Automating the database onboarding using Ansible®

Introduction

In the digital era, infrastructure is a key building block to the connected business, and Infrastructure protection is the need of the hour for business continuity.

With infrastructure ready, the next task for the administrator is to protect the workload. For workload protection, admins would onboard the workloads, configure them and then schedule the backups in backup software. In a typical data center, there are different workloads that need to be protected. It would be easier to deploy and configure manually for a smaller number of deployments (say, less than 10). When the business grows esp. in today's cloud era, infrastructure scales very quickly and grows massively. With growing workloads, manually protecting each workload is a tedious and uphill task for admins and manual operations are more prone to errors.

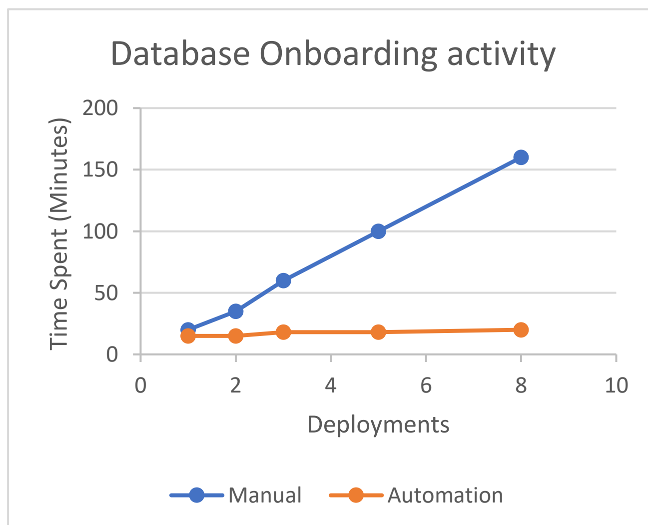
With Infrastructure-as-Code (IaC), administrators' effort in building the infrastructure and protecting it has reduced drastically, which helps the business go live sooner than expected. This document describes how to use IaC (Ansible) to onboard, configure and protect your database workload with Commvault.

This document is intended for administrators, and engineers who can leverage automation to simplify their day-to-day backup management operations.

Automation is key to business

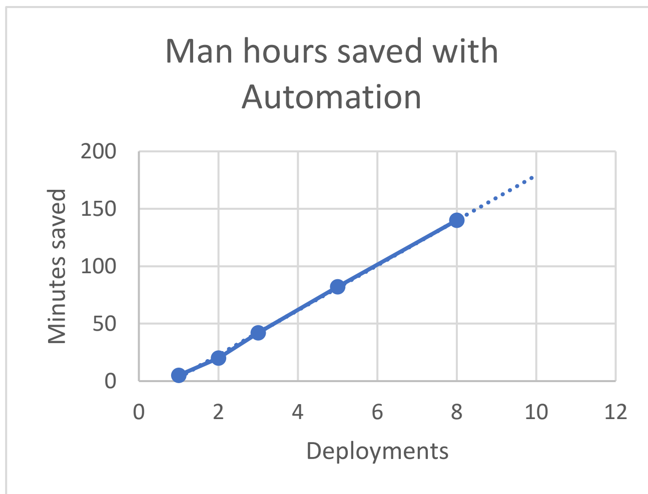
Automation brings in a lot of advantages, as this would simplify the admin's day-to-day activity and allow him to concentrate on critical work. More importantly, the workloads are protected in quick time with reduced business downtime risks and bringing in cost-effectiveness and IT efficiency for the organization.

With automation, there are proven steps that can execute any number of times, faster, and parallel execution for different servers and workloads that needs protection. To automate, you can use Commvault's package with the most used IaC (Infrastructure as service) framework such as Terraform, and Ansible for deployment and configuration management.



To demonstrate the importance of Automation, we did an experiment where we onboarded the Oracle database on Linux manually and automated way.

Manual onboarding involved adding a storage and plan, installing Oracle package on the client, and starting the backup. Similar steps were executed with the automated Ansible (as outlined in the next section) script and the results were not surprising and expected. With automation, the operations were executed in parallel, and the efforts were constant.

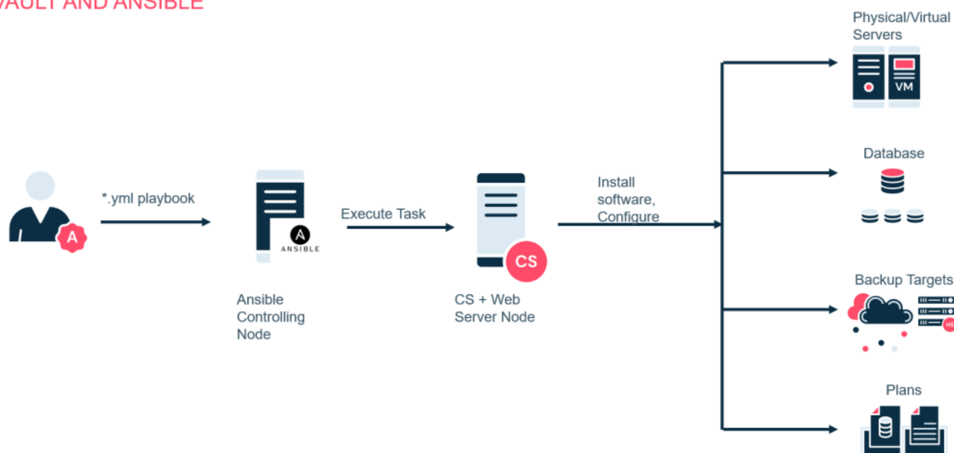


With the increased deployment and repetitive stuff, automation saved a lot of time. With 3 deployments, there were around ~50 minutes of effort that was saved.

