Enterprise Cyber-Physical Edge Virtualization Engine (EVE)

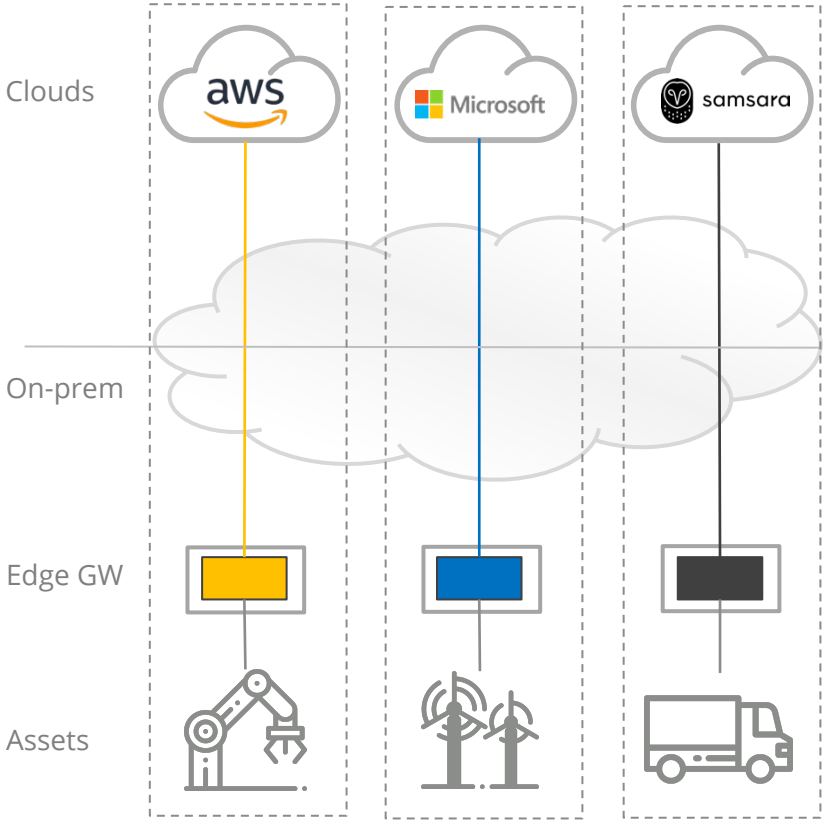
Architecture

ZEDEDA Inc. contribution

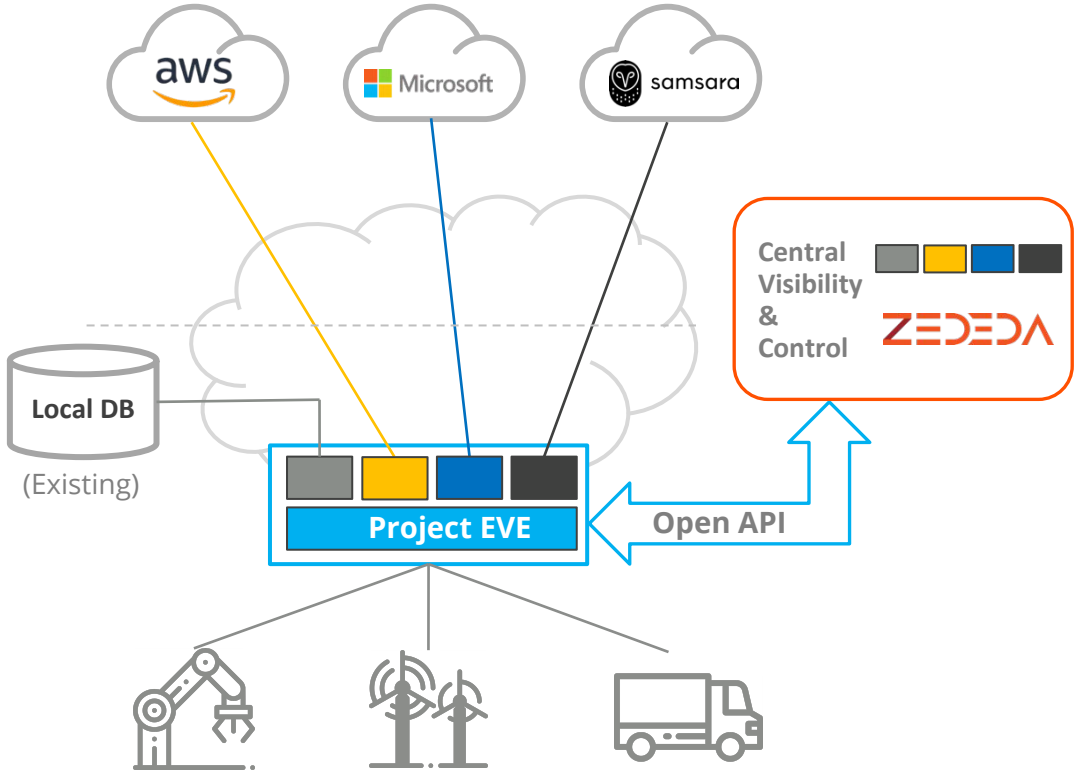
THE LINUX FOUNDATION

