Enterprise Cyber-Physical Edge Virtualization Engine (EVE)

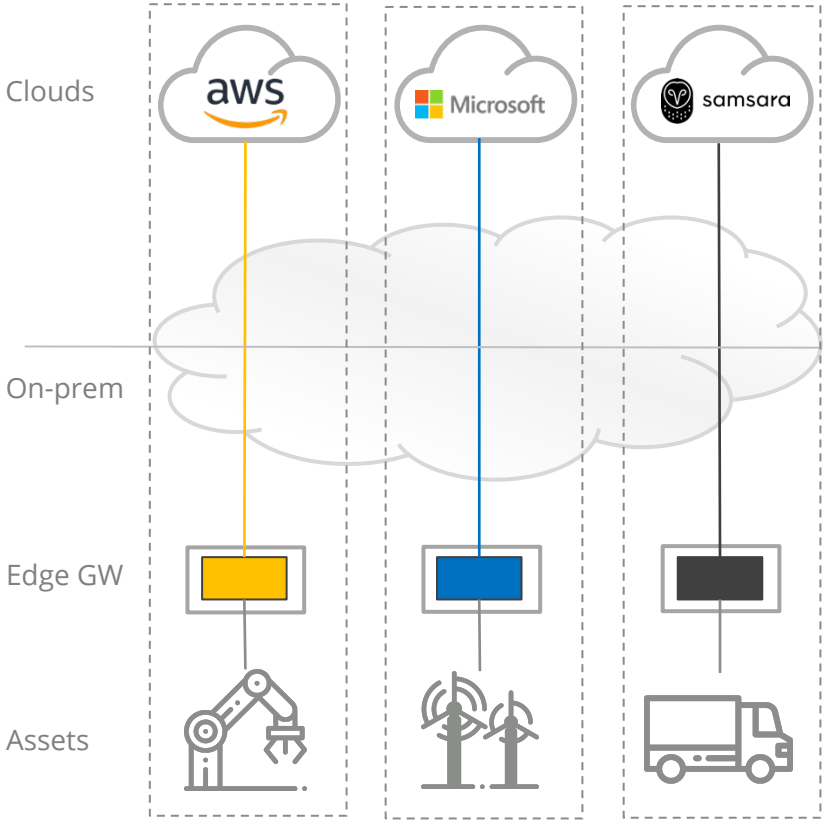
Motivation, Architecture, APIs

ZEDEDA Inc. contribution

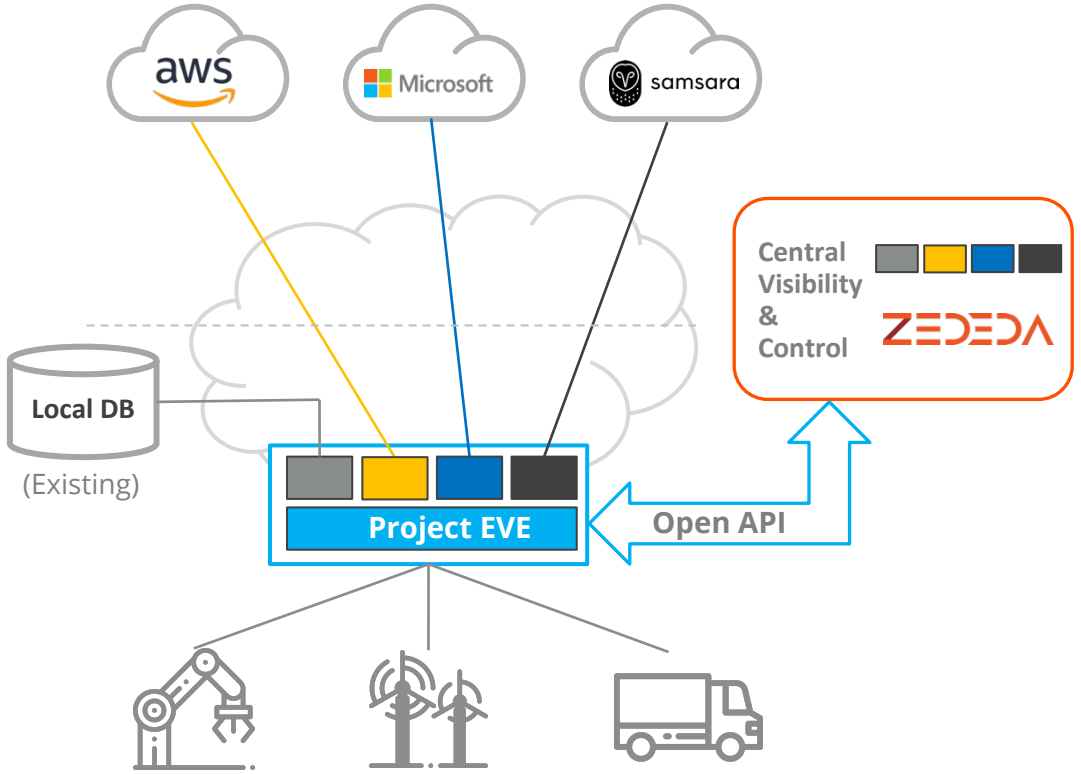
Erik Nordmark, Chief Architect

