Remote and Mobile Assets—Edge Compute

This module is part of the larger Remote and Mobile Assets (RaMA) Cisco Validated Design (CVD). Refer to the other modules for additional details about certain aspects of the architecture that are touched on in this module. All of the RaMA CVD modules are available at: www.cisco.com/go/rama

- Solution Brief—An overview of the RaMA CVD and the available modules.
- Design and Implementation Guide (DIG)—Overall document for architecture, design, and best practice recommendations for remote and mobile asset deployments.
- Technology Guidance Module—Overview of the available hardware options for IoT gateways in the RaMA solution, with recommendations on hardware platform and software features to use for common scenarios.
- Security Module—Describes how the RaMA solution was designed from the ground up with security in mind. Includes detailed descriptions of how the solution fits into the SAFE model, including securing the gateways, data plane, and management plane. Also includes a section on achieving PCI compliance.
- Enterprise Network Integration Module—Best practices for the enterprise headend focusing on resiliency, high-availability, load-balancing, and security. Includes detailed descriptions of FlexVPN and WAN redundancy mechanisms.
- Remote Site Management Module—Best practices for remote site connectivity, covering the use of the full range of Cisco Industrial Routers (IR 807, IR 809, IR829, IR 1101) as the managed gateway, providing wired and cellular connectivity for southbound devices as well as numerous northbound interfaces. This module also covers best practices for inbound connectivity for devices behind the gateway including isolation of management and data planes and whitelisting of applications and devices.
- Fleet Management Module—Architecture for mobile applications in which the IR829 acts as the managed gateway and provides wired and wireless connectivity for southbound devices, as well as numerous northbound interfaces (LTE, Wireless Workgroup Bridge, GPS). Use of edge compute in the form of Cisco IOX is also included.
- Zero Touch Provisioning Module—Use of Kinetic GMM by IT personnel for provisioning and managing Cisco Industrial Routers with a focus on secure, scalable deployment.
- Field Deployment Module—Use of Kinetic GMM by OT personnel for deploying Cisco Industrial Routers in the field, with minimal knowledge of the underlying networking technology required.
Overview

This module provides an overview of the edge compute capabilities embedded within the Cisco Industrial Router portfolio and the Cisco IOx microservice framework for developing containerized applications. These containerized applications can be deployed at the edge of the IoT fabric to help extract IoT data from devices connected behind the gateway and transform that data for upstream consumption.

This module also covers how the cloud-hosted Cisco Kinetic GMM can be used to centrally deploy and manage the entire deployment life-cycle of an IOx application to edge gateways in a scalable fashion. This involves the entire process from deploying the applications, starting, stopping and restarting the application, upgrading to a later version of the application, and finally deleting the application if no longer needed.

The implementation section of the module describes the step-by-step process of developing a Docker container image encapsulating some sample code, creating an IOx deployment package, and then finally deploying the application to the edge gateway using Kinetic GMM. It then illustrates how a developer can package and deploy an updated version of the application in a seamless manner as well as monitor and log application activities within the Kinetic GMM UI.

Requirements

- Transformation of IoT edge generated data into business outcomes
- Rapid system integration and edge application management
- Availability of an Application Development Framework
- Ability to centrally deploy and manage the entire application life-cycle
- Support for Native Docker Tooling
- Distributed edge compute
- Availability of development and test sandboxes

Disclaimer: While this document describes best practices and details on deploying and utilizing IOx for customized microservices, custom microservices are neither created nor supported by Cisco. The customer assumes all responsibility and risk associated with the development and use of such custom microservices.
Remote and Mobile Assets—Edge Compute

Architecture

- Secure communications

Figure 1  Distributed Edge Compute and IOx Microservice Application Architecture

Design

The key design guidelines for creating edge compute solutions include:

- Devising methods of managing the information or sensor data received by the gateway via local processing and preparing it for transmission over the network backhaul.

- Enabling effective bandwidth utilization, as desired, through local processing at the edge to transform, compress, or reformat data for upstream consumption.

- Supporting real-time data processing for low-latency applications.

- Converting or adapting legacy protocols to provide the information northbound via IP-based protocols or APIs.

- Determining the location of decision making based upon data received—on the gateway device, on the local network, or at the head end.
Remote and Mobile Assets—Edge Compute

Design

Key Components of a Solution Using Edge Computing

Key Components enabling Edge Computing include the IoT edge gateway, the IOx framework for creating edge software, and Kinetic GMM for managing the edge applications. Each is described in more detail below.

IoT Edge Gateway Routers

IoT edge gateways are a critical component to the success of your IoT operations because they provide the ability for your IoT network to connect to local devices, applications, and ultimately to your cloud services. Cisco provides IoT gateways that also provide a layer of security to protect your IoT devices and to help prevent your IoT devices from being hacked. Cisco’s IoT gateways can also function as a standalone device, processing data directly from sensors via WiFi, wired, or serial input connections providing a battle-hardened device for the data you want to process at the edge.

