Getting Lean and Distributed at the Edge

Jim White
Dell Technologies
October 2018
Agenda

• 2 minute quick intro to EdgeX Foundry
• EdgeX architecture
• Requirements of an edge platform
• EdgeX performance metrics
  • In the beginning
  • Today
  • Keys to the diet
• Distributing EdgeX micro services
  • Why distribution matters
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
EdgeX Foundry
Backed by Industry Leaders

With more in process!

edgexfoundry.org   |         @edgexfoundry
EdgeX Foundry Goals

• Common open platform unifying edge computing
• Create an ecosystem of interoperable plug-and-play components
• Certify to ensure interoperability and compatibility
• Provide tools to create IoT edge solutions
• Collaborate across the IoT
Cloud, Enterprise, On-Prem...

Local Analytics

Loosely-Coupled Microservices Framework

Export Services
- Client Registration
- Distribution
- Additional Services

Supporting Services
- Scheduling
- Alerts & Notifications
- Logging
- Additional Services

Core Services
- Core Data
- Command
- Metadata
- Registry & Config

Device Services
- REST
- OPC-UA
- Modbus
- BACnet
- Zigbee
- BLE
- MQTT
- SNMP
- Virtual

"Southbound" Devices, Sensors and Actuators

It's 102°C

Stop the machine
EdgeX Architectural Tenets

• Platform agnostic
• Extremely flexible
• Encourages best of breed solutions
• Store and forward
• Facilitate “intelligence” moving closer to the edge
• Support brown and green devices/sensors
• Must be secure and easily managed
EdgeX Enables Tiered Fog Deployments

Diagram showing the integration of EdgeX with various components such as Device Services, Gateway, and Cloud, illustrating the export and analytics services.
Edgex Technology

golang  JS  C++
mongoDB  redis

{ REST }

EdgeX Technology

edgexfoundry.org  |  @edgexfoundry
Key Project Links

https://github.com/edgexfoundry
https://docs.edgexfoundry.org/
https://wiki.edgexfoundry.org/display/FA/EdgeX+Tech+Talks
https://www.edgexfoundry.org/news/blog/
https://lists.edgexfoundry.org/mailman/listinfo
https://chat.edgexfoundry.org/home
https://www.edgexfoundry.org/about/members/join/
https://www.linkedin.com/company/edgexfoundry/
https://twitter.com/EdgeXFoundry
https://www.youtube.com/edgexfoundry
Requirements of an Edge Platform

• Near real-time
• Sensor sampling rates of 100-1000/second
• Handling kilobytes per message
• Latency < 1 second from sensor to analytics to actuation
## Typical Collection or Sampling Rates

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Sampling Rate</th>
<th>Order of Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electric Grid Line Fault Detection</td>
<td>Micro second</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>Vibration Sensor</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>Pressure Sensor</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>PLC</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>CNC Machines</td>
<td>Micro second to seconds</td>
<td>KBs, MBs</td>
</tr>
<tr>
<td>CANBus</td>
<td>Micro second to seconds</td>
<td>MB</td>
</tr>
<tr>
<td>Building Automation</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>Locomotive Telemetry</td>
<td>Micro second to seconds</td>
<td>KBs, MBs</td>
</tr>
<tr>
<td>RTU in Utility</td>
<td>Micro second to seconds</td>
<td>KBs, MBs</td>
</tr>
<tr>
<td>Oil/Gas</td>
<td>Micro second to seconds</td>
<td>KBs</td>
</tr>
<tr>
<td>RFID</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
</tbody>
</table>
EdgeX in the beginning - April 2017

- EdgeX started as a Dell POC in 2015

- 7-5 GB RAM
- ~50% CPU utilization (2CPU, 8GB gateway)
  - Pegged at 100% at startup
- Startup > 30 seconds/service
  - System startup ~5 minutes
## EdgeX @ the start (Hannover Messe 2017)

