

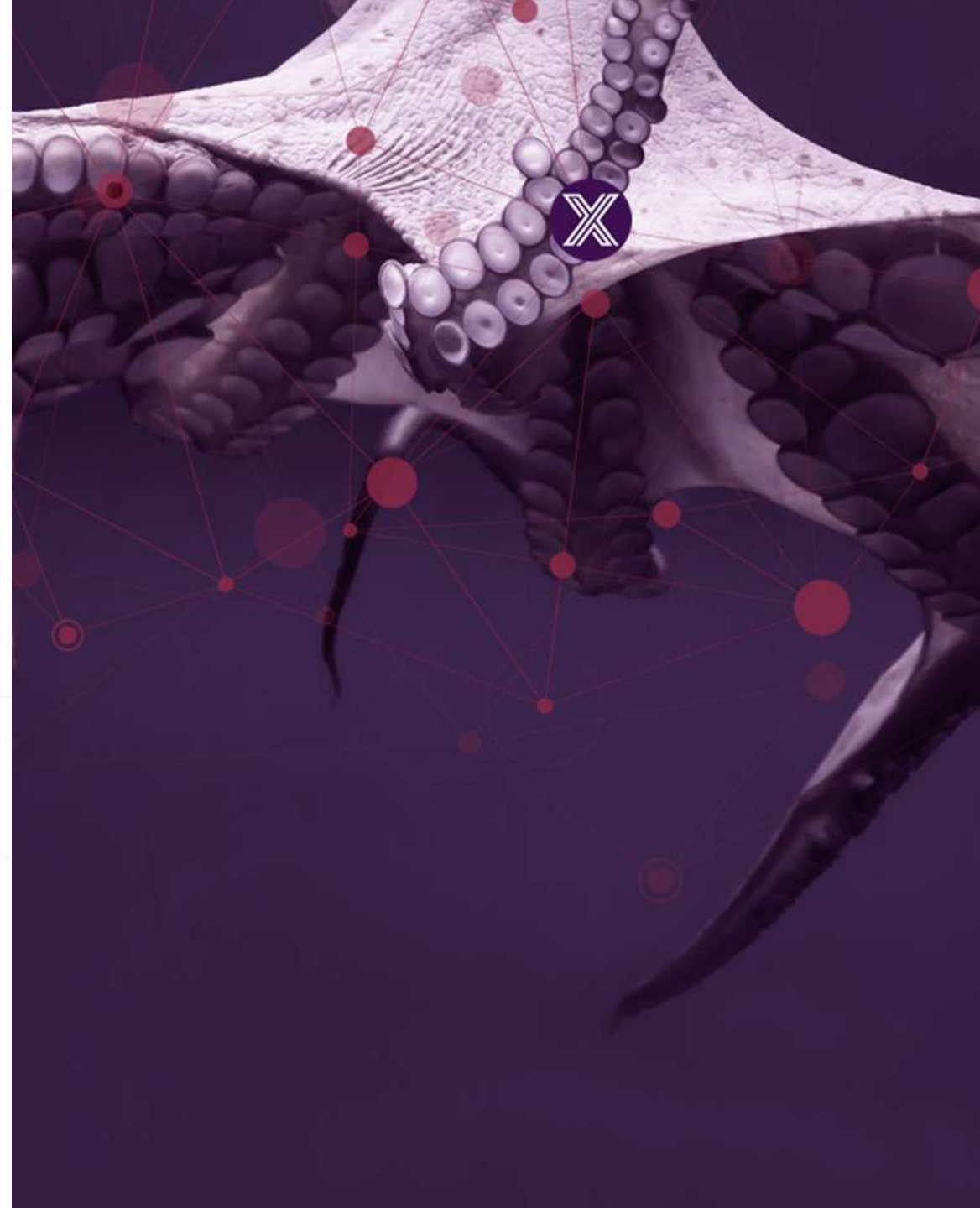


# Getting Started With EdgeX Foundry

Jim White

Dell Technologies

October 2018



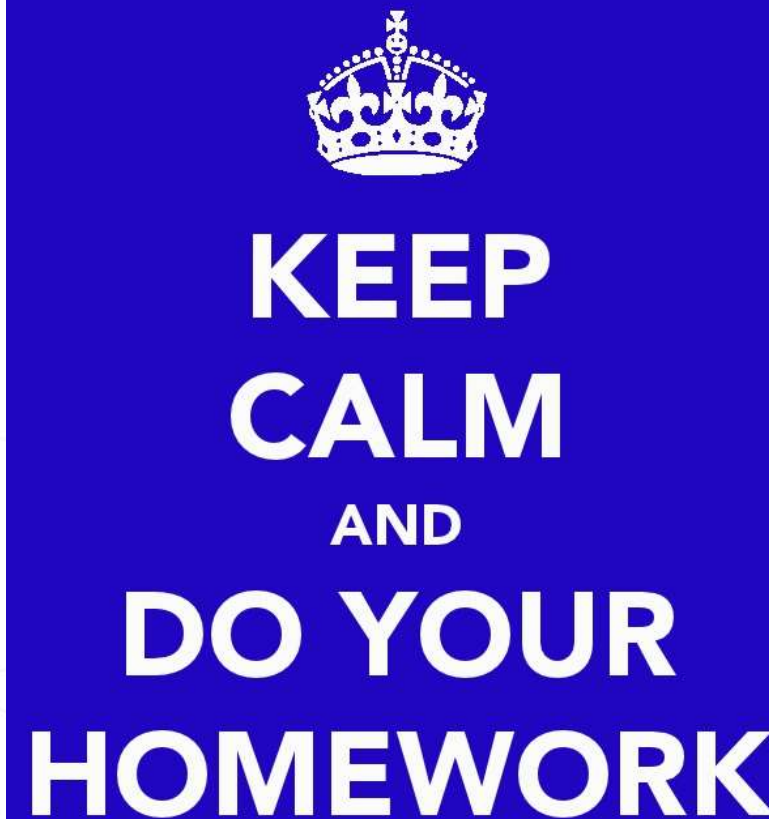
# Workshop requirements

- There are a few assumptions about this workshop
  - You have a laptop
  - You have installed Docker and Docker Compose
  - You have some type of text editor
    - There won't be a lot of editing
  - You are familiar with basic software development principals
    - Example: you know it means to issue a command on the CLI or terminal window



# Post Workshop

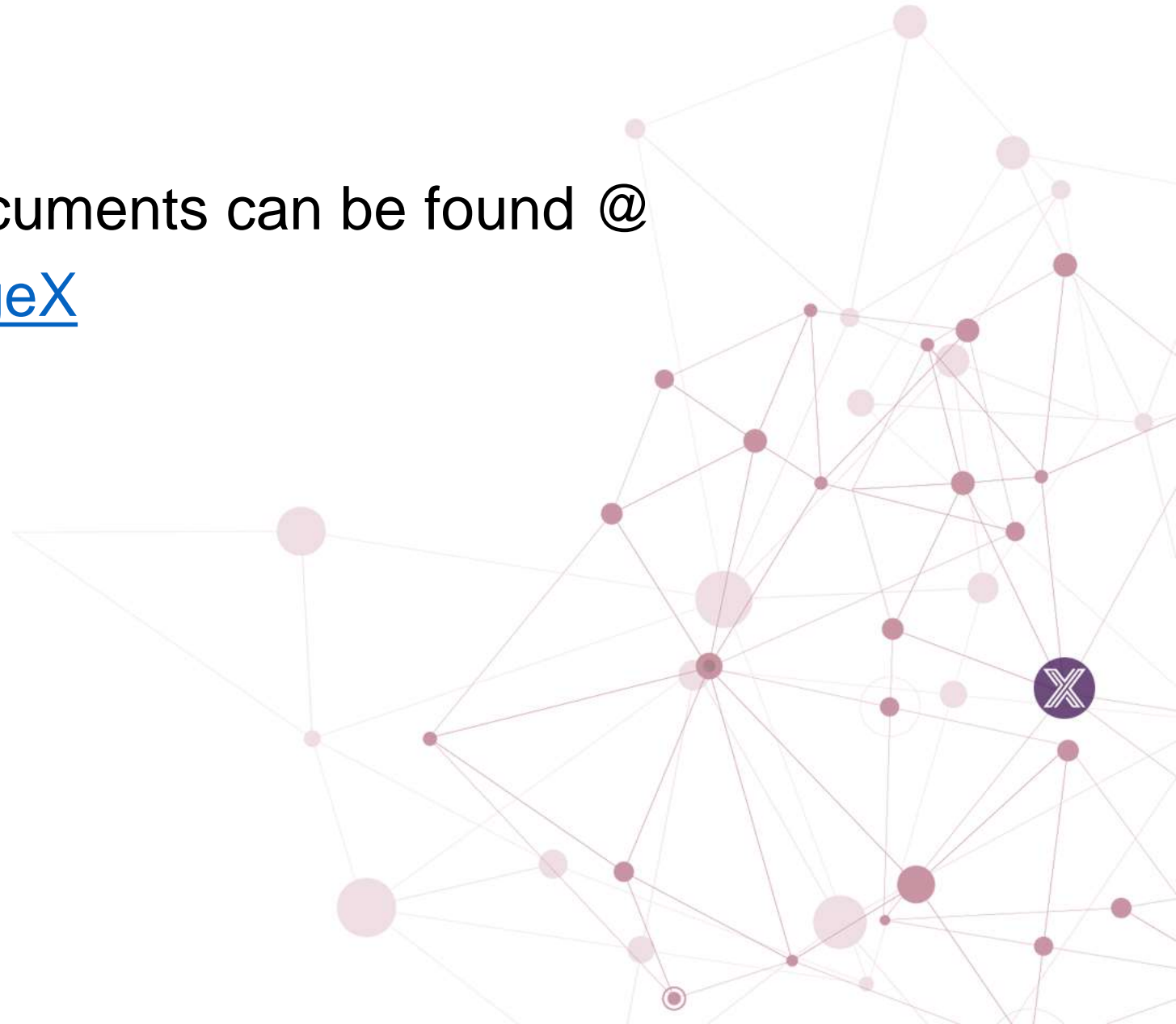
- There is a lot of material here!
  - We won't get through all of it in this ~2 hour session
  - This session will get you started and give you a good EdgeX orientation
  - The additional material and “labs” will get you ready to use or contribute to EdgeX



**KEEP  
CALM  
AND  
DO YOUR  
HOMEWORK**

# Session materials

These slides and the lab documents can be found @  
<http://bit.ly/OSS-Europe-EdgeX>



# About Me



- Jim White (james\_white2@dell.com)
  - Dell Technologies IoT Solutions Division – Distinguished Engineer
  - Team Lead of the IoT Platform Development Team
  - Chief architect and lead developer of Project Fuse
    - Dell's original IoT platform project that became EdgeX Foundry
    - Yes – I wrote the first line(s) of code for EdgeX (apologies in advance)
  - EdgeX Foundry ...
    - Vice Chairman, Technical Steering Committee
    - Systems Management Working Group Chair
    - Ad hoc and unofficial lead architect



# Session Agenda

- Understand what EdgeX is
- Learn how to get and setup EdgeX
- Explore the EdgeX micro services
- Learn how to connect sensors and devices to EdgeX
- See how to connect EdgeX to cloud and enterprise systems
- Understand EdgeX deployment options
- Examine how to customize and extend EdgeX
- Introduce the EdgeX community and how to get involved



# Part 1

- Get a general sense of what EdgeX Foundry is and how it works
- Understand the motivation behind EdgeX
- Explore the basic architecture of EdgeX
- Learn the architectural tenets underlying EdgeX

# Introducing EdgeX Foundry

An open source, vendor neutral project (and ecosystem)

A **micro service**, loosely coupled software framework for IoT edge computing

Hardware and OS agnostic

Linux Foundation, Apache 2 project

Goal: enable and encourage growth in IoT solutions

- The community builds and maintains common building blocks and APIs
- Plenty of room for adding value and getting a return on investment
- Allowing best-of-breed solutions





# EdgeX Foundry Goals

- Build and promote EdgeX as the common open platform unifying edge computing
- Enable and encourage the rapidly growing community of IoT solutions providers to create an ecosystem of interoperable plug-and-play components
- Certify EdgeX components to ensure interoperability and compatibility
- Provide tools to quickly create EdgeX-based IoT edge solutions
- Collaborate with relevant open source projects, standards groups, and industry alliances to ensure consistency and interoperability across the IoT

