Smart Data Transaction for CPS Installation Guide

Introduction

This guide provides instructions for installing and configuring the Smart Data Transaction for CPS blueprint, and also includes recommended hardware and software requirements for the blueprint. The guide describes a minimal installation of the blueprint consisting of a single "master" node and two "edge" nodes, with directions on how the number of nodes can be modified as needed.

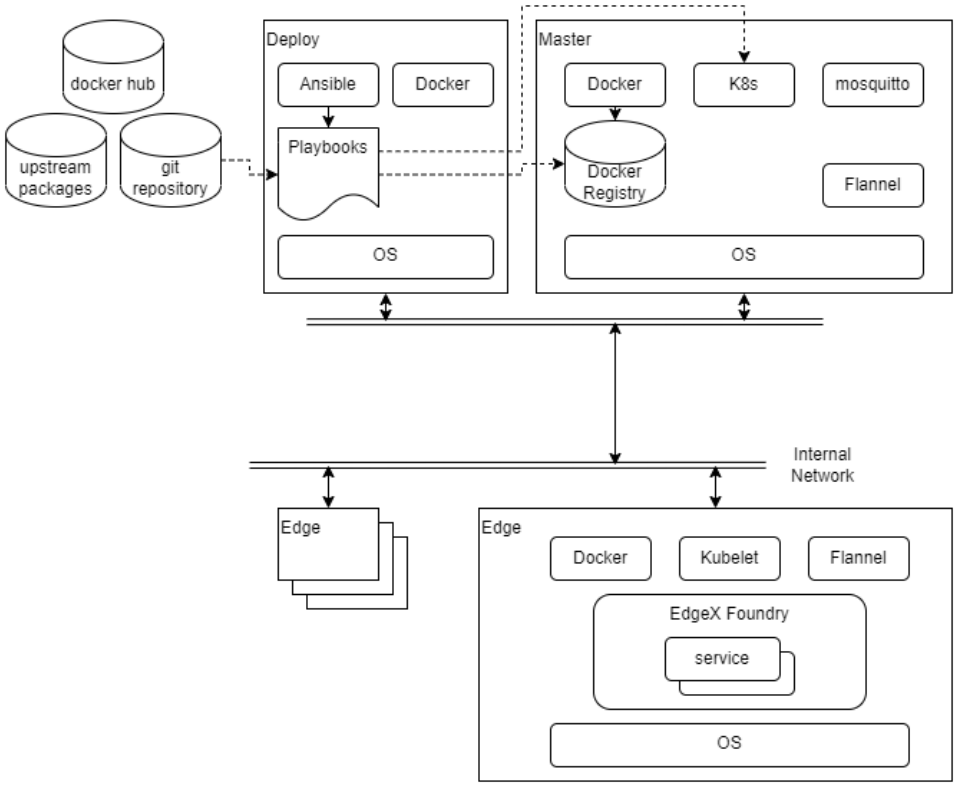
How to Use This Document

This document assumes the reader is familiar with basic UNIX command line utilities and Kubernetes. Familiarity with Ansible and Docker may also be useful. To interact with the EdgeX micro-services in a running setup, use the APIs as described in the EdgeX documentation. Sensor data can be observed through the MQTT broker mosquitto and its command line utility `mosquitto_sub`.

Start by reviewing the deployment architecture and requirements in the following sections, then follow the steps in the Installation section to set up the software and start it running. Confirm the services are functioning as expected by following the instructions in the Verifying the Setup section. The later sections in this document describe other tasks that can be performed on a running setup, alternate configuration options, and how to shut down and uninstall the software.

Deployment Architecture

The diagram below shows the major components and relationships in a deployment of this blueprint.



Deployment, as well as other tasks such as starting and stopping the cluster, is coordinated through a set of Ansible playbooks. (Ansible playbooks are a system used by the Ansible tool for describing the desired state of a system. In many ways they are similar to shell scripts. For more details see the Ansible documentation.) The playbooks are run exclusively by the deploy node, and they execute commands on the deploy node, the master node, and in some cases on the edge nodes. Once the nodes are set up, most activity is carried out by Kubernetes. Kubernetes is configured by the playbooks and told to start or stop services on the edge nodes. These services are run in containers, and the images for these containers are stored in a local Docker registry. There are containers for the Kubernetes components themselves, plus Flannel (a component which provides networking inside the Kubernetes cluster), EdgeX Foundry services, and two custom services (`sync-app` and `device-lora`) built using the EdgeX SDKs.

Note that the deploy node and the master node can be the same host or virtual machine.

The sensor nodes are not shown in the above diagram as they are not envisioned as being connected to the network, and are not configured by the playbooks from the deploy node. See the Sensor Nodes section of Installation for an example of how sensor nodes may be setup.

Pre-Installation Requirements

Hardware Requirements

The table below shows the recommended minimum specifications for the hardware in the testing installation. It is possible that lower spec hardware could be used for many of the nodes. The sensor node hardware in particular is specific to the testing installation and could be swapped out with any number of other platforms as long as LoRa connectivity was possible using the hardware.

