DevOps Engineer – Challenge Answers

Candidate: Iommi Underwood

1 Answer for Question 1

For Consul cluster implementation, VirtualBox was used to install two identical minimal centos 7 virtual nodes. The following section will detail the Consul cluster implementation and will offer justification for this implementation.

1.1 Consul Cluster implementation

This section covers the Consul cluster implementation process. Almost all tasks shown below were first executed on node **consul1**, this was then cloned to **consul2** to create the second node.

1. Installed useful tools: zip, unzip, git, wget, curl, net-tools and htop:
   - Command: `yum install zip unzip git wget curl net-tools htop -y`
   - Confirm:

   ```
   [root@consul1 ~]# which zip
   /usr/bin/zip
   [root@consul1 ~]# which unzip
   /usr/bin/unzip
   ```

2. Downloaded zip file in the consul home directory.

   ```
   wget https://releases.hashicorp.com/consul/1.4.2/consul_1.4.2_linux_amd64.zip
   ```

3. Uzipped download file and removed zip file.

   ```
   [consul@consul1 ~]$ unzip consul_1.4.2_linux_amd64.zip && rm -rf consul_*
   ```

4. Created /etc and /var consul directories.

   ```
   [root@consul1 ~]# mkdir -p /etc/consul.d/scripts
   [root@consul1 ~]# mkdir /var/consul
   ```

5. Moved the consul binary to /usr/local/bin and verified install.

   ```
   [root@consul1 ~]# cp -p /home/consul/consul /usr/local/bin/
   [root@consul1 ~]# consul -v
   Consul v1.4.2
   Protocol 2 spoken by default, understands 2 to 3 (agent will automatically use
   protocol >2 when speaking to compatible agents)
   ```

6. Generated an encryption key to use in the next step.

   ```
   [root@consul1 ~]# consul keygen
   ba/gYoXsAqCtZmF2/vKRlg==
   ```

7. Created a config.json file with the following key value pairs:

   ```
   [root@consul1 ~]# cat /etc/consul.d/config.json
   {
       "bootstrap_expect": 2,
       "client_addr": "0.0.0.0",
       "datacenter": "iommi-home",
       "data_dir": "/var/consul",
       "domain": "iommi",
       "enable_script_checks": true,
   ```

```
    "dns_config": {
        "enable_truncate": true,
        "only_passing": true
    },
    "enable_syslog": true,
    "encrypt": "ba/gYoXsAqCtZmF2/vKRlg==",
    "leave_on_terminate": true,
    "log_level": "INFO",
    "rejoin_after_leave": true,
    "server": true,
    "start_join": [
        "192.168.0.38",
        "192.168.0.39"
    ],
    "ui": true
}
```

8. Created a system service file for Consul.

```
[root@consul1 ~]# cat /etc/systemd/system/consul.service
[Unit]
Description=Consul Startup process
After=network.target

[Service]
Type=simple
ExecStart=/bin/bash -c '/usr/local/bin/consul agent -config-dir /etc/consul.d/'
TimeoutStartSec=0

[Install]
WantedBy=default.target
```

9. Reloaded the systemctl daemon.

```
[root@consul1 ~]# systemctl daemon-reload
```

10. Enabled service to start on statrup.

```
[root@consul1 ~]# systemctl enable consul
Created symlink from /etc/systemd/system/default.target.wants/consul.service to
/etc/systemd/system/consul.service.
```