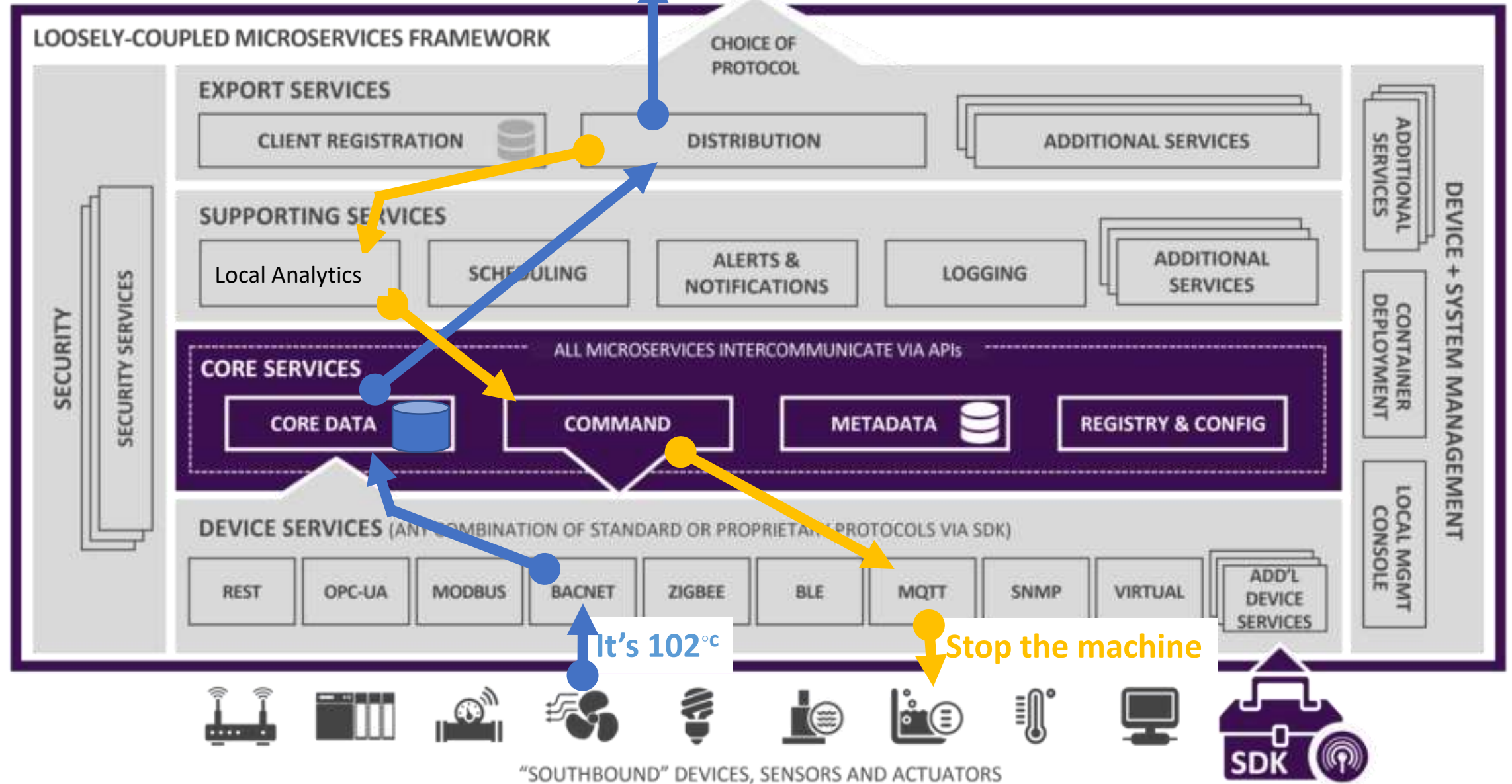
# A Brief EdgeX History

- Chartered by Dell IoT marketing in July 2015
  - A Dell Client CTO incubation project (Project Fuse)
- Designed to meet interoperable and connectivity concerns at the IoT edge
- Started with over 125,000 lines of Dell code
- Entered into open source through the Linux Foundation on April 24, 2017
  - Started with nearly 50 founding member organizations; today we have more than 75
- Release Cadence: 2 formal releases a year



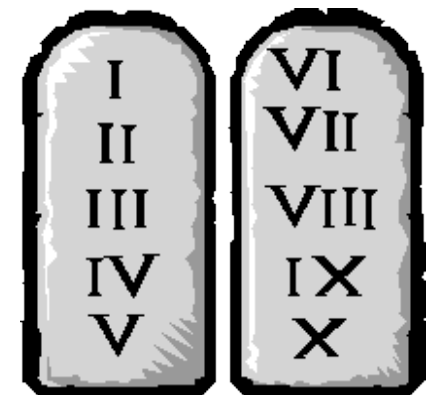
# EdgeX Primer - How it works

- A collection of a dozen+ micro services
  - Written in multiple languages (Java, Go, C, ... we are polyglot believers!!)
- EdgeX data flow:
  - Sensor data is collected by a **Device Service** from a thing
  - Data is passed to the **Core Services** for local persistence
  - Data is then passed to **Export Services** for transformation, formatting, filtering and can then be sent “north” to enterprise/cloud systems
  - Data is then available for edge analysis and can trigger device actuation through Command service
  - Many others services provide the supporting capability that drives this flow
- REST communications between the service
  - Some services exchange data via message bus (core data to export services and rules engine)
- Micro services are deployed via Docker and Docker Compose



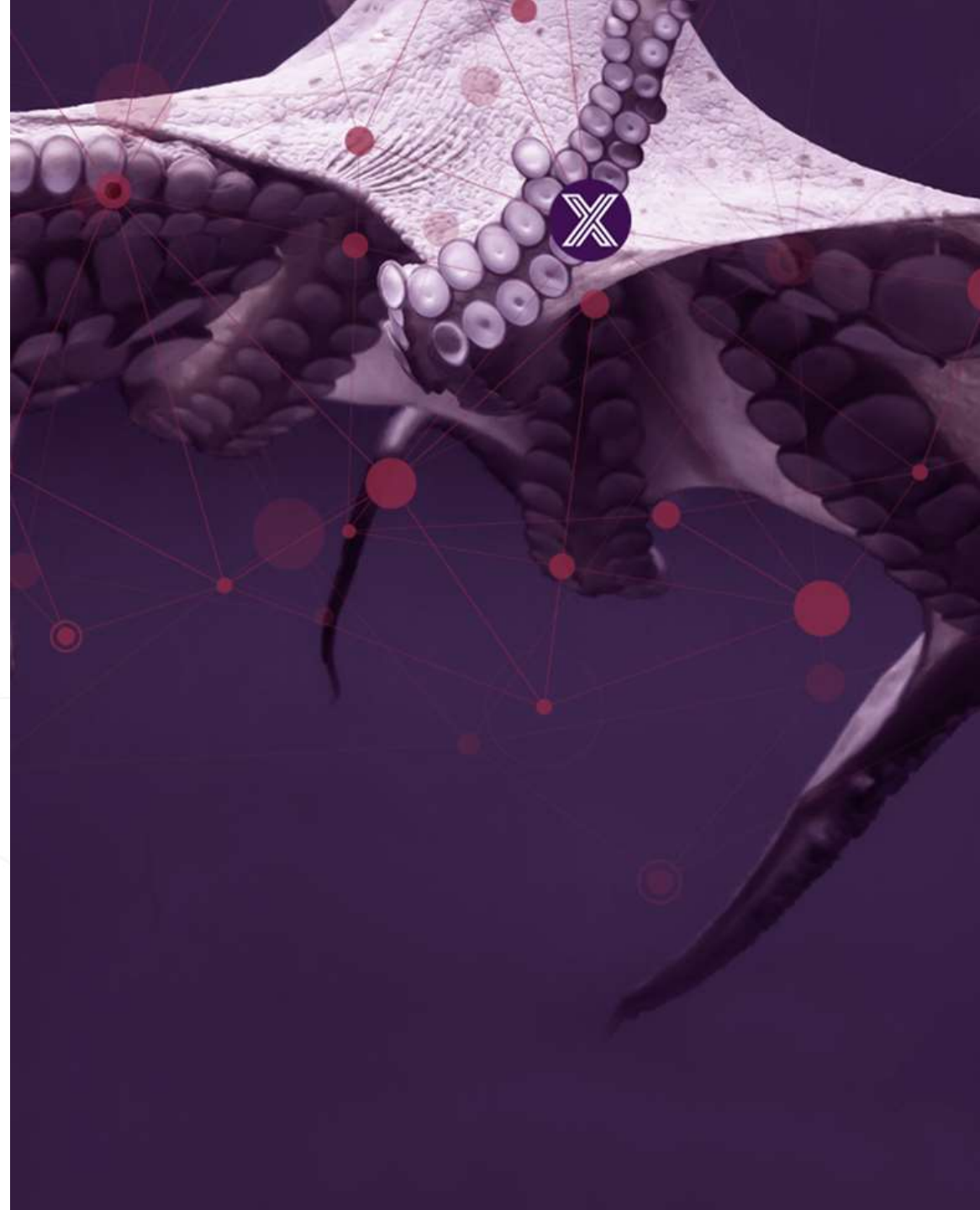
# EdgeX Architectural Tenets

- EdgeX Foundry must be **platform agnostic** with regard to hardware, OS, distribution/deployment, protocols/sensors
- EdgeX Foundry must be **extremely flexible**
  - Any part of the platform may be upgraded, replaced or augmented by other micro services or software components
  - Allow services to scale up and down based on device capability and use case
- EdgeX Foundry should provide “reference implementation” services but **encourages best of breed solutions**
- EdgeX Foundry must provide for **store and forward** capability (to support disconnected/remote edge systems)
- EdgeX Foundry must support and **facilitate “intelligence” moving closer to the edge** in order to address
  - Actuation latency concerns
  - Bandwidth and storage concerns
  - Operating remotely concerns
- EdgeX Foundry must **support brown and green device/sensor** field deployments
- EdgeX Foundry **must be secure and easily managed**



# EDGE X FOUNDRY™

## EdgeX Foundry Demo





# Part 2

- See the options with regard to getting, working with, and running EdgeX
- Learn why EdgeX uses Docker for deployment
- Understand the role of Docker and Docker Compose in deploying EdgeX
- Explore the EdgeX Docker Compose file

# Options to Getting & Running EdgeX

- EdgeX micro services can be built and deployed in a number of ways
  - “Contributors Approach”
    - Get the raw code, build it, and deploy the services to the target platform(s)
  - “Users Approach”
    - Get EdgeX Docker container images and deploy/run to a platform where Docker is installed
  - “Hybrid Approach”
    - Get, build and deploy some of the services on your own
    - Get and use Docker container images for the other services
  - Docker Compose is a tool to help get and run multiple containers
    - Docker Compose can be used with either the User or Hybrid approaches
- We are going to focus on the User Approach today
  - EdgeX Foundry Code is available at GitHub (<https://github.com/edgexfoundry/>)
  - See the EdgeX Getting Started guides for more directions on the contributor and hybrid approaches