<table>
<thead>
<tr>
<th>Service</th>
<th>Container size (in MB)</th>
<th>RAM (in MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>165</td>
<td>301</td>
</tr>
<tr>
<td>Coredata</td>
<td>170</td>
<td>299</td>
</tr>
<tr>
<td>Command</td>
<td>159</td>
<td>244</td>
</tr>
<tr>
<td>Logging</td>
<td>158</td>
<td>249</td>
</tr>
<tr>
<td>Scheduler</td>
<td>161</td>
<td>215</td>
</tr>
<tr>
<td>Notifications</td>
<td>153</td>
<td>222</td>
</tr>
<tr>
<td>Export Client</td>
<td>164</td>
<td>240</td>
</tr>
<tr>
<td>Export Distro</td>
<td>168</td>
<td>291</td>
</tr>
<tr>
<td>Rules Engine</td>
<td>177</td>
<td>270</td>
</tr>
<tr>
<td>SNMP DS</td>
<td>169</td>
<td>265</td>
</tr>
<tr>
<td>Fishertech DS</td>
<td>169</td>
<td>281</td>
</tr>
<tr>
<td>Bacnet DS</td>
<td>437</td>
<td>278</td>
</tr>
<tr>
<td>BLE DS</td>
<td>168</td>
<td>301</td>
</tr>
<tr>
<td>UI</td>
<td>607</td>
<td>50</td>
</tr>
<tr>
<td>Mongo</td>
<td>402</td>
<td>99</td>
</tr>
</tbody>
</table>

CPU % - 34%
Mem: 5.1 of 7.7G used
Swap: 168M of 7.9G used
Performance Targets set Oct 2017

• 2nd EdgeX Technical Steering Committee Face to Face Decision

- Run in < 1GB RAM
- Use < 25% CPU
- Use < 32GB Storage
- Startup < 10 seconds
- Latency < 1 second (ingestion to actuation)
Early Go v. Java Experiment (October 2017)

<table>
<thead>
<tr>
<th>Measure</th>
<th>Go</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable/JAR Footprint</td>
<td>11.7 MB</td>
<td>42.4 MB</td>
</tr>
<tr>
<td>Container Footprint</td>
<td>16.2 MB</td>
<td>165 MB</td>
</tr>
<tr>
<td>Memory Usage (On Startup)</td>
<td>4.3 MB</td>
<td>221 MB</td>
</tr>
<tr>
<td>Memory Usage (Under Load)</td>
<td>9.2 MB</td>
<td>230 MB</td>
</tr>
<tr>
<td>CPU Usage (Steady State)</td>
<td>0.15%</td>
<td>0.30%</td>
</tr>
<tr>
<td>CPU Usage (Under Load)</td>
<td>5.0% - 15.0%</td>
<td>6.0% - 15.0%     (spiked at 90% for heavy load)</td>
</tr>
<tr>
<td>Startup Speed</td>
<td>0.14 Seconds</td>
<td>12.55 Seconds</td>
</tr>
<tr>
<td>Response Speed (Ping)</td>
<td>0.0011 Seconds</td>
<td>0.0022 Seconds</td>
</tr>
<tr>
<td>Response Speed (Post)</td>
<td>0.0091 Seconds</td>
<td>0.0137 Seconds</td>
</tr>
<tr>
<td>Response Speed (Get)</td>
<td>0.0038 Seconds</td>
<td>0.0062 Seconds</td>
</tr>
</tbody>
</table>
EdgeX Today (Oct 2018 – Delhi release)

