Analytics and Location Engine 2.0

User & API Guide

aruba NETWORKS an HP company

Copyright Information

Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett-Packard Company. To obtain such source code, send a check or money order in the amount of US $10.00 to:

Hewlett-Packard Company

Attn: General Counsel

3000 Hanover Street

Palo Alto, CA 94304

USA

Please specify the product and version for which you are requesting source code. You may also request a copy of this source code free of charge at dl-gplquery@arubanetworks.com.

The wireless network has a wealth of information about unassociated and associated devices. The Analytics and Location Engine (ALE) is designed to gather information from the network, process it, and share it through a simple and standard API. ALE includes a location engine that calculates associated and unassociated device location periodically.

ALE uses context streams from WLAN controllers or Instant clusters, including RSSI readings, to calculate device location.

For every device on the network, ALE provides the following information through the Northbound API:

- Client Username,
- IP address
- MAC address
- Device type
- Application firewall data, showing the destinations and applications used by associated devices.
- Current location
- Historical locations

Personal identifying information (PII) can be filtered out of the data, providing the option to view standard or anonymized data with or without MAC addresses, client user names, and IP addresses.

This guide describes ALE installation and configuration, polling APIs, publish and subscribe APIs, ALE features, security, and troubleshooting tips.

Related Documents

The following documents are part of the complete documentation set for the Analytics and Location Engine.

- *Analytics and Location Engine 2.0 API Guide*
- *Analytics and Location Engine 2.0.x Release Notes*
- *Aruba Instant 6.4.3.x-4.2 User Guide*
- *ArubaOS 6.4 Quick Start Guide*
- *ArubaOS 6.4.2.x or 6.4.3.x User Guide*
- *ArubaOS 6.4.2.x or 6.4.3.x Command-Line Reference Guide*
- *AirWave 8.0.8 User Guide*

Contacting Aruba Networks

Table 1: *Contact Information*

Website Support	
Main Site	http://www.arubanetworks.com
Support Site	https://support.arubanetworks.com
Airheads Social Forums and Knowledge Base	http://community.arubanetworks.com
North American Telephone	1-800-943-4526 (Toll Free) 1-408-754-1200
International Telephone	http://www.arubanetworks.com/support-services/aruba-support-program/contact-support/
Support Email Addresses	
Americas and APAC	support@arubanetworks.com
EMEA	emea_support@arubanetworks.com
Wireless Security Incident Response Team (WSIRT) .	wsirt@arubanetworks.com

Conventions

The following conventions are used throughout this manual to emphasize important concepts:

Table 2: *Typographical Conventions*

Type Style	Description
Italics	This style is used to emphasize important terms and to mark the titles of books.
`System items`	This fixed-width font depicts the following: • Sample screen output • System prompts • Filenames, software devices, and specific commands when mentioned in the text
`Commands`	In the command examples, this bold font depicts text that you must type exactly as shown.
<Arguments>	In the command examples, italicized text within angle brackets represents items that you should replace with information appropriate to your specific situation. For example: # **`send`** *<text message>* In this example, you would type "send" at the system prompt exactly as shown, followed by the text of the message you wish to send. Do not type the angle brackets.
`[Optional]`	Command examples enclosed in brackets are optional. Do not type the brackets.
`{Item A | Item B}`	In the command examples, items within curled braces and separated by a vertical bar represent the available choices. Enter only one choice. Do not type the braces or bars.

The following informational icons are used throughout this guide:

Indicates helpful suggestions, pertinent information, and important things to remember.

Indicates a risk of damage to your hardware or loss of data.

Indicates a risk of personal injury or death.

This chapter discusses the ALE architecture and includes the following information:

ALE Architecture

ALE gathers client information from the network, and processes and shares it through a standard API. Businesses can use this client information to analyze a client's Internet behavior, such as shopping preferences.

A fully deployed ALE consists of the following:

- VMWare server available for installing the Analytics and Location Engine (ALE)
- Access Points (AP)
- Controller ArubaOS (optional for IAPs) – 6.4.3
- AirWave (AW, optional) – 8.0.8
- Analytics and Location Engine (ALE) – 2.0
- Instant AP (IAP) – 4.2
- Aruba Nao Logger Android Application (only required for calibration mode)

ALE Scale and Performance Table

The ALE scale and performance table lists the number of APs/clients in a network and their corresponding scaling and performance metrics.

The following table provides suggested metrics. Depending on the scenario, these numbers may change. For example, if there are more clients, resulting in more location calculations, more CPU and memory may be required. Depending on the number of clients, additional memory may also be required.

The APs/Clients column includes both associated and unassociated clients.

Table 3: *ALE Scale and Performance*

Number of APs/Clients	CPU	RAM (default)	Hard Disk or Secondary Storage	Comments
Up to 1K/16K	8	16GB	120GB	Medium
Up to 2K/32K	16	32GB	160GB	Large

NOTE

ALE can support up to 100 controllers and import information from up to eight AirWave servers, with up to 1000 floors per AirWave server and 2000 floors total across all AirWave servers. When dealing with such a large scale, it is recommended that you read from the Visual RF backup file instead of importing online from AirWave.

NOTE

When RTLS is configured, the scale numbers can decrease due to the additional processing required to decode the higher rate of packets received by ALE. Under context mode (without maps or locations), a higher scale can be achieved because there are no location computations.

ALE Deployment Considerations

ALE Predeployment Checklist

The following items must be in place before ALE can be deployed:

- A WLAN controller or Instant cluster must be available for an ALE deployment. Under certain modes of operation, AirWave may be required (context with device location, estimation).
- Communication between ALE and the controller is not secure. When deploying ALE in an Aruba controller-based WLAN, the network must be trusted.

  The ports used for ALE-Controller communication are:

  - UDP 8211 for (AMON/PAPI)
  - 4343 for bootstrap
- Up to 100 controllers can terminate on a single ALE instance.
- Sufficient AP density and optional AM density are required for good location engine performance. See Access Point Placement and Density
- Anonymization is enabled by default. If an application requires non-anonymized data, anonymization must be explicitly enabled. See About Anonymization for more details.
- For higher location accuracy, the deployment must plan for fingerprinting or callibration. This creates a trade-off between ease-of-deployment and location accuracy.
- The chosen mode is an important consideration. If you are deploying ALE for a distributed IAP deployment with only one or two IAPs per location, use **Context Mode (station, application, proximity)** since there are not enough APs located within the site to locate devices on a map. For larger sites that require device location, there is a trade-off between ease-of-deployment and location accuracy. If your use-case requires BLUE-DOT navigation or tight-space notifications, consider using an Aruba BLE-based solution.

Access Point Placement and Density

The distance between deployed APs impacts location performance and voice and data application performance. Though AP spacing requirements are flexible, small or large distances should be avoided, as signal strength varies drastically at different distances.

Since location accuracy is dependent on AP placement and RF design, the following deployment guidelines are recommended to achieve the highest location accuracy in a WLAN:

- Voice Overlay ensures that every location on the map is covered by a minimum of three APs.
  - The AirWave Visual RF can verify this. Though Visual RF may not be 100 percent accurate, it provides a good approximation. In Visual RF, select Voice, and make sure every location is covered by three or more APs.

NOTE

When AirWave is used to validate Voice Overlay, only one frequency can be selected at a time. If both frequencies are enabled, AirWave doubles the AP count and does not provide a true representation of Voice Overlay in the deployment.

Figure 1 *Ensure Voice Overlay at frequency 5 GHz*

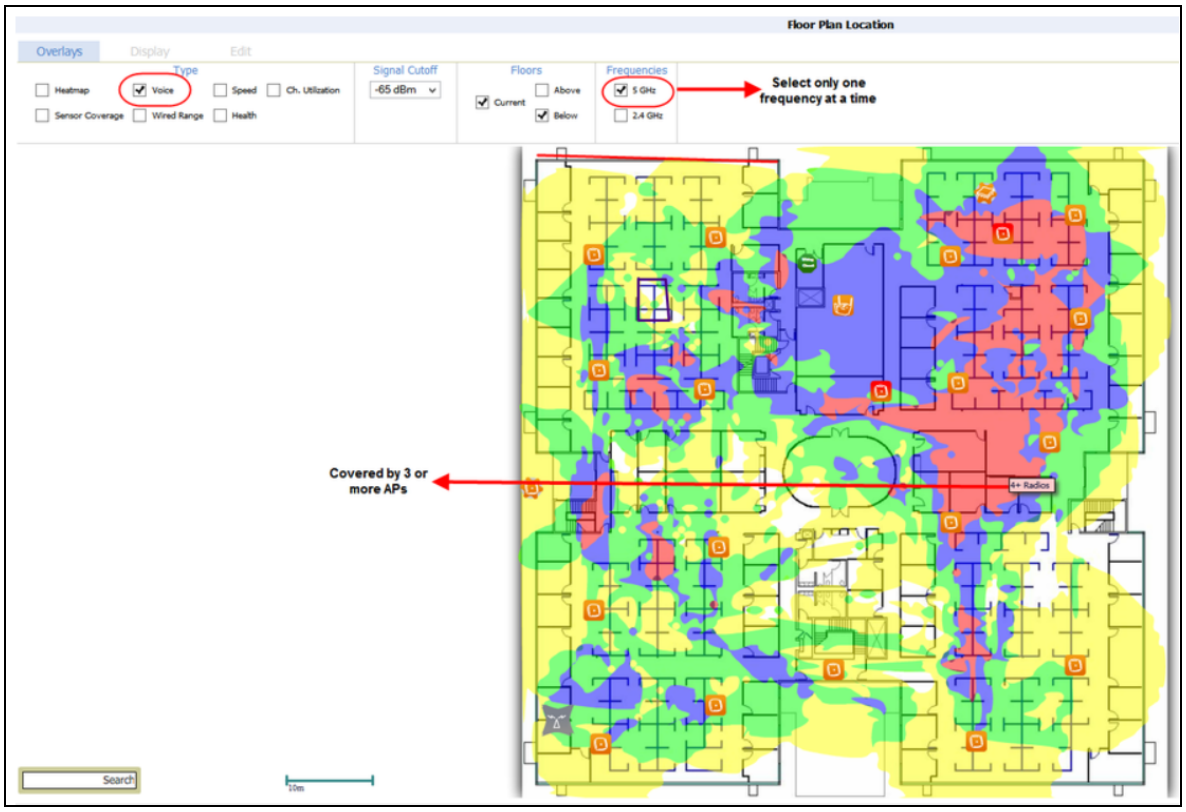
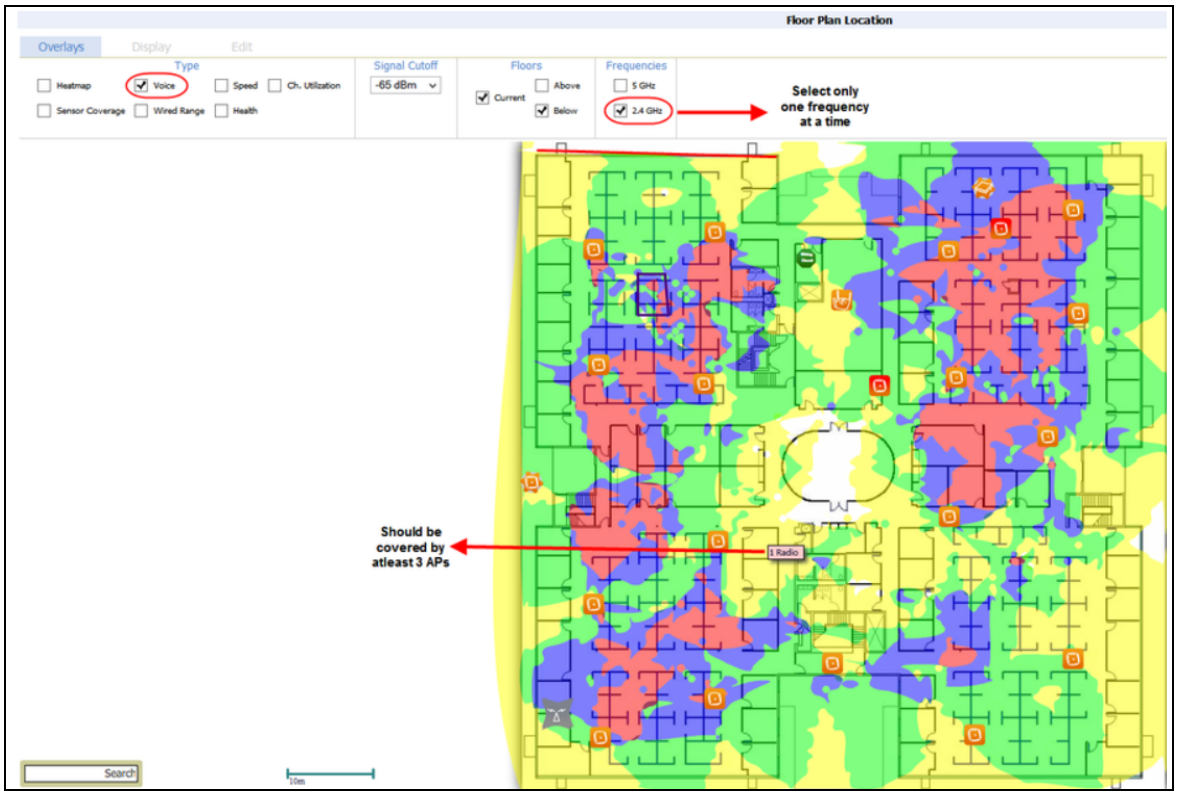
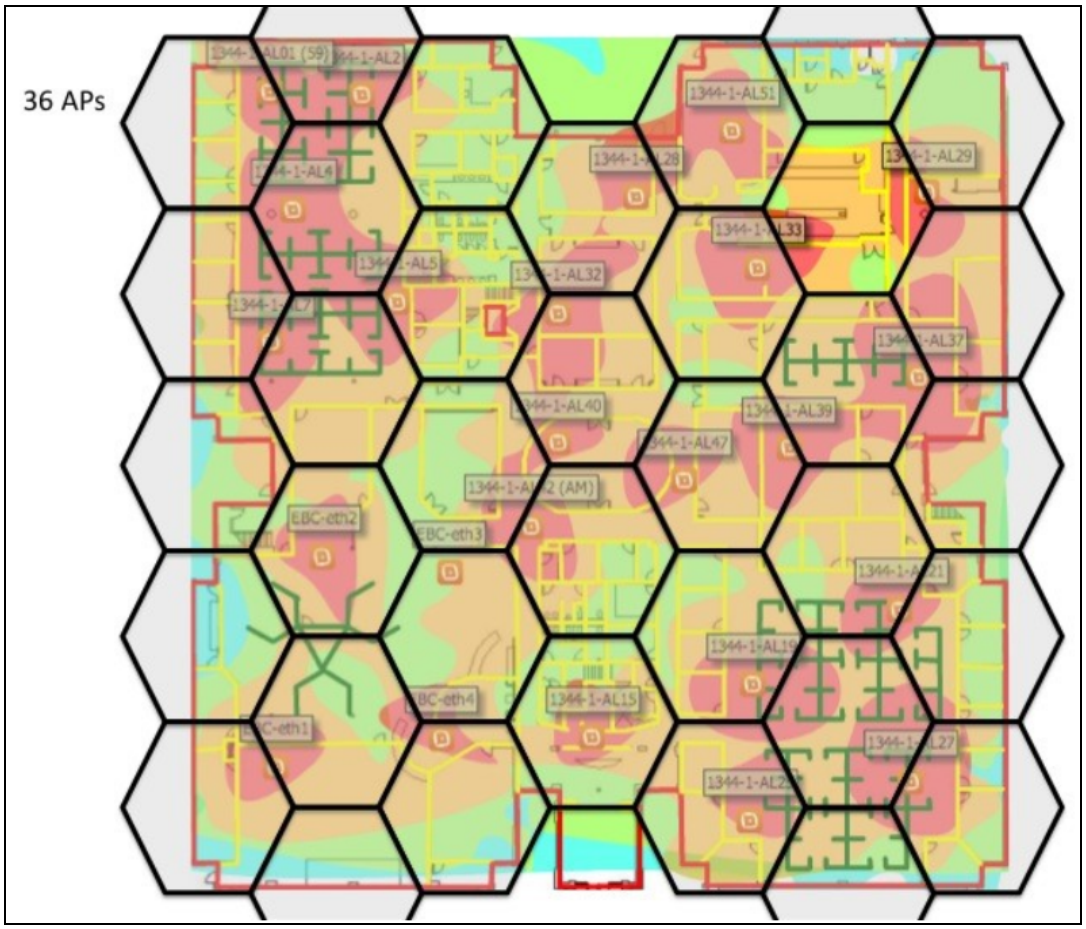


Figure 2 *Ensure Voice Overlay at frequency 24 GHz*



- APs must be deployed at the edge/boundary of the location and scatter inwards at appropriate distances. One AP per 2500 sq. feet is recommended.
- A coverage pattern with a minimum signal strength of -65 dbm for both 5GHz and 2.4 GHz is recommended.
- The popular hexagonal pattern should be used for AP layout to ensure that distance is normalized along all directions.
- Though circles are normalized, it is difficult to create tiles that cover the entire area. The hexagon is the smallest polygon with this property.

Figure 3 *Hexagonal Pattern for AP Layout*



AP Placement Recommendations by Priority

To follow AP placement recommendations by priority, see Table 4

Table 4: *AP Placement Recommendations*

Recommendation	Priority	Comments
Voice Overlay	1	Required in all deployments to achieve triangulation, which is the core requirement of location calculation
One AP every 2500 sq. feet (or 50 feet apart) and covered edges	1	Achieves good coverage pattern and triangulation, and is required for most deployments
Hexagonal pattern for AP layout	2	Recommended, but may be difficult to achieve in certain scenarios due to the physical layout
-65 dbm coverage	2	Strongly recommended, but may be difficult to achieve in certain parts of a building. In this case, ensure that there is at least a -75 dbm coverage in those areas.

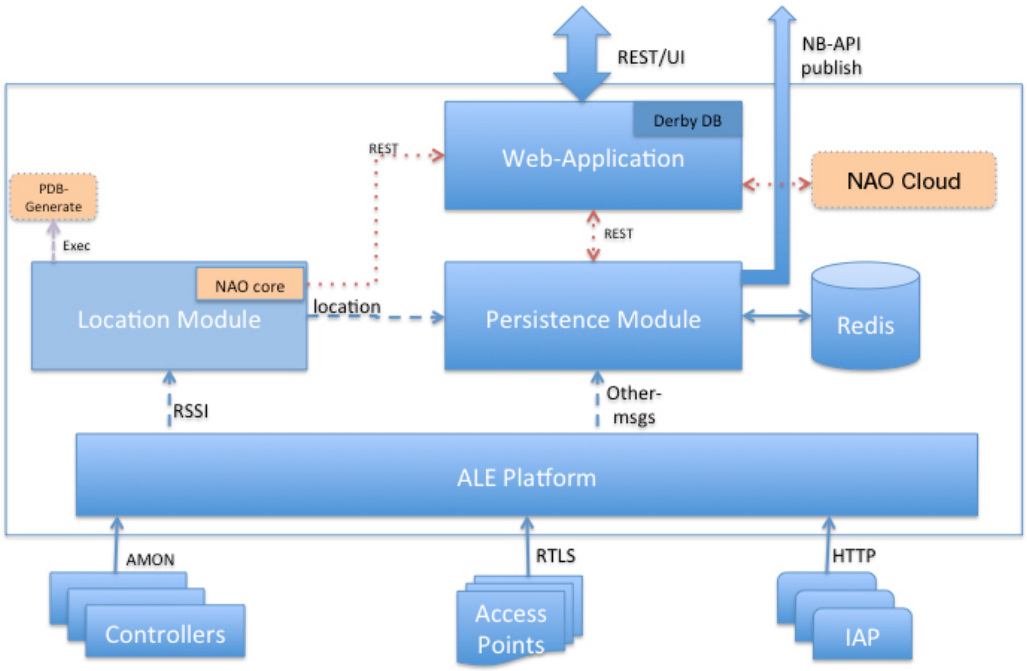
ALE Internal Architecture

ALE 2.0 operates in three modes:

1. **Context with no maps or locations**: Context mode does not use maps or floor plans to compute or report client location, eliminating the need for external AirWave servers or calibrations to deploy ALE. Deployment is simplified and scalability is improved since location, which is compute-intensive, is not calculated.

   All context topics from the NBAPI, such as Station, Application, Security, and Proximity, are reported, with the exception of Campus, Building, Floor, Location, and Geo_Fence. Though the Location topic is absent, this context mode uses Proximity to determine which access point is closest to the wireless client, providing a rough location estimation.
2. **Context with mapped locations**: Under the context mode with device location, ALE enriches the context information with calculated client locations using a map or floor plan. There are two methods of device location under this context mode:
   - **Context with Device Location (Estimated)**: This method uses AirWave to approximate the location of wireless clients using maps and AP placement information. This engine uses AP-AP RSSI messages to build a path-loss model and create a pseudo-positioning database (PDB). Each device is matched against the PDB to estimate client location. Fingerprinting and calibration are not required under this method.
   - **Context with Device Location (Calibration)**: Location accuracy is further improved using calibration data from fingerprinting. Fingerprinting is performed by Aruba Nao Campus, which runs as a web service on ALE, and its companion Android application, Aruba Nao Logger. Prior to deployment, each customer generates a real PDB using Aruba Nao Logger. Devices are matched against the customer-generated PDB to compute client location.

Figure 4 *ALE internal architecture*



ALE 2.0 contains four major components:

1. **Web Applications**: The ale-jwebapps component terminates the REST API to external applications and the WebUI. It also represents the code for interactions between ALE and Aruba Nao Campus or AirWave.
2. **Location**: The ale-location component consists of the NAO Core location engine and the ALE proximity engine.
3. **Persistance**: The ale-persistance component implements the REDIS database and publishes the ZMQ streams to external applications using the North Bound API. Key features, such as GeoFencing, are implemented through this component.
4. **Platform**: The ale-platform component implements software used for communicating with data sources such as AMON, RTLS, and IAP HTTPS.

Communication between the components relies on ZMQ feeds, REST API, or file exchanges. Each component is packaged as a separate RPM. For example, the location engine is part of the ale-location RPM, while Aruba Nao Campus is packaged in its own RPM.

Context Mode (Station, Application, Proximity)

Under context mode, ALE runs without any maps or floorplans for computing location. This simplified deployment relies on context information generated from the NBAPI, with the exception of Location and GeoFencing, to provide users with important analytics information. Context mode is available in both controller and IAP deployments.

Without the use of maps or locations, context mode uses the Proximity topic to estimate a rough location of the client. Proximity indicates which AP hears the client loudest every 30 seconds (default), measuring this value using RSSI.

Context Mode with Device Location

Estimation

Under context mode with estimated device location, ALE communicates with AirWave servers to estimate a Positioning Database (PDB). Location is determined by comparing the client RSSI input with the PDB, based on the locations of fixed APs and AP-AP RSSI information.

For location accuracy, it is important that the AP location on the Visual RF floor plan matches the physical AP location. Data received from the APs, which is not present on the floor plans imported into ALE, is not used for location calculation. Maps must be updated if the physical location of the AP changes or if additional APs are added.

If any map or AP placement is changed in AirWave, the AirWave server must be synced manually using the AirWave import wizard under **Configuration > Mode > Estimation** in the WebUI.

The port used for ALE-AirWave communication is TCP port 443 (HTTPS).

ALE 2.0 provides a facility to upload a Visual RF backup directly into ALE instead of communicating with the AirWave server online.

ALE can connect to and pull information from multiple AirWave servers.

AirWave 8.0.8 or higher is recommended.

The estimated PDB is generated once a datasource is selected, AirWave information is imported, and the mode is set to **Context with device location (estimation)**. The UI prompts the user to generate a PDB if any of these configurations are modified. PDB regeneration can be triggered manually at any time under **Maintenance > Regenerate PDB** in the WebUI.

The lack of specific calibration data can cause floor disambiguation, in which devices appear on the wrong floor.

Calibration

Context mode with calibration uses fingerprinting/calibration to compute client location. Under this deployment mode, ALE generates a PDB of location probabilities based on the RF characteristics of the deployment site.

Prior to deployment, ALE requires manual fingerprinting to create fingerprint locations for each map or floorplan. ALE uses an Android application called Aruba Nao Logger to fingerprint the site using sensors (MEMS) and WiFi RSSI variations recorded on the Android device. Nao Logger is adminstered by Aruba Nao Campus, which is a web-based tool that allows admins to upload floorplans and prepare the site for fingerprinting. To initiate fingerprinting, the site must be positioned in a map, and rails must be drawn to indicate possible device paths. Operators must walk along the path at the site, with an Android device running the Aruba Nao Logger application, and note down markers to indicate their real position from time to time (for example, each time the operator changes direction).

When fingerprinting is completed, the fingerprint measurements are recorded into the PDB. The PDB is retrieved and published by Aruba Nao Campus through an internal REST API. Once it is published, the PDB is downloaded and stored into the location engine by the ale-location component, allowing ALE to begin location computation and publication. ALE must be set to context with device location (calibration) in order to retrieve the published PDB. The PDB can be re-fetched under **Maintenance > Regenerate PDB** in the WebUI. Device location is matched against the stored PDB values to estimate client location, offering the highest location accuracy possible on ALE. To maintain location accuracy, periodic fingerprinting may be required (once every six to twelve months).

Non-location topics are published when AMON or HTTPS streams are established from the controller or IAP.

Aruba Nao Campus provides ALE with the following information to calculate location:

- Sites
- Geo-references
- Zones
- Integration keys
- Positioning database information

This information is stored in the local derby database on the ALE server. Information for the REST API is stored in the REDIS database.

Information retrieved from Aruba Nao Campus (sites, geo-references, etc.) cannot be modified on the ALE server.

ALE and the Aruba Controller

In an Aruba controller-based WLAN, both the controller and ALE must be configured to communicate with each other. See Installing and Configuring ALE on page 23 for more details on how to configure the controller and ALE for a controller-based WLAN.

The ALE and Aruba Controller Workflow

The following describes the communication and workflow between ALE and the controller:

1. By default, every AP in an Aruba controller-based WLAN calculatesthe RSSI of clients in its vicinity, which is sent to the controller. Alternatively, the AP RTLS feed can be directed to ALE.
2. Once ALE and the controllers are configured to communicate with each other, ALE performs an HTTP GET of AMON data to retrieve the available information. This bootstrap operation is performed when ALE initially communicates with a controller.
3. After the initial HTTP GET, all proceeding data from the controller is sent to ALE through a time-based AMON push.
4. The controller sends RSSI data and other information, such as user role, applications, and destinations, as AMON data to ALE.
5. ALE publishes the data as soon as it is calculated, using the northbound API. The northbound API also forwards this data to the Redis DB.

Both CAPs and RAPs are supported.

Integrating ALE with Instant AP

ALE supports integration with IAPs. The ALE server acts as a primary interface for all third-party applications, and the IAP sends client information and other status information to the ALE server. In order for the IAP to integrate with ALE, the ALE server address must be configured on the IAP.

Enabling the IAP for ALE Support

The HTTPS pipe transports data from the IAP to the ALE server.

To enable an IAP for ALE support, run the following commands:

```
(host)#configure terminal
(host)(config)#ale-server <server:port>
(host)(config)#exit
(host)#commit apply
```

Table 5: *ALE Support Configuration Parameters*

Command/Parameter	Description
`ale-server <server:port>`	Allows you to specify the Fully Qualified Domain Name (FQDN) or IP address of the ALE server
`no…`	Removes the specified configuration parameter

The following example enables ALE on an IAP:

```
(host)(config)#ale-server AleServer1:8088
```

Interval Configuration

To configure the interval at which an IAP sends data to the ALE server, use the following command:

```
ale-report-interval <seconds>
```

Table 6: *Interval Configuration Parameters*

Command/Parameter	Description	Range	Default
`ale-report-interval <seconds>`	Configures an interval at which the Virtual Controller can report the IAP and client details to the ALE server	6-60 seconds	30
`no...`	Removes the specified configuration parameter	–	–

The following example configures the ALE server details:

```
(host)(config)# ale-report-interval 60
```

About ALE and Firewalls

The following table displays firewall configuration information for different types of ALE communication:

Table 7: *Firewall Configuration for ALE*

Communication	Port	Protocol	Notes
Controller > ALE	8211	PAPI	For AMON feed from the controller
Instant VC > ALE	8088	HTTPS	Port is configurable
ALE > Controller	4343	HTTPS	For fetching API data
ALE > AirWave	443	HTTPS	Fetch floorplan data from AirWave
ALE > Cloud	7779		ZeroMQ feed
HTTP Client > ALE	443	HTTPS	Restful APIs
User > ALE	443	HTTPS	ALE GUI

About Anonymization

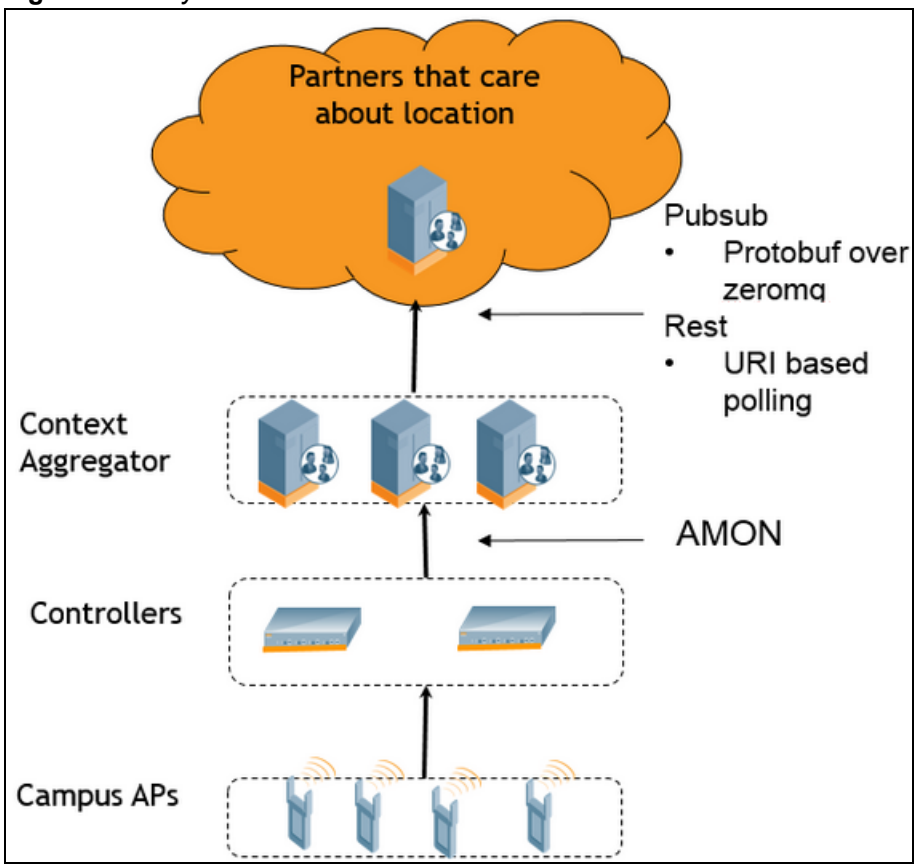
Direct use of a user's personal identifying information, which exists within specific message fields of the AMON data feed, raises privacy concerns within ALE.

Serious privacy issues arise when personal network and mobile device identification information, which is saved by ALE, is shared with outside applications. Information such as the device's MAC address, acquired (associated) IP Address (or a history of it), username, and other sensitive identification information must be anonymized. Anonymization cannot be reversed.

Anonymization Basics

- Anonymization is an optional configuration that is activated by default. Once activated, all subscribers receive anonymized data.
- The following fields are anonymized for both Publish/Subscribe and REST APIs:
  - mac_addresses
  - ip_addresses
  - usernames

Figure 5 *Anonymization Basics*



- When anonymization is activated, only the GET ALL option for REST queries is supported. Filtering by MAC address, IP address, or username returns an error.
- ALE uses a salted keyed SHA-1 hashing algorithm to generate the anonymized fields.

The following example displays the message output for a station query to a northbound API when ALE anonymization is turned *on*:

```
{
   "Station_result":[
      {
         "msg":{
            "role":"Aruba-Employee",
            "bssid":{
               "addr":"6CF37FEC1110"
            },
            "device_type":"iPad",
            "hashed_sta_eth_mac":"041CB396A0844FE3BF3A6F22B7475ED037BD972B",
            "hashed_sta_ip_address":"34A71F00D8A61467739009283665CE47CEC21E1A"
         },
         "ts":1393536217
      }
```

```
   ]
}
```


When anonymization is turned *off*, the same station query displays the following message output:


```
{
   "Station_result":[
      {
         "msg":{
            "role":"Aruba-Employee",
            "username":"jdoe",
            "sta_eth_mac":{
               "addr":"6482FFBB2A35"
            },
            "bssid":{
               "addr":"6CF37FEC1110"
            },
            "sta_ip_address":{
               "af": "ADDR_FAMILY_INET",
               "addr": "10.100.239.186"
            },
            "device_type":"iPad",
            "hashed_sta_eth_mac":"041CB396A0844FE3BF3A6F22B7475ED037BD972B",
            "hashed_sta_ip_address":"34A71F00D8A6146773900928 3665CE47CEC21E1A"
         },
         "ts":1393536217
      }
   ]
}
```


The red text appears in the message output file when anonymization is off.

Changing the Salting so the Hash MAC Addresses cannot be Traced

ALE supports the collection of MAC addresses. However, the salting can be changed so hash MAC addresses cannot be traced, by setting the salting schedule. See Configuring ALE for more information.

This chapter describes how to install and configure ALE, and includes the following topics:

ALE Installation and Configuration Workflow

The following workflow highlights the steps to install and configure ALE. To activiate the ALE services on your network:

1. **Install ALE**: Install ALE 2.0 on a virtual machine or bare metal server. See Installing ALE 2.0 for installation details.
2. **Launch the UI**: Launch the WebUI using your login credentials.
3. **Configure Settings**: Configure system settings through the ALE Setup Wizard, which launches automatically for a factory default ALE.
   a. **Mode**: Set the deployment mode (context, context with estimated location, context with calibrated location)
   b. **Source**: Define the data source (controller or IAP)
   c. **Options**: Configure optional settings (for example, enable/disable GeoFences, anonymization, WebSocket Tunnel, etc.)
   d. **License**: Upload licenses (evaluation or permanent)
4. **Launch ALE**: The setup wizard applies the configuration and displays the ALE dashboard.

For more details on the ALE Setup Wizard, see The ALE Setup Wizard.

Settings can also be configured under the **Configuration** tab of the WebUI. For more details, see Configuring ALE.

Installing ALE 2.0

The ALE 2.0 software can be installed on a virtual machine or a bare metal server with the following minimum requirements:

- 8 cores
- 16 GB RAM
- 120 GB hard disk space

Installing ALE on a Virtual Machine

ALE is delivered as an ISO or OVA file, supported on VMware ESX/ESXi 5.x. The file can be deployed using VMware vSphere. This procedure automatically creates a virtual machine (VM) with the following specifications:

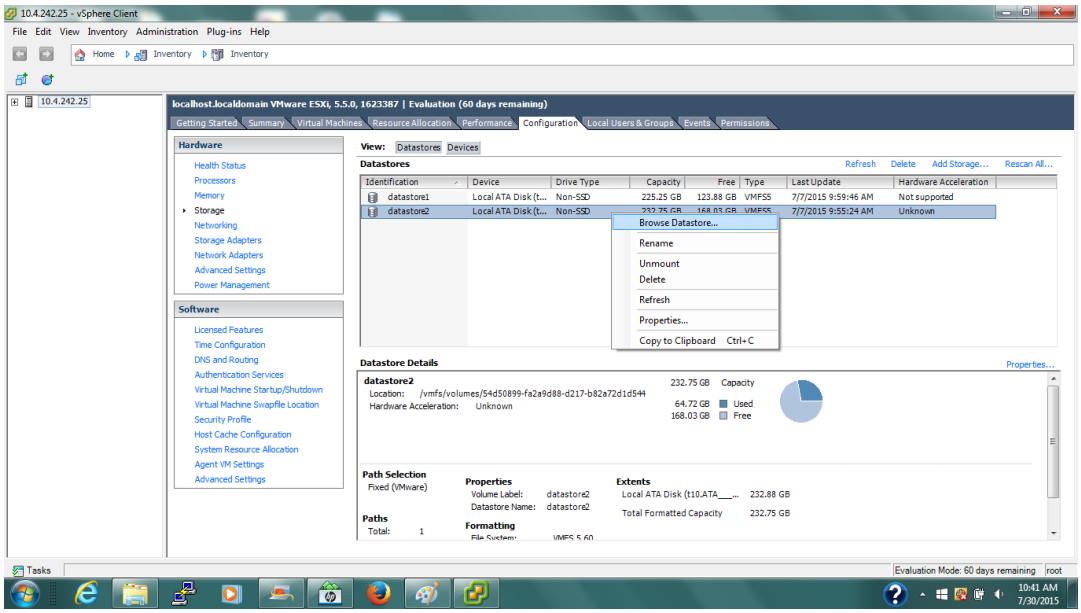
- Guest Operating System = Linux
  - Version = CentOS 6.x with Oracle JDK 1.8
- Number of Virtual Sockets = 1
  - Number of Cores per Virtual Socket = 8
- Memory Configuration = 16 GB
- Number of NICs = 1

During deployment, the VM can be sized based on Aruba Networks, Inc. technical support recommendations. Following deployment, the VM CPU, memory, and hard drive can be modified if any performance issues arise.

