API Big picture: Architecture: From SOA to WOA.

SOA* SERVICE ORIENTED ARCHITECTURE

- Web Services: Inefficient & slow
  - SOAP: heavyweight
  -很难维持一致的沟通, especially when integration is layer

- SOA architecture is mostly Servers

- App Clients are mostly servers

- WOA (Web Oriented Architecture) is a subset of SOA based on RESTful microservices and tends to correct mistakes from past implementations.

- WOA (Web Oriented Architecture) was designed by vendors and tools first: ESB, BPM, BAM...

- It has no ownership of business referential.

- It is heavyweight protocol (WS*).

- SOA results in a huge number of services with slight variations depending on the environment.

- Components are provided by vendor & tools, not for business: ESB, BPM, BAM...

- Often thinking by vendor & tools first... which has no ownership of business referential.

= POTENTIALLY can end up re-inventing the wheel so that developers can quickly use APIs, which offer good affordance.

**API Big picture: Architecture: From SOA to WOA.**

- REST architecture is mostly servers

- Often thinking by vendor & tools first...

- WOA is a subset of SOA, based on RESTful services and tends to correct mistakes from past implementations.

- SOAP: heavyweight specific protocols

- **API levels**: The API level you are targeting can be reflected by the type of consumers you are addressing:

  - Level 1 « Internal API »
    - Agnostic of consumers
    - Independent resources
    - Services based
    - 360° Customer View
    - Monitoring, Throttling
    - APIs designed for internal use (API Gateway)

  - Level 2 « Partner API »
    - Agnostic of consumers
    - Independent resources
    - Services based
    - 360° Customer View
    - Monitoring, Throttling
    - APIs designed for partner consumption (API Gateway)

  - Level 3 « Open API »
    - Agnostic of consumers
    - Independent resources
    - Services based
    - 360° Customer View
    - Monitoring, Throttling
    - APIs available to everyone (API Gateway)

- API architecture strategy.

  - The API level you are targeting can be reflected by the type of consumers you are addressing.

  - The main difference lies in the way you need to "architecture" the enactment protocols and the quality required for your API.

  - **API Strategy**: Think API First. To facilitate and accelerate design and development of your APIs, we share our vision and beliefs with you in the Reference Card. They come from our direct experience or API research.

- Why an API strategy?

  - "Anytime, Anywhere, Any device" are the key problems of digitalisation. API is the answer to "Business Agility" as it allows you to rapidly build new GUI for upcoming devices.

  - An API layer enables
    - Cross device
    - Cross channel
    - 360° customer view

  - The API layer is essential for achieving the business agility.

  - WOA (Web Oriented Architecture) allows you to rapidly build new GUI for upcoming devices.

  - Embrace WOA "Web Oriented Architecture" to:
    - Build a fast, scalable & secure REST API
    - Based on REST, HATEOAS, Stateless, decoupled protocols, Asynchronous, RESTful, REST, OPEN Connect protocols
    - Leverage the power of your existing Web infrastructure

- Why an API strategy?

  - "Anytime, Anywhere, Any device" are the key problems of digitalisation. API is the answer to "Business Agility" as it allows you to rapidly build new GUI for upcoming devices.

  - An API layer enables
    - Cross device
    - Cross channel
    - 360° customer view

  - The API layer is essential for achieving the business agility.

  - WOA (Web Oriented Architecture) allows you to rapidly build new GUI for upcoming devices.

  - Embrace WOA "Web Oriented Architecture" to:
    - Build a fast, scalable & secure REST API
    - Based on REST, HATEOAS, Stateless, decoupled protocols, Asynchronous, RESTful, REST, OPEN Connect protocols
    - Leverage the power of your existing Web infrastructure

DISCLAIMER

This Reference Card does not claim to be 100% accurate. The design concepts shown here are a result of our previous work in the REST area. Please check out our blog on HTTP and API, and feel free to comment or challenge the ideas in our cookbook. We're really looking forward to talking with you.
**API architecture stakess.**

**Stateless**
- You should build stateless resources problems. A stateful API is easier to reason about, scalable and secure.
- Stateless means there is no state.
- There are no side effects or state changes.
- Interfaces are stateless and do not require a session.
- Each request is self-contained.

**Asynchronous**
- A well-designed asynchronous API will provide an acceptable response time.
- An acceptable response time could be a few seconds.

1. On a functional level:
   - You should limit function execution flow if response times exceed the time you consider acceptable. You should use a service mesh to help with API latency and reduce response time.
   - Each API request should have a 300ms response time to the client.