	Master, Deploy	**Edge**	**Sensor**
Platform	VM running on commercial grade PC	NVidia Jetson Nano	Raspberry Pi 3
CPU	x86-64, Intel i5 or similar	ARM 64bit Cortex-A57	ARM 32bit Cortex A-53
Cores	2	4* 2 cores should be possible	4* 1 core should be possible
RAM	4 GB	2 GB	1 GB
Storage	128 GB Hard Disk space	32 GB SD Card	32 GB SD Card
Network	1x Ethernet	1x Ethernet	1x Ethernet* *for provisioning

LoRa	N/A	LRA-1 USB dongle	LRA-1 USB dongle
Sensor	N/A	N/A	DHT-11 GPIO temperature/humidity sensor

At a minimum one node is required for the master and deploy roles together, and at least one edge node and one sensor node. The testing installation uses two edge and sensor nodes.

Network Requirements

All nodes are expected to have IP connectivity to one another during installation and normal operation, with the exception of the sensor nodes. In the installation described here, all the nodes are connected to a private wired network operating at 100Mbps or better. However, there are no strict bandwidth or latency requirements.

During initial software installation all of the nodes will require access to the internet to download required software packages. Once the required software packages are installed and the docker registry is started, only the deploy node will need further access to the internet (unless, of course, software packages need to be changed or updated). The deploy node will need to access the internet when pulling upstream images to install in the docker registry, and when building docker images for custom services. Of course, if external tools are going to be used to access the collected data through the MQTT broker (Mosquitto), those tools will need network access to the master node.

When the edge node services are started, images will be downloaded from the docker registry on the master node to the edge nodes, so bandwidth may be a consideration if, for example, the edge nodes are accessed over a mobile network.

Software Prerequisites

The list below shows the required software for each node type prior to beginning the installation process.

- Deploy node
  - Ubuntu 20.04
  - Ansible 2.11.7
- Master node
  - Ubuntu 20.04
- Edge node
  - Ubuntu 20.04
- Sensor node
  - Rasbian 11.1

Note that Ansible 2.9.6 is installed from the regular Ubuntu repository on Ubuntu 20.04, but needs to be upgraded from the Ansible repository to support the `kubernetes.core` collection used by this blueprint. The `setup_cicd.yml` playbook can be run with Ansible 2.9.6 and will update Ansible to the required version.

Additional Installed Software Packages

Note that the installation process will install several more software packages through Ansible playbooks. These are listed below for reference. Packages included by default in an install of Ubuntu 20.04 server are not included. The version numbers are those that are available/installed at the time of writing by the Ansible playbooks on Ubuntu 20.04.

- Deploy node
  - make 4.2.1, build-essential 12.8, python3-pip 20.0.2
  - Ansible collections `community.docker`, `kubernetes.core`, `community.crypto`
  - Docker (docker.io) 20.10.7
- Master node
  - Docker (docker.io) 20.10.7
  - python3-pip 20.0.2
  - Python packages `cryptography` and `kubernetes`
  - mosquitto 2.0.14, mosquitto-clients 2.0.14
  - Kubernetes (kubectl, kubelet, kubeadm) 1.22.6
  - Flannel 0.16.3, flannel-cni-plugin 1.0.1 (Note: These are containers installed via Kubernetes through a config file)
- Edge node
  - Docker (docker.io) 20.10.7
  - Kubernetes (kubelet, kubeadm) 1.22.6 (kubectl may be installed for debugging purposes)

Installation

Setting Up the Deploy Node

The deploy node will coordinate all other installation and operations, so it needs to be set up first. In the test installation, the deploy node is a VM running on a x86 PC, with Ubuntu Linux 20.04 installed. In addition, the Ansible tool must be installed. The Ansible tool provided in the Ubuntu software repository is a slightly older version which needs to be upgraded, but it is sufficient to execute the `setup_deploy.yml` playbook, which will install the newer version of Ansible and other tools required on the deploy node. But before running that playbook you need to configure a few things described in the section below.

The playbooks for use on the deploy node are stored in the `deploy/playbook` directory of the source repository. These playbooks refer to other files in the source code, so the entire directory tree should be copied onto the deploy node. The easiest way to do this is by cloning the git repository directly as shown below:

```
git clone repository-url
```

Note, using the `--depth=1` option can save some disk space if you don't need to modify the source code.

The git command will create a directory in the directory where it is run named after the repository. Inside the new directory will be the `deploy/playbook` directory. Unless noted otherwise, the commands below should be run in that directory.

Node and Cluster Configuration

Before running the `setup_deploy.yml` playbook, modify the `hosts` file in the `deploy/playbook` directory with the host names and IP addresses of the edge nodes in your cluster. Also update the entry for the master node's host if it is not the same as the deploy node.

```
all:
  hosts:
  children:
    master:
      hosts:
        localhost:
    edge_nodes:
      hosts:
        edge1: # Name of first edge node
          ip: 192.168.2.21 # IP address of first edge node
          lora_id: 1
        edge2: # Name of second edge node
          ip: 192.168.2.25 # IP address of second edge node
          lora_id: 4
```

In addition, if the master node is not the same as the deploy node, remove the line `connection: local` wherever it follows `hosts: master` in the playbooks in `deploy/playbook`.