The following steps describe how to install ALE on a VM using an ISO file:

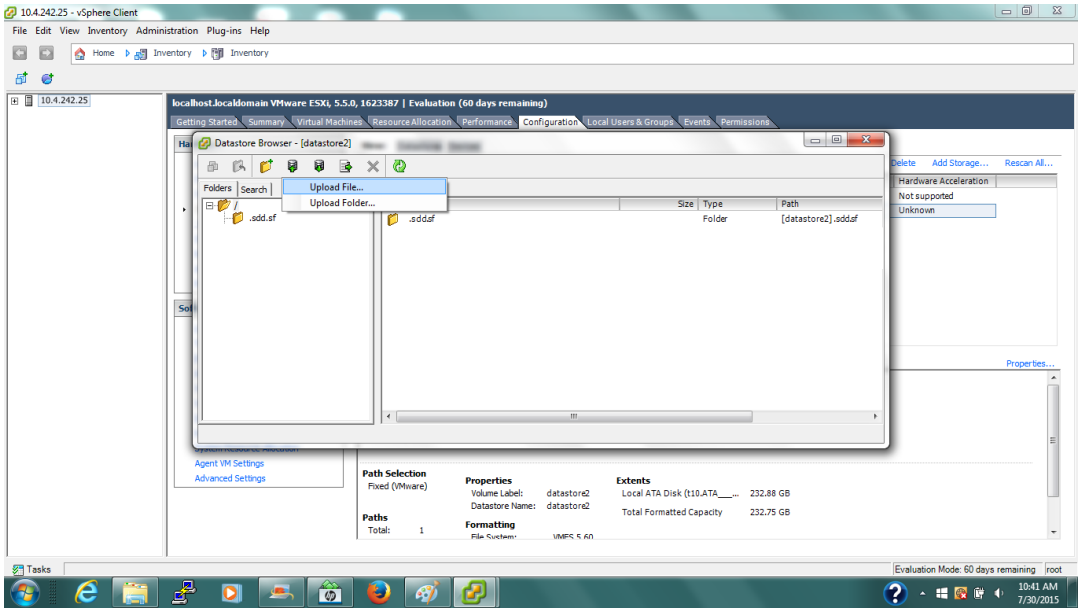
1. Download the ALE ISO file.
2. Open the vSphere client and select a server from the top left window pane.
3. Under the **Configuration** tab, right click on the datastore to upload the ISO file.

Figure 6 *The vSphere Client*



   a. Click **Browse Datastore...**. The **Datastore Browser** opens.
   b. Click **Upload File...**. The file manager opens.

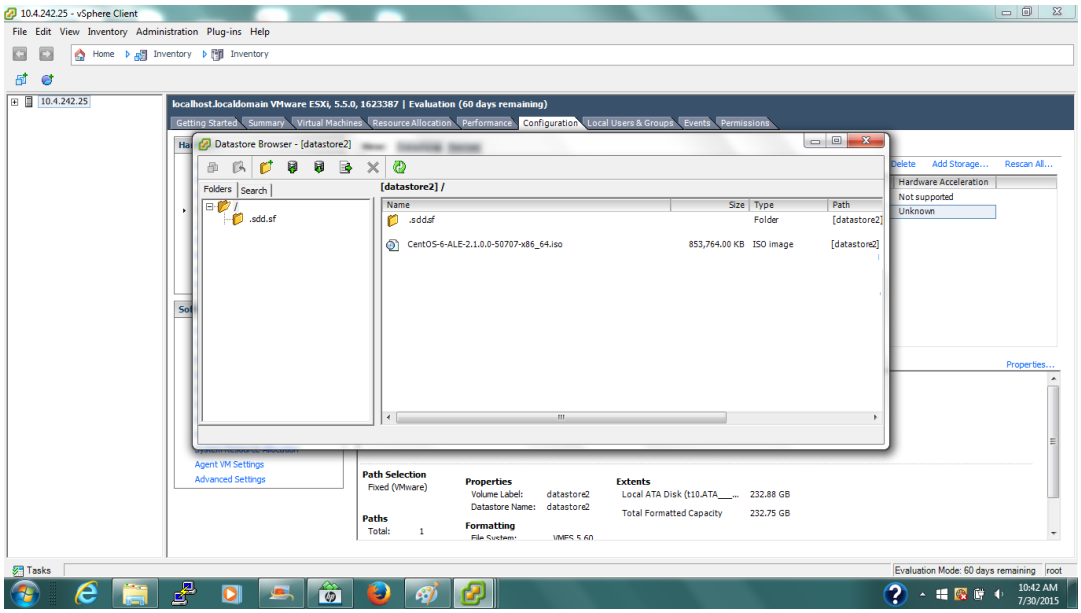
Figure 7 *Uploading the ISO File*



c. Locate and select the ISO file, and then click **Open**.

d. Once the ISO file is uploaded to the server, it appears in the datastore window.

Figure 8 *The Uploaded ISO File*



4. To create a new virtual machine (VM), navigate to **File > New > Virtual Machine**. A popup window appears.

Figure 9 *Creating a new Virtual Machine*

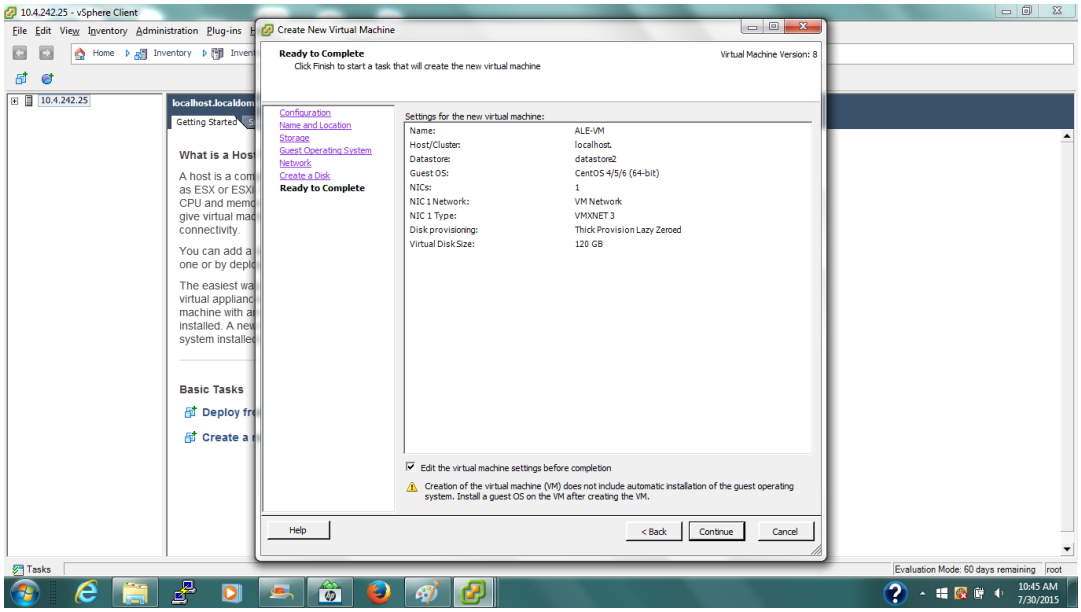
a. Under **Configuration**, select **Typical**. Click **Next**.
b. Enter a name for the ALE VM, and then click **Next**.
c. Select a datastore for the VM, and then click **Next**.

NOTE

The datastore must have at least 120 GB of free space available.

d. Set the **Guest Operating System** to **Linux**, and select **Centos 4/5/6 (64-bit)** from the **Verson** drop-down menu. Click **Next**.
e. Under the **NIC 1** drop-down menu, select **VM Network**. Click **Next**.
f. Set the **Virtual disk size** to 120 GB, and select **Thick Provision Lazy Zeroed**. Click **Next**.

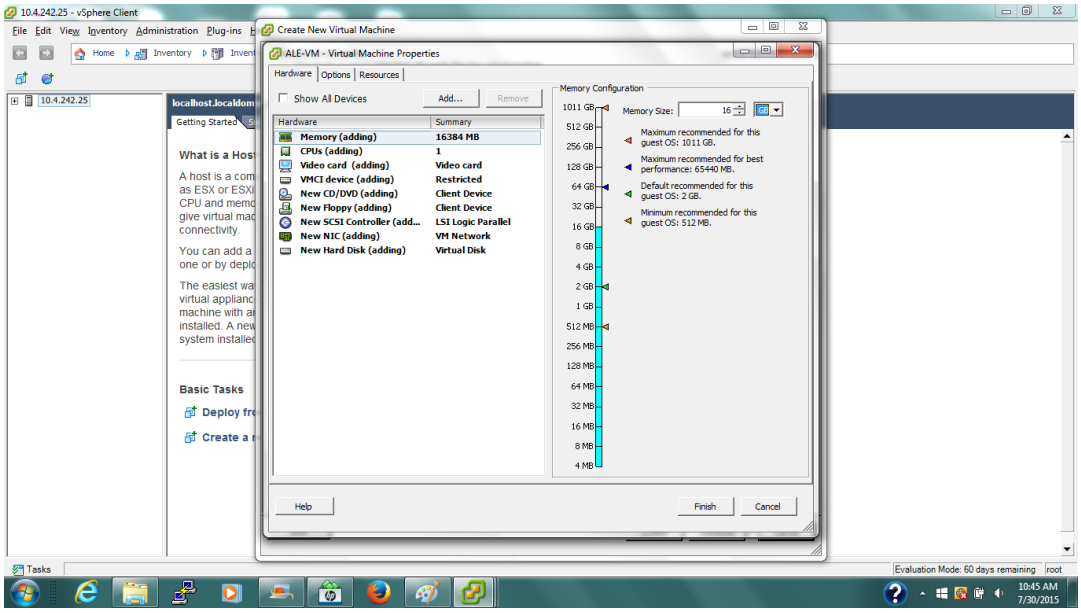
Figure 10 *Configuring the Virtual Machine*



g. A summary of the configuration settings appears under the **Ready to Complete** page.

h. Select the check box for **Edit the virtual machine settings before completion** to configure additional settings, and then click **Continue**. The **Virtual Machine Properties** popup window opens.

Figure 11 *Virtual Machine Properties*

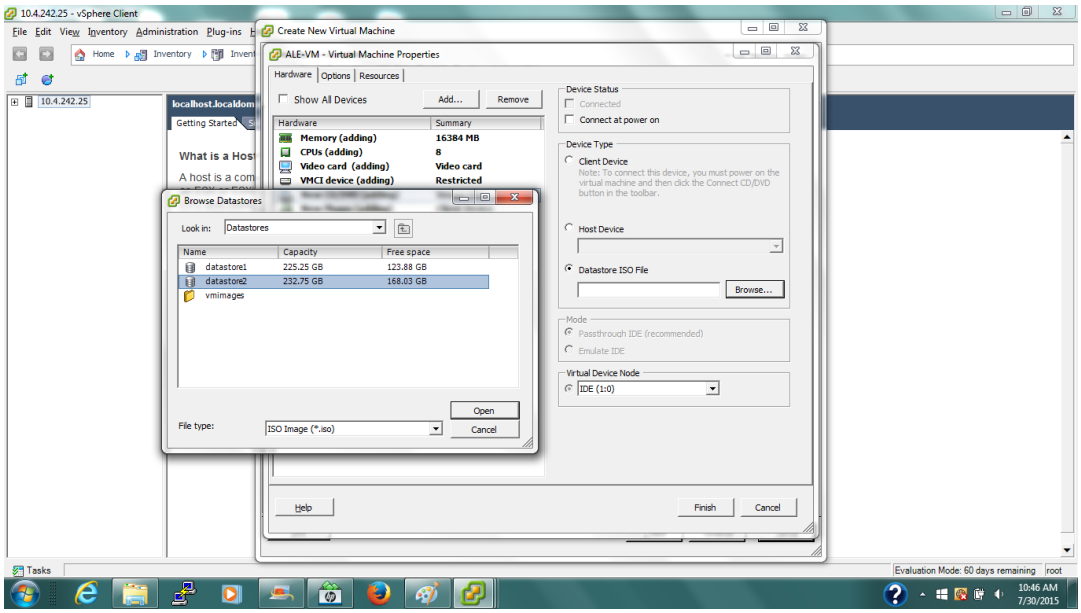


a. Under the **Hardware** tab, select **Memory (adding)** from the left window pane. Set the **Memory Size** to 16 GB.

b. Select **CPUs (adding)** from the left window pane and set the **Number of virtual sockets** to 8.

c. Select **New CD/DVD (adding)** from the left window pane, and then select **Datastore ISO File** under **Device Type**. Click **Browse** to locate and select the datastore containing the ISO file. Click **Open**.

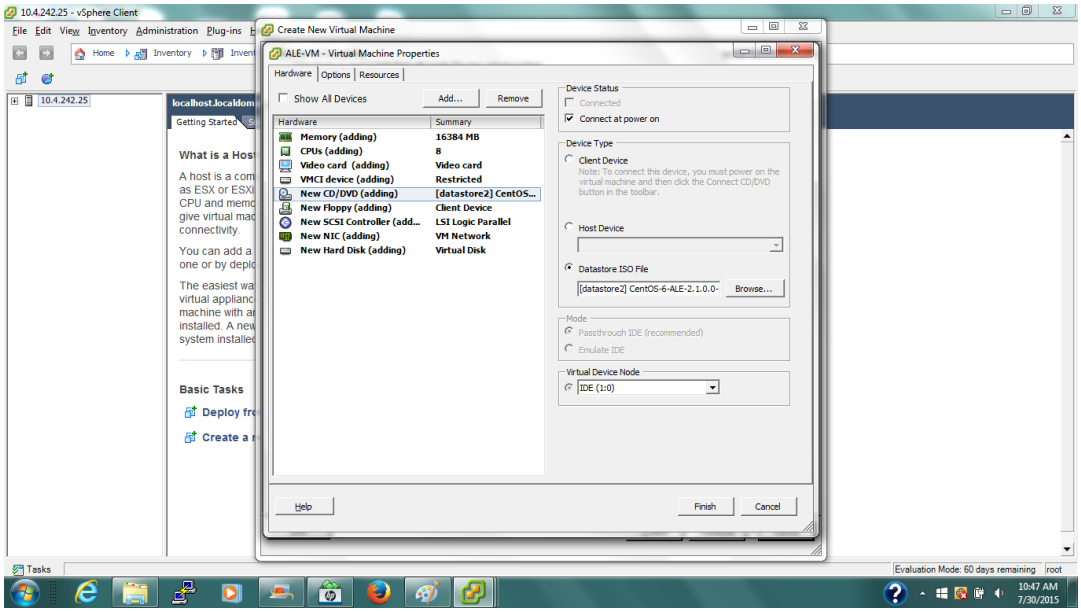
Figure 12 *Selecting Datastores for the Virtual Machine*



d. Select the ISO file, and then click **OK**.

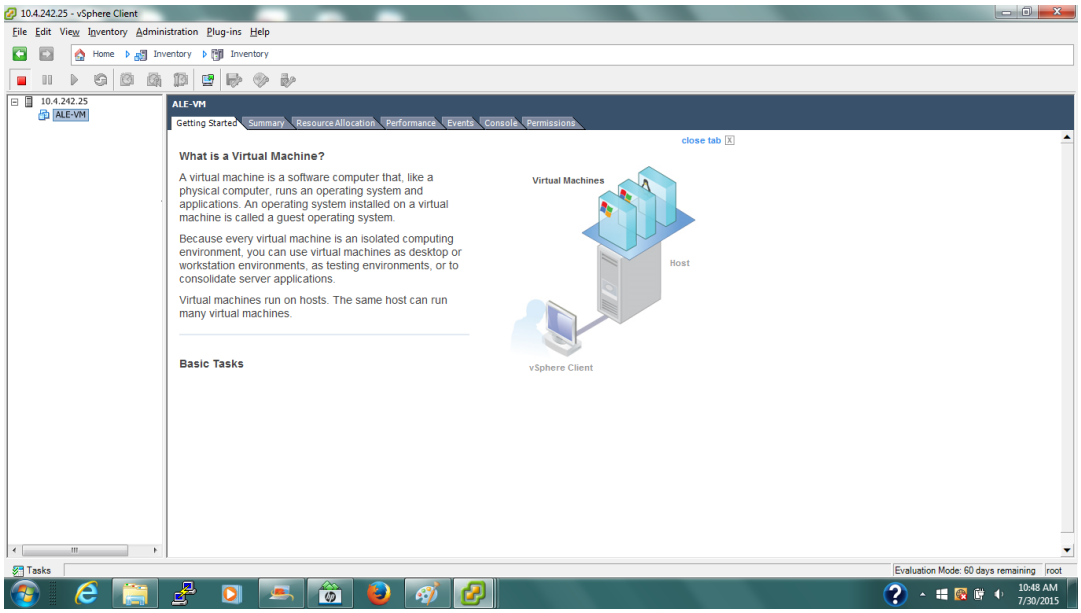
e. Select the check box for **Connect at power on** under **Device Status**, and then click **Finish**.

Figure 13 *Completed Virtual Machine Configuration*



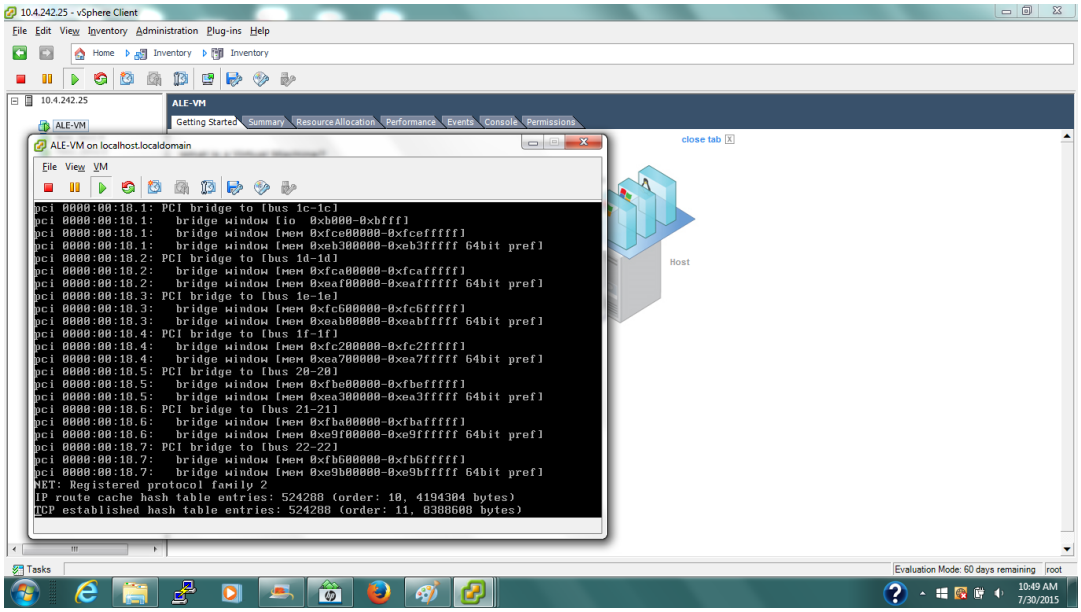
f. Once the VM is created, it appears under the server list in the vSphere client.

Figure 14 *New Virtual Machine on the vSphere Client*



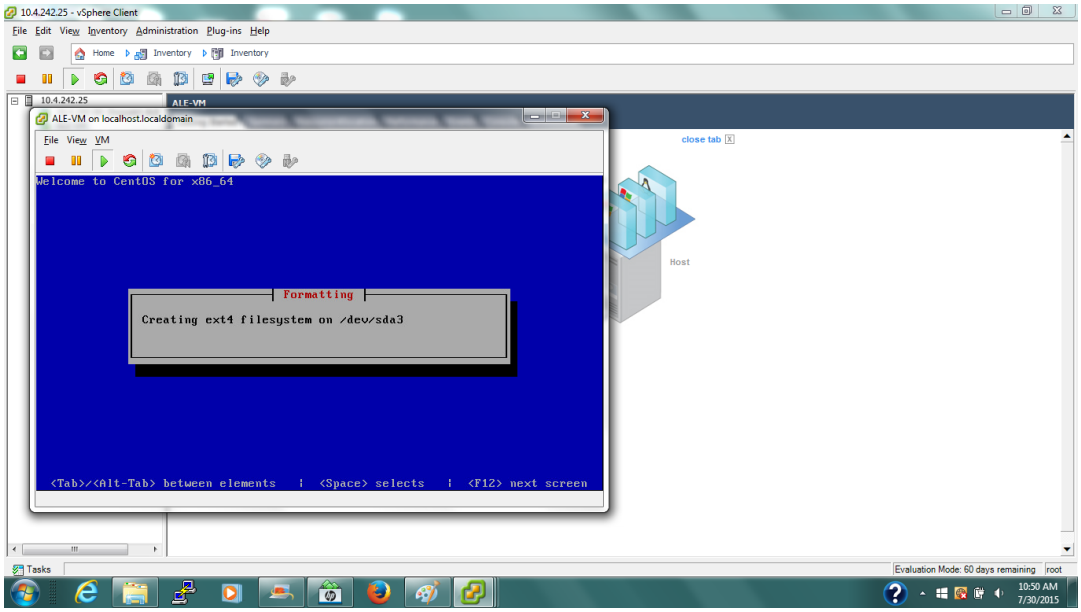
5. Select the newly created VM to begin ALE installation. The VM popup window appears.

Figure 15 *Installing ALE on the New Virtual Machine*



a. Click the green play button to begin installation.

Figure 16 *ALE Installation Process*



b. To view the console, click on the **Console** tab in the vSphere client or click the console button on the VM popup window.

c. Once installation is complete, the VM powers off.

Figure 17 *Completed ALE Installation*

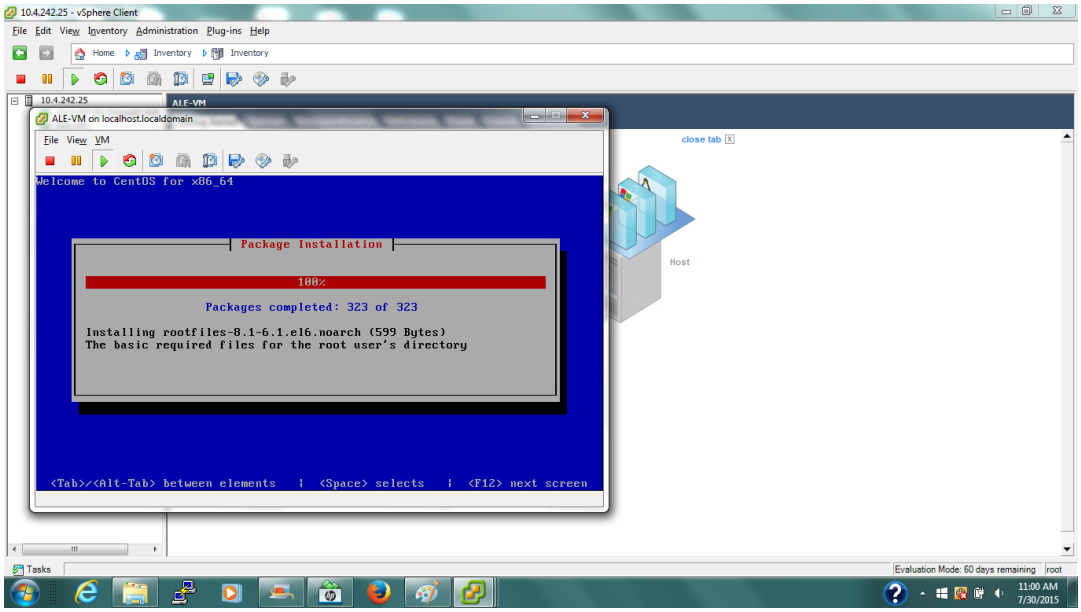
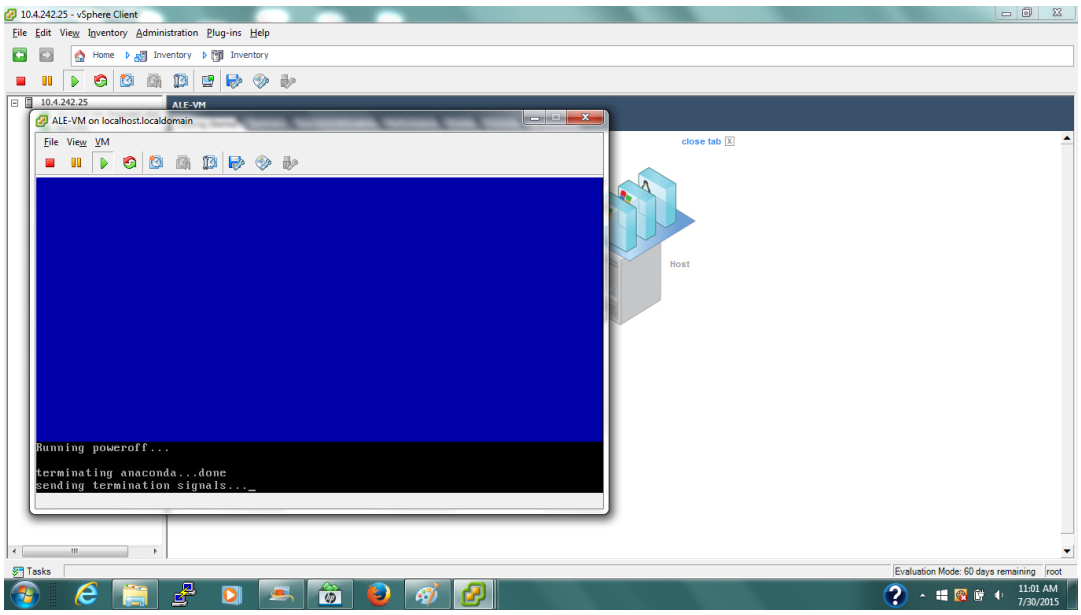
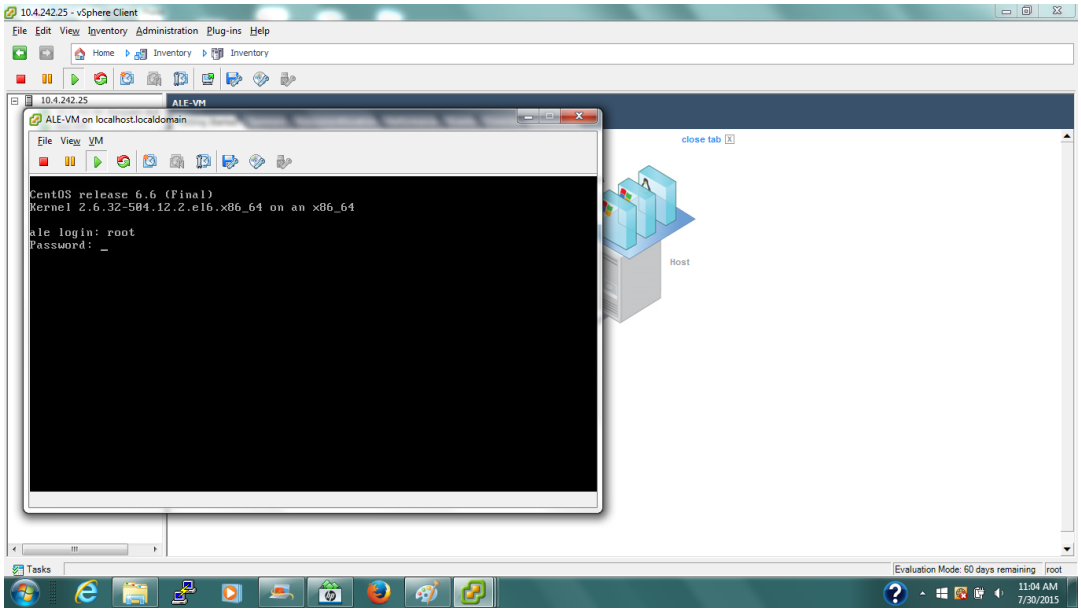


Figure 18 *ALE Poweroff*



6. Click the green play button on the VM popup window to boot up the VM.
7. Once the VM has booted up, login using your ALE credentials to begin using ALE.

Figure 19 *Logging in to ALE*



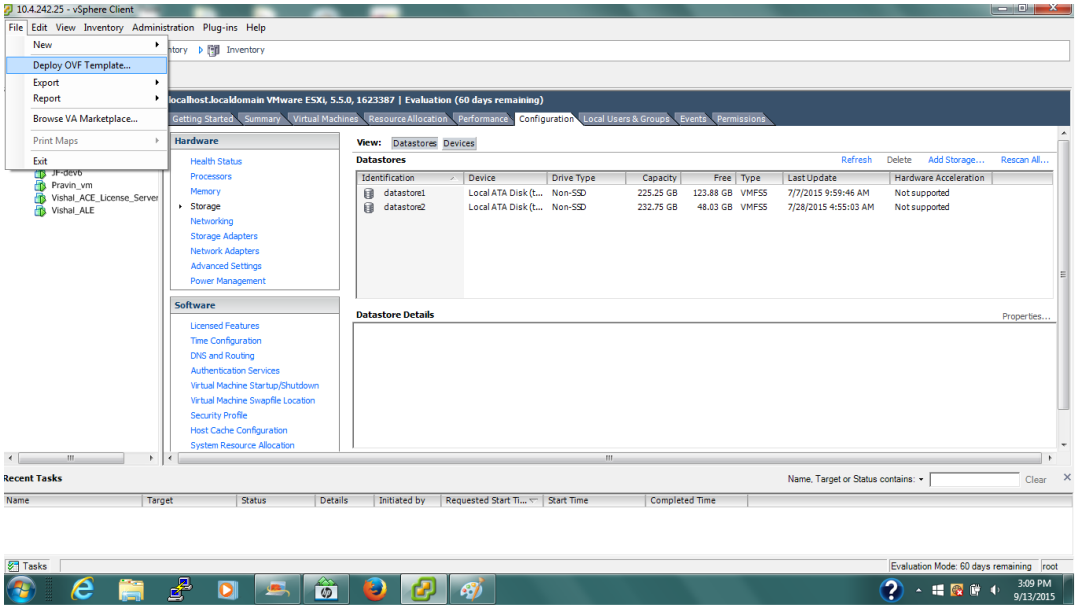
NOTE

NTP servers can be added to the ALE server through the WebUI. Once the NTP server is added, ALE syncs with the NTP and sets to the UTC time zone. For information on how to set the server to a different time zone through the shell (not required for ALE), visit CentOS Blog.

The following steps describe how to install ALE on a VM using an OVA file:

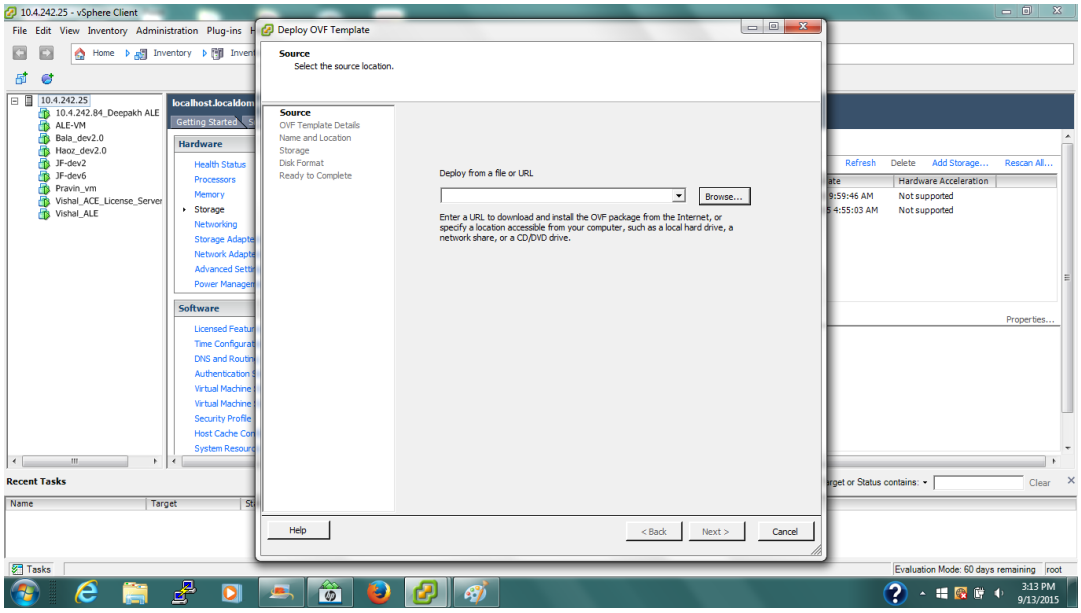
1. Launch the vSphere client.
2. Click **File > Deploy OVF Template**. The file browser opens.

Figure 20 *Deploying the OVF Template*



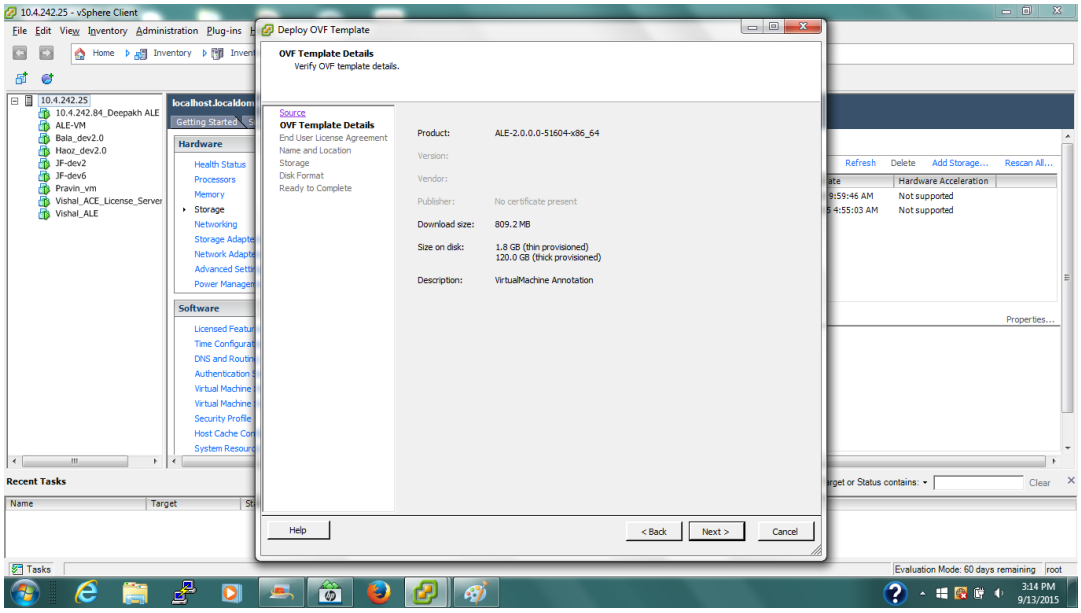
3. Under **Source**, enter the URL of the OVA file or locate and select the file using the **Browse** button.

