SPDX-License-Identifier: Apache-2.0
Copyright © 2019 Intel Corporation and Smart-Edge.com, Inc.

OpenNESS Controller

The OpenNESS Controller is used to manage and control OpenNESS Edge Nodes, and is a component of the OpenNESS multi-access edge computing (MEC) platform. This repository contains the source for the the OpenNESS Controller. Familiarize yourself with the OpenNESS Edge Node if you wish to take full advantage of this repository.

The Controller features the following components:

- ☑ **Core** backend to manage Edge Nodes (gRPC) and provide an administrative REST API.
- ☑ **Web UI** frontend (Controller) as a user-facing alternative to using the REST API.
- ☑ **Web UI** frontend (3GPP CUPS) to send API calls to the 3GPP mobile core.
- ☑ **Database** backend (MySQL 8.0 or higher).
- ☑ **Telemetry** backend (`syslog` & `statsd` compliant) listeners for telemetry from Edge Nodes.
- ☑ **Kubernetes** support (Kubernetes API 1.0 or higher) to have a Kubernetes master manage container orchestration.

OpenNESS supports Network Edge and On-Premise edge deployment. For details of these two deployment model please refer to the OpenNESS architecture specification.

- Network Edge deployment is based on Kubernetes
- On-Premise Edge deployment is based on Docker Containers and VMs based on libvirt.

The OpenNESS Edge Node and OpenNESS controller components build would remain unchanged for both the deployments. For the Network Edge deployment this document assumes that a Kubernetes cluster is setup with Kubernetes master and Node is added to the cluster. The document does not provide the steps for setting up Kubernetes cluster.

Reference for setting up Kubernetes cluster on CentOS: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

The Core, Web UIs, Database and Telemetry components are packaged as Docker containers when you build this project using the instructions below.

Quick Start

> ***Note***: *This is only a quick start and is not a full setup guide. This document has additional sections with details of the setup and operation of the Controller.*

1. Clone this repository to your local system. You must have `make`, `docker` and `docker-compose` utilities installed and in your `PATH`.

2. If this is your first time starting the controller, build the Docker images by running the following command from the project directory. **You should run this command any time there is a new release of the project to rebuild the Docker images**:

(skip to step 3 if you have already done this and you are starting the controller a subsequent time)

```
make build
```

3. From the project directory, run the following command to start the Controller:

```
make all-up
```

4. Confirm that you are able to browse to `http://localhost:3000` and access the UI.

5. Log in to the Web UI with username `admin` and password `changeme`.

To shutdown the Controller and preserve all data run the following command from the project directory:

```
make all-down
```

To clean up all data, you can run the following command from the project directory (this action cannot be reversed):

```
make clean
```

Overview

The following sections detail the design and function of different parts of the Controller and they will help you get familiar with the terminology and where they apply to the platform.

Operational Modes

The Controller is able to operate in two modes with some key difference(s) outlined below:

- ☑ **Premise Edge**: Deploy and manage container apps using the platform's native Docker orchestrator.
- ☑ **Network Edge**: Deploy and manage container apps using a Kubernetes orchestrator.

Authentication

Externally exposed components of the Controller are protected by different levels of authentication:

- ☑ **REST API** backend authenticates requests using JSON web tokens (JWT). The API user must provide this token in their HTTP request authorization headers as a `Bearer` token for all API requests.
- ☑ **gRPC API** endpoints authenticate using TLS mutual authentication with X.509 certificates issued by the platform. These endpoints are used to send/receive RPCs to/from the Controller and Edge Node.
- ☑ **Web UI** frontend (Controller) authenticates requests to the REST API using JWT and automatically adds the token to the HTTP request authorization headers.
- ☑ **Database** authentication uses a data source name (DSN) which by default uses password authentication. You can define a different DSN if you have an existing MySQL database that you want to leverage for the Controller.
- ☑ **Telemetry** endpoints authenticate using TLS mutual authentication with X.509 certificates issued by the platform.
- ☑ **Edge Nodes** authenticate gRPC and Telemetry using TLS mutual authentication with X.509 certificates issued by the platform. When an Edge Node is added to the Controller, a user inputs a serial that is cryptographically validated for the connecting Edge Node.

See the Security Overview for more authentication details for the above components.

If additional security and authentication layering is needed for these components, please see the Prerequisites section below.

Nodes

Users of the Controller manage edge compute systems called Edge Nodes (*Nodes* for short). A majority of the operations of the Controller center around actions you can perform on a Node.

Interfaces

Users of the Controller may manage the network interfaces of a Node to assign an interface to either a kernel or userpace process for network I/O.

For example, a userspace process using the Data Plane Development Kit (DPDK) can accelerate network performance on an Edge Node for a given interface.

Apps

Users of the Controller may add, deploy and manage applications (VMs or containers) to their Edge Nodes. When a user adds an application to the Controller, it joins a registry of other uploaded apps which may be deployed to a Node.

