Analytics and Location Engine 2.0

User Guide

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

The wireless network contains a wealth of information about unassociated and associated devices. The Analytics and Location Engine (ALE) is designed to gather, process, and share information from the network through a simple and standard API. ALE includes a location engine that calculates associated and unassociated device location periodically using context streams, including RSSI readings, from WLAN controllers or Instant clusters.

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, ALE features, security, polling APIs, and publish and subscribe APIs.

What's New in this Release

There are no new features in ALE 2.0.0.7.

The following feature is introduced in ALE 2.0.0.6:

Table 1: *New Features/Enhancements in ALE 2.0.0.6*

Feature	Description
PAPI Security Enhancement	ALE supports a PAPI security enhancement that regulates communication between controllers and ALE by authenticating PAPI messages using an MD5 digest/hash and secret key. This feature provides an increased level of security during ALE-controller communication.

There are no new features in ALE 2.0.0.5.

There are no new features in ALE 2.0.0.4.

The following feature is introduced in ALE 2.0.0.3:

Table 2: *New Features/Enhancements in ALE 2.0.0.3*

Feature	Description
Authentication through Clear-Pass Policy Manager	ALE supports external authentication through the Aruba Clear-Pass Policy Manager (CPPM). Users are assigned specific privileges based on the ALE role assigned through CPPM.

There are no new features in ALE 2.0.0.2.

There are no new features in ALE 2.0.0.1.

Related Documents

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

- *Analytics and Location Engine 2.0.0.x Release Notes*
- *Analytics and Location Engine 2.0.0.x API Guide*
- *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 3: *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) .	sirt@arubanetworks.com

Conventions

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

Table 4: *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 a deployment contains a larger number of clients, resulting in more location calculations, more CPU and memory may be required. Depending on the number of clients, additional memory may be required.

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

Table 5: *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 handling 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

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

- A WLAN controller or Instant cluster must be available for the 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 include:
  - 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 optimal 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. However, this creates a trade-off between ease-of-deployment and location accuracy.
- The chosen deployment 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, use **Context Mode with Device Location (Estimation or Calibration)**. This creates 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.
  - AirWave Visual RF provides a good approximation for the Voice Overlay AP count, though it may not be 100 percent accurate. Through Visual RF, select **Voice**, and verify that every location is covered by three or more APs.

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*

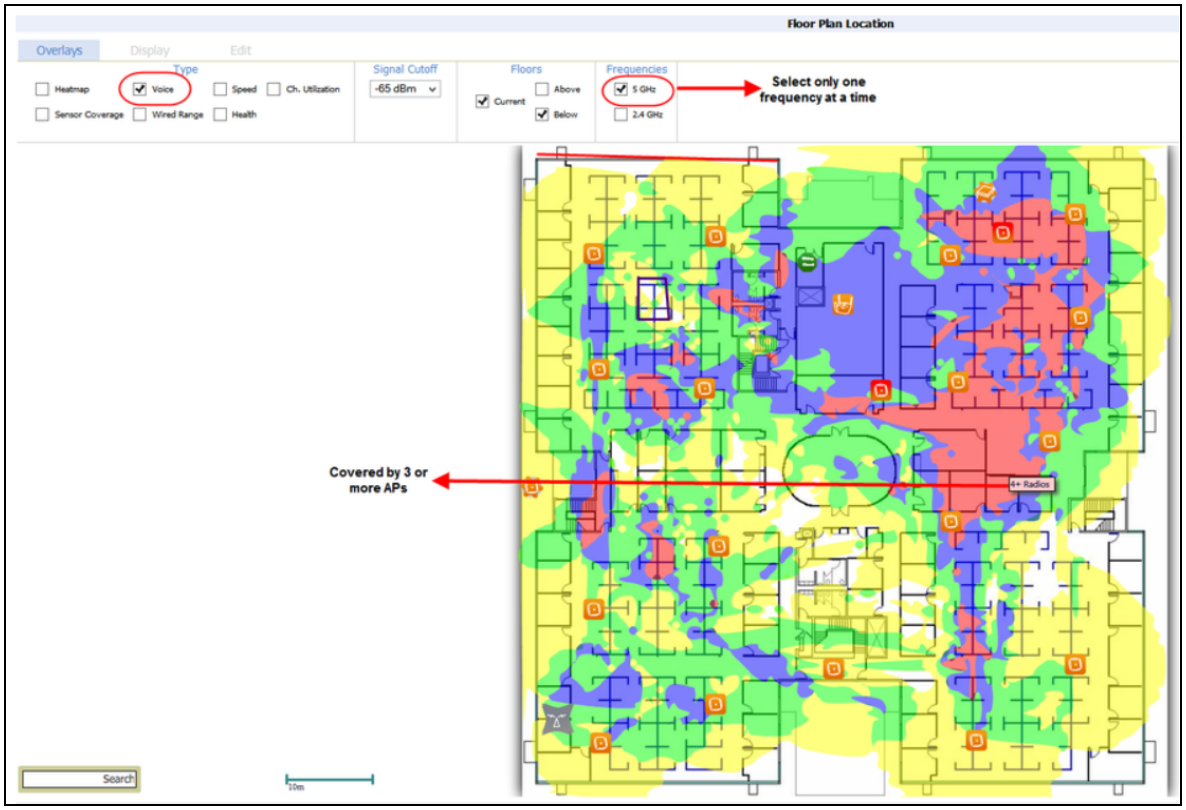
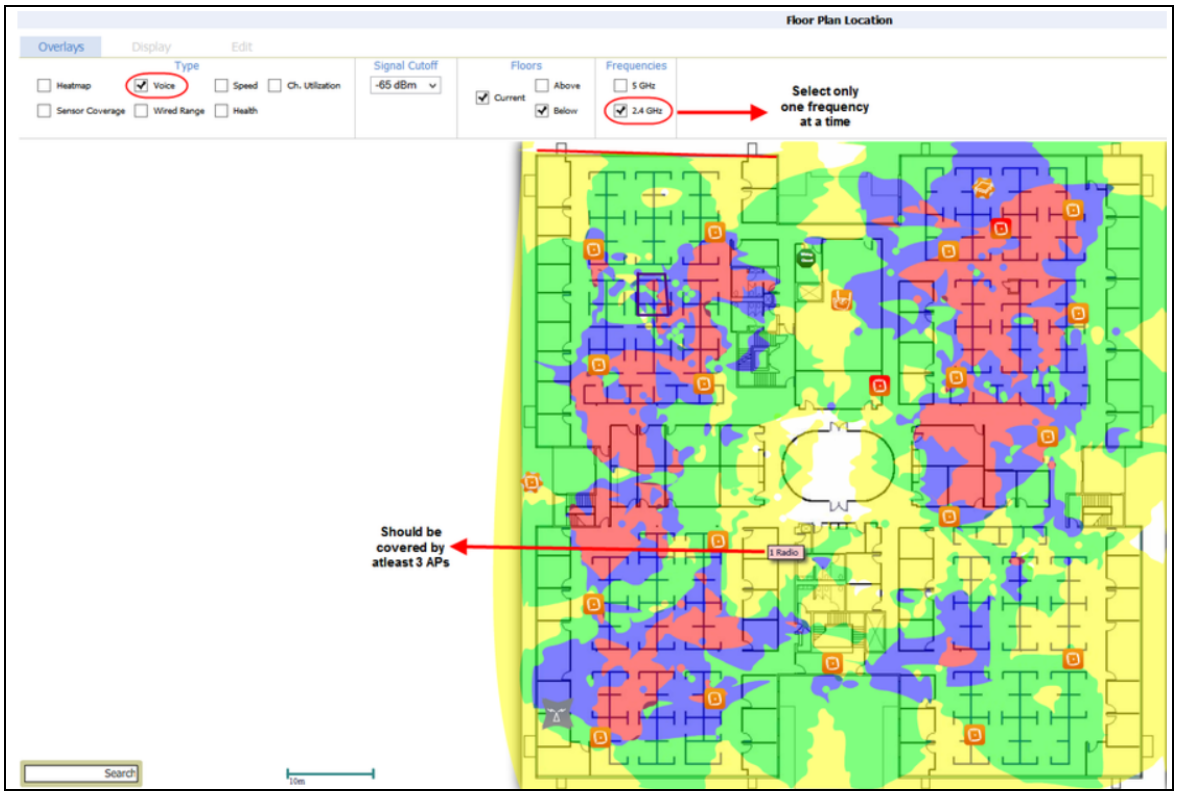
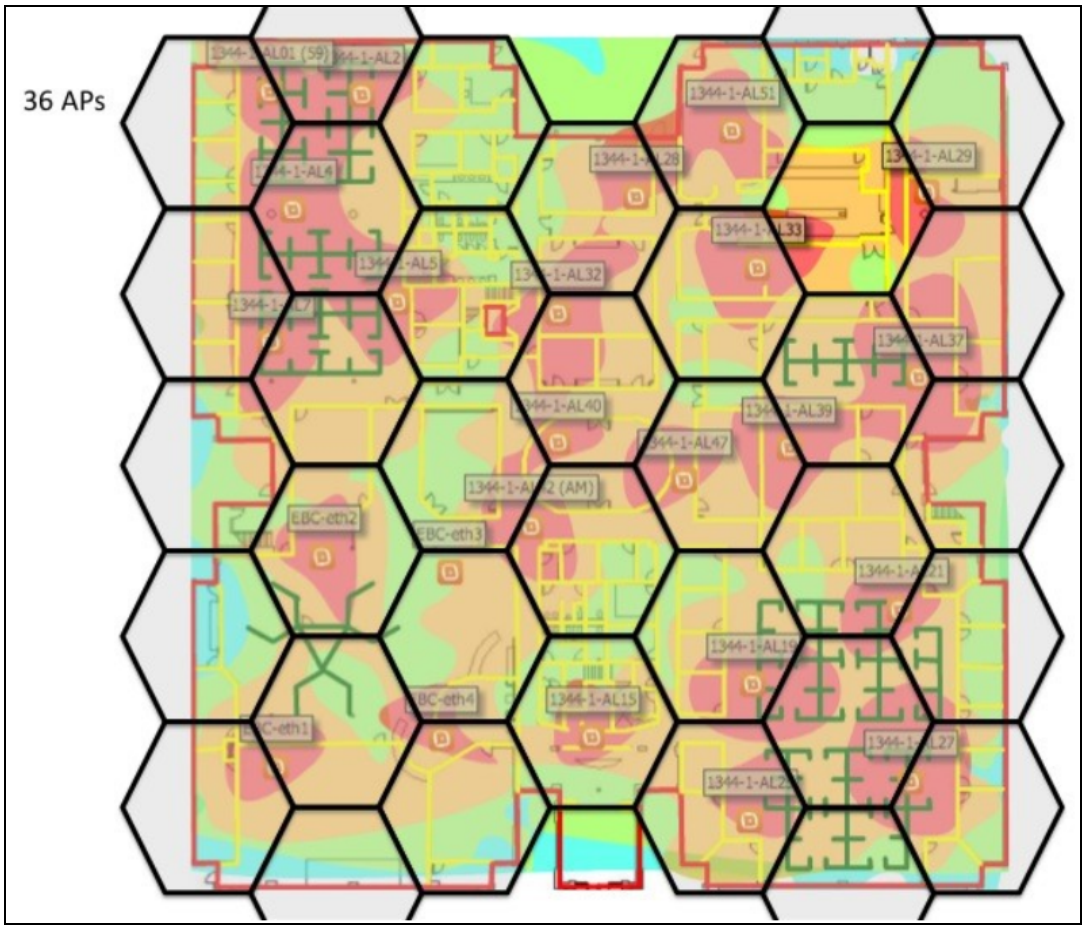


