BitPath-DevKit

AWS IoT Core FleetHub

RADIOSTUDIO

1 Table of Contents

2 Revision History

Version No	Date	Author	Change Log
1.0	29/04/2024	RadioStudio	

3 Requirement

Exploring the remote execution of commands on Raspberry Pi Devices via AWS IoT Core / Fleethub. Also, studying the remote device management

4 Solution

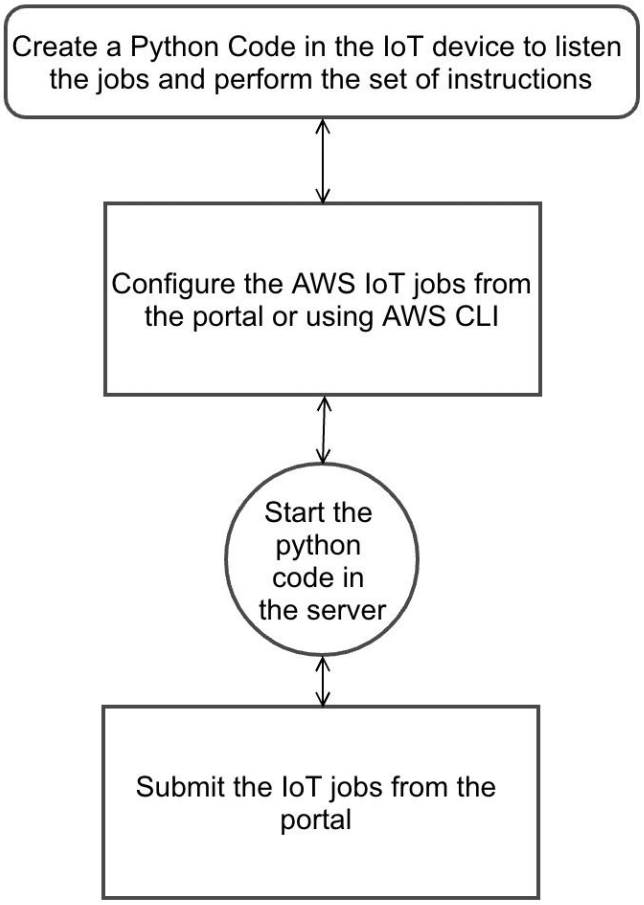
- **AWS IoT Jobs** - AWS IoT jobs can be used to define a set of remote operations sent to and executed on one or more devices connected to AWS IoT.

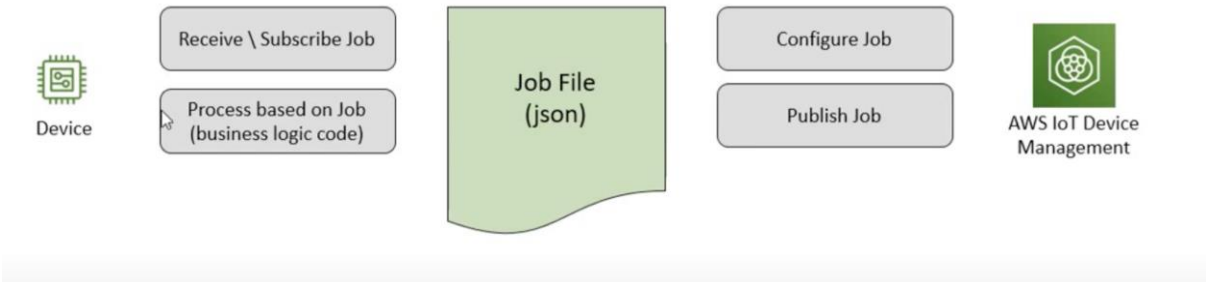
- **AWS Fleet Hub** - Fleet Hub is a web application that enables customers to manage their device fleets connected to AWS IoT.

Operation logic needs to be created in an IoT device. The job needs to be configured with instructions from the AWS IoT console, which the device can listen to and execute according to the instructions in the JSON file.

5 Architecture

5.1 Workflow





*Taken from a different website

5.2 Epics

1. Write a Python script - **joblistener-1.py** to connect with *AWS IoT jobs*

Code Snippet: The complete Python code is on the server under /opt/iotcore-jobs/. We are already running this code as a daemon.

Part I: First half of the code will read the device ID and the certificates installed on the device

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import time
import json
import requests
import os
import subprocess

#read the device id from the registration script output
with open('/opt/aws-iot-fleet-provisioning-v1/thingName.txt', 'r') as file:
   value = file.read().replace('\n', '')
deviceid = str(value)
print(deviceid)

topic = "bitpath/testing" #change the subscription topic as needed in the code and the AWS IoT subscription dashboard.

myMQTTClient = AWSIoTMQTTClient(deviceid)
myMQTTClient.configureEndpoint("", 8883)
myMQTTClient.configureCredentials("/opt/certificate/amazon-root-ca.pem","/opt/certificate/3a1e966d47-private.pem.key", "/opt/certificate/3a1e966d47-certificate.pem.crt")
myMQTTClient.connect()
print("Client Connected")
```

Part - II: This will read the IoT Job Operation Type and execute the jobs accordingly

```
    if operationType == "install":
        print ("Job Execution Started")
        url = payload['execution']['jobDocument']['files']['url']
        r = requests.get(url)
        with open ('/opt/iotcore-jobs/install.py', 'wb') as f:

            f.write(r.content)

        output = subprocess.check_output('python /opt/iotcore-jobs/install.py',shell=True)
        message = str(output)
        myMQTTClient.publish(topic, message, 0)
        updateJobStatus(jobId,"SUCCEEDED")
```

```
    if operationType == "uninstallation":
        print ("Job Execution Started")
        url = payload['execution']['jobDocument']['files']['url']
        r = requests.get(url)
        with open ('/opt/iotcore-jobs/rollback.py', 'wb') as f:

            f.write(r.content)

        output = subprocess.check_output('python /opt/iotcore-jobs/rollback.py',shell=True)
        message = str(output)
        myMQTTClient.publish(topic, message, 0)
        updateJobStatus(jobId,"SUCCEEDED")
```

2. Create an IoT job template in JSON format. You can name it according to the operation. I have tagged install-01.json and kept it in the AWS S3 bucket[us-east-1] named S3: *bitpath-jobs-demo.*

```
{
  "operation": "install",
  "version": "1.0",
  "packageName" : "install.py",
  "autoStart" : "true",
  "workingDirectory": "/opt/iotcore-jobs",
  "files": {
     "fileName": "install.py",
     "url" : "${aws:iot:s3-presigned-url:https://s3.us-east-1.amazonaws.com/bitpath-jobs-demo/install.py}"
  }
}
```

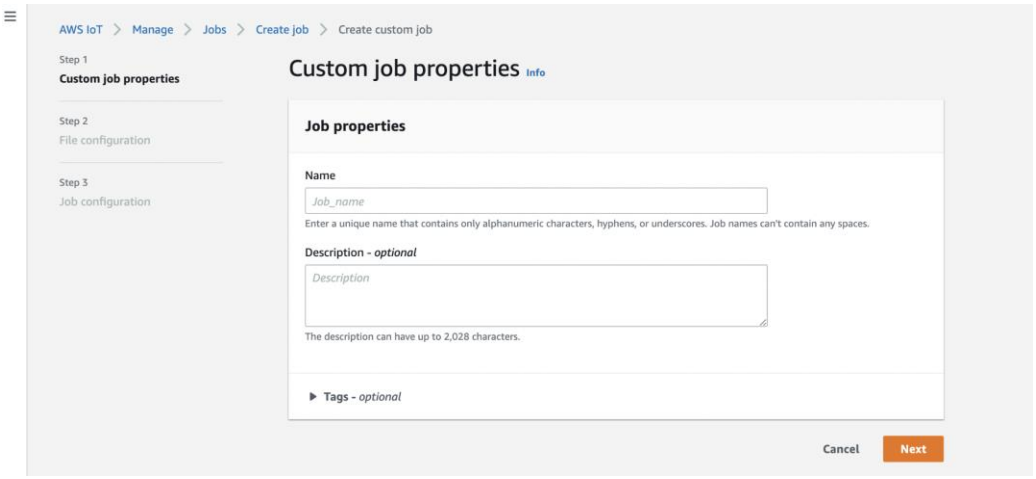
3. Now, as described in the JSON, package name install.py [In this file, we will write the commands that need to be executed on the device to install/download software, etc.]

Create a python file *install.py* with the below sample code and upload it to the same S3 bucket folder; here, this code will print the system time; we can change it as per our requirement, like Github Repo clone and deploy - hardware update, etc.[Discussed in sample deployment steps]

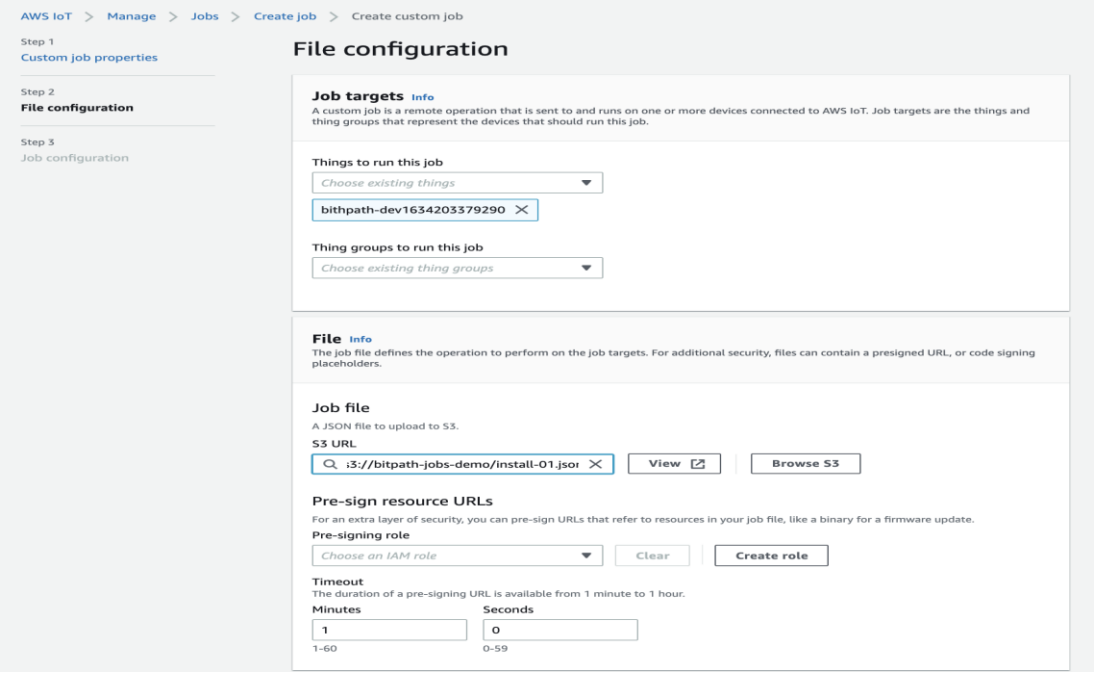
```
import os
cmd = 'uptime'
os.system(cmd)

