September 27, 2019

DYNAMIC ONBOARDING AND MANAGEMENT OF SOPHISTICATED EDGE CLUSTERS FOR INDUSTRIAL INTERNET OF THINGS (IoT) APPLICATIONS

Suresh Sankaran

Umang Tandon

DYNAMIC ONBOARDING AND MANAGEMENT OF SOPHISTICATED EDGE CLUSTERS FOR INDUSTRIAL INTERNET OF THINGS (IoT) APPLICATIONS

AUTHORS:

Suresh Sankaran
Umang Tandon

ABSTRACT

Certain conventional compute solutions are restricted to resources present in the networking hardware (e.g., routers/switches). However, increasing numbers of edge compute workloads are demanding special requirements with Graphical processing unit (GPU)/tensor processing unit (TPU) support, various in-built Internet of Things (IoT) protocol requirements, high system resources, *etc*. The techniques presented herein address the market of those specialized edge compute workloads while also extending IoT container orchestration to off-the-shelf devices.

DETAILED DESCRIPTION

Industrial IoT edge compute workloads are continuously demanding sophisticated applications with specific architectures, artificial intelligence (AI)/ machine learning (ML) services, compute resources (cpu/memory/disk), protocols, *etc*. For example, a predictive edge analytics use-case can be efficiently solved with a machine learning model based inference engine which needs AI/ML services and tighter integration with GPU/TPUs for accelerated computation. Existing Industrial IoT hardware does not meet these needs and there may be low incentives for vendors to provide those special resource requirements in generic network elements, such as like routers/switches.

There is off-the-shelf specialized hardware that satisfies the requirements of special edge compute workloads. Proposed herein is to extend a software framework, referred to as an IoT management framework, to onboard, deploy and manage applications on such off-the-shelf specialized hardware (white box devices). With these techniques, a customer/developer can have the flexibility to choose any off-the-shelf hardware that

satisfies his/her resource consumption, potentially with zero touch onboarding of the white box devices, while maintaining a single pane of glass view for managing applications. That is, the IoT management framework enables a customer to just plug in the white-box device to a router/switch enabled with the software framework, then the IoT management framework will create a local edge cluster where applications can deployed and managed.

In an example workflow:

1. The customer first selects the off-the-shelf compute platform (white box device) to host their container application. This compute platform is expected to run link layer discovery protocol (LLDP) and a container engine (e.g., listening on Transmission Control Protocol socket).
2. The customer plugs the off-the-shelf compute platform into an IoT management framework enabled router/switch. The IoT management framework may be an in-built feature of the router/switch.
3. The IoT management framework will also discover the off-the-shelf compute platform using the LLDP protocol. As an outcome of this LLDP discovery, the IoT management framework will know the off-the-shelf device details such as IP address, system unique identifier, MAC address, Interface name, and hostname.
4. The IoT management framework will establish a keep-alive session with the container engine on the off-the-shelf compute platform and prepare to orchestrate container applications on the off-the-shelf platform.
5. The IoT management framework can be managed from a Single-Device controller (Local Manager) or from Scale controllers. With any of these controllers, the customer can select the off-the-shelf device discovered by the IoT management framework and deploy container applications and manage the state of the application.

Further details of this workflow implementation are provided below.

Step 1: Select off-the-shelf (whitebox) compute platform

The customer can select any white box compute platforms assuming several software requirements are satisfied by the compute platform. For example, the compute platform needs to run the LLDP service, which sends out the device's metadata to directly connected neighbor network elements. By default, the LLDP service sends out metadata of the device, such as the IP address, system unique identifier, MAC address, Interface name and hostname. If there are multiple interfaces on the whitebox compute platform, then the LLDP service needs to be configured to send out the metadata via the ethernet network interface that will be used to connect to the IoT management framework enabled router/switch.

In addition, the compute platform needs to run a container engine service, by default, and be able to run containers. Container applications be able to communicate via a supported serial device, USB device or any desired peripherals or desired protocols for a hosted customer application. Moreover, the container engine listens on the TCP web socket at port 2376 (default is unix socket) so that the remote container client (the IoT management framework in this case) will be able to communicate to this container engine and spawn container applications.

Step 2: Connect off-the-shelf platform to the IoT management framework enabled router/switch

The customer will connect the whitebox compute platform to the IoT management framework enabled router/switch's front panel data interface. The LLDP service needs to be enabled on this router/switch, so that IOS can discover nearby LLDP nodes. Effectively when the system runs "show lldp neighbor details", the system should be able to show directly connected off-the-shelf device and its metadata.

Today IoT routers and switches prepackages the IoT management framework as part of the router/switch image, mainly used to setup Day0 configurations of the router/switch. There is an application container with a special python module to interact with the IOS CLI interface and execute any of IOS CLI SHOW or CONFIG commands. The techniques presented herein propose to run a long-running python script inside this

shell to execute the IOS CLI command "show lldp neighbor details" every few seconds and write the output to /bootflash/lldp_nodes file of the router/switch. The application has access to bootflash dir from inside the container.

Step 3: The IoT management framework onboards off-the-shelf compute platform

The IoT management framework runs in binos linux userspace of an IoT management framework enabled router/switch, also has access to /bootflash dir. The techniques presented herein extend the IoT management framework capability to onboard LLDP nodes. This is done by the IoT management framework reading the LLDP metadata file /bootflash/lldp_nodes and updating internal datastructure objects. As such, the IoT management framework now knows metadata, such as the IP address, system unique identifier, MAC address, Interface name and hostname of each directly connected LLDP nodes. At this point, a customer using a controller now will be able to see off-the-shelf LLDP nodes listed along with an IoT management framework enabled router/switch to orchestrate the IoT management framework container application. An example interface is shown below in FIG. 1.