11. Started the Consul service.

```
[root@consul1 ~]# systemctl start consul
```

12. Checked the consul service status and ensured no errors in logs were present.

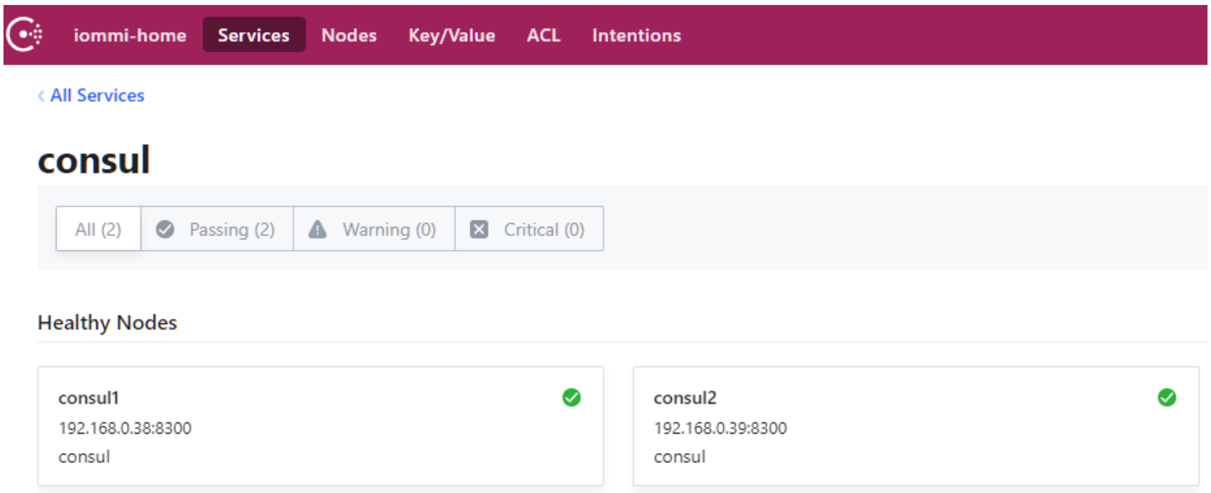
```
[root@consul1 ~]# systemctl status consul
● consul.service - Consul Startup process
   Loaded: loaded (/etc/systemd/system/consul.service; enabled; vendor preset:
disabled)
   Active: active (running) since Sun 2019-02-03 10:35:21 GMT; 5min ago
 Main PID: 3734 (consul)
   CGroup: /system.slice/consul.service
           └─3734 /usr/local/bin/consul agent -config-dir /etc/consul.d/
```

13. At this point I joined node **consul2** into a cluster with **consul1**.

```
[root@consul2 ~]# consul join 192.168.0.38
Successfully joined cluster by contacting 1 nodes.

[root@consul1 ~]# consul members
Node     Address            Status  Type    Build  Protocol  DC          Segment
consul1  192.168.0.38:8301  alive   server  1.4.2  2         iommi-home  <all>
consul2  192.168.0.39:8301  alive   server  1.4.2  2         iommi-home  <all>
```

The figure below illustrates that both nodes in the cluster are healthy.



consul

Healthy Nodes

consul1
192.168.0.38:8300
consul

consul2
192.168.0.39:8300
consul

1.2 Creating key/values

Although key/values can be created using the UI, I felt it was quicker and easier to create these using the CLI. Below are the commands run to create and verify the KVs

1.2.1 Domains

1. Create domains.

```
[root@consul1 ~]# consul kv put domains/domain1 app01-vm-prd.mt.local
Success! Data written to: domains/domain1

[root@consul1 ~]# consul kv put domains/domain2 app01-vm-prd.mt.local
Success! Data written to: domains/domain2
```

2. Verify domains.

```
[root@consul1 ~]# consul kv get -recurse domains
domains/:
domains/domain1:app01-vm-prd.mt.local
domains/domain2:app02-vm-prd.mt.local
```

1.2.2 Whitelist

1. Create whitelists.

```
[root@consul1 ~]# consul kv put whitelist/app01-vm-prd.mt.local/whitelist-1 10.0.0.8
Success! Data written to: whitelist/app01-vm-prd.mt.local/whitelist-1

[root@consul1 ~]# consul kv put whitelist/app01-vm-prd.mt.local/whitelist-2 10.1.8.3
Success! Data written to: whitelist/app01-vm-prd.mt.local/whitelist-2

[root@consul1 ~]# consul kv put whitelist/app01-vm-prd.mt.local/whitelist-3 10.5.2.4
Success! Data written to: whitelist/app01-vm-prd.mt.local/whitelist-3
[root@consul1 ~]# consul kv put whitelist/app02-vm-prd.mt.local/whitelist-1
192.168.20.9
Success! Data written to: whitelist/app02-vm-prd.mt.local/whitelist-1
[root@consul1 ~]# consul kv put whitelist/app02-vm-prd.mt.local/whitelist-2
192.168.50.7
Success! Data written to: whitelist/app02-vm-prd.mt.local/whitelist-2

[root@consul1 ~]# consul kv put whitelist/app02-vm-prd.mt.local/whitelist-3
192.168.20.5
Success! Data written to: whitelist/app02-vm-prd.mt.local/whitelist-3
```

