can note that an opaque token, in addition to improve the security, is smaller than a JWT token which will have a stronger impact on the network, storage and compute resources.
In the interest of sustainability, it is therefore recommended that a lighter token type be preferred in order to reduce the bandwidth, compute and storage resources consumption.

Expected gain: Network, compute and storage impact reduced

API Producer

API Consumer

network compute disk

APITHINKING

Data

DE06 : Allow a part cache refresh and align it on data refresh.

When configuring a cache, it often happens that the data refresh policy (TTL) is not synchronized with the data life cycle.

In this case, the cache is not fully efficient because the data is expired too early or too late.

It is necessary to provide an expiration policy adapted to the data refresh cycle and to allow partial expiration of the cached data in order to be as efficient as possible on all the processed data. To optimize the cache as much as possible, it is also possible to build an architecture where the source of the data notifies, via an event, the cache of the expiration of a specific data.

Expected gain: Volume of data stored reduced & Network impact reduced

API Producer

API Consumer

network compute disk

Data Exchange

DE07 : Construct your API with customer centricity principles.

Sometimes the data returned by an API is structured in such a way that, in order to have all the data the user needs, it is necessary to make several requests to the same API.

This has the consequence of increasing the consumption of bandwidth and computing resources, for the API that has to process several requests, and of bandwidth.

Therefore, it is important to provide a consistent data structure regarding the use of the API. This client-centric best practice prevents the consumer from having to perform multiple queries to retrieve all the information they need.

Expected gain: Compute resources saved & Network impact reduced

API Producer

API Consumer

network compute disk

APITHINKING

Data Exchange

DE08 : Implement filters to limit which fields are returned by the API (send just the data the consumer need).

It often happens that the implementation of filters in the APIs allowing to return only the necessary data to the consumers are forgotten or not efficient.

This forces API consumers to make generic requests that retrieve unnecessary amounts of information, resulting in overconsumption of bandwidth and storage.

It is recommended to design and implement filters that allow the user to limit the amount of data returned to optimize network and storage consumption.

Expected gain: Volume of data stored and network impact reduced & Compute resources saved

API Producer

API Consumer

network compute disk

APITHINKING

Data

DE09 : Leverage OData or GraphQL for your databases APIs

It is quite common to see API backends built to allow database integration. In some cases, these systems are completely redeveloped with data schemas that are not adapted to the usage.

This forces users to perform several queries, often complex, to retrieve all the data they need.

To build an interface to a database, it is recommended to rely on OData or GraphQL technologies that allow consumers to perform complex queries.

Expected gain: Network, compute and storage impact reduced

API Producer

API Consumer

network compute disk

Data Exchange

DE11 : Availability of pagination

Implement pagination to limit which data are returned by the API (send just the data the consumer need) using for exemple "next", "skip", "top", …

Check payload log to validate if pagination keywords are used

Expected gain: Volume of data stored and network impact reduced & Compute resources saved

API Producer

API Consumer

network compute disk

APITHINKING

Data Exchange

US01 : Use query parameters for GET Methods

Optimize queries to limit the information returned to what is strictly necessary.

It is often observed that requests made on APIs are not precise enough, which returns a volume of information greater than necessary.

This results in increased bandwidth consumption during exchanges.

The best practice is to create precise requests that return, as much as possible, the strictly necessary information, thus avoiding the transfer of useless information.

This rule is linked to DE08 : "Implement filters to limit which fields are returned by the API "

Expected gain: Network, compute

API Producer

API Consumer

network compute disk

APITHINKING

API Lifecycle

US02 : Decommission end of life or unused APIs.

It often happens that the APIs of an information system are rarely or no longer used but are not decommissioned.

This leads to the consumption of computing resources for useless or obsolete components.
It is important that the decommissioning phase is also treated as part of the application life cycle in order to free up allocated resources. In the case of a rarely used API, a root cause analysis should be performed prior to decommissioning to understand why it is not used more often.

Expected gain: Compute resources saved

API Producer

API Consumer

network compute disk

API Lifecycle

US03 : Number of API <=2

Have a good lifecycle management of API by reducing the number of API version on production
The value of 2 release can be challenge depending of your context.
Less version permit to have less technical debt.

Expected gain: compute and storage impact reduced

API Producer

API Consumer

network

compute

disk

Data Exchange

US04 :Usage of pagination of results available

Some request can return a huge volume of data. We can optimize the response by using pagination.

A control can be used to check some keywords like next, skip, top, etc …

This rule is linked to DE11 : “Availability of pagination”

Expected gain: Network, compute and storage impact reduced

API Producer

API Consumer

network compute disk

APITHINKING

Data

API Lifecycle

US05 : Choose the correct API based on use case to avoid requests on multiple systems or large number of requests. Refer to the data catalog to validate the data source.

In large information systems, it is common for several APIs to partially meet a need and it is necessary to call on several of them to retrieve all the information needed.

It is then noted that the number of requests for a need increases rather quickly and that the flow of transferred data is rather important.

It is recommended to use the data catalog to identify the API that best meets the needs in order to ensure the optimization of the volume of requests and data and thus avoid excessive consumption of resources.

Expected gain: Compute resources saved & Network impact reduced

API Producer

API Consumer

network compute disk

API Lifecycle

US06 : Well designed and documented API to increase reuse rate

Deploy a well designed and documented API to increase the reuse rate and improve time to market.

Based on documentation provided in the API Portal.

The more accurate the documentation, the easier it will be for consumers to understand and use the API.

This indicator is a percentage rate.

This is a sample rate calculation = (Number of consumers *50) - 50, if you have more than 100%, it will be a bonus.

Expected gain: Compute resources saved & Network impact reduced

API Producer

API Consumer

network compute disk

API Lifecycle

US07 : Error Rate

Decrease the error rate (results different from 2xx) to avoid over processing.

Depending of your context, you can focus on 4xx or 5xx errors, or both.

One of objectives of this rule is to improve the quality of requests (fill all required fields, or better control of contract,etc…) and improved the response if we have to many errors due to tech

This is rule is a rate.

Expected gain: Compute resources saved & Network impact reduced

API Producer

API Consumer

network compute disk

Tools

LO01 : Collect only required data and use the right retention time according to the business requirements.

It is quite common for applications to store a large amount of useless information without time limit.
This results in an excessive consumption of storage services for data that will not be used or no longer used.

It is necessary to clean up the data in order to keep only the data that is useful and to define a coherent retention policy in order to delete them once their validity or exploitation period has passed.

Expected gain: Volume of data stored reduced & Network impact reduced

API Producer

API Consumer

network compute disk

APITHINKING

This document is the 1st release, we need you to improve it, test it and share it !

API GREEN SCORE

https://www.collectif-api-thinking.com