2. On a technical level:
   - Request sent to the server should always be asynchronous. `API_KEY` is required for API calls to the server.
   - The resource provider is responsible for the response and not the server.

3. Business & methodological states are:
   - Short-TIM is a recommendation.
   - Complex systems should use a combination of level 2 and 3.
   - Hierarchical and messages API should be used for 1 and 2.

**Microservices**
- Microservices is a key feature of Web-Oriented Architecture.
- At some point, you will consider refactoring your CORE with small, independent units.
- For instance, you can think about your API as a microservices architecture.
- You should consider a stateless and independent service to build your API.

**API team organization.**

Your API team will impact your teams and organization.

**API MVP**
- You should consider a small autonomous and empowered team to build your API. Let's call it the API squad.

**API SQUAD STAGE ONE**

**API MVP**
- When you have your API defined, then you should consider a small team of individuals as well as functions that need to be included.
- Keep it simple: 4-5 developers.
- A developer, a product owner, a designer.
- Depending on your needs, you can add a QA tester and/or have a developer in charge of security.
- You should consider a small team that would include the following functions.

**API SQUAD STAGE TWO**

**SCALING API**
- When the MVP is validated, you should expand your API team as needed to support the demand.
- Plan on adding 0-1 developers per week.
- Be sure to have a developer on the team who can increase the team size as needed.
- You should consider a small team that would include the following functions.

**API management.**

API management solutions generally offer the following features:

1. **API MANAGEMENT Portal**
   - API Management provides a single point of access for API management.
   - API management allows you to control access to your API.
   - API management provides a centralized view of all your APIs.

2. **Developer Portal**
   - Developer Portals provide a self-service experience for developers.
   - API management provides a centralized view of all your APIs.
   - API management provides a centralized view of all your APIs.

3. **Security**
   - Security concerns should be addressed in early API development.
   - Security concerns should be addressed in early API development.
   - Security concerns should be addressed in early API development.

4. **API Facade**
   - A façade is used to provide a consistent view of your API.
   - A façade is used to provide a consistent view of your API.
   - A façade is used to provide a consistent view of your API.

**API integration patterns.**

**API security.**

HTTP/HTTPS

**Authentication**
- API keys should be used to authenticate client requests.
- OAuth 2.0 and OpenID Connect are widely used to implement API authentication.
- API keys should be used to authenticate client requests.

**Authorization**
- API keys should be used to authorize client requests.
- OAuth 2.0 and OpenID Connect are widely used to implement API authorization.
- API keys should be used to authorize client requests.

**Infrastructures**
- You can consider Cloud hosting over API infrastructure.
- You should be as close as possible to the data center to provide an optimal response time.

**API patterns.**

**WHICH PATTERN SHOULD I USE?**
- In most cases, a “façade” pattern is actually an anti-pattern, that is, you should avoid using it with APIs.
- You should consider façade as a best practice solution.
- You should avoid using a façade pattern with APIs.

**Build vs Buy.**

You should distinguish between building your API from managing your API.

**API - Build**
- You can’t consider building an API from the ground up.
- API - Build

**API - Buy**
- You can’t consider building an API from the ground up.
- API - Buy

**API strategies.**
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.

**API strategies.**
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.

**API management**

**Common pitfalls**
- An API Management tool is not a Golden Ticket.
- API Management tools are not a cure-all for your API problems.
- API Management tools are not a cure-all for your API problems.

**Build vs. Buy**

You should distinguish between building your API from managing your API.

**API - Build**
- API - Build

**API - Buy**
- API - Buy

**API integration patterns.**

**Building integration patterns.**

**Common pitfalls**
- An API Management tool is not a Golden Ticket.
- API Management tools are not a cure-all for your API problems.
- API Management tools are not a cure-all for your API problems.

**API patterns.**

**WHICH PATTERN SHOULD I USE?**
- In most cases, a “façade” pattern is actually an anti-pattern, that is, you should avoid using it with APIs.

**API strategies.**
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.

**API security.**

**HTTP/HTTPS**

**Authentication**
- API keys should be used to authenticate client requests.
- OAuth 2.0 and OpenID Connect are widely used to implement API authentication.
- API keys should be used to authenticate client requests.

**Authorization**
- API keys should be used to authorize client requests.
- OAuth 2.0 and OpenID Connect are widely used to implement API authorization.
- API keys should be used to authorize client requests.

**API strategies.**
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.
- API strategies are often focused on “buying” an API management solution.

**API patterns.**

**WHICH PATTERN SHOULD I USE?**
- In most cases, a “façade” pattern is actually an anti-pattern, that is, you should avoid using it with APIs.