WebPA (Xmidt) Reference Setup

- Overview
- Setting up a single node WebPA cluster
    - System Requirement
    - Software Dependencies
        - Supervisor (Not Required for CentOS 7.x releases)
        - About Supervisor
            - How to install
        - ZooKeeper (Not required in CentOS 7.x releases)
            - About ZooKeeper
            - How to install
            - Adding a startup script
            - Enable the service at bootup
    - Process flow diagram
    - WebPA Server components setup
        - Using the pre-built packages from GitHub
        - Building from the Source
            - Dependencies of Build system
                - Install golang
                - install glide
            - Downloading the source code
            - Building the components
        - Configuring the server components
            - Prerequisite
                - Generating a auth token
            - Talaria configuration
            - Scytale configuration
            - Tr1d1um configuration
        - Enable the services at boot-up
- WebPA Client Setup
    - Configuring Parodus
        - RDK Video Devices
        - RDK Broadband Devices
- Debugging & log files
    - Log files
        - server logs
        - Client logs
    - Common Errors
        - Service fails to start (Specific to older CentOS 6.x releases)
- Use cases
    - Downstream request
    - Upstream request
- Testing the connection
    - Using Postman GUI application
    - Using console command
    - List of connected Devices
    - Common TR181 parameters

Overview

This page presents an brief overview about webPA 2.0 (xmidt) components required for a reference webPA cluster setup and explains how to setup the cluster & establish an end-to-end connection with CPE devices.

WebPA is a secure web protocol messaging system for bi-directional communication between cloud server and RDK devices. It was built from the ground up specifically with security and performance as priorities. WebPA 2.0 commonly known as Xmidt (pronounced "transmit") is a combination of a server cluster and client that provide a highly available data path to devices deployed all over the world.

- Unlike TR-69, which polls wide-and-deep across a device landscape, on a less frequent basis, WebPA can precision-poll for the most useful data, much more quickly. That's mainly because it's lightweight, and because the load can be redistributed into all the apps needing to access the data.
- Specifically, WebPA is a lightweight connectivity socket, with lower-level protocol connectivity between devices, and the cloud — much like a websocket. It works by each client registering with the cloud, so that it maintains a socket connection to the cloud. On the cloud side, APIs enable polls through which other cloud components could receive events, or translate them in to a TR-181 device data model for further analysis.
- And, because WebPA is an "always on" connection (Websocket), asynchronous notifications for a change in device data are a straightforward exercise, where the application "listens" for web hook events.

Setting up a single node WebPA cluster

System Requirement

Operating system	Centos 7.6
Architecture	x86_64
Memory	> 2048 MB
Disk space	> 10 GB free space

It is good to disable the firewall (iptables) during initial setup for avoiding connection related confusions.

Commands described in this page are executed with super user (root) permission

Software Dependencies

Supervisor (Not Required for CentOS 7.x releases)

Only Applicable for Older CentOS 6.x releases

About Supervisor

Supervisor is a client/server system that allows its users to monitor and control a number of processes on UNIX-like operating systems.

Unlike other system initialization services, it is not meant to be run as a substitute for init. Instead it is meant to be used to control processes related to a project or a customer, and is meant to start like any other program at boot time.

How to install

```
a) Enable Extra Packages for Enterprise Linux (EPEL)
        $ wget http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
        $ rpm -Uvh epel-release-6-8.noarch.rpm
b) Install Python meld3
        $ yum install python-meld3
c) Install supervisor
        $ yum install supervisor
```

ZooKeeper (Not required in CentOS 7.x releases)

Only Applicable for Older CentOS 6.x releases

About ZooKeeper

ZooKeeper is a high-performance coordination service for distributed applications. It exposes common services - such as naming, configuration management, synchronization, and group services - in a simple interface so you don't have to write them from scratch. You can use it off-the-shelf to implement consensus, group management, leader election, and presence protocols. And you can build on it for your own, specific needs.