There are three Cisco Industrial router models that provide Edge Compute capabilities with an ability to run IOx applications at the edge of the IoT fabric:

- The IR829 is a fleet-targeted mobile gateway that addresses most use cases for fleets. The goal is to provide an off-the-shelf solution for the widest possible number of fleet applications with the fewest number of SKUs and least amount of custom developments possible. This Fleet Gateway is a ruggedized router integrating WiFi and Cellular radios.

- The IR809 targets markets such as Distribution Automation, ATM, POS, Telemetry, Enterprise Fleet, mobile machine-to-machine (M2M), bill boards, and so forth. IR809 brings in 4G LTE capabilities to small form factor M2M routers.

- The IR1101 is like the IR829 and is a general-purpose industrial router but lacks WiFi capability. Additionally, there are essential IOx differences in the IR1101 that are important to note. This information is provided below.

These routers ship with one Cisco provided IOXVM (virtual machine which enables IOx on the platform). All IOx applications run within this virtual environment. In certain cases, a customer may decide to install a custom operating system on the platform. In this situation, IOx support ceases to exist on the platform (and for the customer).

Figure 2  IOx Framework

The Cisco IOx framework provides a powerful platform for your Ops and App Developers to easily deploy applications to your IoT gateways. Cisco IOx utilizes Docker tooling to allow your development teams to build applications in a standard format that is familiar to cloud-native application developers. The Cisco IOx application environment combines the power of the Cisco IOS and the Linux OS to provide highly secure networking. This enables you to execute IoT applications in the fog or at the edge with secure connectivity to the Cisco IOS software and get powerful services for rapid, reliable integration with IoT sensors and the cloud. Cisco IOx enables development and deployment of IoT business and control applications at the edge.

By bringing application execution capability to the source of the IoT data, customers can overcome challenges with high volumes of data and the need for automated, near-real time system responsiveness. Cisco IOx offers consistent management and hosting across network infrastructure products, including Cisco routers, switches, and compute modules. Cisco IOx allows application developers to work in the familiar Linux application environment with their choice of languages and programming models with familiar open-source development tools.
Kinetic GMM for IOx Life-Cycle Management

Building on Cisco IOx is Cisco Kinetic. Kinetic takes the best of Cisco IOx application management, adds IoT gateway management and automates/manages it for you as a cloud service allowing you to scale our operations. This provides ease of use for your Ops teams by allowing for both:

- Auto-provisioning of Cisco IoT gateway devices
- Easily connecting your gateways to your Kinetic account

Fog apps (for edge computing) are Cisco IOx applications that run on a Cisco gateway and transmit data from devices to the Cisco Kinetic cloud or other data destinations. Applications can be deployed and managed with the Cisco Kinetic Gateway Management Module (GMM).

Refer to Figure 4 for the deployment steps for an edge/fog application:

Figure 4 depicts the high-level steps involved as part of the IOx deployment life-cycle.
The Edge/fog application download process is embedded in the overall management of the gateway. When a gateway is powered on, it communicates with GMM to download provisioning information and update/download Edge/Fog applications and ensure that the connected sensors are detected by the gateway. This way, a field technician can easily deploy a gateway that with minimal hands-on effort.

Beyond just the deployment of IOx applications, Cisco Kinetic GMM can be used to manage the entire life-cycle of an IOx application, including:

- Centrally deploy an IOx application to a set of edge gateways.
- Change the state of an IOx application (start/stop/restart).
- Monitor the status of IOx applications.
- Upgrade IOx application to the latest version.
- View container and application logs.
- Monitor events related to IOx application deployment.

The advantage of using Cisco Kinetic GMM is that you can now centrally deploy and manage the application on hundreds of gateways from a centralized pane of glass without having to log in or connect to each of the gateways individually. IOx applications can be deployed to a set of gateways, helping save deployment time and cost.

**IOx Framework Details**

The IR809/829 gateway routers ship with one Cisco provided IOXVM (virtual machine which enables IOx on the platform. All IOx applications run within this virtual environment. In certain cases, a customer may decide to install a custom operating system on the platform. In this situation, IOx support ceases to exist on the platform (and for the customer).

These routers ship with one Cisco-provided IOXVM which enables IOx on the platform. All IOx applications run within this IOXVM.

In certain cases, a customer may decide to install a custom operating system on the platform. In this situation, IOx support ceases to exist on the platform (and for the customer).

**Figure 5  IOx Architecture for IR8x9**

Both the IR829 and the IR809 use the Intel Rangeley Dual-Core CPU, 2GB DDR3 memory, 8MB SPI Bootflash, and 8GB (4GB usable) eMMC bulk flash.

The hypervisor is provided by LynxWorks and it runs on the bare metal hardware upon which IOS and a Guest OS (e.g., Linux) run as two separate virtual machines (VMs).