Figure 21 *Selecting the OVA File*



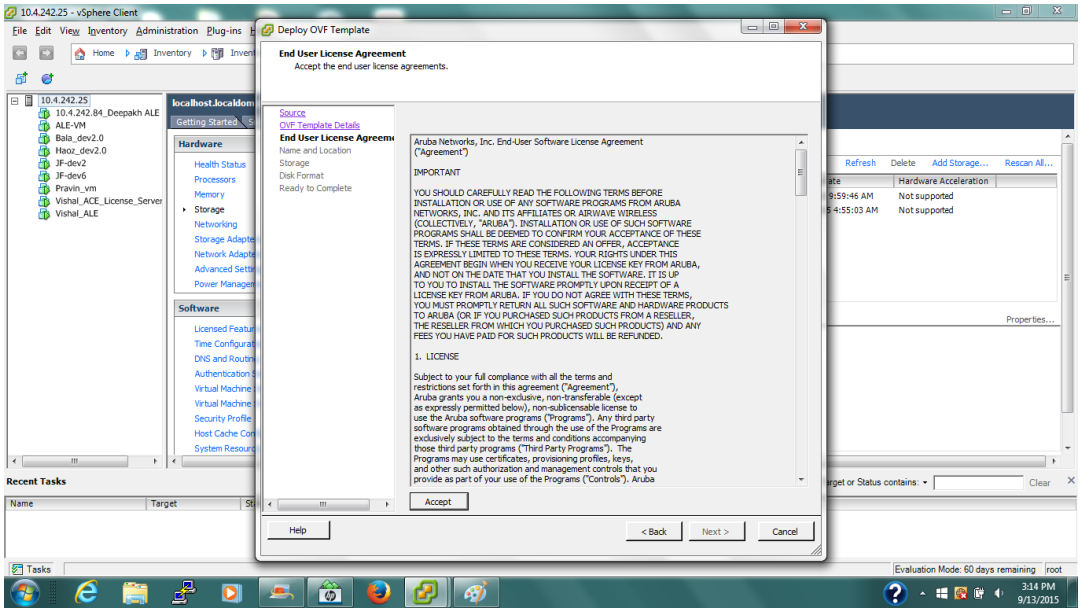
4. After selecting the OVA file, click **Next**.
5. The **OVF Template Details** tab highlights the default OVA configuration. Click **Next**.

Figure 22 *Default OVA Configuration Details*



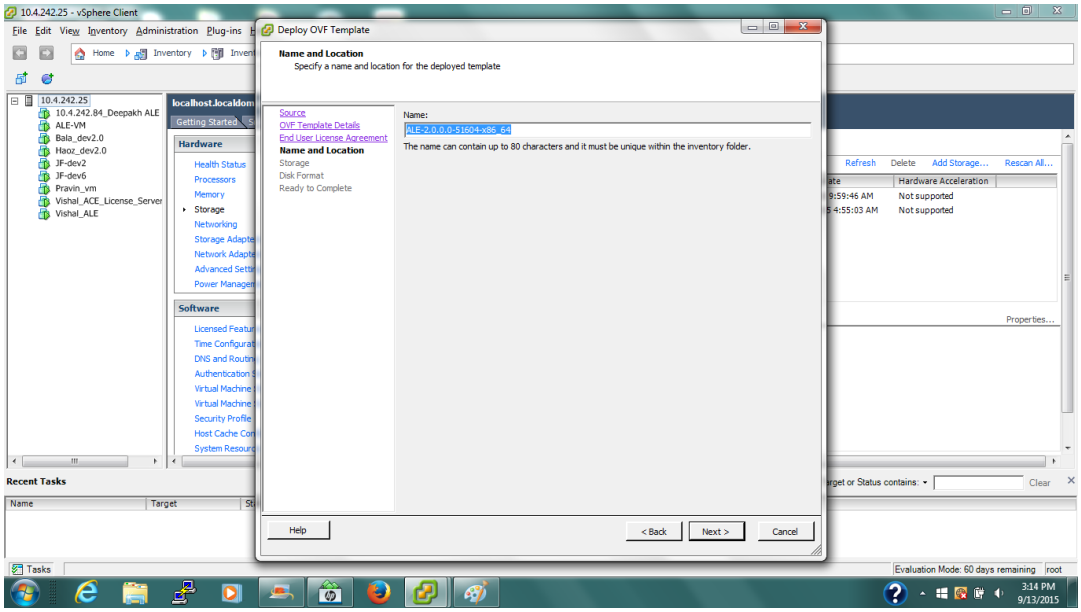
6. The **End User License Agreement** tab displays the Aruba Networks, Inc. end-user software license agreement. Click **Accept**, and then click **Next**.

Figure 23 *The Aruba Networks, Inc. End-User Software License Agreement*



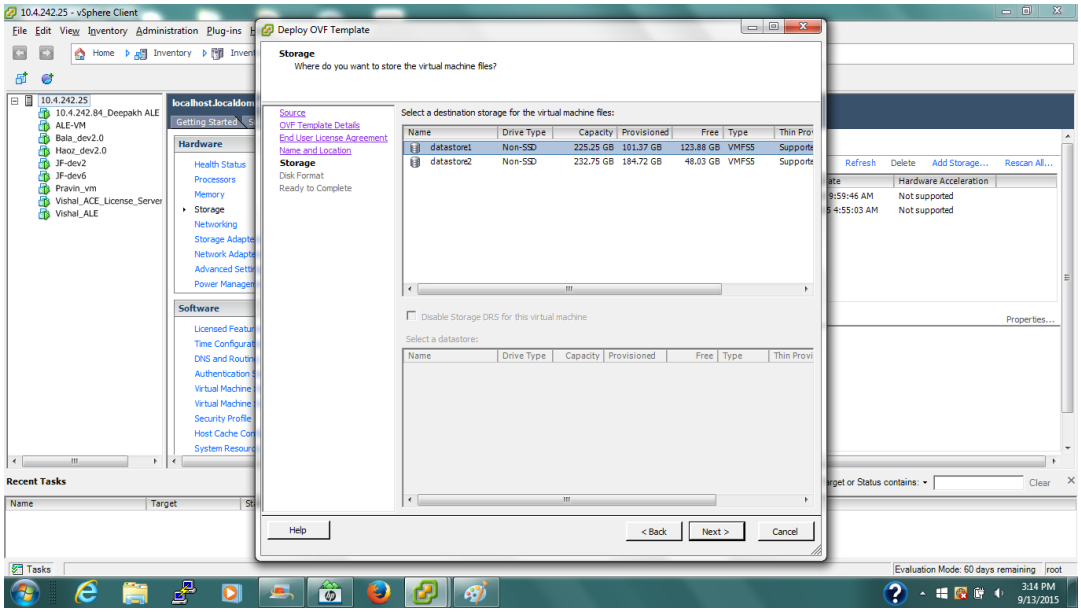
7. Under **Name and Location**, enter a name for the OVA file, and then click **Next**.

Figure 24 *Naming the OVA File*



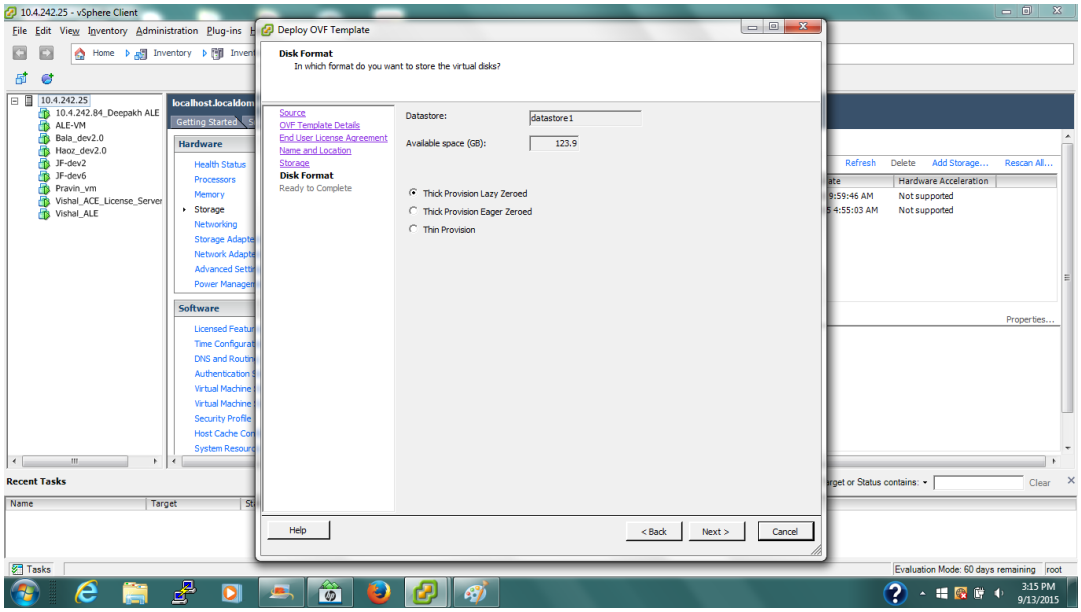
8. Select a datastore with enough free memory available to install the OVA file. Click **Next**.

Figure 25 *Selecting a Datastore*



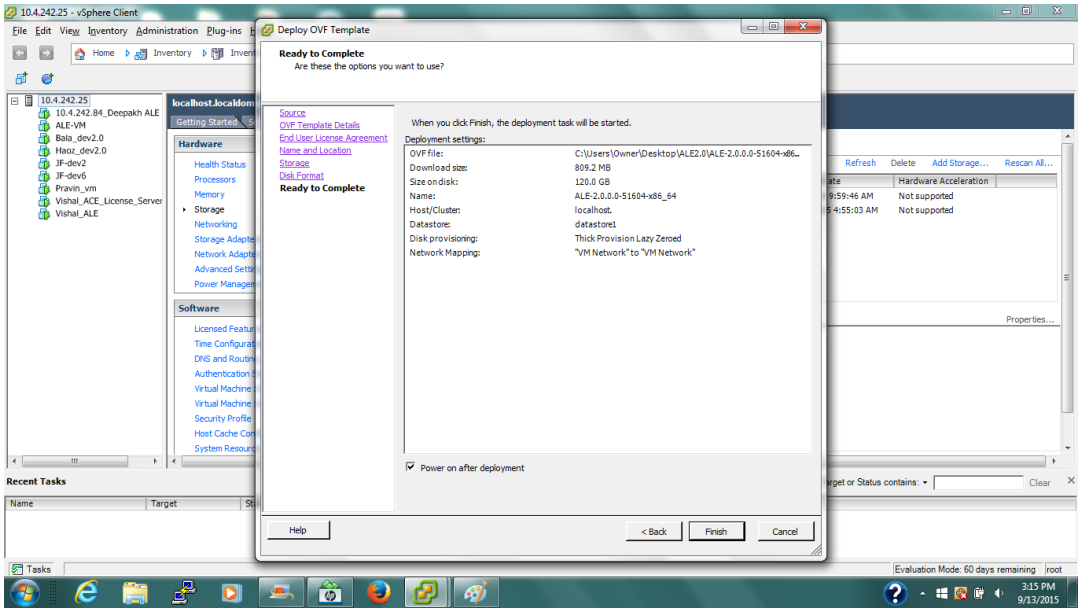
9. Under **Disk Format**, select **Thick Provision Lazy Zeroed**. Click **Next**.

Figure 26 *Setting the Disk Format*



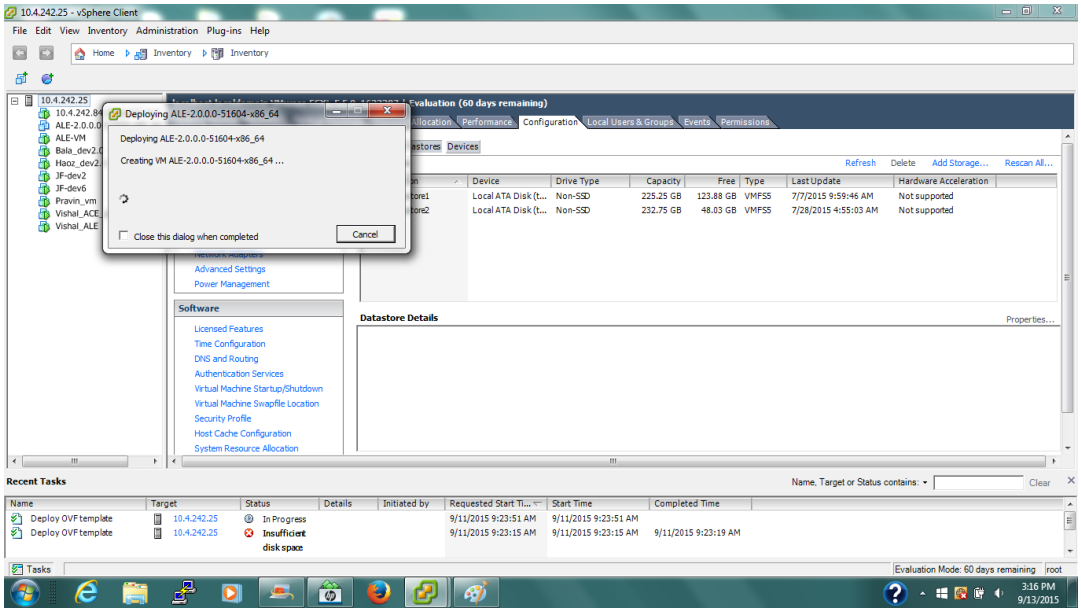
10. Verify all configuration details under the **Deployment Settings** summary page.
    - If any changes must be made, click **Back** to navigate back to any previous pages.
    - If all settings are correct, select **Power on after deployment**, and then click **Finish**.

Figure 27 *The Deployment Settings Summary Page*



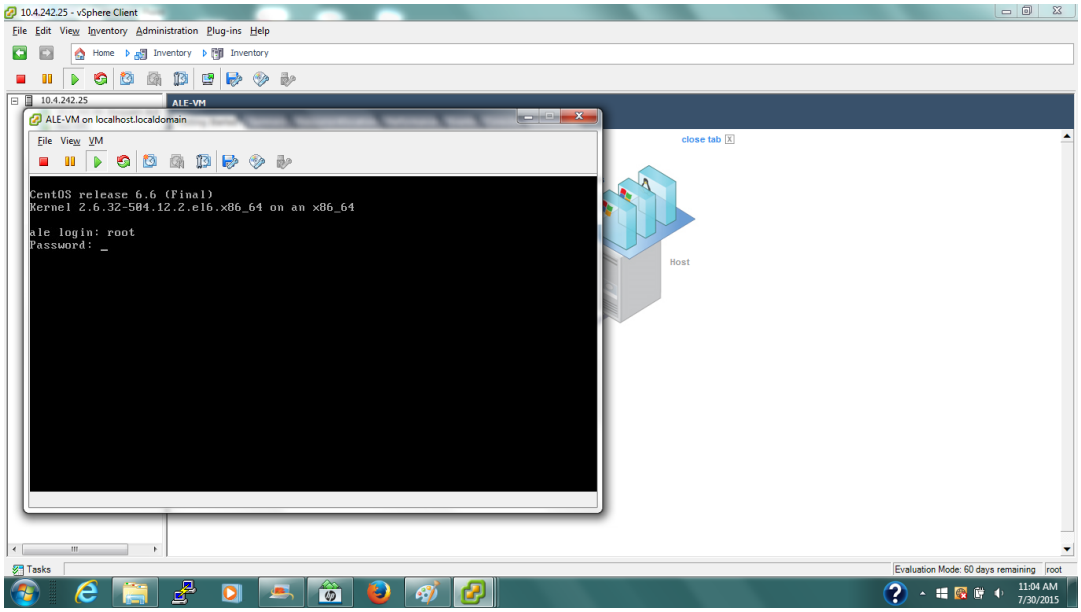
11. OVA installation begins immediately.

Figure 28 *OVA Installation*



12. Once the VM has booted up, login using your ALE credentials to begin using ALE.

Figure 29 *Logging in to ALE*



Configuring the IP Address of the ALE Server

Complete the following steps to configure the IP address of the ALE Server. The commands shown below are standard CentOS commands.

1. Loging to the ALE shell as the root.
2. Issue **ifconfig -a** to make sure the interface you are configuring the IP address on is available.
3. Run the **system-config-network** utility on the ALE console.
4. Edit the IP address, gateway IP, DNS server IP, and netmask for eth0.
5. If eth1 appears in your ALE instance, replace all references to **eth0** with **eth1** in the document.
6. Save and quit
7. Verify that **/etc/sysconfig/networ-scripts/ifcfg-eth0** has been created and check that its contents match what was configured.
8. Execute **/etc/init.d/network restart**.
9. Execute **ifconfig** and **route –n** to verify that the IP address and routes are accurate.

Alternate Method of Configuring the IP Address

The IP address can also be configured using the following alternate method:

1. Login to the ALE server as the root.
2. Run the **system-config-network** utility on the ALE console. The interface eth0 comes up on ALE and **vi/etc/sysconfig/network-scripts/ifcfg-eth0** is created.
3. Using the **IPADDR=<IPADDRESS>** command, set the static IP address.
4. Set the appropriate netmask through the **NETMASK=<NETMASK>** command.
5. Add the DNS1 and DNS2 server IP using the **DNS1=<DNS SERVER IP>** and **DNS2=<DNS SERVER IP>** commands:
6. Assign a gateway using the **GATEWAY=<SPECIFIC GATEWAY>** command.

7. Execute **/etc/init.d/network restart** to restart the server.
8. Execute **ifconfig eth0** to verify that the IP address is assigned.

The following example displays the alternate method for configuring an IP address:

```
## Configure eth0
#
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
NM_CONTROLLED="yes"
ONBOOT=yes
HWADDR=A4:BA:DB:37:F1:04 (REPLACE WITH IP ADDRESS OF ETH0, if different)
TYPE=Ethernet
BOOTPROTO=static
NAME="eth0"
UUID=5fb06bd0-0bb0-7ffb-45f1-d6edd65f3e03
IPADDR=192.168.1.44 (REPLACE WITH THE STATIC IP ADDRESS)
NETMASK=255.255.255.0 (REPLACE WITH THE APPROPRIATE NETMASK)
DNS2=8.8.8.8 (REPLACE WITH THE APPROPRIATE DNS IP ADDRESS)
DNS1=8.8.4.4 (REPLACE WITH THE APPROPRIATE DNS IP ADDRESS)
## Configure Default Gateway
#
# vi /etc/sysconfig/network
NETWORKING=yes
HOSTNAME=centos6
GATEWAY=192.168.1.1 (ASSIGN APPROPRIATE GATEWAY)
## Restart Network Interface
#
/etc/init.d/network restart
```

Installing ALE 2.0 on a Bare Metal Server

You can also install ALE 2.0 on a bare metal server using the released ISO image. The ISO image includes all the needed software and the operating system. Make sure that the server meets the desired specifications based on the size of network.

To install ALE on a bare metal server:

1. Download the ISO image file and burn the image onto a DVD.
2. Install the DVD into the standard server. Continue with the installation process.
3. After the installation is complete, remove the DVD and restart the system.
4. On the login screen, log in as the admin user with the default admin user ID and password: User ID= root, password = admin

You must change the default password after logging in the first time.

5. Configure the network interface based on your local LAN settings. ALE requires a static IP address.

NTP servers can be added to the ALE server through the WebUI. Once the NTP server is added, ALE syncs with the NTP and sets to the UTC time zone. For information on how to set the server to a different time zone through the shell (not required for ALE), visit CentOS Blog.

The default password for the root user on Linux is **admin**.

Upgrading ALE

If an upgrade file is available for your existing ALE 2.0 instance, you can upgrade to the new version of ALE from the shell.

To upgrade ALE:

1. Login to the shell.
2. Download the upgrade file using SCP or any other file transfer protocol (for example, `ale-2.0.0.0-51047.run`).
3. Once the file is downloaded, run it on the root shell to proceed with the upgrade. The progress is displayed on the terminal, as shown in the following example:

```
[root@ale ~]# sh ale-2.0.0.0-51047.run
Verifying archive integrity... All good.
Uncompressing ALE self-extractable installation...................
INFO: Stopping monit servive...
Stopping monit:                                            [  OK  ]
INFO: Stopping ALE services
Shutdown ale-wstunnel:                                     [  OK  ]
Shutdown ale-jwebapp:                                      [  OK  ]
Shutdown ale-location:                                     [  OK  ]
Shutdown ale-persistence:                                  [  OK  ]
Shutdown ale-platform:                                     [  OK  ]
INFO: Upgrading RPMs
Preparing...                ########################################### [100%]
1:zeromq                 ########################################### [  6%]
2:jzmq                   ########################################### [ 12%]
3:ruby                   ########################################### [ 18%]
4:redis                  ########################################### [ 24%]
5:ruby-bundler           ########################################### [ 29%]
6:rubygems               ########################################### [ 35%]
7:nginx                  ########################################### [ 41%]
8:naocloud               ########################################### [ 47%]
You are replacing the current global value of build.nokogiri, which is currently "--use-
system-libraries"
9:ale-jwebapp            ########################################### [ 53%]
10:ale-persistence        ########################################### [ 59%]
11:ale-location           ########################################### [ 65%]
12:ale-platform           ########################################### [ 71%]
13:ale-utils              ########################################### [ 76%]
14:zeromq-devel           ########################################### [ 82%]
15:zookeeper              ########################################### [ 88%]
16:ale-wstunnel           ########################################### [ 94%]
17:ale-monitconfig        ########################################### [100%]
Shutdown ale-persistence: [  OK  ]
Starting ale-persistence: [  OK  ]
Shutdown ale-jwebapp: [  OK  ]
Starting ale-jwebapp: [  OK  ]
Shutdown ale-platform: [  OK  ]
Starting ale-platform: [  OK  ]
Shutdown ale-location: [  OK  ]
Starting ale-location: [  OK  ]
Stopping ZooKeeper daemon (zookeeper): JMX enabled by default
Using config: /usr/sbin/../etc/zookeeper/zoo.cfg
Stopping zookeeper ... STOPPED
[  OK  ]
Starting ZooKeeper daemon (zookeeper): JMX enabled by default
Using config: /usr/sbin/../etc/zookeeper/zoo.cfg
Starting zookeeper ... STARTED
[  OK  ]
Shutdown ale-wstunnel: [  OK  ]
```

```
Starting ale-wstunnel: [  OK  ]
Shutdown naocloud-web: [  OK  ]
Shutdown naocloud-worker: [  OK  ]
Starting naocloud-web: [  OK  ]
Starting naocloud-worker: [  OK  ]
Stopping Redis server on port 6379: [  OK  ]
Starting Redis server on port 6379: [  OK  ]
INFO: Upgrade succeed
INFO: Starting ALE services
ale-wstunnel already running
ale-jwebapp already running
ale-location already running
ale-persistence already running
ale-platform already running
INFO: Restarting monit servive
Stopping monit:                                            [  OK  ]
Starting monit: Starting monit daemon with http interface at [localhost:2812]
[  OK  ]
INFO: Bye !
```

In some cases, the configuration may not upgrade cleanly. If this occurs, reconfiguration of ALE may be required.

The ALE Setup Wizard

The ALE Setup Wizard allows you to configure and initiate your ALE deployment.

The following steps describe how to setup your deployment through the ALE Setup Wizard:

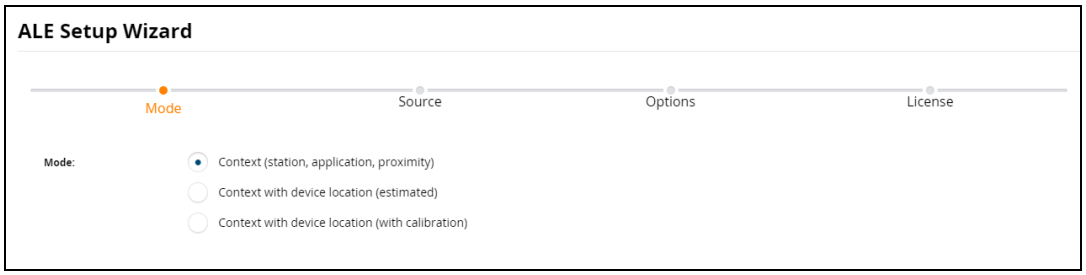
1. Login to the ALE WebUI using your login credentials. Under the default factory settings, the credentials are:
   - **Username**: admin
   - **Password**: welcome123

NOTE

Login credentials can be modified under **Configuration > Admin** in the WebUI.

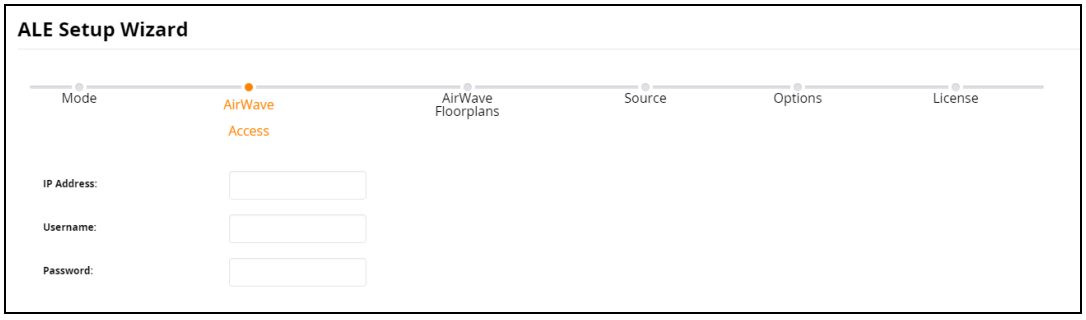
2. Upon entering the setup wizard, the first page that appears is **Mode**. Select the mode for your deployment:
   - **Context (station, application, proximity)**
   - **Context with device location (estimated)**
   - **Context with device location (with calibration)**

Figure 30 *ALE Setup Wizard - Deployment Mode*



3. Click **Next**.
4. Depending on the mode selected, you may be prompted to provide additional information:
   - **Context (station, application, proximity)**: If the **Context** mode is selected, no additional information is required.
   - **Context with device location (estimated)**: If **Context with device location (estimated)** is selected, you must configure the AirWave servers and floorplans used to estimate device location:

Figure 31 *Setup Wizard - AirWave Servers*



a. On the **AirWave Access** page, enter the IP address of the AirWave server.
b. To import floorplans from AirWave, enter the username and password for the AirWave server, or select the check box for **Upload backup from AirWave instead** to locate and select a backup VisualRF file.
c. Click **Next**.
d. Select the required floorplan(s) under the **AirWave Floorplans** page.
e. Click **Next** to continue.

NOTE

Only one AirWave server can be added to the deployment through the setup wizard. To add more AirWave servers, navigate to **Configuration > Mode** in the WebUI.

- **Context with device location (with calibration)**: If **Context with device location (with calibration)** is selected, no additional information is required.

5. After configuring the deployment mode, specify the datasource under the **Source** page:
   - To configure a controller deployment:
     a. Under **Source**, select **Controller**.
     b. Select the desired controller(s) from the **Controllers** table.
     c. To add a new controller, click the **+** button on the bottom left corner of the **Controllers** table. Enter the IP address, username, and password for controller on the **New Controller** popup window. Click **Add** to add the new controller.
     d. If your deployment requires an RTLS feed, select **Enable RTLS Feed**, and enter the port number and secret code in the corresponding text boxes.

NOTE

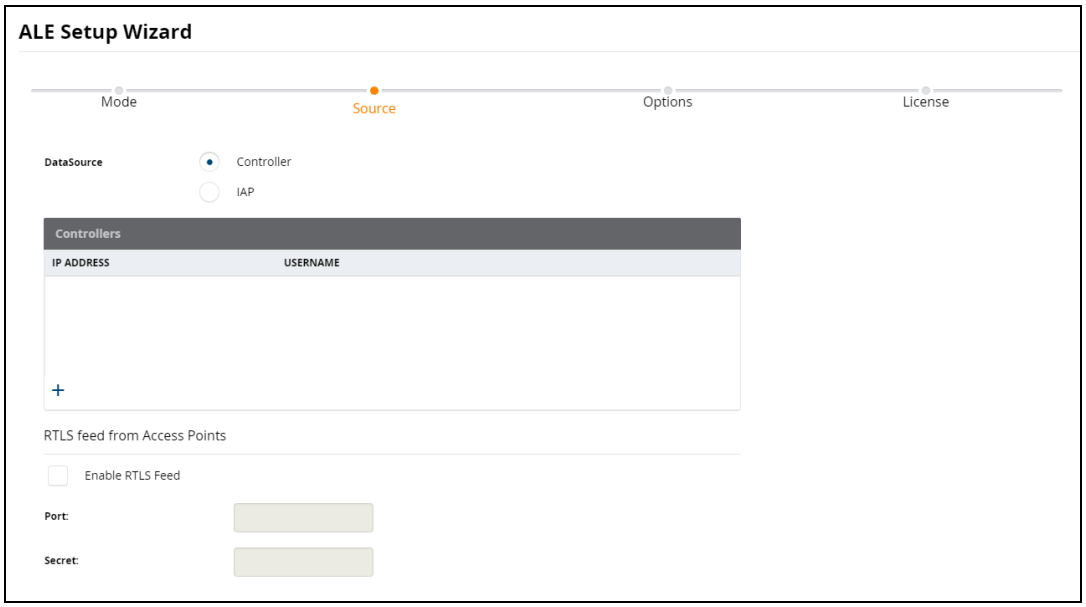
The RTLS secret code and port number must be the same as configured on the controller.

e. Click **Next**.

NOTE

Up to 10 controllers can be configured for a deployment.

Figure 32 *ALE Setup Wizard - Controller Deployment*



- To configure an IAP deployment:
  a. Under **Source**, select **IAP**.
  b. Enter the port number in the corresponding text box.

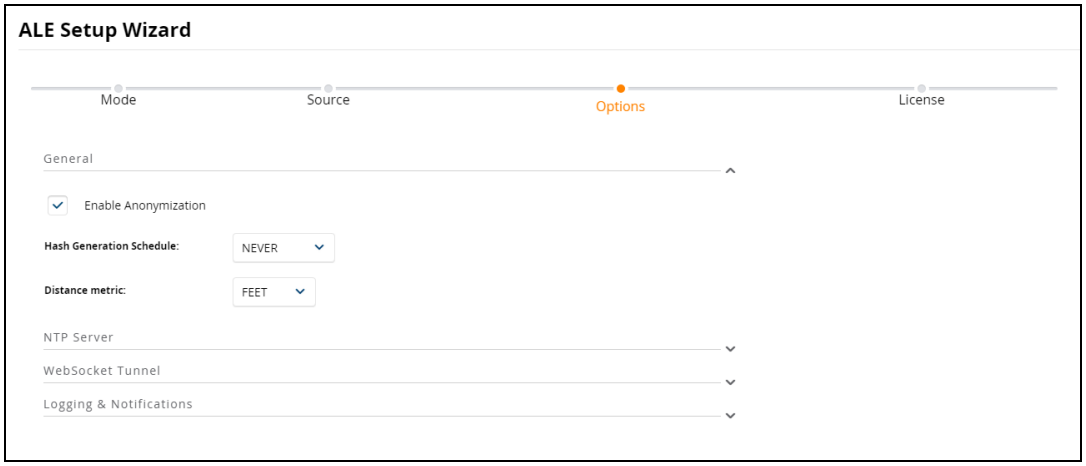
The port number must be the same as configured on the IAP to send ALE data.

  c. Click **Next**.

Figure 33 *ALE Setup Wizard - IAP Deployment*

6. (Optional) Once the datasource has been specified, configure additional settings under the **Options** page:
   - **General**: The **General** tab contains options for general settings.
     a. To enable anonymization, select the check box for **Enable Anonymization**.
     b. Set the **Hash Generation Schedule** through the corresponding drop-down list (never, daily, weekly, monthly).
     c. Specify the **Distance metric** through the corresponding drop-down list (feet or meter)

Figure 34 *ALE Setup Wizard - General Settings*

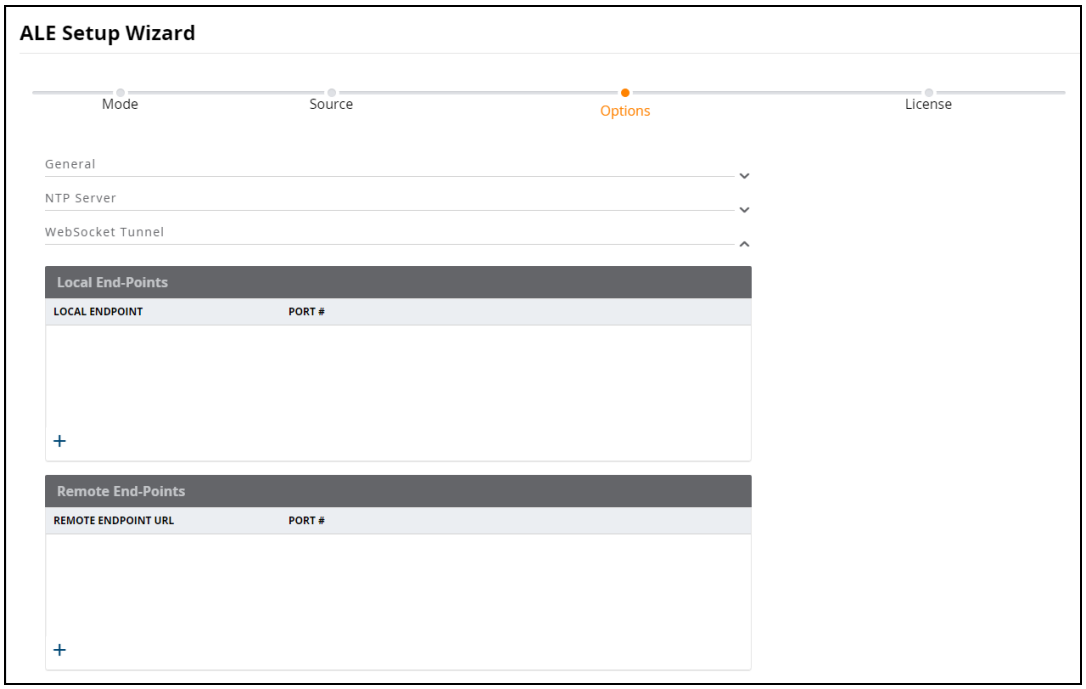


   - **NTP Server**: The **NTP Server** tab allows you to add an NTP server to synchronize all system clocks.
     a. To add an NTP server, enter the IP address of the server in the corresponding text box.
     b. If an NTP server is not added, ALE attempts to synchronize with default NTP servers.

Figure 35 *ALE Setup Wizard - NTP Server*

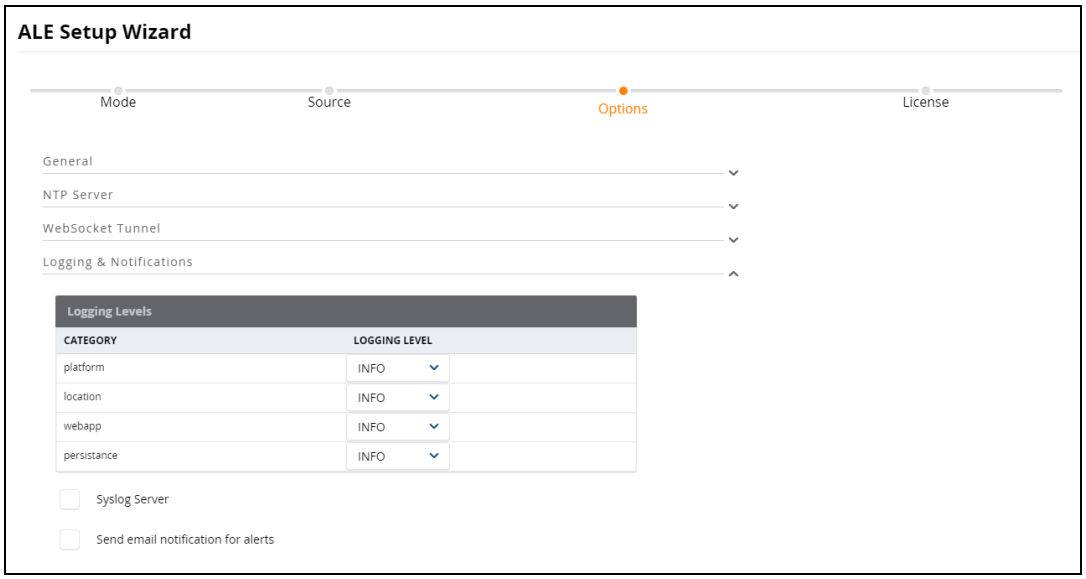
- **WebSocket Tunnel**: The **WebSocket Tunnel** tab allows you to add tunnels for secure communication between local end-points and remote end-points. For more details on WebSocket Tunnel, refer to Configuring WebSocket Tunnel.
  a. Select the desired end-points from the **Local End-Points** table and **Remote End-Points** table.
  b. To add a new end-point, click the **+** button on the bottom left corner of the table. Enter the name and port number of the new end-point when the end-point popup window appears. Click **Add** to add the new end-point.