How to install

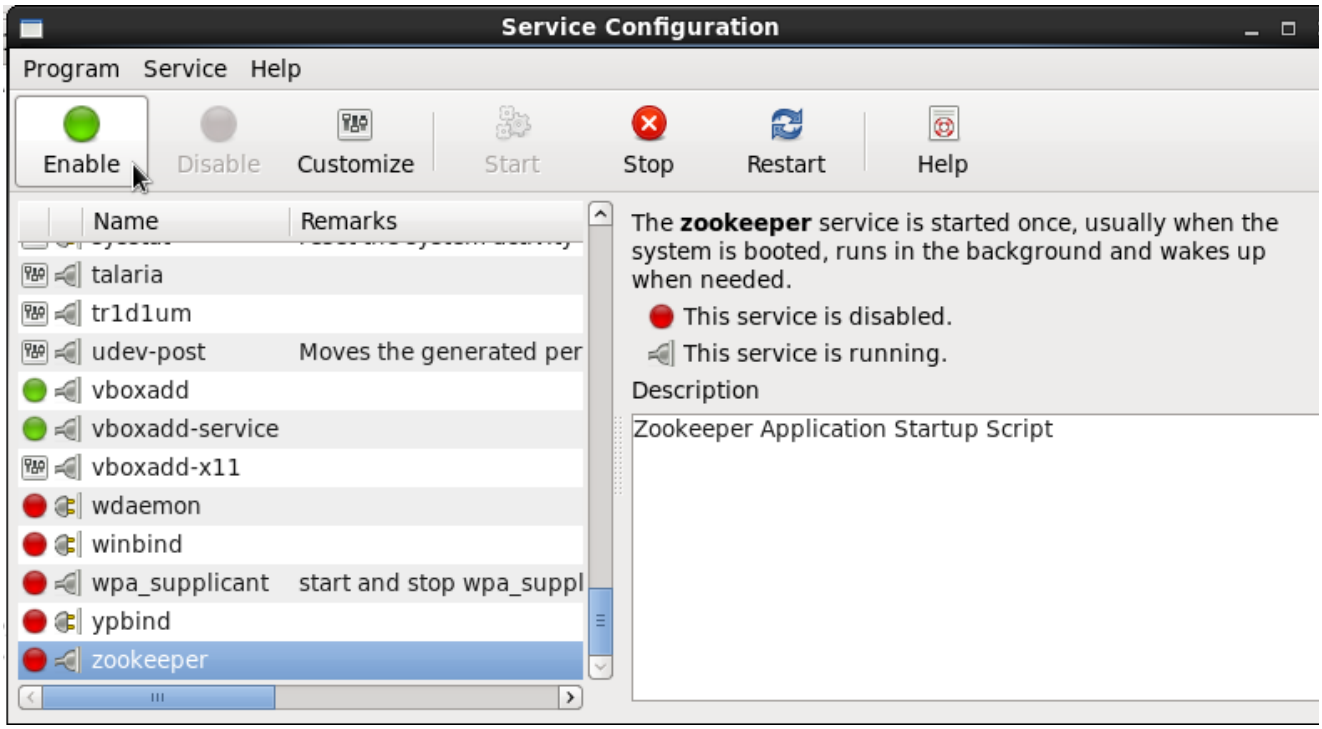
```
$ yum install zookeeper
```

Adding a startup script

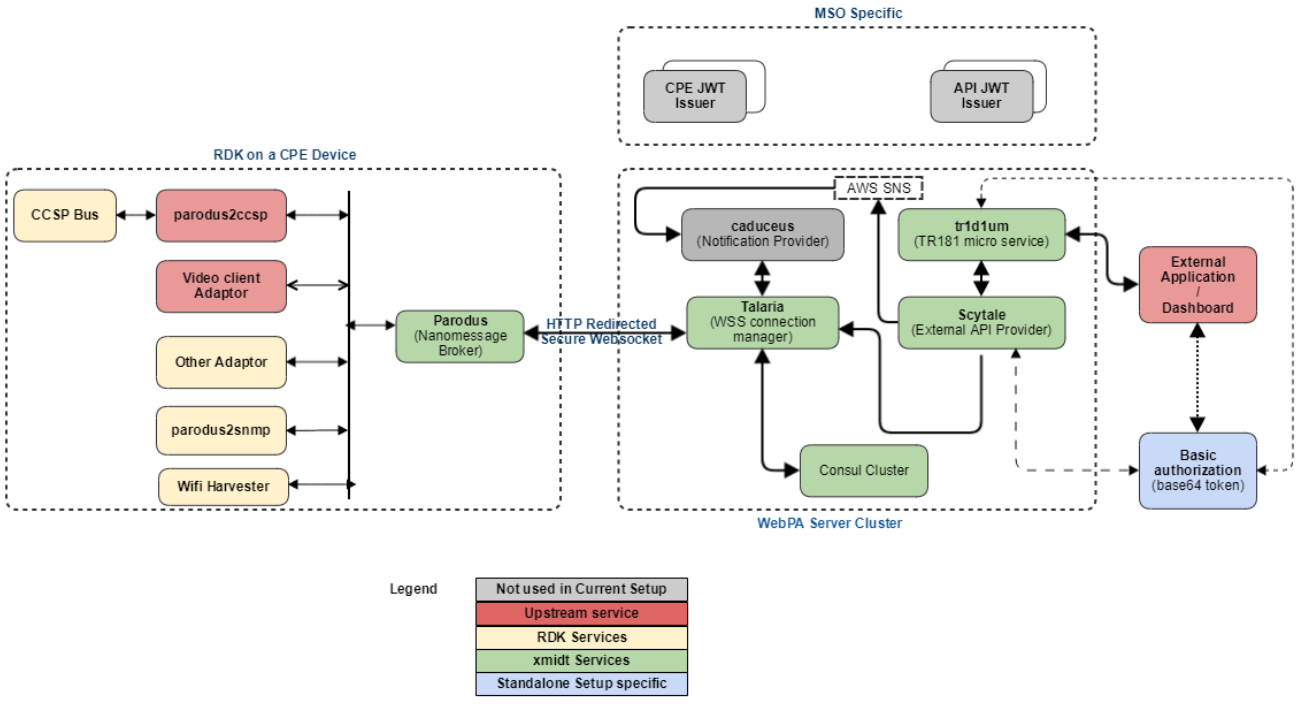
```
# cat /etc/init.d/zookeeper
#!bin/bash
#
# zookeeper Startup Script
#
# chkconfig: 345 90 14
# description: Zookeeper Application Startup Script

# Source function library
. /etc/rc.d/init.d/functions

#-------------------------------------------------------------------------------

start() {
        echo -n $"Starting Zookeeper: "
        /usr/lib/zookeeper/bin/zkServer.sh start
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && echo "[ OK ]"
}

stop() {
        echo -n $"Stopping Zookeeper: "
        /usr/lib/zookeeper/bin/zkServer.sh stop
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && echo "[ OK ]"
}

restart() {
        stop
        start
}

case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  restart|force-reload|reload)
        restart
        ;;
  status)
        /usr/lib/zookeeper/bin/zkServer.sh status
        RETVAL=$?
        ;;
  *)
        echo $"Usage: $0 {start|stop|status|restart|reload|force-reload}"
        exit 1
esac

exit $RETVAL
```

Enable the service at bootup

Launch system-config-services from a console and enable the zookeeper service from the services list.