print("A Sample file. Downloaded to Device and Executed !")
```

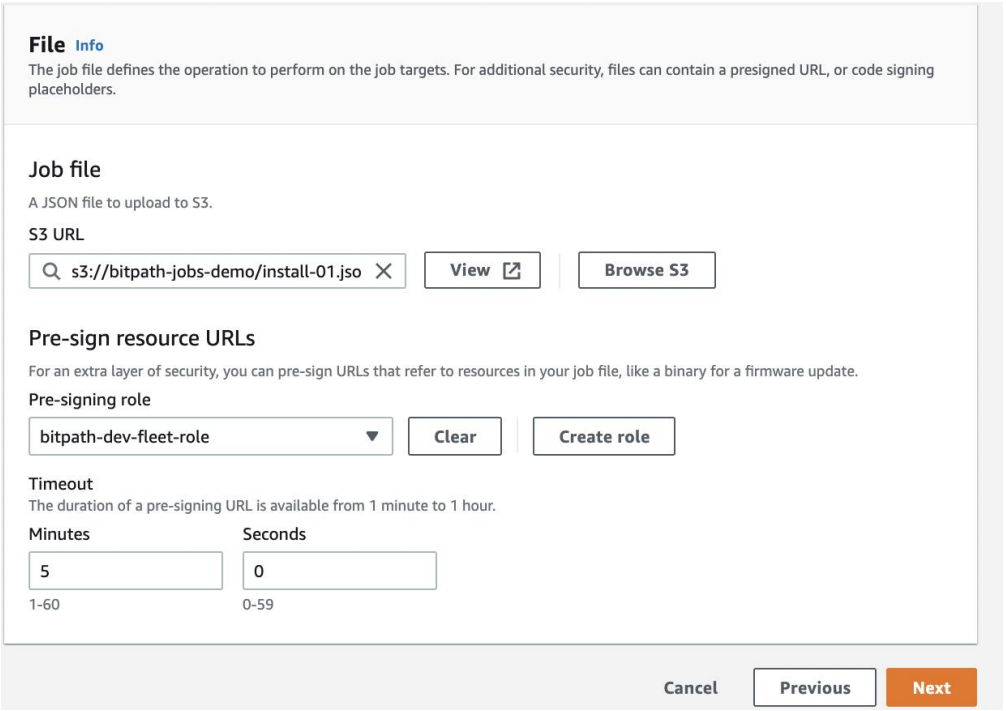
4. Create IoT job from AWS Console: Now, we will go back to the AWS IoT core portal and configure an IoT Job: Name of "*Job and description*" Click on Next



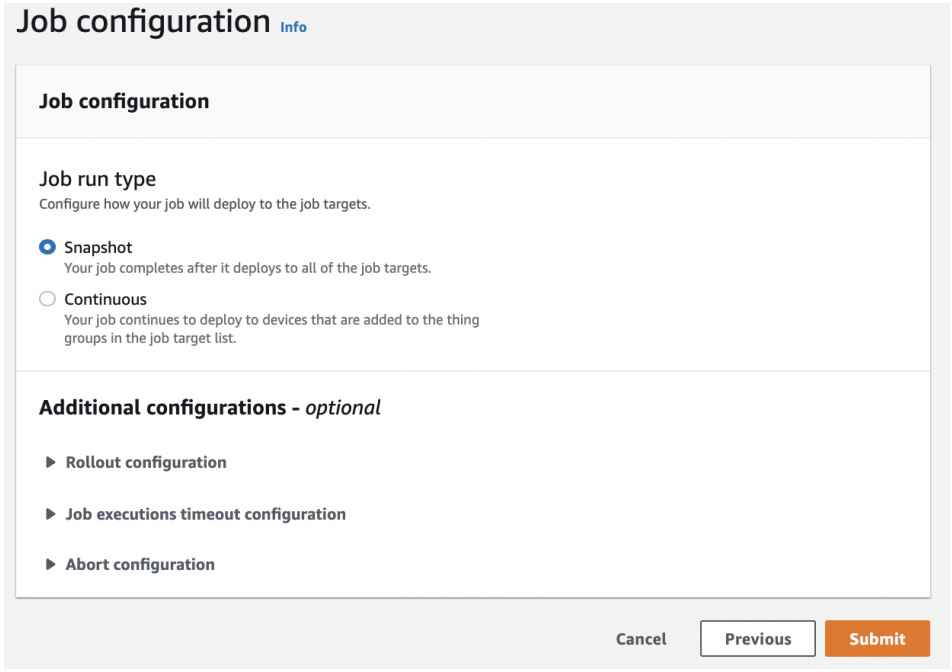
5. Deploy the jobs: Select the device where you want to execute the command; right now, I have only limited devices, so I have selected "*things*" to run this job. In case we have multiple things/device, we can use "*Thing Groups*" :



At the bottom, we must specify the JSON job file we created in step 2. We don't need to manually pre-sign the URL; it's automated as part of the AWS process. Ensure the IAM role is permitted to "GetObject from the S3 bucket you have created." Here, we are using RoleName - bitpath-dev-fleet-role. Enter the signed URL expiration time info.

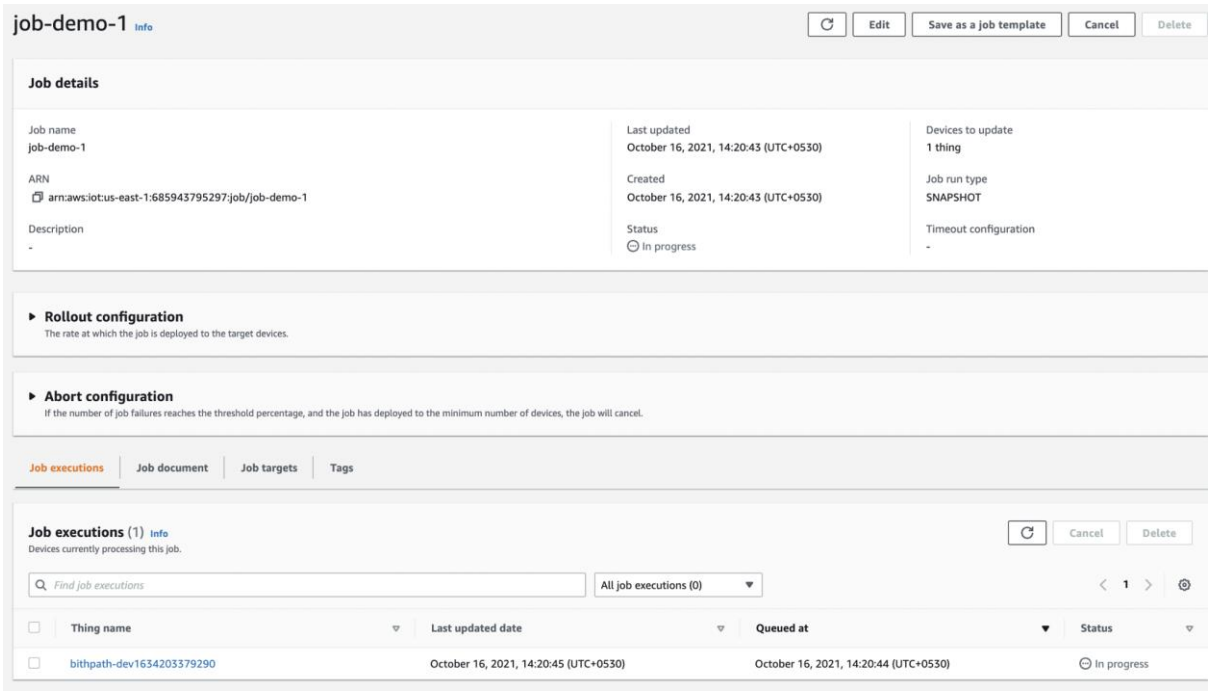


6. Click on next, and we will get options to execute these jobs; it could be one time, or it can be configured to run continuously after adding the new devices. I have selected "snapshot" and kept the default for only the rest. [AWS Documentation reference for Jobs]



7. Before clicking Submit, ensure the process runs on the device created in step 1.[Ignore this step, as it's automatically running as a daemon].

8. Continue step 6 and submit the Job from the AWS portal - The Status will turn “in progress”.

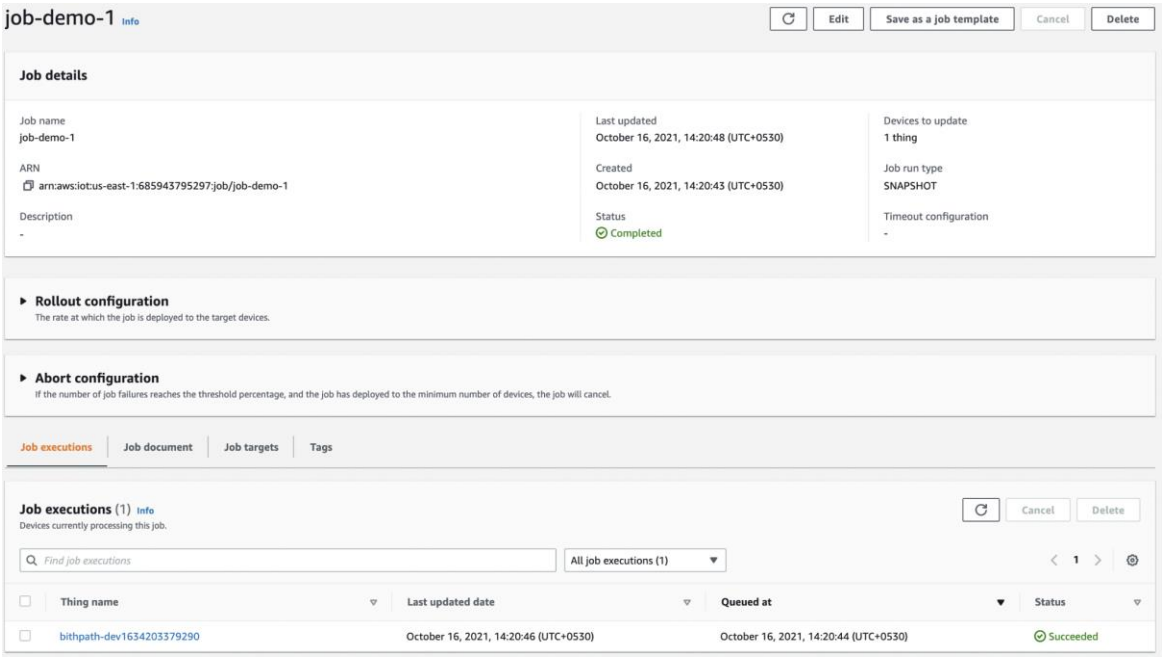