Policies

Users of the Controller may manage traffic policies for interfaces and applications. Traffic policies contain sets of traffic rules that define traffic flows towards an interface or application. When a user creates a policy in the Controller, it joins a registry of other traffic policies which may be deployed to a Node.

DNS

Users of the Controller may manage DNS records in the DNS server on the Edge Node.

For example, a DNS record can be created for the address of an application runnning on the Edge Node. This allows clients to access applications by using a Fully Qualified Domain Name (FQDN) instead of an IP address.

Telemetry

Suppporting basic monitoring and telemetry capabilities, the Controller can receive telemetry from any Edge Node. The data is written to a local file, which may be incorporated into your existing monitoring solutions or used as a basic means of monitoring.

3GPP CUPS

This project includes a Web UI for the management of the 3GPP evolved packet core (EPC) elements that implement Control and Userplane Separation (CUPS). The Control Plane typically resides in a central location, while the User Plane is typically adjacent to or within an Edge Node.

Prerequisites

Clone this repository onto the system you wish to setup the Controller and ensure the prerequisites below are met.

Ansible scripts are provided for users that are familiar with the tooling and can be found here.

OS

The Controller is known to work on the following operating systems (however, any system that supports Docker should work):

- Ubuntu (16.04 or higher)
- CentOS (7.6.1810 or higher)
- macOS (10.13 or higher)

Tools

The Controller depends on these prerequisite utilities:

- `make` (version 3.81 or higher) (https://www.gnu.org/software/make/)
- `docker` (version 18.09.2 or higher) (https://docs.docker.com/install/)
- `docker-compose` (version 1.23.2 or higher) (https://docs.docker.com/compose/)

Networking

The Controller sends and receives on these ports and protocols:

Component	Type	Direction	Port	Protocol	TLS	Description
Controller UI	External	Inbound	3000	TCP	No	Used by users to access the UI.
3GPP CUPS UI	External	Inbound	3010	TCP	No	Used to users to access the UI.

Component	Type	Direction	Port	Protocol	TLS	Description
Syslog API	External	Inbound	6514	TCP	Yes	Used by Nodes to send telemetry.
REST API	External	Inbound	8080	TCP	No	Used by users to access the API.
gRPC API	External	Inbound	8081	TCP	Yes	Used by Nodes to talk to Controller.
MySQL DB	Internal	Outbound	8083	TCP	No	Used by Controller for persistence.
Statsd API	External	Inbound	8125	TCP	Yes	Used by Nodes to send telemetry.
Node (ELA) API	External	Outbound	42101	TCP	Yes	Used by Controller to send Node commands.
Node (EVA) API	External	Outbound	42102	TCP	Yes	Used by Controller to send Node commands.

While the Controller listens on these ports by default, since it is set up with Docker, you can optionally expose the components on any listening ports based on your operating constraints. For outbound traffic, you may find a need to set up a proxy. Docker supports a proxy and more documentation can be found here.

Proxy configuration might be needed in the following components depending your organization environment

1. Linux environment variable

```
cat /etc/environment
http_proxy=http://<proxy>:<port>
https_proxy=http://<proxy>:<port>
ftp_proxy=http://<proxy>:<port>
no_proxy=localhost,127.0.0.1
HTTP_PROXY=http://<proxy>:<port>
HTTPS_PROXY=http://<proxy>:<port>
FTP_PROXY=http://<proxy>:<port>
NO_PROXY=localhost,127.0.0.1
```

2. Yum config file

```
cat /etc/yum.conf
proxy=http://<proxy>:<port>
```

You should always take precaution when exposing ports on your network. Contact your system/network administrator if you require ports to be opened up for any of these communications or if you are uncertain of your network topology.

> **Note**: It is *strongly* encouraged to use TLS termination on **external, non-TLS** components if you plan to set this up as a public-facing system. The TLS certificates should be signed by a trusted Internet certificate authority. For example, NGINX and Let's Encrypt can be used to to secure non-TLS endpoints with publicly trusted certificates.

When deploying applications to the Edge Node, you provide a URI for the source image. The Edge Node downloads this source image directly from the URI. It is expected that you provide an HTTPS URI for the source image.

Any static content hosting software, such as NGINX and Let's Encrypt, or cloud services provide file or object storage over an HTTPS endpoint, can be used to serve image data.

Setup

The following section provides a more detailed overview of the Controller setup and the available customization. It assumes you've already become familiar with the Quick Start.

Basic Setup

The Controller uses user-defined configurations from the `.env` included in this repository. Review the file here to familiarize yourself with the different types of configuration parameters that you must be defined and optionally changed. Some variables in the `.env` require that you rebuild the Docker images. Be sure to see the comments in the `.env` for more details.

For example, you can set a different admin password by editing the file.

Additionally, if the Controller will be deployed on a remote system you will need to update the API URL.

It is highly encouraged that you change the default credentials in the `.env` file and follow industry best practices to protect network services with IDS, IDP, and firewall systems if it will be exposed to the internet.

Advanced Setup

As more advanced setup instructions are contributed, they will be found here. For now, reference the `.env` file for the types of customizations are available.

