BitPath-DevKit

Installed Tools and Configuration

RADIOSTUDIO

1 Table of Contents

2 Revision History

Version No	Date	Author	Change Log
1.0	29/04/2024	Gautham D	

3 OS-level tools installed

Below are the additional tools installed as part of the golden image.

S.N	Name of the Package
1	Build Essentials
2	Nmap
3	TCP Dump
4	Log rotate
5	IP tables
6	UFW
7	WV dial
8	Bsdmainutils
9	python3
10	git
11	Aws SDK for IoT Core
12	SQLite
13	OpenSSL

4 OS Level Custom Configurations

The golden image is customised with the additional configurations below at the OS level.

- Create a new user **bitpath**
- Disable pi user (comes with default user pi and password raspberry) sudo usermod --expire date 1 pi
- Disable wireless interface <<Added dtoverlay=disable-wifi /boot/config.txt>>
- Disable Bluetooth <<Added dtoverlay=disable-bt>>
- Disable root user SSH access
- Disable Password-based authentication and enable key-based authentication
- Change SSH to a nonstandard port (port 2022 is SSH port now)
- Enable Local firewall
- Install and configure
- Systemctl settings configured! init=/bin/systemd to your /boot/cmdline.txt

5 Software installed

The list below contains the tools that were installed during this project.

S.N	Name of the Tool
1	AWS Systems manager
2	Telegraf
3	Promtail
4	IoT Core Registration

6 Installation Guides for the tools

6.1 Installing Systems Manager Components

```
sudo mkdir /tmp/ssm && cd /tmp/ssm/

sudo curl https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/debian_arm/amazon-ssm-agent.deb -o /tmp/ssm/amazon-ssm-agent.deb

sudo dpkg -i /tmp/ssm/amazon-ssm-agent.deb

sudo service amazon-ssm-agent stop

## Be mindful of the quotes in the command below. Wrong quotes can sometimes have unintended consequences.

sudo amazon-ssm-agent -register -code "ooFmfHzQV7YDQwIIKigy" -id "f0228110-fd1c-4b9c-9ef0-e7bf327b63e3" -region "us-east-1"

sudo service amazon-ssm-agent start
```

6.2 Install Promtail

```
sudo cd /usr/local/bin

sudo wget https://github.com/grafana/loki/releases/download/v2.3.0/promtail-linux-arm.zip
sudo unzip promtail-linux-arm.zip && mv promtail-linux-arm promtail
sudo nano config-promtail.yml
#now paste the config-promtail.yml content to this file, don't include -----
-------------------------
server:
 http_listen_port: 9080
 grpc_listen_port: 0
positions:
 filename: /tmp/positions.yaml
clients:
 - url:
scrape_configs:
- job_name: system
 static_configs:
 - targets:
    - localhost
   labels:
    job: varlogs
    __path__: /var/log/*log

--------------------------
```

```
#test the working of the setup
sudo ./promtail -config.file ./config-promtail.yml

Close this, and let's run Promtail as a service.
```

6.2.1 How to run promtail as a Service

```
sudo nano /etc/systemd/system/promtail.service

#don't add ------ to the file
-----------------------------------------------
[Unit]
Description=Promtail service
After=network.target

[Service]
Type=simple
User=root
ExecStart=/usr/local/bin/promtail -config.file=/usr/local/bin/config-promtail.yml

[Install]
WantedBy=multi-user.target
-----------------------------------------------
sudo service promtail start
sudo service promtail status
systemctl enable promtail.service
```

6.3 Install And Configure Telegraf Agent

6.3.1 Install necessary binaries

```
sudo apt-get update && sudo apt-get install apt-transport-https
```

We are on bullseye for Raspberry 4 - so add the influx key

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -

source /etc/os-release

test $VERSION_ID = "10" && echo "deb https://repos.influxdata.com/debian buster stable" | sudo tee /etc/apt/sources.list.d/influxdb.list
```

6.3.2 Install and enable Telegraf Service

```
sudo apt-get update && sudo apt-get install telegraf && systemctl enable telegraf.service
```

Start the service once and check the status to see if it's running. There will be some errors initially because the default configuration will try to talk to a local influxdb but check if the service starts and then stop again.