Figure 36 *Setup Wizard - WebSocket Tunnel*



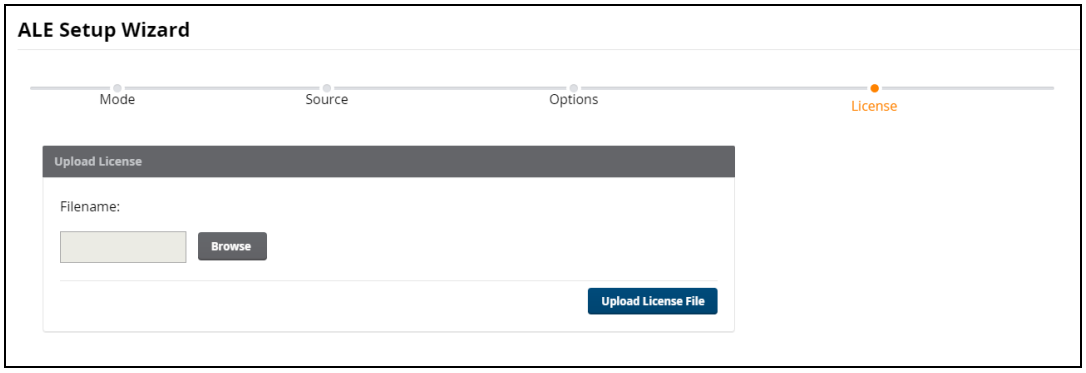
- **Logging & Notifications**: The **Logging & Notifications** tab allows you to set logging levels and notifications for ALE component log files.
  a. Designate the logging level type for each component (error, warning, info, debug, trace).
  b. If a syslog server is required, select the check box for **Syslog Server** and enter the server IP address.
  c. If email notifications are desired for critical events (for example, process crashes), select the check box for **Send email notifications for alerts** and provide an email address.

Figure 37 *ALE Setup Wizard - Logging & Notifications*



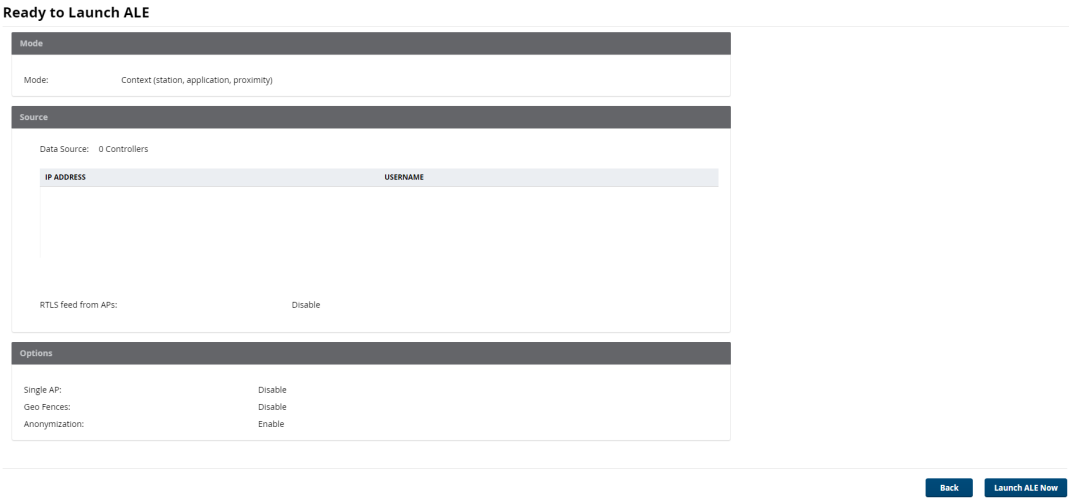
7. After all optional settings are configured, click **Next**.
8. The last page of the ALE Setup Wizard is **License**:
   a. Under **Upload License**, click **Browse** to search for your current ALE license. Windows Explorer opens. See ALE Licenses for more information on the different types of licenses available.
   b. Locate and select your license, and then click **Open**. The license name appears in the **Filename** textbox.
   c. Click **Upload License File** to upload the license. After setup is completed, the current licensing status is displayed on the ALE dashboard. Licenses can be managed through the **Maintenance** page, but ZMQ streaming is unavailable until a valid license is loaded.

Figure 38 *ALE Setup Wizard - Licenses*



9. Click **Finish** to complete deployment setup.
10. To launch ALE, verify all deployment settings under **Ready to Launch ALE**.
    - If any changes must be made, click **Back** to navigate back to any previous pages.
    - If everything is correct, click **Launch ALE Now** to launch ALE and access the ALE dashboard.

Figure 39 *ALE Setup Wizard - Launch ALE*



ALE can also be configured and modified through the **Configuration** tab of the WebUI.

To bypass the ALE Setup Wizard:

1. Login to the ALE WebUI using your login credentials. The ALE Setup Wizard opens.
2. Click **Next** on each page of the setup wizard without changing any configuration options until you reach the **Finish** button.
3. Click **Finish**, and then click **Launch ALE Now** to launch ALE and access the dashboard.
4. Once ALE is launched, navigate to **Configuration** to setup or modify your deployment. See Configuring ALE for more details.

Configuring ALE

ALE deployments can be setup, and existing deployments can be modified under the **Configuration** tab of the WebUI.

Configuring the Deployment Mode

ALE 2.0 runs in three modes: context mode, context mode with device location (estimated), and context mode with device location (calibration).

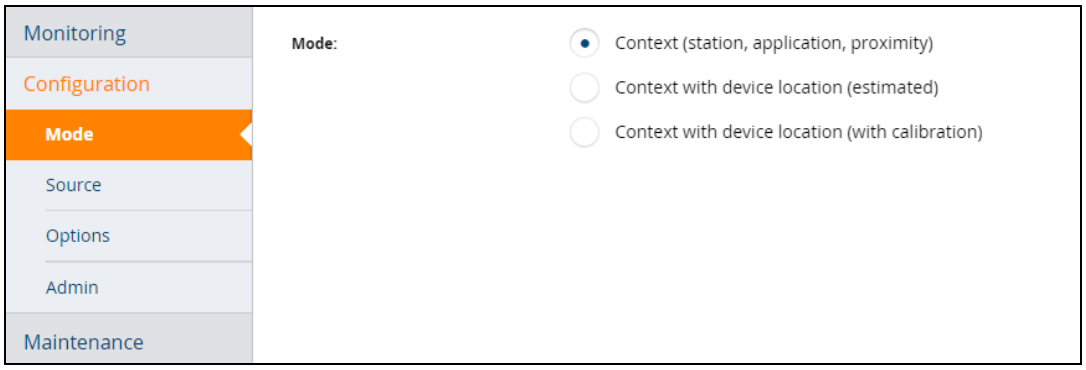
Context (without Maps or Locations)

Context mode (without maps or locations) does not calculate or report client location, simplifying deployment and improving scalability. All context topics from the NBAPI are reported, with the exception of Campus, Building, Floor, Location, and Geo_Fence. This context mode uses Proximity to determine which access point is closest to the wireless client, providing a rough location estimation. Refer to the *Analytics and Location Engine 2.0 API Guide* for more details on the ALE REST and ZMQ APIs.

To configure ALE in the context mode without maps or locations:

1. Navigate to **Configuration > Mode** in the WebUI.
2. Under **Mode**, select **Context (station, application, proximity)**.

Figure 40 *Configuring Context Mode (Station, Application, Proximity)*



3. Click **Apply** to save your configuration.

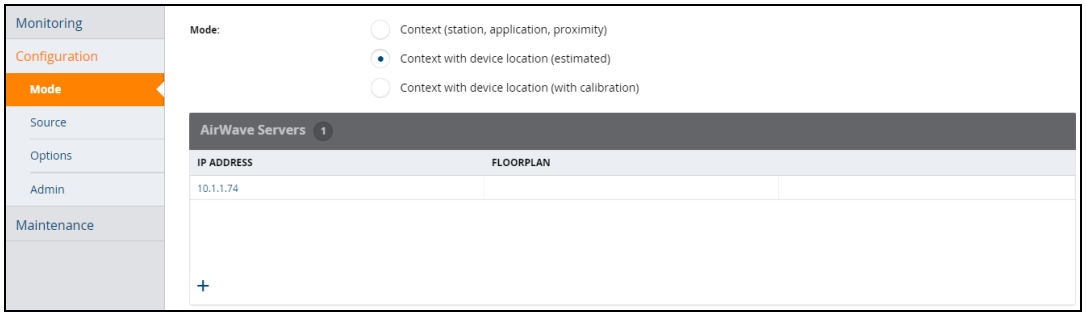
Context with Device Location (Estimated)

Context with device location (estimated) runs an import wizard to extract floorplans, AP placement data, and GeoFence region data from AirWave servers. This extracted information is combined with AP-AP RSSI data from the datasource to generate a Positioning Database (PDB). Device location is calculated and reported under this context mode.

To configure ALE in the context mode with estimated location:

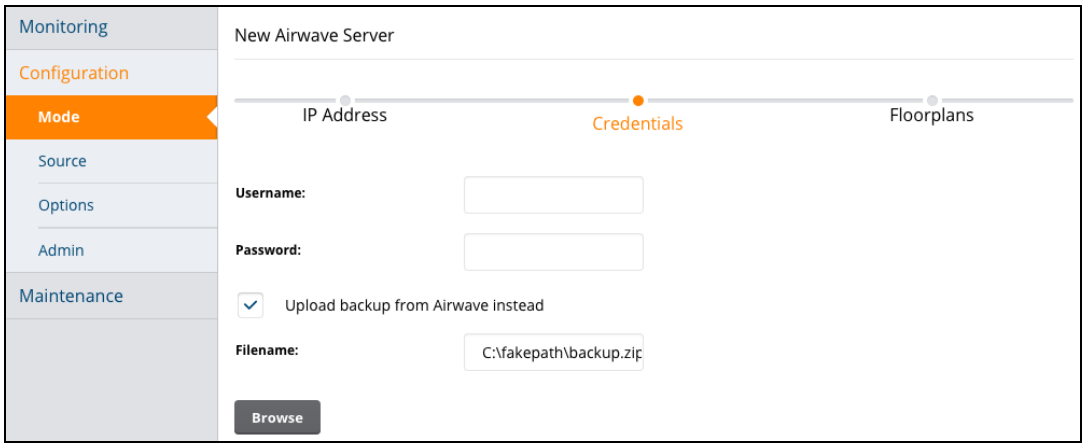
1. Navigate to **Configuration > Mode** in the WebUI.
2. Under **Mode**, select **Context with device location (estimated)**.

Figure 41 *Configuring Context Mode with Device Location (Estimated)*



3. Select the desired AirWave server(s) from the **AirWave Servers** table.
    1. To add a new AirWave server, click the **+** button on the bottom left corner of the **AirWave Servers** table. The **New AirWave Server** page opens.
    2. Enter the server IP address in the **IP Address** text box, and then click **Next**.
    3. Import floorplans from AirWave using one of the following methods:
        a. Enter the server credentials (username and password), and then click **Next**.
        b. Upload a backup VisualRF file from AirWave by selecting the check box for **Upload backup from AirWave instead**. Click **Browse** to locate and select the backup file, and then click **Next**.

Figure 42 *Importing Floorplans from AirWave*



4. Select the floorplan(s) required for data extraction (all floors, or a subset of floors). Click **Next** to pull relevant data from the AirWave server and store it in the ALE database.

NOTE

Once this information is stored in the ALE database, the AirWave servers do not need to be contacted anymore. This is true even if the ALE services are restarted.

5. Click **Apply** to save your configuration.

NOTE

Multiple AirWave servers can be added to a deployment.

4. Click **Apply** to save your configuration.

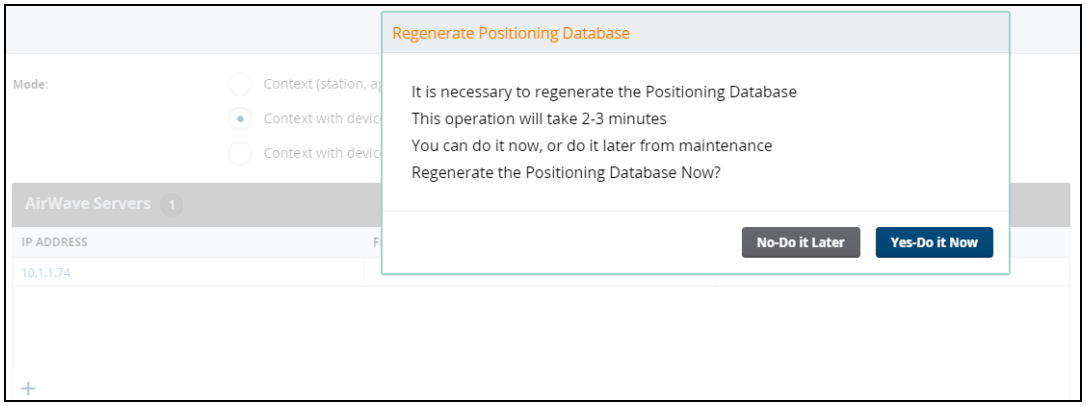
You may be prompted to regenerate the Positioning Database (PDB). Select **Yes-Do it Now** to regenerate the PDB immediately, or **No-Do it Later** to regenerate the PDB at a later time through the **Maintenance** tab.

AP-AP RSSI information from the datasource is combined with AP placement information extracted from AirWave to generate the PDB. The PDB is essential for the location algorithm to calculate device location.

NOTE

If PDB generation is deferred or fails, ALE cannot calculate device location. Campus, Building, Floor, Location, and Geo_Fence APIs are not published, and the unassociated clients dashlet may not show any data.

Figure 43 *Pop-up Window to Regenerate the PDB*



Context with Device Location (Calibration)

Context with device location (calibration) uses advanced fingerprinting techniques for improved location accuracy.

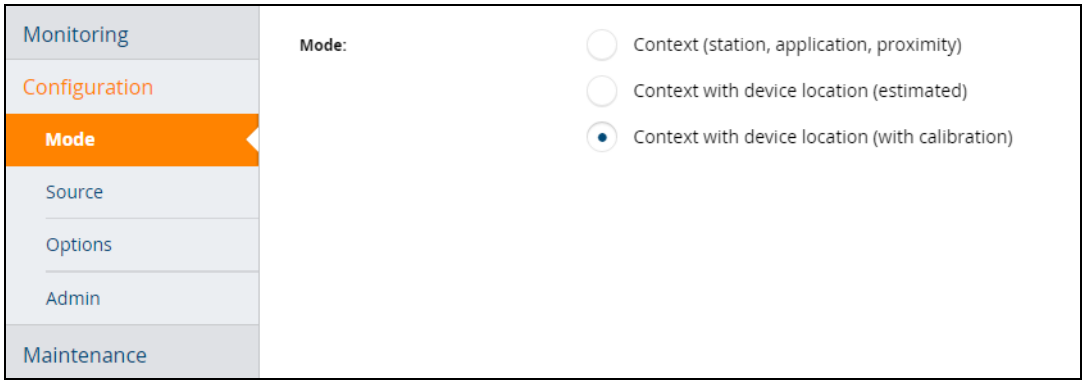
NOTE

Fingerprinting is described in detail in The Aruba Nao Campus Calibration Tool.

To configure ALE in the context mode with calibrated location (fingerprinting):

1. Navigate to **Configuration > Mode** in the WebUI.
2. Under **Mode**, select **Context with device location (with calibration)**.

Figure 44 *Configuring Context Mode with Device Location (with Calibration)*



3. Click **Apply** to save your configuration.

Under context mode with calibrated location, ALE works hand-in-hand with an application called Aruba Nao Campus. Aruba Nao Campus allows network administrators to upload site floorplans and prepare the site for fingerprinting. Aruba Nao Campus is paired with an Android application called Aruba Nao Logger to fingerprint each site using intelligent sensors (MEMS) on the Android device.

After the fingerprints are collected, Aruba Nao Campus uses these measurements to generate a Positioning Database (PDB). When deployed in this context mode, ALE knows where to search for the available PDB. If any changes are made to the Aruba Nao Campus tool, such as a newly drawn GeoFence region, navigate to the **Maintenance** tab of the WebUI and select **Regenerate PDB** to obtain an updated version of the PDB. For more details on how to configure Aruba Nao Campus, see The Aruba Nao Campus Calibration Tool.

Configuring the Data Source

ALE can be configured for either controller or IAP deployments.

Controller Deployment

In a controller-based WLAN, both the controller and ALE must be configured to communicate with each other. ALE performs an HTTP GET of AMON data to retrieve already available information from the controller. Any further information is received as an AMON push from the controller.

The RTLS feed from access points in a controller-based deployment can be used as an alternate source of RSSI data for location computation. If the RTLS feed is enabled on the deployment, RSSI data is retrieved from the RTLS feed instead of AMON. Other context information is sourced from AMON. Buffering occurs on each access point, lowering the latency of the RTLS feed. When combined with a high density of AM mode access points, the RTLS feed can provide a richer source of RSSI data to improve accuracy and latency during location computation.

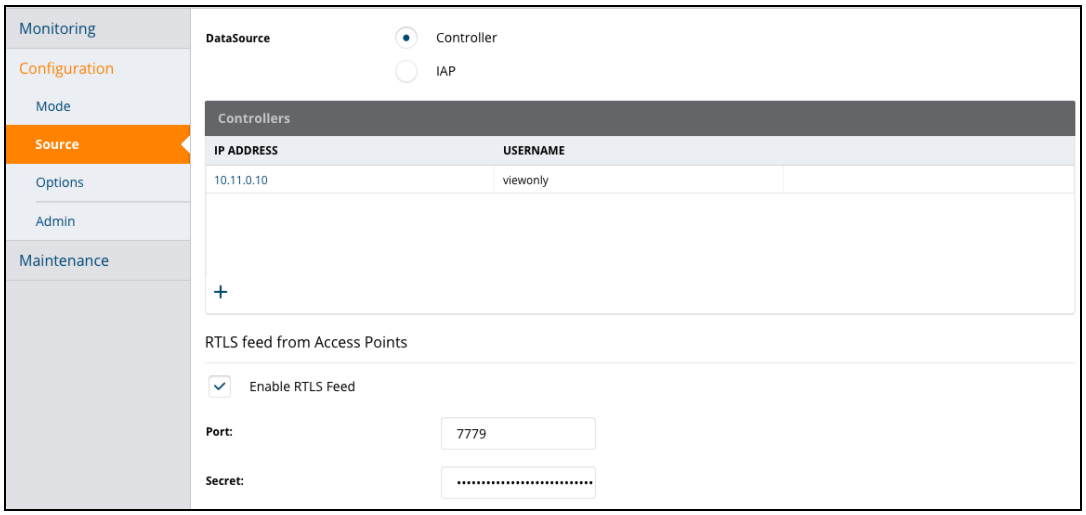
NOTE

Up to 10 controllers can be configured for a deployment.

To configure a controller:

1. Navigate to **Configuration > Source** in the WebUI.
2. Under **Source**, select **Controller**.
3. Select the desired controller(s) from the **Controllers** table.

Figure 45 *Configuring a Controller*



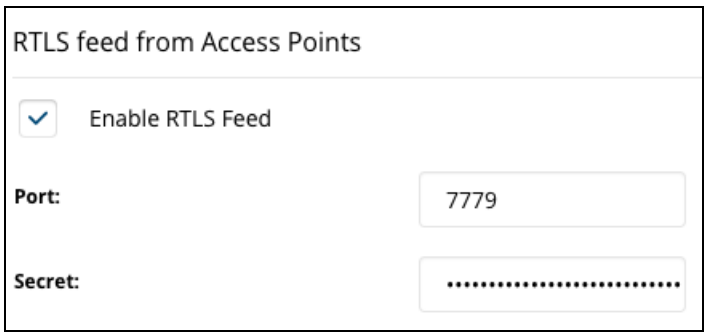
1. To add a new controller, click the **+** button on the bottom left corner of the **Controllers** table. A **New Controller** popup window appears.
2. Enter the IP address of the controller and create a unique username and password in the corresponding text boxes.
3. Click **Add** to add the new controller.

4. If your deployment requires an RTLS feed, select **Enable RTLS Feed** under **RTLS feed from Access Points**.
   1. Enter the port number and secret code in the corresponding text boxes.

NOTE

The RTLS secret code and port number must be the same as configured on the controller.

Figure 46 *Enabling RTLS*



5. Click **Apply** to save your configuration.

IAP Deployment

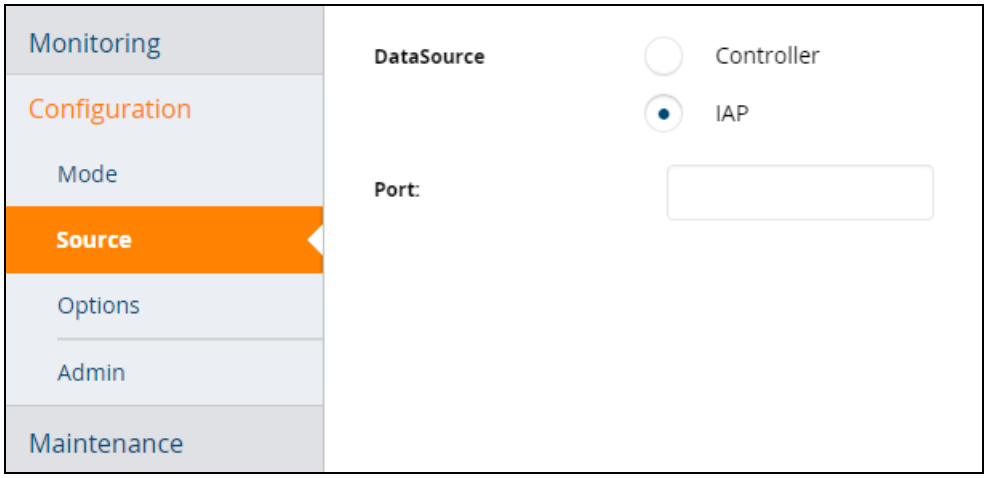
ALE supports integration with IAP. The IAP sends client information and other status information to the ALE server.

To configure an IAP deployment:

1. Navigate to **Configuration > Source** in the WebUI.
2. Under the **DataSource**, select **IAP**.
3. Enter the port number in the corresponding text box.

The port number must be the same as configured on the IAP to send ALE data. IAP-ALE communication is secured in an HTTPS session.

Figure 47 *Configuring an IAP*



4. Click **Apply** to save your configuration.

Configuring General Settings

Under the **Options** page of the WebUI, the **General** tab contains settings to enable or disable optional features.

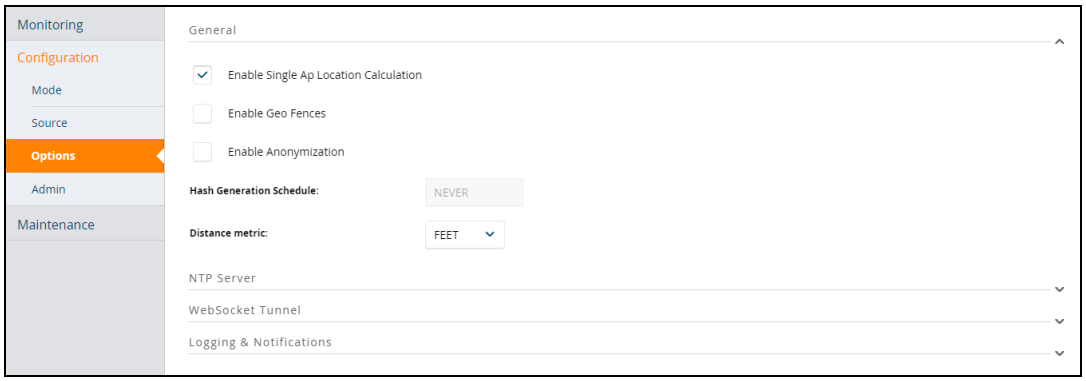
Single AP Location Calculation

If AP density is insufficient but location information is still desired, Single AP Location can be enabled to calculate the rough location of a client.

To enable single AP location calculation:

1. Navigate to **Configuration > Options** in the WebUI.
2. Select the check box to **Enable Single AP Location Calculation**.

Figure 48 *Enabling Single AP Location Calculation*



3. Click **Apply** to save your configuration.

This feature activates the low density location estimation algorithm. Locations calculated by this algorithm are indicated by ALE (refer to the *Analytics and Location Engine 2.0 API Guide* for more information). If a sufficient number of access points report the client RSSI, locations calculated by the regular location algorithm are also reported by ALE.

GeoFences

GeoFencing allows a network administrator to designate regions and leverage client location information to monitor client traffic through those designated regions.

A GeoFence region is defined on the site map using either AirWave or Aruba Nao Campus.

To create a GeoFence region using AirWave:

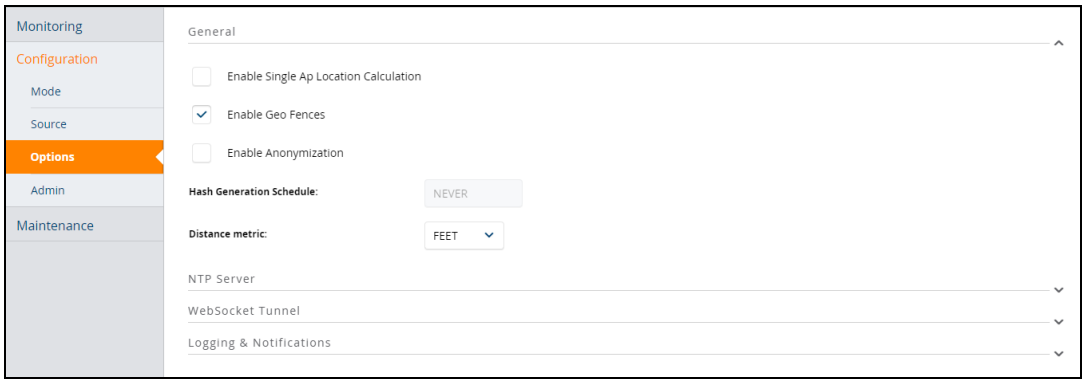
1. Login to your AirWave server.
2. Click on the **VisualRF** tab and select the campus.
3. Select the building and floor.
4. Once the floor map is displayed, click on the **Edit** tab.
5. Select the **Draw Region** tool to create the GeoFence region.
   a. Enter a name for the newly created region.
   b. Select the **Region Type**.
   c. Complete the configuration based on the region type.

For information on how to define a GeoFence region through Aruba Nao Campus, see The Aruba Nao Campus Calibration Tool.

To enable GeoFences:

1. Navigate to **Configuration > Options** in the WebUI.
2. Select the check box for **Enable Geo Fences** to send notifications when a client enters or exits a GeoFence region. Refer to the *Analytics and Location Engine 2.0 API Guide* for more information on the Geo_Fence API.

Figure 49 *Enabling GeoFences*



3. Click **Apply** to save your configuration.

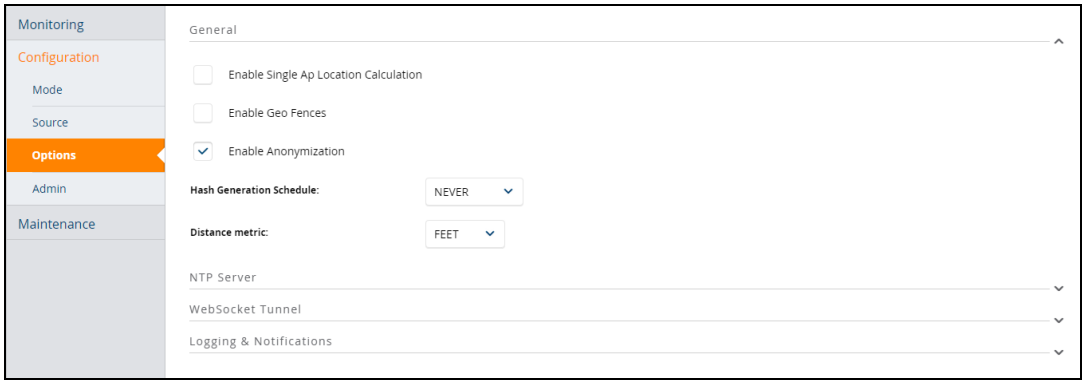
Anonymization

Direct use and sharing of a user's personal identification information raises privacy concerns within ALE. Certain data fields of the AMON message feed require personal network and mobile device identification information, which is shared with outside applications (for example, device MAC addresses, usernames, and IP addresses). This data can be anonymized to protect clients and prevent any sensitive information from being released.

To enable anonymization:

1. Navigate to **Configuration > Options** in the WebUI.
2. Select the check box to **Enable Anonymization**.

Figure 50 *Enabling Anonymization*



3. Select a **Hash Generation Schedule** from the drop-down list.
4. Click **Apply** to save your configuration.

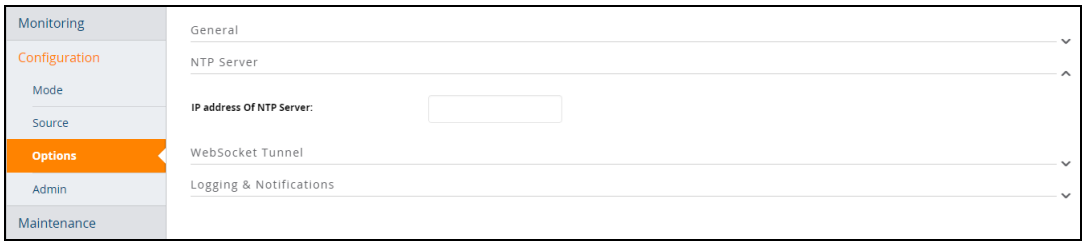
Configuring the NTP Server

The Network Time Protocol (NTP) Server is an Internet protocol server that uses data networks to synchronize computer system clocks to a time reference.

To configure the NTP Server:

1. Navigate to **Configuration > Options** in the WebUI.
2. Open the **NTP Server** tab.
3. Enter the IP address of the NTP Server in the corresponding text box.

Figure 51 *Configuring the NTP Server*



4. Click **Apply** to save your configuration.

If an NTP server is not added, ALE attempts to synchronize with default NTP servers.

Setting Logging Levels

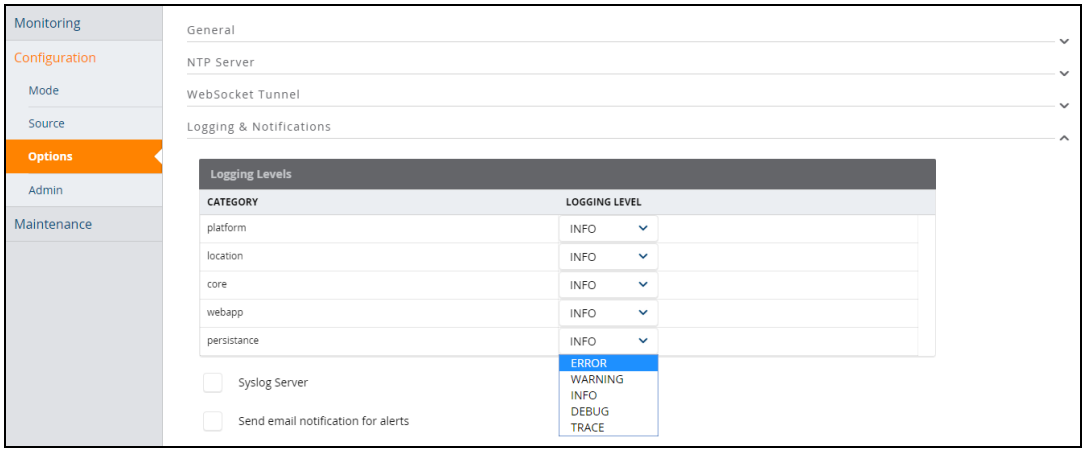
Log files are available for each of the major ALE components:

- ale-jwebapps
- ale-location
- ale-platform
- ale-persistance

To set logging levels for each ALE component:

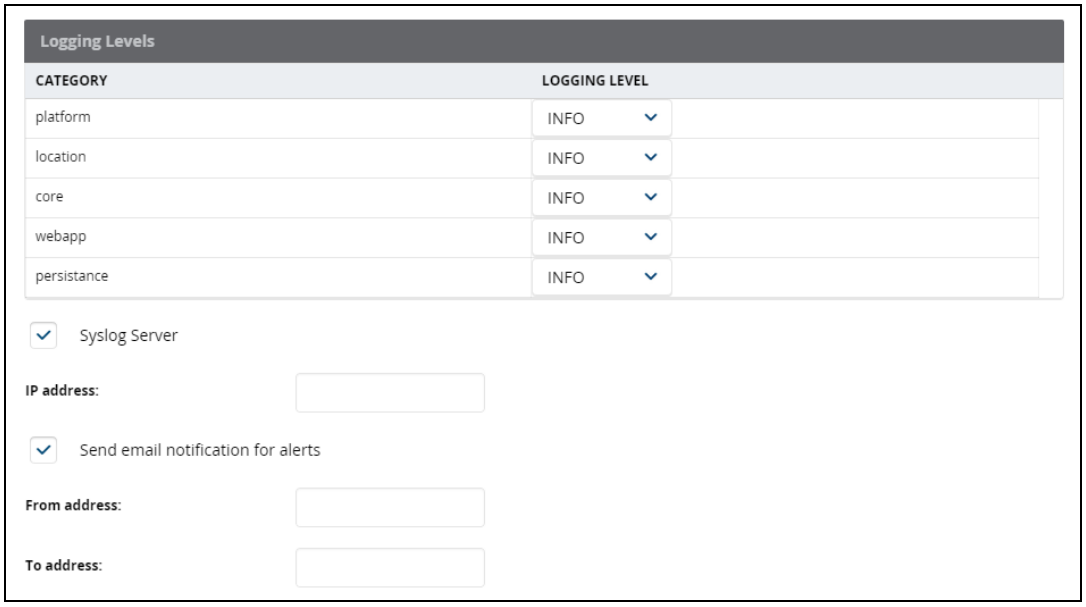
1. Navigate to **Configuration > Options** in the WebUI.
2. Open the **Logging & Notifications** tab.
3. Under **Logging Level** in the **Logging Levels** table, designate the logging level type for each component.

Figure 52 *Logging Levels and Notifications*



4. If these log messages must be sent to an external syslog server, a syslog server can be configured:
   a. Select the check box for **Syslog Server** to add a new server.
   b. Enter the syslog server IP address.
5. Select the check box for **Send email notification for alerts** and provide an email address if email notifications are desired for critical events (for example, process crashes).

Figure 53 *Enabling Syslog Server and Email Notifications*



6. Click **Apply** to save your configuration.

Tech support logs can be found under **Maintenance > Download Tech Support Logs** in the WebUI. Click **Download Tech Support Log**s to download the logs as a ZIP file.

Adding Management Users

Users can be granted management access to the ALE WebUI. Similarly, users can be granted access to the REST API via HTTP authentication.

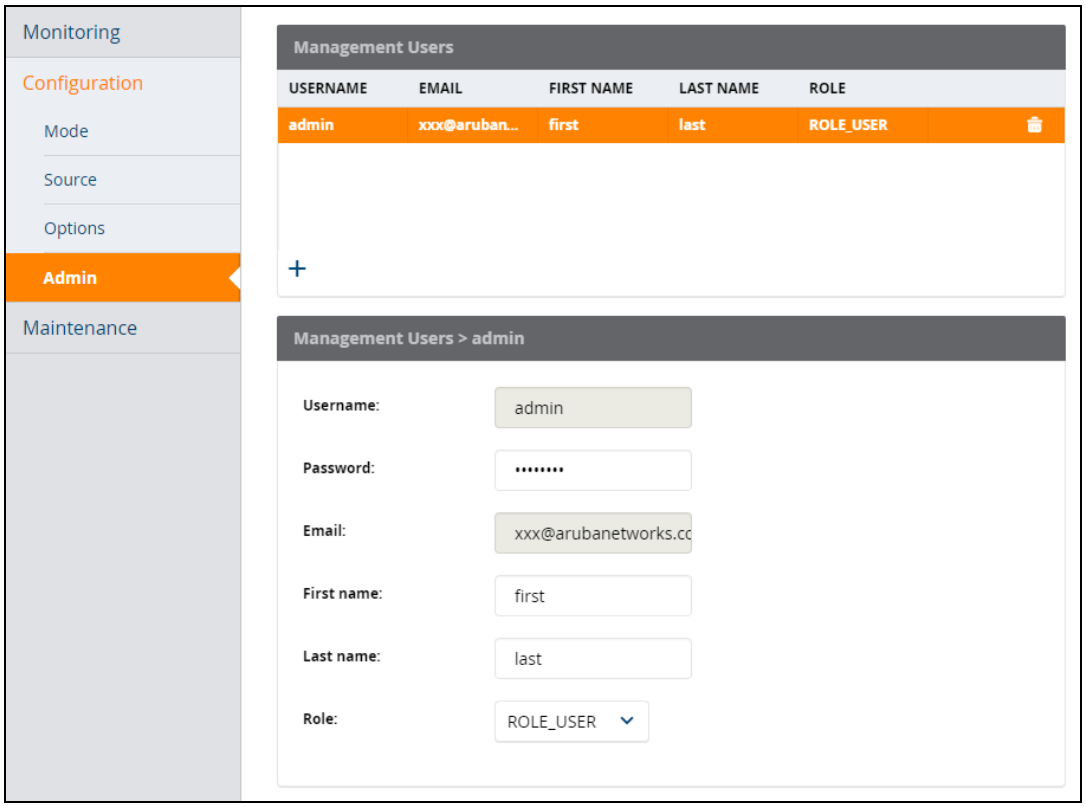
NOTE

Currently, only one user role can be granted both UI management and REST API access rights.