2. Verify whitelists.

```
[root@consul1 ~]# consul kv get -recurse whitelist
whitelist/:
whitelist/app01-vm-prd.mt.local/:
whitelist/app01-vm-prd.mt.local/whitelist-1:10.0.0.8
whitelist/app01-vm-prd.mt.local/whitelist-2:10.1.8.3
whitelist/app01-vm-prd.mt.local/whitelist-3:10.5.2.4
whitelist/app02-vm-prd.mt.local/:
whitelist/app02-vm-prd.mt.local/whitelist-1:192.168.20.9
whitelist/app02-vm-prd.mt.local/whitelist-2:192.168.50.7
whitelist/app02-vm-prd.mt.local/whitelist-3:192.168.20.5
```

1.3 Generating templates

Templates were generated by installing the consul-template module.

```
[root@consul1 ~]# cd /usr/local/bin/
[root@consul1 bin]# ls -lrth
total 103M
-rwxr-xr-x. 1 root consul 7.4M Jun 12  2018 consul-template
-rwxr-xr-x. 1 root consul  96M Jan 28 22:50 consul
 [root@consul1 bin]# consul-template -v
consul-template v0.19.5 (57b6c71)
```

According to the official HashiCorp documentation, a template file using a "tpl" needs to be created using the Go language. The following sub sections indicate how the results were achieved.

1.3.1 Domains

1. To generate the domains '/root/domains' file, I created a template file called domains.tpl, this was used to read values from consul keys, and insert values in the generated file.

```
[root@consul1 consulTemplates]# cat domains.tpl
{{ key "domains/domain1"  }}
{{ key "domains/domain2"  }}
```

2. Using the consul-template command, I loaded the file into its desired folder.

```
[root@consul1 consulTemplates]# consul-template -template="domains.tpl:/root/domains" -once
[root@consul1 consulTemplates]# cat /root/domains
app01-vm-prd.mt.local
app01-vm-prd.mt.local
```

1.3.2 Whitelist

1. To generate the whitelist '/root/whitelist' file, I created a template file called whitelist.tpl, this was used to read values from consul keys, and insert values in the generated file.

```
[root@consul1 consulTemplates]# cat whitelist.tpl
{{ key "domains/domain1" }}:
  - whitelist:
    - permit {{key "whitelist/app01-vm-prd.mt.local/whitelist-1" }} ;
    - permit {{key "whitelist/app01-vm-prd.mt.local/whitelist-2" }} ;
    - permit {{key "whitelist/app01-vm-prd.mt.local/whitelist-3" }} ;
{{ key "domains/domain2" }}:
  - whitelist:
    - permit {{key "whitelist/app02-vm-prd.mt.local/whitelist-1" }} ;
    - permit {{key "whitelist/app02-vm-prd.mt.local/whitelist-2" }} ;
    - permit {{key "whitelist/app02-vm-prd.mt.local/whitelist-3" }} ;
```

Note: Although the next point shows that the expected outcome is achieved, this template can be refactored to have some sort of a loop that automatically includes all domains added to the consul whitelist KVs according to domain.

2. Using the consul-template command, I loaded the file into its desired folder.

```
[root@consul1 consulTemplates]# consul-template -template
"whitelist.tpl:root/whitelist" -once
[root@consul1 consulTemplates]# cat /root/whitelist
app01-vm-prd.mt.local:
  - whitelist:
    - permit 10.0.0.8 ;
    - permit 10.1.8.3 ;
    - permit 10.5.2.4 ;
app01-vm-prd.mt.local:
  - whitelist:
    - permit 192.168.20.9 ;
    - permit 192.168.50.7 ;
    - permit 192.168.20.5 ;
```

2 Answer for Question 2

To build a VirtualBox image I used Packer, as research showed this is the best tool for image creation and service provisioning. Packer fits our requirement because it provides automation of image creation.

Ansible was used as part of the provisioner inside the image template to install and configure Nginx.

2.1 Building VirtualBox image

This section describes the process for creating a virtual box image template.