Note that the results are based on the environment the tests are conducted. As seen from the graph, there is a linear effort saved when the deployment is high and repetitive.

How to onboard database with Commvault Ansible Collection

In an Ansible world, user writes an YAML file with tasks to be run on the managed node. The controlling node where the ansible playbook is executed, takes the YAML as input and runs the suitable package on the managed node to get the desired output.



As an example, the steps show how to onboard the Oracle database with Commvault's Ansible collection. Here we create an individual yaml for each of the tasks for more readable and import in the *main.yml.*

(1) Define the variables in *main.yml.*

```yaml
---
- name: "Database deployment and configuration"
  hosts: localhost
  remote_user: root
  become: true

  vars:
    - var_storageName: 'store1'
    - var_plan: 'plan1'
    - var_clientName: 'oracle.commvault.com'
    - var_dbInst: 'ORCL'
    - var_dbUser: 'sys'
    - var_dbPwd: 'databasePassword'
    - var_dbPwd2: "{{ var_dbPwd|b64encode }}"
    - var_dbHome: "/home/oracle”
    - var_osUser: "oracle"

    - import_tasks:
        createStorage.yml
        createPlan.yml
        addServer.yml
        getClientId.yml
        getInstanceId.yml
        reConfigDb.yml
        getDefaultSubClient.yml
        startBackup.yml
...
```

(2) Create the backup destination (Storage target).
The task would create a disk library on the media server ‘mediaAgent.commvault.com’.

```yaml
# createStorage.yaml
  tasks:
    - name: "Creating disk Storage"
      commvault.ansible.storage.disk.add:
        webserver_hostname: 'webserver.commvault.com'
        commcell_username: 'admin'
        commcell_password: 'password'
        name: "{{ var_storageName }}"
        media_agent: "mediaAgent.commvault.com"
        mount_path: "e:\\DL2"
        deduplication_db_path: "e:\\{{ var_storageName }}"
      tags: storage
```

(3) Create the plan with a suitable RPO

```yaml
# createPlan.yml
    - name: "Creating plan"
      commvault.ansible.plans.add:
        name: "{{ var_plan }}"
        type: "Server"
        storage_pool_name: "{{ var_storageName }}"
        rpo_minutes: 100
      register: plan_resp
      tags: plan

    - set_fact:
        plan_id: "{{plan_resp.id | default(1)}}"
```

(4) Once the plan is created, add the Oracle server to the Commvault cell. This step will install agent package, create sub clients and associate with the server plan.

This can be achieved by using REST API. The REST API is to be obtained from the Command Center “equivalent API” and then you need to convert the *json* payload to *YAML*. Or the REST APIs can be obtained from REST API reference documentation.

```
# addServer.yml
- name: "Add Oracle Server"
  commvault.ansible.request:
    method: 'POST'
    url: '{0}/createTask'
    payload:
      taskInfo:
        task:
          taskFlags:
            disabled: false
          taskType: IMMEDIATE
          initiatedFrom: GUI
        associations:
          - clientId: 0
            commCellId: 2
        subTasks:
          - subTask:
              subTaskType: ADMIN
              operationType: INSTALL_CLIENT
            options:
              adminOpts:
                updateOption:
                  rebootClient: true
                  plan:
                    planId: 1
                clientInstallOption:
                  clientDetails:
                    - clientEntity:
                        clientId: 0
                        clientName: "{{ var_clientName }}"
                        commCellId: 2
                  installOSType: UNIX
                  discoveryType: MANUAL
                  installerOption:
                    RemoteClient: false
                    requestType: PRE_DECLARE_CLIENT
                    User:
                      userId: 1
                      userName: admin
                    Operationtype: INSTALL_CLIENT
                    CommServeHostName: “webserver.commvault.com”
                    clientComposition:
                      - overrideSoftwareCache: false
                        clientInfo:
                          client:
                            cvdPort: 0
                            evmgrcPort: 0
                        components:
```

```yaml
                              - osType: Unix
                                ComponentId: 1204
                              - osType: Unix
                                consumeLicense: false
                                ComponentId: 1301
                            commonInfo:
                              globalFilters: UseCellLevelPolicy
                            fileSystem:
                              configureForLaptopBackups: false
                          packageDeliveryOption: CopyPackage
                    installFlags:
                      install32Base: false
                      disableOSFirewall: false
                      addToFirewallExclusion: true
                      killBrowserProcesses: true
                      ignoreJobsRunning: false
                      stopOracleServices: false
                      skipClientsOfCS: false
                      restoreOnlyAgents: false
                      overrideClientInfo: true
                      firewallInstall:
                        enableFirewallConfig: false
                        firewallConnectionType: 0
                        portNumber: 0
                  clientAuthForJob:
                    userName: "root"
                    password: "base64EncodedPassword"
                  reuseADCredentials: false
              commonOpts:
                subscriptionInfo: <Api_Subscription subscriptionId ="136"/>
  register: response

- set_fact:
    jobid_addServer: "{{ response.response.jobIds[0] }}"

- debug:
    msg: "{{ jobid_addServer }}"

- name: "Wait for Job Status"
  commvault.ansible.job.status:
    job_id: "{{ jobid_addServer | int }}"
    wait_for_job_completion: true
```