<table>
<thead>
<tr>
<th>Service</th>
<th>Go Max CPU</th>
<th>Java Max CPU</th>
<th>Go Max Mem</th>
<th>Java Max Mem</th>
<th>Go Container Size</th>
<th>Java Container Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Data</td>
<td>0.16%</td>
<td>13.71%</td>
<td>29.54</td>
<td>270</td>
<td>15.9</td>
<td>142</td>
</tr>
<tr>
<td>Core Metadata</td>
<td>1.02%</td>
<td>4.09%</td>
<td>8.70</td>
<td>275</td>
<td>10.5</td>
<td>125</td>
</tr>
<tr>
<td>Core Command</td>
<td>0.008%</td>
<td>1.10%</td>
<td>1.55</td>
<td>204</td>
<td>8.67</td>
<td>131</td>
</tr>
<tr>
<td>Support Logging</td>
<td>0.32%</td>
<td>7.31%</td>
<td>7.58</td>
<td>210</td>
<td>9.27</td>
<td>116</td>
</tr>
<tr>
<td>Support Notifications</td>
<td>0.09%</td>
<td>0.71%</td>
<td>4.53</td>
<td>217</td>
<td>9.65</td>
<td>125</td>
</tr>
<tr>
<td>Export Client</td>
<td>0.02%</td>
<td>7.66%</td>
<td>4.12</td>
<td>221</td>
<td>15.9</td>
<td>136</td>
</tr>
<tr>
<td>Export Distro</td>
<td>0.06%</td>
<td>9.99%</td>
<td>3.84</td>
<td>251</td>
<td>16.2</td>
<td>140</td>
</tr>
<tr>
<td>Mongo</td>
<td>0.93%</td>
<td>0.55%</td>
<td>72.30</td>
<td>76</td>
<td>361</td>
<td>361</td>
</tr>
<tr>
<td>Consul</td>
<td>1.71%</td>
<td>0.43%</td>
<td>17.12</td>
<td>12</td>
<td>59.1</td>
<td>168</td>
</tr>
<tr>
<td>Docker Files/Volume</td>
<td>0.01%</td>
<td>0.01%</td>
<td>0.20</td>
<td>0</td>
<td>81.2</td>
<td>122</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>4%</strong></td>
<td><strong>46%</strong></td>
<td><strong>149</strong></td>
<td><strong>1736</strong></td>
<td><strong>587</strong></td>
<td><strong>1566</strong></td>
</tr>
</tbody>
</table>

%CPU %CPU MB MB
EdgeX Delhi Release

- > 90% memory reduction
- > 90% CPU reduction
- > 80% footprint reduction
- > 95% startup time reduction
  - Startup ~ 5 seconds
Keys to the diet success

• Go vs. Java
  • Executable vs virtual machine
  • Frameworks are heavy (Spring, EAI engine, etc.)
  • Library bloat (frameworks and Java use a lot of libraries)

• Micro service architecture
  • Replace one service at a time
  • Allowed for exploration and discovery (best practices, tools, etc.)

• Iteration
  • Crawl-walk-run
  • Get a replacement service, make it better, standardize across services
Distribution – the other tool in lean & mean
Distribution – Run where you can
Question??

Thank you

Jim White:  james_white2@dell.com
Agenda

• 2 minute quick intro to EdgeX Foundry
• EdgeX architecture
• Requirements of an edge platform
• EdgeX performance metrics
  • In the beginning
  • Today
  • Keys to the diet
• Distributing EdgeX micro services
  • Why distribution matters
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
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

A Dell incubated project started in 2015; entered into open source in April 2017

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
Backed by Industry Leaders

With more in process!
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
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
Cloud, Enterprise, On-Prem...

Stop the machine

Local Analytics

It’s 102 °C
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**
EdgeX Enables Tiered Fog Deployments

- In today’s IoT landscape, it is imperative to leverage compute, storage, network resources where every they live
- Loosely-coupled architecture enables distribution across nodes to enable tiered edge/fog computing
- Scope includes embedded sensors to controllers, edge gateways and servers
- Quantity and function of micro services deployed on a given node depends on the use case and capability of hardware
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,…
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
Requirements of an Edge Platform

- Not all platforms are equal
  - Real time vs near real time
- Collection rates
  - Typically dealing with sensor sampling rates of 100-1000/second
- Throughput
  - Typically handling kilobytes per message
  - Surveillance systems much greater
- Latency
  - To the cloud (or application layer)
  - Back to the device
  - < 1 second from sensor to analytics to actuation
# Typical Collection or Sampling Rates

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Sampling Rate</th>
<th>Order of Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electric Grid Line Fault Detection</td>
<td>Micro second</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>Vibration Sensor</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>Pressure Sensor</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>PLC</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>CNC Machines</td>
<td>Micro second to seconds</td>
<td>KBs, MBs</td>
</tr>
<tr>
<td>CANBus</td>
<td>Micro second to seconds</td>
<td>MB</td>
</tr>
<tr>
<td>Building Automation</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
<tr>
<td>Locomotive Telemetry</td>
<td>Micro second to seconds</td>
<td>KBs, MBs</td>
</tr>
<tr>
<td>RTU in Utility</td>
<td>Micro second to seconds</td>
<td>KBs, MBs</td>
</tr>
<tr>
<td>Oil/Gas</td>
<td>Micro second to seconds</td>
<td>KBs</td>
</tr>
<tr>
<td>RFID</td>
<td>Micro second to seconds</td>
<td>Bytes, KBs</td>
</tr>
</tbody>
</table>
EdgeX in the beginning - April 2017