```
systemctl start telegraf.service
systemctl status telegraf.service
# stop service after this.`
```

Rename the default config file and create a new telegraf.conf

```
mv /etc/telegraf/telegraf.conf /etc/telegraf/telegraf.conf.bak
vi /etc/telegraf/telegraf.conf
```

Use the content from the configuration file in this gist. You have to change the hostname and add an authentication key. Use the sample configuration, which is available in the TIG Stack document.

We will execute a custom script as part of the inputs, so create a file vcgencmd.sh under the new directory and copy the contents.

```
mkdir /var/lib/telegraf
vi /var/lib/telegraf/vcgencmd.sh
#!/bin/bash


host=$(cat /proc/sys/kernel/hostname)
data="vcgencmd,host=${host} "

# ------------------------------------------------------------------
#
# telegraf-pi-get-throttled.sh
#
# Get and parse RPi throttled state to produce a single Grok-ready string.

### Data Acquisition
# Poll the VideoCore mailbox using vcgencmd to get the throttle state, then use sed to grab just the hex digits
throttle_state_hex=$(/opt/vc/bin/vcgencmd get_throttled | sed -e 's/.*=0x\([0-9a-fA-F]*\)/\1/')

### Example vars for testing
## Debug: uv=1 uvb=1 afc=0 afcb=0 tr=1 trb=0 str=0 (null) strb=0 (null)
# throttle_state_hex=50005
## Debug: uv=0 uvb=1 afc=0 afcb=0 tr=0 trb=0 str=0 (null) strb=0 (null)
# throttle_state_hex=50000
## Debug: uv=1 uvb=1 afc=1 afcb=1 tr=1 trb=1 str=1 strb=1
# throttle_state_hex=F000F
## Debug: uv=1 uvb=1 afc=0 afcb=1 tr=0 trb=1 str=0 strb=1
# throttle_state_hex=F0008
## Debug: uv=0 uvb=1 afc=0 afcb=1 tr=0 trb=1 str=1 strb=1
# throttle_state_hex=F0001
```

```
### Command Examples
## dc: Convert hex to dec or binary (useful for get_throttled output)
# Example: dc -e 16i2o50005p
# Syntax: dc -e <base_in>i<base_out>o<input>p
#   i = push base_in (previous var) to stack
#   o = push base_out (previous var) to stack
#   p = print conversion output with newline to stdout

get_base_conversion () {
  args=${1}i${2}o${3}p
  binary=$(dc -e $args)
  printf %020d $binary
}

throttle_state_bin=$(get_base_conversion 16 2 $throttle_state_hex)

binpattern="[0-1]{20}"
if [[ $throttle_state_bin =~ $binpattern ]]; then
## Reference: get_throttled bit flags
  #
https://github.com/raspberrypi/firmware/commit/404dfef3b364b4533f70659eafdcefa3b68cd7ae#commitc
omment-31620480
  #
  # NOTE: These ref numbers are reversed compared to the vcencmd output.
  #
  # Since Boot         Now
  #  |                 |
  # 0101 0000 0000 0000 0101
  # ||||              ||||_ [19] throttled
  # ||||              |||_ [18] arm frequency capped
  # ||||              ||_ [17] under-voltage
  # ||||              |_ [16] soft temperature capped
  # ||||_ [3] throttling has occurred since the last reboot
  # |||_ [2] arm frequency capped since the last reboot
  # ||_ [1] under-voltage has occurred since the last reboot
  # |_ [0] soft temperature reached since last reboot

  strb=${throttle_state_bin:0:1}
  uvb=${throttle_state_bin:1:1}
  afcb=${throttle_state_bin:2:1}
  trb=${throttle_state_bin:3:1}
  str=${throttle_state_bin:16:1}
  uv=${throttle_state_bin:17:1}
  afc=${throttle_state_bin:18:1}
  tr=${throttle_state_bin:19:1}