The need for edge virtualization: IIoT 1.0 → IIoT 2.0

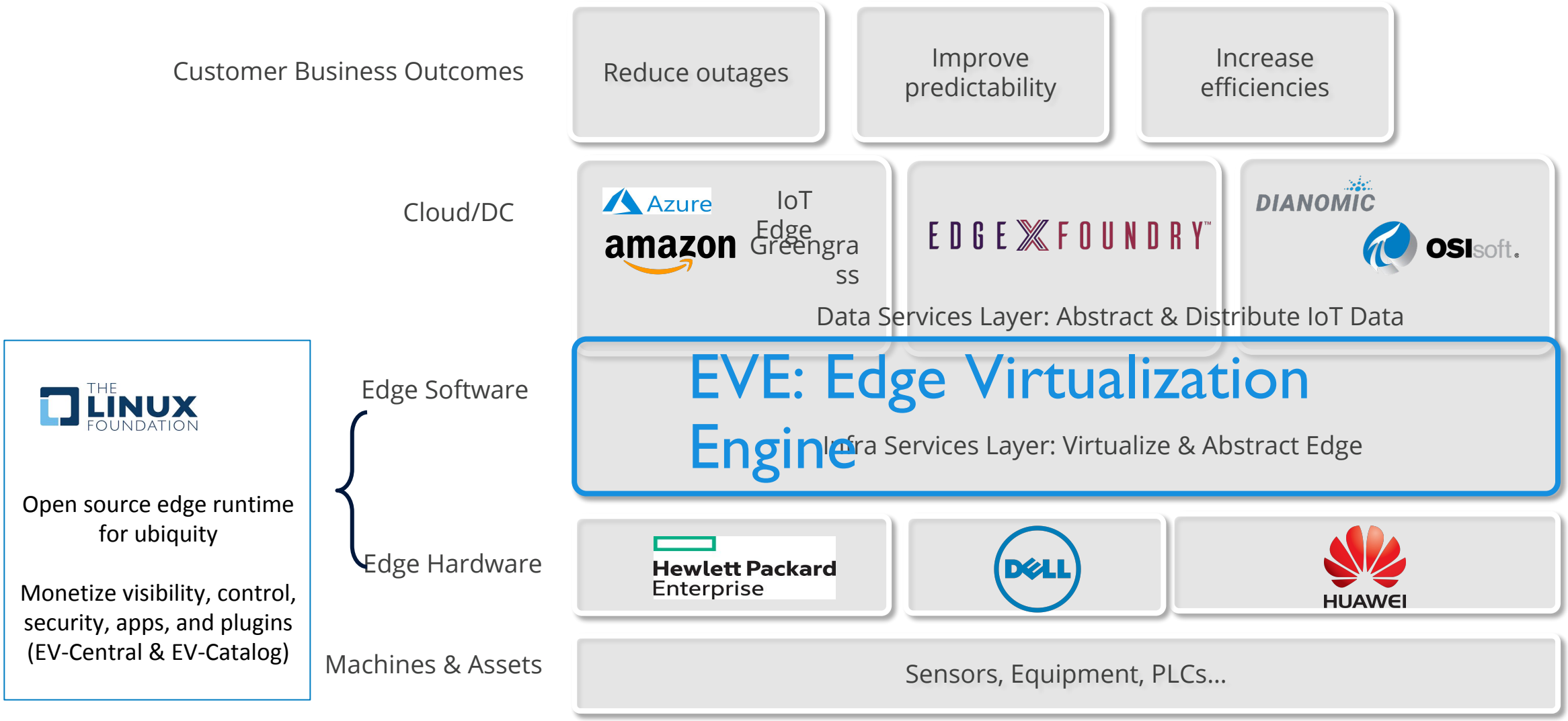
IIoT 1.0: Vertical data silos & platform lock-in
Data/edge sovereignty & control issues
Hardware-defined & unmanaged edge



