JQ and JSON Labs

Task – 1: Create test JSON files

1. Use the following API command to create a JSON file for 250 TCP services on the R81.20 Management Server.

```
mgmt_cli -r true show objects limit 250 type "service-tcp"
details-level full --format json > tcp.json
```

```
[Expert@sms_r8120:0]# mgmt_cli -r true show objects limit 250 type "service-tcp" details-level full --format json > tcp.json
[Expert@sms_r8120:0]# ls
cp_mgmt_api_workshop-main  last_dump.log  tcp.json
```

2. Review the tcp.json file.
   a. How many elements did the API output?
   b. How many elements remain?
   c. Did we capture all the **TCP services**?


```
  "from" : 1,
  "to" : 218,
  "total" : 218,
  "objects" : [ {
    "uid" : "97aeb44f-9aea-11d5-bd16-0090272ccb30",
    "name" : "AOL",
    "type" : "service-tcp",
    "domain" : {
      "uid" : "a0bbbc99-adef-4ef8-bb6d-defdefdefdef",
      "name" : "Check Point Data",
      "domain-type" : "data domain"
```


3. Now we will create another file for a specific object. Find the UID for the gateway named gw_r8120. Use the command below to find the UID of the gateway:

```
mgmt_cli -r true show-gateways-and-servers --format json
```


```
[Expert@sms_r8120:0]# mgmt_cli -r true show-gateways-and-servers --format json
{
  "objects" : [ {
    "uid" : "0bf17d9a-a313-4af5-b380-93c45fa24361",
    "name" : "gw_r8120",
    "type" : "simple-gateway",
    "domain" : {
      "uid" : "41e821a0-3720-11e3-aa6e-0800200c9fde",
      "name" : "SMC User",
      "domain-type" : "domain"
    },
    "icon" : "NetworkObjects/gateway",
    "color" : "black"
  }, {
    "uid" : "9cbb033c-3ff3-c041-bd60-4293199ca027",
    "name" : "sms_r8120",
```


4. Create a json file for the gateway on the R81.20 Management Server

```
mgmt_cli -r true show object uid "0bf17d9a-a313-4af5-b380-93c45fa24361"
details-level full --format json > gateway.json
```

```
[Expert@sms_r8120:0]# mgmt_cli -r true show object uid "0bf17d9a-a313-4af5-b380-93c45fa24361" details-level full --format json > gateway.json
[Expert@sms_r8120:0]# more gateway.json
{
  "object" : {
    "uid" : "0bf17d9a-a313-4af5-b380-93c45fa24361",
    "name" : "gw_r8120",
    "type" : "simple-gateway",
    "domain" : {
      "uid" : "41e821a0-3720-11e3-aa6e-0800200c9fde",
      "name" : "SMC User",
      "domain-type" : "domain"
    },
    "interfaces" : [ {
      "uid" : "6e27c7a9-3676-49fe-86e5-962694001097",
      "name" : "eth0",
      "network-interface-type" : "ethernet",
      "ipv4-address" : "203.0.113.10",
      "ipv4-network-mask" : "255.255.255.0",
      "ipv4-mask-length" : 24,
      "ipv6-address" : "",
      "comments" : "",
      "color" : "black",
      "icon" : "NetworkObjects/network",
      "topology" : "automatic",
      "topology-automatic-calculation" : "external",
      "anti-spoofing" : true,
      "anti-spoofing-settings" : {
```

Task – 2: Filtering with JQ

1. Run the command on the R81 Management server

```
cat gateway.json | jq '.'
```

2. This output from `jq` is the same as running `cat gateway.json`, **except** `jq` will validate the JSON format is proper. The previous command will be used in the next steps to filter the output further.

```
[Expert@sms_r8120:0]# cat gateway.json | jq '.'
{
  "object": {
    "uid": "0bf17d9a-a313-4af5-b380-93c45fa24361",
    "name": "gw_r8120",
    "type": "simple-gateway",
    "domain": {
      "uid": "41e821a0-3720-11e3-aa6e-0800200c9fde",
      "name": "SMC User",
      "domain-type": "domain"
    },
    "interfaces": [
      {
        "uid": "6e27c7a9-3676-49fe-86e5-962694001097",
        "name": "eth0",
        "network-interface-type": "ethernet",
        "ipv4-address": "203.0.113.10",
        "ipv4-network-mask": "255.255.255.0",
        "ipv4-mask-length": 24,
        "ipv6-address": "",
```