Process flow diagram



WebPA Server components setup

Below is the list of components needed for a Xmidt (webPA 2.0) cluster setup. For a single node reference setup, few of the services are not mandatory hence not used.

Component	Type	Description	Used in current setup

Talaria	Server	Talaria maintains the secure websocket connections from the device and passes the messages from or to the device.	Yes
Scytale	Server	Scytale accepts the inbound requests and delivers the messages to the Talaria machines that could be hosting the device connection.	Yes
tr1d1um	Server	The Webpa micro-service that encode TR-181 requests.	Yes
petasos	Server	Petasos helps reduce the load on the Talaria machines by calculating which specific Talaria a device should connect to & redirecting the incoming request.	No
caduceus	Server	Caduceus provides the pub-sub message delivery (notification) mechanism for xmidt.	No
parodus	Client	Parodus is the light weight client that reaches out to the xmidt cloud to establish the connection from CPE devices.	Yes

Using the pre-built packages from GitHub

```
a) Import the GPG Key (Required once, common for all the packages)
        $ rpm --import https://github.com/xmidt-org/talaria/releases/download/v0.1.3/RPM-GPG-KEY-xmidt
b) Install the packages
        $ yum install https://github.com/xmidt-org/talaria/releases/download/v0.1.3/talaria-0.1.3-1.el7.x86_64.
rpm
        $ yum install https://github.com/xmidt-org/scytale/releases/download/v0.1.4/scytale-0.1.4-1.el7.x86_64.
rpm
        $ yum install https://github.com/xmidt-org/tr1d1um/releases/download/v0.1.2/tr1d1um-0.1.2-1.el7.x86_64.
rpm
Note: Change version number for downloading the required package.
```

Building from the Source

If pre-built RPM packages are already installed as explained in previous section & we want to use the same, Skip to configuration section

Dependencies of Build system

Install golang

Required for compiling server components written in go language. (Required version >=1.11

```
Preferred Method:
$ sudo yum install golang

[OR]
For Manual Installation:
$ wget https://dl.google.com/go/go1.11.linux-amd64.tar.gz
$ tar xzf go1.11.linux-amd64.tar.gz
$ sudo mv go /usr/local
** Add Below Lines to the profile file (.bash_profile etc.)
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$GOPATH/bin:$GOROOT/bin:$PATH

Verify the version with below:
$ go version
go version go1.11 linux/amd64
```

install glide

Glide is a package manager for Go that is conceptually similar to package managers for other languages. Glide provides the following functionality:

- Records dependency information in a `glide.yaml` file. This includes a name, version or version range, version control information for private repo or when the type cannot be detected, and more.
- Tracks the specific revision each package is locked to in a `glide.lock` file. This enables recursively fetching the dependency tree.
- Utilizes `vendor/` directories, known as the Vendor Experiment, so that different projects can have differing versions of the same dependencies.

```
$ wget -c https://github.com/Masterminds/glide/releases/download/v0.13.1/glide-v0.13.1-linux-amd64.tar.gz
$ tar -xzf glide-v0.13.1-linux-amd64.tar.gz -C /opt
$ echo "export PATH=$PATH:/opt/linux-amd64/" >> $HOME/.bash_profile
```

Downloading the source code

```
1. create a directory in $HOME say webpa_modules
$ mkdir $HOME/webpa_modules && cd $HOME/webpa_modules

2. Checkout the components from GitHub repository.
$ git clone https://github.com/Comcast/talaria.git
$ git clone https://github.com/Comcast/scytale.git
$ git clone https://github.com/Comcast/tr1d1um.git
```

Building the components

```
1. cd $HOME/webpa_modules/<component-name>
  e.g. cd webpa_modules/talaria
3. Build the component from source
        $ make build

4. Create the package
        $ mkdir $HOME/rpmbuild
        $ ./build_rpm.sh --no-sign

5. Install the locally built webPA component package
e.g. $ cd /root/rpmbuild/RPMS/x86_64/
     $ rpm -Uvh petasos-0.1.1-87.el6.x86_64.rpm
```


If running build_rpm.sh complains about following:

error: Macro %_releaseno has empty body
error: Macro %_releaseno has empty body

Then, modify the following in script to change build number to appropriate value

- to get the latest build number:
  $ git tag -l|sort -V|grep -v alpha (select the latest version from list)
  e.g. BUILD_NUMBER=87

If build_rpm.sh prompts for password, modify the rpmbuild command to disable the signing option

yes "" | rpmbuild -ba \

--define "_signature gpg" \

--define "_ver $release" \

--define "_releaseno ${BUILD_NUMBER}" \

--define "_fullver $new_release" \

${NAME}.spec


```
If the script terminates with "error: Bad owner/group: /root/webpa_modules/petasos/petasos.spec"

change the ownership to match current user name

$ chown root.root petasos.spec
```

Configuring the server components

Prerequisite

Generating a auth token

WebPA server components as well as requesting application has to use a authorization token for bearer authentication. We can either use a basic authorization token or make use of a key server for obtaining a bearer token.