To add management users:

1. Navigate to **Configuration > Admin** in the WebUI.
2. Select users from the **Management Users** list. When a user is selected, the user profile information is displayed below under **Management User > admin**.

Figure 54 *Adding Management Users*



a. To add a new management user, click the **+** button on the bottom left corner of the **Management Users** list.
b. A pop-up window for **Management User > New** appears.
c. Fill in the fields for **Username**, **Password**, **Email**, **First name**, and **Last name**.
d. Select the user role from the **Role** drop-down list.
e. Click **Add** to add the new management user.

3. Click **Apply** to save your configuration.

Custom Certificates for HTTPS

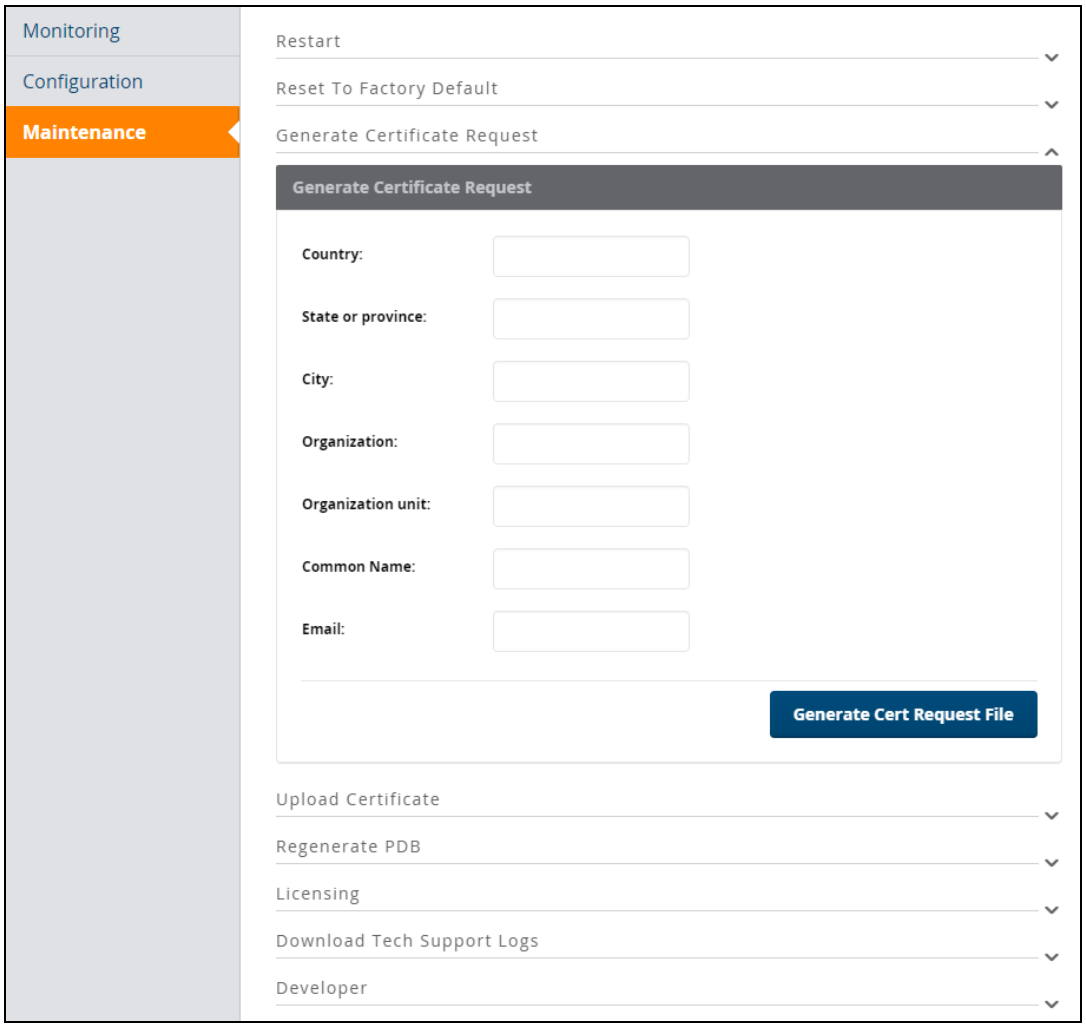
When ALE is installed, self-signed certificates are automatically created and used for HTTPS in the user interface and REST API. To install your own server certificates, generate and upload your certificates manually through the **Maintenance** tab of the WebUI.

To generate a certificate request:

1. Navigate to **Maintenance > Generate Certificate Request** in the WebUI.
2. Fill out the **Generate Certificate Request** form.
3. Click **Generate Cert Request File**. The certificate and key files are returned in a Zip file.

To sign the certificate request, send the certificate (.cer) in a Zip file to a trusted authority. The private key file (.key) should be stored in a secure location, as it is required to upload the custom certificate. Once the trusted certificate is returned, the appropriate certificate chain must be downloaded from the trusted authority (for NGINX), and then the custom certificate can be uploaded.

Figure 55 *Generating a Certificate Request*

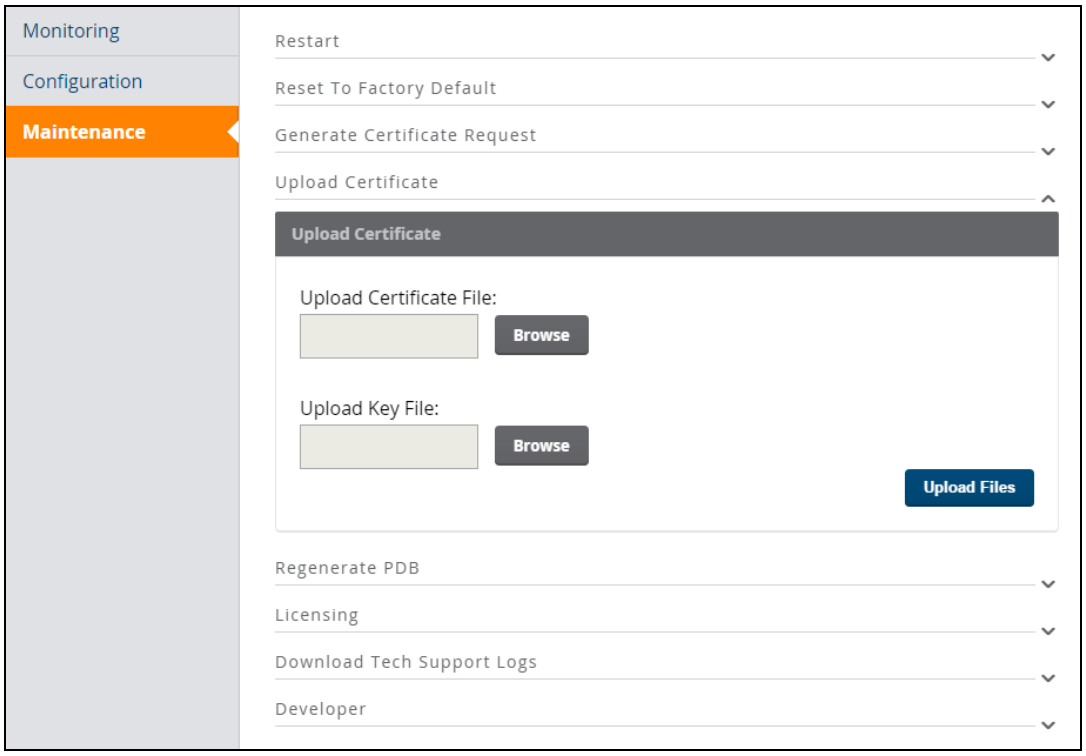


To upload a certificate:

1. Navigate to **Maintenance > Upload Certificate** in the WebUI.
2. Under **Upload Certificate File**, click **Browse**. The file manager opens.
3. Locate and select the correct certificate file, and then click **Open**. The certificate name appears in the **Upload Certificate File** text box.
4. Under **Upload Key File**, click **Browse**. The file manager opens.
5. Locate and select the corresponding key file, and then click **Open**. The key name appears in the **Upload Key File** text box.
6. Click **Upload Files** to upload the certificate and key files.

When you acquire a certificate from a signed authority, make sure you download the appropriate certificate for the NGINX web server.

Figure 56 *Uploading a Certificate*



ALE Licenses

ALE 2.0 operates under three licensing schemes:

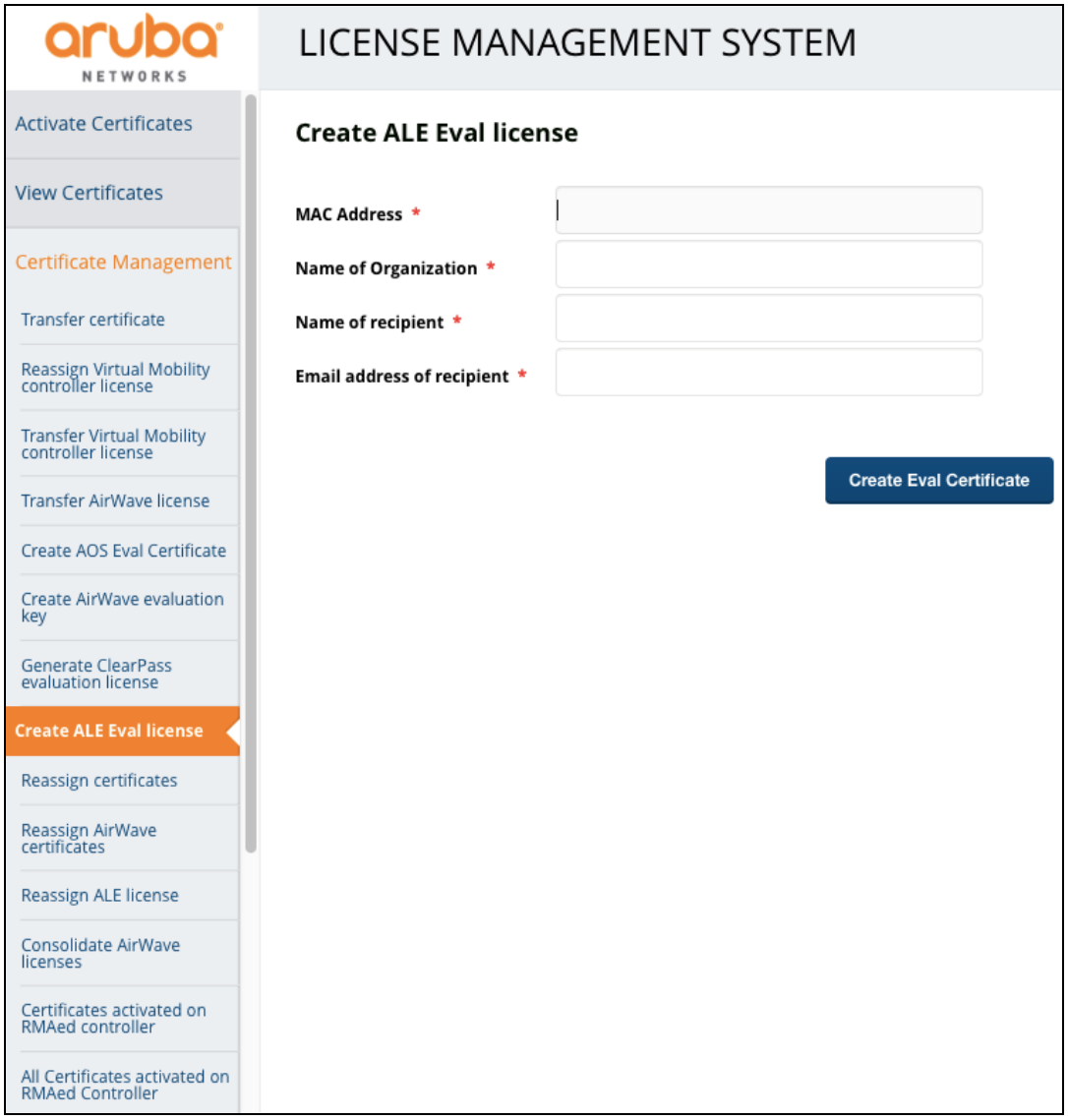
- **Unlicensed**: If ALE is unlicensed, it does not publish any ZMQ messages.
- **Evaluation License**: An evaluation license contains an AP count and expiry period. Once the license expires, ALE switches to the unlicensed state and stops publishing ZMQ messages.
- **Permanent License**: A permanent license contains an AP count. The number of APs that can run on a permanent license is determined by the AP count. Permanent licenses are additive, and the current licensed AP count can be viewed on the ALE dashboard.

Generating a License

To generate an evaluation license:

1. Login to the Aruba Licensing site at http://licensing.arubanetworks.com.
2. Navigate to **Certificate Management > Create ALE Eval License**.
3. Enter the MAC address of your ALE server.
4. Enter the name of your organization.
5. Provide the name and email address of the license recipient.
6. Click **Create Eval Certificate**. The evaluation license is sent as an email attachment to the email address listed on the **Create ALE Eval License** form.

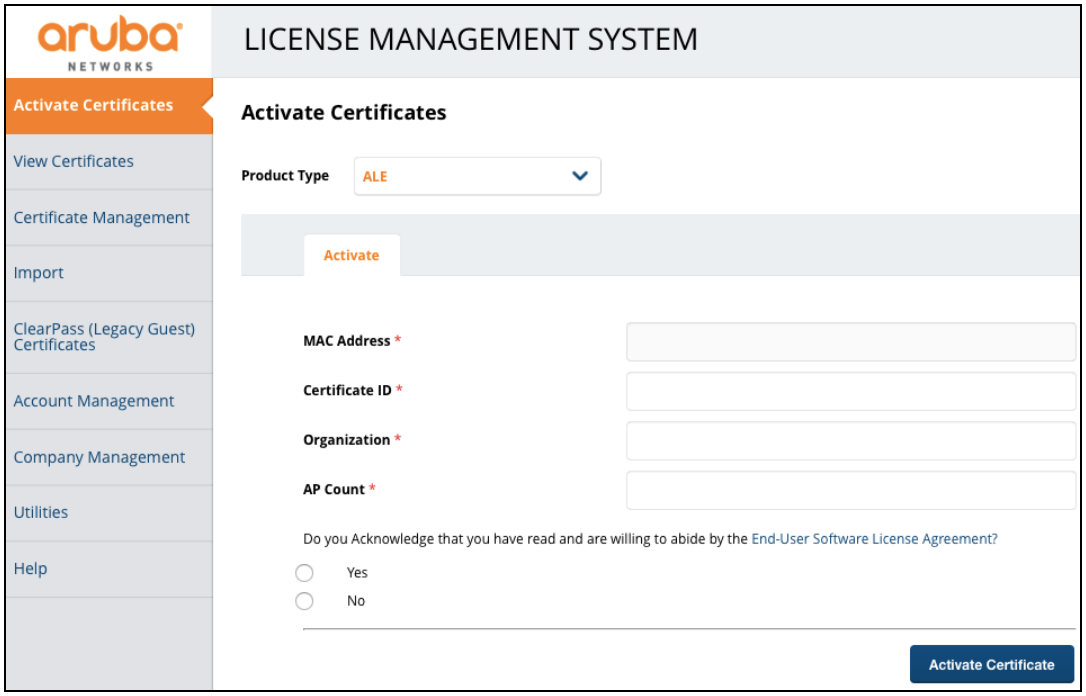
Figure 57 *Generating an Evaluation License*



To generate a permanent license:

1. Login to the Aruba Licensing site at http://licensing.arubanetworks.com.
2. Navigate to **Activate Certificates**.
3. Select **ALE** from the **Product Type** drop-down list.
4. Enter the MAC address of your ALE server.
5. Provide the ID number of your ALE shipping certificate.
6. Enter the name of your organization.
7. Specify the number of APs in your network under **AP Count**.
8. Select **Yes** to acknowledge and comply with the **End-User Software License Agreement**.
9. Click **Activate Certificate**.

Figure 58 *Generating a Permanent License*

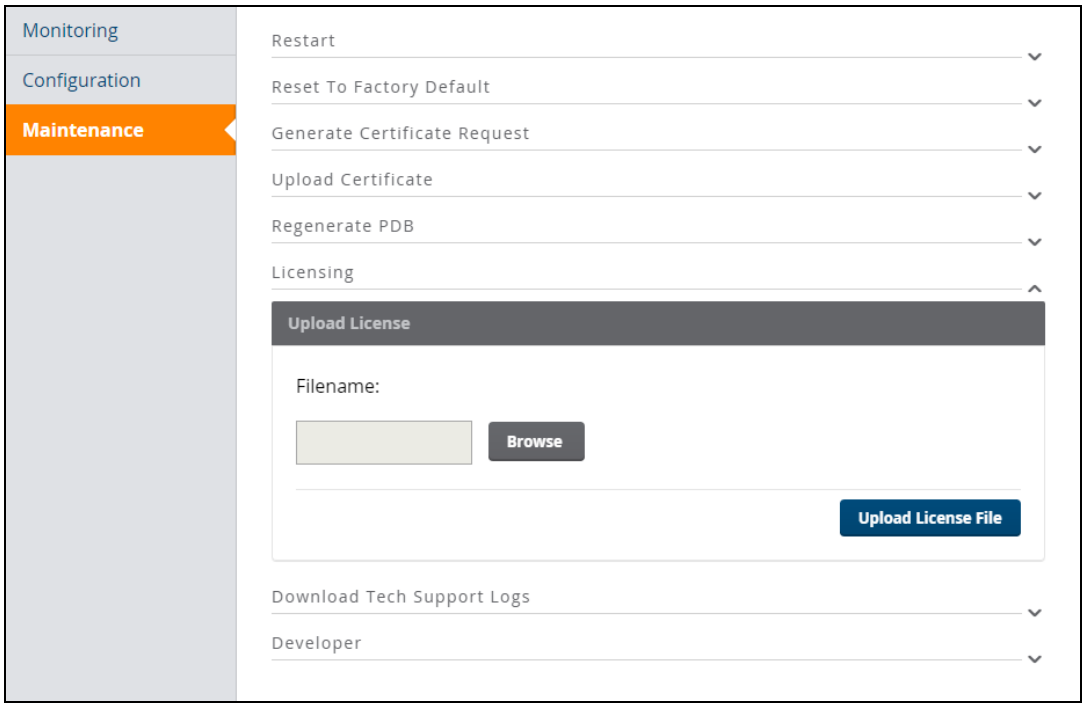


Uploading a License

To upload a new evaluation or permanent license:

1. Navigate to **Maintenance > Licensing** in the WebUI.
2. Under **Upload License**, click **Browse** to search for your current ALE license. The file manager opens.
3. Locate and select your license, and then click **Open**. The license name appears in the **Filename** text box.
4. Click **Upload License File** to upload and begin using the license.

Figure 59 *Uploading a License*



The Aruba Nao Campus Calibration Tool

Aruba Nao Campus and Aruba Nao Logger are calibration tools that enrich the location engine with calibration data (fingerprinting) for improved location accuracy. The web-service Aruba Nao Campus tool and companion Aruba Nao Logger Android application fingerprint sites and generate a Positioning Database (PDB). The Aruba Nao Campus web-service is hosted by ALE, and the Positioning Database is stored in a local database. The PDB is made available for the ALE location engine to compute device location. These tools are required only when ALE is deployed under **Context Mode with Device Location (Calibration)**.

The Calibration Workflow Overview

The following provides an overview on the Aruba Nao Campus tool. Before you begin, check for the following:

- The floorplan you plan on calibrating must be readily accessible (JPEG or PNG).
- Download and install the Aruba Nao Logger app on your Android device. Android version 5.0 and above is recommended. Nexus and Samsung devices work best with this tool.
- If multiple operators calibrate simultaneously, they must all carry the same model of the Android device.

1. Access the Aruba Nao Campus webpage at **https://<ALE IP address>:4000**.
2. Login using the same credentials used to access the ALE WebUI.
3. Create campuses.
4. Create buildings by geo-referencing them and tying them to a campus.
5. Load floorplans for each building.
6. Define paths (also called graphs) on each floorplan.
7. Install the Aruba Nao Logger Android application on an Android device. Point the application to the ALE server, and then complete fingerprinting. Once you are satisfied with your fingerprints, publish a production PDB by selecting the **API Keys** link on the Nao Campus home page.
8. After fingerprinting is completed, set ALE to **Context Mode with Device Location (Calibration)** or use the **Regenerate PDB** feature in the WebUI to allow ALE to download the published PDB from Aruba Nao Campus.

Creating a Campus

To create a new campus:

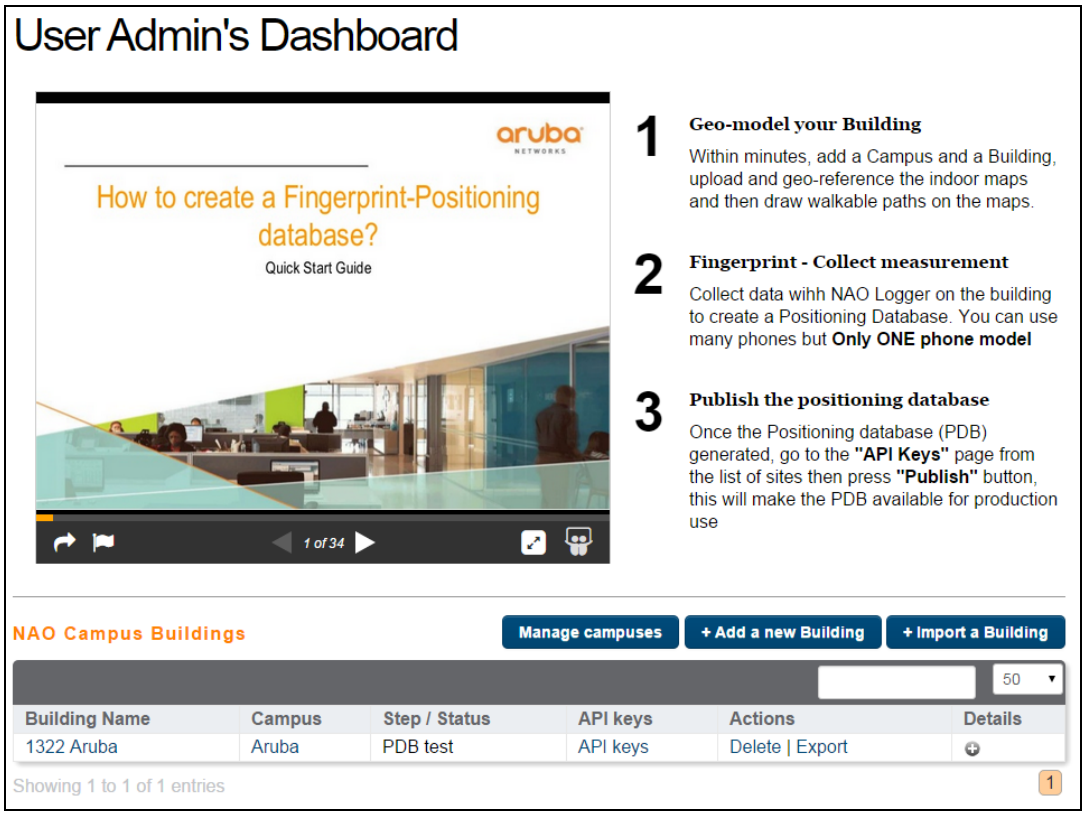
1. Access the Aruba Nao Campus dashboard.
2. Click **Manage campuses** under **NAO Campus Buildings**.
3. Click **New Campus**.
4. Enter a name and description for the campus, and then click **Save**.

Creating a Building

To add a new Aruba Nao Campus building:

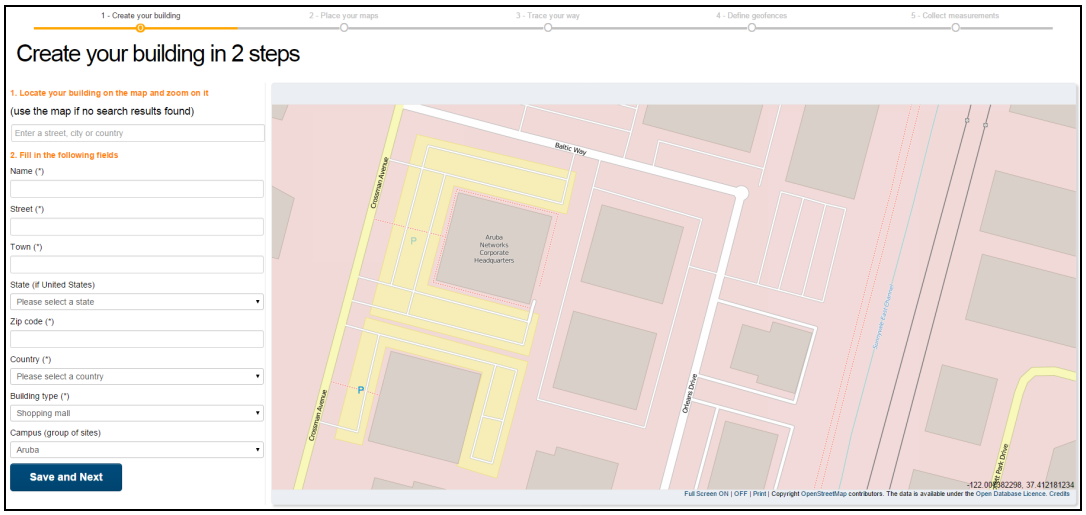
1. Access the Aruba Nao Campus dashboard.

Figure 60 *The Aruba Nao Campus Dashboard*



2. Click **Add a new Building**. Before you proceed, check for the following:
   - You have an Android mobile device
   - You have building maps stored on your computer
3. Under **Create your building**, locate your building using the interactive map, or enter the building address in the address fields. If you cannot locate the building on the map, use an approximate location. If the building address does not appear in your search results, search for a nearby landmark and scroll-over to the approximate building location.
4. Specify the **Building type** and **Campus**.

Figure 61 *Creating an Aruba Nao Campus Building*

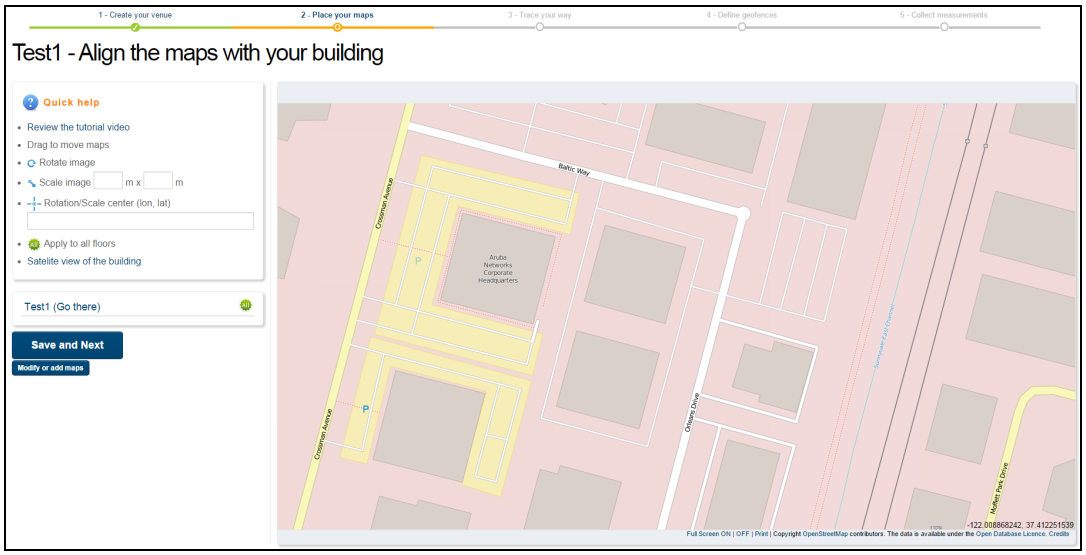


5. Once your building is selected, click **Save and Next**.
6. Under the **Place your maps** page, upload floor plans by dragging and dropping map files (JPG, JPEG, and PNG) to the **Drop files or click here...** box. You can also click the box to select floor plans from your file manager.

Figure 62 *Uploading Maps to Aruba Nao Campus*

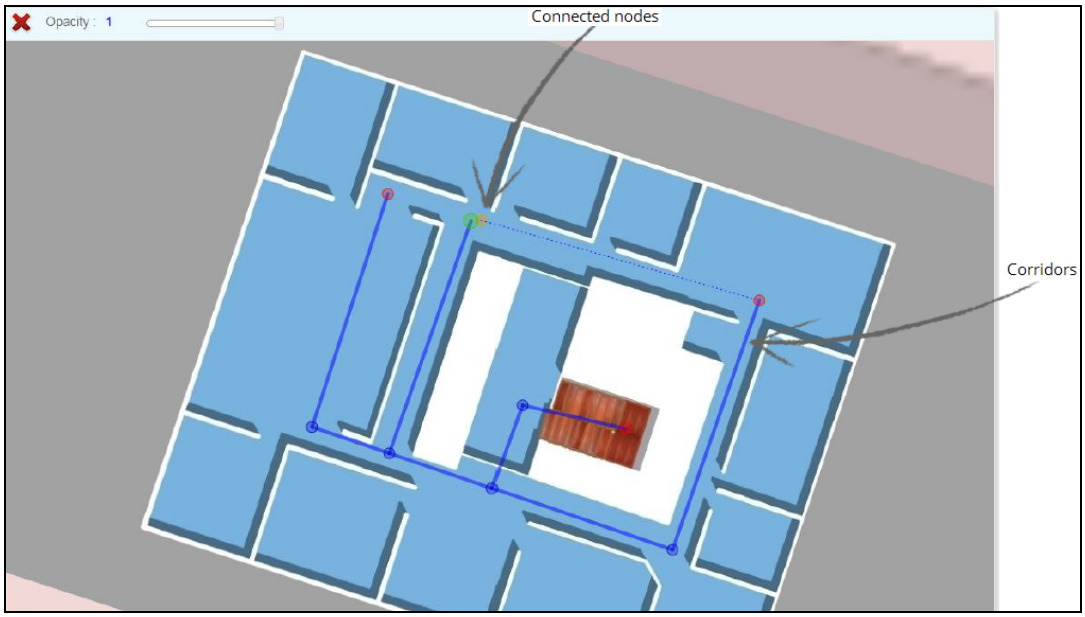
7. Create a name and indicate the floor level number for each floorplan.
8. Click **Save and Next**.
9. Align each floorplan with the building by dragging, rotating, and scaling the image to the correct size and position. The **All** button applies the same alignment to each floorplan.

Figure 63 *Aligning Maps with a Building*



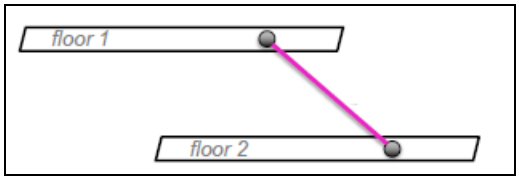
10. Click **Save and Next**.
11. Under **Trace your way**, create a path on the map for collecting fingerprint data. This path appears as a blue line and is later used on the Aruba Nao Logger Android application to generate the Positioning Database (PDB). This is also known as a graph and defines the paths people with devices generally walk through or might be located.

Figure 64 *Creating a Pathway for Fingerprinting*



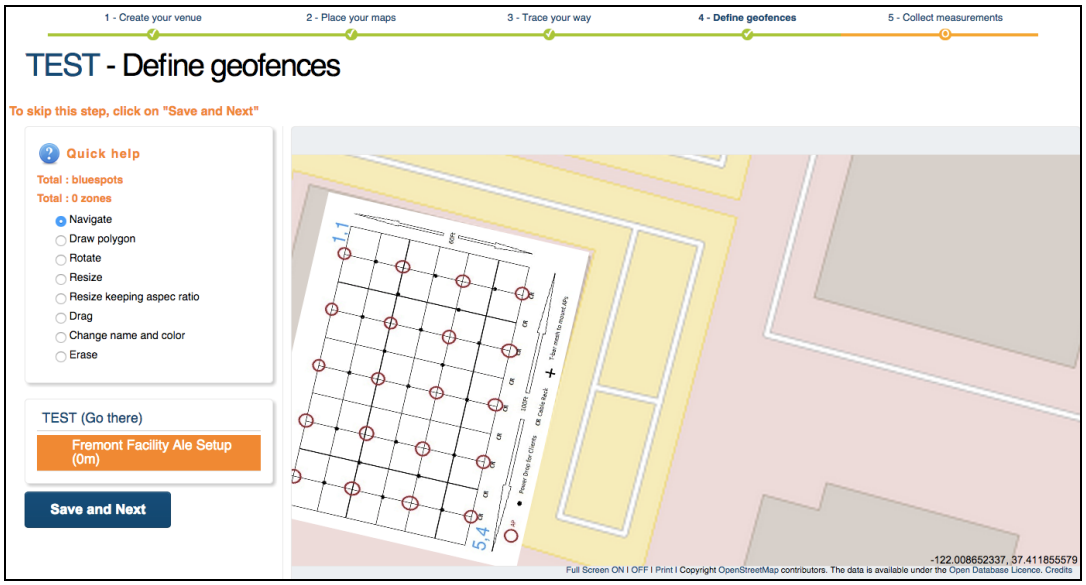
12. If your building contains escalators, stairs, or elevators:
    a. Draw a node on one floor.
    b. Change the floor using the floor selector.
    c. Draw a second node on the new floor. The two nodes connect into a stairway, escalator, or elevator.

Figure 65 *Creating a Stairway between Floors*



13. Create GeoFence regions under **Define geofences**:
    - Use the drawing and editing tools to create the GeoFence region. After the drawing is completed, click **Save and Next**.
    - If your deployment does not require a GeoFence, click **Save and Next** to skip this step.

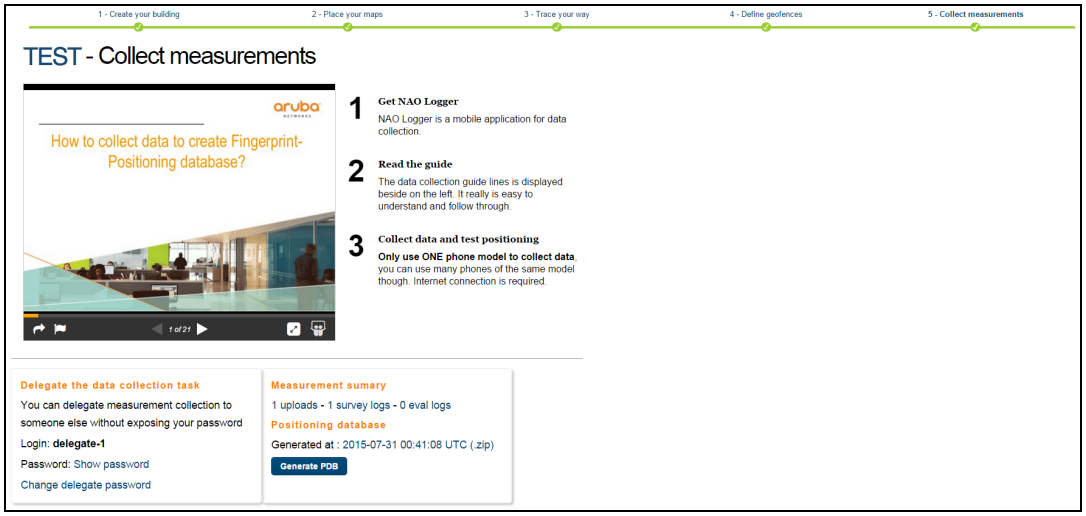
Figure 66 *Defining GeoFences*



14. Once all settings have been configured for the Aruba Nao Campus building, you can begin collecting fingerprint data.

At this point, the workflow moves to the Aruba Nao Logger Android application. Once fingerprinting and testing have been completed through the app, the workflow returns to Aruba Nao Campus, where the PDB can be published. To publish the PDB, locate the relevant building under **NAO Campus Buildings** on the Nao Campus dashboard. Click **API Keys**, and then click **Publish**.