3. Now use JQ to filter the interface details that are configured on the gateway. Run:

```
cat gateway.json | jq '.object.interfaces'
```

4. Notice the output now is filtered to just the interface data from the JSON output

```
[Expert@sms_r8120:0]# cat gateway.json | jq '.object.interfaces'
[
  {
    "uid": "6e27c7a9-3676-49fe-86e5-962694001097",
    "name": "eth0",
    "network-interface-type": "ethernet",
    "ipv4-address": "203.0.113.10",
    "ipv4-network-mask": "255.255.255.0",
    "ipv4-mask-length": 24,
    "ipv6-address": "",
    "comments": "",
    "color": "black",
    "icon": "NetworkObjects/network",
    "topology": "automatic",
    "topology-automatic-calculation": "external",
    "anti-spoofing": true,
    "anti-spoofing-settings": {
      "action": "detect",
      "exclude-packets": false,
      "spoof-tracking": "none"
    },
    "security-zone": false
  },
```

5. Now we can filter just to the interface names using this command

```
cat gateway.json | jq '.object.interfaces[].name'
```
or
```
cat gateway.json | jq '.object.interfaces[] | .name'
```

```
[Expert@sms_r8120:0]# cat gateway.json | jq '.object.interfaces[].name'
"eth0"
"eth1"
"eth2"
"eth3"
"eth4"
[Expert@sms_r8120:0]# cat gateway.json | jq '.object.interfaces[] | .name'
"eth0"
"eth1"
"eth2"
"eth3"
"eth4"
```

6. We will filter the output to only the eth1 interface now. `jq` provides the capability to just pull an element that matches a filter

```
cat gateway.json | jq '.object.interfaces[] | select(.name == "eth1")'
```



```
[Expert@sms_r8120:0]# cat gateway.json | jq '.object.interfaces[] | select(.name == "eth1")'
{
  "uid": "3789084a-3177-44c9-85ee-b655ba8804ff",
  "name": "eth1",
  "network-interface-type": "ethernet",
  "ipv4-address": "10.0.1.10",
  "ipv4-network-mask": "255.255.255.0",
  "ipv4-mask-length": 24,
  "ipv6-address": "",
  "comments": "",
  "color": "black",
  "icon": "NetworkObjects/network",
  "topology": "internal",
  "topology-settings": {
    "ip-address-behind-this-interface": "network defined by the interface ip and net mask",
    "interface-leads-to-dmz": false
  },
  "anti-spoofing": true,
  "anti-spoofing-settings": {
    "action": "detect",
    "exclude-packets": false,
    "spoof-tracking": "none"
  },
  "security-zone": false
}
```

7. Next, we will filter the output to give us the interface name and ip-address for each interface. Run this command:

```
cat gateway.json | jq '.object.interfaces[] | (.name +"," + ."ipv4-address")'
```

```
[Expert@sms_r8120:0]# cat gateway.json | jq '.object.interfaces[] | (.name +"," + ."ipv4-address")'
"eth0,203.0.113.10"
"eth1,10.0.1.10"
"eth2,10.0.2.10"
"eth3,10.0.3.10"
"eth4,10.0.4.10"
```

8. Next, we can output the data as a `csv` output using `jq`. Run the following command

```
cat gateway.json | jq -r '.object.interfaces[] | [.name, ."ipv4-address"]
|@csv'
```

```
[Expert@sms_r8120:0]# cat gateway.json | jq -r '.object.interfaces[] | [.name, ."ipv4-address"] |@csv'
"eth0","203.0.113.10"
"eth1","10.0.1.10"
"eth2","10.0.2.10"
"eth3","10.0.3.10"
"eth4","10.0.4.10"
```

9. Now that we can see how to filter and grab data elements, do the following:
   a. Filter the output to use a semi-colon as a delimiter.

```
"eth0","203.0.113.10"
"eth1","10.0.1.10"
"eth2","10.0.2.10"
"eth3","10.0.3.10"
"eth4","10.0.4.10"
```

   b. Add the network mask to the output on the ip-address output
      i. Example output 10.0.0.51/255.255.255.0