• EdgeX started as a Dell POC in 2015
  • We needed a platform independent language/framework
  • Enter Java/Spring Framework
  • It proved the concepts but wasn’t edge worthy

• Initial Performance was poor and not able to meet requirements
  • Running all the micro services required 7+ GB RAM
  • Improved to 5.1GB RAM with some tuning
  • CPU utilization around ~50% on a 2CPU, 8GB gateway
  • At startup, CPU pegged at 100%
  • Startup time for each service > 30 seconds
  • System startup ~5 minutes (due to service dependencies)
## EdgeX @ the start (Hannover Messe 2017)

<table>
<thead>
<tr>
<th>Service</th>
<th>Container size (in MB)</th>
<th>RAM (in MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata</td>
<td>165</td>
<td>301</td>
</tr>
<tr>
<td>Coredata</td>
<td>170</td>
<td>299</td>
</tr>
<tr>
<td>Command</td>
<td>159</td>
<td>244</td>
</tr>
<tr>
<td>Logging</td>
<td>158</td>
<td>249</td>
</tr>
<tr>
<td>Scheduler</td>
<td>161</td>
<td>215</td>
</tr>
<tr>
<td>Notifications</td>
<td>153</td>
<td>222</td>
</tr>
<tr>
<td>Export Client</td>
<td>164</td>
<td>240</td>
</tr>
<tr>
<td>Export Distro</td>
<td>168</td>
<td>291</td>
</tr>
<tr>
<td>Rules Engine</td>
<td>177</td>
<td>270</td>
</tr>
<tr>
<td>SNMP DS</td>
<td>169</td>
<td>265</td>
</tr>
<tr>
<td>Fishertech DS</td>
<td>169</td>
<td>281</td>
</tr>
<tr>
<td>Bacnet DS</td>
<td>437</td>
<td>278</td>
</tr>
<tr>
<td>BLE DS</td>
<td>168</td>
<td>301</td>
</tr>
<tr>
<td>UI</td>
<td>607</td>
<td>50</td>
</tr>
<tr>
<td>Mongo</td>
<td>402</td>
<td>99</td>
</tr>
</tbody>
</table>

CPU % - 34%  
Mem: 5.1 of 7.7G used  
Swap: 168M of 7.9G used
Setting new Performance Targets in Oct 2017

• Clearly, EdgeX was not going to cut it as an edge solution
• 2nd EdgeX Technical Steering Committee Face to Face Decision  
  • Move to an executable language (Go Lang) – based on early experimental testing  
  • Set reasonable edge performance targets  
• The target is to run all of EdgeX on a Raspberry Pi 3 type of device  
  • 1 GB RAM, 64bit CPU, at least 32GB storage space  
• Additional performance targets  
  • Startup in 10 seconds or less (post OS boot)  
  • Latency for one piece of data from data ingestion to actuation will be < 1 second  
• Remaining OS and Hardware agnostic  
  • Windows, Linux, *nix, …  
  • Intel/Arm 64/Arm 32
# Early Go v. Java Experiment (October 2017)

<table>
<thead>
<tr>
<th>Measure</th>
<th>Go</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executable/JAR Footprint</td>
<td>11.7 MB</td>
<td>42.4 MB</td>
</tr>
<tr>
<td>Container Footprint</td>
<td>16.2 MB</td>
<td>165 MB</td>
</tr>
<tr>
<td>Memory Usage (On Startup)</td>
<td>4.3 MB</td>
<td>221 MB</td>
</tr>
<tr>
<td>Memory Usage (Under Load)</td>
<td>9.2 MB</td>
<td>230 MB</td>
</tr>
<tr>
<td>CPU Usage (Steady State)</td>
<td>0.15%</td>
<td>0.30%</td>
</tr>
<tr>
<td>CPU Usage (Under Load)</td>
<td>5.0% - 15.0%</td>
<td>6.0% - 15.0% (spiked at 90% for heavy load)</td>
</tr>
<tr>
<td>Startup Speed</td>
<td>0.14 Seconds</td>
<td>12.55 Seconds</td>
</tr>
<tr>
<td>Response Speed (Ping)</td>
<td>0.0011 Seconds</td>
<td>0.0022 Seconds</td>
</tr>
<tr>
<td>Response Speed (Post)</td>
<td>0.0091 Seconds</td>
<td>0.0137 Seconds</td>
</tr>
<tr>
<td>Response Speed (Get)</td>
<td>0.0038 Seconds</td>
<td>0.0062 Seconds</td>
</tr>
</tbody>
</table>
EdgeX Today (Oct 2018 – Delhi release)