Figure 2 *Ensure Voice Overlay at frequency 24 GHz*



- APs must be deployed at the edge/boundary of a 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 the 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 6

Table 6: *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 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.

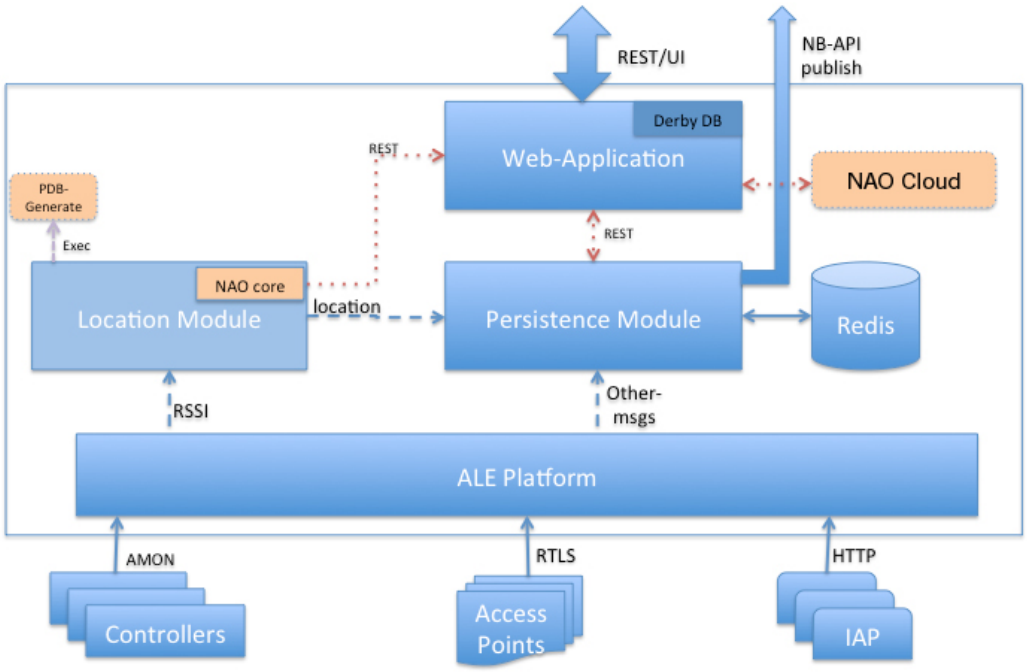
ALE Internal Architecture

ALE 2.0 operates in three modes:

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

   All context topics from the NBAPI 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 modes with device location, ALE enriches the context information with calculated client locations using a map or floor plan. There are two methods of determining device location under this context mode:
   - **Context with Device Location (Estimation)**: This method uses AirWave to approximate the location of wireless clients using maps and AP placement information. The 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. **Persistence**: The ale-persistence 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 22 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 7: *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 8: *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 9: *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

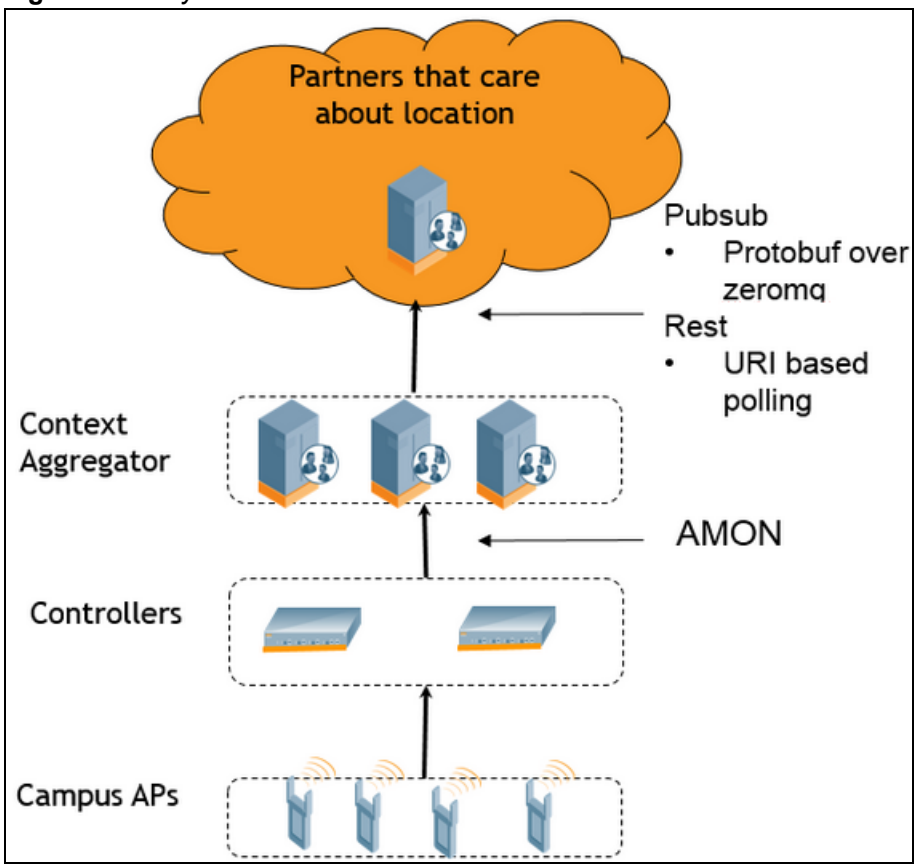
Direct use of a user's personal identifying information, which exists within specific message fields of the AMON data feed, raises privacy concerns within ALE.