The need for edge virtualization: IIoT 1.0 → IIoT 2.0

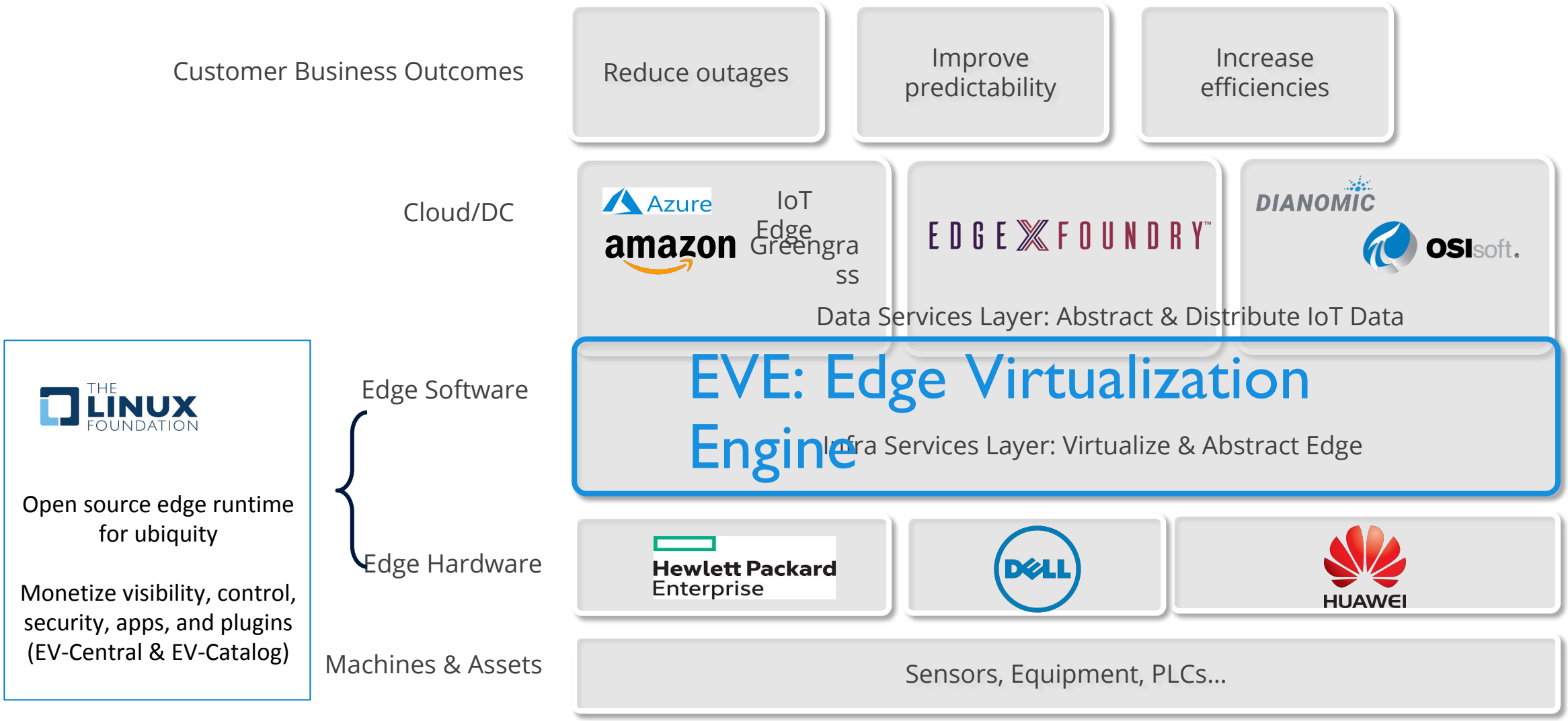
IIoT 1.0: Vertical data silos & platform lock-in
Data/edge sovereignty & control issues
Hardware-defined & unmanaged edge