The hypervisor presents underlying hardware to virtual machines (IOS and Guest OS) as a subset of actual physical hardware. The allocation of devices to VMs is provided by a configuration to the hypervisor. The VMs access a virtual CPU, pre-configured memory regions, pre-divided flash disk storage, and other hardware devices. LynxSecure Separation Kernel v. 5.1 has been selected as a hypervisor.
Each PCI device can be owned exclusively by one VM in the hypervisor architecture. However, IOS and the Guest OS need to access some shared devices, for example eMMC flash. The solution is a Virtual Device Server (VDS), which is a separate VM that owns shared devices. IOS and Guest OS access the virtual devices emulated in hypervisor. The hypervisor and VDS then coordinate access to the shared devices. The VDS also provides communication channels between VMs using emulated Ethernet interfaces.

IOS acts as a gateway to network resources for the Linux partition, using IP and a virtual switch connection provided to the hypervisor. IOS and Linux each operate as a Guest OS of the hypervisor.

The network path from IOXVM to IOS is available via an emulated Ethernet link between them. Either IPv4 or IPv6 protocol can be run on the Ethernet link.

The IOXVM runs CAF and other IOx infrastructure elements on the host Linux and hosts all the LXC applications.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>IR8x9 Platforms—Key Resource Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IOS Image</strong></td>
<td>The minimum IOS version required for IOx support is 15.6(1)T1.</td>
</tr>
<tr>
<td><strong>Application Types Supported</strong></td>
<td>IR800 platforms support PaaS, LXC, and Docker type applications.</td>
</tr>
<tr>
<td><strong>Application Resources Limit</strong></td>
<td>IR800 platforms have a total of 732 units of CPU and 767 MB of memory for applications. Docker type applications can have maximum of 42 layers per application. At any time, IOx supports hosting 100 layers in total for all Docker applications installed on the device.</td>
</tr>
<tr>
<td><strong>Application Networking</strong></td>
<td>IR8x9 platforms support NAT and bridge mode for application networking.</td>
</tr>
<tr>
<td><strong>Device Resources</strong></td>
<td>There are two serial ports (async0 and async1) available for applications. USB storage and USB serial capabilities for application are supported on this platform.</td>
</tr>
<tr>
<td><strong>Application Security</strong></td>
<td>IR8x9 platforms support application signature verification, which is enabled by default.</td>
</tr>
<tr>
<td><strong>IOx Services</strong></td>
<td>IOx services is enabled for IR800 platforms. IOx services like GPS, Motion, Host Device Management, and more are supported out-of-the-box.</td>
</tr>
</tbody>
</table>

**Note:** IOx developer documentation is available on Cisco DEVNET at: [https://developer.cisco.com/site/iox/](https://developer.cisco.com/site/iox/)

Self-provisioning developer sandboxes are available for training and testing. Developer sample code and how-to guides are also available.

IOx services sample applications and tutorials: [https://github.com/CiscoIOx/iox-services-samples/](https://github.com/CiscoIOx/iox-services-samples/)

IOx SDE—IOx SDE is an Ubuntu VM (14.04) with all the tools (Docker, ioxclient) required to build an IOx application package pre-installed. Download and import it as a VM to get started.

**Login Credentials for IOx SDE VM:**

Username: root
Password: ioxsde

**Note:** Use the NAT mode if VM does not obtain the IP address in bridge mode.

Download SDE V1.7.0: [https://developer.cisco.com/files/iox-sde.ova](https://developer.cisco.com/files/iox-sde.ova)

The IR1101 modular gateway also supports hosting IOx applications. However, the IOx architecture for the IR1101 differs from other Cisco platforms that use the hypervisor approach. IOx runs as a process on the IR1101 versus as a virtual machine on others. Additionally, the only type of container supported on the IR1101 is the LXC container.
IOx nodes may have different CPU architectures, so it becomes very complex to characterize an application’s performance or requirements on each of them. IOx attempts to bring consistency and uniformity by using the notion of resource profiles. A resource profile encapsulates a set of resources (CPU, memory, etc.) under a unique name in a consistent fashion across all Cisco IOx platforms.

The intent of resource profiles is to allow developers to obtain some level of consistency when they test their applications on one platform and want to deploy the same applications on other platforms. Currently, only CPU and memory are characterized under a resource profile.

**Resource Profile Definitions**

Currently, the pre-defined resource profiles in Table 2 are provided. However, the exact set of profiles supported on a specific platform is a function of the available resources on that platform. Also, if one of the pre-defined resource profiles do not meet your requirements, you can always define your own custom profile based on the resources required for your application.

<table>
<thead>
<tr>
<th>Profile Name</th>
<th>Memory (mb)</th>
<th>CPU (units)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1.tiny</td>
<td>32</td>
<td>100</td>
</tr>
<tr>
<td>c1.small</td>
<td>64</td>
<td>200</td>
</tr>
<tr>
<td>c1.medium</td>
<td>128</td>
<td>400</td>
</tr>
<tr>
<td>c1.large</td>
<td>256</td>
<td>600</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>512</td>
<td>1200</td>
</tr>
</tbody>
</table>

- Memory assignment is platform agnostic and can be directly assigned based on the application requirements irrespective of the platform. However, CPU resources are highly platform dependent and performance may vary based on the underlying CPU architecture.