  # Note: first field, do not start with a,
  data+="under_volted=${uv:-0}i,under_volted_boot=${uvb:-0}i,arm_freq_capped=${afc:-
```

```
0}i,arm_freq_capped_boot=${afcb:-0}i,throttled=${tr:-0}i,throttled_boot=${trb:-0}i,soft_temp_limit=${str:-
0}i,soft_temp_limit_boot=${strb:-0}i"
fi
# ------------------------------------------------------------

# SOC temp
soc_temp=$(/opt/vc/bin/vcgencmd measure_temp | sed -e "s/temp=//" -e "s/'C//")
data+=",soc_temp=${soc_temp}"

# Clock speeds
arm_f=$(/opt/vc/bin/vcgencmd measure_clock arm | sed -e "s/^.*=//")
data+=",arm_freq=${arm_f}i"
core_f=$(/opt/vc/bin/vcgencmd measure_clock core | sed -e "s/^.*=//")
data+=",core_freq=${core_f}i"
#h264_f=$(/opt/vc/bin/vcgencmd measure_clock h264 | sed -e "s/^.*=//")
#data+=",h264_freq=${h264_f}i"
#isp_f=$(/opt/vc/bin/vcgencmd measure_clock isp | sed -e "s/^.*=//")
#data+=",isp_freq=${isp_f}i"
#v3d_f=$(/opt/vc/bin/vcgencmd measure_clock v3d | sed -e "s/^.*=//")
#data+=",v3d_freq=${v3d_f}i"
uart_f=$(/opt/vc/bin/vcgencmd measure_clock uart | sed -e "s/^.*=//")
data+=",uart_freq=${uart_f}i"
#pwm_f=$(/opt/vc/bin/vcgencmd measure_clock pwm | sed -e "s/^.*=//")
#data+=",pwm_freq=${pwm_f}i"
#emmc_f=$(/opt/vc/bin/vcgencmd measure_clock emmc | sed -e "s/^.*=//")
#data+=",emmc_freq=${emmc_f}i"
#pixel_f=$(/opt/vc/bin/vcgencmd measure_clock pixel | sed -e "s/^.*=//")
#data+=",pixel_freq=${pixel_f}i"
#vec_f=$(/opt/vc/bin/vcgencmd measure_clock vec | sed -e "s/^.*=//")
#data+=",vec_freq=${vec_f}i"
#hdmi_f=$(/opt/vc/bin/vcgencmd measure_clock hdmi | sed -e "s/^.*=//")
#data+=",hdmi_freq=${hdmi_f}i"
#dpi_f=$(/opt/vc/bin/vcgencmd measure_clock dpi | sed -e "s/^.*=//")
#data+=",dpi_freq=${dpi_f}i"

# Voltages
core_v=$(/opt/vc/bin/vcgencmd measure_volts core | sed -e "s/volt=//" -e "s/0*V//")
data+=",core_volt=${core_v}"
#sdram_c_v=$(/opt/vc/bin/vcgencmd measure_volts sdram_c | sed -e "s/volt=//" -e "s/0*V//")
#data+=",sdram_c_volt=${sdram_c_v}"
#sdram_i_v=$(/opt/vc/bin/vcgencmd measure_volts sdram_i | sed -e "s/volt=//" -e "s/0*V//")
#data+=",sdram_i_volt=${sdram_i_v}"
#sdram_p_v=$(/opt/vc/bin/vcgencmd measure_volts sdram_p | sed -e "s/volt=//" -e "s/0*V//")
#data+=",sdram_p_volt=${sdram_p_v}"



time_stamp=$(date +%s%N)
data+=" ${time_stamp}"
```

```
echo "${data}"
```

Make it executable and test whether this works with the telegraf user:

```
chmod +x /var/lib/telegraf/vcgencmd.sh
sudo -u telegraf /var/lib/telegraf/vcgencmd.sh
```

We also need to gather the GPU-related information, so let's add the telegraf user to the video group.

