JSON

L'estàndard JSON (definit en el document RFC 4527, http://www.ietf.org/rfc/rfc4627.txt) és un conjunt de regles que defineixen com s'ha d'estructurar la informació de tipus textual perquè aquesta pugui ser emmagatzemada (normalment en fitxers amb extensió ".json") i/o intercanviada entre diferents sistemes d'una manera lleugera, llegible i independent del llenguatge o plataforma utilitzada. En aquest sentit, el seu objectiu és comparable al d'un altre estàndard anomenat XML (també molt comú), però la sintaxi de JSON és molt més senzilla de codificar, més fàcil de llegir i més ràpida d'interpretar (per a les màquines i per als humans) .

> NOTA: Els usos més comuns de JSON és, d'una banda, permetre la descàrrega des d'un servidor remot (normalment de tipus HTTP/S) d'una cadena contenint les dades ja estructurats correctament en JSON per a la seva fàcil processament per part de client i, de l'altra, permetre l'enviament des del client al servidor (normalment a través de mètodes HTTP, bé via GET o via POST) de cadenes escrites en JSON perquè el servidor les pugui interpretar adequadament.

JSON obliga a estructurar d'una determinada manera les dades abans d'emmagatzemar-les i/o enviar-les. Aquesta estructura pot venir en dues formes: bé especificant entre claudàtors una llista de valors (separats entre si per comes) o bé especificant entre claus una llista de parelles clau-valor (separats entre si també per comes). Un cop la informació està organitzada segons un d'aquests dos patrons possibles (els quals es poden niar un dins d'un altre per crear estructures més complexes), ja pot ser emmagatzemada o tramesa en forma de cadena de caràcters.

Per exemple, suposem que obtenim d'un servidor web certa informació sobre una persona, Concretament el seu nom, la seva edat i el seu color de pèl. Per poder reconèixer aquestes dades com JSON vàlid, haurien de seguir una disposició similar a aquesta: *{ "nom": "Pepe", "edat": 22, "colorPel": "negre"}*. També podrien representar-se de forma més simple, així: *["Pepe", 22, "negre"]* però en aquest cas perdríem semàntica.

Una altra cosa que també podríem fer és usar un "array" per contenir les dades de diverses persones, així (els salts de línia són opcionals: siguin s'han afegit per augmentar la claredat de l'exemple):

```
[       { "nom": "Pepe", "edat": 22, "colorPel": "negre" },
        { "nom": "Pepa", "edat": 35, "colorPel": "bru" },
        { "nom": "Pepi", "edat": 17, "colorPel": "ros" }                ]
```

O fins i tot convertir aquest "array" en el valor d'una determinada clau:

```
{       "persones": [
                { "nom": "Pepe", "edat": 22, "colorPel": "negre" },
                { "nom": "Pepa", "edat": 35, "colorPel": "bru" },
                { "nom": "Pepi", "edat": 17, "colorPel": "ros" }
                  ]
        "nombre": 3                                                     }
```

Hi ha algunes regles que hem de tenir en compte a l'hora de treballar amb el tipus JSON:

* Les claus de les parelles clau-valor han d'estar escrites sempre entre cometes dobles.

* Els valors possibles només poden ser nombres (escrits sempre en format decimal), cadenes (escrites sempre entre cometes dobles), valors booleans (*true* o *false)* o el valor especial *null*, així com llistes de parells de clau-valor i llistes simples de valors.

* Tant els noms de les claus com els valors de tipus cadena poden contenir qualsevol caràcter Unicode que estigui codificat en UTF-8.

* Tant la parella clau-valor buida ({}) com la llista de valors buida ([]) són perfectament vàlides.

* Si tenim un únic valor, cal escriure'l entre claudàtors (o bé, entre claus si ho vinculem a una clau)

Hi ha diversos serveis en línia que permeten comprovar la correcta sintaxi de les dades JSON introduïdes. Aquestes eines ens poden ser d'ajuda en moments puntuals per detectar possibles fallades difícils de veure a simple vista. Alguns exemples són http://www.freeformatter.com/json-validator.html , http://jsonlint.com , http://jsonviewer.stack.hu , http://jsoneditoronline.org , http://www.jsonviewer.com , http://json.parser.online.fr... però n'hi ha molts més. També és interessant la web https://onlinejsontools.com , la qual disposa de multitud d'eines per validar, minimitzar, transformar dades JSON a altres formats (com ara YAML, XML, CSV, una simple cadena, etc) i viceversa, etc.

ADDENDUM: Comanda *jq*

Jq (https://stedolan.github.io/jq) és una comanda de terminal que permet consultar i manipular documents JSON. Seria "l'equivalent al grep/cut/sed/awk" per processar informació amb aquest tipus de format. A continuació mostrem alguns exemples d'ús, implementats sobre un fitxer anomenat "prova.json" amb el següent contingut:

```
{
  "name": "something",
  "version": "0.0.0",
  "unarray": [1,2],
  "params": {
    "name1": "value1",
    "name2": "value2"
  }
}
```