For example, a UI application needs to invoke some Preference setting or to obtain some diagnostics information on behalf of a MSO partner, deviceId, serviceAccountId or combination of the three. It will first obtain or use a predefined AUTH token, set it as a HTTP header and then invoke the GET/SET operation.

In a production environment, webPA server components & requesting applications use SAT as a bearer token for AUTHZ and AUTHN. SAT stands for Service Access Token. As the name implies, it is used by the calling applications to request access to CPE API's. From a implementation point of view, A SAT is a Json Web Token which if shortened to "jwt". It is a base64 encoded strings of pre-defined bytes with 3 distinct parts separated by a period.

However in the standalone setup, we have used basic base64 encoded authorization token because SAT requires access to operator specific key servers. This auth token will be used when configuring different webPA components as well while performing GET/SET requests to the CPE from a 3rd party application.

We can use either of the below 2 methods to generate a basic authorization string.

```
Note: For newer releases the basic auth token should be in username:password format.
1. Using openssl command to generate the base64 encoded token.
[root@webpa-node1 ~]# openssl enc -base64 <<< "user123:webpa@1234567890"
[OUTPUT] : dXNlcjEyMzp3ZWJwYUAxMjM0NTY3ODkwCg==
2. Using Linux coreutils tools to generate the base64 encoded token
[root@webpa-node1 ~]# echo "user123:webpa@1234567890"|base64
[OUTPUT] : dXNlcjEyMzp3ZWJwYUAxMjM0NTY3ODkwCg==
```

Talaria configuration

Edit the configuration file & modify port values if you need to run talaria service in a different port (default value is 8080).

Sample configuration file [/etc/talaria/talaria.yaml]

```
---
########################################
#   Labeling/Tracing via HTTP Headers Configuration
########################################

# The unique fully-qualified-domain-name of the server.  It is provided to
# the X-Talaria-Server header for showing what server fulfilled the request
# sent.
# (Optional)
  fqdn: <Fully Qualified Domain Name / IP>
  server: "xxx.xxx.xxx.xxx"
  env: test
  scheme: http

# Provides this build number to the X-Tr1d1um-Build header for
# showing machine version information.  The build number SHOULD
# match the scheme `version-build` but there is not a strict requirement.
# (Optional)
  build: "0.1.3-1"

# Provides the region information to the X-Tr1d1um-Region header
# for showing what region this machine is located in.  The region
# is arbitrary and optional.
# (Optional)
  region: "east"

# Provides the flavor information to the X-Tr1d1um-Flavor header
# for showing what flavor this machine is associated with.  The flavor
# is arbitrary and optional.
# (Optional)
  flavor: "mint"
```

```
  primary:
    address: ":8080"
  health:
    address: ":8180"
  pprof:
    address: ":8280"
  control:
    address: ":8203"
  metric:
    address: ":8380"
    metricsOptions:
      namespace: "xmidt"
      subsystem: "talaria"

########################################
#   Service Discovery Configuration
########################################

  # service defines the parameters needed to interact with the consul cluster
  # for service discovery.  Presently only consul is supported.  This is
  # presently only used by Prometheus to discover machines to monitor, but
  # in the not-too-distant future talaria will use this interaction to load
  # balance across all caduceus machines instead of using DNS.
  # (Optional)
  service:
    # consul configures the consul library in caduceus to use the local
    # service discovery agent
    consul:
      # client defines how to connect to the local consul agent (on the same
      # VM/container)
      client:
        # address is the address of the local consul agent
        address: "127.0.0.1:8500"
        # scheme is how the consul library should interact with the local
        # consul agent
        scheme: "http"
        # waitTime is TBD
        waitTime: "30s"

      # disableGenerateID is TBD
      disableGenerateID: true

      # registrations defines what services caduceus should register with
      # consul
      #
      #     id      - the VM/container instance name registered with consul
      #     name    - the name of service being registered
      #     tags    - a list of tags to associate with this registration
      #     address - the mechanism to reach the service (generally unique fqdn)
      #     port    - the port to reach the service at
      #     checks  - the list of checks to perform to determine if the service
      #               is available/healthy
      #         checkID                       - TBD
      #         ttl                           - how long the check is valid for
      #         deregisterCriticalServiceAfter - the duration to wait before the
      #                                          service is removed due to check
      #                                          failures
      registrations:
        -
          id: "example_talaria.xmidt.net"
          name: "talaria"
          tags:
            - "prod"
            - "mint"
            - "stage=prod"
            - "flavor=mint"
          address: <WEBPA_SERVER_IP>
          port: 6001
          checks:
            -
              checkID: "example_talaria.xmidt.net:ttl"
```

```
            ttl: "30s"
            deregisterCriticalServiceAfter: "70s"

log:
  file: "/var/log/talaria/talaria.log"
  level: "DEBUG"
  json: false

device:
  manager:
    upgrader:
      handshakeTimeout: "10s"
    initialCapacity: 100000
    maxDevices: 100
    deviceMessageQueueSize: 100
    pingPeriod: "45s"
    idlePeriod: "135s"
    requestTimeout: "15s"
  outbound:
    method: "POST"
    eventEndpoints:
      default: http://127.0.0.1:6300/api/v3/notify
    requestTimeout: "125s"
    defaultScheme: "http"
    allowedSchemes:
      - "http"
      - "https"
    outboundQueueSize: 1000
    workerPoolSize: 100
    transport:
      maxIdleConns: 0
      maxIdleConnsPerHost: 100
      idleConnTimeout: "120s"
    clientTimeout: "160s"
    authKey: <xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
inbound:
  authKey: <xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>

eventMap:
  default: http://127.0.0.1:6300/api/v3/notify

service:
  defaultScheme: http
  fixed:
    - http://127.0.0.1:8080
```

Scytale configuration

Edit the configuration file under /etc/scytale and modify following values

"fqdn" : Fully qualified domain name of the server
"server" : Listening IP address (using "localhost" will allow connections only from the current machine.)

