JSON (JavaScript Object Notation):

JSON is a syntax for storing and exchanging data. JSON is text, written with JavaScript object notation. Python has a built-in package called JSON, which can be used to work with JSON data. JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. JavaScript Object Notation (JSON) is a human readable and very popular format used by web services, programming languages (including Python) and APIs to read/write data. Import the json module: Import JSON. When exchanging data between a browser and a server, the data can only be text. JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server. JSON is a lightweight data-interchange format. Both JSON and XML can be used to receive data from a web server. JSON is a human readable data format used by applications for storing, transferring and reading data. JSON is a very popular format used by web services and APIs to provide public data. This is because it is easy to parse and can be used with most modern programming languages, including Python. Both JSON and XML can be used to receive data from a web server. XML is much more difficult to parse than JSON. JSON is parsed into a ready-to-use JavaScript object. JSON is light-weight as compare to XML. It is language independent. Easy to read and write. Text based, human readable data exchange format. When we query a device through a script or software application, we can receive that data in JSON format. The data being transferred can be used for machine-to-machine interaction. It's human-readable as well. While we may be used to CLI output, as network engineers, it may not be very useful for machines. Therefore, we have a data structure being used such as JSON. Using the Meraki sandbox as an example, we make a request and we receive a response in JSON format.

JSON Syntax Structure:

- Uses curly braces {} to hold objects and square brackets [] to hold arrays.
- JSON data is written as key/value pairs.
- A key/value pair consists of a key (must be a string in double quotation marks ""), followed by a colon :, followed by a value. For example: "name":"John"
- Each key must be unique.
- Values must be of type string, number, object, array, Boolean or null.
- Multiple key/value within an object are separated by commas ,

JSON Data:

In JSON, the data known as an object is one or more key/value pairs enclosed in braces { }. JSON data is written as name/value pairs, just like JavaScript object properties. A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value: JSON names require double quotes. JavaScript names do not.

"firstName":"Ali"

JSON Objects:

JSON objects are written inside curly braces. Just like in JavaScript, objects can contain multiple name/value pairs: Always surrounded by curly brackets. Composed of one-or-more name-value

{"firstName":"Ali", "lastName":"Khan"}

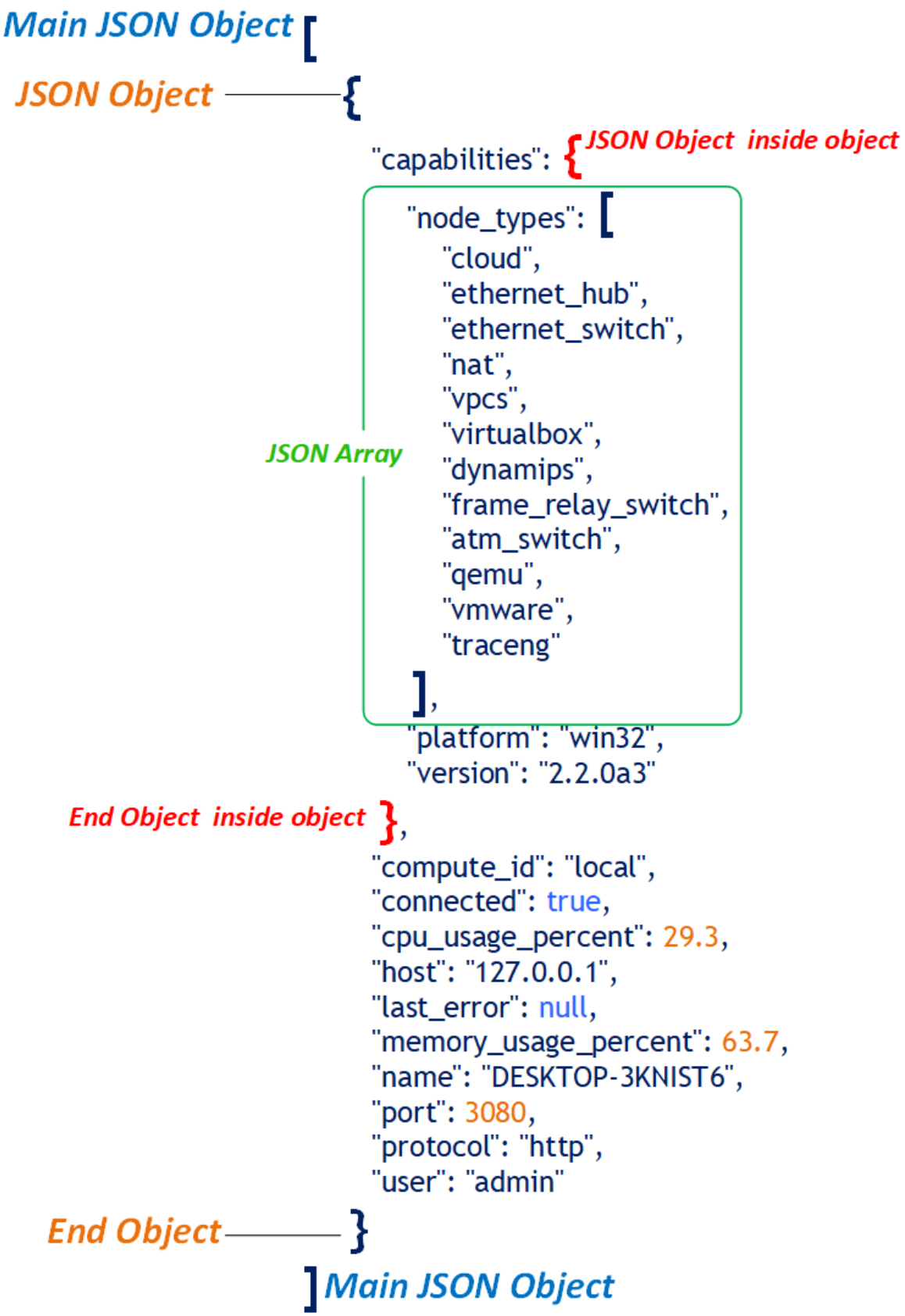
JSON Arrays:

JSON can use arrays. Arrays are used to store multiple values in a single variable. JSON arrays are written inside square brackets. Just like in JavaScript, an array can contain objects: the object "employees" is an array. It contains three objects. Each object is a record of a person (with a first name and a last name).

```
"employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
]
```

JSON List of IPv4 Addresses:

```
{
 "addresses": [
  {
   "ip": "172.16.0.2",
   "netmask": "255.255.255.0"
  },
  {
   "ip": "172.16.0.3",
   "netmask": "255.255.255.0"
  },
  {
   "ip": "172.16.0.4",
   "netmask": "255.255.255.0"
  } ] }
```

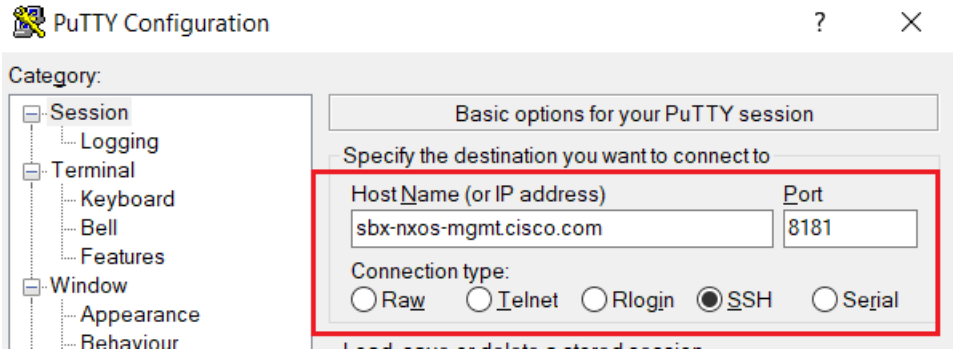


Lab Time:

S1 Configuration
S1#terminal length 0
S1#show running-config | format

When run this command: show running-config | format it will show the running configuration in XML format copy the data go to any XML to JSON convertor in google such as https://codebeautify.org/xmltojson