In the file `master.yml` in the `deploy/playbook/group_vars/all` directory, set the `master_ip` value to the IP address of the master node. Note that this is required even if the master node is the same as the deploy node.

```
master_ip: 192.168.2.16
```

Set Up the Deploy Node

The account which runs the deploy playbooks will need to be able to use `sudo` to execute some commands with super-user permissions. The following command can be used (by root or another user which already has super-user permissions) to enable the use of sudo for a user:

```
sudo usermod -aG sudo username
```

After setting IP addresses and node names in the `master.yml` and `hosts` files, you can run the `setup_deploy.yml` playbook using the command below.

```
ansible-playbook -i ./hosts setup_deploy.yml --ask-become-pass
```

This will add the node names and addresses to the deploy node's `/etc/hosts` file as well as upgrade the version of Ansible if necessary. It will also install Ansible collections `community.docker`, `kubernetes.core`, and `community.crypto`, required by the other Ansible playbooks in this blueprint.

Ansible will be installed using root permissions on the deploy node, so supply the `sudo` password (by default the user's password) when prompted for the "become" password.

Preparing the Master Node

If the master node is not on the same host as the deploy node, the user that runs the deploy playbooks must have an account on the master host under the same name, and that account must have `sudo` privileges like the account on the deploy node (see above). Also, the account should have password-less SSH login configured. See the description of configuring password-less login for the edge node administrator account in the Preparing Edge Nodes section.

The following command will prepare the master node for use:

```
ansible-playbook -i ./hosts master_install.yml --ask-become-pass
```

This playbook requires the password for sudo on the master node (the "become" password).

It will perform the following initialization tasks:

- Make sure there are entries for the master and edge node names in `/etc/hosts`
- Install required software packages including Docker, Kubernetes, pip, and mosquitto
- Install Python packages used by other playbooks (`kubernetes` and `cryptography`)
- Make sure the user can run Docker commands
- Prepare basic configuration for Docker and Kubernetes

- Set up a user name and password for the MQTT service

Note, you can customize the MQTT user name and password using the `mqtt_user` and `mqtt_pwd` variables in the `docker/playbook/group_vars/all/mqtt.yml` file. By default the user name is "edge" and the password "edgemqtt". These credentials must be used if you want to, for example, use the `mosquitto_sub` command to monitor incoming MQTT messages from the edge nodes.

Master Node Kubernetes Requirements

Kubernetes' initialization tool `kubeadm` requires that swap be disabled on nodes in the cluster. Turn off swap on the master mode by editing the `/etc/fstab` file (using sudo) and commenting out the line with "swap" as the third parameter:

```
# /swap.img none swap sw 0 0
```

In addition, if you have proxy settings `kubeadm` will warn that you should disable the proxy for cluster IP addresses. The default cluster IP ranges `10.96.0.0/12` and `10.244.0.0/16` should be added to the `no_proxy` and `NO_PROXY` variables in `/etc/environment` if necessary.

```
no_proxy=localhost,127.0.0.0/8,192.168.2.0/24,10.96.0.0/12,10.244.0.0/16,*.local,*.fujitsu.com
NO_PROXY=localhost,127.0.0.0/8,192.168.2.0/24,10.96.0.0/12,10.244.0.0/16,*.local,*.fujitsu.com
```

Creating the Docker Registry

This blueprint sets up a private Docker registry on the master node to hold all the images which will be downloaded to the edge nodes. The following command with start the registry. This command also creates and installs a cryptographic key that is used to identify the registry to the edge nodes.

```
ansible-playbook -i ./hosts start_registry.yml --ask-become-pass
```

Once this command has been run the registry will run as a service and will automatically restart if the master node reboots for some reason. If you need to stop the registry or clear its contents, see the instructions in the Stopping and Clearing the Docker Registry section of the Uninstall Guide.

Note that if you stop and restart the registry new keys will be generated and you will need to run the `edge_install.yml` playbook again to copy them to the edge nodes.

Populating the Registry

The following command will download the required images from their public repositories and store copies in the private repository:

```
ansible-playbook -i ./hosts pull_upstream_images.yml
```

Note that this process can take some time depending on the speed of the internet connection from the master node.

If the version of Kubernetes or Flannel changes you will need to populate the registry with updated images using the above command again. Note that you can force Kubernetes to use a specific patch version by editing the `deploy/playbook/k8s/config.yml` file and adding the line `kubernetesVersion: v1.22.7` (with the version you require) under the the `kind: ClusterConfiguration` line, and running the `master_install.yml` playbook again. (You can also make the same change to `~/.lfedge/config.yml` directly to avoid having to run `master_install.yml` again.)

Populating the registry will leave extra copies of the downloaded images on the master node. You can remove these using the following command (the images will remain in the private registry):

```
ansible-playbook -i ./hosts clean_local_images.yml
```

Preparing Edge Nodes

Add an administrative account to all the edge nodes. This account will be used by the deploy node when it needs to run commands directly on the edge nodes (e.g. for installing base software, or for joining or leaving the cluster). The following commands run on each edge node will add a user account named "edge" and add it to the group of users with `sudo` privileges.

```
sudo adduser edge
sudo usermod -aG sudo edge
```

Note, if you use an administrative account with a different name, change the variable `ansible_user` in the `edge_nodes` group in the `deploy/playbook/hosts` file to match the user name you are using.

The deploy node needs to log in via SSH to the edge nodes using a cryptographic key (rather than a password), so that a password does not need to be provided for every command. Run the following command on the deploy node to create a key called "edge" for the administrative user.

```
ssh-keygen -t ed25519 -f ~/.ssh/edge
```

The parameter `~/.ssh/edge` is the name and location of the private key file that will be generated. If you use a different name or location, change the `ansible_ssh_private_key_file` variable for the `edge_nodes` group in `deploy/playbook/hosts` to match.