- To abstract these disparate characteristics to application developers and provide a uniform and consistent view, CPU resources are expressed in the form of “units”. This means an app with “x” CPU units would have similar performance on all Cisco supported heterogeneous platforms.

- The CPU unit values for a platform are obtained by executing standard benchmarking tools on that platform and assigning a unit value based on their relative score when compared against a standard base platform.

- To get a relative idea of what those units could mean in comparison to a standard CPU, here is a sample comparison with a generic x86 Intel platform:
Based on the benchmarking results, an x86-based 64 bit Intel Xeon processor with one core of CPU @ 2GHz will have 10000 CPU units. Based on this value developers can extrapolate and test an application in their devnet sandbox environment based on the same CPU characteristics and check the CPU units required for an app.

A note about resource bounds:

- The CPU units allocated to an app are the minimum guaranteed. This means that at any given point in time depending upon the number of applications running, an app under consideration will get the guaranteed CPU units and may even get more than that if there is no other contention for CPU units.

- Memory, however, is a **hard limit**. That is, at no point in time will the application get more memory than what is defined. Going beyond this limit typically results in a KILL signal to the application.

### Table 3  Resources Available on Devices

<table>
<thead>
<tr>
<th>Platform</th>
<th>CPU Architecture</th>
<th>CPU (Units)</th>
<th>Memory (MB)</th>
<th>Storage (MB)</th>
<th>Supported Application Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR829/IR809</td>
<td>Intel 64 bit (x86_64)</td>
<td>732</td>
<td>767</td>
<td>512-1800</td>
<td>PaaS, LXC, Docker</td>
</tr>
<tr>
<td>IR1101</td>
<td>ARM 64-bit (aarch64)</td>
<td>1000</td>
<td>862MB</td>
<td>512MB</td>
<td>LXC, Docker</td>
</tr>
</tbody>
</table>

### IOx Application Package

#### Anatomy of an Application Package

An IOx application package consists of:

- One package descriptor file named package.yaml, which should be in the root of the package.

- Zero or one application configuration file named package_config.ini. If present, it should be in the root of the package.

- Zero or one application manifest named package.mf. If present, it should be in the root of the package.

- Zero or one certificate containing signing information named package.cert. If present, it should be in the root of the package.

- One tar.gz envelope containing application or service artifacts with the name artifacts.tar.gz. These artifacts may be binaries, application code, application libraries, virtual disks, rootfs, etc. More details are provided in the following sections.

#### Package Format

An IOx application package should be packaged in one of the following file formats: “tar” or “tar.gz”.

#### Package Descriptor File

The contents of this file capture application and service metadata, requirements, etc., in a YAML format. It should be named "package.yaml". The specifications of this file are covered in [https://developer.cisco.com/docs/iox/#!/package-descriptor/iox-package-descriptor](https://developer.cisco.com/docs/iox/#!/package-descriptor/iox-package-descriptor)
Package Configuration File

In order to bootstrap applications or services present in the package, IOx framework supports externalizing this content into a separate .ini file and provides mechanisms to edit and update the contents of this file during the provisioning of the application. This file contains sections with key, value pairs adhering to .ini file format. The name of the file if present should be package_config.ini. The administrative tools provide mechanisms to modify this file with the correct values during the installation process so that the application can be bootstrapped with the correct values.

Application Artifacts

All app artifacts are maintained in its own tarball, which offers a cleaner separation of application artifacts. This inner envelope will always be a tar.gz and is named artifacts.tar.gz. This will typically be generated by tooling (ioxclient, SDK, etc.).

Implementation

Building a Dockerized IOx NodeJS Application and Deploying it Using Kinetic GMM

This section describes how to use Docker tooling to create an IOx application. Specifically, it shows how to create a Docker image to run a simple Node.js-based HTTP server and create an IOx application from it.

Figure 6  Using Docker Tools to Generate IOx Applications

Installing Docker Tools and Images

The development machine should have the following installed:

- **ioxclient (version > 1.4.0)**
- **Docker >= version 1.12 (version 1.26 preferred)**
  - Install Docker for Mac: [https://docs.docker.com/docker-for-mac/install/](https://docs.docker.com/docker-for-mac/install/)
Remote and Mobile Assets—Edge Compute

Implementation

Figure 7  Verifying the Docker and IOx Client Versions on the Development Server

```
SAZAD-M-C4B0:~ Sameer$ docker --v
Docker version 18.09.2, build 6247962
SAZAD-M-C4B0:~ Sameer$ ioxclient --version
ioxclient version 1.7.0.0
```

The sample application code is maintained at: https://github.com/CiscoIOx/docker-nodejs.git

Clone this sample code and use branch master.