```
root@bitpath-dev-xx:/opt/jobs-testing# python3 joblistener2.py
Client Connected
Device Listening Started
Job Completed !
```

Verify your device terminal that the command has executed successfully, and you will be able to view the system time and the message as per the testconfig.py

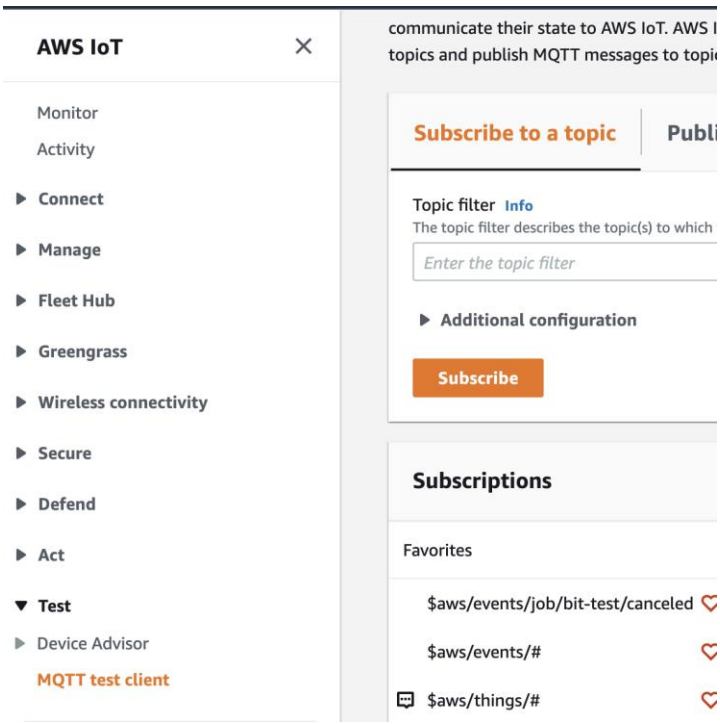
```
root@bitpath-dev-xx:/opt/jobs-testing# python3 joblistener2.py
Client Connected
Device Listening Started
Job Completed !
Job Execution Started
 09:50:46 up 1 day, 15:18,  1 user,  load average: 0.02, 0.08, 0.08
A Sample file. Downloaded to Device and Executed !
```

Job-status also has switched from in progress to complete once completed.



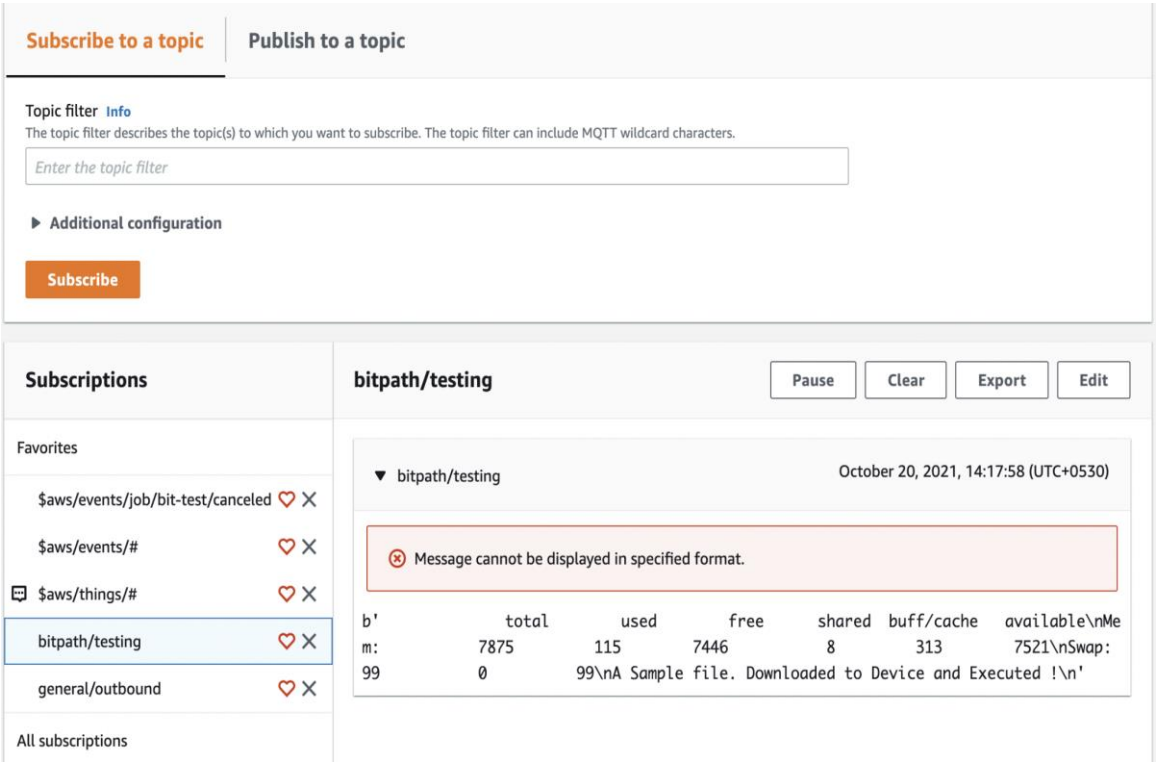
5.3 To view the execution output

To check the Inbound/Outbound request, please subscribe below topics from the Device



5.4 Subscribe to the below topics

1. Subscribe to a topic
2. Subscription topic: $aws/events/#
3. Subscribe
4. Subscribe to a topic
5. Subscription topic: $aws/things/#
6. Subscribe
7. Subscribe to a topic
8. Subscription topic:bitpath/testing
9. Subscribe to topic: #: In case no topic is selected to view all the events.



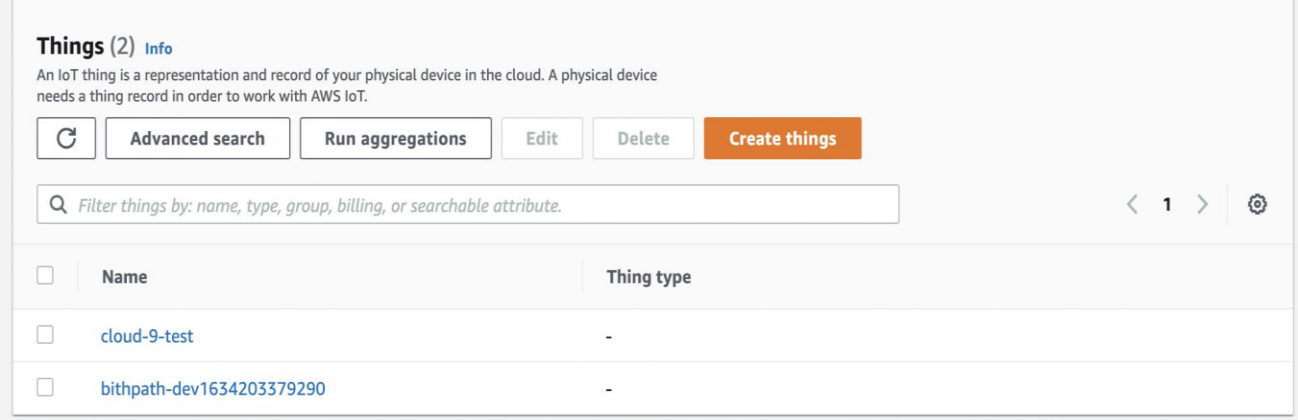
6 Running IoT Jobs on multiple devices

Starting the listener scripts in both devices [In our golden image, it's automated. We can ignore this step]:

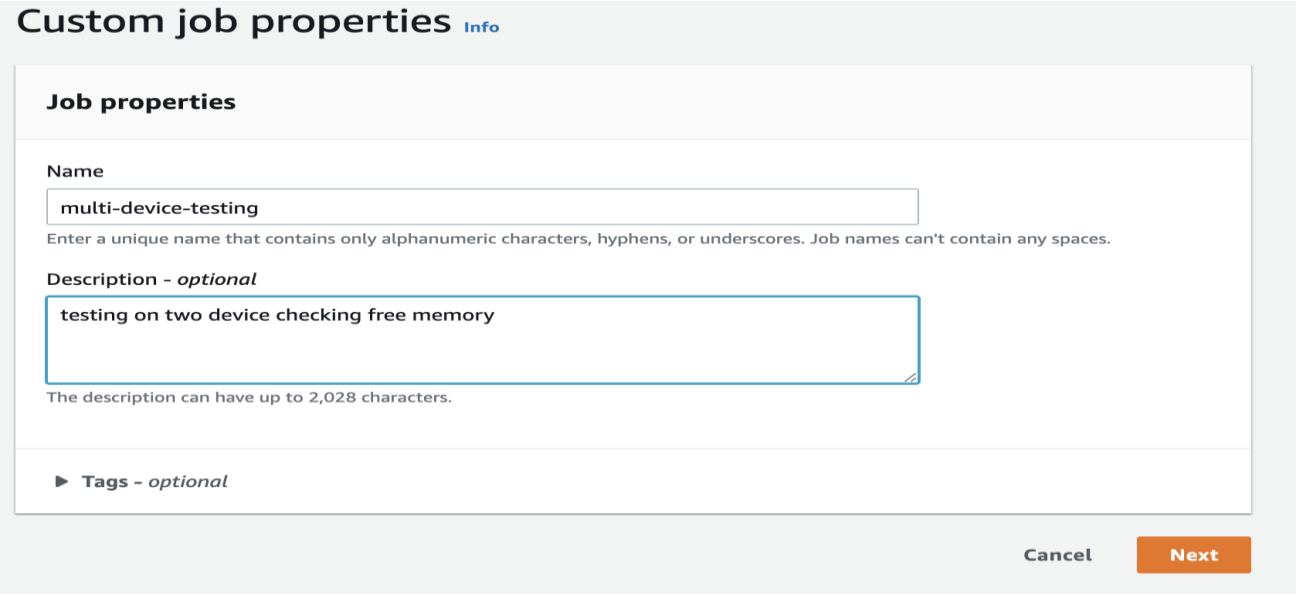
```
root@bitpath-dev-xx:/opt/jobs-testing# python3 joblistener4.py
bithpath-dev1634203379290
Client Connected
Device Listening Started
```