Once the key files have been created, the following command can be run from the deploy node to copy the key to each edge node so a password will not be required for each login. (The administrative user's password will be requested when running this command.)

```
ssh-copy-id -i ~/.ssh/edge.pub edge@nodename
```

After the administrative account has been created, the following command will perform initial setup on all edge nodes configured in the `deploy/playbook/hosts` file:

```
ansible-playbook -i ./hosts edge_install.yml
```

The playbook will perform the following initialization tasks:

- Make sure there is an entry for the master node in `/etc/hosts`
- Install required software packages including Docker and kubelet
- Make sure the user can run Docker commands
- Configure Docker, including adding the certificates to secure access to the private registry

Edge Node Kubernetes Requirements

Like the master node, swap should be disabled and the cluster IP address ranges should be excluded from proxy processing if necessary.

Note that on the Jetson Nano hardware platform has a service called `nvzramconfig` that acts as swap and needs to be disabled. Use the following command to disable it:

```
sudo systemctl disable nvzramconfig.service
```

Building the Custom Services

At this time, images for the two custom services, `sync-app` and `device-lora`, need to be built from source and pushed to the private Docker registry. (In the future these images should be available on Docker Hub or another public registry.) Use the following playbooks from the `cicd/playbook` directory on the deploy node to do so.

This command will install components that support cross-compiling the microservices for ARM devices:

```
ansible-playbook -i ./hosts setup_build.yml
```

This command will build local docker images of the custom microservices:

```
ansible-playbook -i ./hosts build_images.yml
```

The build command can take some time, depending on connection speed and the load on the deploy host, especially the compilation of cross-compiled images.

This command will push the images to the private registry:

```
ansible-playbook -i ./hosts push_images.yml
```

At time of writing this step will also create some workaround images required to enable EdgeX security features in this blueprint's Kubernetes configuration. Hopefully, these images will no longer be needed once fixes have been made upstream.

Starting the Cluster

With the base software installed and configured on the master and edge nodes, the following command will start the cluster:

```
ansible-playbook -i ./hosts init_cluster.yml --ask-become-pass
```

This command only starts the master node in the Kubernetes cluster. The state of the master node can be confirmed using the `kubectl get node` command on the master node.

```
admin@master:~$ kubectl get node
NAME   STATUS ROLES                AGE VERSION
master Ready  control-plane,master 14m v1.22.7
```

Adding Edge Nodes to the Cluster

Once the cluster is initialized, the following command will add all the configured edge nodes to the cluster:

```
ansible-playbook -i ./hosts join_cluster.yml
```

The `kubectl get node` command on the master node can be used to confirm the state of the edge nodes.

```
admin@master:~$ kubectl get node
NAME   STATUS ROLES                AGE   VERSION
edge1  Ready  <none>               2m50s v1.22.7
edge2  Ready  <none>               2m45s v1.22.7
master Ready  control-plane,master 17m   v1.22.7
```

Starting EdgeX

After adding the edge nodes to the cluster, the following command will start the EdgeX services on the edge nodes:

```
ansible-playbook -i ./hosts edgex_start.yml
```

You can confirm the status of the EdgeX microservices using the `kubectl get pod` command on the master node. (EdgeX micro-service containers are grouped into one Kubernetes "pod" per node.)

```
admin@master:~$ kubectl get pod
NAME                          READY STATUS  RESTARTS AGE
edgex-edge1-57859dcdff-k8j6g 20/20 Running 16       1m31s
edgex-edge2-5678d8fbbf-q988v 20/20 Running 16       1m26s
```

Note, during initialization of the services you may see some containers restart one or more times. This is part of the timeout and retry behavior of the services waiting for other services to complete initialization and does not indicate a problem.

Sensor Nodes

In the test installation sensor nodes have been constructed using Raspberry Pi devices running a Python script as a service to read temperature and humidity from a DHT-1 sensor, and forward those readings through an LRA-1 USB dongle to a pre-configured destination.

The Python script is located in `sensor/dht2lra.py`, and an example service definition file for use with systemd is `dht2lra.service` in the same directory.

The destination edge node is configured by connecting to the LRA-1 USB dongle, for example using the `tio` program (tio needs to be installed using `sudo apt-get install tio`):

```
pi@raspi02:~ $ sudo tio /dev/ttyUSB0
[tio 09:31:52] tio v1.32
[tio 09:31:52] Press ctrl-t q to quit
[tio 09:31:52] Connected
i2-ele LRA1
Ver 1.07.b+
OK
>
```

At the ">" prompt, enter `dst=`*`N`*, where N is the number in the `lora_id` variable for the edge node in `deploy/playbook/hosts`. Then enter the `ssave` command and disconnect from the dongle (using `Ctrl+t q` in the case of tio). The destination ID will be stored in the dongle's persistent memory (power cycling will not clear the value).