1. After installing packer, I created a template file.

    ```
    C:\Users\lommi\Desktop\imageBuilder\centosImageTemplate.json
    ```

2. Built a template file for image creation.

```
{
  "variables": {
  "iso_url": "./iso/centos-1810.iso",
  "iso_checksum_type": "sha256",
  "iso_checksum": "38d5d51d9d100fd73df031ffd6bd8b1297ce24660dc8c13a3b8b4534a4bd291c",
  "vm_name": "xcaliber-centos"
  },

  "builders": [
    {
      "type": "virtualbox-iso",
      "vm_name": "{{ user `vm_name` }}",
      "iso_url": "{{ user `iso_url` }}",
      "iso-checksum_type": "{{ user `iso_checksum_type` }}",
      "iso_checksum": "{{ user `iso_checksum` }}",
      "guest_os_type": "RedHat_64",
      "cpus": 1,
      "memory": 1024,
      "disk_size": 30720,
      "hard_drive_interface": "scsi",
      "headless": false,
      "communicator": "ssh",
      "ssh_username": "root",
      "ssh_password": "toor",
      "ssh_port": 22,
      "ssh_timeout": "1h",
      "shutdown_timeout": "30m",
      "shutdown_command": "sys-unconfig",
      "boot_wait": "5s",
      "http_directory": ".\\kickstart\\",
      "boot_command": [
        "<tab> text ks=http://{{ .HTTPIP }}:{{ .HTTPPort }}/kickstart.cfg<enter><wait>"
      ],
      "guest_additions_mode": "attach",
      "output_directory": "./builds/xcaliber-centos",
      "format": "ova"
    }
  ]

  "provisioners": [
    {
      "type": "shell",
      "script":  "./provision/script.sh"
    },
    {
      "type": "ansible-local",
      "playbook_file": "./ansible/nginxPlaybook.yml"
    }
  ]
}
```

3. Additional to the bash script, the provisioners list also contains a dictionary for installing Nginx using a playbook file.

```
---
- hosts: vm-prd
  become: true
  tasks:
    - name: Installing nginx
      apt: name=nginx update_cache=yes
    - name: Starting nginx on boot
      service: name=nginx enabled=yes state=started
    - name: Setup nginx conf
      template:
        src=nginx.tpl
        dest=/etc/nginx/nginx.conf
      notify: restart nginx
  handlers:
    - name: restart nginx
      service:
       name=nginx
       state=restarted
```

4. I have also used a template to convert to nginx.conf, this holds all configuration information, including log file destination and naming.

```
user root;
worker_processes 4;
pid /run/nginx.pid;

events {
  worker_connections 500;
  # multi_accept on;
}

www_sites {
  vhost1: "app01-vm-prd.mt.local"
  vhost2: "app02-vm-prd.mt.local"
}

http {
  ##
  # Basic Settings
  ##
  sendfile on;
  tcp_nopush on;
  tcp_nodelay on;
  keepalive_timeout 65;
  types_hash_max_size 2048;
  server_tokens off;
  include /etc/nginx/mime.types;
  default_type application/octet-stream;
  ##
  # SSL Settings
  ##
  ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
  ssl_prefer_server_ciphers on;
  ##
  # Logging Settings
  ##
  access_log /var/log/nginx/$vhost_access.log timed_combined;
  error_log /var/log/nginx/$vhost_error.log;
  ##
  # Gzip Settings
  ##
  gzip on;
  gzip_disable "msie6";
  gzip_vary on;
  gzip_proxied any;
  gzip_comp_level 6;
  gzip_buffers 16 8k;
  gzip_http_version 1.1;
  gzip_min_length 256;
  gzip_types text/plain text/css application/json application/javascript text/xml
application/xml application/xml+rss text/javascript;
```

```
##
# Virtual Host Configs
##
include /etc/nginx/conf.d/*.conf;
include /etc/nginx/sites-enabled/*;
```

3 Conclusion

While being very knowledgeable in Linux and several I.T. Systems tools, I did not have any prior about the HashiCorp suite of products. Through research I found that this suite of products is very useful in a DevOps culture, and I became interested in knowing more about them.

Because of my keen interest in CI/CD, and the automation / orchestration of I.T. infrastructure, I did have a very high-level overview of what Ansible is and what it is used for, I was aware of its advantages, and therefore decided to use it during the image creation process in the packer template. I found this challenge to be quite interesting.

Although I am aware of some possible misconfigurations, I am very confident that further research and training will lead me to solve problems encountered.

Throughout this challenge I attempted to demonstrate my technical skills to the best of my ability. Given the opportunity I am more than willing to exhibit this work ethic consistently throughout my employment with XCaliber.

Looking forward to meeting you and discuss this challenge.

Best regards,
Iommi Underwood