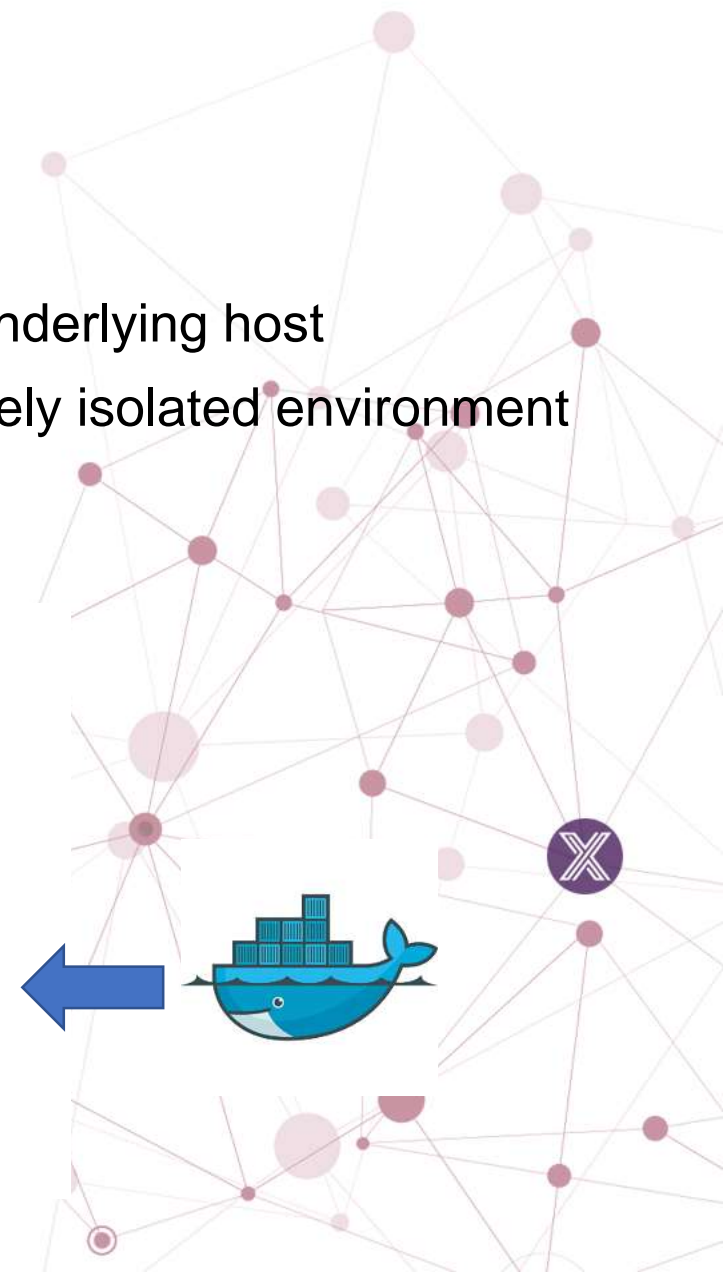
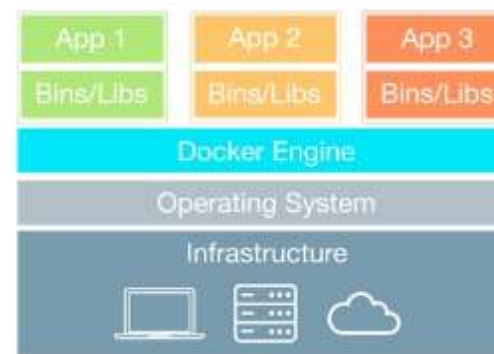
# User Approach to Get & Run

- The EdgeX community provides a Docker container image for each micro service (and underlying infrastructure such as the database)
  - This convenience allows users to quickly get pre-built EdgeX micro services
  - Because the container images have all the necessary environment (OS, configuration, etc.) for the micro services, it makes deploying EdgeX easier
  - The container images can be run on any platform that runs Docker
    - There are different container images for hardware platforms (Intel or Arm)
- The EdgeX Docker container images are available in Docker Hub ([hub.docker.com](https://hub.docker.com))
  - The most recent code is always built to “developer” container images
  - These are made available from a Linux Foundation Nexus repository
  - These should only be used when you need the latest developer work



# What is Docker?

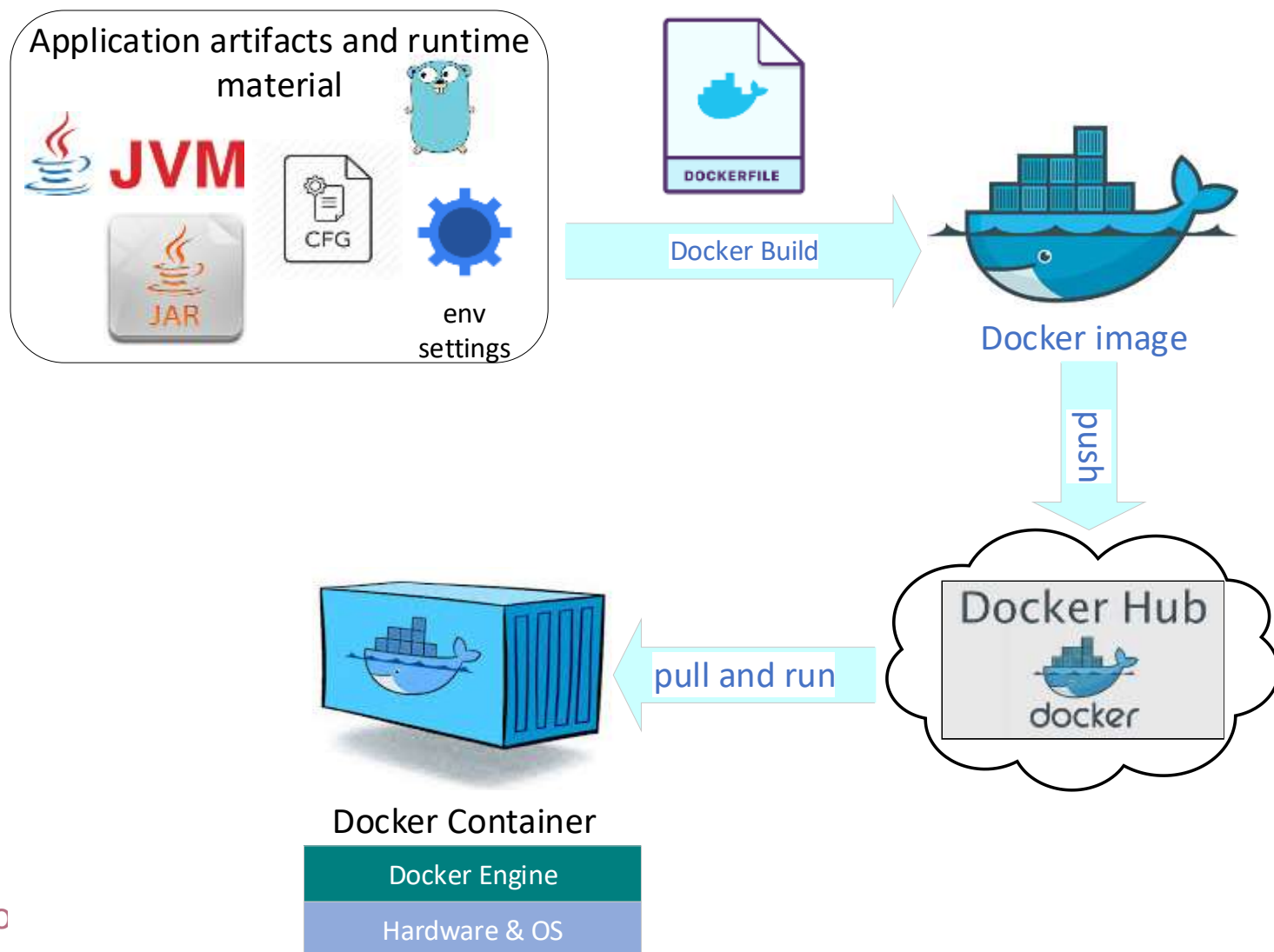
- Docker is a bit like virtualization
  - ... but allowing some elements (like OS) to be obtained from the underlying host
- Docker provides the ability to package and run an application in a loosely isolated environment called a **container**
- Many containers can run simultaneously on a given host



# Docker Terminology

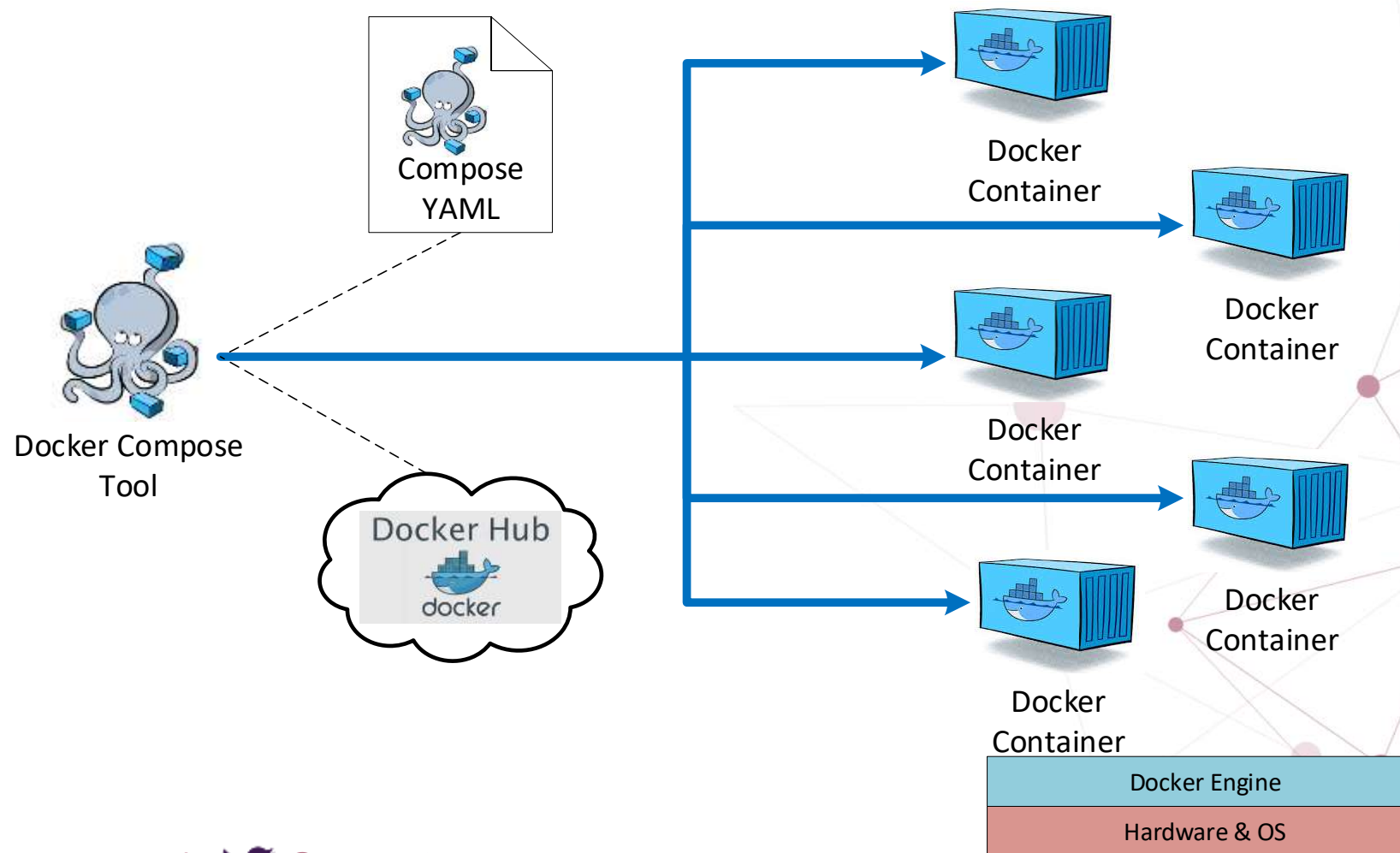
- Docker Engine: is the underlying client-server technology that builds and runs containers
- Docker Image: a package of application “bits” and all its dependencies, information, etc. needed to create a container
  - An image is to a container what a class is to an object
- Docker Container: an instance of a Docker image or execution of a single application
  - The container is created from the image
- Docker Build: act of creating the container image from the Dockerfile and bits of the application
- Dockerfile: instructions (in a text file) to the build facility on how to create the Docker image and what to include in the image
- Docker Repo: a repository of Docker images (could be private or public)
- Docker Hub: the Docker managed repository of Docker images (allows for public or private images)
- Orchestration: a tool that helps manage, build and deploy many Docker containers to multiple hosts
  - Kubernetes is an example of an orchestration tool

# Docker Images, Containers, Engine





# User Approach to Get & Run With Docker Compose



# The EdgeX Docker Compose YAML

version: '3'

services:

volume:

image: edgexfoundry/docker-edgex-volume

container\_name: edgex-files

networks:

- edgex-network

volumes:

- /data/db
- /edgex/logs
- /consul/config
- /consul/data

...

logging:

image: edgexfoundry/docker-support-logging

ports:

- "48061:48061"

container\_name: edgex-support-logging

hostname: edgex-support-logging

networks:

- edgex-network

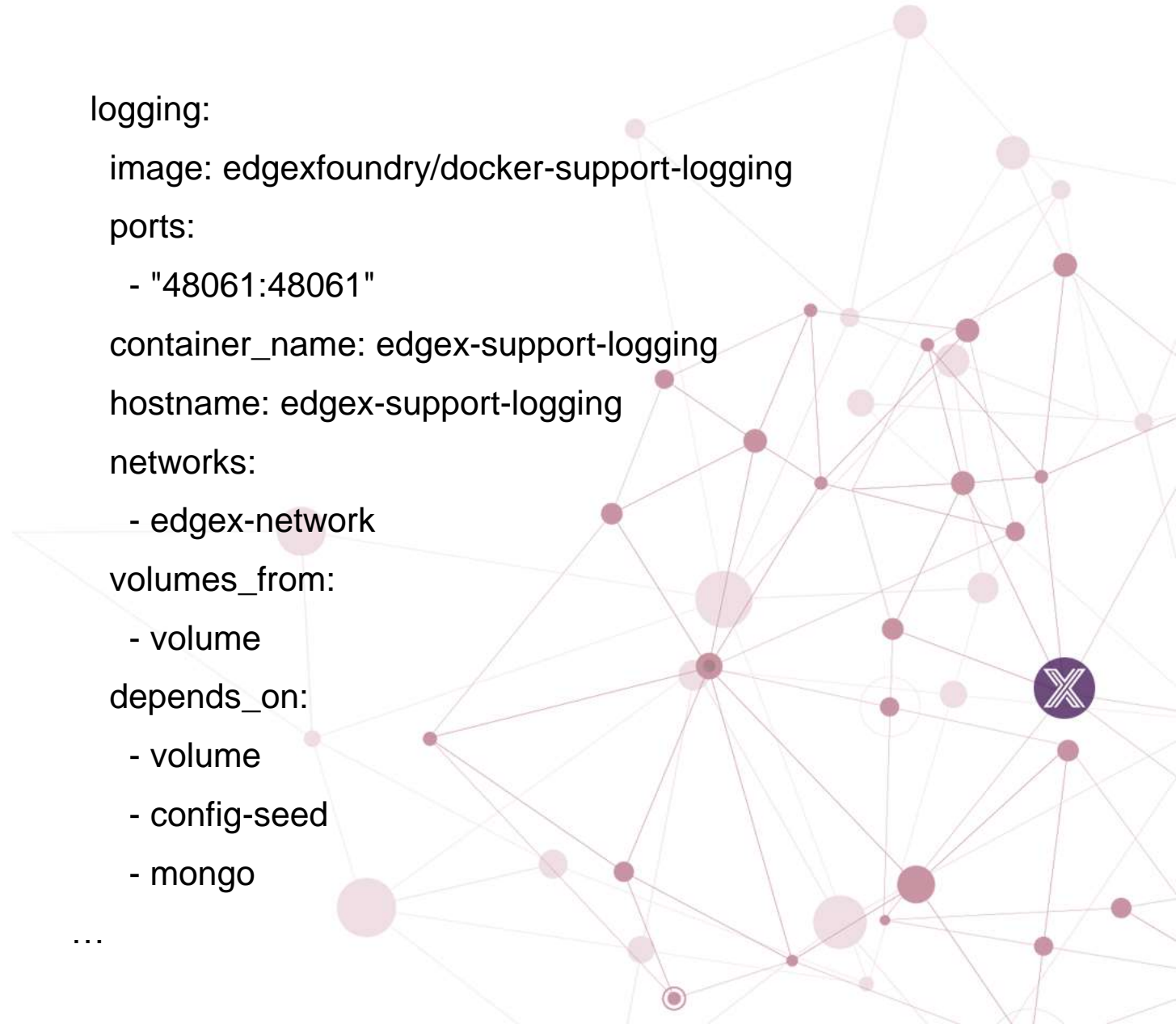
volumes\_from:

- volume

depends\_on:

- volume
- config-seed
- mongo

...



# Explore the EdgeX Docker Compose File

- The Docker Compose YAML is a manifest file
  - It specifies to Docker ...
    - What containers to pull down and start
    - What infrastructure (like a network) is needed for your containers
    - The order in which to start/stop containers
    - ...
- In class exercise: use a browser
  - Go to <https://github.com/edgexfoundry/developer-scripts>
  - Click on compose-files folder
  - Click on the docker-compose-california-0.6.X.yml file



# Docker Compose Commands

- Docker Compose is a command line tool
- Common commands

`docker-compose pull -f <compose file name>`

*pull the images but don't start them*

`docker-compose -f <compose file name> up -d`

*create and start all containers – the default compose file name is docker-compose.yml*

*pull the images if not already pulled*

*-d means to start them all as daemon processes*

`docker-compose -f <compose file name> up <docker image> -d`

*without the image name, all containers are brought up*

`docker-compose -f <compose file name> stop <docker image>`

*stop an existing container*

*stop all images if the image name is left off*

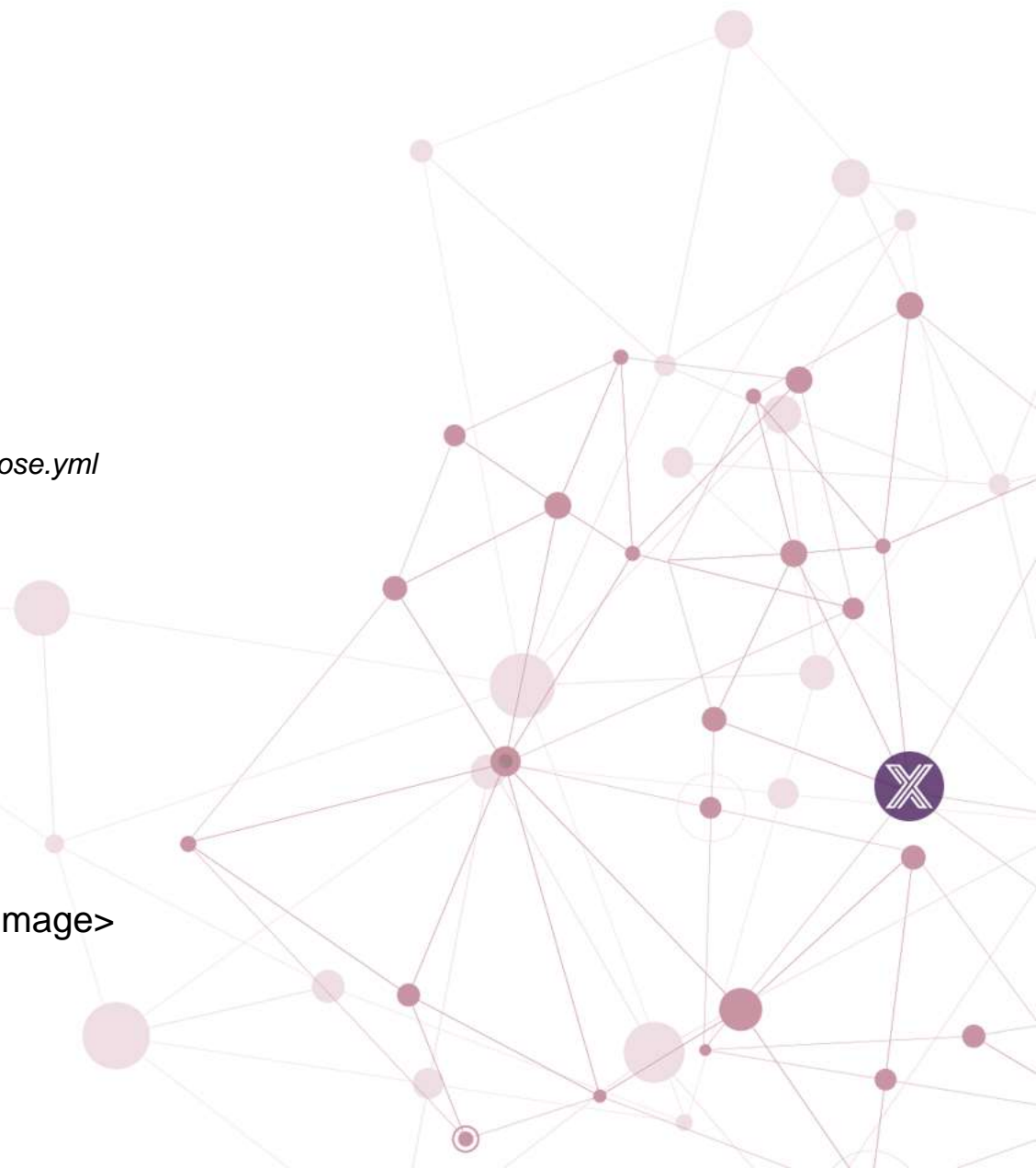
`docker-compose -f <compose file name> start <docker image>`

*start an existing container that has been stopped*

*start all images if the image name is left off*

`docker-compose -f <compose file name> logs -f --tail=100 <docker image>`

*look at the last 100 lines of a micro services logs*



# The EdgeX Containers

- Depending on the version of EdgeX as well as your use case, the EdgeX Docker Compose file will list several EdgeX containers

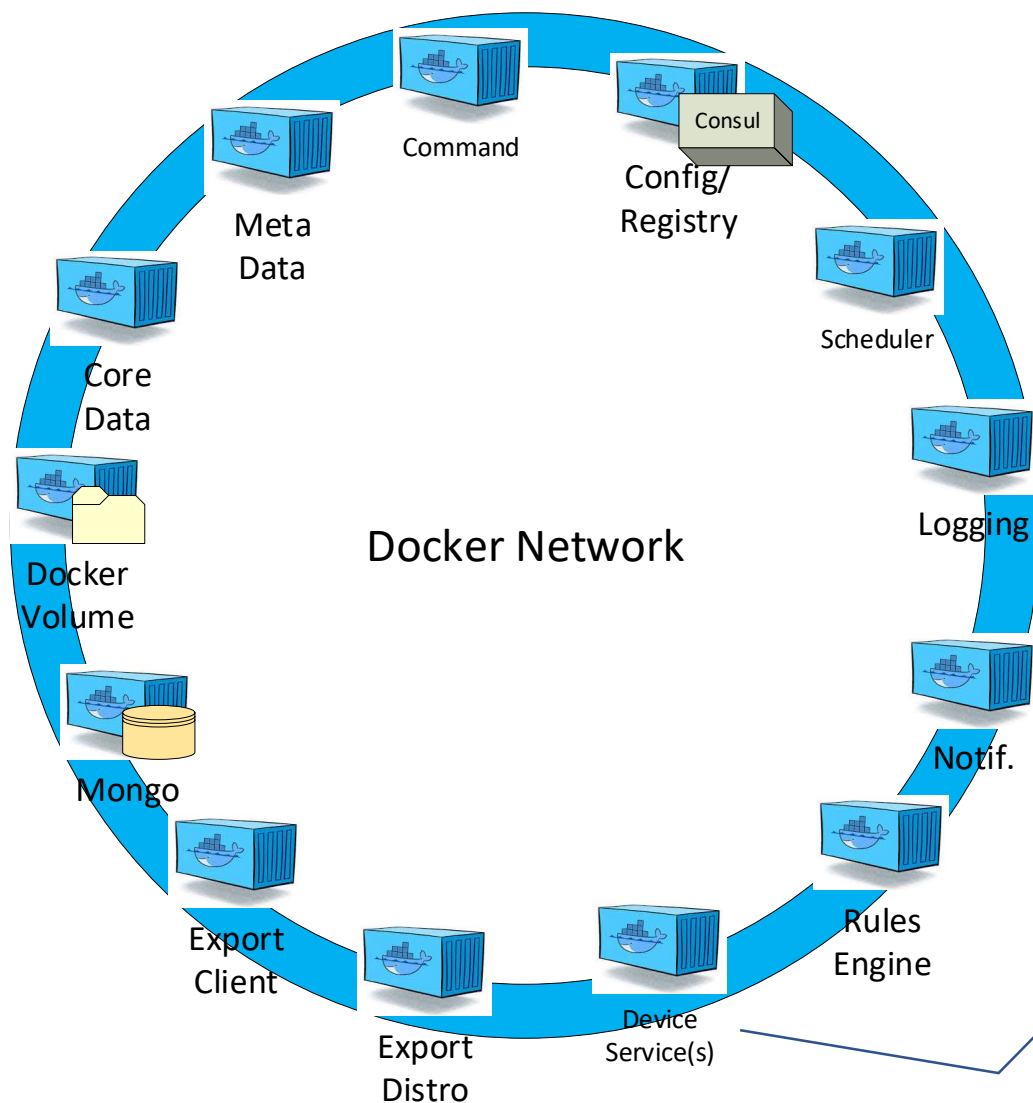
Container	Purpose
mongo	Mongo Database instance, and data initialization for the default NoSQL database for all of EdgeX
consul	Hashicorp's Consul configuration and registry service
data	Core Data, centralized persistence facility for data readings collected by devices and sensors
metadata	Core Metadata, knowledge about the devices and sensors and how to communicate with them
command	Core Command, enables the issuance of commands or actions to devices and sensors on behalf of other micro services, other applications, external systems
scheduler	Support Scheduling, provides facilities to kick off various events/actions on a timed schedule such as old data scrubbing
logging	Support Logging, central logging service for all micro services
notifications	Support Notifications, central alert and notification service for all micro services
rulesengine	Support Rules Engine, micro service "wrapped" Drools Rules Engine that monitors incoming sensor or device data for readings within target ranges and triggers immediate device actuation
export-client	Export Client, enables clients, whether they are on-gateway or off-gateway, to register as recipients of data coming through Core Data
export-distro	Export Distribution, receives data from Core Data, through a message queue, then filters, transforms, and formats the data per client request, and distributes to the appropriate endpoint by pre-registered protocol