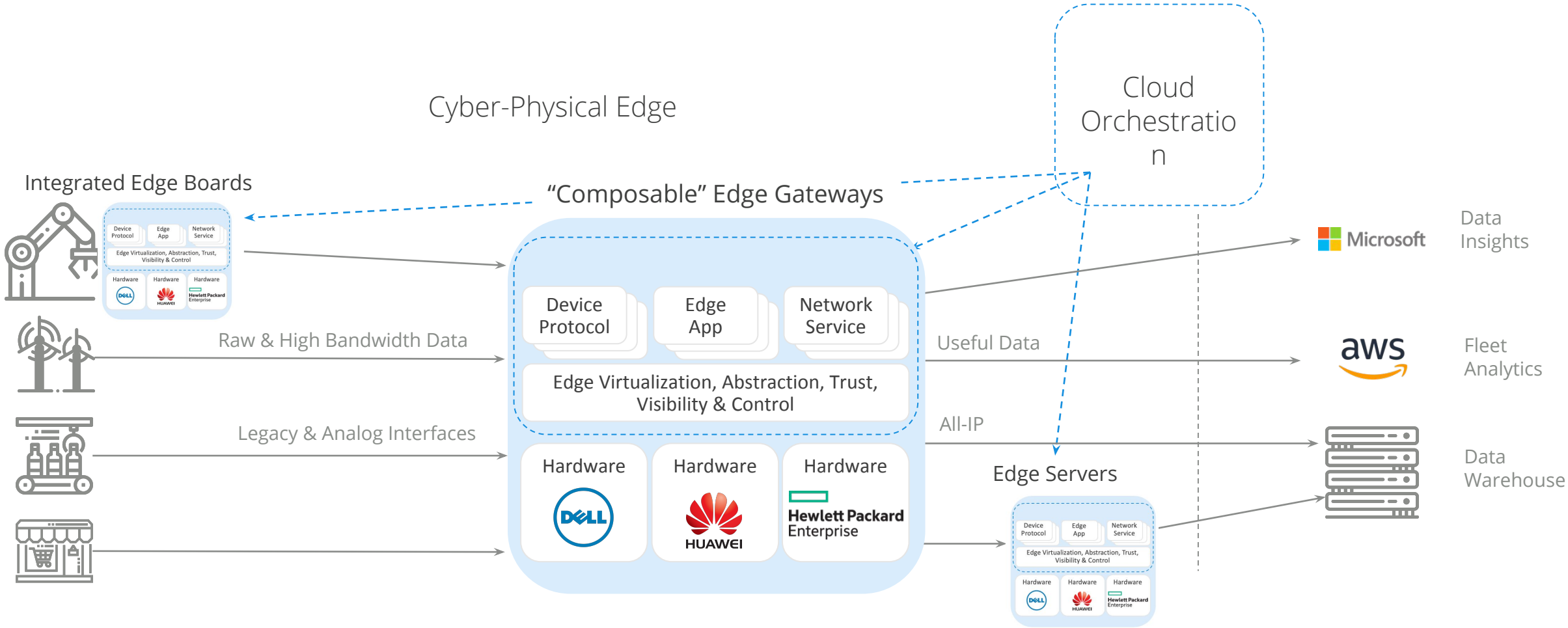
IIoT 2.0: Open IoT data architecture, no lock-in
Data & edge belong to the enterprise
Software-defined & ubiquitous edge



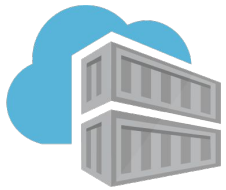
The Enterprise Cyber-Physical Edge Stack



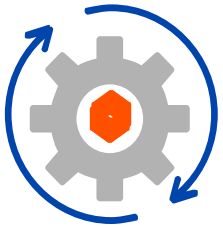
The virtualized, software-defined & composable edge