If you wish to use Ansible to set up the Controller, it follows a slightly different procedure. Please see the Ansible documentation to learn more about how to set it up.

Usage

You can use the web UI or the REST API to perform Controller operations. The following sections detail some common usages. If you are using the REST API directly, please use the API schema as a primary reference, and use the sections below as complementary information. The usage sections below are describing the UI interactions and they assume you are already logged into the Controller.

Check out our documentation repository for more tips on operating the platform.

Telemetry

If you wish to know about the runtime status of the Controller, you should utilize `docker-compose` to monitor the logs.

If you wish to know about the runtime status of the Edge Node(s), inspect the `artifacts/syslog.log` and `artifacts/statsd.log` files.

These files are written to disk on the Controller host and can be inspected for telemetry.

Note: If the stats file is empty the connected Edge Nodes do not support sending statistics.

Nodes

One of the first things you'll likely do is add an Edge Node to the Controller. To do this, click the "ADD EDGE NODE" button.

You will be prompted to provide identifying information of the Node you're about to add. The most critical field is the `Serial` field. The Edge Node outputs this serial key upon setup and will try to connect to the Controller continously. You must add the Node by its serial or the Controller will not recognize it.

Once you fill out the details for the Node and submit the form you will see the Node added to your Controller's list of Nodes. Currently, the best way to know that your Node is connected is by observing telemetry. If you are receiving telemetry (logs or stats) from the Edge Node, it successfully acquired a certificate and can be used for other functions.

If you need to make edits or delete the Node at a later time, you can do so from the UI and/or API.

Interfaces

Another thing you'll likely do after adding the Node to the Controller is configure its interfaces. These operations currently occur live on the Node, so it important that you are aware of the impact of making these changes if the Node is being used in a more mature capacity.

You may wish to set an interface to operate in userspace mode (DPDK-managed) and make changes to its flow direction. It is recommended you read the Edge Node documentation to learn more about these options in setting up the Edge Node.

Apps

Apps (applications) can be added to the Controller for deployment onto one or more Edge Nodes. Your app may be a Docker image or a Virtual Machine (VM) image. To add the app, click on the "ADD APPLICATION" button. You will be prompted to provide identifying information of the app you're about to add. For example, you'll be able to provide memory, compute and networking details. You'll also be able to provide a source URL for where to pull the image. **This URL must be reachable from the Edge Node.**

Once you fill out the details for the app and submit the form you will see the app added to your Controller's list of apps. When you want to deploy the app to the Node, navigate to the Node's detail page and choose the app that you uploaded.

If you need to control the state of the app or delete it at a later time, you can do so from the UI and/or API.

Policies

Policies can be added to the Controller for deployment onto one or more Edge Nodes. The policies, can be assigned to an interface or to an app. To add the policy, click on the "ADD POLICY" button. You will be prompted to provide identifying information of the policy you're about to add. For example, you'll be able to set up traffic rules to identify source, destination and target actions for traffic on the Edge Node.

Once you fill out the details of the policy and submit the form you will see the policy added to your Controller's list of policies. When you want to apply the policy to the interface or app on the Node, navigate

to the Node's detail page and click the "Policy" button.

If you need to edit or delete the policy at a later time, you can do so from the UI and/or API.

DNS

DNS can be configured directly on the Edge Node individually. The primary DNS usage is setting up `A` records on the Edge Node. The A records that you add may point at either IP address(es) or an application. When you point an `A` record at an application, it is referred to as an alias, since the IP is determined on the Edge Node. To add a DNS record, click on the DNS tab on the Edge Node's detail page. The DNS changes apply to the Node as soon as you submit the form, so make sure you are satisified with your changes before submitting them.

If you need to edit or delete DNS data at a later time, you can do so from the UI and/or API.

Contributing

Contributing information is coming soon. For now, please reach out to our support email if you notice an issue that should be addressed in an upcoming release. If you are eager to try out development, here are some key tips to consider.

This project uses development languages such as:

- Go (1.12)
- Node (v10 LTS)

We welcome all types of contributions, such as improvements to the tooling or documentation.

See the Controller UI documentation for more information on the design, implementation, and features of the Controller Web UI.

If you are contributing to any Go components, we currently are pinned to Go 1.12. While you may be able to compile this source using other versions of Go, this is the recommended version for testing. You should download this repository into your GOPATH using `git pull` or `go get`, as certain tests depend on this. The Go authors report that in Go 1.13, the GOPATH will be deprecated; we will update this project as the alternatives become available. More about the progress on this can be found on the Go blog here.

Testing

Testing should be performed in a local development environment. You should not try to perform tests on a running instance of the Controller, as the tests may impact your environment.

To run unit and integration tests:

```
make test
```

To run only unit tests:

```
make test-unit
```

To run only integration tests:

```
make test-api
```

To perform static analysis:

```
make lint
```

Additional `make` commands can be checked in the `Makefile`. Contributions to the repository are run through continuous integration (CI) as well on each build using these same commands.