Smart Data Transaction for CPS Blueprint Architecture

Blueprint Overview/Introduction

The Smart Data Transaction for CPS (Cyber-Physical Systems) Blueprint implements methods to adapt the edge network, possibly dynamically, to reduce pressure on network bandwidth and expand the potential range of the edge network's reach while reducing power consumption. The two main features of the blueprint are data sharing between edge nodes and the addition of support for low-power radio connections to sensors.

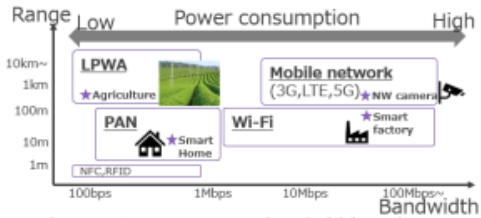


Figure1:Sensor network bandwidth and range

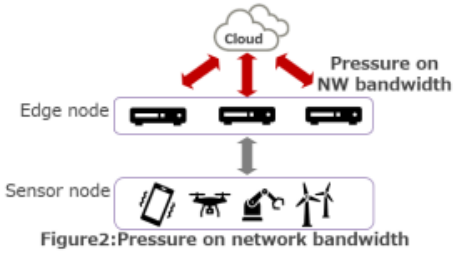


Figure2:Pressure on network bandwidth

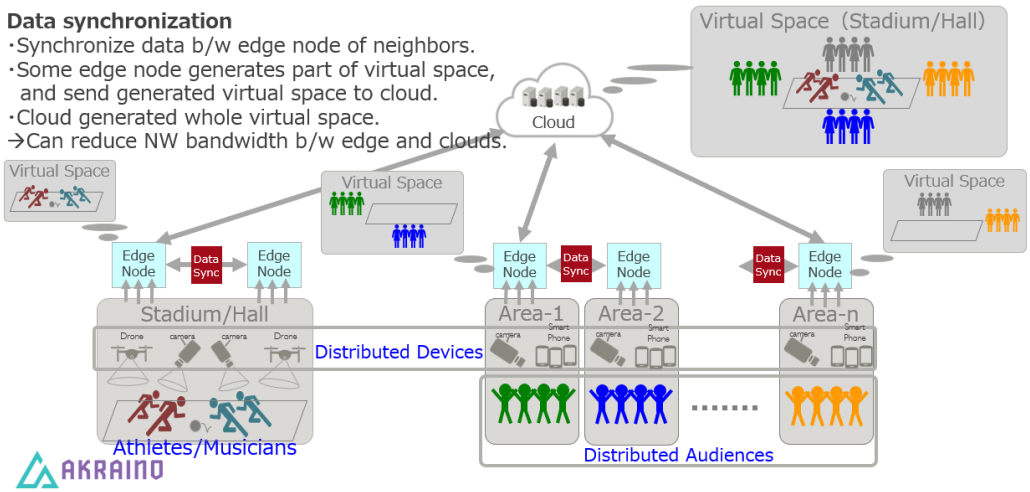
In Akraino Release 6 the Smart Data Transaction for CPS blueprint consists of a test implementation of these features based on EdgeX and Kubernetes technologies, suitable for evaluating the concepts and identifying future directions.

Use Cases

Below are some sample use cases which could benefit from the approaches implemented in this blueprint.

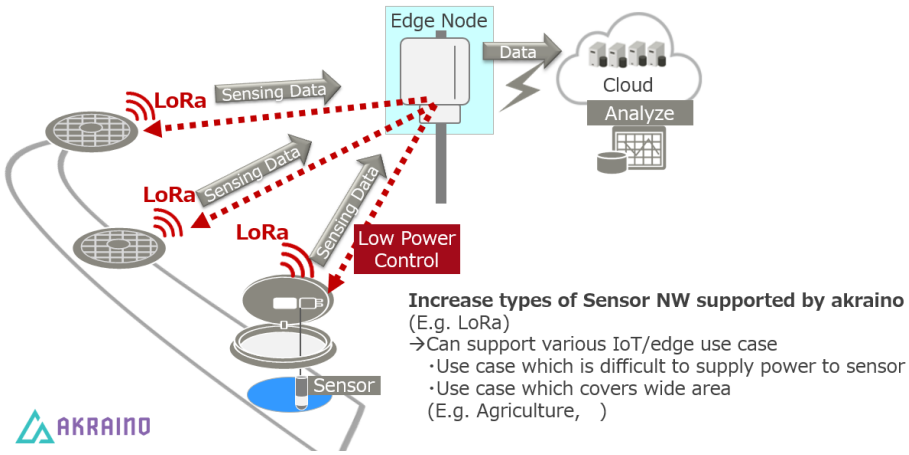
Interactive Live Sports/Music

In a live event featuring audiences in multiple remote spaces, data can be combined by sharing between groups of edge nodes and sensors in each physical space. The combined data, such as video and audio feeds, can be integrated to provide a shared experience in a 3D virtual space to all audiences while reducing overall network bandwidth usage.