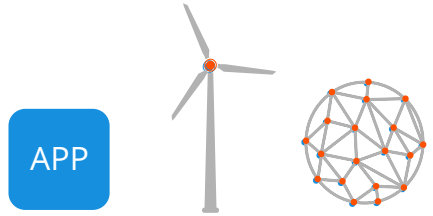
Key Requirements



EDGE CONTAINERS



ZERO TOUCH



ANY

APP | HARDWARE | NETWORK

ZERO TRUST

Enterprise Edge challenges

- Very distributed deployment - location is key
  - Minimize device installation and onboarding effort; maximize security
  - Never visit for software issues; have no separate management connectivity
  - Limited physical security; can't assume firewalls
- Long lifecycle - patch applications and runtime for 7+ years
- Rich downstream connectivity
  - Legacy serial ports, Industrial Ethernet, various radio technologies
- Diverse upstream connectivity - might need redundancy
  - Ethernet, LTE, etc
  - Might not control network; NATs, proxies etc deployed by someone else
- Yet want same/similar applications and devops as in cloud
  - Apps shouldn't need to worry about above differences

Zero Touch

- Enabling drop ship to installer
  - Factory/supply chain installs EVE; handles unique device identity
  - Installer connects power and network/serial cables
  - Feedback to installer that device connected to cloud
  - Everything else done from the cloud
- Edge container lifecycle (install, update, pause, snapshot)
- Device lifecycle (EVE patch/update, EVE connectivity changes)
  - Without any risk of turning the device into a brick
- Only broken hardware or cabling changes requires touching the device