# EdgeX Infrastructure

- EdgeX micro services won't be the only thing listed and brought up with Docker Compose
- EdgeX relies on a shared file space among services (called a Docker volume)
  - Allows the database files to be shared across services
  - Allows log file space to be shared across services
- EdgeX use MongoDB as its default persistence storage
  - Mongo has been containerized for EdgeX use
- EdgeX uses Consul as its registry and configuration service
  - Consul has been containerized for EdgeX use
- EdgeX config-seed is a service that initializes Consul with EdgeX configuration data
  - config-seed exits quickly after populating Consul (i.e. it is not long running)
- EdgeX needs all micro services to be connected to a virtual network
  - Docker provides a virtual network facility
  - The Docker Compose file specifies the network and includes all the services and infrastructure on that network



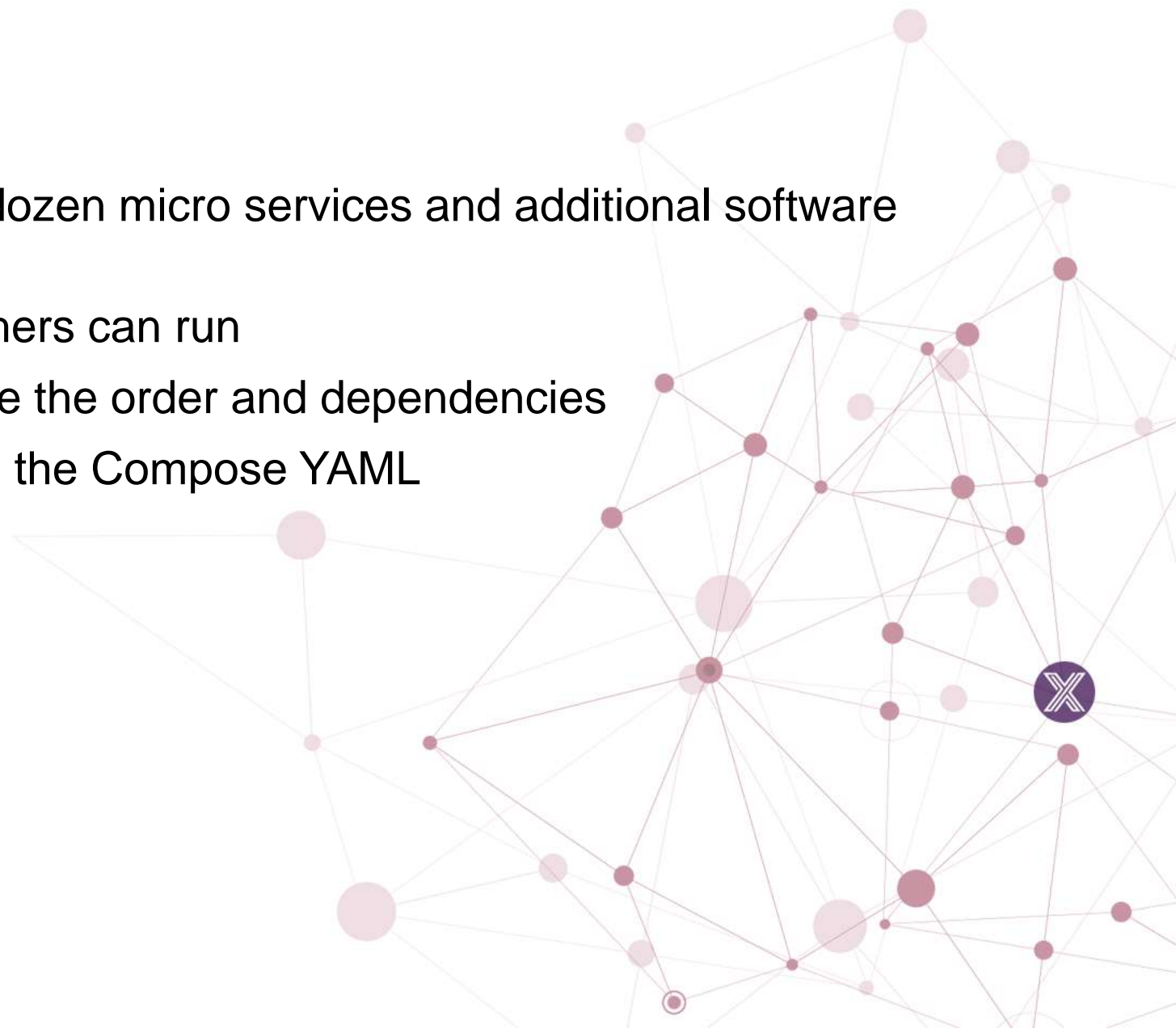
# The Typical EdgeX Deployment



Number of device services will vary per use case and number of sensors/devices

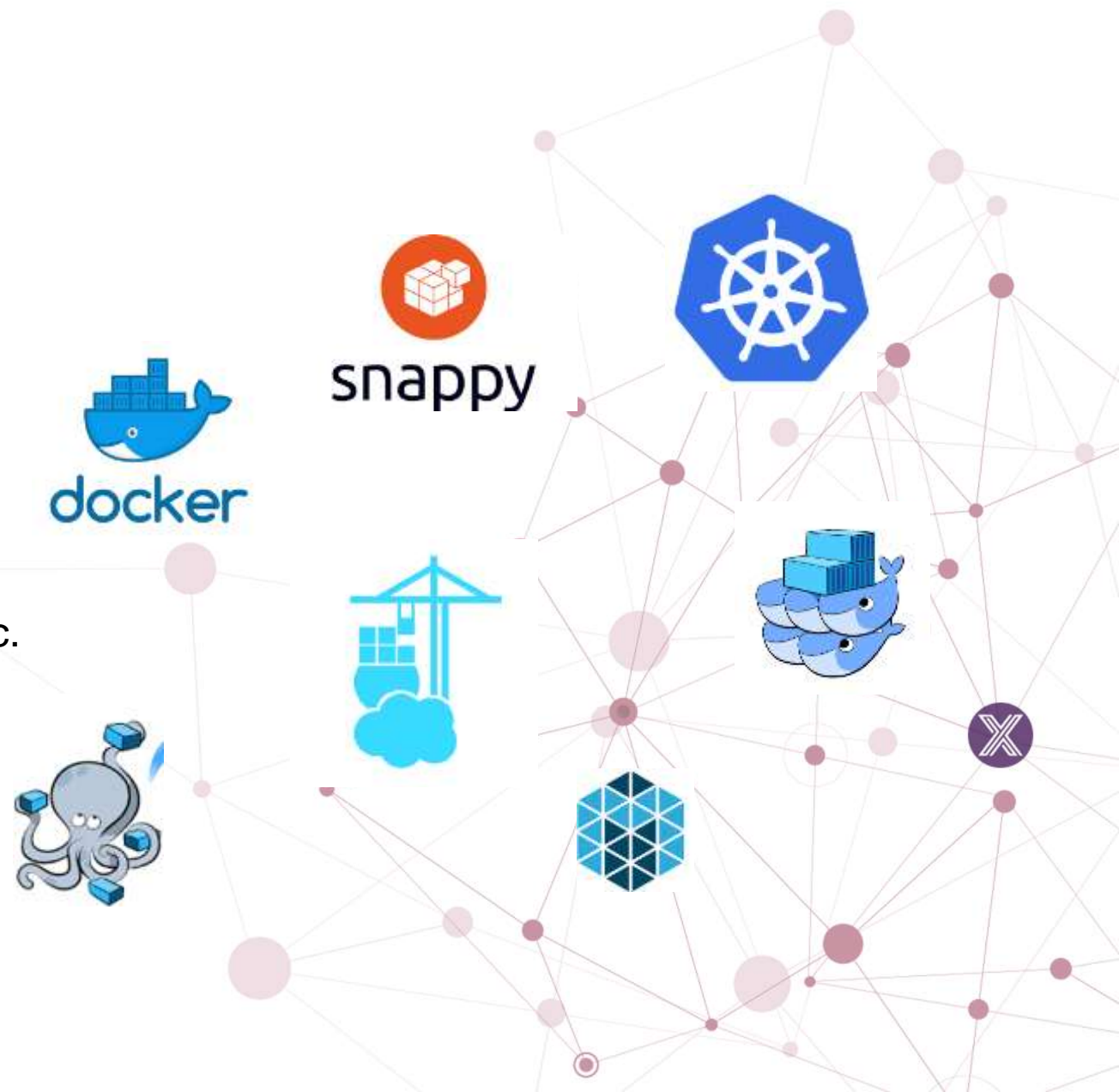
# EdgeX Dependencies

- EdgeX is comprised of more than a dozen micro services and additional software infrastructure (like the database)
- Some services must be up before others can run
- Docker Compose helps to orchestrate the order and dependencies
- Note the “dependencies” elements in the Compose YAML



# Alternate Deployments

- EdgeX is agnostic with regard to deployment (and orchestration)
  - Docker is used as our reference deployment capability
  - We also provide Ubuntu Snaps
    - For Canonicals' Ubuntu Core
  - Members of our community use Portainer to deploy and orchestrate EdgeX
  - You could use Kubernetes, Swarm, Mesos, etc.
- EdgeX is BYOD