(5) Get the client ID after the successful client import to the Commcell.

```yaml
  - name: "Get client id"
    commvault.ansible.request:
      method: 'GET'
      url: '{0}/GetId?clientname={{ var_clientName }}'
    register: cliIdResp

  - set_fact:
      client_id: "{{ cliIdResp.response.clientId }}"
```

(6) Get the instance Id

```yaml
  - name: "Get instance id"
    commvault.ansible.request:
      method: 'GET'
      url: '{0}/instance?clientId={{ client_id }}'
    register: InstIdResp

  - set_fact:
      instance_id:
        "{{ InstIdResp.response.instanceProperties[0].instance.instanceId
}}"
      instance_guid:
        "{{InstIdResp.response.instanceProperties[0].instance.instanceGUID}}
```

(7) Re-configure the instance to use the database credentials

```
  - name: "Re-configure the instance"
    commvault.ansible.request:
      method: 'POST'
      url: '{0}/instance/{{ instance_id }}'
      payload:
        instanceProperties:
          instance:
            instanceId: "{{ instance_id|int }}"
            applicationId: 22
            clientId: "{{ client_id|int }}"
          oracleInstance:
            oracleUser:
              userName: "{{ var_osUser }}"
            oracleHome: "{{var_dbHome|safe}}"
            sqlConnect:
              userName: "{{var_dbUser}}"
              password: "{{var_dbPwd2}}"
              domainName: "{{ var_dbInst }}"
              confirmPassword: "{{var_dbPwd2}}"
              savedCredential: {}
            useCatalogConnect: false
            blockSize: 1048576
            oracleStorageDevice:
              commandLineStoragePolicy:
                storagePolicyId: "{{plan_id|int}}"
                storagePolicyName: "{{var_plan}}"
              logBackupStoragePolicy:
                storagePolicyId: "{{plan_id|int}}"
                storagePolicyName: "{{var_plan}}"
            crossCheckTimeout: 600
            oracleWalletAuthentication: false
          planEntity:
            planId: "{{plan_id|int}}"
        association:
          entity:
            - displayName: "{{var_clientName}}"
              clientId: "{{ client_id|int }}"
              instanceGUID: "{{ instance_guid }}"
              instanceName: "{{ var_dbInst }}"
              appName: Oracle
              applicationId: 22
              clientName: "{{var_clientName}}"
              instanceId: "{{ instance_id|int }}"
    register: reconf_resp
```

(8) Get default subclient id

```
  - name: "Get subclient ID"
    commvault.ansible.request:
      method: 'GET'
      url: '{0}/subclient?clientId={{ client_id }}'
    register: subCliIDResp
    when: reconf_resp is defined

  - set_fact:
      subclient_Id: "{{ subCliIDResp.response.\
          subClientProperties[0].subClientEntity.subclientId }}"
```

(9) Execute the backup

```yaml
  - name: "Start Backup"
    commvault.ansible.request:
      method: 'POST'
      url: "{0}/Subclient/{{ subclient_Id }}/\
            action/backup?backupLevel=Incremental&runIncrementalBackup=false
&incrementalLevel=AFTER_SYNTH"
    register: backupResp
    when: reconf_resp is defined
    tags: bkup

  - debug:
      msg: "backup response: {{ backupResp }} "

  - name: "Wait for Job Status"
    commvault.ansible.job.status:
      job_id: "{{ backupResp.response.jobIds[0] | int }}"
      wait_for_job_completion: false
```

More information on the supported Ansible functionality can be found in the github.

Summary

With Commvault Ansible collection, the functionality is not limited to just the onboarding of workloads. It could be extended to another use case such as governance and compliance, Job management, and so on.

In conclusion, enterprises deal with different kinds of workloads that need to managed and protected. Automation becomes a key to cost-effective and efficient backup and recovery for these workloads.

Automation speeds up deployment, and configuration operations while minimizing error and reducing costs. It guarantees a consistent experience for database configuration and deployment thus boosting the confidence of admins in managing their workloads.

Reference

1. Commvault Ansible Collection
2. Commvault REST API Reference