Running the script, either directly with `python ./dht2lra.py`, or using the service, will periodically send readings to the edge node. These readings should appear in the `core-data` database and be possible to monitor using the `edgex-events-`*`nodename`* channel. For example, the following command run on the master node should show the readings arriving at an edge node named "edge1":

```
mosquitto_sub -t edgex-events-edge1 -u edge -P edgemqtt
```

Verifying the Setup

Test cases for verifying the blueprint's operation are provided in the `cicd/tests` directory. These are Robot Framework scripts which can be executed using the `robot` tool. In addition, the `cicd/playbook` directory contains playbooks supporting setup of a Jenkins-based automated testing environment for CI/CD. For more information, consult the `README.md` files in those directories.

Developer Guide and Troubleshooting

EdgeX Service Configuration UI

The configuration parameters of EdgeX micro-services can be accessed through a Consul server on each edge node. The UI is accessible at the address h`ttp://`*`node-address`*`:8500/ui`. The node address is automatically assigned by Kubernetes and can be confirmed using the `kubectl get node -o wide` command on the master node.

In order to access the configuration UI a login token is required. This can be acquired using the `get-consul-acl-token.sh` script in the `edgex` directory. Execute it as follows and it will print out the Consul access token:

```
get-consul-acl-token.sh pod-name
```

The *`pod-name`* parameter is the name of the EdgeX pod running on the node. This can be obtained with the `kubectl get pod` command on the master node. The name of the pod will be shown in the first column of the output, and will be "edgex-*nodename*-..."

Access the UI address through a web browser running on the master node, and click on the "log in" button in the upper right. You will be prompted to enter the access token. Copy the access token printed by the `get-consul-acl-token.sh` script into the text box and press enter to log in to the UI. See the EdgeX documentation and Consul UI documentation for more information.

EdgeX API Access

The EdgeX micro-services each support REST APIs which are exposed through an API gateway running on `https://node-address:8443`. The REST APIs are documented in the EdgeX documentation, and they are mapped to URLs under the API gateway address using path names based on the names of each micro-service. So, for example, the `core-data` service's `ping` interface can be accessed through `https://node-address:8443/core-data/api/v2/ping`. A partial list of these mappings can be found in the EdgeX introduction to the API gateway.

Note that the blueprint does not automatically generate signed certificates for the API gateway, so the certificate it uses by default will cause warnings if accessed using a web browser and require the `-k` option if using the `curl` tool.

There is more information about the API gateway in the EdgeX documentation.

Enabling and Disabling Optional Services

Three EdgeX micro-services can be enabled and disabled using variables in the `deploy/playbook/group_vars/all/edgex.yml` file. Set the variable to `true` to enable the micro-service the next time the `edgex_start.yml` playbook is run. Set the variable to `false` to disable that micro-service. The micro-service controlling variables are listed below:

- `device_virtual`: Enable or disable the `device-virtual` service, provided by EdgeX Foundry, used for testing.
- `device_lora`: Enable or disable the `device-lora` service, one of the custom services provided by this blueprint, which provides support for receiving readings and sending commands to remote sensors over LoRa low-power radio links.
- `sync_app`: Enable or disable the `sync-app` application service, the other custom service provided by this blueprint, which provides a way to forward sensor data to other edge nodes.

Debugging Failures

Consult the sections under Troubleshooting for commands to debug failures. In particular, using the `kubectl` commands described in Accessing Logs, and changing the log levels of services using the configuration UI described above, which can change the logging level of running services, can be useful.

Reporting a Bug

Contact the Smart Data Transaction for CPS mailing list at sdt-blueprint@lists.akraino.org to report potential bugs or get assistance with problems.

Uninstall Guide

Stopping EdgeX

The EdgeX services can be stopped on all edge nodes using the `edgex_stop.yml` playbook. (It is not currently possible to stop and start the services on individual nodes.)

```
ansible-playbook -i ./hosts edgex_stop.yml
```

Confirm that the services have stopped using the `kubectl get pod` command on the master node. It should show no pods in the default namespace.

After stopping the EdgeX services it is possible to restart them using the `edgex_start.yml` playbook as usual. Note, however, that the pod names and access tokens will have changed.

Removing Edge Nodes

The edge nodes can be removed from the cluster using the following command:

```
ansible-playbook -i ./hosts delete_from_cluster.yml
```

This command should be run before stopping the cluster as described in the following section, in order to provide a clean shutdown. It is also possible to re-add the edge nodes using `join_cluster.yml`, perhaps after editing the configuration in the `hosts` file.

Stopping Kubernetes

Kubernetes can be stopped by running the following command. Do this after all edge nodes have been removed.

```
ansible-playbook -i ./hosts reset_cluster.yml --ask-become-pass
```

Stopping and Clearing the Docker Registry

If you need to stop the private Docker registry service for some reason, use the following command:

```
ansible-playbook -i ./hosts stop_registry.yml
```

With the registry stopped it is possible to remove the registry entirely. This will recover any disk space used by images stored in the registry, but means that pull_upsteam_images.yml, build_images.yml, and push_images.yml will need to be run again.

```
ansible-playbook -i ./hosts remove_registry.yml
```

Uninstalling Software Components

Installed software components can be removed with `sudo apt remove` *`package-name`*. See the list of installed software components earlier in this document. Python packages (`cryptography` and `kubernetes`) can be removed with the `pip uninstall` command.

Ansible components installed with ansible-galaxy (`community.docker`, `kubernetes.core`, `community.crypto`) can be removed by deleting the directories under `~/.ansible/collections/ansible_collections` on the deploy node.