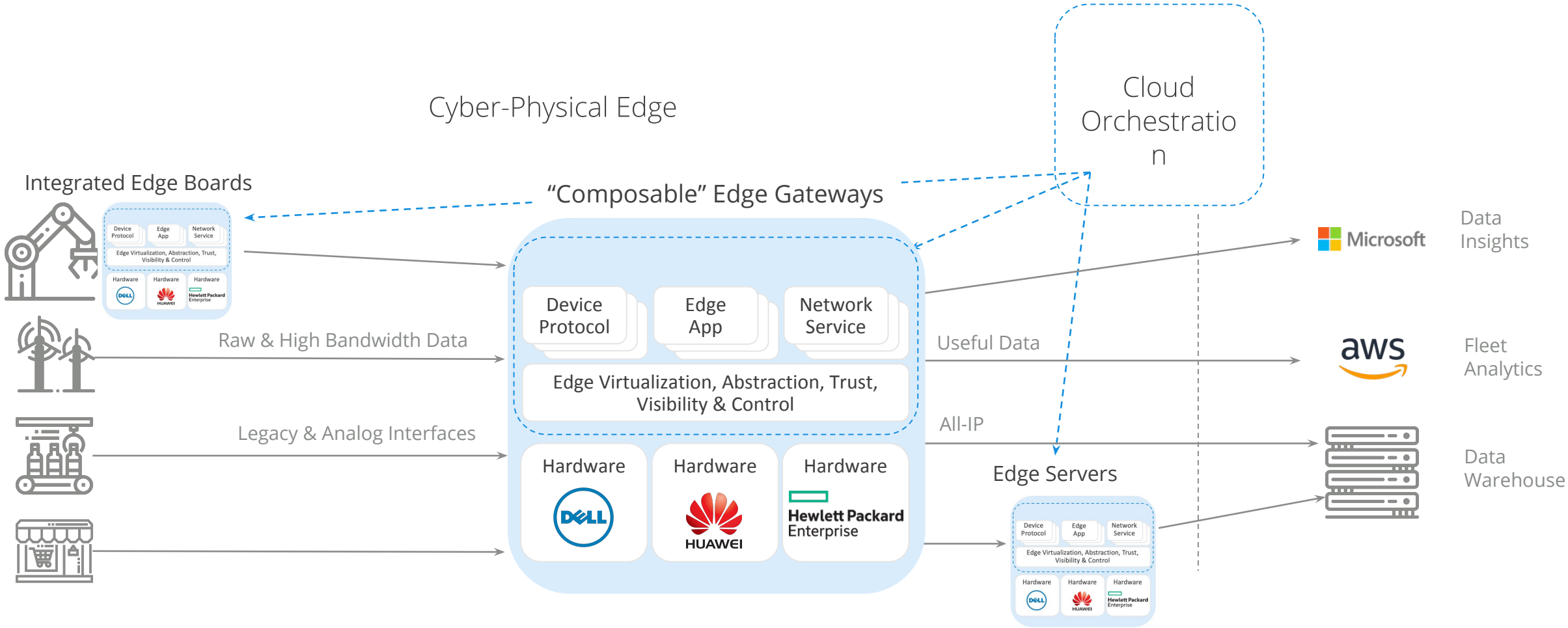
IIoT 2.0: Open IoT data architecture, no lock-in
Data & edge belong to the enterprise
Software-defined & ubiquitous edge