```
root@ip-172-31-71-169:/opt/jobs-testing# python3 joblisterner1.py
/usr/lib/python3/dist-packages/requests/__init__.py:80: RequestsDependencyWarning: urllib3 (1.26.7) or chardet (3.0.4) doesn't match a supported version!
  RequestsDependencyWarning)
Client Connected
Device Listening Started
Job Completed !
```

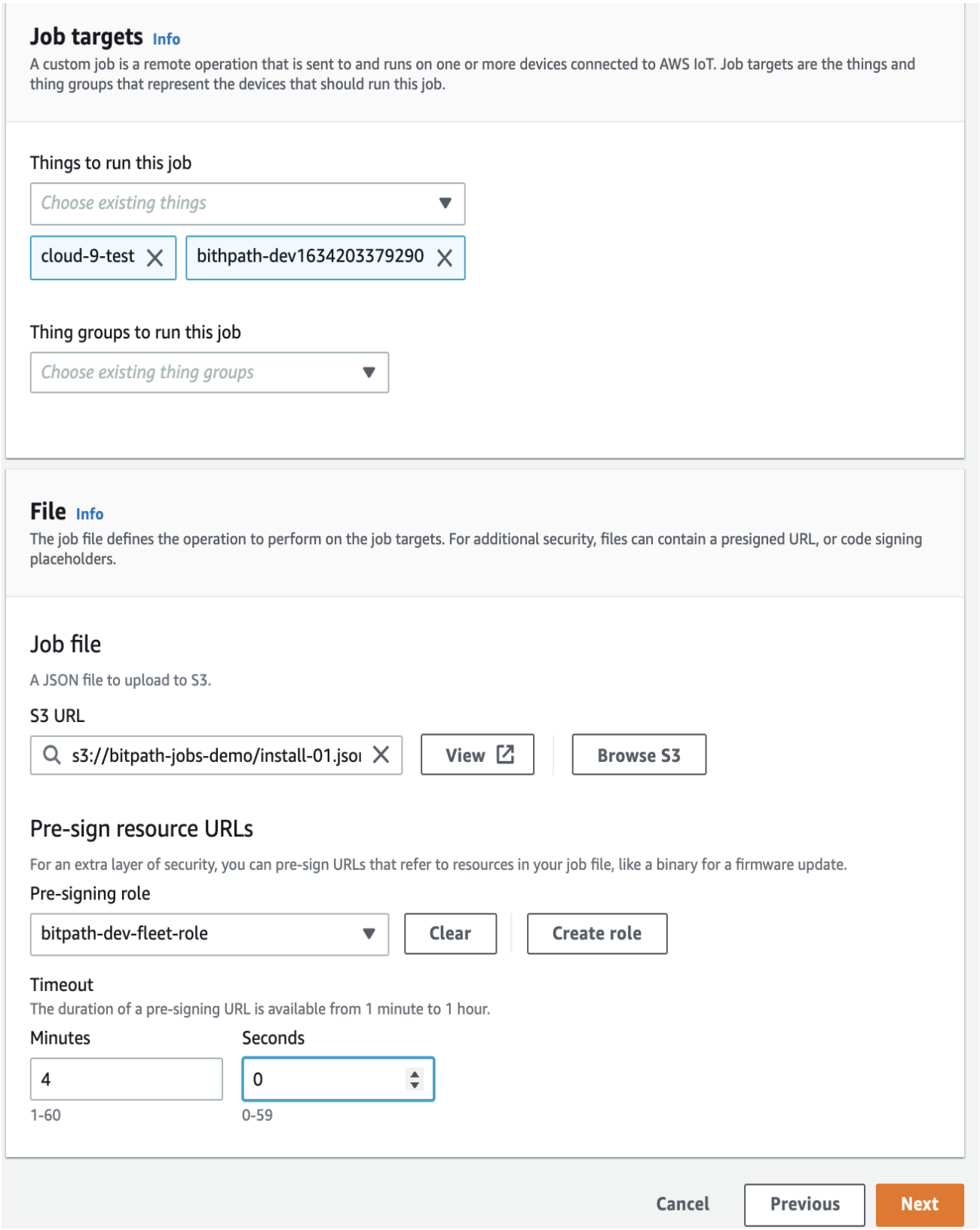
From the AWS IoT core Manage Jobs, we will trigger jobs to both devices:



For the same steps, click on Create Job and enter the required details.



6.1 Select the IoT devices

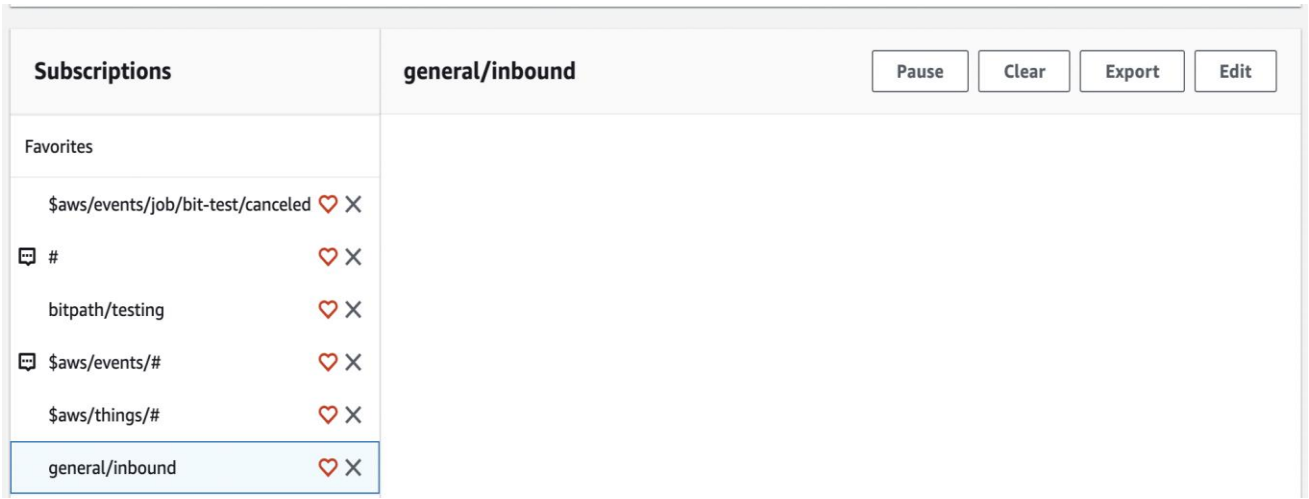


6.2 Publishing message to topics

As discussed above, we must subscribe to topics to view the logs. On job completion, we can view the output to the respective subscribed topics:

Here, we have subscribed to the following topics:

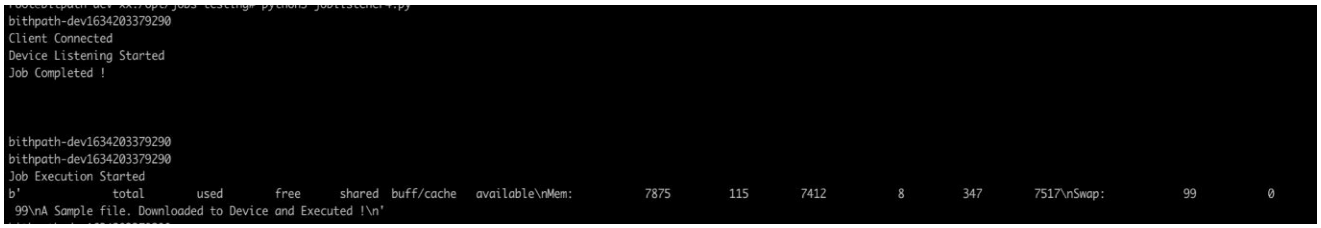
1. bitpath/testing
2. general/inbound



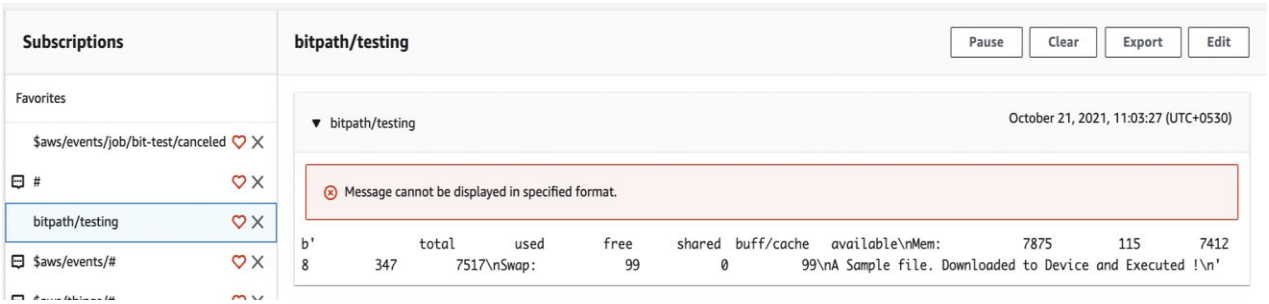
6.3 Post-execution of the jobs

Cloud-9-test: Output from device:

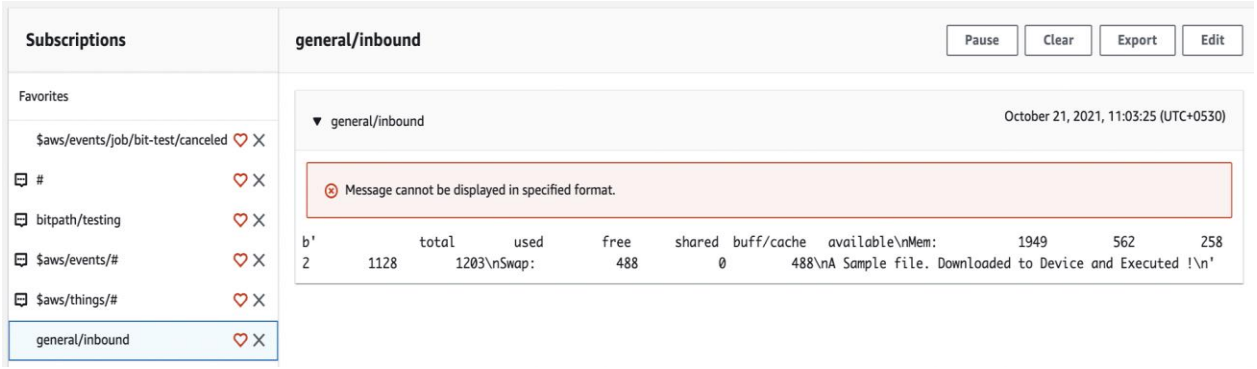
Bithpath-dev1634203379290 output from the device:



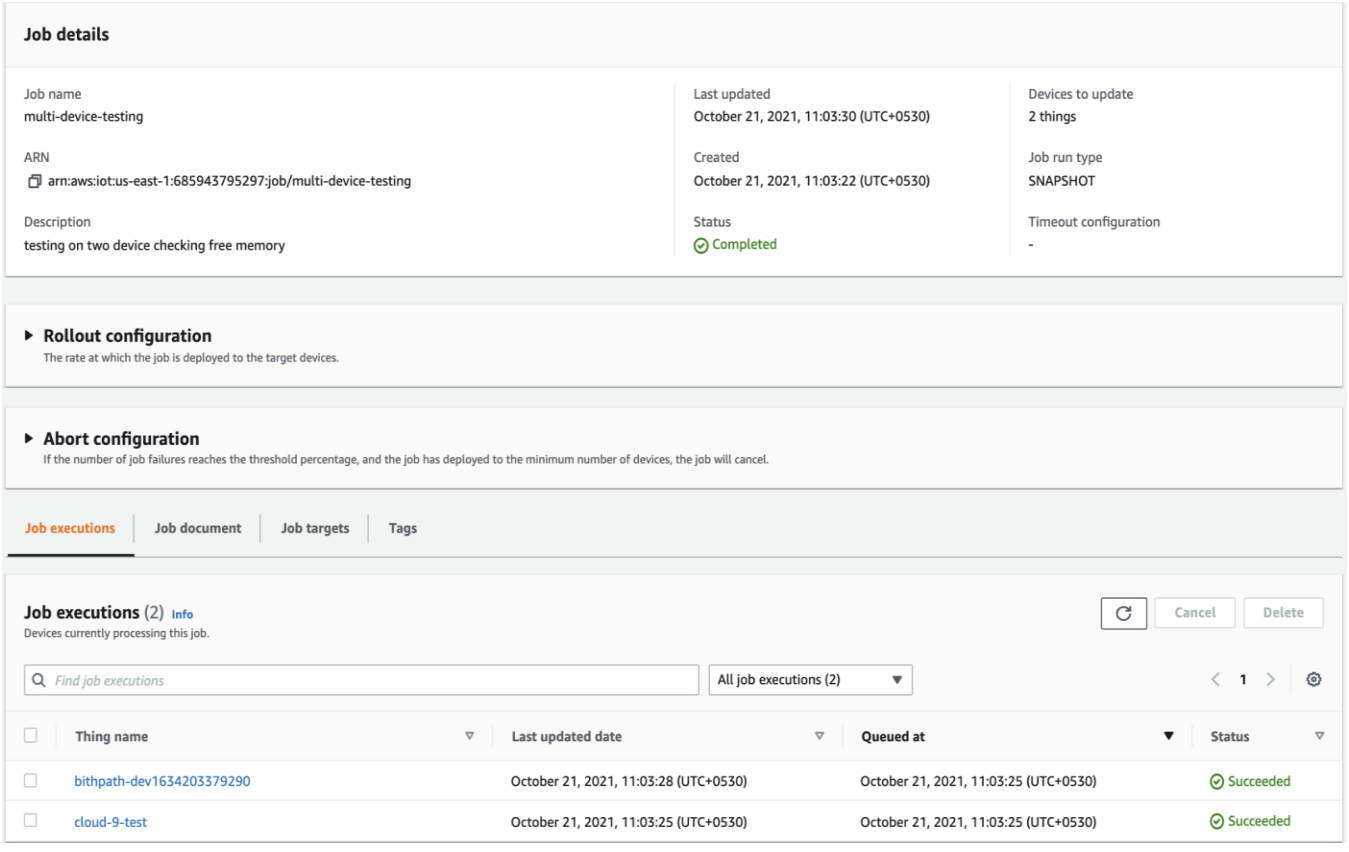
Same output from the IoT subscription:
bitpath/testing



general/inbound



6.4 Job Status



7 Executing the subscriber program as a daemon

Multiple ways to execute the jobs in the backend, like crontab and .bashrc. In this case, we have used the default OS service, **systemd.**

Systemd should be installed on all the machines, in case if it's missing, install it using the below command :

sudo apt-get install -y systemd

Once, installed create a file under : /etc/systemd/system/<NameOftheService>

