Elastic Search Cheat Sheet

App Process

The **app process** is a combination of servers, APIs, operating systems, and databases that come together to power an app. What the end-user sees and with which he interacts has been established by coding interaction between the framework and the API.

```
Visit a library to read a book.  You are the app.  The
shelves, carts of books, and the building containing
all this is the framework.  The search for a book and
the seat you take to read it is the API.
```

Platform

A **platform** is simply the hardware or software for which the piece of software is built. For example, software may be built for a Windows, OS X, Android, iOS, XBOX One, PS4, etc. These are all platforms. Since different platforms have different requirements and interface differently to the software, the code you write may not run on all platforms and it is important to know which platforms you are building for.

Library

A **Library** refers to code that provides functions that you can call from your own code to deal with common tasks. For example a math library will provide you with common mathematical functionality such as trigonometric or logarithmic functions. Programming languages usually have libraries for all sorts of tasks such as data processing, plotting of graphs, text parsing, etc. Once included, libraries save you the trouble of writing all those functions yourself.

API

Short for *Application Programming Interface.* This term refers to the "face" of the library, as it is accessible to the programmer. Think of it as a logical representation of what is in the library, and the relevant documentation that explains what the programmer can do with the library. The difference being that library refers to the code itself, whereas API refers to the interface. There are many interesting APIs, some of the exciting ones are provided by websites we use daily such as Google APIs, Facebook Messenger API, etc.

IDE

Short for *Integrated Development Environment.* The IDE is an application which helps you during the process of writing the code itself by automating many useful processes such as debugging, refactoring, code generation, etc. An IDE is just a tool to help programmers, and you may simply use Notepad if you wish. Examples of IDEs include: Eclipse, IntelliJ IDEA, Netbeans, Visual Studio, etc.

SDK

Short for Software Development Kit. This is a complete kit of software development tools for a specific platform. This "kit" can include all sorts of things such as: Libraries, APIs, IDEs, Documentation, etc. For example the Android SDK, which provides everything you may need for Android development.

Toolkit

Seems to me a loose term to refer to any collection of "tools" (another loose term) that have a common goal.

Framework

A *Framework* is a generic structure that provides a skeleton architecture with which specific software can be implemented. The abstraction allows for common design patterns to be easily reused while still allowing the specific details to be left to the developers. Reusing common design patterns means having the general structure for solving similar sorts of problems. For example the Java Swing Framework provides the functionality and structure for Java GUI programming; it can be used for whatever GUI programming you may need to do. Another example is the Model-View-Controller Framework that describes in very abstract terms the three main parts of a common web application. The framework may be manifested as functions and classes that necessarily need to be implemented such as the run() method in Java Swing, requiring the user to conform with the design pattern that the framework is all about.

World Wide Web

HTTP

HTTP stands for Hypertext Transfer Protocol and is used to structure requests and responses over the internet. HTTP requires data to be transferred from one point to another over the network.

REST

Short for *Representional State Trasnfer*, REST is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server

Separation of Client and Server

In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other. This means that the code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client.

As long as each side knows what format of messages to send to the other, they can be kept modular and separate.

Stateleness

Systems that follow the REST paradigm are stateless, meaning that the server does not need to know anything about what state the client is in and vice versa. In this way, both the server and the client can understand any message received, even without seeing previous messages. This constraint of statelessness is enforced through the use of resources, rather than commands. Resources are the nouns of the Web - they describe any object, document, or thing that you may need to store or send to other services.

Communication between Client and Server

In the REST architecture, clients send requests to retrieve or modify resources, and servers send responses to these requests. Let's take a look at the standard ways to make requests and send responses.

Making Requests

- GET - retrieve a specific ressource (by id) or a collection of ressources
- POST - create a new resource
- PUT - update a specific ressource (by id)
- DELETE - remove a specific ressource by id

Headers and Accept parameters

In the header of the request, the client sends the type of content that it is able to receive from the server. This is called the Accept field, and it ensures that the server does not send data that cannot be understood or processed by the client. The options for types of content are MIME Types. MIME Types, used to specify the content types in the Accept field, consist of a type and a subtype. They are separated by a slash (/). For example, a client accessing a resource with id 23 in an articles resource on a server might send a GET request like this:

```
GET /articles/23
Accept: text/html, application/xhtml
```

Paths

Requests must contain a path to a resource that the operation should be performed on. In RESTful APIs, paths should be designed to help the client know what is going on. Conventionally, the first part of the path should be the plural form of the resource. This keeps nested paths simple to read and easy to understand.

Sending Responses

Content Types

In cases where the server is sending a data payload to the client, the server must include a content-type in the header of the response. This content-type header field alerts the client to the type of data it is sending in the response body. These content types are MIME Types, just as they are in the accept field of the request header. The content-type that the server sends back in the response should be one of the options that the client specified in the accept field of the request. For example, when a client is accessing a resource with id 23 in an articles resource with this GET Request:

```
GET /articles/23 HTTP/1.1
Accept: text/html, application/xhtml
```

The server might send back the content with the response header:

```
HTTP/1.1 200 (OK)
Content-Type: text/html
```

Response Code

1. 200 - OK (GET-PUT)
2. 201 - CREATED (POST)
3. 204 - NO CONTENT (DELETE)
4. 400 - BAD REQUEST
5. 403 - FORBIDDEN
6. 404 - NOT FOUND
7. 500 - INTERNEL SERVER ERROR

Fast API

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

Swagger

Swagger is an Interface Description Language for describing RESTful APIs expressed using JSON.

Interface Description Language

An interface description language or interface definition language (IDL), is a generic term for a language that lets a program or object written in one language communicate with another program written in an unknown language. IDLs describe an interface in a language-independent way, enabling communication between software components that do not share one language, for example, between those written in C++ and those written in Java.

The Web Server Gateway Interface

The Web Server Gateway Interface is a simple calling convention for web servers to forward requests to web applications or frameworks written in the Python programming language. a Calling convention is an implementation-level (low-level) scheme for how subroutines receive parameters from their caller and how they return a result.

ASGI

ASGI is a spiritual successor to WSGI, the long-standing Python standard for compatibility between web servers, frameworks, and applications.

What is wrong with WSGI?

WSGI applications are a single, synchronous callable that takes a request and returns a response; this doesn't allow for long-lived connections, like you get with long-poll HTTP or WebSocket connections. WSGI applications are a single, synchronous callable that takes a request and returns a response; this doesn't allow for long-lived connections, like you get with long-poll HTTP or WebSocket connections. ASGI is structured as a single, asynchronous callable. It takes a scope, which is a dict containing details about the specific connection, send, an asynchronous callable, that lets the application send event messages to the client, and receive, an asynchronous callable which lets the application receive event messages from the client.

```
SYNCHRONE

| -------- A -------- |
                       |-------- B -------- |
ASYNCHRONE

| -------- A -------- |
      | -------- B -------- |
```

Microservices

Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

The microservice architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.