Running Control-M Workloads in Kubernetes

Best Practices for using Control-M to run a pod to completion in a Kubernetes-based cluster

July 2020

Table of Contents

Running Control-M Workloads in Kubernetes

This white paper describes how to run Control-M workloads in Kubernetes-based clusters, including OpenShift (from RedHat) and public cloud platforms such as EKS (from Amazon), AKS (from Microsoft Azure), GKE (from Google). The information in this document focuses on the scenario of running a pod to completion with a Control-M/Agent running within the cluster. The current document demonstrates how to do this while leveraging Control-M scheduling and monitoring capabilities.

The methodology and configuration described in this document were tested and verified by BMC. The recommended best practices do not impact or change BMC licensing considerations.

Working Assumptions

The information in this document is based on the following assumptions:

- The following guidelines and configuration are based on Amazon Elastic Kubernetes Service. Configuration on other flavors of Kubernetes (such as OpenShift) should be adjusted accordingly.
- Deployment of Control-M/Agent on Kubernetes requires Control-M 9.0.18 or later as the basis for running Control-M Automation API.
- Control-M/Server resides outside of the Kubernetes cluster.
- The kubectl utility is configured in your environment.

Methodological Approach

There are Several ways to run a Kubernetes (K8s) pod to completion using Control-M. This can be done using a custom job type defined in Application Integrator or using an OS job.

This document describes a selected methodology for utilizing an OS job with a Python script to run your Control-M workloads in Kubernetes. The Python script resides in the docker image. The Python script starts the Kubernetes job entity and monitors the status until the job ends. After the job ends, the pod’s output is captured and is presented in the Control-M job output. The Kubernetes job is then deleted.

The agent pod is run as a StatefulSet, so that the hostname will be identified consistently across pod shutdown and startup. This enables Control-M/Server to uniquely identify the Control-M/Agent continuously and consistently.

The pod uses a persistent storage, so that job data and Control-M/Agent state are kept across pod shutdown and startup.

The connection between the Control-M/Agent and Control-M/Server is set up as a persistent connection that is initiated by the agent. This was designed to avoid exposing the Kubernetes cluster to outside connections.

To view a sample container, pod, and Python script, refer to Sample Objects.

The Python script uses the following Kubernetes building blocks:

Objective	**Corresponding CLI command**
Start the job	`kubectl apply -f job.yaml`
Get details	`kubectl describe job “job name”` This returns the pod name.
Monitor the job’s pod	`kubectl get pod “pod name”`
Get the job output	`kubectl logs “pod name”`

Sample Objects

Use the following links to access and obtain sample objects in GitHub. Note that you will need to edit these samples and customize them to match the unique needs of your environment.

File Name and Link to GitHub	Description
build_docker_example.sh	A bash script that is used to build a Docker image
Dockerfile	A file that contains instructions for how to construct a Control-M/Agent Docker image.
runJob.py	A Python script to run as an OS job in Control-M, mediating between Control-M and Kubernetes
pvc.yaml	Kubernetes definitions for creation of the Persistent Volume Claim
container_agent_startup.sh	A bash script that is used to start up the Control-M/Agent
stateful.yaml	A file that contains StatefulSet definitions for running a Control-M/Agent container
sleep_for_20sec_job.yaml	A sample Kubernetes job: Print the current time and sleep for 20 seconds
controlm_jobs_example.json	A .json file with definitions of two Control-M OS jobs: • The first job uses the runJob.py Python script to run an existing job (sleep_for_20sec_job.yaml). • The second job uses the runJob.py Python script to create a “Hello World” job during run time.

Process Steps

A typical process of deploying Control-M/Agent on Kubernetes consists of the following steps:

Step 1: Create a Control-M/Agent image

Step 2: Create persistent storage for the agent

Step 3: Run the agent

Step 4: Verify agent-server connection

Step 5: Add the agent to a Host Group

Step 6: Verify permissions

Step 7: Run Control-M jobs

Note: The processes described here are meant as guidelines for running workloads in Kubernetes-based clusters. These processes have been tested in BMC environments, but may need to be adjusted to run in your environments. If you need support in deploying Control-M in this manner, please request assistance from BMC Support.