```
-rw-r--r-- 1 root root  128 Oct 23 05:19 iot-services.service
root@bitpath-dev-01:/etc/systemd/system#
```

We have kept it as iot-services.service.

With very minimal configuration, to start the service

```
root@bitpath-dev-01:/etc/systemd/system# cat iot-services.service
[Unit]
Description=My IoT job
[Service]
Type=simple
Restart=always
ExecStart=/usr/bin/python3 /opt/jobs-testing/joblistener4.py
```

Once configured, reload the daemon with sudo systemctl daemon-reload and start the service with sudo systemctl start iot-services.service, which will start the listener service in the background.

8 Sample deployment steps

Deploying a sample Python code and installing dependencies on devices:

1. Cloning of a Sample Python code:

First, change the install.py as per the requirement[refer to step 3 code]; here, we are cloning a public repo on the device and installing the packages.

```
import os
clone1 = 'git clone https://github.com/stevemar/sample-python-app.git'
executer= 'cd sample-python-app && pip install -r requirements.txt'
os.system(clone1)
os.system(executer)

print("A Sample file. Downloaded to Device and Executed !")
```

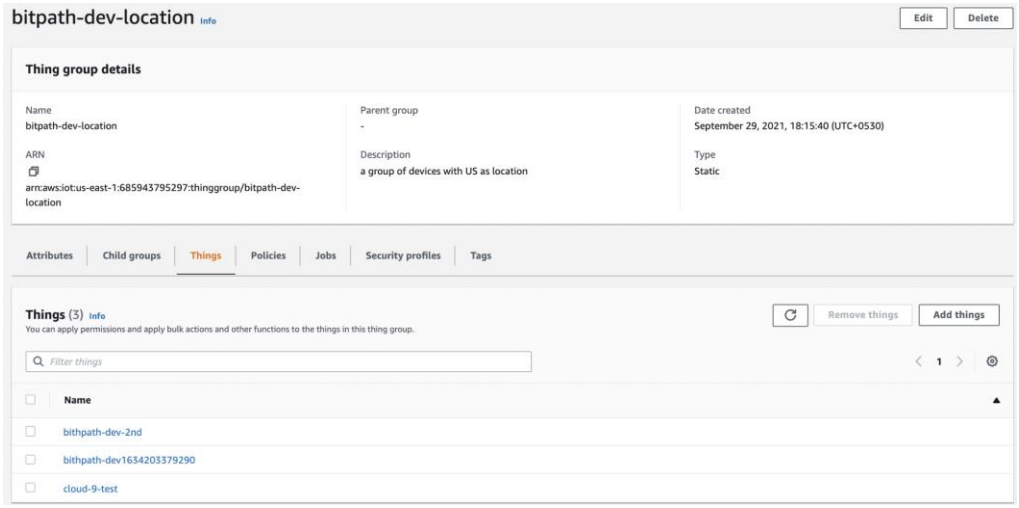
Once the modification is completed, save the file and upload the file in the AWS S3 bucket.

2. IoT job configuration file: The install-01.json script does not change, as the filename is the same.