Agriculture/Infrastructure Monitoring

Widely separated sensor nodes can benefit from low-power radio connectivity to edge nodes, and edge nodes can react in real time to events in neighboring areas. The illustration below shows a storm drain monitoring application where, for example, sudden changes in water level can be used to trigger increased sampling rate in nearby sensors.



Where on the Edge

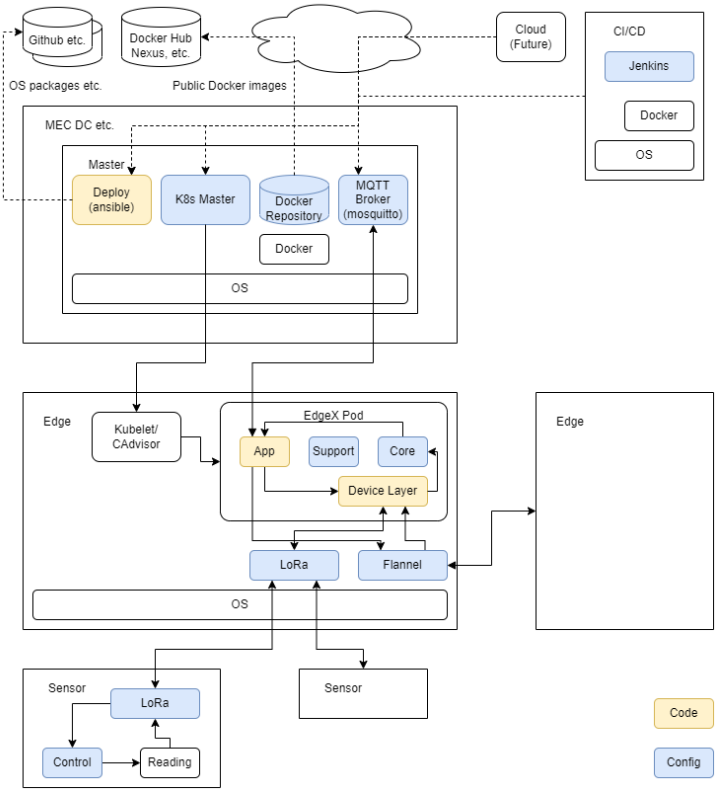
The blueprint is mainly focused on interactions between groups of smart edge nodes or gateways and sensors which can be grouped, either by physical location or by some other shared characteristics. The blueprint applies to situations where a large amount of data is generated from large numbers of sensor nodes, and which can benefit from localized integration and analysis of this data. Some such situations are described in the use cases above.

Business Drivers

Cyber-physical systems (CPS), which combine sensor networks with computing to monitor and control the physical environment are becoming more popular each day. The bandwidth of the sensor network depends on use cases. Large amounts of data from large numbers of sensor nodes will put pressure on the available network bandwidth between the edge and the cloud. Therefore, we need to have a means to optimize bandwidth utilization according to use cases. This blueprint propose a solution for network bandwidth optimization through data sharing at the edge, also enabling a more responsive and smarter total system.

Overall Architecture

The diagram below shows the overall architecture of this blueprint. The colored components represent customizations or new implementations added for this blueprint. The details of the edge node and inter-edge data sharing are further illustrated below.