Figure 67 *Collecting Measurements*



Generating and Publishing the Positioning Database

Aruba Nao Logger is an Android application that operates as a companion to the Aruba Nao Campus tool. This application allows an operater to collect WiFi fingerprints (also known as measurements or logs) along the path (graph) defined in Nao Campus, generating a database of Radio Signal Strength distribution across a floor.

1. Download the Aruba Nao Logger application on an Android mobile device.
2. Launch Aruba Nao Logger.
3. Login using your Aruba Nao Campus credentials.

Figure 68 *Logging in on an Android tablet*

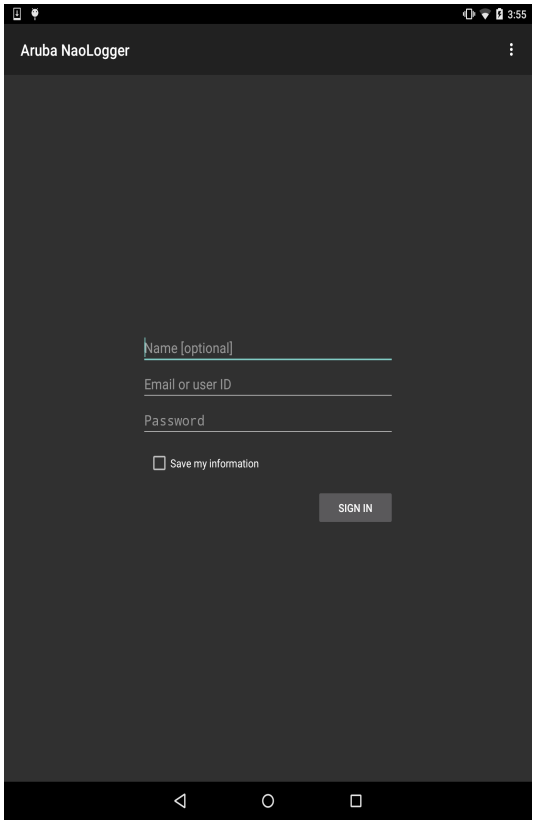
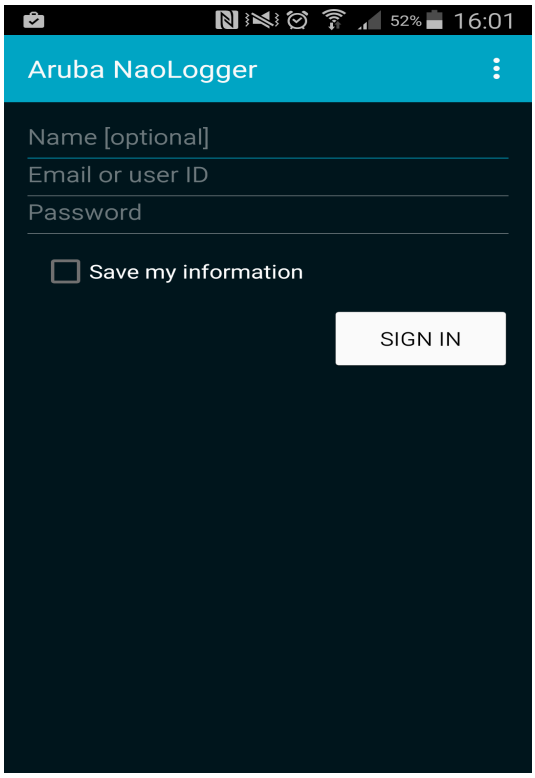


Figure 69 *Logging in on an Android phone*

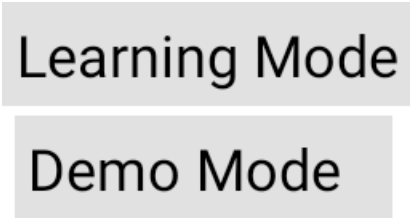


4. Set the **Server URL** to the Nao Campus URL: **https://<ALE IP address>:4000**.
5. Select and download the building map.

6. The map opens and displays the following:
   - Mode:
     - Learning Mode is used for data collection.
     - Demo Mode is used for positioning testing, which uses the location algorithm running inside the Android app (based on sensors on the phone) to validate fingerprint quality. Whenever a fingerprinting operator switches to demo mode, a test PDB is generated on Nao Campus and downloaded on the device.

NOTE

ALE cannot download the PDB to create a production PDB until it is published.



- Floor level number

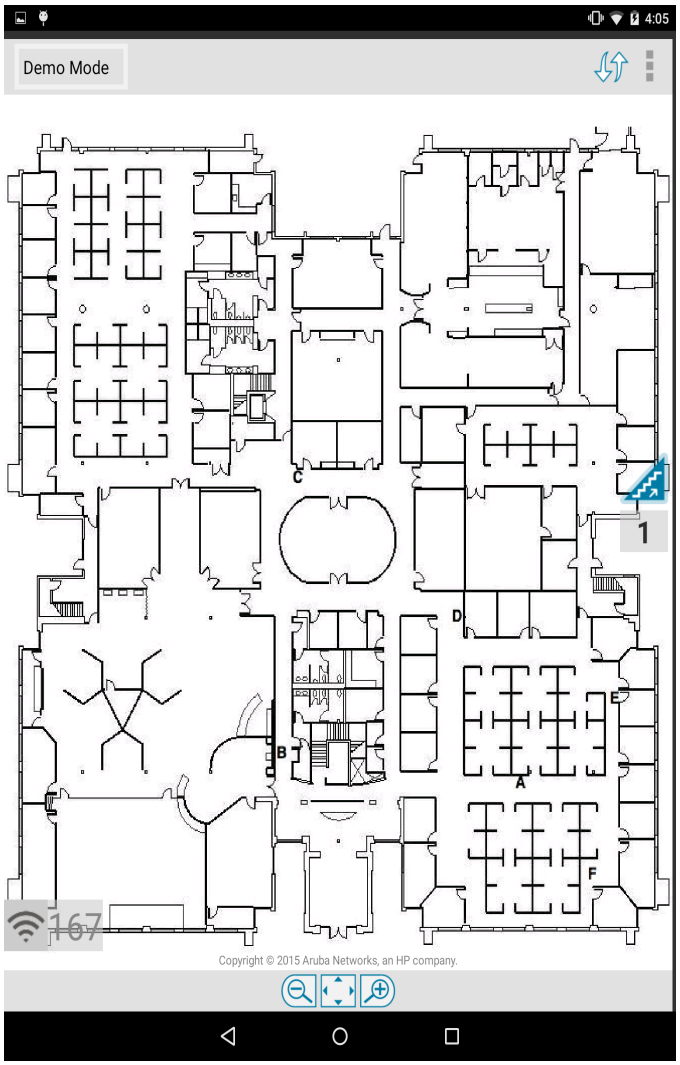
- Number of Wi-Fi beacons

- Synchronize button

- Start and stop button

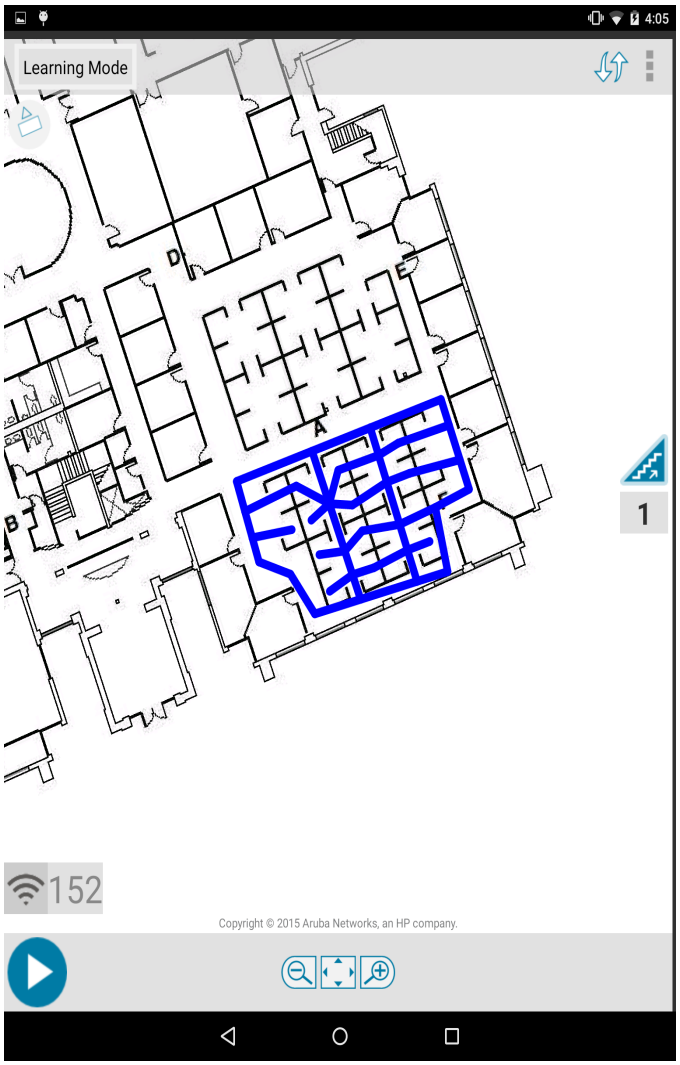
- Blue pathway configured in Aruba Nao Campus

Figure 70 *Aruba Nao Logger Overview*



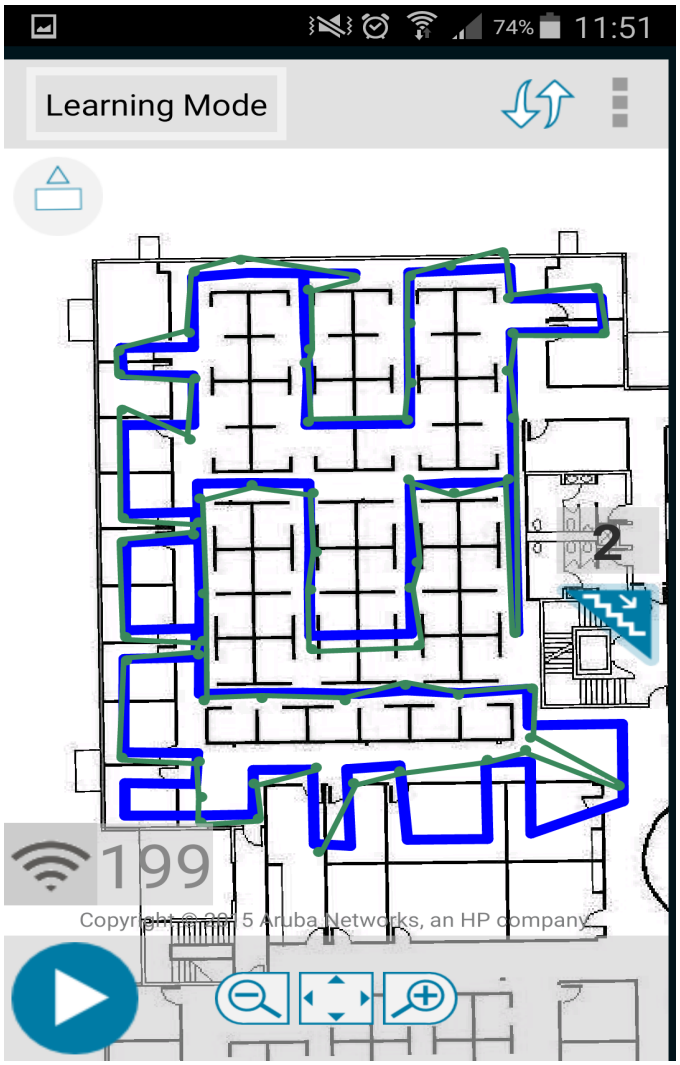
7. Set the map to **Learning Mode**, and then press the **Start** button to begin fingerprinting. The green crosshair indicates your current location.

Figure 71 *Learning Mode*



8. To create your first fingerprint location, press the **Marker** button on the bottom right corner of the screen.
9. After the first marker has been created, walk along the blue pathway to your next fingerprint location. Click the **Marker** button again to create a new fingerprint location. The path between fingerprint locations is displayed as a green line.

Figure 72 *Fingerprinting along the Defined Path*



10. Continue until you have covered all blue pathways with fingerprint locations.
11. Once fingerprinting is completed, press the **Stop** button to save your data. Fingerprinting can also be stopped and saved periodically.
12. Press the **Synchronize** button to send all collected data to the server to generate the PDB.

NOTE

Anytime the floorplan or walkable path is changed, all fingerprinting data is deleted and another data collection session is required.

NOTE

It is recommended to walk each path twice, once in each direction, to improve fingerprint quality.

Once the data has been synchronized, a test PDB can be generated directly from the Nao Logger app by switching the app to Demo Mode. There is no limit to the number of times this can be done. After generating the test PDB, access the Aruba Nao Campus dashboard to publish the PDB and use the data for location calculation.

To publish the PDB:

1. Login to the Aruba Nao Campus dashboard.
2. Under the **NAO Campus Buildings** table, click **API keys** for the fingerprinted site.
3. Click **Publish** to publish the PDB and begin using the data to calculate client location.

Once the production PDB is published, it is highly recommended that you export and save the measurements and PDB in a secure location. To export, access the Nao Campus dashboard and click **Export** for the fingerprinted site listed under **NAO Campus Buildings**. This can be imported later to recreate the PDB.

Once the PDB is published, it is ready to be used by the location engine. ALE must be set to **Context Mode with Device Location (Calibration)**. If ALE is already set to calibration mode before the PDB is published on Aruba Nao Campus, or any further changes are made on Aruba Nao Campus (for example, adding or modifying GeoFence regions), navigate to **Maintenance > Regenerate PDB** to load the new PDB manually.

Restarting ALE Services

ALE processes can be restarted through the following steps:

1. Access the ALE WebUI.
2. Navigate to **Maintenance** on the left window pane.
3. Under the **Restart** tab, click the **Restart Now** button to restart the ALE server.

Downloading the schema.proto File

ALE Publish/Subscribe APIs are defined and generated using a .proto file and protocol buffer compiler (protoc).

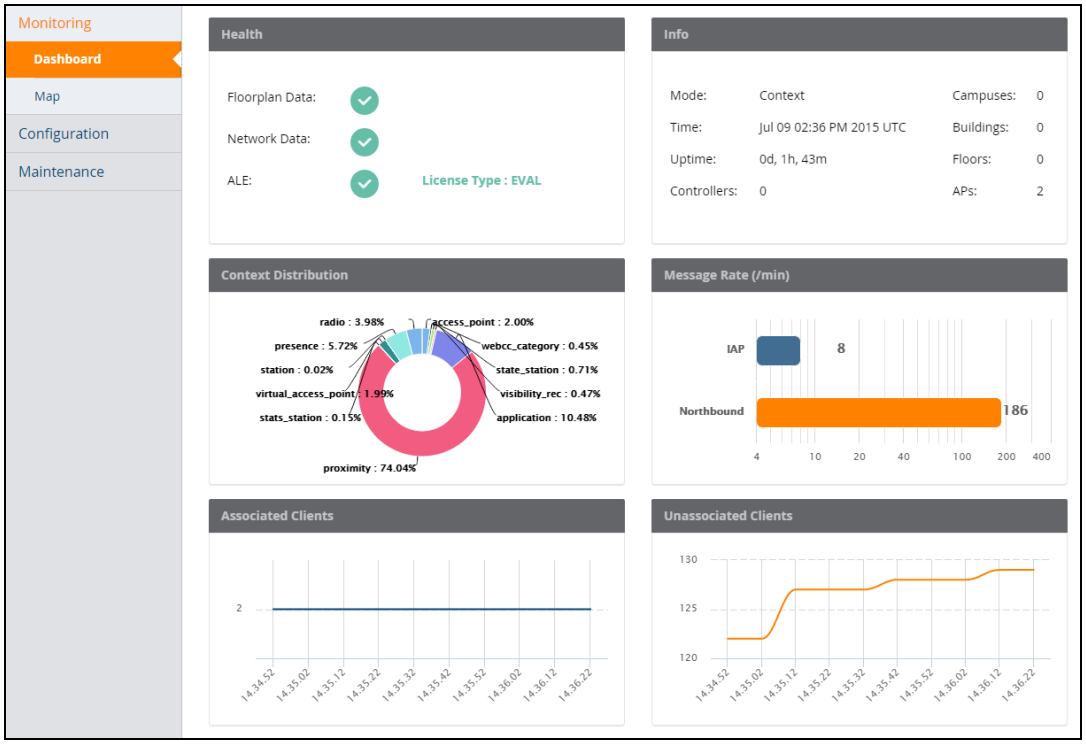
To download the .proto file:

1. Access the ALE WebUI.
2. Navigate to **Maintenance** on the left window pane.
3. Under the **Developer** tab, click **Download schema.proto** to download the latest schema.proto as a ZIP file.

For more information on the ALE Publish/Subsribe APIs, see Publish/Subscribe APIs.

The ALE Dashboard is displayed upon logging in to the ALE server. It is divided into six sections and displays the status of many ALE components at a glance.

Figure 73 *The ALE 2.0 Dashboard Overview*



Health

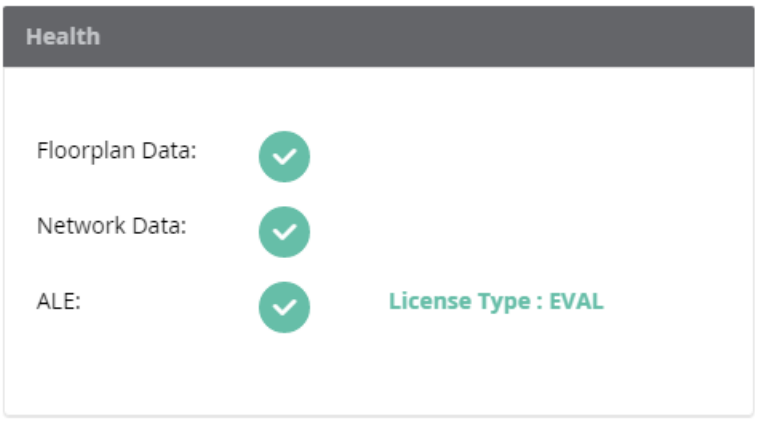
The **Health** section of the ALE Dashboard displays an overview of data and system health.

Table 8: *Health Information Fields*

Field	Description
Floorplan Data	Displays the health status of floorplan data
Network Data	Displays the health status of network data
ALE	Displays the license type and health status of the ALE deployment

A mouse over on each field reveals more details about the health of your data or system.

Figure 74 *ALE Dashboard - Health*



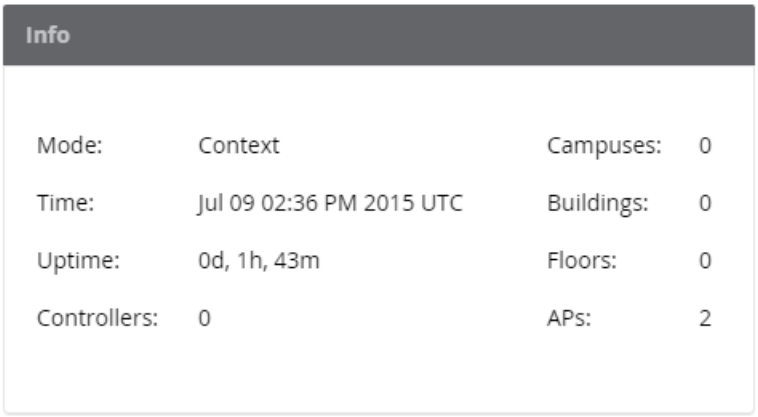
Info

The **Info** section of the ALE Dashboard displays general information about the deployment.

Figure 75 *Info Information Field*

Field	Description
Mode	The configured deployment mode
System time	Current date and time
Uptime	Amount of time the system has been fully functional and operational
Controllers	Number of controllers configured on this ALE. A mouse over on the controllers field reveals details about each controller.
Campuses	Number of campuses (shows 0 if the ALE is deployed under Context Mode)
Buildings	Number of buildings
Floors	Number of floors
APs	Number of APs deployed on the network

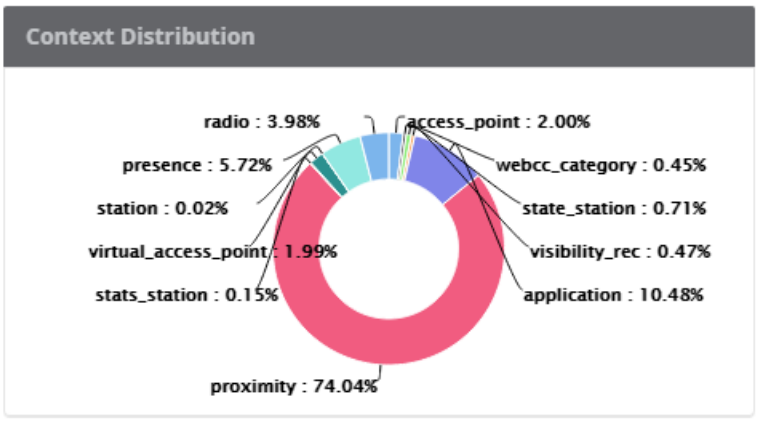
Figure 76 *ALE Dashboard - Info*



Context Distribution

The **Context Distribution** chart provides a percentage-based representation of the various context messages that were generated recently. A mouse over on the dashlet heading reveals the total number of messages generated for each context topic.

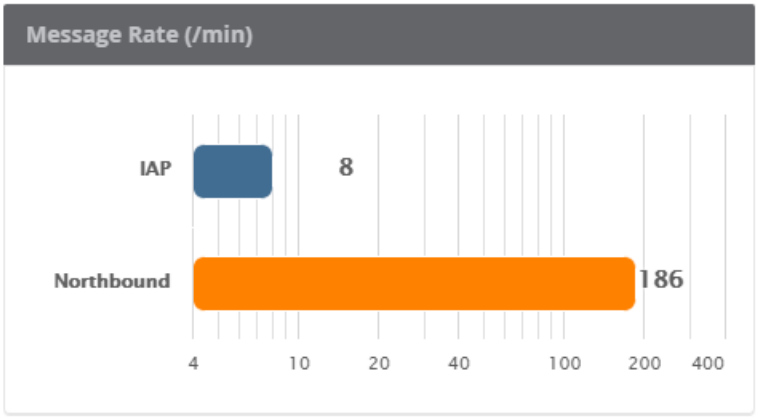
Figure 77 *ALE Dashboard - Context Distribution*



Message Rate

The **Message Rate** section of the ALE Dashboard displays the message rate of input data sources, such as AMON, RTLS, and IAP, and streaming APIs (labelled "Northbound").

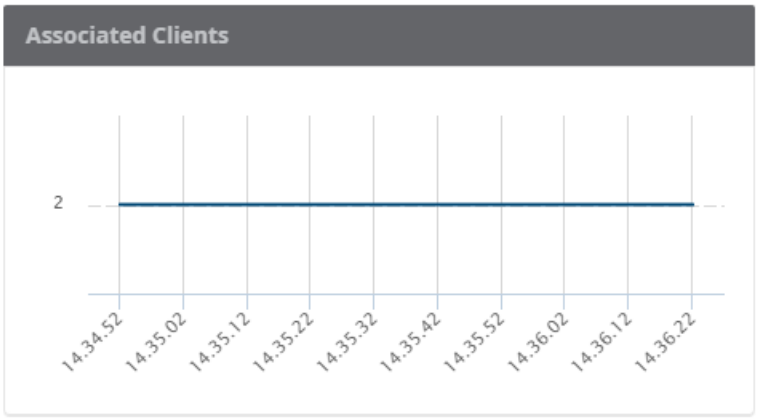
Figure 78 *ALE Dashboard - Message Rate (/min)*



Associated Clients

The **Associated Clients** graph shows how many associated clients are observed by ALE.

Figure 79 *ALE Dashboard - Associated Clients*



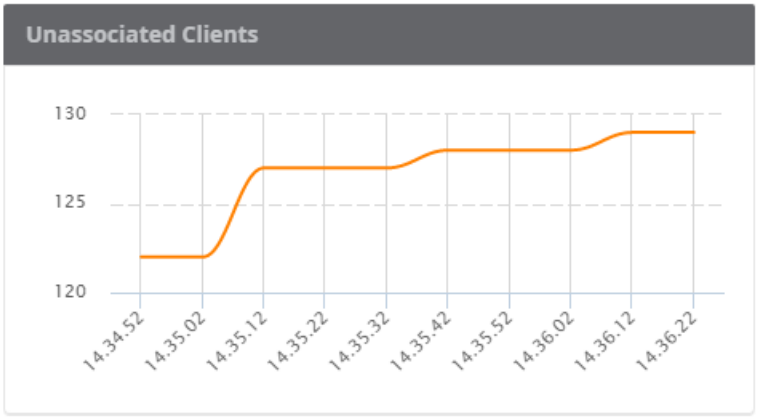
Unassociated Clients

The **Unassociated Clients** graph shows how many unassociated clients are observed by ALE.

NOTE

If the PDB is not available in the estimated and calibration context modes, this chart may not show any data.

Figure 80 *ALE Dashboard - Unassociated Clients*



This chapter discusses the WebSocket Tunnel.

Configuring WebSocket Tunnel

WebSocket Tunnel allows TCP connections to pass through strict firewalls with ease to communicate data between the ALE server and third-party analytics applications.

WebSocket Tunnel consists of a tunnel client and server. The server acts as a middleman to route connections securely through firewalls using an SSL, reducing network traffic and latency. The WebSocket Tunnel server application is distributed with ALE and can be instantiated on Unix-like operating systems. The WebSocket Tunnel client runs on the ALE instance and can be configured from the ALE user interface.

The WebSocket Tunnel Workflow

1. Obtain or generate the WebSocket certificate and key.
2. Instantiate and configure a WebSocket server by downloading the server from **Maintenance > Developer** on the ALE WebUI and running it on a Unix-like server on the third-party network.
3. Configure the WebSocket client on ALE.
4. Test the connection by issuing REST API calls on the server side. Run the sample feed-reader application to test the ZMQ stream. The port number on which these are available on the WebSocket server can be discovered from details under Integration after Establishing the WebSocket Tunnel Connection.

The WebSocket Certificate and Key

WebSocket Tunnel requires a certificate and key file for secure communication between the ALE server and third-party applications. Installation of certificates on the WebSocket Tunnel server is required. Installation of certificates on the WebSocket Tunnel client is optional and only required for 2-way authentication.

Important Points to Remember

Before installing the WebSocket certificate and key, note the following:

- The key file must be unencrypted to work with WebSocket Tunnel.
- Aruba recommends that the certificate and key are separate files.
- The common-name attribute for the certificate and key is set to either the hostname of the WebSocket server or a wildcard to accommodate any host in the domain.

To generate a self-signed certificate for the WebSocket Tunnel, see Generating a Self-Signed Certificate.

Tunnel Servers

Tunnel servers run on an intermediate server and work very similarly to tunnel clients. Servers mirror the ports that are shared by clients. Data from third-party applications is communicated back to the ALE server, and ZMQ stream subscriptions are availalbe on the third-party network.

Downloading Tunnel Server Software:

1. Navigate to **Maintenance > Developer** in the WebUI.
2. Click **Download WebSocket Server** to download the tunnel server software as a TAR.GZ file.

Figure 81 *Downloading WebSocket Tunnel Server Software*

3. Extract the software using a ZIP file archiver to begin server installation.

Installing the Server

The server executable is delivered as a Tar/Gz file and must be installed on the intermediate server.

To install the server, untar the following Tar/Gz file:

```
# tar -xzf ale-wstunnel-<version>-bin.tar.gz -C /my/path/
# cd /my/path/ale-wstunnel-<version>
```

Configuring the Server

To configure the server:

1. Open the property file located at
   `/my/path/ale-wstunnel-<version>/conf/server.properties`
2. Place your certificate and key files in a directory on the WebSocket server to configure in the **server.properties** file.
   a. Specify a directory for client certificates using the
   `nbapi.tunnel.server.twowayssl.certificates.trusted.directory` file.
3. Configure the server properties shown in Table 9.

Table 9: *Server Properties*

Server Property	Description	Default
nbapi.tunnel.server.host	Local interface to listen for clients	0.0.0.0 (all interfaces)
nbapi.tunnel.server.port	Local port to listen for clients (Required)	443
nbapi.tunnel.server.nossl	Does not use SSL You may need this when running behind a load-balancer that handles SSL	False

Server Property	Description	Default
nbapi.tunnel.server.ssl.certificate.file	Location of the certificate file (Required)	–
nbapi.tunnel.server.ssl.certificate.keyfile	Location of the certificate key file (Required)	–
nbapi.tunnel.server.twowayssl.certificates.trusted.directory	Directory for client certificates	–
nbapi.tunnel.server.webhost	Local interface to listen for web/API requests	localhost
nbapi.tunnel.webport	Local port to listen for web/API requests	8700
nbapi.tunnel.server.proxyportstart	LOWEST local port open to relay traffic to the client	12000
nbapi.tunnel.server.proxyportend	HIGHEST local port open to relay traffic to the client	22000

Launching the Server

To launch the server:

1. Open the launch script located at
   `/<install path>/ale-wstunnel-<version>/bin/startup-server`
2. The arguments in Table 10 can be executed in the launch script.

Table 10: *Server Launch Arguments*

Launch Argument	Description
-p <filename>	Saves the PID of the running process to the specified file
-f	Forces the executable to run as a foreground process
-h	Prints usage then exits
-v	Prints current version then exits
-D <key=value>	Additional property to pass to the Java executable
-H	Java heap size used

Tunnel Clients

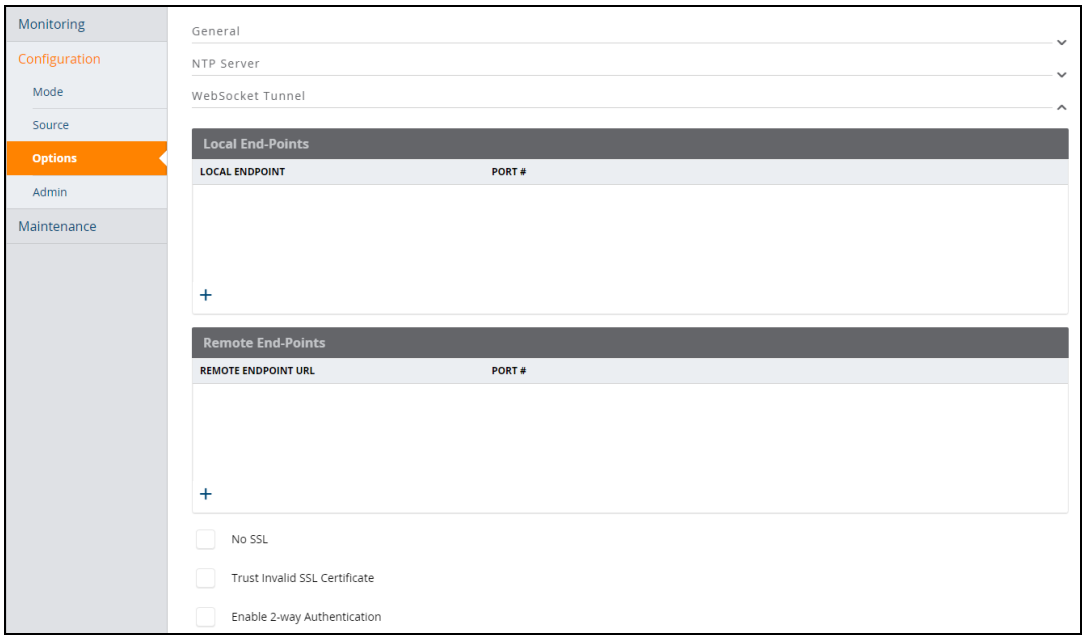
The tunnel client runs on the ALE server.

Configuring a Client

To configure a tunnel client:

1. Navigate to **Configuration > Options > WebSocket Tunnel** in the WebUI.
2. Select local and remote end-points from the **Local End-Points** or **Remote End-Points** table.

Figure 82 *Configuring Tunnel Clients*



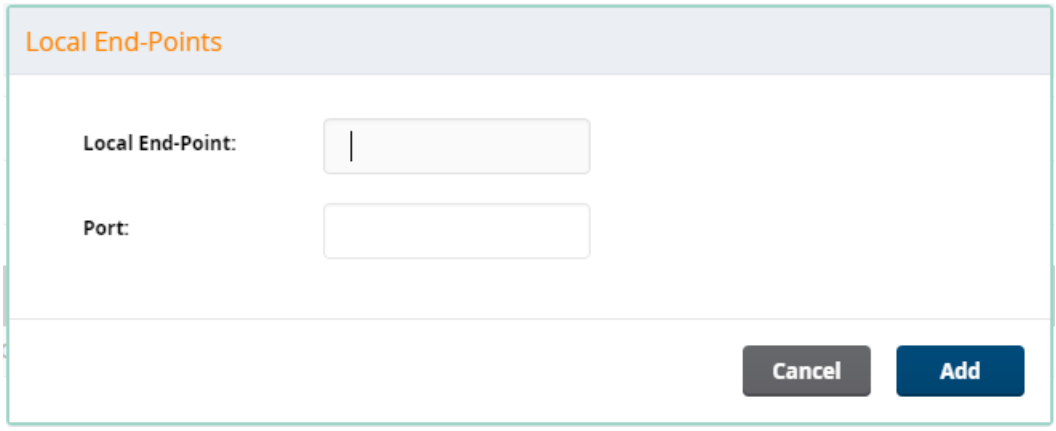
a. To add a new end-point, click the **+** button on the bottom left corner of the **Local End-Points** or **Remote End-Points** table. A pop-up window to add a new end-point opens.

b. Enter the name and port number of the end-point, and then click **Add**.

   a. When adding a local end-point, specify the server that is running the client. In most cases this is "localhost", since the WebSocket client runs on ALE. The port numbers on the local end-points are the ports that will be tunneled. For example, localhost, 7779 is a local end-point for the ZMQ API, and localhost, 8080 is a local end-point for the REST API.

   b. To specify a remote end-point, enter the hostname or IP address of the server running the WebSocket server. The port must be set to the same port configured in the **nbapi.tunnel.server.port** field of the WebSocket server configuration.

Figure 83 *Adding Local End-Points*

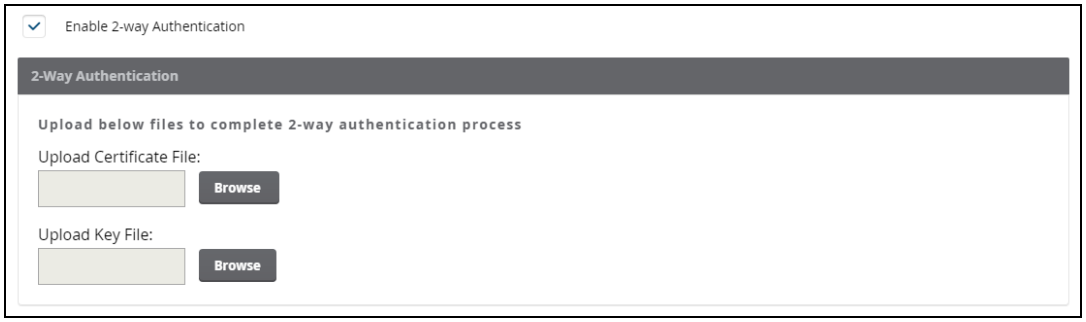


3. Optionally, enable or disable the following settings for SSL and authentication:

   - **No SSL**: This setting enables or disables the Secure Sockets Layer (SSL), which provides a secure connection between web browsers and websites.
   - **Trust Invalid SSL Certificate**: This setting allows you to trust an invalid SSL certificate to bypass invalid SSL certificate errors.

- **Enable 2-way Authentication**: This settings enables or disables 2-way authentication between the tunnel client and tunnel server.
  a. If 2-way authentication is enabled, upload a certificate file to the ALE server by clicking **Browse** under **Upload Certificate File**. The file manager opens.
  b. Select a certificate from the WebSocket Server directory containing your certificate files. Click **Open**.
  c. To upload a key file, click **Browse** under **Upload Key File**. The file manager opens.
  d. Select a key file from the WebSocket Server directory containing your key files. Click **Open**.