"endpoints" : Under "fanout" section, change the IP / Port value to match to the one where **Talaria** service is listening.

"authHeader" : Auth token Use the auth token which was generated in previous section

"file" : Under "log" section, change the value from "stdout" to a file name if we need to redirect debug messages to a separate log file.

Add the **"aws"** section with following values for suppressing few error messages

```
"aws": {
      "accessKey": "fake",
       "secretKey": "fake",
       "env": "fake",
       "sns": {
          "region": "us-east-1",
          "topicArn": "arn:aws:sns:us-east-1:999999999999:fake",
          "urlPath" : "/api/v2/aws/sns"
       }
 },
```

This will set AWS & SNS parameters with fake ones since we don't use actual keys and SNS (amazon simple notification service) in the current setup.

Sample configuration [/etc/scytale/scytale.yaml]

```
---

########################################
#   Labeling/Tracing via HTTP Headers Configuration
########################################

# The unique fully-qualified-domain-name of the server.  It is provided to
# the X-Scytale-Server header for showing what server fulfilled the request
# sent.
# (Optional)
server: "xxx.xxx.xxx.xxx"

# Provides this build number to the X-Tr1d1um-Build header for
# showing machine version information.  The build number SHOULD
# match the scheme `version-build` but there is not a strict requirement.
# (Optional)
build: "0.1.4-1"

# Provides the region information to the X-Tr1d1um-Region header
# for showing what region this machine is located in.  The region
# is arbitrary and optional.
# (Optional)
region: "east"

# Provides the flavor information to the X-Tr1d1um-Flavor header
# for showing what flavor this machine is associated with.  The flavor
# is arbitrary and optional.
# (Optional)
flavor: "mint"


################################################################################
# WebPA Service configuration
################################################################################

# For a complete view of the service config structure,
# checkout https://godoc.org/github.com/Comcast/webpa-common/server#WebPA

########################################
#   Primary Endpoint Configuration
########################################

# primary provides the configuration for the main server for this application
primary:
  address: ":7000"

########################################
#   Health Endpoint Configuration
########################################

# health defines the details needed for the health check endpoint.  The
# health check endpoint is generally used by services (like AWS Route53
# or consul) to determine if this particular machine is healthy or not.
health:
  address: ":7001"
```

```
########################################
#   Debugging/Pprof Configuration
########################################

# pprof defines the details needed for the pprof debug endpoint.
# (Optional)
pprof:
  address: ":7002"

########################################
#   Metrics Configuration
########################################

# metric defines the details needed for the prometheus metrics endpoint
# (Optional)
metric:
  address: ":7082"
  metricsOptions:
    # namespace is the namespace of the metrics provided
    # (Optional)
    namespace: "webpa"

    # subsystem is the subsystem of the metrics provided
    # (Optional)
    subsystem: "scytale"

fanout:
  fanoutTimeout: "125s"
  clientTimeout: "45s"
  endpoints:
      - "http://localhost:8080/api/v2/device"
  authorization: <xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>

########################################
#   Logging Related Configuration
########################################

# log configures the logging subsystem details
log:
  # file is the name of the most recent log file.  If set to "stdout" this
  # will log to os.Stdout.
  # (Optional) defaults to os.TempDir()
  file: "/var/log/scytale/scytale.log"

  # level is the logging level to use - INFO, DEBUG, WARN, ERROR
  # (Optional) defaults to ERROR
  level: "DEBUG"

  # maxsize is the maximum log file size in MB
  # (Optional) defaults to max 100MB
  maxsize: 50

  # maxage is the maximum number of days to retain old log files
  # (Optional) defaults to ignore age limit (0)
  maxage: 30

  # maxbackups is the maximum number of old log files to retain
  # (Optional) defaults to retain all (0)
  maxbackups: 10

  # json is a flag indicating whether JSON logging output should be used.
  # (Optional) defaults to false
  json: true
aws:
  accessKey: "fake-accessKey"
  secretKey: "fake-secretKey"
  env: "fake-env"
  sns:
    awsEndpoint: http://goaws:4100
    region: "ap-east-1"
```

```
    topicArn: "arn:aws:sns:ap-east-1:999999999991:fake-env"
    urlPath: "/api/v2/aws/sns"
waitForDns: 0
authHeader: ["xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"]
start:
  duration: 1
  apiPath: http://127.0.0.1:6300/hooks
  authHeader: <xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
```

Tr1d1um configuration

Edit the configuration file from /etc/tr1d1um to set following parameters

"fqdn" : Fully qualified domain name of server

"server" : IP Address to which the service has to listen

"version" : Current version of the service

"region" : Region of deployment

"flavor" : Development, Production etc.

"address" : Under "primary" section, change the value to point to the port where tr1d1um service will listen for incoming requests.

"targetURL" : Change to IP-Address:Port value where **SCYTALE** service is running.