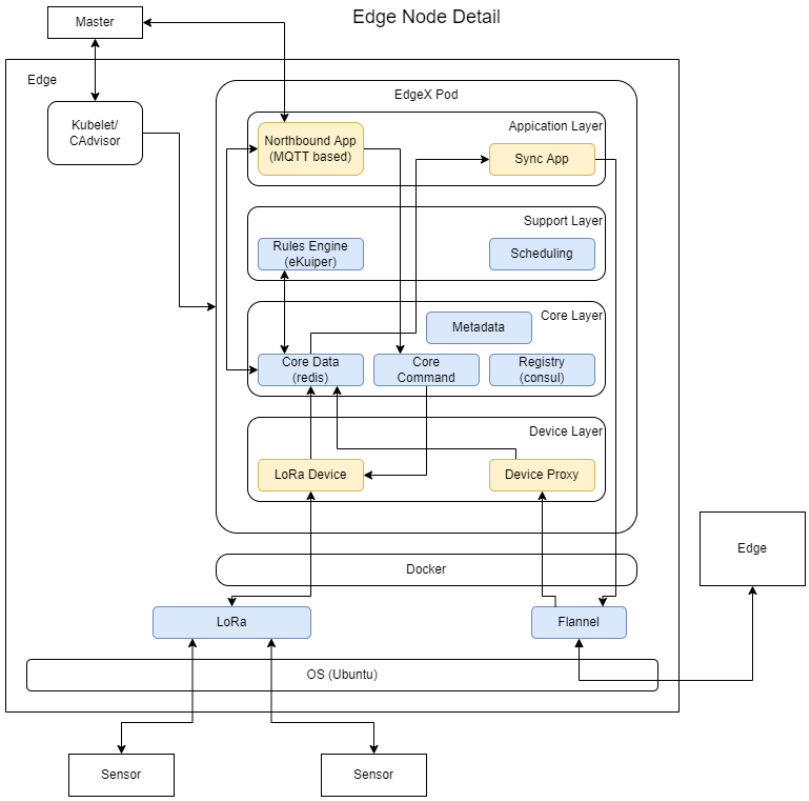


The Smart Data Transaction for CPS blueprint consists of the following types of node roles.

- CI/CD: Runs Jenkins to pull source and scripts, build components and run tests.
- Deploy: Runs scripts (mainly Ansible playbooks) to install components on master and edge nodes.
- Master: Runs the Kubernetes controller for orchestrating the edge nodes, a local docker registry providing container images for the edge nodes, and mosquitto (MQTT broker) for collecting data from edge nodes and sending commands to them.
- Edge: Collects sensor data, performs edge processing, and forwards data to the MQTT broker on the master node.
- Sensor: May be a smart sensor or a device containing sensor hardware and a communications device (e.g. LoRa) so that sensing data can be collected remotely by the edge nodes.

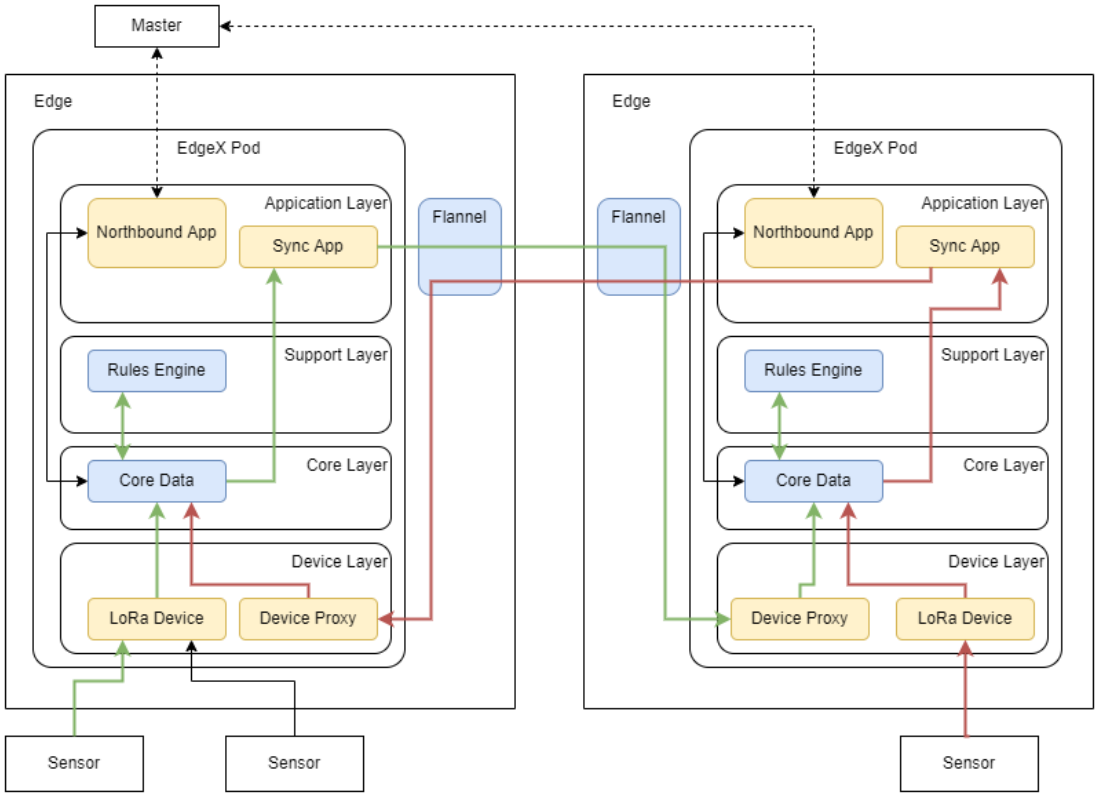
Some of the node roles can be combined on a single piece of hardware. For example, the "deploy" and "master" nodes can be the same server in practice.