Figure 8  Cloning the Sample Code Using git

```
SAZAD-M-C4B0:~ Sameer$ git clone https://github.com/CiscoIOx/docker-nodejs.git -b master
Cloning into 'docker-nodejs'...
remote: Enumerating objects: 10, done.
remote: Total 10 (delta 0), reused 0 (delta 0), pack-reused 10
Unpacking objects: 100% (10/10), done.
Checking connectivity... done.
```

Figure 9  Inspecting the Contents of the Cloned Directory

```
SAZAD-M-C4B0:~ Sameer$ cd docker-nodejs/
SAZAD-M-C4B0:~/docker-nodejs Sameer$ ls
Dockerfile README.md activation.json package.yaml server.js
```

This is a simple Node.js-based HTTP server that performs the following:

- Sets up signal handlers to shutdown gracefully.
- Inspects CAF_APP_LOG_DIR environment variable and sets up logging to a file accordingly.
- Starts HTTP server on port 8000.
- Logs the source of request and responds with “Hello World!”.

Creating the Docker Image

This section describes how to create a Docker image with the above application in it. The process involves using alpine:3.3 as the base image, installing Node.js, and setting up the application.

The following Dockerfile accomplishes these tasks:

```
FROM alpine:3.3

RUN apk add --update nodejs
COPY server.js /server.js

EXPOSE 8000
CMD ["node", "/server.js"]
```

Next build an image from this Dockerfile and tag it with a name (samplenode:1.0).
Remote and Mobile Assets—Edge Compute

Implementation

Figure 10  Building the Docker Image

```
$AZAD-M-C4BQ:docker-nodejs Sameer$ docker build -t samplenode:1.0 .
$Sending build context to Docker daemon 65.02kB
Step 1/5 : FROM alpine:3.3
  ---> a6f1dbf804
Step 2/5 : RUN apk add --update nodejs
  ---> Running in 322bcefe5627
fetch http://dl-cdn.alpinelinux.org/alpine/v3.3/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.3/community/x86_64/APKINDEX.tar.gz
(1/4) Installing libc6 (5.3.0-r0)
(2/4) Installing libstdc++ (5.3.0-r0)
(3/4) Installing libuv (1.7.5-r0)
(4/4) Installing nodejs (4.3.2-r1)
Executing busybox-1.24.2-r2.trigger
OK: 29 MIB in 15 packages
Removing intermediate container 322bcefe5627
  ---> 830723b6b88a
Step 3/5 : COPY server.js /server.js
  ---> afa9ad21dc99
Step 4/5 : EXPOSE 8000
  ---> Running in 684fcdb49a5
Removing intermediate container 684fcdb49a5
  ---> 1f4c88b6b1e38
Step 5/5 : CMD ["node", "/server.js"]
  ---> Running in 70908b44b042
Removing intermediate container 70908b44b042
  ---> ebaba2d5ae4c
Successfully built ebaba2d5ae4c
Successfully tagged samplenode:1.0
```

Figure 11  Viewing the Docker Image

```
$AZAD-M-C4BQ:docker-nodejs Sameer$ docker images
REPOSITORY                  TAG       IMAGE ID           CREATED             SIZE
samplenode                  1.0       ebaba2d5ae4c       7 seconds ago       24.6MB
```

Run the image locally and test to ensure it is functioning correctly.

Figure 12  Verifying the Docker Image Locally on the Development Server

```
$AZAD-M-C4BQ:docker-nodejs Sameer$ docker run -it -p 8000:8000 samplenode:1.0
Setting up logging to file server.log
Server running at http://0.0.0.0:8000/
Request from 172.17.0.1
Response sent...
Request from 172.17.0.1
Response sent...
```

If the server sends a response to an incoming request, then the Docker image and the application should be working correctly.

Writing Package Descriptor File

Once the Docker image has been created, the developer needs to write a package descriptor file specifying requirements for the application. Here is a sample package.yaml file:

```
---
descriptor-schema-version: "2.2"

info:
  name: SampleNodeApp
```
Implementation

```yaml
description: "Simple Docker Style app that runs a nodejs server"
version: "1.0"
author-link: "http://www.cisco.com"
author-name: "Cisco Systems"

app:
  # Indicate app type (vm, paas, lxc etc.,)
cpuarch: "x86_64"
type: docker
resources:
  profile: c1.small
  network:
    - interface-name: eth0
      ports:
        tcp: [8000]

  # Specify runtime and startup
startup:
  rootfs: rootfs.tar
  target: ["node", "/server.js"]
```

The following are important to note for compatibility:

- Descriptor schema version is 2.2, which is the minimum version that supports Docker style apps.
- Note that the cpuarch is x86_64. Alpine-based apps can only run on x86_64 bit machines.
- App type is docker.
- The required port (8000) to be opened is specified under network->ports.
- rootfs.tar is the name of the file containing the Docker image (output of Docker save command). More details are provided in the following sections.
- Command to be run when starting up the app is ["node", "/server.js"]. Note that server.js was copied to "/" of rootfs.