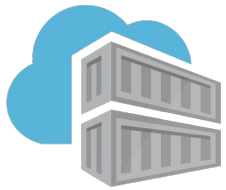
The Enterprise Cyber-Physical Edge Stack



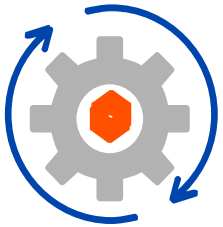
The virtualized, software-defined & composable edge



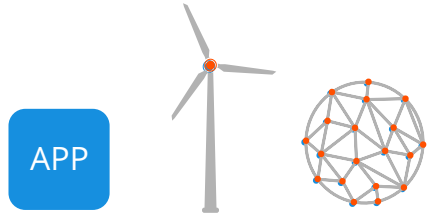
Key Requirements



EDGE CONTAINERS



ZERO TOUCH

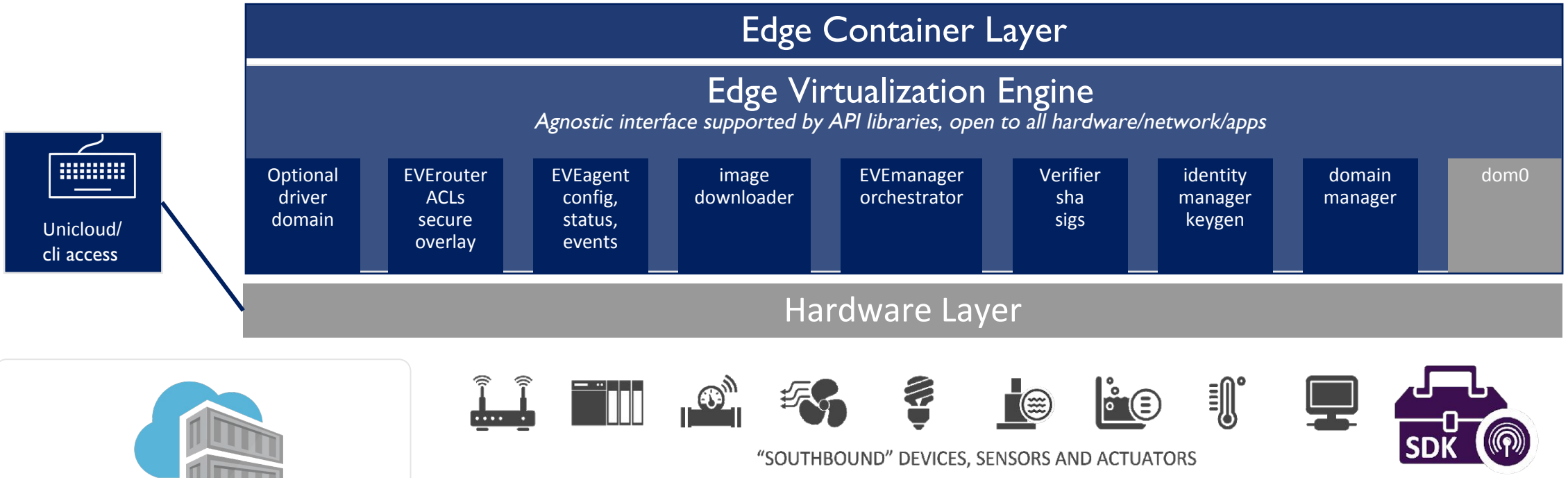


ANY

APP | HARDWARE | NETWORK

