A Simplified Implementation of Cloud Computing

Yuxin Cui

School of Computer Science and Engineering

South China University of Technology, Guangzhou, Guangdong

Aug. 2022

ABSTRACT

Cloud computing is becoming an important part of every-day lives because it is powering critical applications from banking, social media, news, entertainment, and much more. Cloud computing is one of the most important parts of the modern infrastructure that is impacting our daily lives. In the course of Artificial Intelligence and Cloud Computing, I developed a simplified cloud system and deploy it to run some example applications.

Keywords: Artificial Intelligence, Cloud Computing, Docker

# Assignment #1 create cluster manager

I designed a simple cluster manager which support these commands below:

>>> python3 cm.py create 3

This command will create three docker containers with Ubuntu operating system.

>>> python3 cm.py create 3 centos

This command will create three docker containers with Centos operating system.

>>> python3 cm.py start

This command will start all docker containers.

>>> python3 cm.py exec “ls”

This command will make all docker containers execute “ls” command.

>>> pyhton3 cm.py exec 2 “ls”

This command will make two docker containers execute “ls” command.

>>> python3 cm.py stop

This command will stop all docker containers immediately.

>>> python3 cm.py stop 3

This command will wait three seconds and stop all docker containers.

>>> python3 cm.py list

This command will list all docker containers.

The cluster manager is able to create any number docker containers and start executing some commands in these docker containers.

The main function of cluster manager is given as follows:

1. **if** \_\_name\_\_ == '\_\_main\_\_':
2. argc = len(sys.argv)
3. **assert** argc >= 2
4. match sys.argv[1]:
5. case "create":
6. **if** len(sys.argv) == 3:
7. cm\_create(int(sys.argv[2]))
8. **elif** len(sys.argv) == 4:
9. cm\_create(int(sys.argv[2]), list(sys.argv[3]))
10. **elif** len(sys.argv) == 5:
11. cm\_create(int(sys.argv[2]), list(sys.argv[3]), sys.argv[4])
12. case "start":
13. cm\_start()
14. case "exec":
15. **if** len(sys.argv) == 3:
16. cm\_exec(len(containers), sys.argv[2])
17. **elif** len(sys.argv) == 4:
18. cm\_exec(int(sys.argv[2]), sys.argv[3])
19. case "stop":
20. **if** len(sys.argv) == 2:
21. cm\_stop()
22. **elif** len(sys.argv) == 3:
23. cm\_stop(int(sys.argv[2]))
24. case "delete":
25. cm\_delete()
26. case "list":
27. cm\_print\_status()
28. case "createVolume":
29. cm\_create\_volume(sys.argv[2])

# Assignment #2 Data processing in cluster manager

I designed a simple cluster manager which support these commands below:

>>> python3 cm.py createVolume disk1

This command will create a docker volume named disk1.

Through this command, we can put some data in the volume so that all the docker containers could access it. It allows us to process data concurrently. Each docker container can process a portion of the array concurrently and print the local results to screen.

Here is the implementation of creating volume:

1. **def** cm\_create\_volume(name):
2. """
3. Create a volume and return.
4. :param name: Name of the volume.
5. """
6. **try**:
7. volume = client.volumes.create(name=name, driver='local',
8. driver\_opts={'foo': 'bar', 'baz': 'false'},
9. labels={"key": "value"})
10. **except** docker.errors.APIError:
11. **print**("The server returns an error.")
12. **return**
13. **return** volume

# ASSigment #3 Ai tasks in cluster manager

If we use a image contains AI environment such as Tensorflow, we can easily write a program to get some work done in those docker containers.