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:

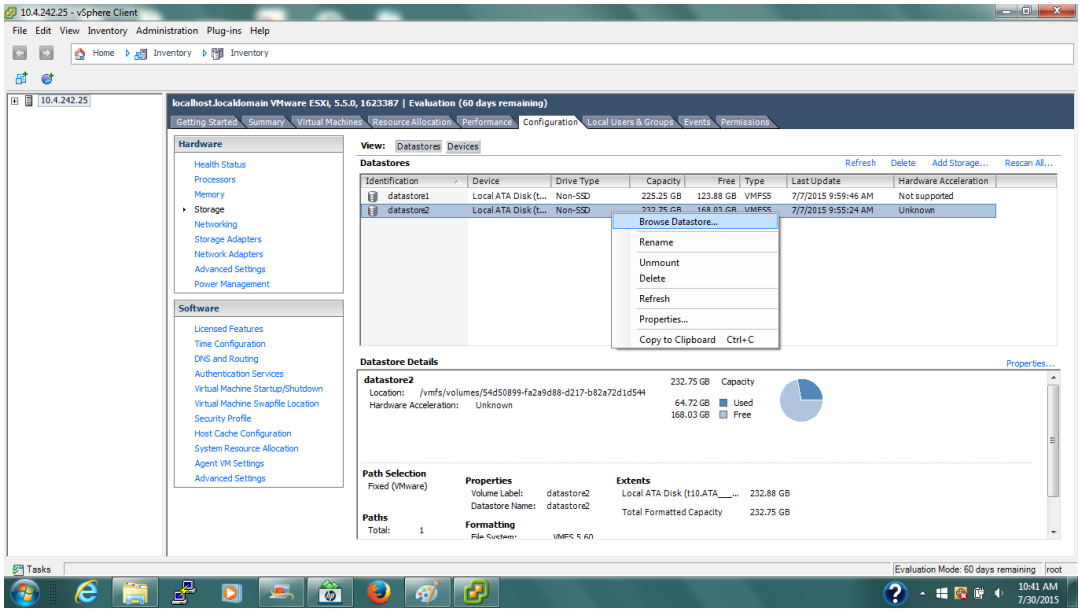
- Guest Operating System = Linux
  - Version = CentOS 6.x with Oracle JDK 1.8
- Number of Virtual Sockets = 1
  - Number of Cores per Virtual Socket = 8
- Memory Configuration = 16 GB
- Number of NICs = 1

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.

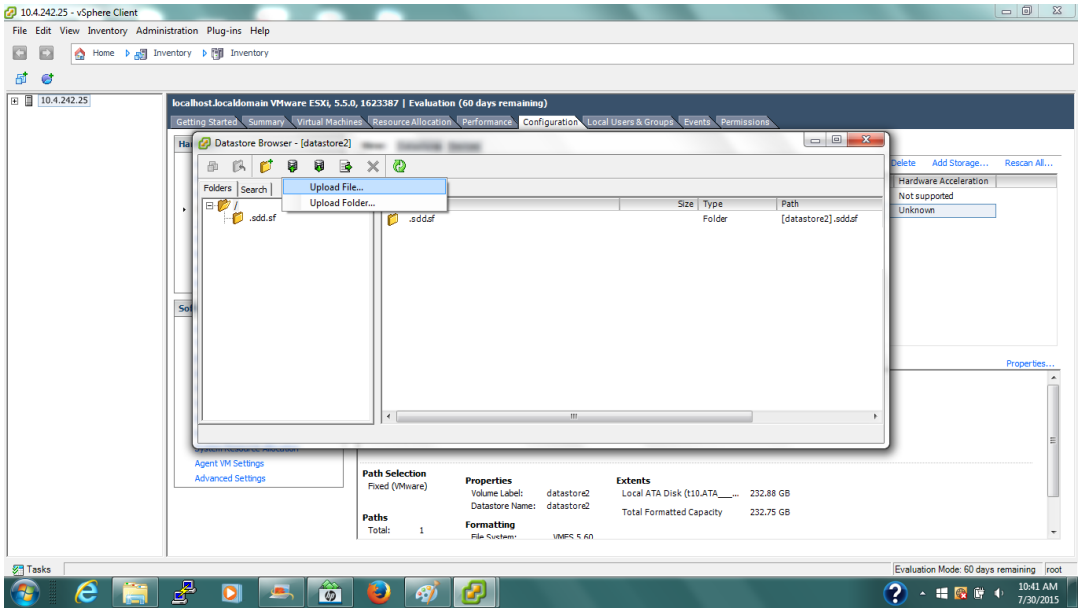
Figure 6 *The vSphere Client*



a. Click **Browse Datastore...**. The **Datastore Browser** opens.

b. Click **Upload File...**. The file manager opens.