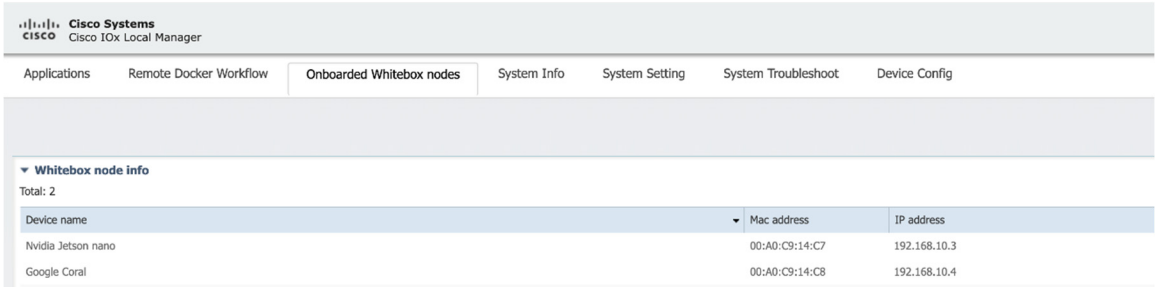

FIG. 1[JP1]

Step 4: The IoT management framework establish keep-alive session with off-the-shelf compute platform

The IoT management framework will use python container client to establish a session with the remote container engine on LLDP nodes and to retrieve container service

information and other primitives of container system like container images, containers, volumes, network *etc*.

Step 5: Orchestrate container app on off-the-shelf compute platform.

The customer can use one of the controllers to select any onboarded off-the-shelf LLDP device and install their container application. This application will be first pushed to IoT management framework enabled router/switch that is directly connected to the LLDP device. The IoT management framework on that IoT management framework enabled router/switch will then relay the container app to selected LLDP node's container engine and create containers using a remote container API /create URL. The IoT management framework will start monitoring the state of this container application and report the status back to the controller(s). This is shown below in FIG. 2

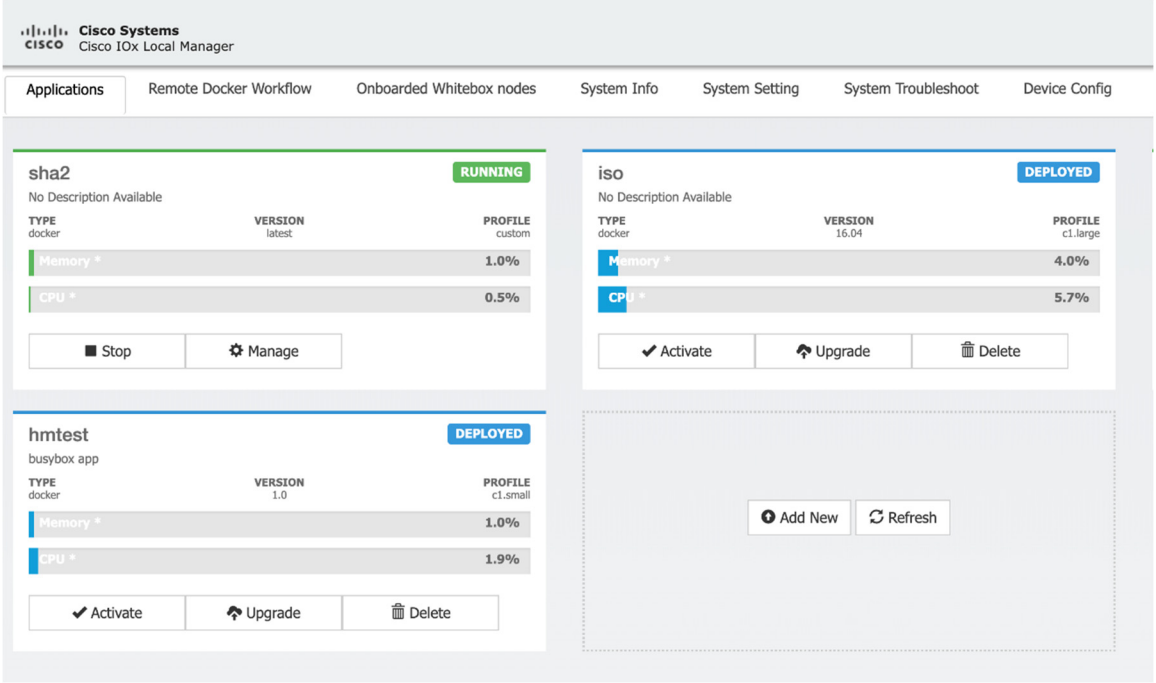

FIG. 2[JP2]

Certain container management systems are inherently not designed to operate with an IoT management framework enabled router/switch as it needs mesh connectivity between each node and pod in the cluster. The IoT management framework proposed

herein solves this problem by having 1:1 pull based controller connectivity to the on-boarded nodes. The techniques presented herein create a local cluster tied to a IoT management framework enabled router/switch instead of a distributed cluster. With a local cluster there is no connectivity issues and the IoT management framework can securely manage the container applications on off-the-shelf compute devices.

Current implementations are lacking due to constraint of managing container applications only on certain vendor router/switches that will not be able to adapt to the various specialized requirements of customer applications like GPU/TPU support, Bluetooth/Zigbee protocol capability, OPCUA support and high CPU/memory/disk requirements to container applications. With the techniques presented herein, a customer is free to choose the off-the-shelf compute blade with all their requirements.