Here is all code for cluster manager:

1. # Python Version 3.10
2. **import** docker
3. **import** sys

6. **def** cm\_create(num, images=['ubuntu'], volume=''):
7. """
8. Create containers without starting them.
9. :param volume: The volume you want to mount to the container.
10. :param num: The number of containers you want to create.
11. :param images: A list of images of containers.
12. """
13. **assert** len(images) != 0
14. **if** volume != '':
15. **try**:
16. volume = client.volumes.get(volume)
17. mp = docker.types.Mount("/home", volume)
18. **except** docker.errors.NotFound:
19. **print**("The volume does not exist.")
20. **return**
21. **except** docker.errors.APIError:
22. **print**("The server returns an error.")
23. **return**
24. **for** i **in** range(num):
25. **try**:
26. **if** volume != '':
27. temp = client.containers.create(images[i % len(images)], "sh", tty=True, mounts=[mp], name=str(i))
28. **else**:
29. temp = client.containers.create(images[i % len(images)], "sh", tty=True)
30. **except** docker.errors.ImageNotFound:
31. **print**("The specified image does not exist.")
32. cm\_stop()
33. **return**
34. **except** docker.errors.APIError:
35. **print**("The server returns an error.")
36. cm\_stop()
37. **return**
38. **else**:
39. containers.append(temp)
40. **print**("Container created .. " + temp.id)
41. cm\_print\_status()
42. cm\_save()

45. **def** cm\_start():
46. """
47. Start all containers.
48. """
49. **for** container **in** containers:
50. container.start()
51. cm\_print\_status()
52. cm\_save()

55. **def** cm\_print\_status():
56. """
57. Print containers status.
58. """
59. containers = client.containers.list(True)
60. **for** i **in** range(len(containers)):
61. **print**("Container", i, containers[i].id, " status:", containers[i].status)

64. **def** cm\_exec(num, command):
65. """
66. Run a command inside containers.
67. :param num: The number of containers you want to run the command.
68. :param command: Command to be executed.
69. """
70. **for** i **in** range(num):
71. **try**:
72. exitCode, output = containers[i % len(containers)].exec\_run(command, tty=True)
73. **except** docker.errors.APIError:
74. **print**("The No.", i, " server returns an error.")
75. **else**:
76. **print**(str(output, "utf-8"))
77. cm\_save()

80. **def** cm\_save():
81. """
82. Save container status in a file.
83. """
84. containers = client.containers.list(True)
85. file = open("containers.txt", "w")
86. **for** container **in** containers:
87. file.write(container.id + " " + container.status + '\n')
88. file.write('\n')
89. file.close()

92. **def** cm\_stop(time=0):
93. """
94. Stop containers.
95. :param time: Timeout in seconds to wait for the container to stop before sending a SIGKILL.
96. """
97. **for** container **in** containers:
98. container.stop(timeout=time)
99. cm\_print\_status()
100. cm\_save()

103. **def** cm\_delete():
104. """
105. Delete containers.
106. """
107. client.containers.prune()
108. cm\_print\_status()
109. cm\_save()

112. **def** cm\_create\_volume(name):
113. """
114. Create a volume and return.
115. :param name: Name of the volume.
116. """
117. **try**:
118. volume = client.volumes.create(name=name, driver='local',
119. driver\_opts={'foo': 'bar', 'baz': 'false'},
120. labels={"key": "value"})
121. **except** docker.errors.APIError:
122. **print**("The server returns an error.")
123. **return**
124. **return** volume

127. client = docker.from\_env()
128. containers = client.containers.list(True)
129. cm\_save()
131. **if** \_\_name\_\_ == '\_\_main\_\_':
132. argc = len(sys.argv)
133. **assert** argc >= 2
134. match sys.argv[1]:
135. case "create":
136. **if** len(sys.argv) == 3:
137. cm\_create(int(sys.argv[2]))
138. **elif** len(sys.argv) == 4:
139. cm\_create(int(sys.argv[2]), list(sys.argv[3]))
140. **elif** len(sys.argv) == 5:
141. cm\_create(int(sys.argv[2]), list(sys.argv[3]), sys.argv[4])
142. case "start":
143. cm\_start()
144. case "exec":
145. **if** len(sys.argv) == 3:
146. cm\_exec(len(containers), sys.argv[2])
147. **elif** len(sys.argv) == 4:
148. cm\_exec(int(sys.argv[2]), sys.argv[3])
149. case "stop":
150. **if** len(sys.argv) == 2:
151. cm\_stop()
152. **elif** len(sys.argv) == 3:
153. cm\_stop(int(sys.argv[2]))
154. case "delete":
155. cm\_delete()
156. case "list":
157. cm\_print\_status()
158. case "createVolume":
159. cm\_create\_volume(sys.argv[2])