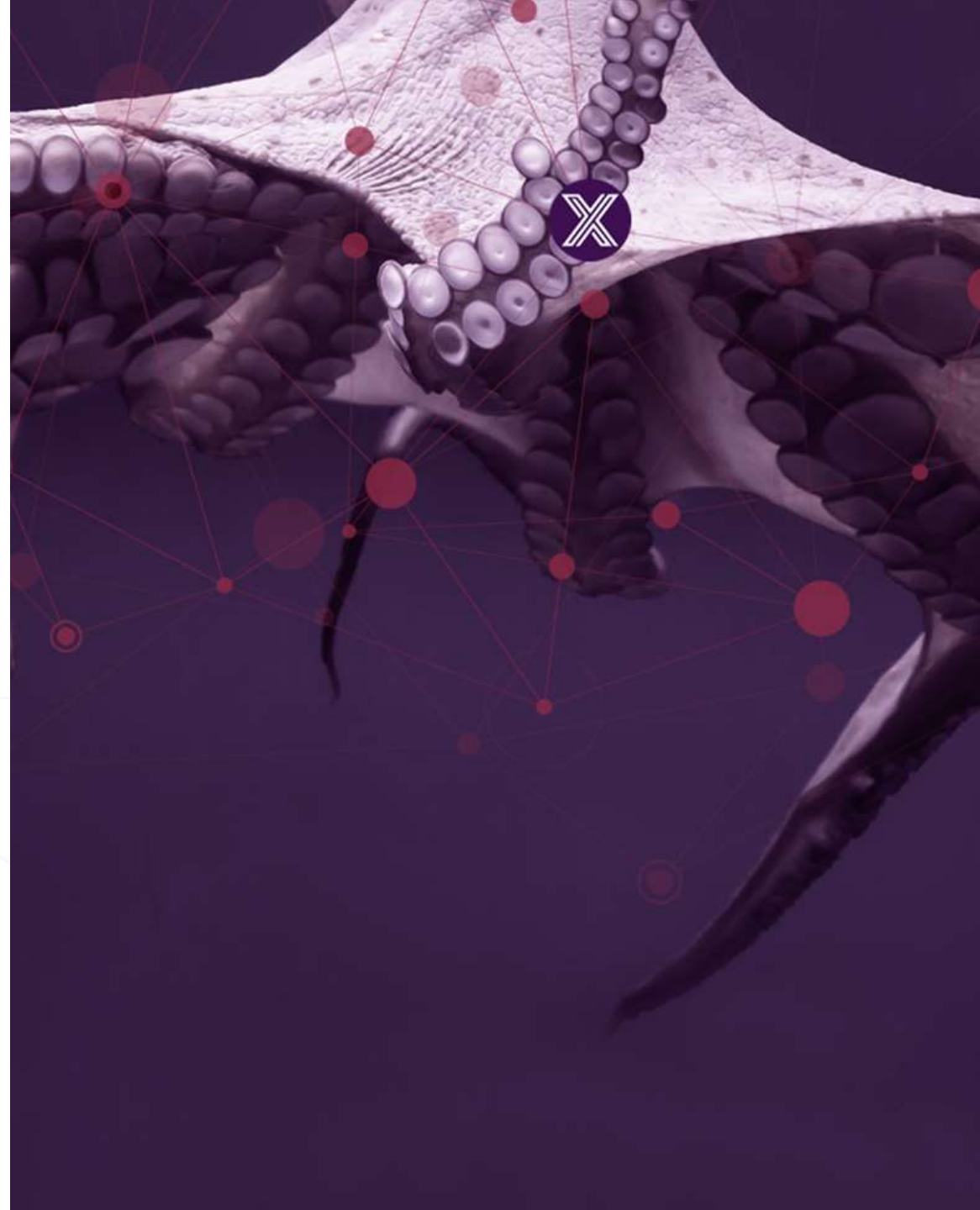


# EdgeX Documentation

- Throughout the workshop and your EdgeX development experience, you may need to reference the EdgeX documentation
- There are several places to go for more EdgeX help, details and information
- Documentation: <https://docs.edgexfoundry.org/>
  - The Getting started guides can be a great place to start: <https://docs.edgexfoundry.org/Ch-GettingStarted.html>
- Wiki pages: <https://wiki.edgexfoundry.org/>
- Github: <https://github.com/edgexfoundry/>
- Rocket Chat: <https://chat.edgexfoundry.org/channel/general>
  - Question/answer forum with channels dedicated to particular EdgeX topics
- Mailing Lists: <https://lists.edgexfoundry.org/mailman/listinfo>
  - Again, several lists for emailing the community

# EDGE X FOUNDRY™

## Lab 1: Getting and running EdgeX Foundry





# Part 3

- Explore the roles and responsibilities of EdgeX micro services
- Understand key EdgeX concepts and model elements
- Examine the device service and device service SDK purpose
- See some of the EdgeX service APIs



# EdgeX Micro Service Layers

- Contextually, EdgeX micro services are divided into 4 layers
- Crudely speaking, the layers of EdgeX provide a dual transformation engine
  - 1x - Translating information coming from sensors and devices via hundreds of protocols and thousands of formats into EdgeX
  - 2x - Delivering data to applications, enterprises and cloud systems over TCP/IP based protocols in formats and structures of customer choice



# EdgeX Technology

- A majority of the micro services are written in Go Lang
  - Previously written in Java
  - Some Device Services written in C/C++
  - A user interface is provided in JavaScript
  - Polyglot belief – use the language and tools that make sense for each service
- Each service has a REST API for communicating with it
- Uses MongoDB to persist sensor data at the edge
  - Also stores application relevant information
  - Allows for alternate persistence storage (and has been done in the past)
- A message pipe connects Core Data to Export Services and/or Rules Engine
  - Uses ZeroMQ by default
  - Allow use of MQTT as alternate if broker is provided
- Uses open source technology where appropriate
  - Ex: Consul for configuration/registry, Kong for reverse proxy, Drools for rules engine,...



{ REST }



ØMQ



# Core Services

*core data, metadata, command & configuration/registration*

- Offers temporary persistence of edge data and facilitates actuation of things
  - Needed to support disconnected edge modes, latency concerns, costs of transport to the cloud
- Collects sensor data
- Understand what sensors/devices are connected how to communicate with them
- Provision facility for new sensors/devices and device services
- Manage device actuation requests to device services/devices
- Provide micro service registry
- Provide micro service configuration

# Supporting Services

*logging, scheduling, notifications, rules engine*

- Normal software application duties plus “edge intelligence”
- Logging provides centralized EdgeX logging to location of choice
  - Log to file system, database (MongoDB), other...
- Notifications gives the ability for any EdgeX service to send an alert
  - Notify internal or external to EdgeX
  - Send via communication means of choice (email, SMS, etc.)
- Scheduling allows any EdgeX service to put tasks on the clock
  - Clean out old sensor data, check for new sensors, etc.
- Rules engine provides the means to watch sensor data and trigger local actuation as necessary.
  - The rules engine is a place holder for any “edge analytics”
  - Drools wrapped engine today

# Export Services

## *export client, export distribution*

- Provides ability to get EdgeX sensor/device data to other external systems or other EdgeX services
  - External systems include cloud providers like Azure IoT Hub, Google IoT Cloud, etc.
- Export Client – allows for internal or external clients to:
  - Register for sensor/device data of interest
  - Specify the way they want it delivered (format, filters, endpoint of delivery, etc.)
- Export Distribution – performs the act of delivering the data to registered clients
  - Receives all the sensor/device data from Core Data
  - Performs the necessary transformations, filters, etc. on the data before sending it to the registered client endpoints

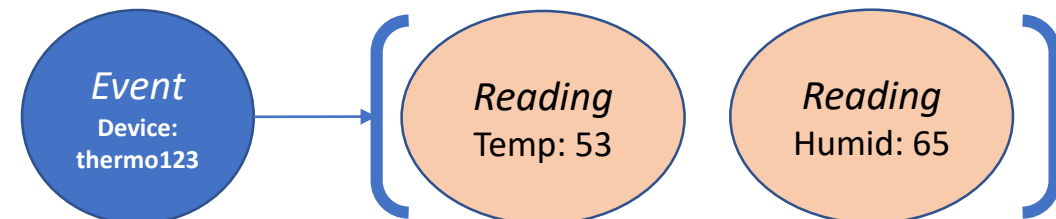
# Device Services

- A Device Service (DS) serves to translate the data produced and communicated by the thing into a common EdgeX data structure
- A DS also receives and translates generic commands from EdgeX and communicates that request to the devices for actuation in a language that the device understands.
  - For example, a DS may receive a request to turn off a Modbus PLC controlled motor. The DS would translate the generic EdgeX "shutoff" request into a Modbus serial command that the PLC/motor understands for actuation.
- A device service may service one or a number of devices (sensor, actuator, etc.) at one time.
  - A "device" that a DS manages could also be something other than a simple single physical device.
  - It could also be another gateway (and all of its device), a device manager, or device aggregator that acts like a device or collection of devices to EdgeX.
- The device service software developer kit (SDK) is a tool for generating the shell (the “scaffolding”) of a device service.
  - Initial SDK generates Java DS
  - It makes the creation of new device services easier
- The DS SDK & Meta Data Profile makes defining and provisioning new types of devices easier.



# Events & Readings

- EdgeX deals in Events and Readings
  - Collected by device services, persisted by core data, sent to cloud and other applications by export distro
- Events are collections of Readings
  - Associated to a device
- Readings represent a sensing on the part of a device/sensor
  - Simple Key/Value pair
  - Key is a value descriptor (next slide)
  - Value is the sensed value
  - Ex: Temperature: 72
- Event would need to have one Reading to make sense
- Reading has to have an “owning” event



# Value Descriptor

- Provides context and unit of measure to a reading
- Has a unique name
- Specifies unit of measure for associated Reading value
- Dictates special rules around the associated Reading value
  - Min value
  - Max value
  - Default value
- Specifies the display formatting for a Reading
- Reading key == Value Descriptor name
- In MetaData, Devices use Value Descriptors to describe data they will send and actuation command parameters/results

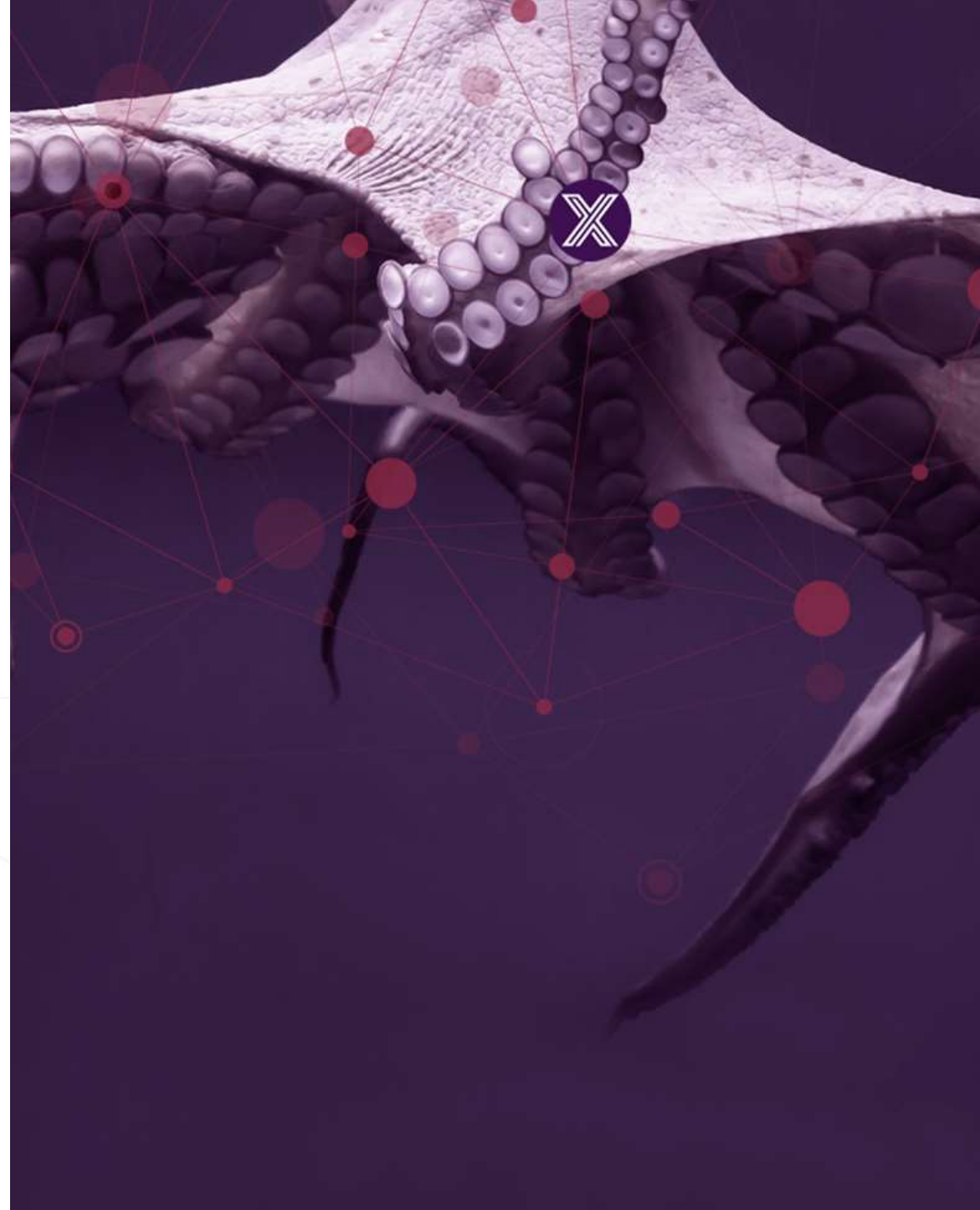
```
name: temperature
description: ambient temperature in Celsius
min: -25
max: 125
type: I (I = integer)
uomLabel: "C"
defaultValue: 25
formatting: "%s"
labels: ["room", "temp"]
```