Step 1: Create a Control-M/Agent image

Edit the build_docker_example.sh script with local values for the variables that appear in uppercase (AAPI_END_POINT, AAPI_USER, AAPI_PASS, and AGENT_IMAGE_NAME), and then run the script.

This produces an image that contains a Control-M/Agent and a Python script. The Python utilizes the Kubernetes SDK to run and monitor pods.

The container contains the following items:

- Control-M/Agent
- Python run time
- Python K8S SDK
- Java run time
- NodeJS
- Control-M Automation API CLI
- Python script
- Bash startup script

Automation API instructions in the Dockerfile are invoked during the build of the image for provisioning of an agent.

The startup script configures the agent to use the persistent storage.

Step 2: Create persistent storage for the agent

Use the pvc.yaml file to allocate persistent storage for agent usage, so that the state of the Control-M/Agent is kept across shutdown and startup of the pod.

Run the following command:
```
kubectl apply -f pvc.yaml
```

Step 3: Run the agent

Use the stateful.yaml file to run your container as a StatefulSet that uses the persistent storage defined in the previous step. Running the container as a StatefulSet ensures that the hostname will stay the same.

1) Update the placeholder values in the **stateful.yaml** file with your relevant data.
2) Run the following command to start the StatefulSet service:
   ```
   kubectl apply -f stateful.yaml
   ```

Upon startup, the container executes the startup script, which performs the following actions:

- Maps data directories to the persistent storage
- Uses Automation API to configure the agent:
  1) Registers the agent in the requested Control-M/Server.
  2) Configures a persistent connection.
  3) Starts the agent process.

Step 4: Verify agent-server connection

To verify the connection between the Control-M/Agent and Control-M/Server, run the following checks:

Check	Command	Expected Output

Control-M/Agent side	`kubectl exec -it statefulset-agent-0 -- tcsh -c ag_ping`
	This command runs the ag_ping utility, which resides in the Control-M/Agent image. The StatefulSet is named satefulset-agent, with a number suffix for the instance number (0).
		`Server is alive.`
		`Result: Success.`

Control-M/Server side	Run the following command from the Control-M/Server account: `ctmping -HOSTID statefulset-agent-0` This command runs the ctmping utility to verify the connection between the server and the agent.	`Agent : statefulset-agent-0 is alive`

Step 5: Add the agent to a Host Group

Use Automation API to add the agent to a host group to create a *virtual agent name*. This allows you to maintain several Agents in Kubernetes that serve the workloads of a certain application.

You can then add or remove agents as needed, without impacting job definitions.

```
ctm config  server:hostgroup:agent::add <ctm> <hostgroup> <host>
```

Step 6: Verify permissions

Verify with your Kubernetes Administrator that the agent's pod has permissions to start the requested "run pod to completion".

Step 7: Run Control-M jobs

To run the Control-M job, perform the following actions:

1) In the controlm_jobs_example.json file, update the following properties:
   - `Host` – agent name or host group
   - `ControlmServer` – name of the Control-M/Server, if you have more than one in your environment
2) To ensure that the agent container has access to the **sleep_for_20sec_job.yaml** file, copy this file to the container. The following command is one way of getting the file into the pod:
   ```
   kubectl cp sleep_for_20sec_job.yaml  statefulset-agent-0:/home/controlm/
   ```
   Note: An alternative, more flexible method to ensure that the agent container has access to this file is to load a persistent volume and pull the file from there.
3) Through the Automation API CLI, run the following commands:
   a. Run the job: `ctm run controlm_jobs_example.json`
   Keep note of the returned run ID.
   b. Check job status: `ctm run status <run id>`
   c. Check job output: `ctm job:output::get <job id>`

Note that the K8s pod is deleted after execution to allow another job with the same name to run.

controlm_jobs_example.json contains 2 Control-M jobs:

- A predefined K8s job that runs to completion, in the form of a yaml file (that was copied to the container). The pod (container) prints the time, sleeps for 20sec, and prints the time again.
- A K8s job created during runtime using parameters (there is no yaml file for it), which runs to completion. The pod (container) prints the string *“Hello World!”*, passed as a parameter.