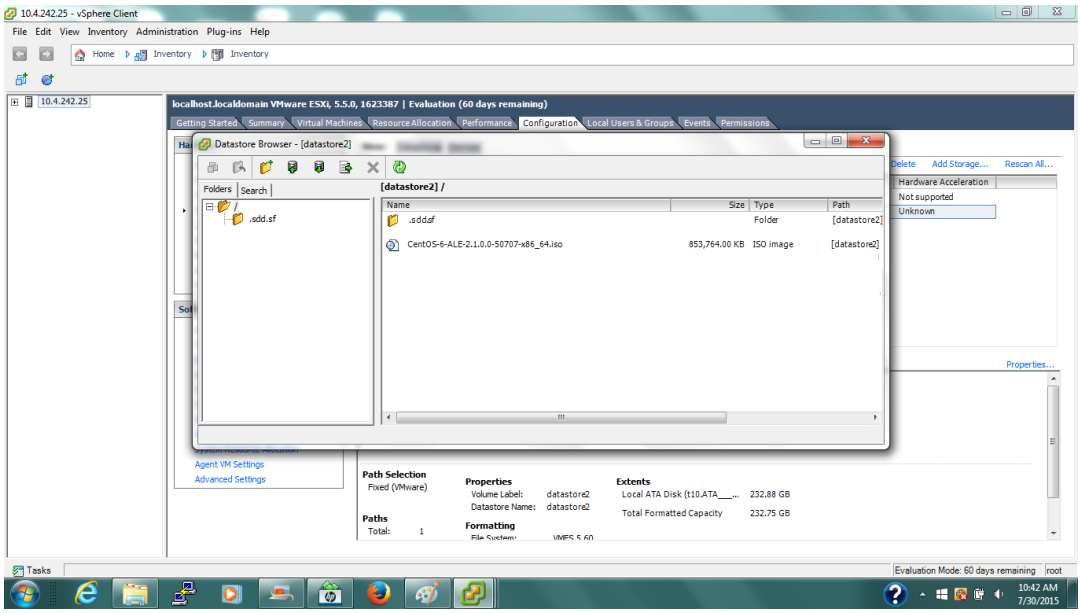
Figure 7 *Uploading the ISO File*



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*

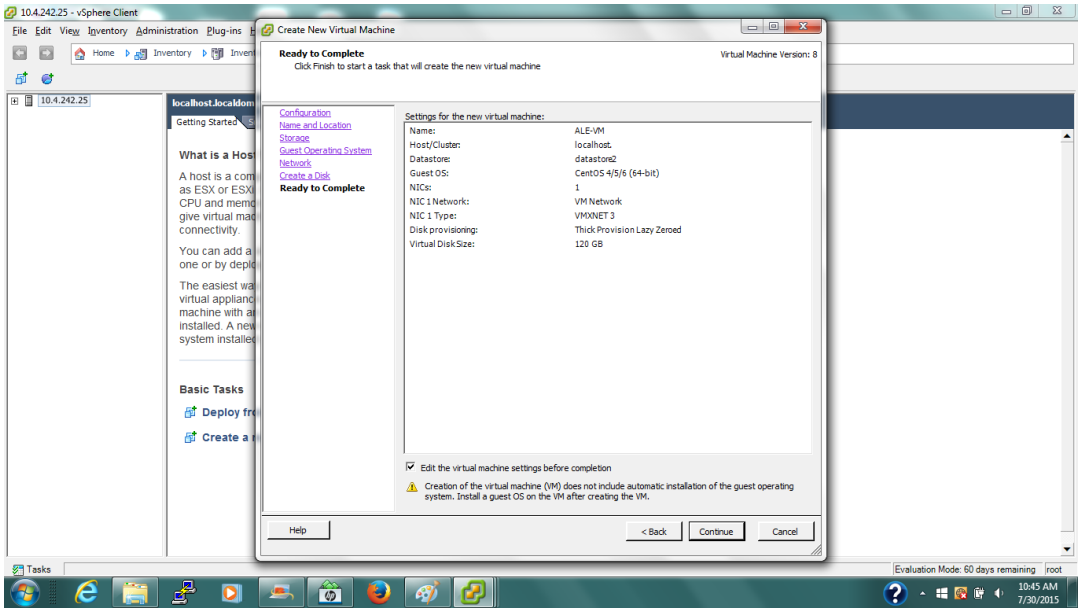
a. Under **Configuration**, select **Typical**. Click **Next**.
b. Enter a name for the ALE VM, and then click **Next**.
c. Select a datastore for the VM, and then click **Next**.

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 **Version** 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**.

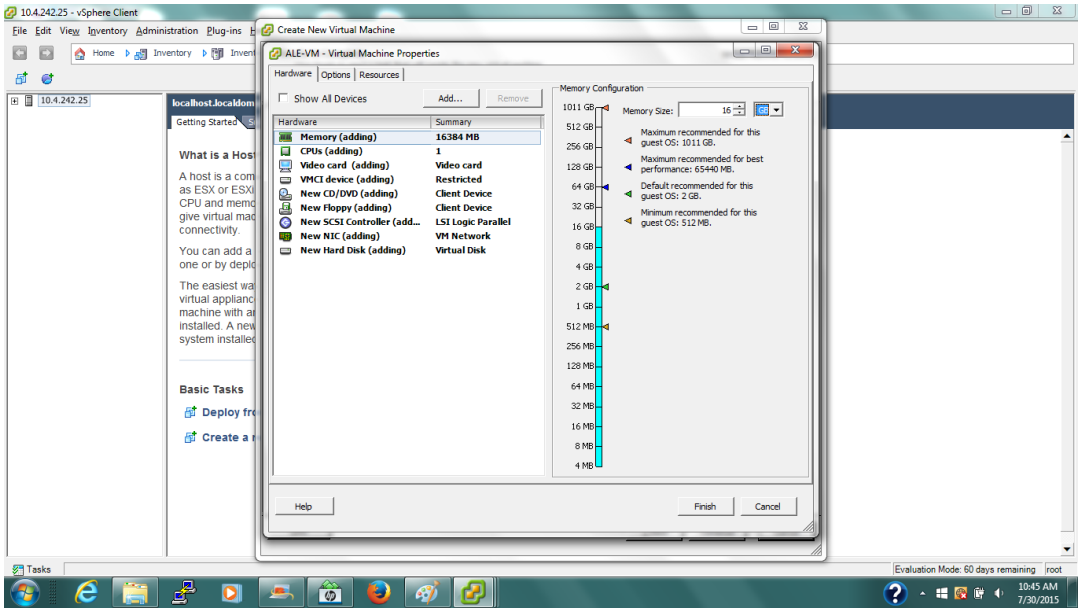
Figure 10 *Configuring the Virtual Machine*



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*

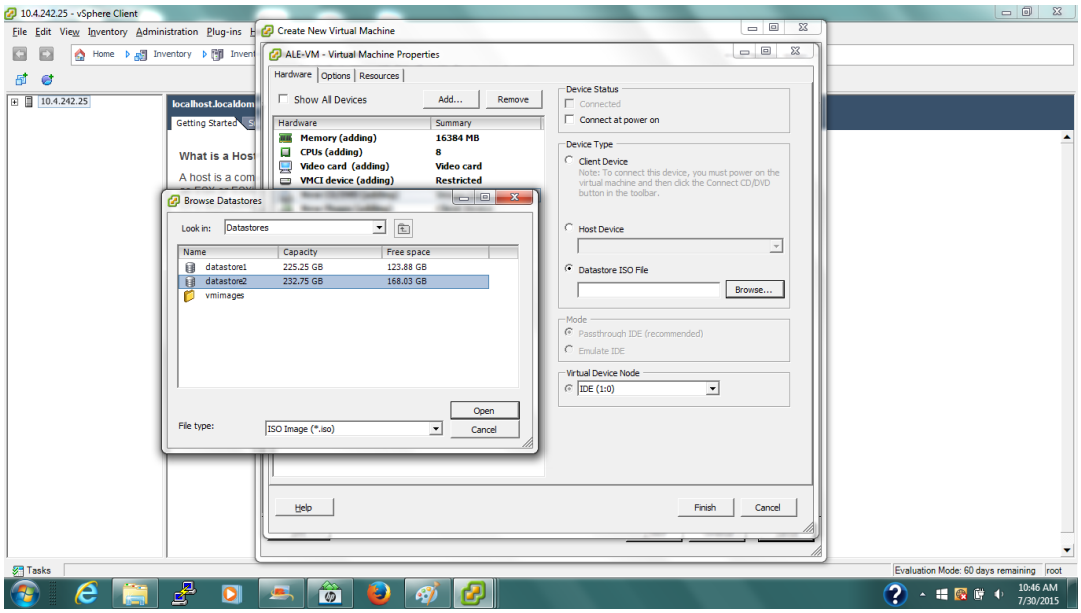


a. Under the **Hardware** tab, select **Memory (adding)** from the left window pane. Set the **Memory Size** to 16 GB.

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**.

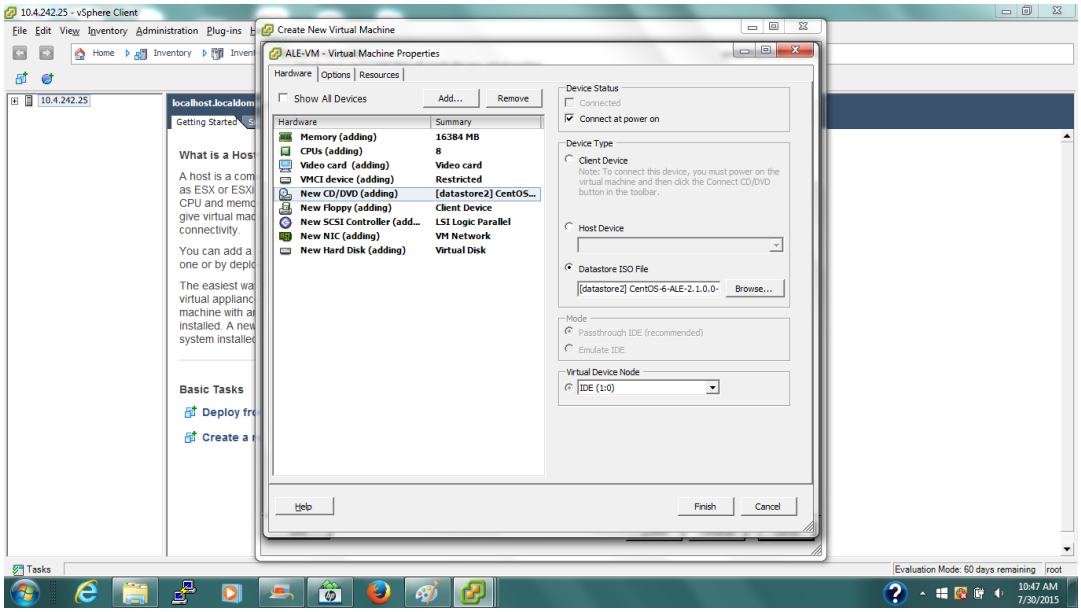
Figure 12 *Selecting Datastores for the Virtual Machine*