# Device Profiles

- A Device Profile can be thought of as a template of a type or classification of device.
- A device profile provides general characteristics for the types of data a device sends and what types of commands or actions can be sent to the device
- A BACnet thermostat device profile would provide general characteristics of thermostats
  - Specifically those communicating via the BACnet protocol
- It would describe the types of data a BACnet thermostat sends
  - Current temperature (as a float and in Celsius)
  - Current humidity (as a float and a percentage)
- It would describe what commands it responds to and how to send those commands
  - Get or Set the cooling set point (passing a float as a parameter)
  - Get or Set the heating set point (passing a float as a parameter)



## Lab 2: Exploring EdgeX services and service APIs



# Part 4

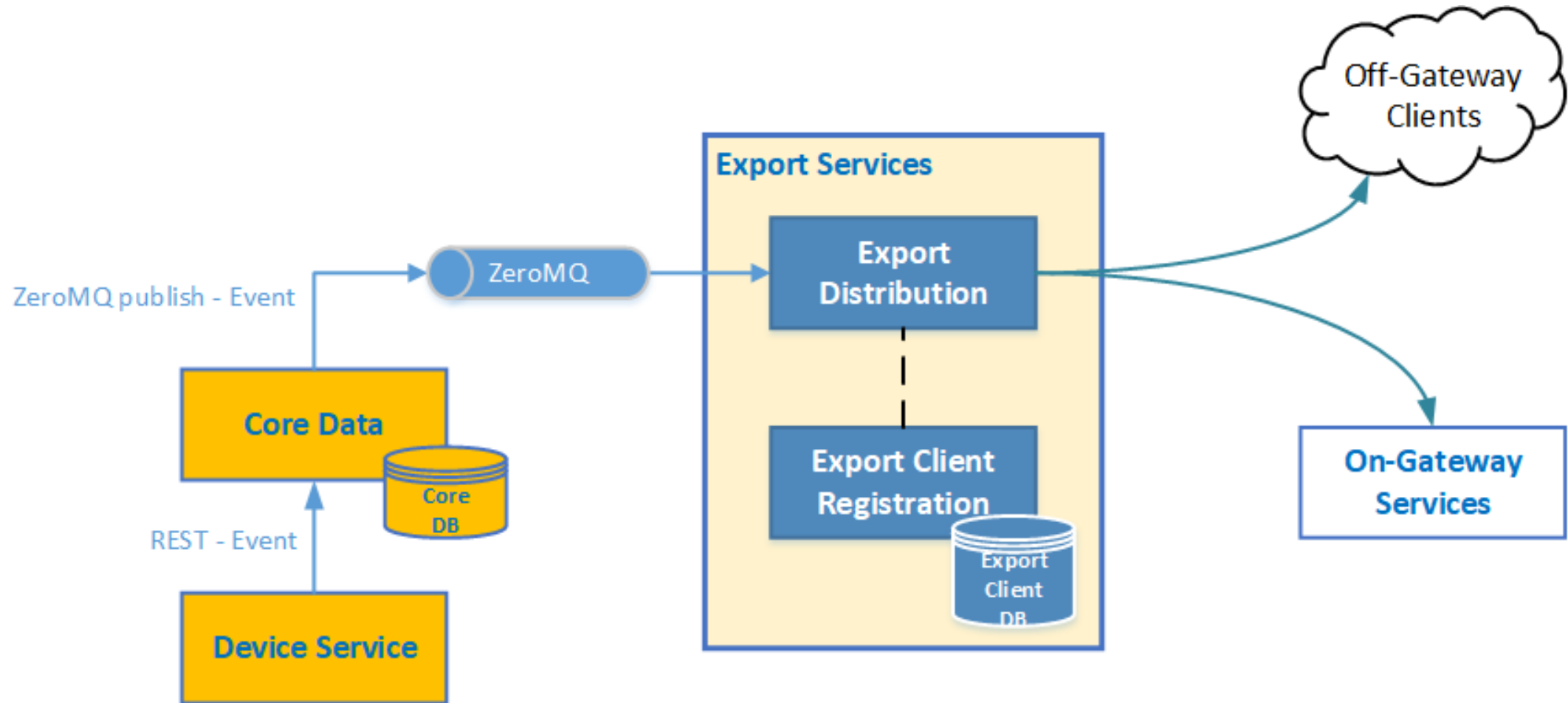
- Learn how export services get data out of EdgeX to other systems and the cloud
- Configure EdgeX to send data to the cloud

# Export Distribution – How it works

- This is an Enterprise Application Integration engine
  - Follows the EAI patterns (see <http://www.enterpriseintegrationpatterns.com/patterns/messaging/>)
  - Essentially a Pipe & Filter architecture
- Takes each incoming Core Data Event/Reading (via 0MQ) and...
  - Filters out irrelevant or incorrect data
  - Transforms the data to client's format of choice (XML, JSON, ...)
  - Optionally compresses the data
  - Optionally encrypts the data
  - Sends the data to the client's registered endpoint (REST, MQTT, 0MQ, ...)



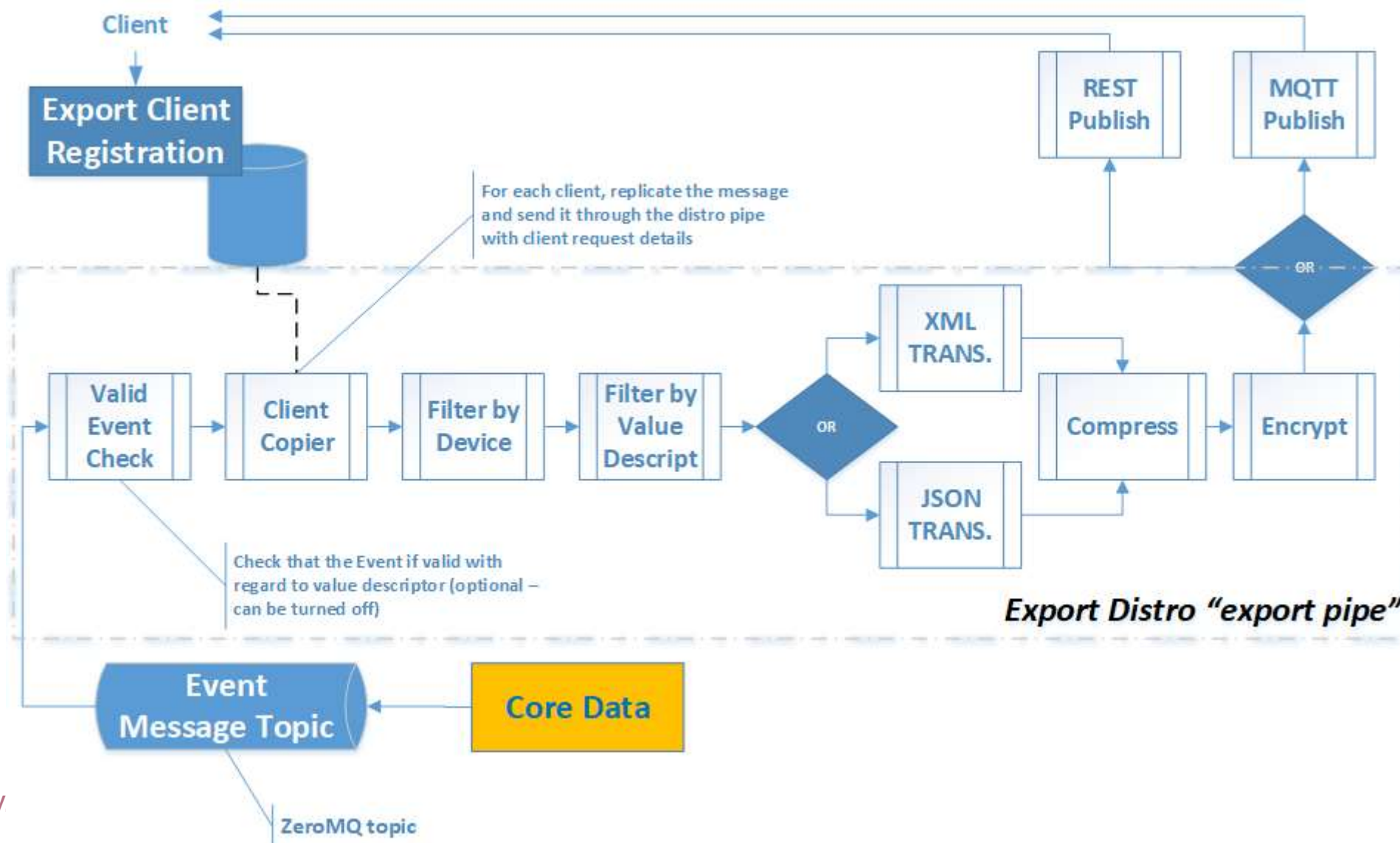
# Export Services “data flow”



# Export Services Support Today

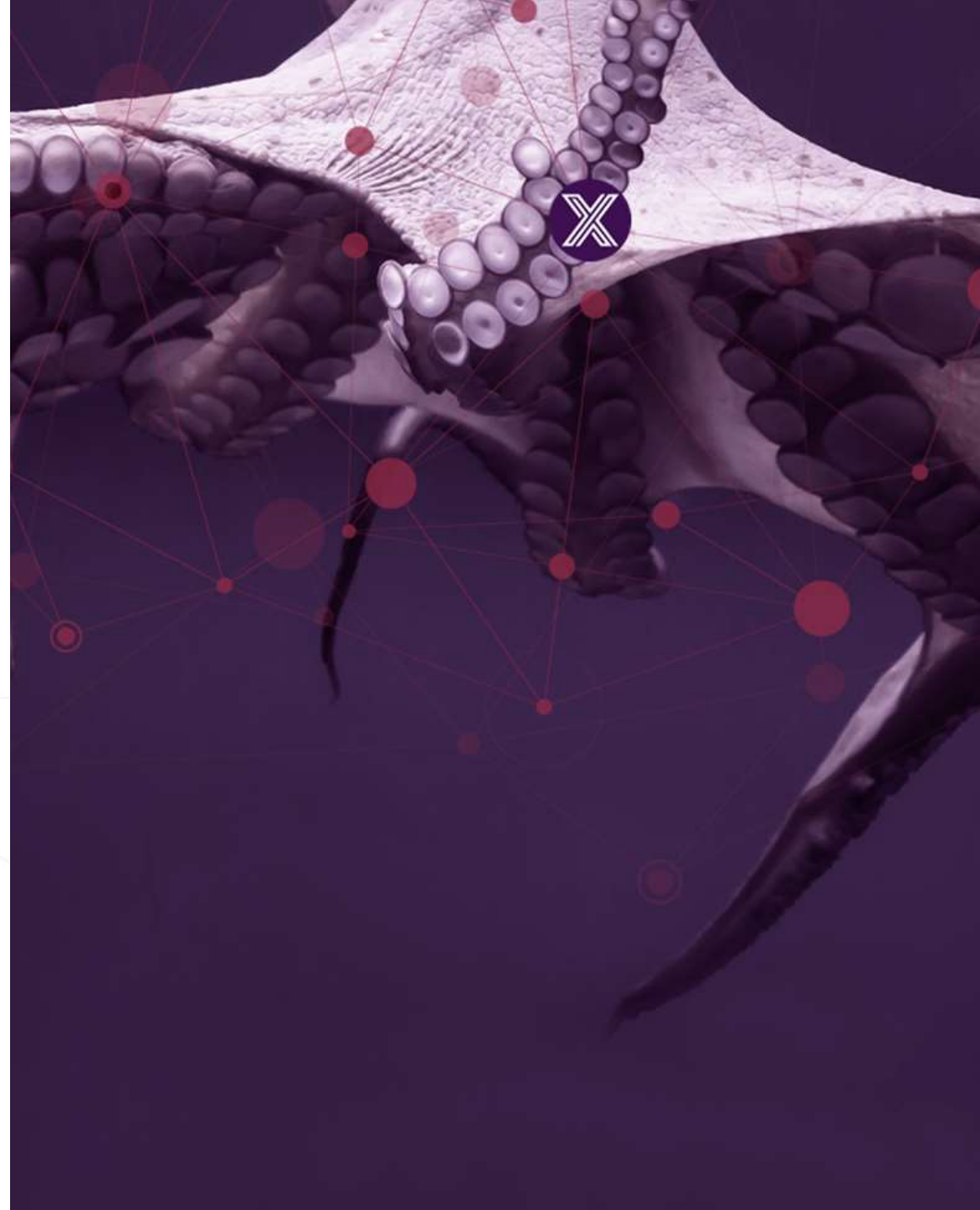
- HTTP/S Endpoints – via REST calls
- MQTT/S Topics
- Azure IoT Hub
- Google IoT Core
- XMPP
- ThingsBoard
- Brightics IoT
- Other exports (AWS, Influx, ...) being worked by the community