Creating an IOx Application Package

Once the required Docker image (samplenode:1.0) and package.yaml are created, create an IOx application package from these artifacts. ioxclient (>= 1.4.0) provides a convenience command that generates an IOx application package from a Docker image and package.yaml file.
Remote and Mobile Assets—Edge Compute

Implementation

**Figure 13 Creating the IOx Application Package**

```
SAZAD-M-C4B0:docker-nodejs Sameer$ ioxclient docker package samplenode:1.0 .
Currently active profile: 889-1
Command Name: docker-package
Using the package descriptor file in the project dir
Validating descriptor file package.yaml with package schema definitions
Parsing descriptor file...
Found schema version 2.2
Loading schema file for version 2.2
Validating package descriptor file...
File package.yaml is valid under schema version 2.2
Generating IOx package of type docker with rootfs consisting of layers
Replacing symbolically linked layers in docker rootfs, if any
No symbolically linked layers found in rootfs. No changes made in rootfs
Removing emulation layers in docker rootfs, if any
The docker image is better left in it's pristine state
Updated package metadata file: /Users/Sameer/docker-nodejs/.package.metadata
No rsa key and/or certificate files provided to sign the package

Generating the envelope package
```

Checking if package descriptor file is present...
Skipping descriptor schema validation..
Created Staging directory at: /var/folders/jp/49h_w0vd1rb251pyssp0f1pbc0000gp/T/693267774
Copying contents to staging directory
Creating an inner envelope for application artifacts
Including rootfs.tar
Generated /var/folders/jp/49h_w0vd1rb251pyssp0f1pbc0000gp/T/693267774/.artifacts.tar.gz
Calculating SHA1 checksum for package contents..
Parsing Package Metadata file: /private/var/folders/jp/49h_w0vd1rb251pyssp0f1pbc0000gp/T/693267774/.package.metadata
Updated package metadata file: /private/var/folders/jp/49h_w0vd1rb251pyssp0f1pbc0000gp/T/693267774/.package.metadata
Root Directory: /private/var/folders/jp/49h_w0vd1rb251pyssp0f1pbc0000gp/T/693267774
Output file: /var/folders/jp/49h_w0vd1rb251pyssp0f1pbc0000gp/T/001268101
Path: .package.metadata
SHA1: 53e0ff931dd88848b390e21534672528ba13d17
Path: .artifacts.tar.gz
SHA1: 11cc365eb2a53d3827a116e1407f2b2b86cd9
Path: envelope_package.tar.gz
SHA1: b95ea38d324732d623458cc3cd5d641cf3f314
Path: package.yaml
SHA1: 852be35786058253da12646a45723e95e0ef9d0
Generated package manifest at package.yaml
Generating IOx Package..
Package docker image samplenode:1.0 at /Users/Sameer/docker-nodejs/package.tar
```

The package.tar file is an IOx application package that can be used to deploy on an IOx platform.

**Note:** The package.yaml uses rootfs.tar as the name of startup->rootfs. This is essential, since ioxclient saves the Docker image with the name rootfs.tar.

**Deploying the IOx Application to the Gateway using Cisco Kinetic GMM**

To deploy the IOx application to the gateway, follow these steps:

1. Upload the IOX Application tar file onto Kinetic GMM.

   Log in to the Kinetic GMM UI using your credentials. Click **Applications** → **Add Application**. Select the package.tar file created above. Click **Import**.
Remote and Mobile Assets—Edge Compute

Implementation

Figure 14  Uploading IOx Application tar File onto Kinetic GMM

Add Application

2. Verify that the upload was successful.

Once the upload is successful, the application should show in the Available state under the Applications tab.

Figure 15  Verifying IOx Application tar Successfully Uploaded onto Kinetic GMM

3. Install the application onto a gateway.

- Select the SampleNodeApp application and click Install. In the pop-up window, select the appropriate application resource profile. In this case, the c1.small profile providing 200 CPU units and 64 MB RAM should be sufficient.
- Select the gateway or gateways on which you want this application to be installed from the drop-down menu.
- Leave the interface Name as default “eth0”.
- For the network name, select IOx-nat0 since access to the Node.js application from outside is required. Internally, the Node.js application listens on port 8000. Externally map this to the desired port. In the example below, it is mapped it to an external port of 8000. Hence the Node.js application can also be accessed on port 8000 externally.

Note: For more information about IOx application networking, refer to:
https://developer.cisco.com/docs/iox/#!application-networking/application-networking

- Click Install to begin installation of the IOx application onto the selected gateway(s). In the example below, a single gateway is selected on which to install this application.
4. Monitor the deployment.

Once you click Install, you can navigate to the Instances tab and see the application state and status as “Deploying”.

Figure 17  Monitoring IOx Application Deployment

In a few minutes you should see your application in the “Running” state. It will also display a corresponding IP address/port combination on which the application can be externally accessed.

Figure 18  IOx Application in Running State

A similar application status can be viewed under the Gateways tab by clicking the appropriate gateway on which the application has been installed and navigating to Apps. You can also view the application state in the Summary tab.
Remote and Mobile Assets—Edge Compute

5. Viewing the Gateway Event Log.

In case of any deployment errors, you can view the logs on the **Event Log** tab.

6. Externally accessing the application.

One way to access the Node.JS IOx application is to VPN into the router using the Kinetic GMM management tunnel and access the application on the specified IP address and port.

In order to VPN to the router, navigate to the **Gateways** tab, select the gateway in question, and click **VPN**.

A pop-up window displays the requisite information needed to connect to the gateway using the Cisco AnyConnect VPN application. You need to enter your credentials to view the decrypted password.
Open up your AnyConnect Client and connect to the gateway using a secure VPN tunnel.

Once connected, open a web browser and connect to the gateway on the IP address and port specified for the IOx application that was deployed above. You should see the message “Hello World!”
7. Viewing Application and Container logs.

Navigate to the Apps tab for the gateway on which the IOx application is deployed and click the application. Expand the Application Logs tab by clicking the + (plus) icon.
8. Building an updated version of the IOx application.

You can modify the application so that it responds with “Hello World! Welcome to my IOx Application” instead of just “Hello World!”. Do this by modifying the “response.end” line in the server.js file.