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

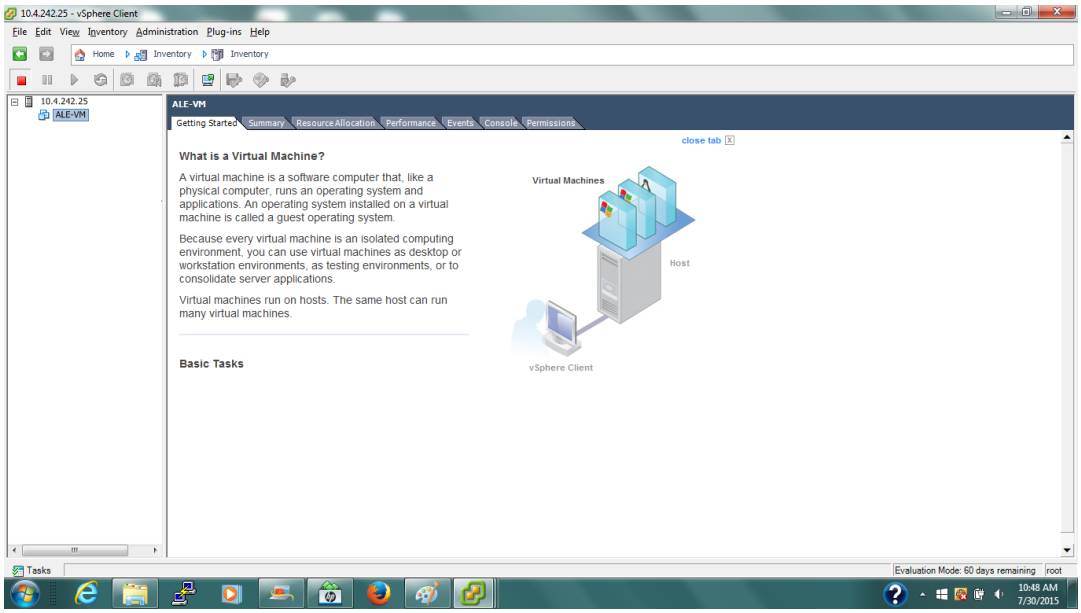
e. Select the check box for **Connect at power on** under **Device Status**, and then click **Finish**.

Figure 13 *Completed Virtual Machine Configuration*



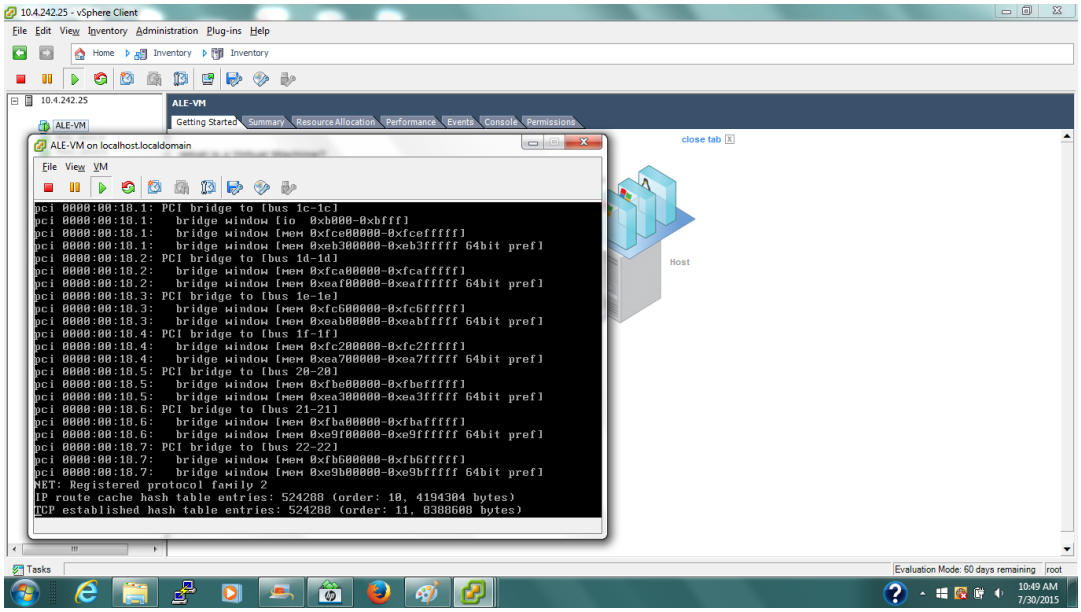
f. Once the VM is created, it appears under the server list in the vSphere client.

Figure 14 *New Virtual Machine on the vSphere Client*



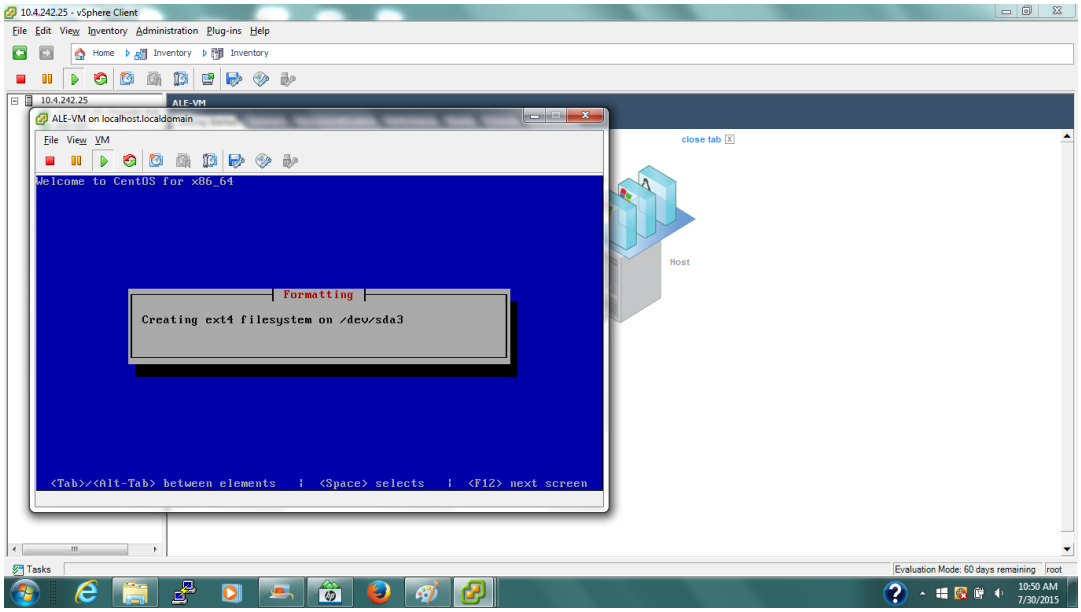
5. Select the newly created VM to begin ALE installation. The VM popup window appears.