# Export Distro EAI Flow





## Lab 3: Exploring EdgeX Export Services



# Part 5

- Explore where to go to learn more about EdgeX
  - In particular, learn where to go to learn how to develop EdgeX micro services
  - Learn how to become familiar with the EdgeX community guidelines and procedures
- See the community participation and how to become engaged yourself
- Understand the EdgeX project cadence and roadmap



# EdgeX Development

- You now have a basis for understanding EdgeX
  - You see how its collection of micro services provide an IoT framework for building IoT platform solutions
  - You have an appreciation for some of its services, APIs, models, etc.
  - You know how to get and deploy EdgeX
- You may want to get the code and start working with the micro services
  - Replacing an existing service (for use case, for performance, etc.)
  - Augment an existing service (ex: add a connector to IBM Watson IoT)
  - Debugging and fixing a issue you identify in EdgeX
- EdgeX is a reference implementation platform, but you'll need to customize it to fit your use case & deployment



# Key Project Links

Access the code:

<https://github.com/edgexfoundry>

Access the technical documentation:

<https://docs.edgexfoundry.org/>

Access technical video tutorials:

<https://wiki.edgexfoundry.org/display/FA/EdgeX+Tech+Talks>

EdgeX Blog:

<https://www.edgexfoundry.org/news/blog/>

Join an email distribution:

<https://lists.edgexfoundry.org/mailman/listinfo>

Join the Rocket Chat:

<https://chat.edgexfoundry.org/home>

Become a project member:

<https://www.edgexfoundry.org/about/members/join/>

LinkedIn:

<https://www.linkedin.com/company/edgexfoundry/>

Twitter:

<https://twitter.com/EdgeXFoundry>

Youtube:

<https://www.youtube.com/edgexfoundry>

# What's with the 'X'?

- Fundamental goal of the EdgeX project is to provide a stable, product-quality open source foundation for interoperable commercial offers
- The 'X' in EdgeX allows the project name to be trademarked for use as a certification mark
- A certification program will be established in the project for commercial offerings to verify that key EdgeX interoperability APIs were maintained alongside proprietary value-add
- Initial program launch targeted for 'Delhi' release (~ Oct 2018) with ramp in 2019
- Stability for key elements (e.g. core APIs and certification process) is maintained through the EdgeX Technical Steering Committee (TSC) and clear versioning system
- Licensed under Apache 2.0, anyone can leverage the EdgeX code base as a foundation for their commercial offerings
- Can be a full EdgeX-compliant IoT platform, value-added plug-in micro service(s) or a services model

# Backed by Industry Leaders



With more in process!

# Engagement Options

- Project is a technical meritocracy.
  - Anyone can contribute to or use the EdgeX Foundry code for free.
- Technical Steering Committee (TSC) and Working Group (WG) meetings are open to the public
- TSC and WG Chairs in addition to code committers and maintainers are voted in based on technical acumen and alignment to project tenets.
  - This ensures robustness and stability in the architecture, technology choices, roadmap and code base.
- Joining as a paid project member affords maximum influence over project direction

# Member Benefits

- Additional influence to shape the overall platform architecture to enable commercialization needs
- Recognition for Industry thought and technology leadership
- Marketing and networking within the EdgeX Project for business opportunities (effectively a vendor-neutral partner program)
- Discounted sponsorships at Linux Foundation and EdgeX Foundry-produced events (e.g. trade shows, hackathons, etc.)
- Learning and engagement



# EdgeX Foundry Governance Structure

EDGEX FOUNDRY MEMBER COMPANIES (60+)

## GOVERNING BOARD (GB)

*Composed of appointed and elected individuals; manages the business of the EdgeX Foundry.*

## TECHNICAL STEERING COMMITTEE (TSC)

*Leads the technical work of EdgeX Foundry. Oversees and aligns working groups.*

WG

WG

WG

WG

WG

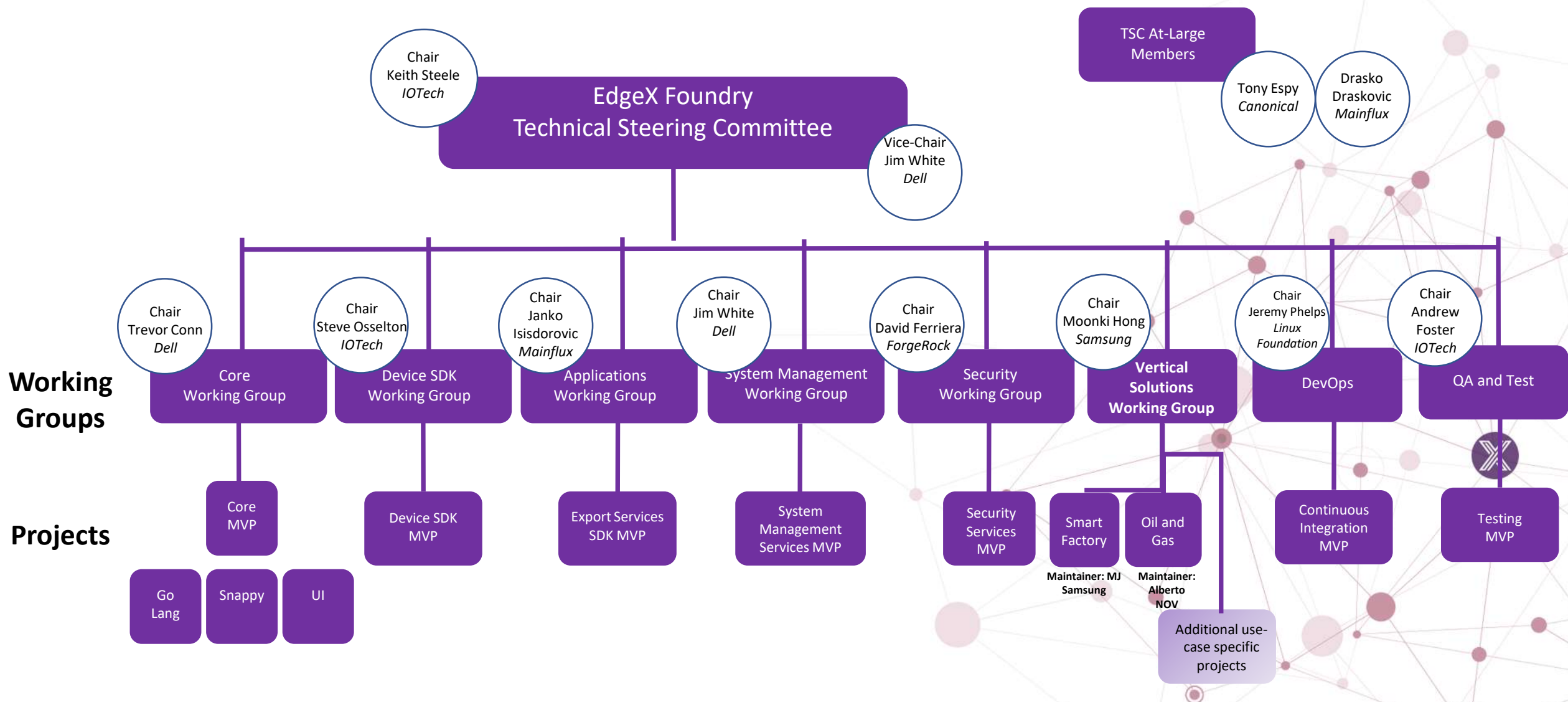
## CERTIFICATION COMMITTEE

*Develops and oversees the certification program for EdgeX Certified components.*

LF SUPPORT TEAM

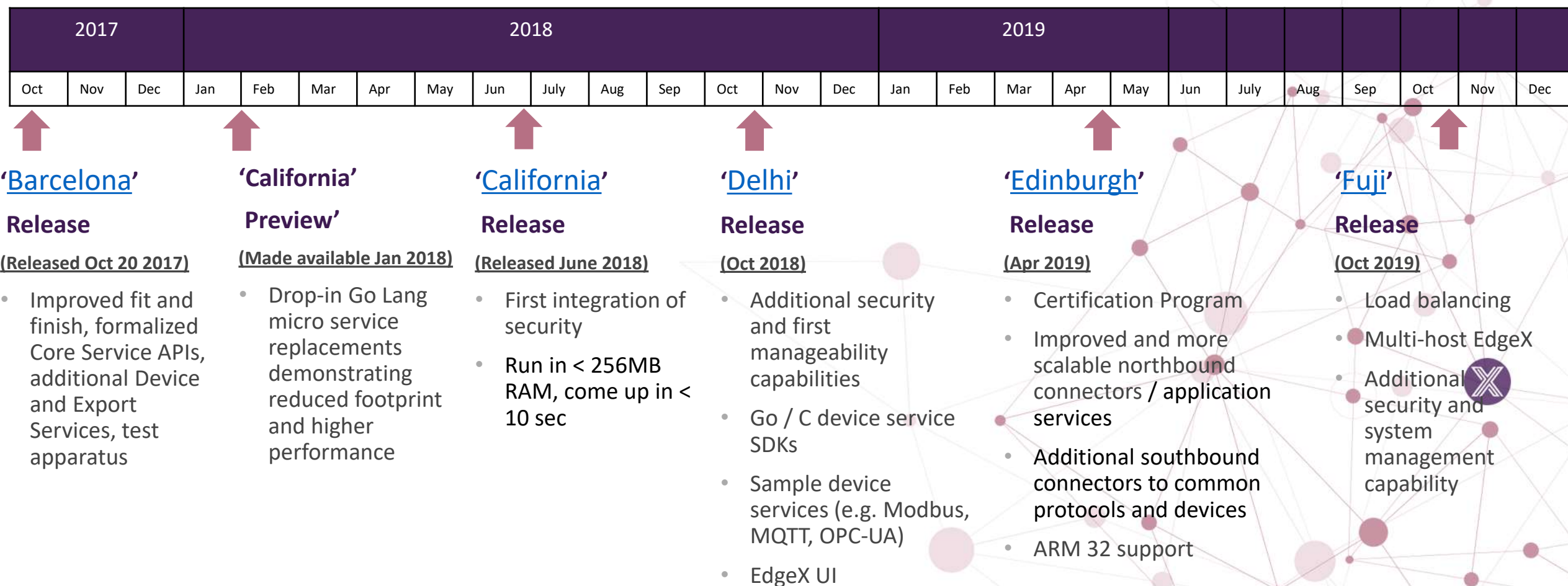


# EdgeX Project Organization



# Target Bi-Annual Release Roadmap

In order to provide EdgeX consumers with a predictable foundation to base their commercial offerings on it is the goal of the TSC to outline key release themes at least 12 months in advance and to plan features to be delivered in a given release 6 months in advance. As with any open source software project, delivery of planned features is based on priority and available developer bandwidth.



# Delhi Release - Major Themes & Objectives

- Smaller development cycle so scope has to match
  - Release date – sometime between Oct 29 and Nov 5.
- High level scope
  - Initial System Management APIs and agent
  - Device Service SDKs (Go/C) and some example device services
  - The next wave of security features (access control lists to grant access to appropriate services, and improved security bootstrapping, ...\_
  - Improve testing(better/more unit, complete black box and add performance testing)
  - Refactored and improved Go Lang micro services
  - Design and architecture work in advance of Edinburgh release
    - Options and implementation plan for database replacement
    - Design and implementation plans for export service replacement with application services
  - An EdgeX UI suitable for demos and smaller installations

# Edinburgh Release – Major Themes & Objectives

- Define and implement a certification program for micro service drop in replacements
- Replace (or have as alternate) existing export services with application services
- Provide an alternative to MongoDB as reference database for EdgeX data persistence
- Continue delivering security and system management functionality from 2018 plan
- Provide a full complement of reference implementation device services created with the Go and C SDKs
  - E.g. Modbus, BACNet, BLE, MQTT, SNMP and more
  - Encourage platform providers and device makers to provide commercial services as well
- Support ARM 32 deployments



Thank You

Jim White

james\_white2@dell.com