Removing Configuration and Temporary Data

This blueprint stores configuration and data in the following places. When uninstalling the software, these folders and files can also be removed, if present, on the master, deploy and edge nodes.

- Master node:
  - ~/.lfedge
  - /opt/lfedge
  - /etc/mosquitto/conf.d/edge.conf
  - /usr/share/keyrings/kubernetes-archive-keyring.gpg
- Edge node:
  - /opt/lfedge
  - /etc/docker/certs.d/master:5000/registry.crt
  - /usr/local/share/ca-certificates/master.crt
  - /etc/docker/daemon.json
  - /usr/share/keyrings/kubernetes-archive-keyring.gpg
- Deploy node:
  - /etc/profile.d/go.sh
  - /usr/local/go
  - ~/edgexfoundry

Troubleshooting

Confirming Node and Service Status

The `kubectl` command can be used to check the status of most cluster components. `kubectl get node` will show the health of the master and edge nodes, and `kubectl get pod` will show the overall status of the EdgeX services. The `kubectl describe pod` *`pod-name`* command can be used to get a more detailed report on the status of a particular pod. The EdgeX configuration UI, described in the section EdgeX Service Configuration UI above, also shows the result of an internal health check of all EdgeX services on the node.

Accessing Logs

The main tool for accessing logs is kubectl logs, run on the master node. This command can be used to show the logs of a running container:

```
kubectl logs -c container-name pod-name
```

It can also be used to check the logs of a container which has crashed or stopped:

```
kubectl logs --previous -c container-name pod-name
```

And it can be used to stream the logs of a container to a terminal:

```
kubectl logs -c container-name pod-name -f
```

The container names can be found in the output of `kubectl describe pod` or in the `edgex/deployments/edgex.yml` file (the names of the entries in the `containers` list).

For the rare cases when the Kubernetes log command does not work, it may be possible to use the `docker log` command on the node you wish to debug.

Maintenance

Stopping and Restarting EdgeX Services

As described in the Uninstall Guide subsection Stopping EdgeX, the EdgeX services can be stopped and restarted using the `edgex_stop.yml` and `edgex_start.yml` playbooks.

Stopping and Restarting the Kubernetes Cluster

Similar to stopping and restarting the EdgeX services, the whole cluster can be stopped and restarted by stopping EdgeX, removing the edge nodes, stopping Kubernetes, starting Kubernetes, adding the edge nodes, and starting EdgeX again:

```
ansible-playbook -i ./hosts edgex_stop.yml

ansible-playbook -i ./hosts delete_from_cluster.yml

ansible-playbook -i ./hosts reset_cluster.yml --ask-become-pass

ansible-playbook -i ./hosts init_cluster.yml --ask-become-pass

ansible-playbook -i ./hosts join_cluster.yml

ansible-playbook -i ./hosts edgex_start.yml
```

Adding and Removing Edge Nodes

Edge nodes can be added an removed by stopping the cluster and editing the `deploy/playbook/hosts` file. The `master_install.yml` and `edge_install.yml` playbooks need to be run again to update `/etc/hosts` and certificates on any added nodes.

Updating the Software

Running setup_deploy.yml, master_install.yml, and edge_install.yml playbooks can be used to update software packages if necessary. Note that Kubernetes is specified to use version 1.22 to avoid problems that might arise from version instability, but it should be possible to update if so desired.

Rebuilding Custom Services

The custom services can be rebuilt by running the `build_images.yml` playbook in `cicd/playbook`. After successfully building a new version of a service, use push_images.yml to push the images to the private Docker registry. The source for the services is found in `edgex/sync-app` and `edgex/device-lora`.

License

The software provided as part of the Smart Data Transaction for CPS blueprint is licensed under the Apache License, Version 2.0 (the "License");

> You may not use the content of this software bundle except in compliance with the License.
>
> You may obtain a copy of the License at <https://www.apache.org/licenses/LICENSE-2.0>
>
> Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
> See the License for the specific language governing permissions and limitations under the License.

The synchronization application and LoRa device service are linked with other Go packages/components when compiled, which are each covered by their own licenses, listed below. Other components downloaded and installed during the blueprint's installation process are covered by their own licenses.

Synchronization Application

The synchronization application is linked with the following packages when compiled:

Package	License Type	License URL
bitbucket.org/bertimus9/systemstat	MIT	https://bitbucket.org/bertimus9/systemstat/src/master/LICENSE
github.com/armon/go-metrics	MIT	https://github.com/armon/go-metrics/blob/master/LICENSE
github.com/cenkalti/backoff	MIT	https://github.com/cenkalti/backoff/blob/master/LICENSE
github.com/diegoholiveira/jsonlogic	MIT	https://github.com/diegoholiveira/jsonlogic/blob/master/LICENSE
github.com/eclipse/paho.mqtt.golang	BSD-3-Clause	https://github.com/eclipse/paho.mqtt.golang/blob/master/LICENSE
github.com/edgexfoundry/app-functions-sdk-go/v2	Apache-2.0	https://github.com/edgexfoundry/app-functions-sdk-go/blob/master/v2/LICENSE