"authHeader" : Auth token Use the auth token which was generated in previous section.

"aws" : Add fake values as described previously.

Sample configuration file [/etc/tr1d1um/tr1d1um.yaml]

```
---

########################################
#   Labeling/Tracing via HTTP Headers Configuration
########################################

# The unique fully-qualified-domain-name of the server.  It is provided to
# the X-Tr1d1um-Server header for showing what server fulfilled the request
# sent.
# (Optional)
server: "xxx.xxx.xxx.xxx"

# Provides this build number to the X-Tr1d1um-Build header for
# showing machine version information.  The build number SHOULD
# match the scheme `version-build` but there is not a strict requirement.
# (Optional)
build: "0.1.2-1"

# Provides the region information to the X-Tr1d1um-Region header
# for showing what region this machine is located in.  The region
# is arbitrary and optional.
# (Optional)
region: "east"

# Provides the flavor information to the X-Tr1d1um-Flavor header
# for showing what flavor this machine is associated with.  The flavor
# is arbitrary and optional.
# (Optional)
flavor: "mint"


################################################################################
# WebPA Service configuration
################################################################################

# For a complete view of the service config structure,
# checkout https://godoc.org/github.com/Comcast/webpa-common/server#WebPA
```

```
########################################
#   Primary Endpoint Configuration
########################################

# primary provides the configuration for the main server for this application
primary:
  address: ":9003"

########################################
#   Health Endpoint Configuration
########################################

# health defines the details needed for the health check endpoint.  The
# health check endpoint is generally used by services (like AWS Route53
# or consul) to determine if this particular machine is healthy or not.
health:
  address: ":9004"

########################################
#   Debugging/Pprof Configuration
########################################

# pprof defines the details needed for the pprof debug endpoint.
# (Optional)
pprof:
  address: ":9005"

########################################
#   Metrics Configuration
########################################

# metric defines the details needed for the prometheus metrics endpoint
# (Optional)
metric:
  address: ":9082"
  metricsOptions:
    # namespace is the namespace of the metrics provided
    # (Optional)
    namespace: "webpa"

    # subsystem is the subsystem of the metrics provided
    # (Optional)
    subsystem: "tr1d1um"

########################################
#   Logging Related Configuration
########################################

# log configures the logging subsystem details
log:
  # file is the name of the most recent log file.  If set to "stdout" this
  # will log to os.Stdout.
  # (Optional) defaults to os.TempDir()
  file: "/var/log/tr1d1um/tr1d1um.log"

  # level is the logging level to use - INFO, DEBUG, WARN, ERROR
  # (Optional) defaults to ERROR
  level: "DEBUG"

  # maxsize is the maximum log file size in MB
  # (Optional) defaults to max 100MB
  maxsize: 50

  # maxage is the maximum number of days to retain old log files
  # (Optional) defaults to ignore age limit (0)
  maxage: 30

  # maxbackups is the maximum number of old log files to retain
  # (Optional) defaults to retain all (0)
  maxbackups: 10
```

```
  # json is a flag indicating whether JSON logging output should be used.
  # (Optional) defaults to false
  json: true


################################################################################
# Webhooks Related configuration
################################################################################

# webhooksEnabled indicates whether or not the webhooks server should be started
# It is disabled for local testing
webhooksEnabled: false

# The unique fully-qualified-domain-name of the server.  The webhooks library uses it
# to know which host to use to confirm this service is ready to receive events
# (Optional if not running webhooks)
fqdn: "tr1d1um-local-instance-123.example.com"

# start contains configuration for the logic by which Tr1d1um can
# fetch the current WebPA webhooks without having to wait for SNS
# It does so by pinging the rest of the cluter at the specified apiPath
# More detaisl at https://godoc.org/github.com/Comcast/webpa-common/webhook#StartConfig
start:
  # duration is the max amount of time allowed to wait for webhooks data to be retrieved
  duration: "20s"

  # path used to query the existing webhooks
  apiPath: http://localhost:6100/hooks

########################################
# Webhooks DNS readiness Configuration
########################################

# WaitForDns is the duration the webhooks library will wait for this server's DNS record to be
# propagated. This waiting logic is important so AWS SNS webhook confirmations are not missed
waitForDns: "30s"

#soa stands for Start of Authority and it's a type of record in a DNS
soa:
  # provider is the SOA provider used to verify DNS record readiness of this service
  provider: "example-123.awsdns-00.com:17"

########################################
# Webhooks AWS SNS Configuration
########################################

# aws provides the AWS SNS configurations the webhooks library needs
aws:
  #AWS access key
  accessKey: "fake-accessKey"

  #AWS secret key
  secretKey: "fake-secretKey"

  env: local-dev

  sns:
    # awsEndpoint is the AWS endpoint
    # this must be left out in produ
    awsEndpoint: http://goaws:4100

    #region is the AWS SNS region
    region: "us-east-1"

    # topicArn describes the SNS topic this server needs to subscribe to
    topicArn: arn:aws:sns:us-east-1:000000000000:xmidt-local-caduceus

    #urlPath is the URL path SNS will use to confirm a subscription with this server
    urlPath: "/api/v2/aws/sns"
```

```
################################################################################
# Testing Authorization Credentials
################################################################################

# authHeader is a list of Basic Auth credentials intended to be used for local testing purposes
# WARNING! Be sure to remove this from your production config
authHeader: "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"


################################################################################
# WRP and XMiDT Cloud configurations
################################################################################

# targetURL is the base URL of the XMiDT cluster
targetURL: http://localhost:7000

# WRPSource is used as 'source' field for all outgoing WRP Messages
WRPSource: "dns:tr1d1um.xmidt.comcast.net"

# supportedServices is a list of endpoints we support for the WRP producing endpoints
# we will soon drop this configuration
supportedServices:
  - "config"


################################################################################
# HTTP Transaction Configurations
################################################################################

# clientTimeout is the timeout for the HTTP clients used to contact the XMiDT cloud
clientTimeout: "135s"

# respWaitTimeout is the max time Tr1d1um will wait for responses from the XMiDT cloud
respWaitTimeout: "129s"

# netDialerTimeout is the timeout used for the net dialer used within HTTP clients
netDialerTimeout: "5s"

# requestRetryInterval is the time between HTTP request retries against XMiDT
requestRetryInterval: "2s"

# requestMaxRetries is the max number of times an HTTP request is retried against XMiDT in
# case of ephemeral errors
requestMaxRetries: 2
```