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*

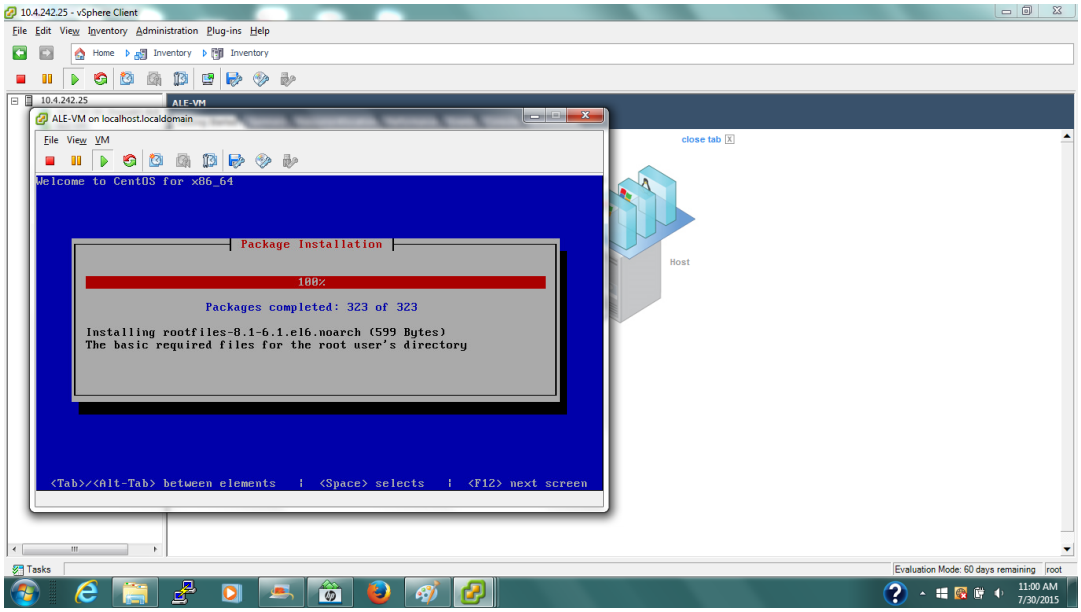
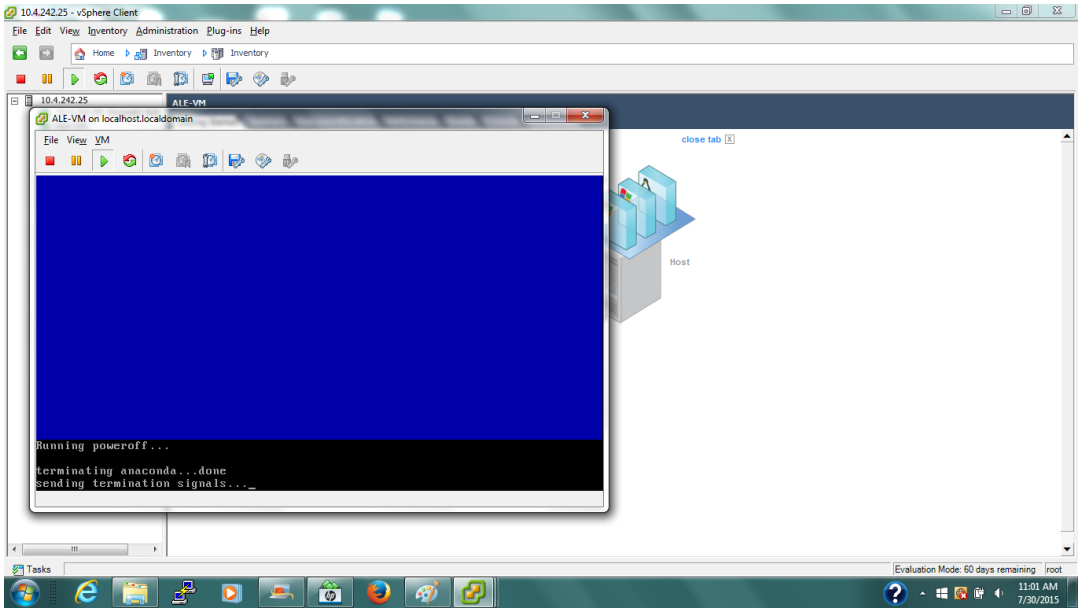
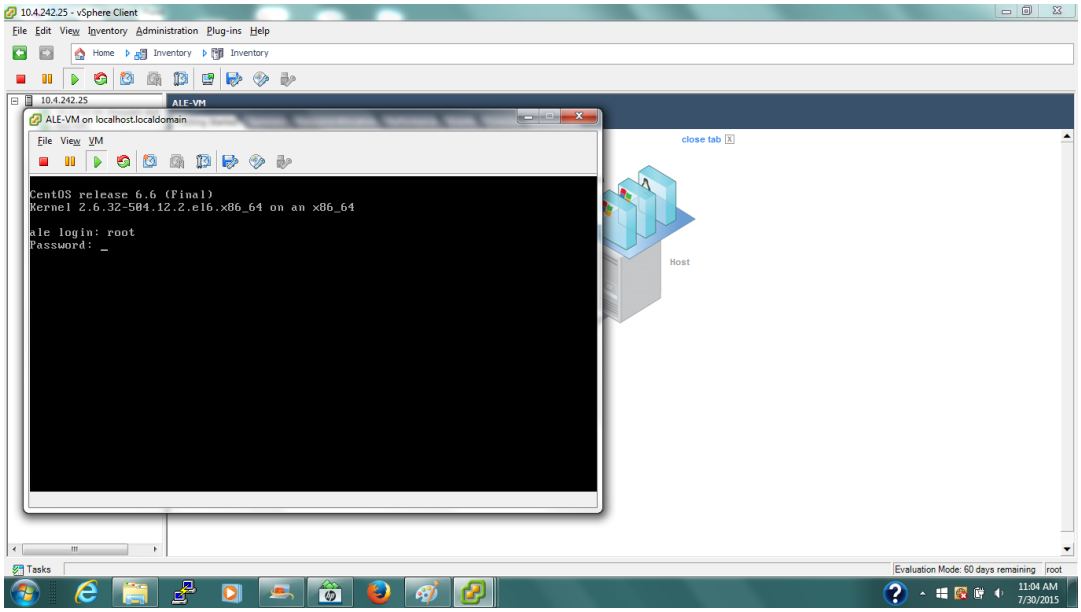


Figure 18 *ALE Poweroff*



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*



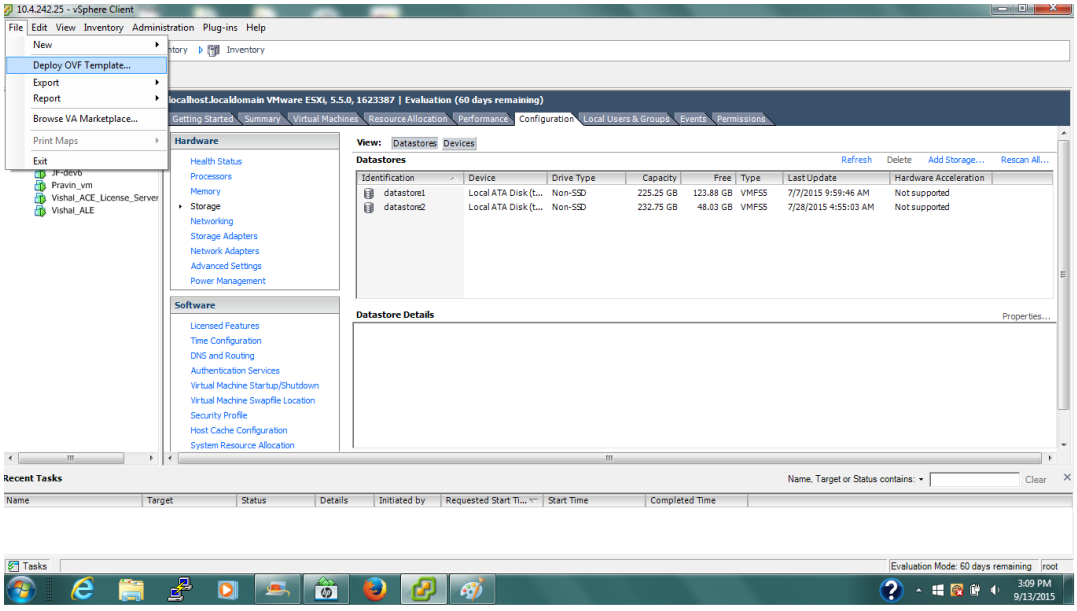
NOTE

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:

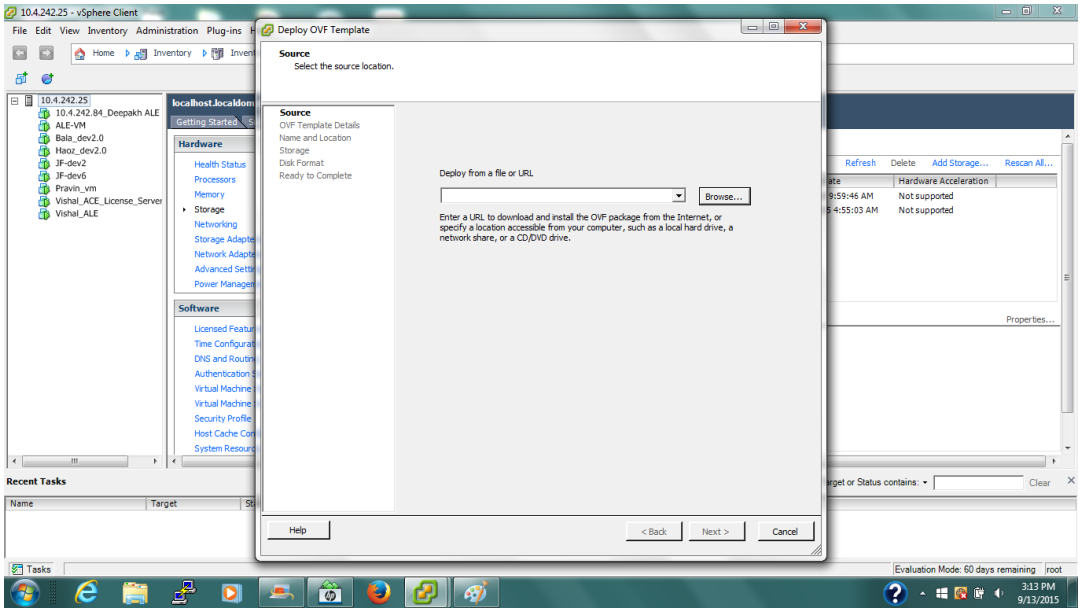
1. Launch the vSphere client.
2. Click **File > Deploy OVF Template**. The file browser opens.

Figure 20 *Deploying the OVF Template*



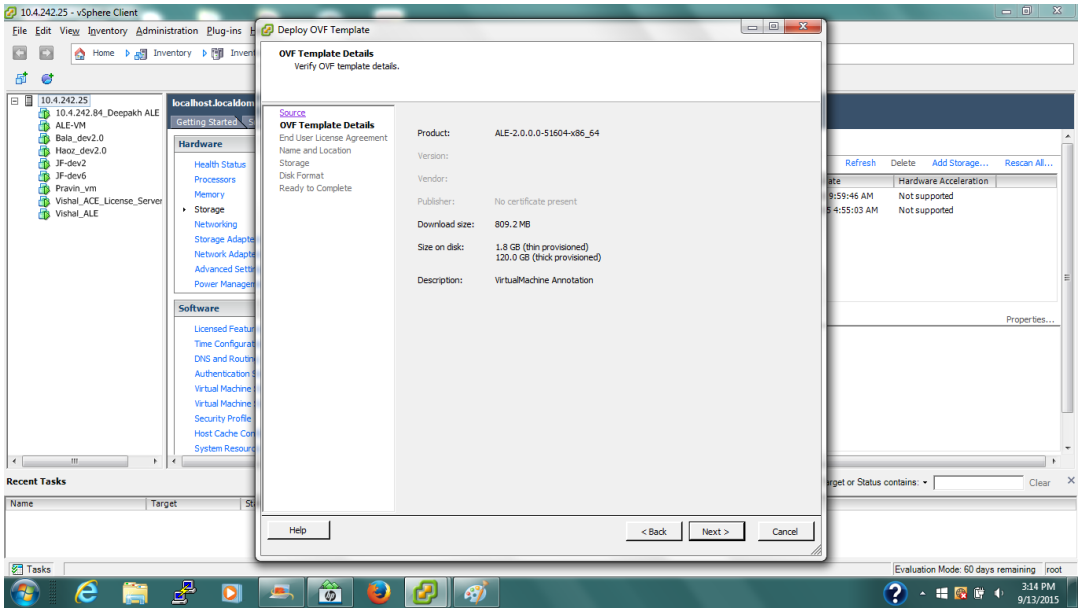
3. Under **Source**, enter the URL of the OVA file or locate and select the file using the **Browse** button.

Figure 21 *Selecting the OVA File*



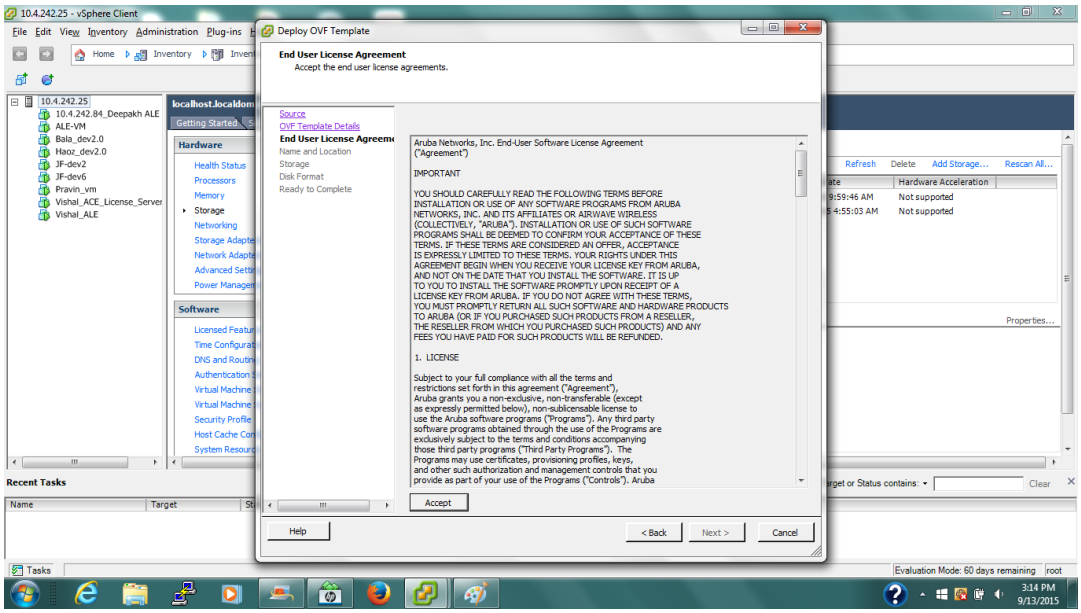
4. After selecting the OVA file, click **Next**.
5. The **OVF Template Details** tab highlights the default OVA configuration. Click **Next**.