Figure 84 *Enabling 2-way Authentication*



4. Click **Apply** to save your configuration.

NOTE

Logs can be viewed in the tech support logs or under /opt/ale/ale-wstunnel/logs/ale-wstunnel.log in the shell.

Integration after Establishing the WebSocket Tunnel Connection

A REST API call can be issued to the WebSocket Tunnel server to discover port mappings between clients and servers.

- **Resource URL:** http://localhost:8700/clients
- **Response Format:** JSON
- **HTTP Method:** GET

Figure 85 *Example Request: GET http://localhost:8700/clients*

```
{
   "clients":[
      {
         "id":"A364012E9E0C",
         "uptime":145422,
         "endpoints":[
            {
               "local_host":"localhost",
               "local_port":12000,
               "remote_host":"localhost",
               "remote_port":22
            }
         ]
      }
   ]
}
```

The tunnel server maps port 12000 to the remote client port 22 (client ID A364012E9E0C). Port 12000 can now be accessed on the intermediate server to establish an SSH connection to the remote client using the following command:

```
# ssh localhost:12000
```

Similarly, if the WebSocket client makes ports 443 (REST API) and 7779 (ZMQ) accessible, they are mapped to ports on the WebSocket server (for example, local_port 12001 is mapped to remote_port 443, and local_port 12002 is mapped to remote_port 7779).

To run a REST API query through the WebSocket server:

1. Authenticate the REST API by using `curl -v http://localhost:12001/api/j_spring_security_check --data "j_username=<username>&j_password=<password>" --cookie-jar /tmp/cookie.txt.`
2. Run a curl query for the API using `curl -v http://localhost:12001/api/v1/<API> --cookie /tmp/cookie.txt`
3. Point the feed-reader to a local port on the WebSocket server by using `[root@ale bin]# ./feed-reader -e tcp://localhost:12000.`

```
[root@ale bin]# ./feed-reader -e tcp://localhost:12000
Attempting to 'connect' to endpoint: tcp://localhost:12000
Connected to endpoint: tcp://localhost:12000
Subscribed to topic: ""
[1] Recv event with topic "access_point"
seq: 5874
timestamp: 1438284521
op: OP_UPDATE
topic_seq: 655
access_point {
ap_eth_mac {
addr: 24:de:c6:ca:61:72
}
ap_name: 24:de:c6:ca:61:72
```

WebSocket Tunnel Server Logs

To view WebSocket server logs, navigate to `/<installpath>/ale-wstunnel-<version>/logs/ale-wstunnel.log`.

Generating a Self-Signed Certificate

To generate a self-signed certificate for the WebSocket Tunnel:

1. Execute the **openssl genrsa -des3 -out server.key 1024** command on an openssl toolkit to generate an RSA private key.

```
openssl genrsa -des3 -out server.key 1024

Generating RSA private key, 1024 bit long modulus
.....................................................++++++
........++++++
e is 65537 (0x10001)
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

2. After the private key is generated, run the **openssl req -new -key server.key -out server.csr** command to create a Certificate Signing Request (CSR). Fill out all required information fields. Self-signed certificates can be generated for testing or internal use (see step 4 on page 81).

```
openssl req -new -key server.key -out server.csr

Country Name (2 letter code): <country code>
State or Province Name (full name): <state/province name>
Locality Name (eg, city): <locality name>
```

```
Organization Name (eg, company): <organization name>
Organizational Unit Name (eg, section): <unit name>
Common Name (eg, your name or your server's hostname): <server domain name>
Email Address: <email address>
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password: <password>
An optional company name: <company name>
```

3. Remove the triple-DES encryption (passphrase) from the private key so the server can startup at any time without requiring a passphrase.

```
cp server.key server.key.org
openssl rsa -in server.key.org -out server.key
```

The unencrypted key file must only be readable by the root user to prevent any third-parties from obtaiing the key.

4. If your certificate has not been signed by the Certification Authority (CA), or testing is required while the certificate is being signed, you can generate a self-signed certificate using the **openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt** command. This temporary certificate expires after 365 days.

The Analytics and Context Engine supports two types of APIs: a polling-based REST API, and a publish/subscribe API based on Google Protobuf and ZeroMQ. This chapter describes the format of information included in the API, the types of data each API returns, and the steps required to use these APIs to view ALE data.

- The REST-based APIs support HTTP GET operations by providing a specific URL for each query. This information is returned in the JSON format. For more information on ALE Polling APIs, see Polling APIs on page 82
- The publish/subscribe API is based on the ØMQ transport. A subscriber uses ØMQ client libraries to connect to ALE and receive information from ALE asynchronously. This information is delivered in the Google Protobuf format. For more information on this group of APIs, see Publish/Subscribe APIs on page 99

When the Analytics and Context Engine integrates with a third-party analytics partner, secure communication between the ALE server and the analytics application may be required. ALE uses a WebSocket Tunnel to secure polling and publish/subscribe APIs and retrieve important context information through a secure channel. For more information, see Configuring WebSocket Tunnel.

Polling APIs

The Representational State Transfer (REST) polling-based API supports HTTPS GET operations by providing a specific URL for each query. The following sections describe each of the Polling APIs supported by ALE. Outputs are displayed in JSON format.

- Access Point API
- Virtual Access Point API
- Stations API
- Presence API
- Proximity API
- Campus API
- Building API
- Floor API
- Location API
- Application API
- Destination API
- Geo_Fence API
- System Information API
- WebCC Category API
- Topology API
- Controller API
- Cluster Info API

Some output parameter fields are optional; certain fields are absent under different modes of operation.

Access Point API

The Access Point (AP) API displays information about APs that terminate on the controllers and IAPs configured to send information to ALE. This API is available in controller and IAP deployments for all operational modes.

AP API queries use the following URL syntax:

`https://1.2.3.4/api/v1/access_point`

The JSON response to this query type displays the following information about the AP:

Table 11: *AP API Output Parameters*

Output Parameter	Definition
ap_eth_mac	MAC address of the AP
ap_name	Name of the AP. If the AP does not have a name, the API returns the IP address of the AP.
ap_group	Name of the AP group
ap_model	Model number of the AP
depl_mode	The AP's deployment mode: • DEPLOYMENT_MODE_CAMPUS • DEPLOYMENT_MODE_REMOTE
ap_ip_address	IP address of the AP

```
https://1.2.3.4/api/v1/access_point
```

This query displays output similar to the example below:

```
{
   "Access_point_result": [
      {
         "msg": {
            "ap_eth_mac": {
               "addr": "D8C7C8C0C7BE"
            },
            "ap_name": "1344-1-AL5",
            "ap_group": "1344-hq",
            "ap_model": "135",
            "depl_mode": "DEPLOYMENT_MODE_CAMPUS",
            "ap_ip_address": {
               "af": "ADDR_FAMILY_INET",
               "addr": "10.6.66.67"
            }
         },
         "ts": 1382046667
      }
   ]
}
```

Virtual Access Point API

The Virtual Access Point API displays information about virtual APs (VAP) configured for WLAN connections on an AP.

VAP API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/virtual_access_point
```

Table 12: *VAP API Output Parameters*

Output Parameter	Definition
bssid	BSSID of the virtual AP
ssid	SSID of the virtual AP
radio_bssid	MAC address of a radio on the virtual AP

```
https://1.2.3.4/api/v1/virtual_access_point
```

This query displays output similar to the example below:

```
{
   "Virtual_access_point_result":[
      {
         "msg":{
            "bssid":
            {
               "addr": "94B40FF11080"
            },
            "ssid": "instant",
            "radio_bssid":
            {
               "addr": "94B40FF11080"
            }
         },
         "ts": 1434644130
      }
   ]
}
```

Stations API

The Stations API displays information about clients associated to the WLAN that sends information to ALE. Every associated and authenticated user on the wireless network is represented by a station object. This API is available in controller and IAP deployments for all operational modes. The JSON response to this query type displays the following types of information about a station:

Station API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/station
```

Table 13: *Stations API Output Parameters*

Output Parameter	Definition
sta_mac_address	MAC address of the client station
username	Corresponding username from the user table on the WLAN controller or IAP
role	Name of the user role currently assigned to the client. This is applicable to only authenticated users
bssid	BSSID that the client is connecting to
device_type	Type of device used by the client • Windows 7 • iOS devices
sta_ip_address	IP address of the client
hashed_sta_eth_mac	Anonymized value of the client MAC address
hashed_sta_ip_address	Anonymized value of the client IP address

```
https://1.2.3.4/api/v1/station
```

This query displays output similar to the example below:

```
{
   "Station_result": [
      {
         "msg": {
            "sta_eth_mac": {
               "addr": "F4F15AA2B8E0"
            },
            "username": "Vjammula",
            "role": "Aruba-Employee",
            "bssid": {
               "addr": "D8C7C888D0D0"
            },
            "device_type": "iPhone",
            "sta_ip_address": {
               "af": "ADDR_FAMILY_INET",
               "addr": "10.11.9.248"
            },
            "hashed_sta_eth_mac": "09097EA8F4DACD5A55F3A9D2F456EFE557D35F09",
            "hashed_sta_ip_address": "436738A08110E88906F8A14CCEF66949A3DBAE01"
         },
         "ts": 1381977108
      },
   ]
}
```

Presence API

The Presence API enumerates all associated and unassociated devices detected or "sighted" by ALE. There is only one presence entry per device.

In distributed IAP deployments, the Presence API enables presence analytics to detect devices. In controller deployments with many access points in a single location, the Proximity or Location APIs provide a more granular position of the devices.

Presence API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/presence
```

The Presence API can focus on one or more individual presence objects by including the following (optional) input parameter in your polling request. Queries that include multiple parameters must format the query with an ampersand (&) symbol between each parameter.

Table 14: *Presence API Query Parameter*

Query Parameter	Definition
associated	Returns a boolean value of true or false. If true, the MAC address is associated. If false, the MAC address is not associated anymore.

The output is filtered to only include information on the records that match your request.

Table 15: *Presence API Output Parameters*

Output Parameter	Definition
stat_eth_mac	MAC address of the client
bool_associated	Indicates whether the client is associated with an AP on the network
hash_stat_eth_mac	Anonymized value of the client MAC address
ap_name	Name of the AP
radio_mac	MAC address of the radio that hears the client

```
https://1.2.3.4/api/v1/presence
```

This query displays output similar to the example below:

```
{
   "Presence_result": [
      {
         "msg": {
            "sta_eth_mac": {
               "addr": "00FF00043DFF"
            },
            "associated": true,
            "hashed_sta_eth_mac": "B9081DE2290A1670307F127D07935F788C7614DE"
            "ap_name": "9c:1c:12:c0:19:5a"
            "radio_mac": {
               "addr": "1864720B8460"
            },
         },
         "ts": 1381871586
      }
   ]
}
```

Proximity API

The Proximity API reports which AP hears the station at the highest RSSI, indicating which AP is closest to the station. This API provides a rough location estimation of the client when ALE runs in context mode without maps.

The Proximity API is particularly useful in distributed deployments, such as IAP deployments, in which mapped locations may not be practical for calculating client location or the number of APs may be too low to determine client location.

Proximity API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/proximity
```

Table 16: *Proximity API Output Parameters*

Output Parameter	Definition
ap_name	Name of the AP
radio_mac	MAC address of the corresponding radio
rssi_val	RSSI value at which the AP hears the station **NOTE:** Attenuation (dBm) = RSSI - 96 - client Tx power. If client power is unknown, a value of 10 is used.
sta_eth_mac	MAC address of the client

```
https://1.2.3.4/api/v1/proximity
```

This query displays output similar to the example below:

```
{
   "Proximity_result": [
      {
         "msg": {
            "ap_name": "9c:1c:12:c0:19:5a"
            "sta_eth_mac": {
               "addr": "FEF50B25A59C"
            },
            "radio_mac": {
               "addr": "1864720B8460"
            },
            "rssi_val": -49,
            "hashed_sta_eth_mac": "D170F995BA2BADA23B27F5412C997F3864BF6FEA"
         }
      }
   ]
}
```

Campus API

Campuses contain buildings with individual floor maps. This API is available in context modes with location.

Campus API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/campus
```

Table 17: *Campus API Output Parameters*

Output Parameter	Definition
campus_id	ID number identifying a specific campus
campus_name	Name of the campus location

```
https://1.2.3.4/api/v1/campus
```

This query displays output similar to the example below:

```
{
   "Campus_result": [
      {
         "msg": {
            "campus_id": "6F9DEC79839D458B9F148D16A46A353E",
            "campus_name": "GAP"
         },
         "ts": 1382046667
      },
   ]
}
```

Building API

The Building API provides information about the buildings within each campus structure. Though each building name must be unique within a campus, other campuses can use the same building name. This API is available in context modes with location.

Building API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/building
```

Table 18: *Building API Output Parameters*

Output Parameter	Definition
building_id	ID number identifying a specific campus building
building_name	Name of the campus building
campus_id	ID number identifying a specific campus location

```
https://1.2.3.4/api/v1/building
```

This query displays output similar to the example below:

```
{
   "Building_result": [
      {
         "msg": {
            "building_id": "83393A922FB249C1929B95393A2AAFDA",
```

```
            "building_name": "3600-RFBOX",
            "campus_id": "ECDDE4535C8E4723B8AF849B3F86E7BF"
         },
         "ts": 1382046667
      }
   ]
}
```


Floor API

The Floor API retrieves floor definitions for each building in a campus. Campuses contain buildings with individual floor maps. Though each floor name must be unique within a building, other buildings can use the same floor name. This API is only available in context modes with location and can be used to retrieve the URL of the floor plan image.

Floor API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/floor
```

Table 19: *Floor API Output Parameters*

Output Parameter	Definition
floor_id	ID number identifying a specific building floor
floor_name	Name of the building floor
floor_latitude	Latitude coordinate of the top left corner of this floor
floor_longitude	Longitude coordinate of the top left corner this floor
floor_img_path	URL path to retrieve the background image for this floor
floor_img_width	Floor width in the configured units
floor_img_length	Floor length in the configured units
building_id	ID number identifying the building containing this floor

```
https://1.2.3.4/api/v1/floor
```

This query displays output similar to the example below:


```
{
   "Floor_result": [
      {
         "msg": {
            "floor_id": "1C48EF7D78DC4B948F1A15D7CD14FDED",
            "floor_name": "Floor 1",
            "floor_latitude": 0,
            "floor_longitude": 0,
            "floor_img_path": "/images/plan/img_1c48ef7d-78dc-4b94-8f1a-15d7cd14fded.jpg",
            "floor_img_width": 246.33,
            "floor_img_length": 249.92,
            "building_id": "DAD3A7092AC04AA4B2F5DAF266EBD81B"
         },
         "ts": 1382046667
      }
   ]
}
```

Location API

The Location API retrieves the last known location for a specific MAC client. If historical locations are important to your use-case, they can be stored in an external database or filesystem by listening to and saving the publish/subscribe Location API.

The Location API is only available in context modes with location. If AP density is insufficient but location information is desired, "single AP location" can be enabled to ensure that a rough location is still calculated, though at a higher uncertainty.

Location API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/location?sta_eth_mac=AA:BB:CC:DD:EE:FF
```

The Location API only supports querying using MAC addresses.

Table 20: *Location API Query Parameter*

Query Parameter	Definition
stat_eth_mac	Returns presence objects in context of a specific MAC address. For example, AA:BB:CC:DD:EE:FF.

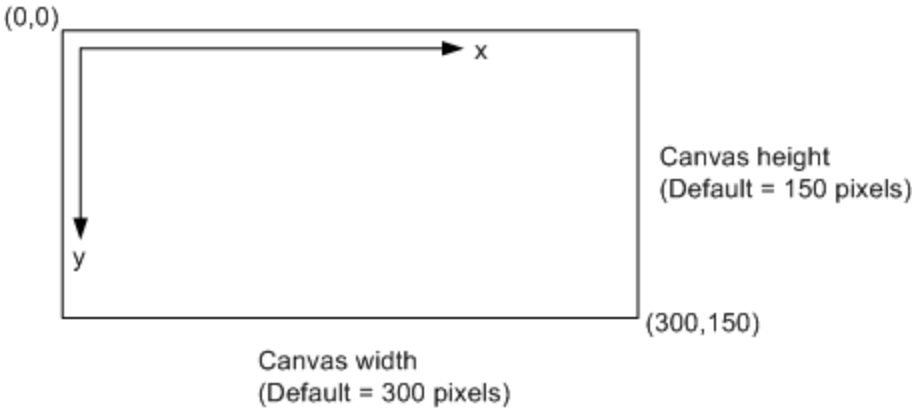
The output is filtered to only include information on the records that match your request.

Table 21: *Location API Output Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client
sta_location_x	X coordinate of the client (in feet or meters, if configured)
sta_location_y	Y coordinate of the client (in feet or meters, if configured)
error_level	Indicates the radius of horizontal uncertainty, computed at 95%. This means the sum of the probability of potential locations contained in this uncertainty circle represents 95% of the whole venue probability. The unit of this radius is configured and published in the unit field.
associated	Indicates whether the client is associated with an AP on the network
campus_id	ID number identifying a specific campus
building_id	ID number identifying a specific campus building
floor_id	ID number identifying a specific building floor
hashed_sta_eth_mac	Anonymized value of the client MAC address
loc_algorithm	Indicates how the (X,Y) coordinates are populated: • **ALGORITHM_TRIANGULATION**: Triangulation for ALE 1.3.x and earlier • **ALGORITHM_AP_PLACEMENT**: Single AP location for ALE 1.3.x and earlier • **ALGORITHM_CALIBRATION**: Calibration mode • **ALGORITHM_ESTIMATION**: Estimation mode • **ALGORITHM_LOW_DENSITY**: Location from a low AP density area; single AP location for ALE 2.0 onwards
longitude	Longitude coordinate of the client

Output Parameter	Definition
latitude	Latitude coordinate of the client
altitude	Altitude of the client
unit	Metric of distance, in feet or meters

Figure 86 *Determining the X,Y Coordinates of a Client*



This query displays output similar to the example below:

```
{
   Location_result": [
      {
         "msg":{
            "sta_eth_mac": {
               "addr": "c0:bd:d1:56:81:f3"
            },
            "sta_location_x": 17.033,
            "sta_location_y": 16.5164,
            "error_level": 9,
            "associated": true,
            "campus_id": "08FBBBBF81D937759B5DAC4963DFBC1A",
            "building_id": "24C73B58A1F33C3ABE427485A9977BFF",
            "floor_id": "D635A61B06673775ADFF61D70B55785C",
            "hashed_sta_eth_mac": "A09B5D8F99F9BB8034A8ADBBEC11B24494981096",
            "loc_algorithm": "ALGORITHM_CALIBRATION",
            "longitude": -122.008,
            "latitude": 37.4129,
            "altitude": 5,
            "unit": METERS
         }
         "ts": 1434750262
      }
   ]
}
```

Application API

The Application API maps an application ID, which is specified in a Visibility Record, to an application name string. This API is available is both controller and IAP deployments. See Publish/Subscribe APIs for more information on the Visibility Record.

Client Application API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/application
```

The response to this query type displays the following information:

Table 22: *Application API Output Parameters*

Output Parameter	Definition
app_id	ID number identifying a specific application
app_name	Name of the application
app_family	The application category **NOTE:** Only available in IAP deployments
app_long_name	Name of the application or the domain name **NOTE:** Only available in IAP deployments

```
https://1.2.3.4/api/v1/application
```

This query displays output similar to the example below in a controller deployment:

```
{
   "Application_result": [
      {
         "msg": {
            "app_id": 50331801,
            "app_name": "Smarter Balanced Testing"
         }
      }
   ]
}
```

This query displays output similar to the example below in an IAP deployment:

```
{
   "Application_result": [
      {
         "msg": {
            "app_id": 999,
            "app_name": "foxnews",
            "app_family": "Web",
            "app_long_name": "FoxNews.com"
         }
      }
   ]
}
```

Destination API

The Destination API maps the destination IP address from the Visibility Record to a domain name. This API is not available in IAP deployments. See Publish/Subscribe APIs for more information.

Destination API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/destination
```

Table 23: *Destination API Output Parameters*

Output Parameter	Definition
dest_ip	IP address of the client receiving traffic through the network
dest_name	Name of the client destination
dest_alias_name	Alternate name for the client destination

```
https://1.2.3.4/api/v1/destination
```

This query displays output similar to the example below:

```
{
   "Destination_result": [
      {
         "msg": {
            "dest_ip": {
               "af": "ADDR_FAMILY_INET",
               "addr": "98.175.77.106"
            },
            "dest_name": "adserving.autotrader.com",
            "dest_alias_name": "autotrader"
         }
      }
   ]
}
```

Geo_Fence API

GeoFencing allows a network administrator to designate regions and leverage client location information to monitor client traffic through those designated regions. Once ALE downloads the region information from AirWave or the NaoCloud tool, it uses this information to identify the relationship between the devices and the regions. The Geo_Fence API describes the shape and coordinates of the polygon defining the region.

Geo_Fence API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/geo_fence
```

Table 24: *GeoFence API Output Parameters*

Output Parameter	Definition
flood_id	ID number identifying a specific building floor
geofence_id	ID number identifying a specific GeoFence region
geofence_name	Name of the GeoFence region
point_list	The (X,Y) coordinates of the points designating the GeoFence region

```
https://1.2.3.4/api/v1/geo_fence
```

This query displays output similar to the example below:

```
{
   "Geofence_result":[
      {
         "msg":{
            "floor_id": "260BE76B0DD13E7AAF18EB3B47DD7F7B",
            "geofence_id": "51C67C0A356F35C39089664022AB4BED",
            "geofence_name": "b8b87ef6-da7b-4d51-b7dc-d901547189fd",
            "point_list":[
               {
               "x": 146.52,
               "y": 207.9
               },
               {
               "x": 137.65,
               "y": 207.55
               },
               {
               "x": 138.01,
               "y": 213.22
               },
               {
               "x": 147.94,
               "y": 213.58
               }
            ]
         }
      }
   ]
}
```

System Information API

The System Information API displays general information about your ALE configuration.

System Information API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/info
```

Table 25: *System Information API Output Parameters*

Output Parameter	Definition
current_mode	The current deployment mode: • CONTEXT • CONTEXT_AND_ESTIMATED_LOCATION • CONTEXT_AND_LOCATION_WITH_CALIBRATION
license_valid	Returns a boolean value of true or false. If true, a valid license is installed on your system. If false, a valid license is not installed.

```
https://1.2.3.4/api/v1/info
```

This query displays output similar to the example below:

```
{
   "Info_result":[
      {
         "msg":{
            "current_mode": "CONTEXT_AND_LOCATION_WITH_CALIBRATION",
```

```
            "license_valid": true,
            "key_value":[
               {
               "key": "geofence_enabled",
               "value": "false"
               }
            ]
         },
         "ts": 1435017107
      }
   ]
```

WebCC Category API

The WebCC Category API retrieves information about the types of websites clients visit when they are connected to the network. This API is available in both controller and IAP deployments. The optional `webcc_cat_id` input parameter, as shown in the example below, restricts the response to a specific `webcc_cat_id`. If this parameter is not supplied, the entire table is returned.

WebCC Category API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/webcc_category
```

Table 26: *WebCC Category API Output Parameters*

Output Parameter	Definition
cat_id	ID number specifying the website category
category	The website category

```
https://1.2.3.4/api/v1/webcc_category?webcc_cat_id=2
```

This query displays output corresponding to the category ID 2 (building):

```
{
   "WebCCCategory_result":[
      {
         "msg":{
            "cat_id": 2,
            "category": "computer/internet-security"
            },
            "ts": 1428011789
         }
      }
   ]
}
```

Topology API

The Topology API describes the hierarchy between networking components in both controller and IAP deployments. For example, a controller deployment runs on the following network hierarchy:

Controllers > APs connected to the controller > radios in the AP > virtual APs connected to the radios.

Topology API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/topology
```

The default API query returns a compressed network topology. For a more detailed network topology, use the following optional query parameters:

Table 27: *Topology API Query Parameters*

Query Parameter	Definition
expand	Returns a boolean value of true or false. If true, the API displays an expanded network topology tree. If false, the API displays the compressed network topology tree.
managed_by	Allows you to request for a specific controller IP address.

The Topology API displays the following output parameters, but not all fields are present in both controller and IAP deployments. Refer to Table 28 and the example ouputs to determine which fields are present in each mode.

Table 28: *Topology API Output Parameters*

Output Parameter	Definition
controller_ip_address	IP address of the controller. **NOTE:** Only available in controller deployments
ap_eth_mac	MAC address of the AP
ap_name	Name of the AP
radio_bssid	MAC address of a radio on the AP
bssid	MAC address of the virtual AP
cluster_key	Unique key identifying the cluster deployment. **NOTE:** Only available in IAP deployments
cluster_ip	IP address of the cluster. **NOTE:** Only available in IAP deployments

```
https://1.2.3.4/api/v1/topology
```

The following query displays the topology output for a controller deployment:

```
{
   "Topology_result":[
      {
         "msg":{
            "controller":{
               "controller_ip_address.addr":"10.11.0.10",
            "access_points":[
               {"ap_eth_mac.addr":"D8.C7.C8.CC.6D.80",
               "ap_name":"1344-1-AL59 (AM)",
            "radios":[
               {"radio_bssid.addr":"D8.C7.C8.46.D8.10"},
               {"radio_bssid.addr":"D8.C7.C8.46.D8.00"}]},
            {"ap_eth_mac.addr":"D8.C7.C8.CC.6D.7C",
               "ap_name":"1344-1-AL56 (AM)",
            "radios":[
               {"radio_bssid.addr":"D8.C7.C8.46.D7.D0"},
               {"radio_bssid.addr":"D8.C7.C8.46.D7.C0"}]},
            "virtual_access_points":[
               {"bssid.addr":"18.64.72.D7.69.21"},
               {"bssid.addr":"18.64.72.D7.69.20"}]}]},
            {"ap_eth_mac.addr":"18.64.72.C5.75.F0",
               "ap_name":"1344-2-AP36",
            "radios":[
               {"radio_bssid.addr":"18.64.72.D7.5F.10",
```

```
            "virtual_access_points":[
               {"bssid.addr":"18.64.72.D7.5F.11"},
               {"bssid.addr":"18.64.72.D7.5F.10"}]},
               {"radio_bssid.addr":"18.64.72.D7.5F.00",
            "virtual_access_points":[
               {"bssid.addr":"18.64.72.D7.5F.01"},
               {"bssid.addr":"18.64.72.D7.5F.00"}]}]}]}},
         ,"ts":1437496948
      }
   ]
}
```


The following query displays the topology output for an IAP deployment:


```
{
   "Topology_result":[
      {
         "msg":{
            "cluster_info":{
               "cluster_key":"c80a4ace01ea6b8ed5908826d2afc391c1aeb2355e2d7155ba",
               "cluster_ip.addr":"10.5.166.57",
            "access_points":[
               {"ap_eth_mac.addr":"94.B4.0F.C7.11.08",
               "ap_name":"94:b4:0f:c7:11:08",
            "radios":[
               {"radio_bssid.addr":"94.B4.0F.F1.10.90"},
               {"radio_bssid.addr":"94.B4.0F.F1.10.80",
            "virtual_access_points":[
               {"bssid.addr":"94.B4.0F.F1.10.80"}]}]},
         "ts":1437763117
      }
   ]
}
```


Controller API

The Controller API provides the list of controllers within a network.

Controller API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/controller
```

Table 29: *Controller API Output Parameter*

Output Parameter	Definition
controller_ip_address	IP address of the controller

```
https://1.2.3.4/api/v1/controller
```

This query displays output similar to the example below:


```
{
   "Controller_result": [
      {
         "msg": {
            "controller_ip_address": {
               "af": "ADDR_FAMILY_INET",
               "addr": "10.11.0.10"
            }
         },
         "ts": 1437499995
      }
   ]
```

Cluster Info API

The Cluster Info API displays information about clusters in an IAP deployment.

Cluster Info API queries use the following URL syntax:

```
https://1.2.3.4/api/v1/cluster_info
```

Table 30: *Cluster Info API Output Parameters*

Output Parameter	Definition
cluster_key	Unique key identifying the cluster deployment
cluster_name	Name of the cluster
cluster_ip	IP address of the cluster

```
https://1.2.3.4/api/v1/cluster_info
```

This query displays output similar to the example below:

```
{
   "Cluster_result": [
      {
         "msg": {
            "cluster_key": "c80a4ace01ea6b8ed5908826d2afc391c1aeb2355e2d7155ba",
            "cluster_name": "VC-VTHAKKAR",
            "cluster_ip": {
               "af": "ADDR_FAMILY_INET",
               "addr": "10.5.166.57"
            }
         },
         "ts": 1437678800
      }
   ]
}
```

Publish/Subscribe APIs

The ALE publish/subscribe API uses ØMQ client libraries to allow network administrators to connect to ALE and subscribe to selected topics. Once a user has subscribed to a topic, ALE begins publishing messages and sends them to the subscribers.

All messages are encoded using Google Protocol Buffer and specified using a .proto file. Network application developers, who create applications to process this data, use the .proto file and protocol buffer compiler (protoc) to generate a message parsing code in the desired programming language (for example, C++, Java, or Python).

The Northbound API (NBAPI) publishes messages as events. An event message is the only message type NBAPI will publish. The NBAPI embeds other message types depending on the event.

The station is removed from the ALE table if the controllers also remove the station from it's user table. The user idle-timeout for removal is 5 minutes by default and is configurable by using "aaa timers idle-timeout x" where x is the number of minutes for the user to be idled out of the system.

The event protobuf schema is as follows:

```
// Event message definition

message nb_event {
   enum event_operation {
      OP_ADD = 0;
      OP_UPDATE = 1;
      OP_DELETE = 2;
   }
   optional uint64 seq = 1;
   optional uint32 timestamp = 2;
   optional event_operation op = 3;
   optional uint64 topic_seq = 4;
   optional bytes source_id = 5;

   // One of the following is populated depending on the topic
   optional location location = 500;
   optional presence presence = 501;
   optional rssi rssi = 502;
   optional station station = 503;
   optional radio radio = 505;
   optional destination destination = 507;
   optional application application = 509;
   optional visibility_rec visibility_rec = 510;
   optional campus campus = 511;
   optional building building = 512;
   optional floor floor = 513;
   optional access_point access_point = 514;
   optional virtual_access_point virtual_access_point = 515;
   optional geofence geofence = 516;
   optional geofence_notify geofence_notify = 517;
   optional stats_radio stats_radio = 518;
   optional stats_vap stats_vap = 519;
   optional stats_station stats_station = 520;
   optional ap_neighbor_list ap_neighbor_list = 521;
   optional utilization_stats_radio utilization_stats_radio = 522;
   optional sta_rssi sta_rssi = 523;
   optional ap_rssi ap_rssi = 524;
   optional proximity proximity = 525;
   optional webcc_category webcc_category = 526;
```

```
   optional webcc_info webcc_info = 527;
   optional security_message security_message = 528;
   optional spectrum_info spectrum_info = 529;
   optional state_station state_station = 530;
}
```

The global field definitions are as follows:

- **seq**: Uniquely assigned global sequence number
- **timestamp**: Time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds when this event occurred
- **op**: Event operation code. Reflects the new object state. Possible values are: OP_ADD, OP_UPDATE, OP_DELETE
- **topic_seq**: Per topic uniquely assigned sequence number
- **source_id**: Random number of bytes used as a unique ID for the source ALE. This number is unique per ALE instance. The unique source_ids are persisted across reboots.

Access Point

The Access Point message is sent when a new access point (AP) is deployed into the network. This message is sent as soon as the AP joins the network. It is also sent when an AP goes down, comes back up, and then joins a controller.

When ALE is initiated, this information is bootstrapped from the controller and published immediately. In IAP deployments, this is sent periodically.

The output for this message type displays the following information:

Table 31: *Access Point API Message Parameters*

Output Parameter	Definition
ap_eth_mac	MAC address of the AP
ap_name	Name of the AP
ap_group	Name of the AP group
ap_model	AP model type
depl_mode	AP's deployment mode strings, displayed as an integer corresponding to a mode type
ap_ip_address	IP address of the AP

The Access Point API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "access_point"
Protobuf schema

message access_point {
   enum deployment_mode {
   DEPLOYMENT_MODE_CAMPUS = 0;
   DEPLOYMENT_MODE_REMOTE = 1;
}
   optional mac_address ap_eth_mac = 1;
   optional string ap_name = 2;
   optional string ap_group = 3;
   optional string ap_model = 4;
```

```
   optional deployment_mode depl_mode = 5;
   optional ip_address ap_ip_address = 6;
}
```

Application

The Application message is sent when a new application is classified or otherwise recognized, mapping an application ID from the Visibility Record message to an application name. This API is available in both controller and IAP deployments. In IAP deployments, this message is only sent once during ALE startup.

The output for this message type displays the following information:

Table 32: *Application API Message Parameters*

Output Parameter	Definition
app_id	ID number identifying a specific application
app_name	Name of the application
app_family	The application category **NOTE:** Only available in IAP deployments
app_long_name	Name of the application or the domain name **NOTE:** Only available in IAP deployments

The Application API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "application"
Protobuf schema

message application {
   optional uint32 app_id = 1;
   optional string app_name = 2;
   optional string app_family = 3;
   optional string app_long)name = 4;
}
```

Campus

The Campus message announces a campus. The Campus message is sent only once during ALE startup. This API is available in both controller and IAP deployments and only sends messages in context modes with location (estimated and calibration).

The output for this message type displays the following information:

Table 33: *Campus API Message Parameters*

Output Parameter	Definition
campus_id	ID number identifying a specific campus
campus_name	Name of the campus

The Campus API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "campus"
Protobuf schema

message campus {
```

```
   optional bytes campus_id = 1; // 16 bytes id
   optional string campus_name = 2;
}
```

Building

The Building message announces a campus building. Each campus can have multiple buildings, but a building can be located on only one campus. The Building message is sent only once during ALE startup. This API is available in both controller and IAP deployments and only sends messages in context modes with location (estimated and calibration).

The output for this message type displays the following information:

Table 34: *Building API Message Parameters*

Output Parameter	Definition
building_id	ID number identifying a specific campus building
building_name	Name of the campus building
campus_id	ID number identifying a specific campus

The Building API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "building"
Protobuf schema

message building {
   optional bytes building_id = 1; // 16 bytes id
   optional string building_name = 2;
   optional bytes campus_id = 3;  // 16 bytes id;
}
```

Floor

The Floor message announces a building floor. Each building can have multiple floors, but a floor can be located on only one building. The Floor message is sent only once during ALE startup. This API is available in both controller and IAP deployments and only sends messages in context modes with location (estimated and calibration).

The output for this message type displays the following information:

Table 35: *Floor API Message Parameters*

Output Parameter	Definition
floor_id	ID number identifying a specific building floor
floor_name	Name of the floor
floor_latitude	Latitude coordinate of the top left corner of this floor (defined in Visual RF)
floor_longitude	Longitude coordinate of the top left corner this floor (defined in Visual RF)
floor_img_path	URL path to retrieve the background image for this floor
floor_img_width	Floor width (in feet, or meters if configured)

Output Parameter	Definition
floor_img_length	Floor length (in feet, or meters if configured)
building_id	ID number identifying a specific campus buliding
floor_level	Floor level within a building
units	Unit of measurement used for floor specifications (for example, width and length)

The Floor API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "floor"
Protobuf schema

message floor {
   optional bytes floor_id = 1; // 16 bytes id
   optional string floor_name = 2;
   optional float floor_latitude = 3;
   optional float floor_longitude = 4;
   optional string floor_img_path = 5;
   optional float floor_img_width = 6;
   optional float floor_img_length = 7;
   optional bytes building_id = 8; // 16 bytes id
   optional float floor_level = 9;
   optional string units = 10;
}
```

Destination

The Destination message is sent when a new destination is classified. This API can be used to map a destination IP address from the Visibility Records to a destination domain name. This API is only available in controller deployments.

The output for this message type displays the following information:

Table 36: *Destination API Message Parameters*

Output Parameter	Definition
dest_ip	IP address of the client receiving traffic through the network
dest_name	Name of the client destination
dest_alias_name	Alternate name for the client destination

The Destination API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "destination"
Protobuf schema

message destination {
   optional ip_address dest_ip = 1;
   optional string dest_name = 2;
   optional string dest_alias_name = 3;
}
```

Location

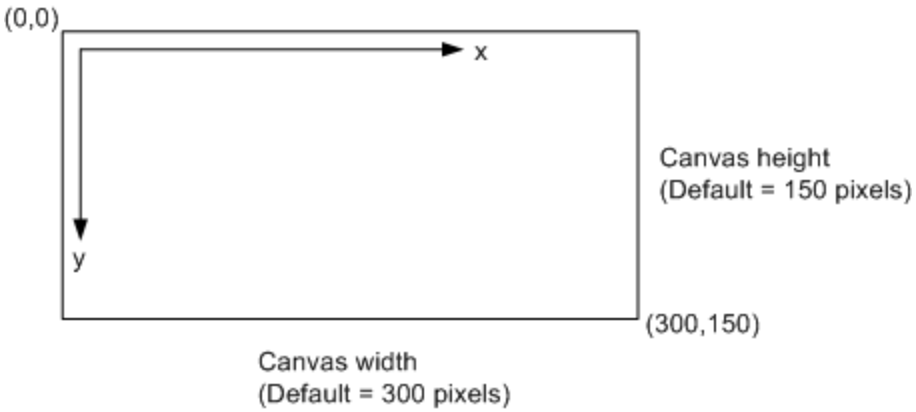
The Location message sends location updates for a specific station. This message is sent as soon as ALE calculates the location of an associated or unassociated client with a specified MAC address. The X and Y location values indicate the number of feet, or meters if configured, the station is from the top left corner of the floor map. This message is only sent in context modes with location (estimated and calibration).