Enable the services at boot-up

CentOS 7 uses a systemd based boot-up mechanism. Hence below commands will enable the required services.

```
$ sudo systemctl enable talaria
$ sudo systemctl enable scytale
$ sudo systemctl enable tr1d1um
```

WebPA Client Setup

Parodus is the client-end service running on the RDK-V CPE devices which establishes a connection with webPA service on device boot-up and delivers request-response between the webPA server & CPE device services. Parodus provides following functionalities in a CPE device.

Websocket client: Nopoll library used as Websocket Client. It allows building pure WebSocket solutions or to provide WebSocket support to existing TCP oriented applications. Nopoll handles all the messages coming from or to the server asynchronously.

Nanomsg Server: Parodus acts as Nanomsg server to distribute messages upstream and downstream.

Configuring Parodus

RDK Video Devices

Edit parodus startup script for enabling the CPE device to use local webPA server

--webpa-url : Set with IP Address and Port of talaria service

--force-ipv4 : Force use of IPv4 for communication.

```
vi /lib/rdk/startParodus.sh
/bin/systemctl set-environment PARODUS_CMD=" --hw-mac=$HwMac --webpa-ping-time=$PingWaitTime --webpa-interface-used=$NwInterface --webpa-url=http://<WEBPA_SERVER_IP> --partner-id=comcast --webpa-backoff-max=9 --force-ipv4 --ssl-cert-path=$SSL_CERT_FILE"
```

Restart the service after changes are done.

```
# systemctl restart parodus
```

⚠ Remove the jwt related parameters from PARODUS_CMD e.g. (-acquire-jwt, --dns-txt-url, --jwt-public-key-file, --jwt-algo etc.)

RDK Broadband Devices

Edit parodus startup script for enabling the CPE device to use local webPA server

ServerURL : Set to IP Address and Port of talaria service.

--force-ipv4 : Force use of IPv4 for communication.

```
vi /lib/rdk/parodus_start.sh
ServerURL=http://<webpa-ip>:8080
   command="/usr/bin/parodus --hw-model=$ModelName --hw-serial-number=$SerialNumber --hw-manufacturer=$Manufacturer --hw-last-reboot-reason=$LastRebootReason --fw-name=$FirmwareName --boot-time=$BootTime --hw-mac=$HW_MAC --webpa-ping-time=180 --webpa-interface-used=erouter0 --webpa-url=$ServerURL --webpa-backoff-max=$BackOffMax  --parodus-local-url=$PARODUS_URL --partner-id= --ssl-cert-path=$SSL_CERT_PATH --force-ipv4 "
```

start the service after changes are done.

```
#  rm –rf /tmp/parodusCmd.cmd
# systemctl restart parodus
```

Debugging & log files

Log files

server logs

- WebPA server logs are distributed among following locations.
  - /var/log/<webpa-service>/ : keeps debug log files.
    - supervisord.log : Log messages related to service boot-up & initialization
    - console.out : console logs (debug message will appear here if **"file": "stdout"** is configured in <webpa-service>.json file
  - /var/run/<webpa-service>/ : keeps service specific debug messages
    - <service-name>Log.log : component specific debug messages will appear here if **"file" : "fileName.log"** is configured in <webpa-service>.json file

Client logs

Parodus service log file is located as /opt/logs/parodus.log, provides debug information such as connection details, service initialization, which protocols are enabled/disabled etc.

Common Errors

Service fails to start (Specific to older CentOS 6.x releases)

When we see an error similar to the below, it is related to zookeeper service failed to load or not running currently. restarting zookeeper & subsequent restart of other services solves the issue.

```
$ cat talariaLog.log
ts=2018-01-16T13:14:55.143587713Z caller=talaria.go:133 level=error msg="Unable to obtain service discovery
instancer" error="zk: could not connect to a server"
```

Use cases

Downstream request

[External Application --> CPE Device] (e.g. Query from a dashboard application to request for how long has a specific router been up):

i. External service request goes to tr1d1um service
ii. tr1d1um forwards the request to Scytale API server cluster
iii. Scytale sends request to talaria server cluster
iv. Talaria then sends request to appropriate WebPA device

Upstream request

[CPE Device --> external service] (e.g. WebPA client notifications to external services, for instance, at boot time):