github.com/edgexfoundry/app-functions-sdk-go/v2/internal/etm	MIT	https://github.com/edgexfoundry/app-functions-sdk-go/blob/master/v2/internal/etm/LICENSE
github.com/edgexfoundry/go-mod-bootstrap/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-bootstrap/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-configuration/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-configuration/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-core-contracts/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-core-contracts/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-messaging/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-messaging/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-registry/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-registry/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-secrets/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-secrets/blob/master/v2/LICENSE
github.com/fatih/color	MIT	https://github.com/fatih/color/blob/master/LICENSE.md
github.com/fxamacker/cbor/v2	MIT	https://github.com/fxamacker/cbor/blob/master/v2/LICENSE
github.com/go-kit/kit/log	MIT	https://github.com/go-kit/kit/blob/master/log/LICENSE
github.com/go-logfmt/logfmt	MIT	https://github.com/go-logfmt/logfmt/blob/master/LICENSE
github.com/gomodule/redigo	Apache-2.0	https://github.com/gomodule/redigo/blob/master/LICENSE
github.com/google/uuid	BSD-3-Clause	https://github.com/google/uuid/blob/master/LICENSE
github.com/go-playground/locales	MIT	https://github.com/go-playground/locales/blob/master/LICENSE
github.com/go-playground/universal-translator	MIT	https://github.com/go-playground/universal-translator/blob/master/LICENSE
github.com/go-playground/validator/v10	MIT	https://github.com/go-playground/validator/blob/master/v10/LICENSE
github.com/go-redis/redis/v7	BSD-2-Clause	https://github.com/go-redis/redis/blob/master/v7/LICENSE
github.com/gorilla/mux	BSD-3-Clause	https://github.com/gorilla/mux/blob/master/LICENSE
github.com/gorilla/websocket	BSD-2-Clause	https://github.com/gorilla/websocket/blob/master/LICENSE
github.com/hashicorp/consul/api	MPL-2.0	https://github.com/hashicorp/consul/blob/master/api/LICENSE
github.com/hashicorp/errwrap	MPL-2.0	https://github.com/hashicorp/errwrap/blob/master/LICENSE
github.com/hashicorp/go-cleanhttp	MPL-2.0	https://github.com/hashicorp/go-cleanhttp/blob/master/LICENSE
github.com/hashicorp/go-hclog	MIT	https://github.com/hashicorp/go-hclog/blob/master/LICENSE
github.com/hashicorp/go-immutable-radix	MPL-2.0	https://github.com/hashicorp/go-immutable-radix/blob/master/LICENSE
github.com/hashicorp/golang-lru/simplelru	MPL-2.0	https://github.com/hashicorp/golang-lru/blob/master/simplelru/LICENSE
github.com/hashicorp/go-multierror	MPL-2.0	https://github.com/hashicorp/go-multierror/blob/master/LICENSE
github.com/hashicorp/go-rootcerts	MPL-2.0	https://github.com/hashicorp/go-rootcerts/blob/master/LICENSE
github.com/hashicorp/serf/coordinate	MPL-2.0	https://github.com/hashicorp/serf/blob/master/coordinate/LICENSE
github.com/leodido/go-urn	MIT	https://github.com/leodido/go-urn/blob/master/LICENSE
github.com/mattn/go-colorable	MIT	https://github.com/mattn/go-colorable/blob/master/LICENSE
github.com/mattn/go-isatty	MIT	https://github.com/mattn/go-isatty/blob/master/LICENSE
github.com/mitchellh/consulstructure	MIT	https://github.com/mitchellh/consulstructure/blob/master/LICENSE
github.com/mitchellh/copystructure	MIT	https://github.com/mitchellh/copystructure/blob/master/LICENSE
github.com/mitchellh/mapstructure	MIT	https://github.com/mitchellh/mapstructure/blob/master/LICENSE
github.com/mitchellh/reflectwalk	MIT	https://github.com/mitchellh/reflectwalk/blob/master/LICENSE
github.com/pebbe/zmq4	BSD-2-Clause	https://github.com/pebbe/zmq4/blob/master/LICENSE.txt

github.com/pelletier/go-toml	Apache-2.0	https://github.com/pelletier/go-toml/blob/master/LICENSE
github.com/x448/float16	MIT	https://github.com/x448/float16/blob/master/LICENSE
golang.org/x/crypto/sha3	BSD-3-Clause	https://pkg.go.dev/golang.org/x/crypto/sha3?tab=licenses
golang.org/x/net	BSD-3-Clause	https://pkg.go.dev/golang.org/x/net?tab=licenses
golang.org/x/sys	BSD-3-Clause	https://pkg.go.dev/golang.org/x/sys?tab=licenses
golang.org/x/text	BSD-3-Clause	https://pkg.go.dev/golang.org/x/text?tab=licenses

LoRa Device Service

The LoRa device service is linked with the following packages when compiled:

Package	License Type	License URL
bitbucket.org/bertimus9/systemstat	MIT	https://bitbucket.org/bertimus9/systemstat/src/master/LICENSE
github.com/armon/go-metrics	MIT	https://github.com/armon/go-metrics/blob/master/LICENSE
github.com/cenkalti/backoff	MIT	https://github.com/cenkalti/backoff/blob/master/LICENSE
github.com/eclipse/paho.mqtt.golang	BSD-3-Clause	https://github.com/eclipse/paho.mqtt.golang/blob/master/LICENSE
github.com/edgexfoundry/device-sdk-go/v2	Apache-2.0	https://github.com/edgexfoundry/device-sdk-go/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-bootstrap/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-bootstrap/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-configuration/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-configuration/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-core-contracts/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-core-contracts/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-messaging/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-messaging/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-registry/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-registry/blob/master/v2/LICENSE
github.com/edgexfoundry/go-mod-secrets/v2	Apache-2.0	https://github.com/edgexfoundry/go-mod-secrets/blob/master/v2/LICENSE
github.com/fatih/color	MIT	https://github.com/fatih/color/blob/master/LICENSE.md
github.com/fxamacker/cbor/v2	MIT	https://github.com/fxamacker/cbor/blob/master/v2/LICENSE
github.com/go-kit/kit/log	MIT	https://github.com/go-kit/kit/blob/master/log/LICENSE
github.com/go-logfmt/logfmt	MIT	https://github.com/go-logfmt/logfmt/blob/master/LICENSE
github.com/google/uuid	BSD-3-Clause	https://github.com/google/uuid/blob/master/LICENSE
github.com/go-playground/locales	MIT	https://github.com/go-playground/locales/blob/master/LICENSE
github.com/go-playground/universal-translator	MIT	https://github.com/go-playground/universal-translator/blob/master/LICENSE
github.com/go-playground/validator/v10	MIT	https://github.com/go-playground/validator/blob/master/v10/LICENSE
github.com/go-redis/redis/v7	BSD-2-Clause	https://github.com/go-redis/redis/blob/master/v7/LICENSE
github.com/gorilla/mux	BSD-3-Clause	https://github.com/gorilla/mux/blob/master/LICENSE
github.com/gorilla/websocket	BSD-2-Clause	https://github.com/gorilla/websocket/blob/master/LICENSE
github.com/hashicorp/consul/api	MPL-2.0	https://github.com/hashicorp/consul/blob/master/api/LICENSE
github.com/hashicorp/errwrap	MPL-2.0	https://github.com/hashicorp/errwrap/blob/master/LICENSE
github.com/hashicorp/go-cleanhttp	MPL-2.0	https://github.com/hashicorp/go-cleanhttp/blob/master/LICENSE
github.com/hashicorp/go-hclog	MIT	https://github.com/hashicorp/go-hclog/blob/master/LICENSE
github.com/hashicorp/go-immutable-radix	MPL-2.0	https://github.com/hashicorp/go-immutable-radix/blob/master/LICENSE

github.com/hashicorp/golang-lru/simplelru	MPL-2.0	https://github.com/hashicorp/golang-lru/blob/master/simplelru/LICENSE
github.com/hashicorp/go-multierror	MPL-2.0	https://github.com/hashicorp/go-multierror/blob/master/LICENSE
github.com/hashicorp/go-rootcerts	MPL-2.0	https://github.com/hashicorp/go-rootcerts/blob/master/LICENSE
github.com/hashicorp/serf/coordinate	MPL-2.0	https://github.com/hashicorp/serf/blob/master/coordinate/LICENSE
github.com/leodido/go-urn	MIT	https://github.com/leodido/go-urn/blob/master/LICENSE
github.com/mattn/go-colorable	MIT	https://github.com/mattn/go-colorable/blob/master/LICENSE
github.com/mattn/go-isatty	MIT	https://github.com/mattn/go-isatty/blob/master/LICENSE
github.com/mitchellh/consulstructure	MIT	https://github.com/mitchellh/consulstructure/blob/master/LICENSE
github.com/mitchellh/copystructure	MIT	https://github.com/mitchellh/copystructure/blob/master/LICENSE
github.com/mitchellh/mapstructure	MIT	https://github.com/mitchellh/mapstructure/blob/master/LICENSE
github.com/mitchellh/reflectwalk	MIT	https://github.com/mitchellh/reflectwalk/blob/master/LICENSE
github.com/OneOfOne/xxhash	Apache-2.0	https://github.com/OneOfOne/xxhash/blob/master/LICENSE
github.com/pebbe/zmq4	BSD-2-Clause	https://github.com/pebbe/zmq4/blob/master/LICENSE.txt
github.com/pelletier/go-toml	Apache-2.0	https://github.com/pelletier/go-toml/blob/master/LICENSE
github.com/tarm/serial	BSD-3-Clause	https://github.com/tarm/serial/blob/master/LICENSE
github.com/x448/float16	MIT	https://github.com/x448/float16/blob/master/LICENSE
golang.org/x/crypto/sha3	BSD-3-Clause	https://pkg.go.dev/golang.org/x/crypto/sha3?tab=licenses
golang.org/x/net	BSD-3-Clause	https://pkg.go.dev/golang.org/x/net?tab=licenses
golang.org/x/sys	BSD-3-Clause	https://pkg.go.dev/golang.org/x/sys?tab=licenses
golang.org/x/text	BSD-3-Clause	https://pkg.go.dev/golang.org/x/text?tab=licenses
gopkg.in/yaml.v3	MIT	https://github.com/go-yaml/yaml/blob/v3/LICENSE

References

- EdgeX Foundry Documentation (release 2.1): https://docs.edgexfoundry.org/2.1/

Definitions, Acronyms and Abbreviations

- CPS: Cyber-Physical System
- MQTT: A lightweight, publish-subscribe network protocol designed for connecting remote devices, especially when there are bandwidth constraints. (MQTT is not an acronym.)