Use the DevNet Nexus always on Sandbox:
DNS name: sbx-nxos-mgmt.cisco.com
Protocol: SSH
Port: 8181
Username: admin
Password: Admin_1234!
Example:
Windows: Use Putty to connect



```
sbx-ao# show ip int brief | json
{"TABLE_intf": ["ROW_intf": [{"vrf-name-out": "default", "intf-name": "Vlan100", "proto-state": "do
p-disabled": "FALSE"}, {"vrf-name-out": "default", "intf-name": "Vlan101", "proto-state": "down", "
bled": "FALSE"}, {"vrf-name-out": "default", "intf-name": "Vlan102", "proto-state": "down", "link-s
 "FALSE"}, {"vrf-name-out": "default", "intf-name": "Vlan103", "proto-state": "down", "link-state":
E"}, {"vrf-name-out": "default", "intf-name": "Vlan104", "proto-state": "down", "link-state": "down
"vrf-name-out": "default", "intf-name": "Vlan105", "proto-state": "down", "link-state": "down", "ad
ame-out": "default", "intf-name": "Lo1", "proto-state": "up", "link-state": "up", "admin-state": "u
t", "intf-name": "Lo98", "proto-state": "up", "link-state": "up", "admin-state": "up", "iod": "15",
 "Lo99", "proto-state": "up", "link-state": "up", "admin-state": "up", "iod": "16", "prefix": "10.9
o-state": "down", "link-state": "down", "admin-state": "down", "iod": "21", "prefix": "172.16.1.1",
```

```
sbx-ao# show ip int brief | json-pretty
{
    "TABLE_intf": {
        "ROW_intf": [
            {
                "vrf-name-out": "default",
                "intf-name": "Vlan100",
                "proto-state": "down",
                "link-state": "down",
                "admin-state": "down",
                "iod": "7",
                "prefix": "172.16.100.1",
                "ip-disabled": "FALSE"
            },
```

Use Local Cisco Nexus Operating System (NX-OS) Switch in GNS3 or EVE NG.

```
switch# show access-lists | json
{"TABLE_ip_ipv6_mac": {"ROW_ip_ipv6_mac": [{"op_ip_ipv6_mac": "ip", "acl_name":
"copp-system-p-acl-auto-rp", "TABLE_seqno": {"ROW_seqno": [{"seqno": "10", "perm
itdeny": "permit", "ip": "ip", "src_any": "any", "dest_ip_prefix": "224.0.1.39/3
2"}, {"seqno": "20", "permitdeny": "permit", "ip": "ip", "src_any": "any", "dest
_ip_prefix": "224.0.1.40/32"}]}}, {"op_ip_ipv6_mac": "ip", "acl_name": "copp-sys
tem-p-acl-bgp", "TABLE_seqno": {"ROW_seqno": [{"seqno": "10", "permitdeny": "per
mit", "proto_str": "tcp", "src_any": "any", "src_port_op": "gt", "src_port1_num"
: "1023", "dest_any": "any", "dest_port_op": "eq", "dest_port1_str": "bgp", "des
t_port1_num": "179"}, {"seqno": "20", "permitdeny": "permit", "proto_str": "tcp"
, "src_any": "any", "src_port_op": "eq", "src_port1_str": "bgp", "src_port1_num"
: "179", "dest_any": "any", "dest_port_op": "gt", "dest_port1_num": "1023"}]}},
{"op_ip_ipv6_mac": "ipv6", "acl_name": "copp-system-p-acl-bgp6", "TABLE_seqno":
{"ROW_seqno": [{"seqno": "10", "permitdeny": "permit", "proto_str": "tcp", "src_
any": "any", "src_port_op": "gt", "src_port1_num": "1023", "dest_any": "any", "d
est_port_op": "eq", "dest_port1_str": "bgp", "dest_port1_num": "179"}, {"seqno":
 "20", "permitdeny": "permit", "proto_str": "tcp", "src_any": "any", "src_port_o
p": "eq", "src_port1_str": "bgp", "src_port1_num": "179", "dest_any": "any", "de
st_port_op": "gt", "dest_port1_num": "1023"}]}}, {"op_ip_ipv6_mac": "ip", "acl_n
ame": "copp-system-p-acl-dhcp", "TABLE_seqno": {"ROW_seqno": [{"seqno": "10", "p
ermitdeny": "permit", "proto_str": "udp", "src_any": "any", "src_port_op": "eq",
 "src_port1_str": "bootpc", "src_port1_num": "68", "dest_any": "any"}, {"seqno":
 "20", "permitdeny": "permit", "proto_str": "udp", "src_any": "any", "src_port_o
p": "neq", "src_port1_str": "bootps", "src_port1_num": "67", "dest_any": "any",
```

Python

```
from cli import *
test1 = 'show access-lists| json-pretty'
out1 = cli(test1)
print(out1)
```