```
// Configure our HTTP server to respond with Hello World to all requests.
var server = http.createServer(function (request, response) {
  console.log("Request from " + request.headers['x-forwarded-for'] || request.connection.remoteAddress)
  response.writeHead(200, {'Content-Type': 'text/plain'});
  response.end("Hello World! Welcome to my IOx Application\n");
  console.log("Response sent.");
});
```

Create a new Docker image with a version tag of, for example, “2.0”.
Remote and Mobile Assets—Edge Compute

Implementation

Figure 27  Building Docker Image with Updated Code

```
SAZAD-M-C48Q:docker-nodejs Sameer$ docker build -t samplenode:2.0 .
Sending build context to Docker daemon 9.389MB
Step 1/5 : FROM alpine:3.3
 ---> a6fc1dbfa81a
Step 2/5 : RUN apk add --update nodejs
 ---> Using cache
 ---> 0560723b068a
Step 3/5 : COPY server.js /server.js
 ---> ff9474b66fba
Step 4/5 : EXPOSE 8080
 ---> Running in 41a3cd81ef67
Removing intermediate container 41a3cd81ef67
 ---> a3d0959de4a7
Step 5/5 : CMD ["node", "/server.js"]
 ---> Running in ebbae1b2038
Removing intermediate container ebbae1b2038
 ---> ae7d79f897d7
Successfully built ae7d79f897d7
Successfully tagged samplenode:2.0
SAZAD-M-C48Q:docker-nodejs Sameer$
```  

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>CREATED</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>samplenode</td>
<td>2.0</td>
<td>a87d79f697d7</td>
<td>5 seconds ago</td>
<td>24.6MB</td>
</tr>
<tr>
<td>samplenode</td>
<td>1.0</td>
<td>ebba2d5ae04c</td>
<td>20 hours ago</td>
<td>24.6MB</td>
</tr>
</tbody>
</table>

Change the application version number in the package.yaml file to "2.0".

Figure 28  Updating Version in package.yaml