The output for this message type displays the following information:

Table 37: *Location API Message Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client
sta_location_x	X coordinate used to determine client location on a map
sta_location_y	Y coordinate used to determine client location on a map
error_level	Indicates the error radius in the configured distance units
associated	Indicates whether the client is associated with an AP on the network
campus_id	ID number identifying a specific campus
building_id	ID number identifying a specific campus building
floor_id	ID number identifying a specific building floor
hashed_sta_eth_mac	Anonymized value of the client MAC address
geofence_ids	Indicates whether the client is located in a GeoFence region
loc_algorithm	Algorithm indicating how the (X,Y) coordinates are populated: • **ALGORITHM_TRIANGULATION**: Triangulation for ALE 1.3.x and earlier • **ALGORITHM_AP_PLACEMENT**: Single AP location for ALE 1.3.x and earlier • **ALGORITHM_CALIBRATION**: Calibration mode • **ALGORITHM_ESTIMATION**: Estimation mode • **ALGORITHM_LOW_DENSITY**: Location from a low AP density area; single AP location for ALE 2.0 onwards

Figure 87 *Determining the X,Y Coordinates of a Client*



The Location API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "location"
Protobuf schema

message location {
   optional mac_address sta_eth_mac = 1;
   optional float sta_location_x = 2;
   optional float sta_location_y = 3;
   optional uint32 error_level = 7;
   optional bool associated = 8;
   optional bytes campus_id = 9;
   optional bytes building_id = 10;
   optional bytes floor_id = 11;
   optional bytes hashed_sta_eth_mac = 12;
   repeated bytes geofence_ids = 13;
   optional algorithm loc_algorithm = 14;
}
```

Presence

The Presence message is sent as soon as an AP radio hears a client MAC address. If the client associates to the network, then the associated field is set to **true**; if the client is not associated, this is set to **false**. Removal of a client (when silent for a long time interval) and changes in the associated state are indicated by the OP_DELETE and OP_UPDATE fields.

The output for this message type displays the following information:

Table 38: *Presence API Message Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client
associated	Indicates whether the client is associated with an AP on the network
hashed_sta_eth_mac	Anonymized value of the client MAC address
ap_name	Name of the AP
radio_mac	MAC address of the radio that hears the client

The Presence API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "presence"
Protobuf schema

message presence {
   optional mac_address sta_eth_mac = 1;
   optional bool associated = 2;
   optional bytes hashed_sta_eth_mac = 3;
   optional string ap_name = 4;
   optional mac_address radio_mac = 5;
}
```

Proximity

The Proximity message indicates which AP is closest to the station, providing a rough location estimation of the client when ALE runs in context mode without maps.

The output for this message type displays the following information:

Table 39: *Proximity API Message Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client
radio_mac	MAC address of the corresponding radio
rssi_val	RSSI value at which the AP hears the station **NOTE:** Attenuation (dBm) = RSSI - 96 - client Tx power. If client power is unknown, a value of 10 is used.
ap_name	Name of the AP
hashed_sta_eth_mac	Anonymized value of the client MAC address

The Proximity API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "proximity"
Protobuf schema

message proximity {
   optional mac_address sta_eth_mac = 1;
   optional mac_address radio_mac = 2;
   optional sint32 rssi_val = 3;
   optional string ap_name = 4;
   optional bytes hashed_sta_eth_mac = 5;
}
```

Radio

The Radio message sends information about each radio on a newly added AP.

The output for this message type displays the following information:

Table 40: *Radio API Message Parameters*

Output Parameter	Definition
ap_eth_mac	MAC address of the AP
radio_bssid	BSSID of the radio
radio_mode	Specifies the radio mode on an AP; each mode provides a different function
phy_type	Physical radio type: • 802.11a • 802.11a-HT-40 • 802.11b/g • 802.11b/g-HT-20
ht_type	Type of high-throughput traffic sent by the AP:

Output Parameter	Definition
	• **HT-20mhz**: The AP radio uses a single 20 mHz channel • **HT-40mhz**: The AP radio uses a 40 MHz channel pair comprised of two adjacent 20 MHz channels.

The Radio API output schema is as follows:

```
ØMQ endpoint     "tcp://localhost:7779"
ØMQ message filter       "radio"
Protobuf schema

message radio {
   enum radio_mode {
      RADIO_MODE_AP = 0;
      RADIO_MODE_MESH_PORTAL = 1;
      RADIO_MODE_MESH_POINT = 2;
      RADIO_MODE_AIR_MONITOR = 3;
      RADIO_MODE_SPECTRUM_SENSOR = 4;
      RADIO_MODE_UNKNOWN = 5;
   }
   enum phy_type {
      PHY_TYPE_80211B = 0;
      PHY_TYPE_80211A = 1;
      PHY_TYPE_80211G = 2;
   }
   enum ht_type {
      HTT_NONE = 0;
      HTT_20MZ = 1;
      HTT_40MZ = 2;
      HTT_VHT_20MZ = 3;
      HTT_VHT_40MZ = 4;
      HTT_VHT_80MZ = 5;
   }
   optional mac_address ap_eth_mac = 1;
   optional mac_address radio_bssid = 2;
   optional radio_mode mode = 4;
   optional phy_type phy = 5;
   optional ht_type ht = 6;
}
```

Radio Statistics

The Radio Statistics message returns information on network coverage, traffic between APs and devices, client performance, and potential causes of poor performance or connectivity. This API is available in both controller and IAP deployments.

The output for this message type displays the following information, but not all fields are present in both controller and IAP deployments. Refer to Table 41 and the example ouputs to determine which fields are present in each mode.

Table 41: *Radio Statistics API Message Parameters*

Output Parameter	Definition
ap_eth_mac	MAC address of the AP (only available in controller deployments)
radio_number	Radio interference number for the given AP (only available in controller deployments)

Output Parameter	Definition
channel	Assigned channel for the radio (only available in controller deployments)
phy_type	Physical radio type: • 802.11a • 802.11a-HT-40 • 802.11b/g • 802.11b/g-HT-20 (only available in controller deployments)
radio_mode	Configured radio mode: • AP • Mesh • Monitor (only available in controller deployments)
noise_floor	Noise floor for the given radio (available in both controller and IAP deployments)
tx_power	Transmitting power for the given radio, in dB (available in both controller and IAP deployments)
channel_utilization	Channel utilization for the given radio (only available in controller deployments)
rx_channel_utilization	Channel utilization for incoming packets (only available in controller deployments)
tx_channel_utilization	Channel utilization for outgoing packets (only available in controller deployments)
tx_received	Total number of 802.11 frames received for transmission (only available in controller deployments)
tx_transmitted	Total number of transmitted 802.11 frames (only available in controller deployments)
tx_dropped	Total number of dropped 802.11 frames (available in both controller and IAP deployments)
tx_data_received	Total number of 802.11 data frames received for transmission (only available in controller deployments)
tx_data_transmitted	Total number of transmitted 802.11 data frames (only available in controller deployments)
tx_data_retried	Total number of retried 802.11 data frames (only available in controller deployments)
rx_frames	Total number of received 802.11 frames (only available in controller deployments)
rx_retried	Total number of retried 802.11 frames (only available in controller deployments)
rx_data_frames	Total number of received 802.11 data frames (only available in controller deployments)
rx_data_retried	Total number of received 802.11 data frames that are retried (only available in controller deployments)
rx_frame_errors	Frames that the radio cannot decode (available in both controller and IAP deployments)
data_traffic_type_stats	Data traffic type performance statistics (only available in controller deployments)
data_prio_stats	Data priority performance statistics (only available in controller deployments)

Output Parameter	Definition
data_rate_stats	Data rate bucket performance statistics (only available in controller deployments)
radio_mac	MAC address of the radio (only available in IAP deployments)
tx_data_bytes	Amount of transmitted data traffic, in bytes (only available in IAP deployments)
rx_data_bytes	Amount of received data traffic, in bytes (only available in IAP deployments)

The Radio Statistics API output schema is as follows:

```
ØMQ message filter "stats_radio"
Protobuf schema:

message stats_radio {
   enum radio_mode {
      RADIO_MODE_AP = 0;
      RADIO_MODE_MESH_PORTAL = 1;
      RADIO_MODE_MESH_POINT = 2;
      RADIO_MODE_AIR_MONITOR = 3;
      RADIO_MODE_SPECTRUM_SENSOR = 4;
      RADIO_MODE_UNKNOWN = 5;
   }

   // MAC address of the Access Point
   optional mac_address ap_eth_mac = 1;
   // Radio Interface Number on the given AP (0, 1)
   optional uint32 radio_number = 2;
   // Assigned Channel for the Radio
   optional uint32 channel = 3;
   // 802.11 Phy type (a/b/g)
   optional phy_type phy = 4;
   // Configured Mode Radio (ap/mesh/monitor)
   optional radio_mode mode = 5;
   // Instantaneous Noise Floor for the given radio
   optional uint32 noise_floor = 7;
   // Instantaneous TX Power LEvel for the given radio
   optional uint32 tx_power = 8;
   // Instantaneous Channel Utilization for the given radio
   optional uint32 channel_utilization = 9;
   // Instantaneous RX Channel Utilization for the given radio
   optional uint32 rx_channel_utilization = 10;
   // Instantaneous TX Channel Utilization for the given radio
   optional uint32 tx_channel_utilization = 11;
   // Cumulative count of all 802.11 frames received for transmission
   optional uint32 tx_received = 12;
   // Cumulative count of all 802.11 frames transmitted
   optional uint32 tx_transmitted = 13;
   // Cumulative count of all 802.11 frames dropped
   optional uint32 tx_dropped = 14;
   // Cumulative count of 802.11 data frames received for transmission
   optional uint32 tx_data_received = 15;
   // Cumulative count of 802.11 data frames transmitted
   optional uint32 tx_data_transmitted = 16;
   // Cumulative count of 802.11 data frames retried
   optional uint32 tx_data_retried = 17;
   // Cumulative count of all 802.11 frames received
   optional uint32 rx_frames = 18;
   // Cumulative count of all 802.11 frames retried
   optional uint32 rx_retried = 19;
```

```
   // Cumulative count of data 802.11 frames received
   optional uint32 rx_data_frames = 20;
   // Cumulative count of data 802.11 frames received
   optional uint32 rx_data_retried = 21;
   // Frames that Radio couldn't decode
   optional uint32 rx_frame_errors = 22;
   // Data Traffic Type stats
   repeated data_traffic_type_stats traffic_stats = 23;
   // Priority Stats
   repeated data_prio_stats prio_stats = 24;
   // Data Rate Bucket stats
   repeated data_rate_stats rate_stats = 25;
   // Radio Mac address (IAP)
   optional mac_address radio_mac = 27;
   // Trasmitted data in bytes (IAP)
   optional uint64 tx_data_bytes = 28;
   // recieved data in bytes (IAP)
   optional uint64 rx_data_bytes = 29;
}
```

Radio Utilization/Histogram Statistics

The Radio Utilization/Histogram Statistics message displays information about the radio usage on an AP. This API is only available in controller deployments.

The output for this message type displays the following information:

Table 42: *Radio Utilization/Histogram API Message Parameters*

Output Parameter	Definition
ap_eth_mac	MAC address of the AP
radio_number	Radio interface number of the AP (0,1)
util_stats	List of all utilization statistics
util_stat_type	Type of histogram/utilization statistic

The Radio Utilization/Histogram API output schema is as follows:

```
ØMQ message filter "utilization_stats_radio"
Protobuf schema:

message utilization_stats_radio {
   // MAC Address of the Access Point
   optional mac_address ap_eth_mac = 1;
   // Radio Interface Number on the given AP (0, 1)
   optional uint32 radio_number = 2;
   // List of all the Utilization Stats
   repeated util_stats ustats = 3;
}
message util_stats {
   // Histogram/Utilization Stat Type
   enum util_stat_type {
      UTIL_STAT_TYPE_CHANNEL = 0;
      UTIL_STAT_TYPE_CHANNEL_TX = 1;
      UTIL_STAT_TYPE_CHANNEL_RX = 2;
      UTIL_STAT_TYPE_QUEUE_SWTX = 3;
      UTIL_STAT_TYPE_QUEUE_BE = 4;
      UTIL_STAT_TYPE_QUEUE_BK = 5;
      UTIL_STAT_TYPE_QUEUE_VI = 6;
```

```
    UTIL_STAT_TYPE_QUEUE_VO = 7;
    UTIL_STAT_TYPE_QUEUE_BCMC = 8;
    UTIL_STAT_TYPE_QUEUE_ATIM = 9;
  }
```

Station RSSI

The Station RSSI message displays the Received Signal Strength Indication (RSSI) value for a selected station at a specific point in time. This value is heard by an AP radio and identified by the radio_mac field.

This API is available as a debug tool and should not be used.

The output for this message type displays the following information:

Table 43: *Station RSSI API Message Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client
radio_mac	MAC address of the AP radio that hears the RSSI value
rssi_val	RSSI value at which the AP hears the station **NOTE:** Attenuation (dBm) = RSSI - 96 - client Tx power. If client power is unknown, a value of 10 is used.
associated	Indicates whether the client is associated with an AP on the network
age	Age of the RSSI value (for example, Age=0 indicates the most current RSSI value)
noise_floor	Noise floor for the radio that hears the RSSI value
assoc_bssid	BSSID of the AP associated with the station

The Station RSSI API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7778"
ØMQ message filter      "sta_rssi"
Protobuf schema

message sta_rssi {
   optional mac_address sta_eth_mac = 1;
   optional mac_address radio_mac = 2;
   optional sint32 rssi_val = 3;
   optional bool associated = 4;
   optional int32 age = 5;
   optional int32 noise_floor = 6;
   optional mac_address assoc_bssid = 7;
}
```

Security

The Security message provides information related to user authentication, strengthening security and ensuring only legitimate users are accessing the network. This API is only available in controller deployments.

The output for this message type displays the following information:

Table 44: *Security API Message Parameters*

Output Parameter	Definition
security_msg_type	Type of security message used for client authentication: • auth_srvr_timeout • macauth • captive_portal • wpa_key_handshake • dot1x
timestamp	Time that the authentication server times out
start_timestamp	Time that authentication begins
finish_timestamp	Time that authentication ends
station_mac	MAC address of the client attempting to authenticate to the network
username	A unique identifier for the client
bssid	BSSID of the client
authtype	Type of authentication used to authenticate a client into the network
result	Indicates whether authentication is successful
reason	Reason for a failed authentication attempt
trigger_reason	Reason the system triggers another authentication attempt
retry_cnt	Number of times a client attempts to authenticate to the network after a timeout
server_retry_cnt	Number of times the server attempts to authenticate a client to the network
client_retry_cnt	Number of times the client attempts to authenticate to the network
key1_retry_cnt	The number of times a client attempts to authenticate to the network using WPA key 1
key3_retry_cnt	The number of times a client attempts to authenticate to the network using WPA key 3
serverip	IP address of the server
userip	IP address of the client
srvr_elapsed_time	Elapsed time it takes the server to authenticate a client to the network
clnt_elapsed_time	Elapsed time is takes the client to authenticate to the network
replay_counter_mismatch	Displays mismatches in replay counter errors, indicating the response to an initial packet is not accepted if a later version of the packet is also sent out

Output Parameter	Definition
util_stat_type	The type of utilization statistic used in authentication
max	Maximum length of the WPA passphrase
min	Minimum length of the WPA passphrase

The Security API output schema is as follows:

```
ØMQ message filter "security_message"
Protobuf schema:

message security_message {
   enum security_msg_type {
      AUTH_SRVR_TIMEOUT_MSG = 0;
      MACAUTH_MSG = 1;
      CAPTIVE_PORTAL_MSG = 2;
      WPA_KEY_HANDSHAKE_MSG = 3;
      DOT1X_MSG = 4;
   }
   optional security_msg_type msg_type = 1
   optional auth_srvr_timeout auth_srvr_timeout = 2;
   optional macauth macauth = 3;
   optional captive_portal captive_portal = 4;
   optional wpa_key_handshake wpa_key_handshake = 5;
   optional dot1x dot1x = 6;
}
message dot1x {
   optional uint64 start_timestamp  = 1;
   optional uint64 finish_timestamp = 2;
   optional mac_address station_mac = 3;
   optional string username         = 4;
   optional mac_address bssid       = 5;
   optional uint8 result            = 6;
   optional uint8 reason            = 7;
   optional uint8 server_retry_cnt  = 8;
   optional uint8 client_retry_cnt  = 9;
   optional ip_address serverip     = 10;
   optional uint32 srvr_elapsed_time= 11;
   optional uint32 clnt_elapsed_time= 12;
}

message wpa_key_handshake {
   optional uint64 start_timestamp         = 1;
   optional uint64 finish_timestamp        = 2;
   optional mac_address station_mac        = 3;
   optional mac_address bssid              = 4;
   optional uint8 result                   = 5;
   optional uint8 trigger_reason           = 6;
   optional uint8 reason                   = 7;
   optional uint8 key1_retry_cnt           = 8;
   optional uint8 key3_retry_cnt           = 9;
   optional uint32 replay_counter_mismatch = 10;
}

   optional util_stat_type type = 1;
   optional uint32 bucket1 = 2;
   optional uint32 bucket2 = 3;
   optional uint32 bucket3 = 4;
   optional uint32 bucket4 = 5;
```

```
   optional uint32 bucket5 = 6;
   optional uint32 max = 7;
   optional uint32 min = 8;
   optional uint32 curr = 9;
   optional uint64 stat = 10;
}
message captive_portal {
   optional uint64 start_timestamp  = 1;
   optional uint64 finish_timestamp = 2;
   optional mac_address station_mac = 3;
   optional string username         = 4;
   optional mac_address bssid       = 5;
   optional uint8 result            = 6;
   optional uint8 reason            = 7;
   optional uint8 server_retry_cnt  = 8;
   optional ip_address serverip     = 9;
   optional ip_address userip       = 10;
}

message macauth {
   optional uint64 start_timestamp  = 1;
   optional uint64 finish_timestamp = 2;
   optional mac_address station_mac = 3;
   optional mac_address bssid       = 4;
   optional uint8 result            = 5;
   optional uint8 reason            = 6;
   optional uint8 server_retry_cnt  = 7;
   optional ip_address serverip     = 8;
}

message auth_srvr_timeout {
   optional uint64 timestamp  = 1;
   optional mac_address station_mac = 2;
   optional mac_address bssid       = 3;
   optional uint8 authtype            = 4;
   optional uint8 retry_cnt  = 5;
   optional ip_address userip       = 6;
   optional ip_address serverip     = 7;
}
```

Station

The Station message is sent when a user associates to the WLAN. This API is available in both controller and IAP deployments.

The station is removed from the ALE table if the controllers also remove the station from its user-table. The user idle-timeout for removal is 5 minutes by default and is configured using "aaa timers idle-timeout x", where x is the number of minutes until the user is idled out of the system.

The output for this message type displays the following information:

Table 45: *Station API Message Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client station
username	Corresponding username from the user table on the WLAN controller or IAP

Output Parameter	Definition
role	Name of the user role currently assigned to the client. This is applicable to only authenticated users
bssid	BSSID of the client
device_type	Type of device used by the client • Windows 7 • iOS devices
sta_ip_address	IP address of the client station
hashed_sta_eth_mac	Anonymized value of the client MAC address
hashed_sta_ip_address	Anonymized value of the client IP address

The Station API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "station"
Protobuf schema

message station {
   optional mac_address sta_eth_mac = 1;
   optional string username = 2;
   optional string role = 3;
   optional mac_address bssid = 4;
   optional string device_type = 5;
   optional ip_address sta_ip_address = 6;
   optional bytes hashed_sta_eth_mac = 7;
   optional bytes hashed_sta_ip_address = 8;
}
```

Station Statistics

The Station Statistics message returns performance information for a given station after a client associates to the station and successfully authenticates to the network. This API is available in both controller and IAP deployments, but not all fields are present in both deployments. Refer to the schema below to determine which fields are present in each mode.

The output for this message type displays the following information:

Table 46: *Station Statistics API Message Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client station
ap_eth_mac	MAC address of the AP associated with the station
bssid	BSSID of the client station
snr	Signal-to-noise ratio used to measure the station signal strength
tx_received	Total number of 802.11 frames received for transmission **NOTE:** Only available in controller deployments
tx_transmitted	Total number of transmitted 802.11 frames

Output Parameter	Definition
	NOTE: Only available in controller deployments
tx_dropped	Total number of dropped 802.11 frames **NOTE:** Only available in controller deployments
tx_data_received	Total number of 802.11 data frames received for transmission **NOTE:** Only available in controller deployments
tx_data_transmitted	Total number of transmitted 802.11 data frames **NOTE:** Only available in controller deployments
tx_data_retried	Total number of retried 802.11 data frames **NOTE:** Only available in controller deployments
rx_data_received	Total number of received 802.11 data frames **NOTE:** Only available in controller deployments
rx_data_retried	Total number of received 802.11 data frames that are retried **NOTE:** Only available in controller deployments
data_prio_stats	Data priority performance statistics **NOTE:** Only available in controller deployments
data_rate_stats	Data rate bucket performance statistics **NOTE:** Only available in controller deployments
rate	Data rate **NOTE:** Only available in controller deployments
tx_frame_count	Number of transmitted 802.11 frames **NOTE:** Only available in controller deployments
tx_byte_count	Number of transmitted bytes **NOTE:** Only available in controller deployments
rx_frame_count	Number of received 802.11 frames **NOTE:** Only available in controller deployments
rx_byte_count	Number of recieved bytes **NOTE:** Only available in controller deployments
data_prio	Priority for the data type **NOTE:** Only available in controller deployments
tx_drop_count	Number of dropped transmission frames **NOTE:** Only available in controller deployments
data_traffic_type_stats	Data traffic type performance statistics **NOTE:** Only available in controller deployments
traffic_type	Data traffic type **NOTE:** Only available in controller deployments
speed	Application usage on the station **NOTE:** Only available in IAP deployments
rx_rate	Connected data rate for received data **NOTE:** Only available in IAP deployments

Output Parameter	Definition
tx_rate	Connected data rate for transmitted data **NOTE:** Only available in IAP deployments
rx_data_bytes	Amount of received data traffic, in bytes **NOTE:** Only available in IAP deployments
tx_data_bytes	Amount of transmitted data traffic, in bytes **NOTE:** Only available in IAP deployments
ssid_up	Number of SSIDs or VAPs **NOTE:** Only available in IAP deployments
rogue_ap	Number of rogue APs detected **NOTE:** Only available in IAP deployments

The Station Statistics API output schema is as follows for a controller deployment:

```
ØMQ message filter "station_stats"
Protobuf schema:

message station_stats {
   // MAC address of the Station
   optional mac_address sta_eth_mac = 1;
   // MAC address of the AP this station is associated with
   optional mac_address ap_eth_mac = 2;
   // BSSID this station is associated with
   optional mac_address bssid = 3;
   // Signal strength
   optional uint32 snr = 4;
   // Cumulative count of all 802.11 frames received for transmission
   optional uint32 tx_received = 5;
   // Cumulative count of all 802.11 frames transmitted
   optional uint32 tx_transmitted = 6;
   // Cumulative count of all 802.11 frames dropped
   optional uint32 tx_dropped = 7;
   // Cumulative count of data 802.11 frames received for transmission
   optional uint32 tx_data_received = 8;
   // Cumulative count of data 802.11 frames transmitted
   optional uint32 tx_data_transmitted = 9;
   // Cumulative count of data 802.11 frames retried
   optional uint32 tx_data_retried = 10;
   // Cumulative count of data 802.11 frames received
   optional uint32 rx_data_received = 11;
   // Cumulative count of data 802.11 frames received
   optional uint32 rx_data_retried = 12;
   // Priorty (AC) stats for this station
   repeated data_prio_stats prio_stats = 13;
   // Data Rate Bucket stats for this station
   repeated data_rate_stats rate_stats = 14;
}
message data_rate_stats {
   optional uint32 rate = 1;
   optional uint32 tx_frame_count = 2;
   optional uint32 tx_byte_count = 3;
   optional uint32 rx_frame_count = 4;
   optional uint32 rx_byte_count = 5;
}
```

```
message data_prio_stats {
   optional data_prio prio = 1;
   optional uint32 tx_frame_count = 2;
   optional uint32 rx_frame_count = 3;
   optional uint32 tx_drop_count = 4;
}
message data_traffic_type_stats {
   optional traffic_type type = 1;
   optional uint32 tx_frame_count = 2;
   optional uint32 rx_frame_count = 3;
```

The Station Statistics API output schema is as follows for an IAP deployment:

```
ØMQ message filter "station_stats"
Protobuf schema:

message station_stats {
   // MAC address of the Station
   optional mac_address sta_eth_mac = 1;
   // MAC address of the AP this station is associated with
   optional mac_address ap_eth_mac = 2;
   // BSSID this station is associated with
   optional mac_address bssid = 3;
   // Signal strength
   optional uint32 snr = 4;
   // Application usage of the station
   optional uint32 speed = 5;
   // Station connected data rate for rec
   optional uint64 rx_rate = 6;
   // Station connected data rate for trans
   optional uint64 tx_rate = 7;
   // Station traffic passed for rec
   optional uint64 rx_data_bytes = 8;
   // Station traffic passed for trans
   optional uint64 tx_data_bytes = 9;
   // Number of SSID up or number of VAPs
   optional uint32 ssid_up = 10;
   // Number of Rouge seen
   optional uint32 rougue_ap = 11;
}
```

State Station

The State Station message provides information about the state of a station after a client associates to the station and successfully authenticates to the network. This API is only available in IAP deployments.

The output for this message type displays the following information:

Table 47: *State Station API Message Parameters*

Output Parameter	Definition
sta_eth_mac	MAC address of the client station
ap_eth_mac	MAC address of the AP associated with the station
snr	Signal strength of the station
rx_tries	Retry count for frames received by a station
tx_tries	Retry count for frames transmitted by a station

Output Parameter	Definition
phy_type	Physical radio type: • 802.11a • 802.11a-HT-40 • 802.11b/g • 802.11b/g-HT-20
security_type	Station security type based on the following values: • **0**: opensystem • **1**: static-wep • **2**: dynamic-wep • **3**: wpa-tkip • **4**: wpa_aes • **5**: wpa-psk-tkip • **6**: wpa-psk-aes • **7**: wpa2-aes • **8**: wpa2-tkip • **9**: wpa2-psk-aes

The State Station output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "state_station"
Protobuf schema

message state_station {
   // MAC address of the Station
   optional mac_address sta_eth_mac = 1;
   // MAC address of the AP this station is asso
   optional mac_address ap_eth_mac = 2;
   // Signal strength
   optional uint32 snr = 3;
   // Station retry count for rec
   optional uint32 rx_tries = 4;
   // Station retry count for trans
   optional uint32 tx_tries = 5;
   // Station Phy_type
   optional phy_type phy_type = 6;
   // Set security op_mode
   optional uint32 security_type = 7;
}
```

Virtual Access Point (VAP)

The Virtual Access Point message is sent for each virtual AP (VAP) associated with a newly added AP. This API is available in both controller and IAP deployments unless an exception has specifically been called out.

The output for this message type displays the following information:

Table 48: *VAP API Message Parameters*

Output Parameter	Definition
bssid	BSSID of the virtual AP
ssid	SSID of the virtual AP
radio_bssid	BSSID of a radio on the virtual AP

The VAP API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "virtual_access_point"
Protobuf schema

message virtual_access_point {
   optional mac_address bssid = 1;
   optional string ssid = 2;
   optional mac_address radio_bssid = 3;
```

Virtual Access Point (VAP) Statistics

The Virtual Access Point (VAP) Statistics message provides performance information for virtual APs. The VAP Statistics API is only available in controller deployments.

The output for this message type displays the following information:

Table 49: *VAP Statistics API Message Parameters*

Output Parameter	Definition
ap_eth_mac	MAC address of the AP
bssid	BSSID of the virtual AP
tx_received	Total number of 802.11 frames received for transmission
tx transmitted	Total number of transmitted 802.11 frames
tx_dropped	Total number of dropped 802.11 frames
tx_data_received	Total number of 802.11 data frames received for transmission
tx_data_transmitted	Total number of transmitted 802.11 data frames
tx_data_retried	Total number of retried 802.11 data frames
rx_frames	Number of received 802.11 frames
rx_retried	Total number of received 802.11 frames that are retried
rx_data_frames	Total number of received 802.11 data frames
rx_data_retried	Total number of received 802.11 data frames that are retried
traffic_stats	Data traffic type statistics
prio_stats	Priority statistics for the given station
rate_stats	Data rate bucket statistics for the given station

The VAP Statistics API output schema is as follows:

```
ØMQ message filter "vap_stats"
Protobuf schema:

message vap_stats {
   // MAC address of the Access Point
   optional mac_address ap_eth_mac = 1;
   // BSSID of the VAP
   optional mac_address bssid = 2;
```

```
   // Cumulative count of all 802.11 frames received for transmission
   optional uint32 tx_received = 5;
   // Cumulative count of all 802.11 frames transmitted
   optional uint32 tx_transmitted = 6;
   // Cumulative count of all 802.11 frames dropped
   optional uint32 tx_dropped = 7;
   // Cumulative count of data 802.11 frames received for transmissio
   optional uint32 tx_data_received = 8;
   // Cumulative count of data 802.11 frames transmitted
   optional uint32 tx_data_transmitted = 9;
   // Cumulative count of data 802.11 frames retried
   optional uint32 tx_data_retried = 10;
   // Cumulative count of all 802.11 frames received
   optional uint32 rx_frames = 18;
   // Cumulative count of all 802.11 frames retried
   optional uint32 rx_retried = 19;
   // Cumulative count of data 802.11 frames received
   optional uint32 rx_data_frames = 11;
   // Cumulative count of data 802.11 frames received
   optional uint32 rx_data_retried = 12;
   // Data Traffic Type stats
   repeated data_traffic_type_stats traffic_stats = 23;
   // Priorty (AC) stats for this station
   repeated data_prio_stats prio_stats = 13;
   // Data Rate Bucket stats for this station
   repeated data_rate_stats rate_stats = 14;
}
```

WebCC

The Web Content Classification (WebCC) message provides information about the types of websites clients visit when they are connected to the network. This also provides safety/security information about each website (for example, whether the website contains malicious malware, spyware, or adware).

The message filters for the WebCC API include `webcc_category` and `webcc_info`. The `webcc_category` is available in both controller and IAP deployments, while `webcc_info` is only available in controller deployments. In IAP deployments, information similar to the `webcc_info` output is available through the Visibility Record API.

The WebCC API is only used by HTTP/HTTPS.

The output for this message type displays the following information:

Table 50: *WebCC API Message Parameters*

Output Parameter	Definition
cat_id	ID number specifying the website category
category	The website category
webcc_md5	The 16 byte md5 checksum of the `webcc_url_prefix`. This field maps to the table build from the `webcc_info` feed.
webcc_cat_id	ID number specifying the WebCC category **NOTE:** Only available in controller deployments
webcc_rep_score	Reputation score for websites based on security risks (for example, malware and phishing)

Output Parameter	Definition
	NOTE: Only available in controller deployments
webcc_url_prefix	Maps the local WebCC file with a URL **NOTE:** Only available in controller deployments

The following WebCC message filter builds the `webcc_category` table. In controller deployments, this message is only sent when the website category changes. In IAP deployments, this message is sent when ALE starts up.

```
ØMQ message filter "webcc_category"
Protobuf schema:

message webcc_category {
   optional uint32 cat_id = 1;
   optional string category = 2;
}
```

The following WebCC message filter displays WebCC information relevant to controllers. In IAP deployments, WebCC information is embedded as fields in the Visibility Records API.

```
ØMQ message filter "webcc_info"
Protobuf schema:
   message webcc_info {
      optional bytes webcc_md5 = 1; //16 bytes md5 cksum of the
      //webcc_url_prefix
      optional uint32 webcc_cat_id = 2;
      optional uint32 webcc_rep_score = 3;
      optional string webcc_url_prefix = 4;
   }
}
```

After md5 maps to the webcc_info table build, the tables built from the webcc_category or REST API can generate the string category fields.

Visibility Record

The Visibility Record message represents a unique station session for each associated client on a given application or destination.

The output for this message type displays the following information:

Table 51: *Visibility Record API Message Parameters*

Output Parameter	Definition
client_ip	IP address of the client
dest_ip	IP address of the client receiving traffic through the network
ip_proto	Protocol defining packet formatting and the addressing scheme **NOTE:** Only available in controller deployments
app_id	ID number identifying the application
tx_pkts	Number of packets transmitted by the client **NOTE:** Only available in controller deployments
tx_bytes	Number of bytes transmitted by the client

Output Parameter	Definition
rx_pkts	Number of packets received by the client **NOTE:** Only available in controller deployments
rx_bytes	Number of bytes received by the client
hashed_client_ip	Anonymized value of the client IP address
ap_mac	MAC address of the access point **NOTE:** Only available in IAP deployments
cc_cat_id	ID number specifying the WebCC category **NOTE:** Only available in IAP deployments
cc_rep_score	Reputation score for websites based on security risks (for example, malware and phishing) **NOTE:** Only available in IAP deployments

The Visibility Record API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "visibility_rec"
Protobuf schema:

message visibility_rec {
   enum ip_protocol {
      IP_PROTOCOL_VAL_6 = 6;
      IP_PROTOCOL_VAL_17 = 17;
   }
   optional ip_address client_ip = 1;
   optional ip_address dest_ip = 2;
   optional ip_protocol ip_proto = 3;
   optional uint32 app_id = 4;
   optional uint64 tx_pkts = 5;
   optional uint64 tx_bytes = 6;
   optional uint64 rx_pkts = 7;
   optional uint64 rx_bytes = 8;
   optional bytes hashed_client_ip = 9;
   optional mac_address ap_mac = 10;
      optional uint32 cc_cat_id = 13;
      optional uint32 cc_rep_score = 14;
   }
}
```

The cc_cat_id and cc_rep_score parameters are not available for applications or sessions without a valid/extractable HTTP/HTTPS URL or URI. These sessions only contain an app-id.

Controller

The Controller message returns the list of controllers within a network.

The output for this message type displays the following information:

Table 52: *Controller API Message Parameter*

Output Parameter	Definition
ip_address	IP address of the controller

The Controller API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "controller"
Protobuf schema

message controller {
   optional ip_address controller_ip = 1;
}
```

OP_UPDATE and OP_DELETE are never sent for the Controller API.

Cluster Info

The Cluster Info message returns information about clusters in an IAP deployment.

The output for this message type displays the following information:

Table 53: *Cluster Info API Message Parameters*

Output Parameter	Definition
vc_key	Unique key identifying the cluster deployment
vc_name	Name of the cluster
vc_ip	IP address of the cluster

The Cluster Info API output schema is as follows:

```
ØMQ endpoint    "tcp://localhost:7779"
ØMQ message filter      "cluster"
Protobuf schema

message cluster {
   optional string vc_key = 1;
   optional string vc_name = 2;
   optional ip_address vc_ip = 3
}
```

OP_UPDATE and OP_DELETE are never sent for the Cluster Info API.