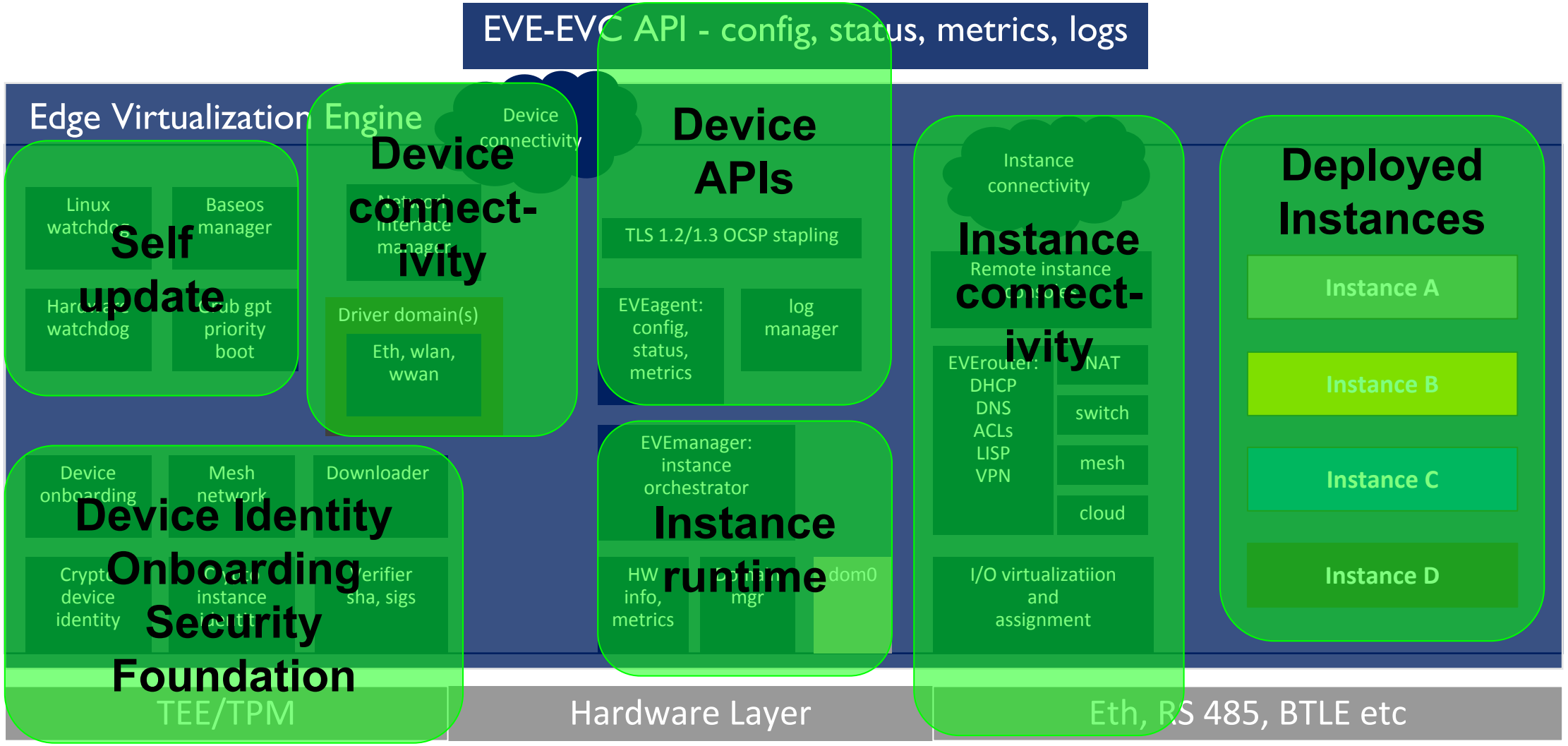
Any/Freedom - Edge Virtualization

- EVE today support ARM and Intel/AMD
  - Requires processor support for type 1 hypervisor
- Supports a range of upstream and downstream IP connectivity
  - Ethernet, WiFi, LTE, plus anything else supported by Linux
- Supports a range of downstream I/O connectivity
  - RS-232, RS-485 serial ports
  - USB, Audio, etc
- Runs any application
  - Existing VMs, VMs with container runtimes (Azure IoT Edge, AWS Greengrass Core)
  - Applications are not concerned with the variations in IP connectivity