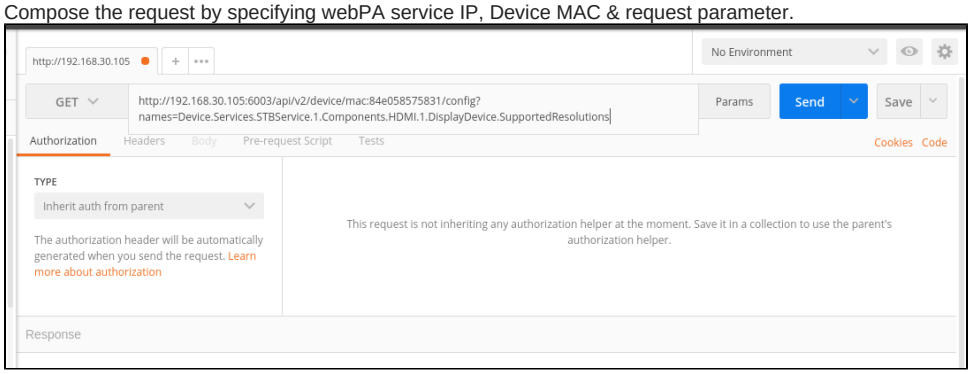
i. Request is sent through the same web socket connection that is established to Talaria
ii. Talaria sends request upstream to Scytale
iii. Scytale sends request to external service.

Testing the connection

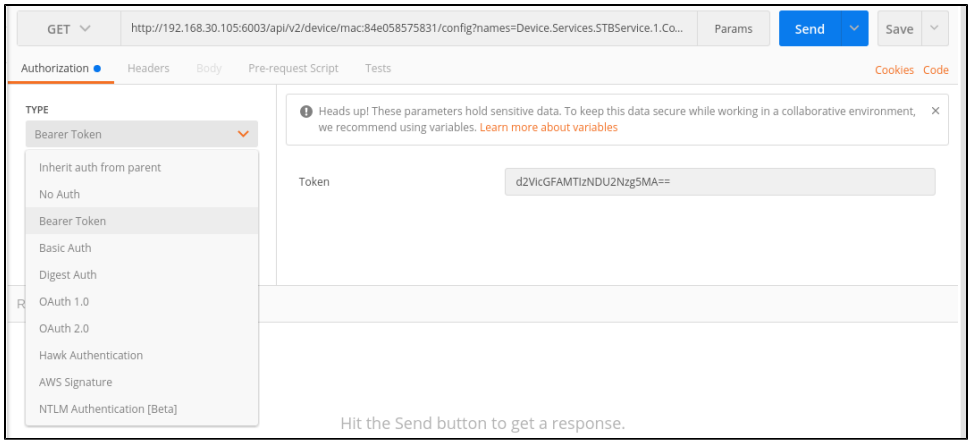
Using Postman GUI application

Postman is a powerful HTTP client for testing web services.Get Postman from here

1. Launch Postman and create a GET request
   Compose the request by specifying webPA service IP, Device MAC & request parameter.



2. Add an authorization token (This token should match with the one configured with webPA server-end components.
   Click on Authorization tab and select bearer token as Type



3. Send the API request to webPA server & check the result

Using console command

AUTH_TOKEN : Basic base64 encoded auth token or SAT (if enabled).

WEBPA-URL : URL of Tr1d1um service in IP:PORT format.

DEVICE_MAC : MAC address of the CPE device.

PARAMETER : GET/SET Parameter that need to be requested.

GET Parameter

```
$ curl -H ''Authorization:Basic <AUTH_TOKEN>' -i http://<WEBPA-URL>/api/v2/device/mac:<DEVICE_MAC>/config?
names=<PARAMETER>
e.g.
$ curl -H 'Authorization:Basic d2VicGFAMTIzNDU2Nzg5MA==' -i http://<WEBPA_SERVER_IP:PORT>/api/v2/device/mac:
84e058575831/config?names=Device.DeviceInfo.ModelName
```

SET Parameter

```
$ curl -X PATCH http://<IP>:9003/api/v2/device/mac:<MAC>/config -d '{"parameters": [ {"dataType": 0, "name":
"<TR181_PARAM>", "value": "<Value-to-Set>"}]}' -H 'Authorization:Basic <TOKEN>' e.g.
$ curl -X PATCH http://35.155.171.121:9003/api/v2/device/mac:b827eb5681cd/config -d '{"parameters": [
{"dataType": 0, "name": "Device.WiFi.SSID.10001.SSID", "value": "Testing"}]}' -H 'Authorization:Basic
d2VicGFAMTIzNDU2Nzg5MAo='
```

List of connected Devices

```
$ curl -H "<AUTH_TOKEN>" http://<IP>:8080/api/v2/devices
e.g. curl -H "Authorization: Basic d2VicGFAMTIzNDU2Nzg5MA==" http://<webpa_serverURL>:8080/api/v2/devices
```

Common TR181 parameters

Device.DeviceInfo.Manufacturer
Device.DeviceInfo.ManufacturerOUI
Device.DeviceInfo.ModelName
Device.DeviceInfo.SerialNumber
Device.DeviceInfo.HardwareVersion
Device.DeviceInfo.SoftwareVersion
Device.DeviceInfo.UpTime
Device.DeviceInfo.ProcessorNumberOfEntries
Device.DeviceInfo.MemoryStatus.Total
Device.DeviceInfo.MemoryStatus.Free
Device.DeviceInfo.ProcessStatus.CPUUsage
Device.DeviceInfo.ProcessStatus.ProcessNumberOfEntries