ZERO TRUST

Edge Virtualization Engine (Project EVE) Components

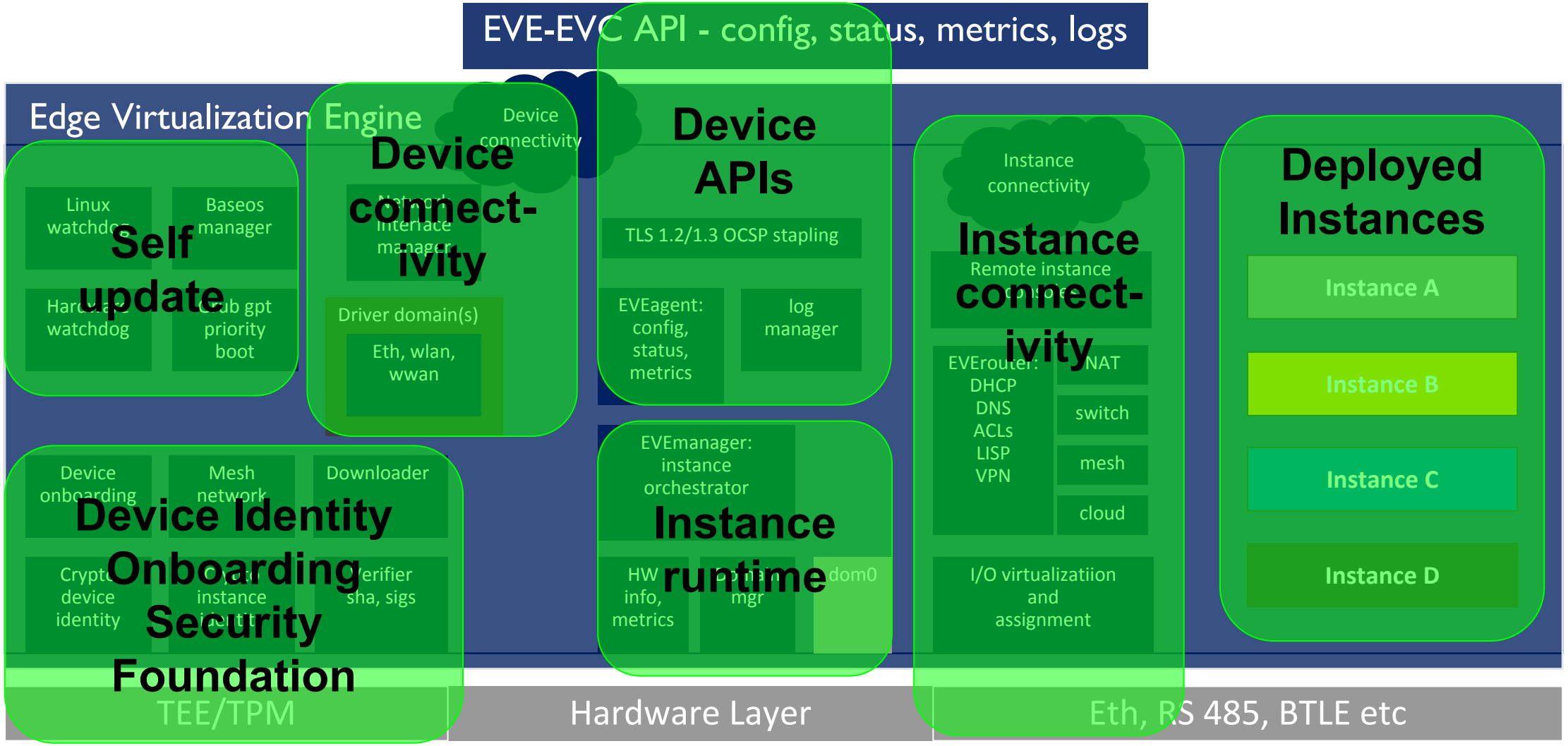


EDGE CONTAINERS

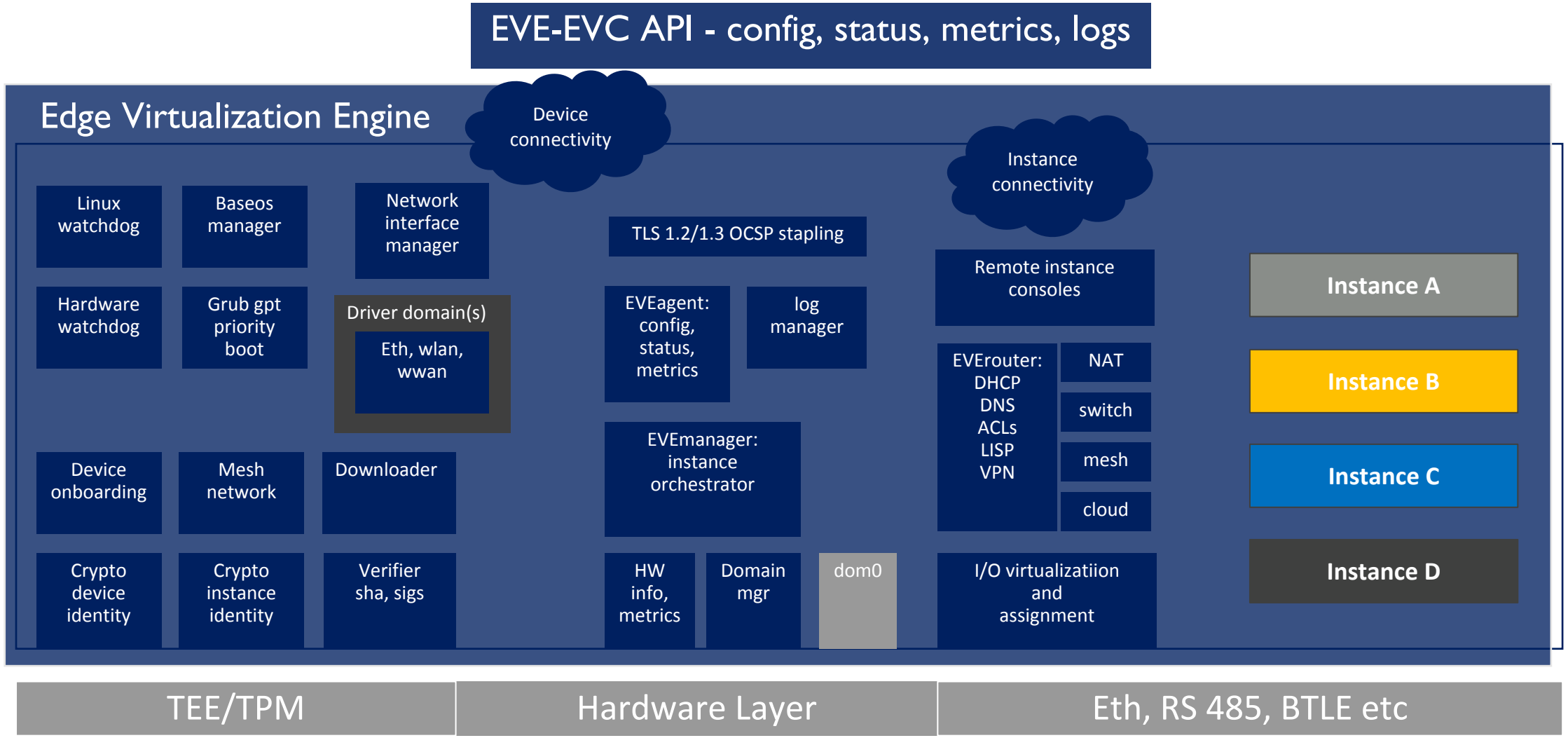
Edge Virtualization Engine Status

- Some pieces already in open repos
  - github.com/zededa/zenbuild
    - grub patches for gpt priority
    - linuxkit-based build
  - github.com/farinacci/lispers.net
    - reference implementation of mesh network
- Opening up other EVE pieces and EVE-EVC API 1H2019
  - Onboarding, self-update, application aka edge container mgmt, connectivity
  - APIs for onboarding, config, info, metrics, logs, [events]
  - In the process of adding some developer documentation
- Moving to github.com/lf-edge