```
{
  "operation" : "install",
  "version" : "1.0",
  "packageName" : "install.py",
  "autoStart" : "true",
  "workingDirectory" : "/opt/jobs-testing",
  "files" : {
     "fileName" : "install.py",
     "url" : "${aws:iot:s3-presigned-url:https://s3.us-east-1.amazonaws.com/bitpath-jobs-demo/install..py}"
  }
}
```

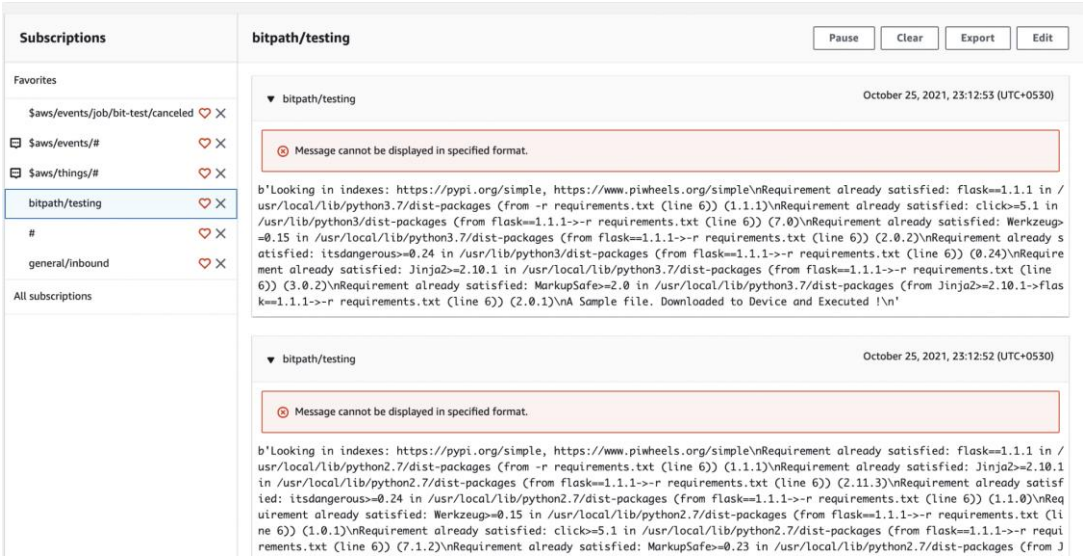
3. Deploying to a group :
    a. Bithpath-dev-2nd
    b. Cloud-9-test

As a prerequisite, we already run Python listener code as a background process on all devices. Now, we will create an IoT job from the AWS Console or CLI and execute it—refer to Epics: 4th point.



Output

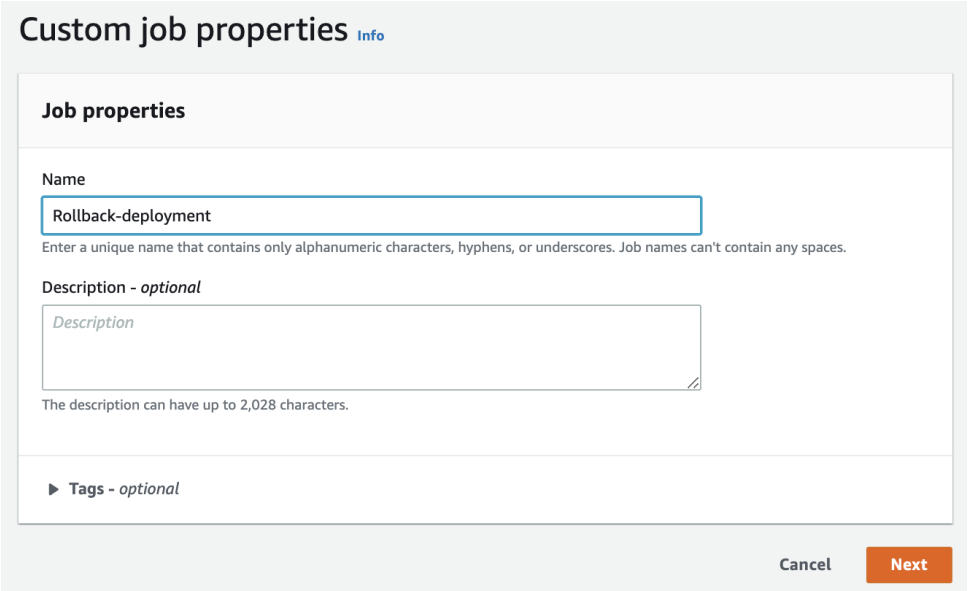
To view the Output, go to the MQTT test client:



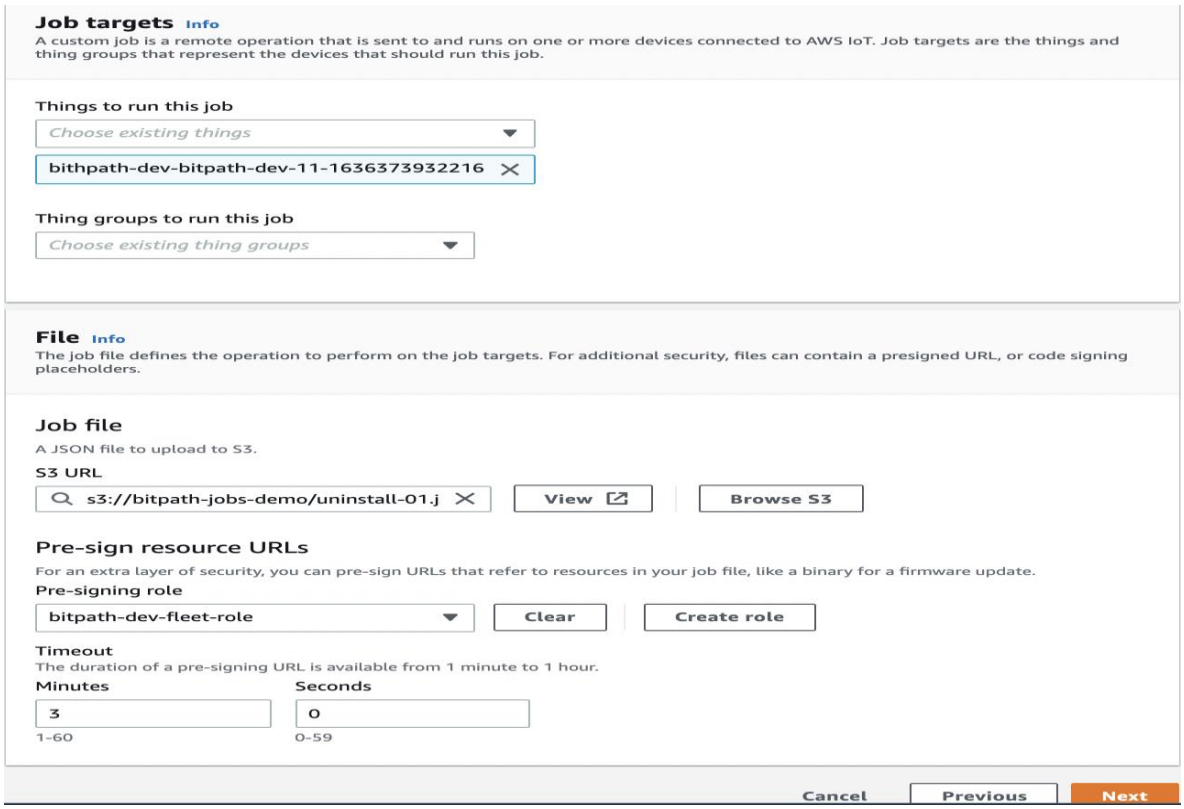
9 Rollback of the deployment

In case of deployment issues, we can roll back similarly.

1. Create a rollback Python script; here, we have created a sample rollback Python script—rollback.py [For this demo, no modification required] script in S3 and mentioned the commands that must be executed.