```
sudo usermod -G video telegraf
```

Start the service now, and it should start shipping data to InfluxDB.

```
systemctl start telegraf.service
```

6.4 Install & Configure the AWS IOT Core Fleet Management

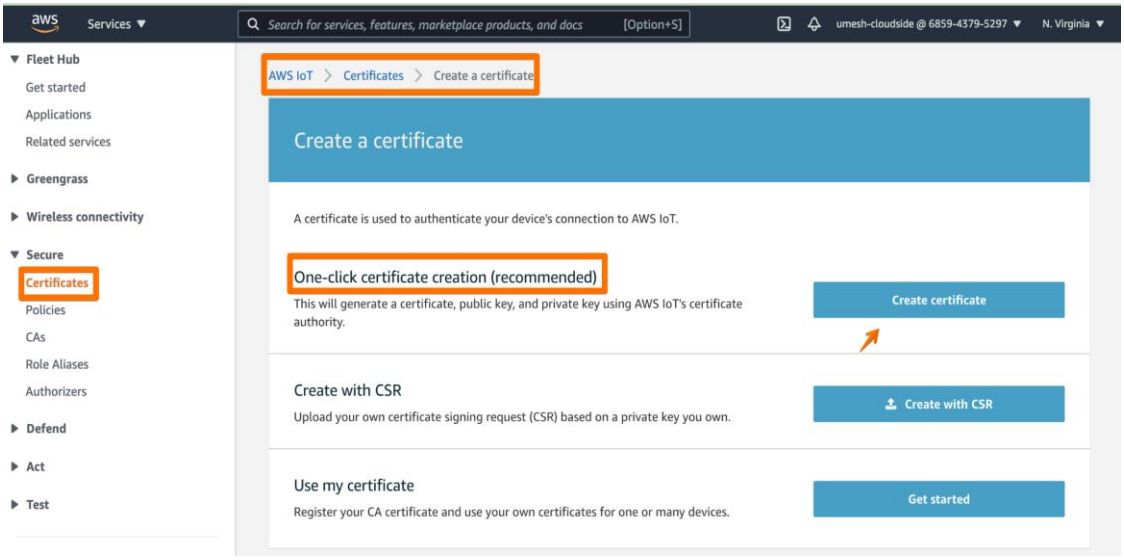
Resources

1. **https://aws.amazon.com/blogs/iot/how-to-automate-onboarding-of-iot-devices-to-aws-iot-core-at-scale-with-fleet-provisioning/**
2. **https://github.com/aws-samples/aws-iot-fleet-provisioning**

Approach

- Create the temp certificates to be used for all devices
- Move the certificates to your rasp device and store them at some location (in our case, /opt/certificates)
- Clone the aws-iot-fleet-provisioning template and configure the config.ini file with the correct details around the certificate path, IOT Core endpoint, and fleet provisioning template name.
- Run Python3 main.py to register the device with the aws iot core
- This will create and store a new certificate in your certification path folder. Your device will be registered to the IOT core, and a thing will be made.

Create the temp certificates & move them to the device

- Go to *IOT core,* and inside *Secure,* you will find ***certificates***
  - 

- Download the certificates and keep them secure.
- Assign a policy to the certificates (need to fill in more details here)
- Now, move these certificates to the rasp device. Use ***cat*** or scp for transfer.
- You can store them in one location like /opt/certificates
- We will reference them now while configuring the IOT core part.

Using aws-iot-fleet-provisioning template

```
# let's clone the fleet provisioning template
git clone https://github.com/aws-samples/aws-iot-fleet-provisioning

cd aws-iot-fleet-provisioning/

# let's update the certificate path, names, IOT_ENDPOINT, and PRODUCTION_TEMPLATE names.
nano config.ini

#once you have updated, let's run the main.py
pip3 install -r requirements.txt
Python3 main.py

# This should now install the new certificate and register this device as a thing in the AWS IOT core.
```

Startup Script

- Change the hostname
- Dev registration script with IOT core (restart the daemon)
- IOT Jobs (Sub) start the service using systemctl

7 References