The diagram below shows the details of the edge node. The edge node runs an instance of the EdgeX Foundry microservices stack. The EdgeX Foundry stack is instantiated on multiple edge nodes using Kubernetes from the master node.



The new LoRa device service can provide low-power radio connectivity to remote sensor nodes from the edge nodes. The device proxy service and synchronization application allow edge nodes to share data.

The diagram below illustrates the architecture for data sharing between two edge nodes. (The green and red lines show flows of data originating on the left and right nodes respectively.) The decision whether to share a piece of data and with which nodes is under the control of custom rules configured in the EdgeX Foundry rules engine service. The synchronization application picks up the data marked by the rules engine and forwards it to the appropriate edge node through the cluster network. The shared data is received by the device proxy service, which forwards it to the core data service where it can be processed as a regular event, including such processing as sending commands to local sensors based on the content of that event and configured rules.



Platform Architecture

This blueprint describes a self-contained system with the master, deploy and CI/CD nodes running on dedicated on-premises hardware and edge and sensor nodes running on a private network under the customer/user's control. The details of the hardware and network setup can be customized by the user as they see fit.

The table below summarizes the platform for each node in the blueprint.

	Master, Deploy, CI/CD	Edge	Sensor
Hardware Platform	VM running on commercial-grade PC or better	Jetson Nano	Raspberry Pi 3
CPU	x86 (Intel i5, 2 cores)	ARM (Cortex-A57, 4 cores)	ARM (Cortex-A53, 4 cores)
OS	Linux (Ubuntu 20.04)	Linux (Ubuntu 20.04)	Linux (Rasbian 11.1)
Memory	4 GB	2 GB	1 GB
Storage	128 GB HD	32 GB SD card	32 GB SD card
Network	Wired ethernet x1	Wired ethernet x1	Wired ethernet x1* *For management/provisioning
Other		LoRa dongle (LRA-1)	LoRa dongle (LRA-1) Sensor (DHT-11)

The specifications above are those used for the blueprint implementation for testing, but other base hardware or OS choices could easily be supported in many cases.

Software Platform Architecture

The software components used by the blueprint are listed below.

- Master node:
  - OS: Ubuntu 20.04
  - Cluster management: Kubernetes 1.22.6

    - Cluster networking: Flannel 0.16.3 (CNI plugin 1.0.1)
    - Container repository: Docker Registry 2.7.1
    - Container runtime: Docker 20.10.7
    - MQTT broker: Mosquitto 2.0.14
- Deploy node:
    - OS: Ubuntu 20.04
    - Deploy scripting: Ansible 4.9.0
- Edge node:
    - OS: Ubuntu 20.04
    - Cluster service: kubelet 1.22.6
    - Container runtime: Docker 20.10.7
    - Edge services: EdgeX Foundry 2.1.0
- Sensor node:
    - OS: Rasbian 11.1
    - Scripting: Python 3.9.2
- CI/CD:
    - Build automation: Jenkins 2.319.2
    - Test scripting: Robot Framework 4.1.2

The current blueprint does not describe external interfaces e.g. to cloud services. The MQTT broker could be configured to supply data and control interfaces to an external service, but the blueprint does not define that functionality.

APIs

No additional APIs are defined by this blueprint.

The LoRa device service adds a new device profile to the EdgeX Foundry metadata supporting temperature and humidity readings from the DHT-11 sensor over the LoRa connection, and control of the sensor sampling interval. The LoRa connection parameters (channel, group and station IDs) are controlled via configuration values for the LoRa service.

Hardware and Software Management

Hardware provisioning and software management is described in the installation guide. Software is managed using Ansible playbooks and Kubernetes tools such as kubectl via the command line.

Licensing

- Scripts and additional source for the LoRa device service and synchronization application are licensed under the Apache 2.0 license.