2. For uninstallation, we will use a different JSON file name as uninstall-01.json

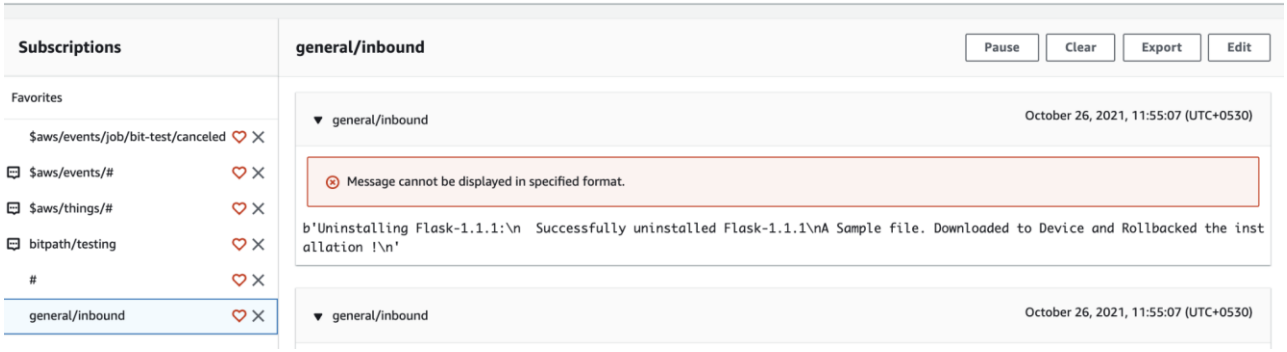


3. Select the device:



Rollback of the deployment successful :

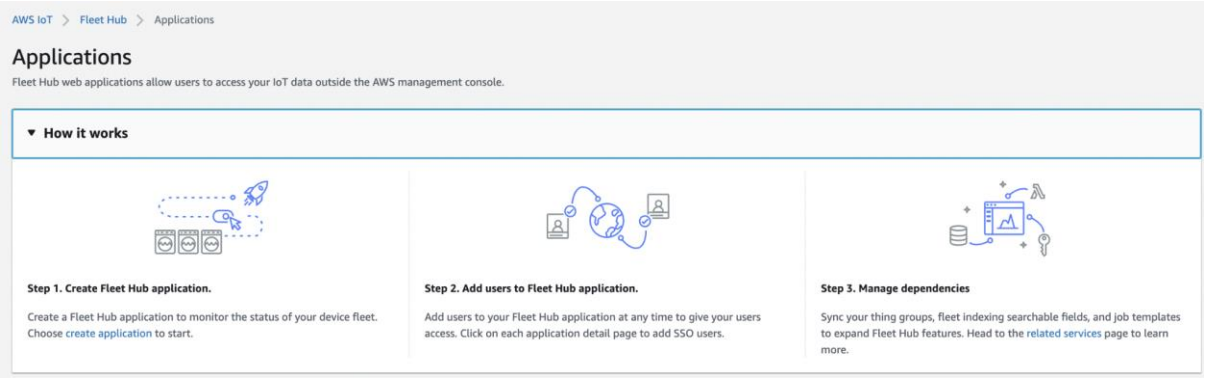
Outputs
Device1:

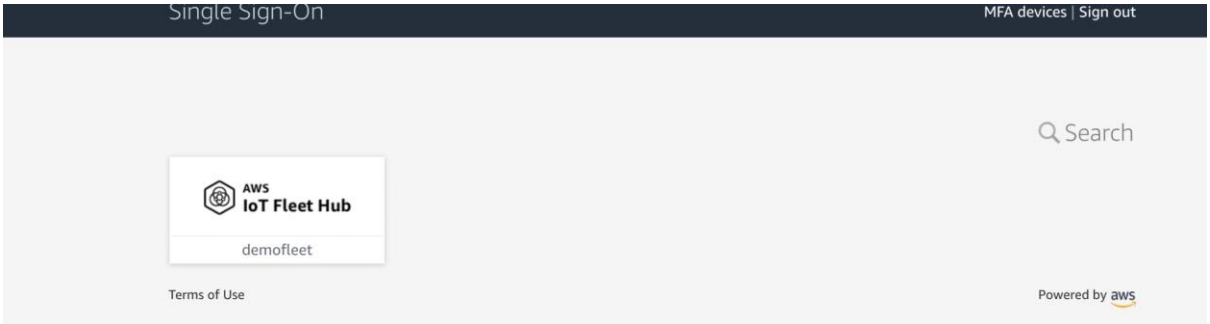


Device2:

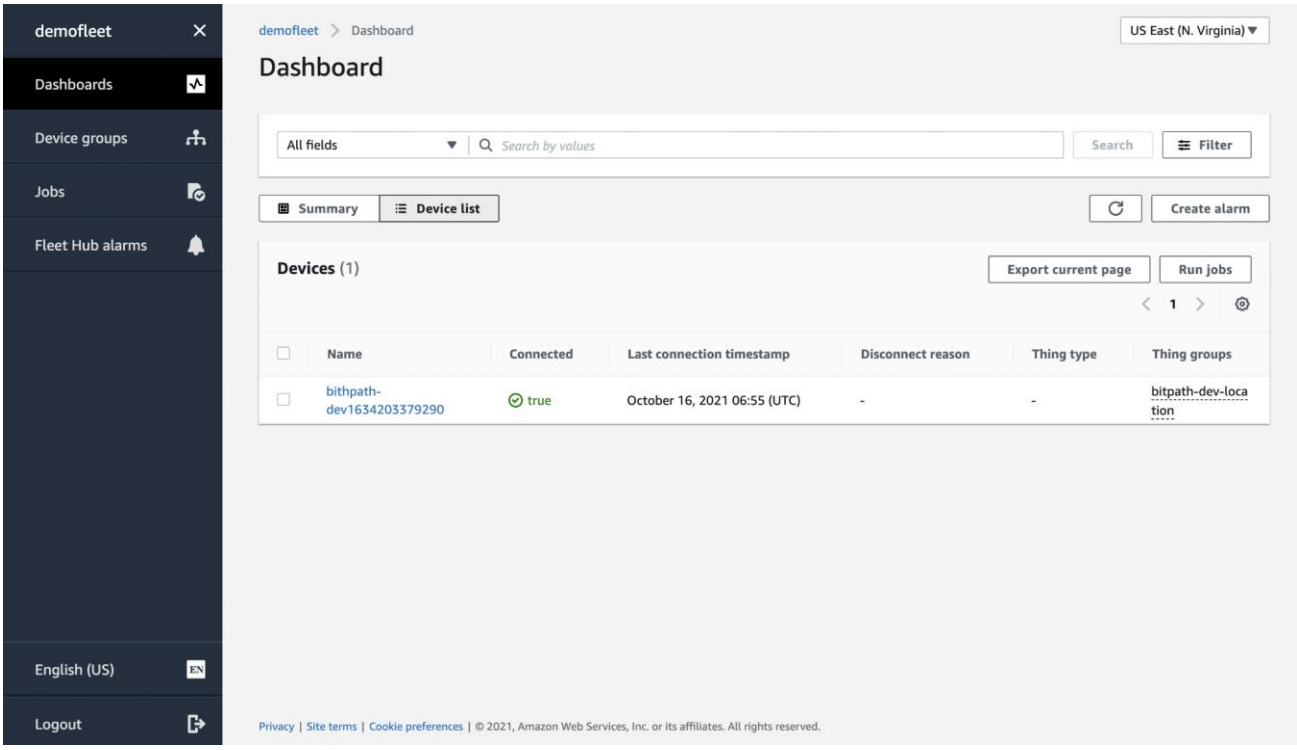
10 Fleet Hub

The diagram below shows the process of using the fleet hub.

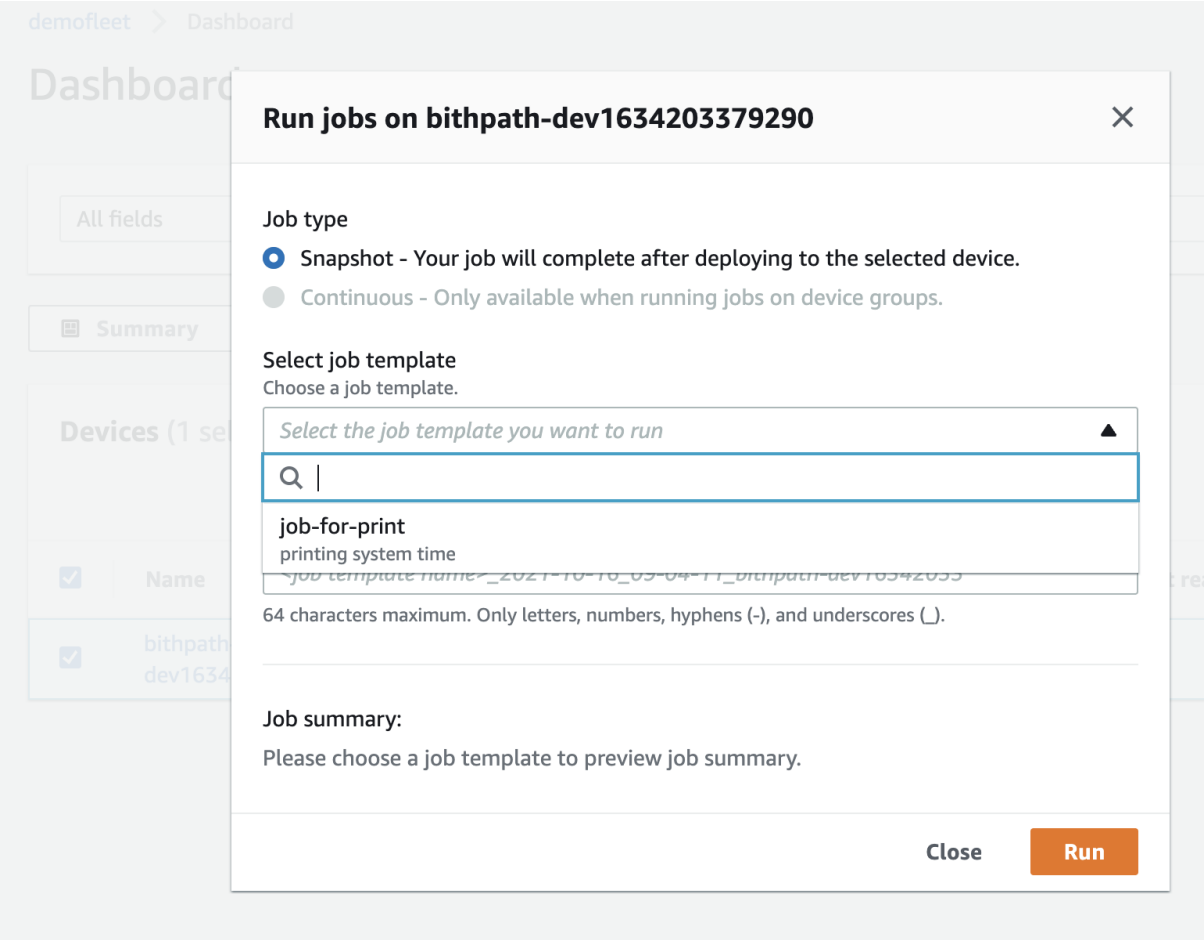


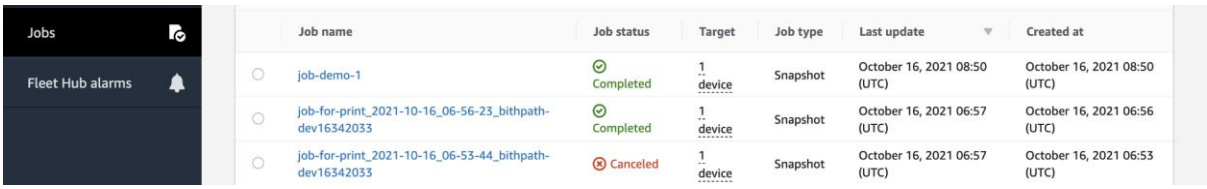


Once we have set up the application and added the SSO user, the user will get a signup request in his email box and a URL to manage all the IoT devices. This portal will have all the information, including a device list, job status, etc., and also give permissions to execute the above jobs that we have just executed.



Users can view the device list in the same portal and run jobs by selecting the device. [Note: to use jobs, the admin must create the job template first. Only then can the user choose and perform the operations.





11 References