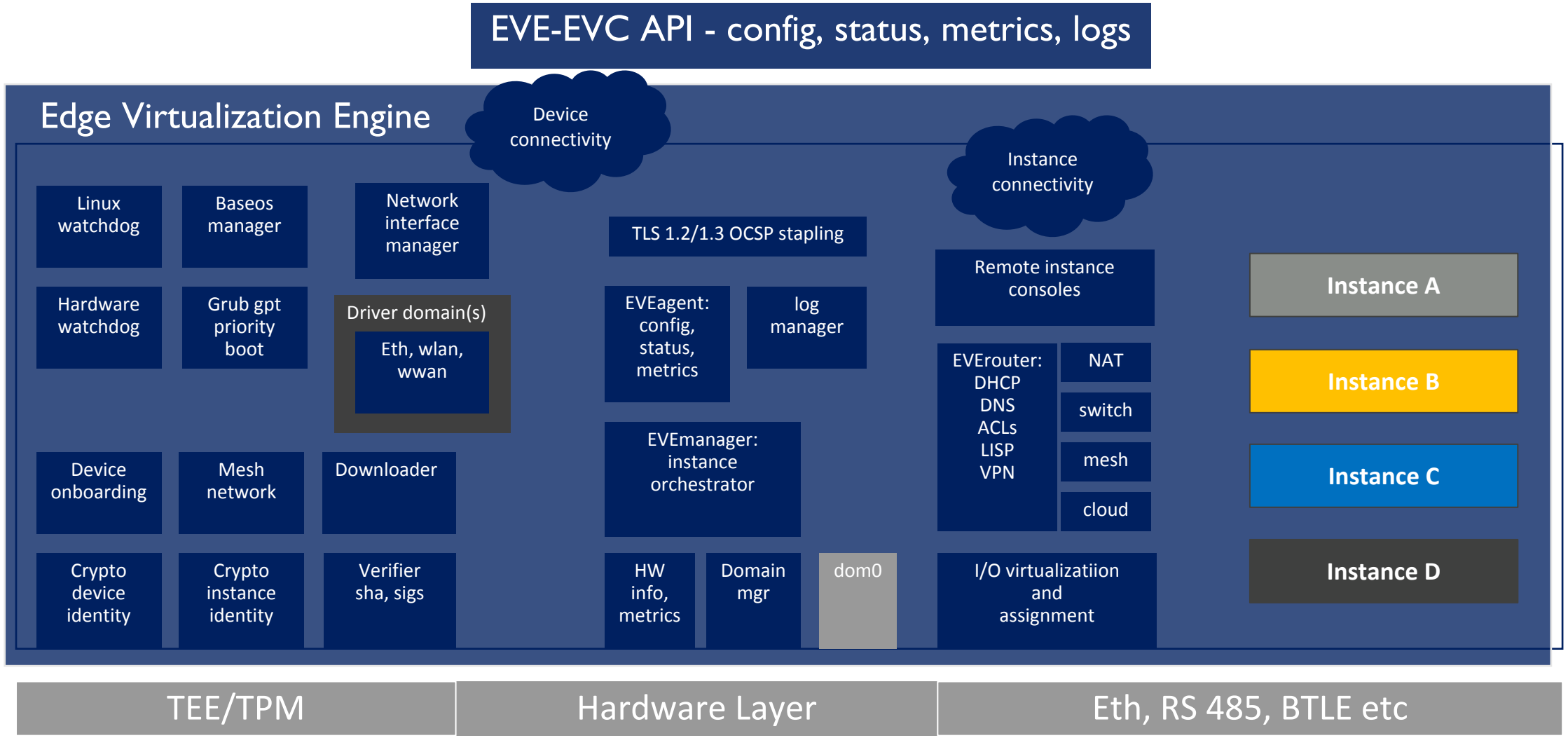
Zero Trust

- Sole device access is through EVE API
  - No usernames and passwords(*)
  - Trust expressed in the form of a root CA certificate
- Strong device identity
  - 20 year lifetime ECC device certificate with private key in TPM
  - Onboarding associates device certificate with enterprise/user
- Data in transit and data at rest protected using state of the art
- Defense in depth plus logging; detect and correct before damage
- Applications can be observed and fenced
  - ACLs in application manifest are enforced and logged by EVE

Project EVE Architecture



Project EVE Architecture



App Instance Connectivity

- Default is local network with NATed connectivity
- Can provision a switch network - an L2 network e.g, on eth1
- Can provision USB 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 Device 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 for 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
  - POST api/v1/edgedevice/flowlog for app network flows logs (new)
- All messages encoded using protobuf

Register API

https://github.com/lf-edge/eve/blob/master/api/proto/register/register.proto

- Used in some onboarding scenarios
  - A device starts out with a (single use) onboarding token
  - Device creates device certificate using its TPM on first boot
  - Register API binds the device certificate to enterprise/user via token
- POST to /api/v1/edgeDevice/register
- Simple message:

```
message ZRegisterMsg {
        bytes pemCert = 2;
        string serial = 3;
        string softSerial = 4;
}
```

Config API

https://github.com/lf-edge/eve/blob/master/api/proto/config/devconfig.proto

- All of the configuration from the cloud to the device
  - Device configuration
  - App instance configuration
  - (Local) network instance configuration for apps
  - Datastores (for images and other objects)
- GET to /api/v1/edgeDevice/config
- Typically used on boot and periodically to check for updates

Config: Device parts

- PhysicalIO - describe the physical (networking, USB, etc) ports on device
- SystemAdapter - describe the device' desired IP+DNS+proxy configuration
- NetworkConfig - referenced by SystemAdapter for static IP and proxy
- DeviceListDetails - specify parameters for LISP overlay network
- BaseOsConfig - specify update of EVE itself
- DevOpsCmd - specify device reboot etc
- ConfigItem - open-ended way to specify e.g., timers, debug settings, etc

Config: App Instance parts

- DataStoreConfig - from where to fetch images (S3, https, sftp)
- NetworkInstanceConfig - virtual networks for the apps
- Each app instance includes:
  - CPU, memory requirements
  - Virtual disks, including SHA and signatures of images
  - Physical Adapter assignment (e.g., USB and serial ports)
  - Virtual Adapter assignment (virtual Ethernets)
    - Attachment to the network instances
    - Includes Access Control Lists with optional rate limits
  - UserData for cloud-init
  - Enabling of remote console

Info API

https://github.com/lf-edge/eve/blob/master/api/proto/info/info.proto

- Reports sent on state change for device or for app instance
- Device:
  - Hardware and BIOS info (serial numbers, versions, TPM info)
  - Status of EVE version and any version update in progress
  - Resource usage (CPU, memory, disk, physical adapters)
  - Network status (IP, DNS, tried/failed information)
- Application instance:
  - Up/down, boot time
  - Image download status and progress
  - Virtual network status

Metrics API

https://github.com/lf-edge/eve/blob/master/api/proto/metrics/metrics.proto

- Sent periodically by device; cumulative counters
  - Missed messages merely result in reduced time resolution
- Device metrics:
  - CPU, memory, disk, network usage
- App instance metrics:
  - CPU, memory, disk, network usage
  - ACL violations; network rate limits exceeded
- Network instance metrics:
  - Including VPN and LISP counters

Log API

https://github.com/lf-edge/eve/blob/master/api/proto/logs/log.proto

- Device logs for debugging
  - Amount of logging can be controlled with a configItem
- Logs are bundled in a LogBundle and sent when full or after timeout
- POST to /api/v1/edgeDevice/logs

Flowlog API

https://github.com/lf-edge/eve/blob/master/api/proto/flowlog/flowlog.proto

- For application network flows
  - Accepted
  - Dropped
  - Plus hostname to IP address mappings as seen by device
- Brand new; implementation underway
- POST to /api/v1/edgeDevice/flowlog

More info

https://www.lfedge.org/projects/eve/

https://github.com/lf-edge/eve

https://github.com/lf-edge/eve/tree/master/api

Questions?