jq "." prova.json	Mostra tot el JSON (el símbol "." representa l'arrel)
jq ".name" prova.json	Mostra el valor de la clau "name" (de primer nivell)
jq ".name, .version" prova.json	Mostra els valors de les claus "name" i "version" (de primer nivell)
jq ".unarray" prova.json	Mostra els valors de l'array "array" (de primer nivell) NOTA:També es pot escriure *jq ".unarray[]" prova.json*. La sortida és diferent
jq ".unarray[0]" prova.json	Mostra el valor del primer element de l'array "array" NOTA:També es poden escriure rangs; per exemple, el filtre *".unarray[1:4]"* seleccionaria des de l'element 1 (inclós) fins el 4 (sense incloure)
jq ".params" prova.json	Mostra els valors de l'objecte "params" (de primer nivell)
jq ".params.name1" prova.json	Mostra el valor de l'element "name1" de l'objecte "params"
jq ".params | length " prova.json	Mostra la quantitat d'elements de l'objecte (o array) indicat
jq ". | keys " prova.json	Mostra els noms de les claus presents a tot el JSON
jq ".[] | .name" prova.json o directament *jq ".[].name" prova.json*	Mostra els valors de totes les claus ".name" del JSON (si l'arrel d'aquest fos un array) NOTA: El filtre pot ser, a més d'una clau estàndar, un array o un objecte JSON intern: *jq ".[] | .unarray" prova.json* o *jq ".[] | .params" prova.json*
jq ". | {uncamp:.version, altre:.unarray}" prova.json	Crea un objecte JSON nou format per elements concrets de l'objecte original. En aquest exemple, concretament crea un objecte de dos elements, un anomenat "uncamp" amb el valor que tenia l'element "version" de l'objecte original i un altre anomenat "altre" amb el valor que tenia l'element "unarray" de l'original NOTA:També es poden escollir elements concrets d'un array, així: *jq ". | {uncamp:.version, altre:.unarray[1]}" prova.json* NOTA: Un exemple pràctic: *lshw -c memory -json | jq ". | {MEMORIA: .size}"*
jq "[.name, .unarray[]]" prova.json	Crea un array a partir dels valors individuals del elements indicat de l'objecte original
jq '.| select(.name == "anything")' prova.json	Mostra tots els objectes JSON complets (en l'exemple només tenim un) que tinguin el seu element "name" amb el valor "anything" (en l'exemple no en tenim cap) NOTA: Altres filtres possibles són "contains" o "endswith" (entre altres!), que es poden emprar així: *jq '.| select(.name | contains("anything"))' prova.json* i així *jq '.| select(.name | endswith("anything"))' prova.json* , respectivament
jq '.| select(.name == "anything") | .version' prova.json	Mostra només el valor de l'element "version" de tots els objectes filtrats pel valor del seu camp ".name".

jq ".params.name3 = [5,6]" prova.json	Afegeix un nou element anomenat "name3" al subobjecte "params" de l'objecte JSON original. Aquest nou element en aquest cas és un array amb dos valors (5 i 6).
jq ".unarray |= .+[3]" prova.json	Afegeix un nou element a l'array. NOTA: |= reassigna el valor i .+ [...] afegeix un nou ítem
jq '.version = "1.0.0"' prova.json	Canvia el valor de l'element simple indicat per l'especificat
jq ".name = .params.name1" prova.json	Canvia el valor de l'element simple indicat pel valor d'un altre element de l'objecte
jq ".unarray[0] + 1" prova.json	Realitza un operació aritmètica bàsica amb el valor de l'element indicat (+,-,*,/,%)
jq '.version == "1.0.0"' prova.json	Comprova si l'element simple indicat té el valor especificat (retorna *true* o *false*)
jq "del(.params,.unarray)" prova.json	Elimina els elements indicats

NOTA: Altres funcions interessants de *jq* són "*.nomArray | min*" (retorna l'element de l'array amb valor mínim), "*.nomArray | max*" (retorna l'element de l'array amb valor màxim), "*.nomArray | unique*" (retorna un array amb els elements sense repetir), "*.nomArray | sort*" (retorna un array ordenat a partir d'un array original desordenat), "*.nomArray | sort[]*" (retorna una llista de valors ordenats a partir d'un array desordenat), etc.

NOTA: La comanda *jq* disposa de varis paràmetres interessants, com ara *-c* (per mostrar la sortida sense embellir) o *-r* (per mostrar els valors dels camps extrets sense cometes, és a dir tal qual són), entre d'altres.

NOTA: En https://jqplay.org es poden provar de forma interactiva tots els filtres *jq* que es vulguin i observar online el seu resultat. Molt útil per aprendre aquesta eina. D'altra banda, també existeix l'eina *jiq* (https://github.com/fiatjaf/jiq), la qual permet provar interactivament els filtres *jq* i veure en temps real el seu resultat.

NOTA: Es pot combinar *jq* amb la sortida JSON de *journalctl* per tal de construir filtres que el propi *journalctl* no incorpora. Per exemple, així: *journalctl -o json |jq ' select((has("_SYSTEMD_UNIT") and (._SYSTEMD_UNIT \| in({"noisy.service": "", "chatty.service": ""})) or (._TRANSPORT == "kernel" and (.MESSAGE \|startswith("[PREFIX]")))) \| not) ' \|less*

NOTA: Existeixen diverses especificacions estàndard molt complexes (i completes) que defineixen com consultar i manipular informació en format JSON d'una manera que va més enllà de l'eina *jq* concreta Exemple són http://jsoniq.org o https://jmespath.org (la qual ve acompanyada de l'eina de terminal *jp*, https://github.com/jmespath/jp, alternativa a *jq*). D'altra banda, també existeixen especificacions que el que fan és "ampliar" el JSON estàndard en incorporar elements extra com ara variables, condicionals, bucles, funcions, etc els quals permeten escriure documents pseudoJSON d'una forma molt més compacta i breu; exemple són https://jsonnet.org o https://dhall-lang.org, entre altres. D'altra banda, també existeix l'eina *jello* (https://github.com/kellyjonbrazil/jello), que pretén fer el mateix que *jq* però emprant pels filtres sintaxi Python