- Can run on a Rasp Pi 3 level of device
  - < 1GB RAM
- > 90% memory reduction
- > 90% CPU reduction
- > 80% footprint reduction
- > 95% startup time reduction
  - Startup ~ 5 seconds

<table>
<thead>
<tr>
<th>Service</th>
<th>Go Max CPU</th>
<th>Java Max CPU</th>
<th>Go Max Mem</th>
<th>Java Max Mem</th>
<th>Go Container Size</th>
<th>Java Container Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Data</td>
<td>0.16%</td>
<td>13.71%</td>
<td>29.54</td>
<td>270</td>
<td>15.9</td>
<td>142</td>
</tr>
<tr>
<td>Core Metadata</td>
<td>1.02%</td>
<td>4.09%</td>
<td>8.70</td>
<td>275</td>
<td>10.5</td>
<td>125</td>
</tr>
<tr>
<td>Core Command</td>
<td>0.008%</td>
<td>1.10%</td>
<td>1.55</td>
<td>204</td>
<td>8.67</td>
<td>131</td>
</tr>
<tr>
<td>Support Logging</td>
<td>0.32%</td>
<td>7.31%</td>
<td>7.58</td>
<td>210</td>
<td>9.27</td>
<td>116</td>
</tr>
<tr>
<td>Support Notifications</td>
<td>0.09%</td>
<td>0.71%</td>
<td>4.53</td>
<td>217</td>
<td>9.65</td>
<td>125</td>
</tr>
<tr>
<td>Export Client</td>
<td>0.02%</td>
<td>7.66%</td>
<td>4.12</td>
<td>221</td>
<td>15.9</td>
<td>136</td>
</tr>
<tr>
<td>Export Distro</td>
<td>0.06%</td>
<td>9.99%</td>
<td>3.84</td>
<td>251</td>
<td>16.2</td>
<td>140</td>
</tr>
<tr>
<td>Mongo</td>
<td>0.93%</td>
<td>0.55%</td>
<td>72.30</td>
<td>76</td>
<td>361</td>
<td>361</td>
</tr>
<tr>
<td>Consul</td>
<td>1.71%</td>
<td>0.43%</td>
<td>17.12</td>
<td>12</td>
<td>59.1</td>
<td>168</td>
</tr>
<tr>
<td>Docker Files/Volume</td>
<td>0.01%</td>
<td>0.01%</td>
<td>0.20</td>
<td>0</td>
<td>81.2</td>
<td>122</td>
</tr>
<tr>
<td>Total</td>
<td>4%</td>
<td>46%</td>
<td>149</td>
<td>1736</td>
<td>587</td>
<td>1566</td>
</tr>
</tbody>
</table>

- %CPU | MB
- %CPU | MB

edgxfoundry.org | @edgxfoundry
Keys to the diet success

• Go vs. Java
  • Executable vs virtual machine
  • Frameworks are heavy (Spring, EAI engine, etc.)
  • Library bloat (frameworks and Java use a lot of libraries)

• Micro service architecture
  • Replace one service at a time
  • Allowed for exploration and discovery (best practices, tools, etc.)

• Iteration
  • Crawl-walk-run
  • Get a replacement service, make it better, standardize across services
Distribution – the other tool in lean & mean

• EdgeX is a collection of independent and loosely couple micro services
  • Allows for distribution across the available compute resources
• In today’s IoT landscape, it is imperative to leverage compute, storage, network resources where every they live
• Loosely-coupled architecture enables distribution across nodes to enable tiered edge/fog computing
• Scope includes embedded sensors to controllers, edge gateways and servers
• Quantity and function of micro services deployed on a given node depends on the use case and capability of hardware
Distribution – Run where you can
Question??

Thank you

Jim White:  james_white2@dell.com