Task 3 – Filter the larger dataset

1. Now that we have shown how to filter a small dataset, we can begin by filtering a larger set of data. We can use the tcp.json file with over 200 elements to show how to filter out what we want.
2. First run the following command to make sure the dataset is proper JSON

```
cat tcp.json | jq '.'
```

3. Notice in the top of the output we still have the total counts in the JSON file.

```
[Expert@sms_r8120:0]# cat tcp.json |jq '.'
{
  "from": 1,
  "to": 218,
  "total": 218,
  "objects": [
    {
      "uid": "97aeb44f-9aea-11d5-bd16-0090272ccb30",
      "name": "AOL",
      "type": "service-tcp",
      "domain": {
        "uid": "a0bbbc99-adef-4ef8-bb6d-defdefdefdef",
        "name": "Check Point Data",
        "domain-type": "data domain"
      },
      "enable-tcp-resource": false,
      "sync-connections-on-cluster": true,
      "use-delayed-sync": false,
      "delayed-sync-value": 30,
```

4. Begin by selecting only the objects by running this command

```
cat tcp.json |jq '.objects[]'
```

5. Select only the names of the services in the file

```
cat tcp.json |jq '.objects[].name'
```

```
[Expert@sms_r8120:0]# cat tcp.json |jq '.objects[].name'
"AOL"
"AP-Defender"
"AT-Defender"
"Backage"
"BGP"
"Bionet-Setup"
"CheckPointExchangeAgent"
"Citrix_ICA"
"ConnectedOnLine"
"CP_Exnet_PK"
"CP_Exnet_resolve"
"CP_redundant"
"CP_reporting"
"CP_rtm"
"CP_seam"
"CP_SmartPortal"
```


6. Next select all of the services that have Yahoo or Napster in their name. Use the following command:

```
cat tcp.json |jq '.objects[] | select(.name | contains("Yahoo" , "Napster"))'
```

```
[Expert@sms_r8120:0]# cat tcp.json |jq '.objects[] | select(.name | contains("Yahoo" , "Napster"))'
{
  "uid": "3bb26988-e0a5-45d6-8018-d4d4de8b96fa",
  "name": "Napster_Client_6600-6699",
  "type": "service-tcp",
  "domain": {
    "uid": "a0bbbc99-adef-4ef8-bb6d-defdefdefdef",
    "name": "Check Point Data",
    "domain-type": "data domain"
  },
  "enable-tcp-resource": false,
  "sync-connections-on-cluster": true,
  "use-delayed-sync": false,
  "delayed-sync-value": 0,
  "port": "6600-6699",
  "match-by-protocol-signature": false,
  "override-default-settings": false,
  "session-timeout": 3600,
  "use-default-session-timeout": true,
  "match-for-any": false,
  "aggressive-aging": {
    "enable": true,
    "timeout": 600,
    "use-default-timeout": true,
```

8. Now filter the output to only display the names of the services that match the output:

```
cat tcp.json |jq '.objects[] | select(.color | contains("black" , "blue"))
|.name'
```

```
[Expert@sms_r8120:0]# cat tcp.json |jq '.objects[] | select(.color | contains("black" , "blue")) |.name'
"BGP"
"CheckPointExchangeAgent"
"Citrix_ICA"
"ConnectedOnLine"
"CP_SSL_Network_Extender"
"daytime-tcp"
"Direct_Connect_TCP"
"discard-tcp"
"domain-tcp"
"echo-tcp"
"EDGE"
"eDonkey_4661"
"eDonkey_4662"
"FIBMGR"
"finger"
"GNUtella_rtr_TCP"
"GNUtella_TCP"
"GoToMyPC"
"Hotline_client"
"HTTP_and_HTTPS_proxy"
"HTTPS_proxy"
```

Task – 4: Find and export Group Information

1. Create a new network group and place a few hosts within this new group. We will use it to find the following information
   - Number of members in the group
   - Type of objects in the group
   - Output members as CSV to be used in other scripts

2. Using the name of the group created in step 1, run the following command

```
mgmt_cli -r true show group name GroupName -f json |jq -r '.members | length'
```

   - What was output?

3. Now run the following command to see what types of objects are in the group

```
mgmt_cli -r true show group name GroupName -f json |jq -r '.members[] |
[.type] | @csv'
```

4. Can you output the uid, name, and type of members of this group? Use the command in step 3 as a basis for the new command.

```
End of Lab 2
```