```
descriptor-schema-version: "2.2"
info:
  name: SampleNodeApp
  description: "Simple Docker Style app that runs a nodejs server"
  version: "2.0"
  author-link: "http://www.cisco.com"
  author-name: "Cisco Systems"
```

Build an updated version of the package.tar file with the updated version of the application. Make sure to use the updated tag of "2.0" for the Docker image.
There is now an updated version of the package.tar file based on the updated Docker image of samplenode:2.0.

9. Upgrading the deployed application to version 2.0 using Kinetic GMM.

Navigate to Applications and click the application you want to upgrade.
Click the **Upgrade** button and you see a pop-up screen where you can upload the updated version of the package.tar file that was built in the previous step. You see a small warning at the bottom of the pop-up screen indicating that all of the IOx applications will be upgraded to the newer version on all applicable gateways. Check the box **I understand the risks. Proceed with editing** and click **Upgrade**.

Once the upgrade is successful, the application version is updated to “2.0”.

---

**Figure 30** Upgrading to the Newer Version of the IOx Application onto Kinetic GMM

**Figure 31** Uploading Updated IOx Application onto Kinetic GMM

**Figure 32** Verifying Updated Version of IOx Application
It is also configuring the updated version of the IOx application on any associated gateways.

Figure 33  Gateway Being Updated with Newer Version of IOx Application

Once version 2.0 of the application is rolled out to all the associated gateways, AnyConnect VPN to any one of them and verify that the updated version of the application is running.

Figure 34  Verifying Newer Version of IOx Application Successfully Running

10. Stopping the application.

To stop the application, click the check box next to the application and click **Stop App**.

Figure 35  Stopping IOx Application

In a similar manner you can click the checkbox next to the application and click **Start App** to start the application.

11. Uninstalling the application.

To uninstall the application, click the check box next to the application and click **Uninstall**. A pop-up screen is displayed asking you to confirm that you really want to uninstall the application. You can confirm by clicking the **Delete** button. You see the status message “Removing” and within a short time you will see that the application has been uninstalled from your gateway.
### Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>Authentication, Authorization, and Accounting</td>
</tr>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>APN</td>
<td>Access Point Name</td>
</tr>
<tr>
<td>AR</td>
<td>Active Router</td>
</tr>
<tr>
<td>CAPWAP</td>
<td>Control and Provisioning of Wireless Access Points</td>
</tr>
<tr>
<td>CLB</td>
<td>Cluster Load Balancing</td>
</tr>
<tr>
<td>CVD</td>
<td>Cisco Validated Design</td>
</tr>
<tr>
<td>DMVPN</td>
<td>Dynamic Multipoint VPN</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DPD</td>
<td>Dead Peer Detection</td>
</tr>
<tr>
<td>EAP</td>
<td>Extensible Authentication Protocol</td>
</tr>
<tr>
<td>EAPoL</td>
<td>EAP over LAN</td>
</tr>
<tr>
<td>EEM</td>
<td>Embedded Event Manager</td>
</tr>
<tr>
<td>GMM</td>
<td>Cisco Gateway Management Module</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>GPT</td>
<td>Cisco Kinetic Gateway Provisioning Tool</td>
</tr>
<tr>
<td>GRE</td>
<td>Generic Routing Encapsulation</td>
</tr>
<tr>
<td>HER</td>
<td>Headend Router</td>
</tr>
<tr>
<td>HSPA</td>
<td>High Speed Packet Access</td>
</tr>
<tr>
<td>HSRP</td>
<td>Hot Standby Router Protocol</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IKE</td>
<td>Internet Key Exchange</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IPS</td>
<td>Intrusion Prevention System</td>
</tr>
<tr>
<td>IR</td>
<td>Industrial Router</td>
</tr>
<tr>
<td>ISAKMP</td>
<td>Internet Security Association and Key Management Protocol</td>
</tr>
<tr>
<td>ISE</td>
<td>Cisco Identity Services Engine</td>
</tr>
<tr>
<td>LAP</td>
<td>Lightweight Access Point</td>
</tr>
<tr>
<td>LLG</td>
<td>Least Loaded Gateway</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>LWAP</td>
<td>Lightweight Access Point</td>
</tr>
<tr>
<td>MIMO</td>
<td>Multiple-Input and Multiple-Output</td>
</tr>
<tr>
<td>MPLS</td>
<td>Multiprotocol Label Switching</td>
</tr>
<tr>
<td>MQC</td>
<td>Modular QoS</td>
</tr>
<tr>
<td>mSATA</td>
<td>mini-Serial Advanced Technology Attachment</td>
</tr>
<tr>
<td>NAT</td>
<td>Network Address Translation</td>
</tr>
<tr>
<td>NGE</td>
<td>Cisco Next-Generation Encryption</td>
</tr>
<tr>
<td>NHRP</td>
<td>Next Hop Resolution Protocol</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>PoE</td>
<td>Power over Ethernet</td>
</tr>
<tr>
<td>PSK</td>
<td>Pre-Shared Keys</td>
</tr>
<tr>
<td>RaMA</td>
<td>Cisco Remote and Mobile Assets</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
</tr>
<tr>
<td>RHEL</td>
<td>Red Hat Enterprise Linux</td>
</tr>
<tr>
<td>RTU</td>
<td>Remote Terminal Unit</td>
</tr>
<tr>
<td>SCADA</td>
<td>Supervisory Control and Data Acquisition</td>
</tr>
<tr>
<td>SFP</td>
<td>Small Form-Factor Pluggable</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identification Module</td>
</tr>
<tr>
<td>SVI</td>
<td>Switched Virtual Interface</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>VIP</td>
<td>Virtual IP address</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>VRF</td>
<td>Virtual Route Forwarding</td>
</tr>
<tr>
<td>VTI</td>
<td>Virtual Tunnel Interface</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>vWLC</td>
<td>virtual Wireless LAN Controller</td>
</tr>
<tr>
<td>WAF</td>
<td>Web Application Firewall</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>WGB</td>
<td>Workgroup Bridge</td>
</tr>
<tr>
<td>WLC</td>
<td>Cisco Wireless LAN Controller</td>
</tr>
<tr>
<td>ZTD</td>
<td>Zero-Touch Deployment</td>
</tr>
</tbody>
</table>