Project EVE Architecture



Project EVE Architecture



Identity, onboarding, and security foundation

- Using self-signed certificates using elliptic curve key pairs
  - Reasonable key size for 20 year time frame
  - Considering adding certificate signing request
  - At factory/install specify EVC plus root CA certificate for EVC
- Leverage TEE/TPM for secure key storage, measured boot, etc
  - Device private key never needs to leave TEE/TPM
- Several variants for onboarding depending on factory constraints
  - Want strong binding between user/purchaser and device identity
- Images are signed; verified by device; can pull from any datastore
- No remote (ssh) or keyboard access to EVE(*)

(*) Can enable using API for developer debug

Self-update

- Requirement to never have to visit device due to software bugs and failures
  - Including due to power failure during flashing of base image
  - Either fall back to old image or be able to do another update
- Dual partition boot (IMGA/IMGB)
  - grub patches for gpt priority boot
  - Additional partitions for identity (CONFIG) and app instances (PERSIST)
- Policies and timers for fallback vs. commit to new
  - "Test" that new base image can connect to EVC etc
  - Deployed app instances are not tested as part of this
- Using hardware watchdog plus Linux watchdog to detect hangs and core dumps and reboot
- Been using this approach in dev for 12 months without bricking a device

Device Connectivity

- Device needs to connect to EVC; can also specify local connectivity for app instances
- By default connects using DHCP/IPv4 over eth0, wlan0, and wwan0
  - Will use multiple ports for failover and load spreading if available
- Can specify different ports, static IPs, enterprise proxy config, etc
  - At software install time with a json file in /config/, or USB stick
  - Using device API
- Device tests connectivity to EVC with fallback to old, retry of new
  - Reports results using API
- Prints connectivity diagnostics on console (useful if local console; e.g., to debug proxy config)

Current Edge Container definition

- Images are qcow2 or raw format; manifest refers to one or more images. Includes Access Control Lists. Example:

```
{
    "acKind": "VMManifest",
    "acVersion": "1.1.1",
    "name": "xenial2intf",
    "owner": {},
    "enablevnc": true,
    "vmmode": "HV_HVM",
```

```
"images": [
    {
        "imagename": "xenial-amd64-docker-20180725",
        "maxsize": 1195376,
        "readonly": false,
        "preserve": true,
        "target": "Disk",
        "drvtype": "HDD",
        "maxsizeUnit": "GB",
        "maxsizeDisplayUnit": "GB"
    }
],
```

```
"interfaces": [ {
    "name": "indirect",
    "directattach": false,
    "acls": [ {
         "matches": [ {
             "type": "host",
             "value": "amazonaws.com"
          } ] } ] },
 {  "name": "direct",
    "directattach": false,
    "acls": [ {
        "matches": [ {
            "type": "ip",
            "value": "0.0.0.0/0"
          } ] } ] } ],
```

```
"resources": [
    {
        "name": "cpus",
        "value": 2
    },
    {
        "name": "memory",
        "value": 512000
    },
    {
        "name": "storage",
        "value": 3145728
    }
]
```

App Instance Connectivity

- Default is local network with NATed connectivity
- Can provision a switch network - an L2 network e.g, on eth1
- Can provision PCI controller or COM port if instance has its own drivers (industrial Ethernet, TSN, BTLE, modbus over serial)
- Can provision a cloud network - connect to AWS, Azure VPN
- Can provision a mesh network - connect device to device
  - Uses LISP (https://tools.ietf.org/html/rfc6830)
  - Handles multihoming, mobility, NAT traversal, authentication, encryption
  - No changes to app; uses DHCP to get IP addresses as normal
- Can provision a local network with no external port; local-only
- If vnc is enabled in manifest can use Guacamole for remote console

EVE-EVC API

- Connection from device (through NAT) using TLS1.2 (soon 1.3)
- Different services:
  - POST api/v1/edgedevice/register for device onboarding
  - GET api/v1/edgedevice/ping for connectivity test
  - GET api/v1/edgedevice/config complete device + instance config
  - POST api/v1/edgedevice/info for triggered device/instance status
  - POST api/v1/edgedevice/metrics for periodic device/instance metrics
  - POST api/v1/edgedevice/logs for logs from microservices on device
- Protobuf encoded messages

Questions?