Figure 22 *Default OVA Configuration Details*



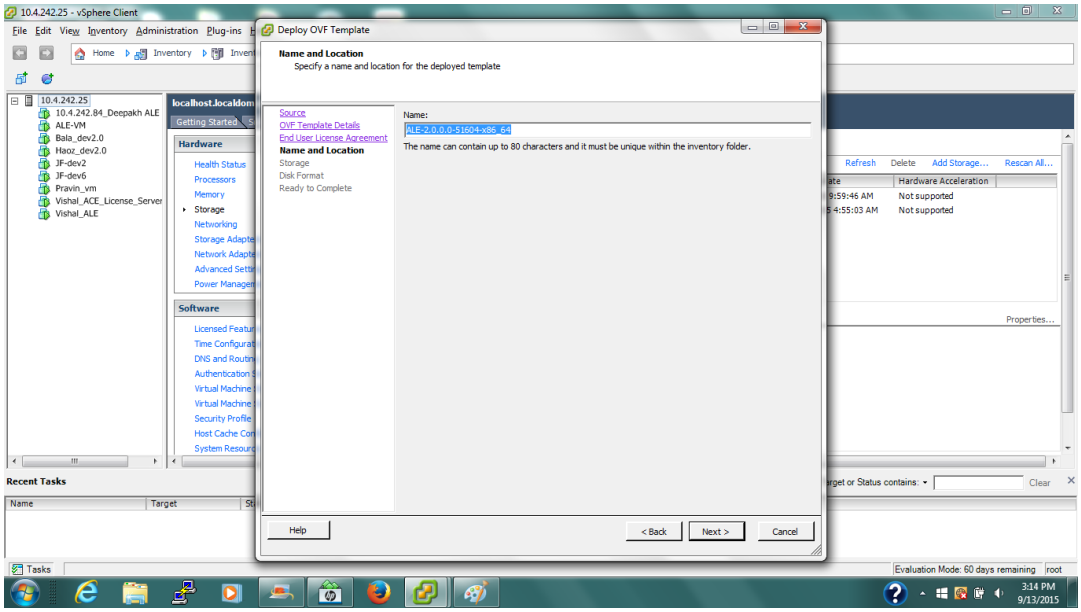
6. The **End User License Agreement** tab displays the Aruba Networks, Inc. end-user software license agreement. Click **Accept**, and then click **Next**.

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



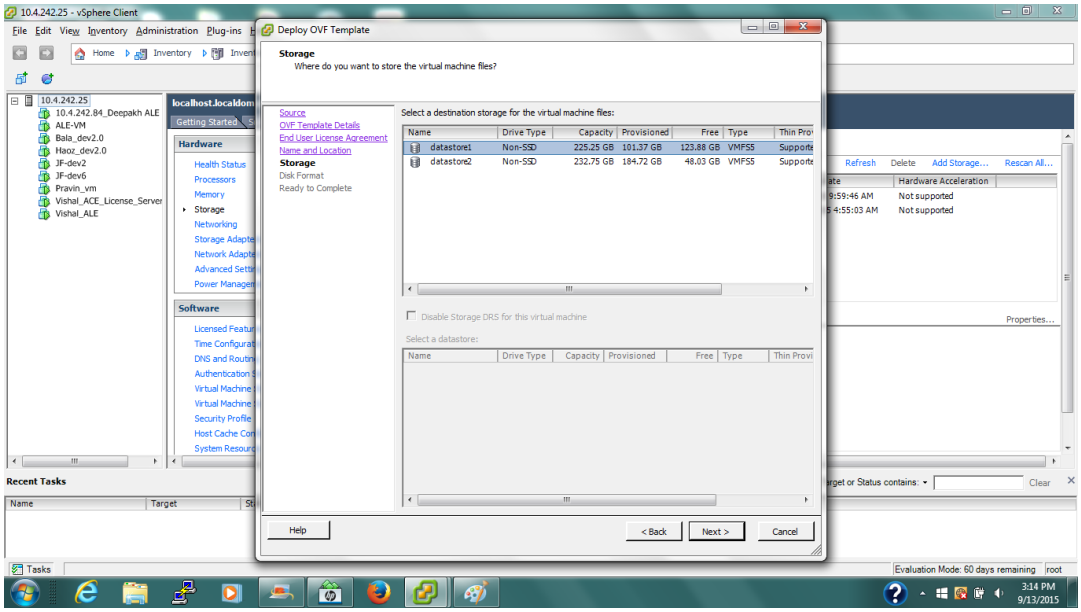
7. Under **Name and Location**, enter a name for the OVA file, and then click **Next**.

Figure 24 *Naming the OVA File*



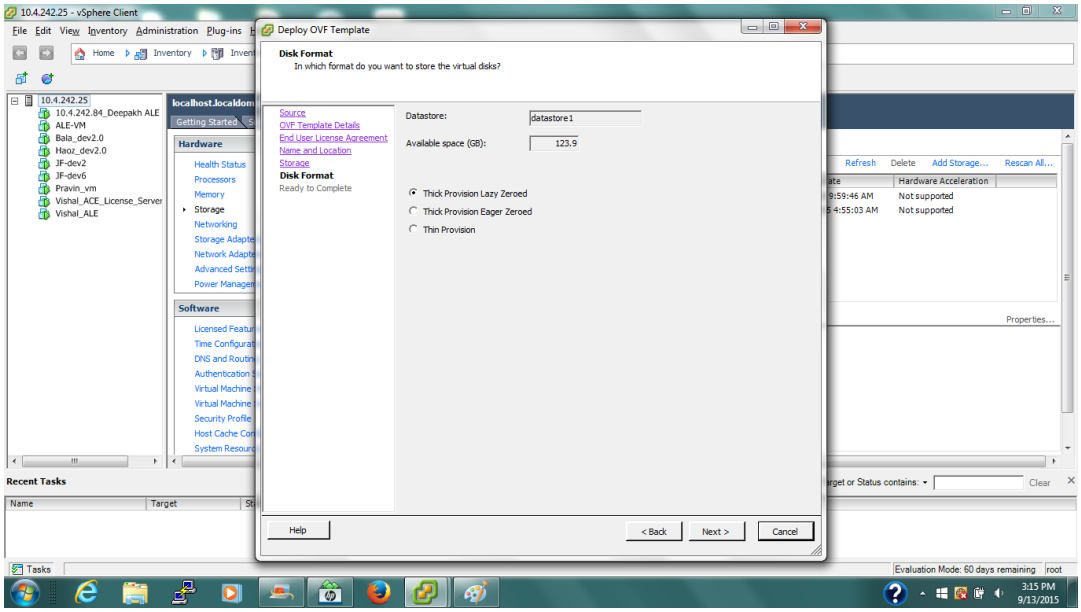
8. Select a datastore with enough free memory available to install the OVA file. Click **Next**.

Figure 25 *Selecting a Datastore*



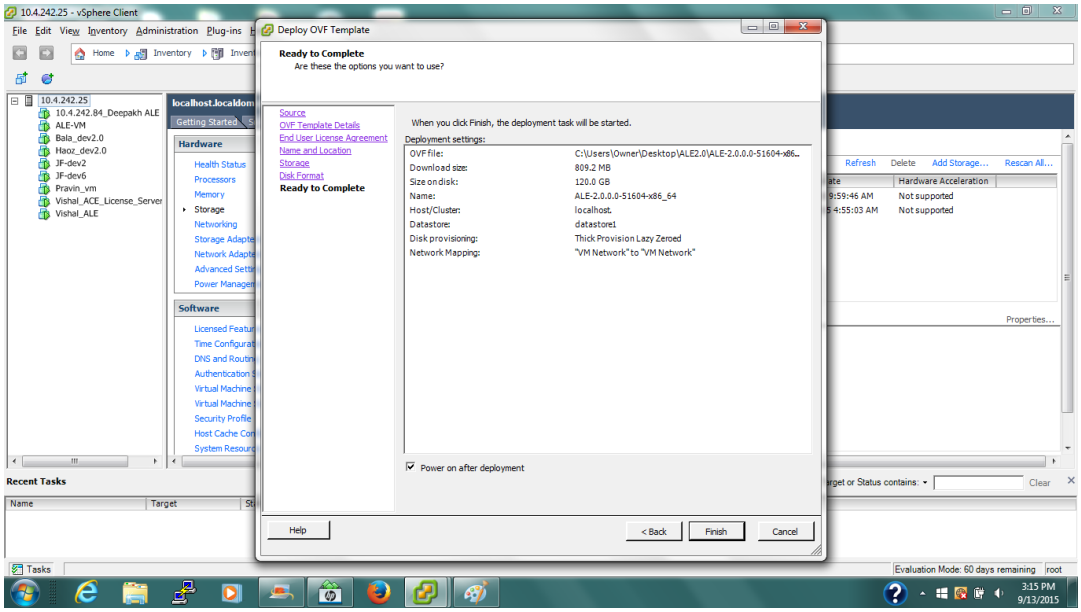
9. Under **Disk Format**, select **Thick Provision Lazy Zeroed**. Click **Next**.

Figure 26 *Setting the Disk Format*



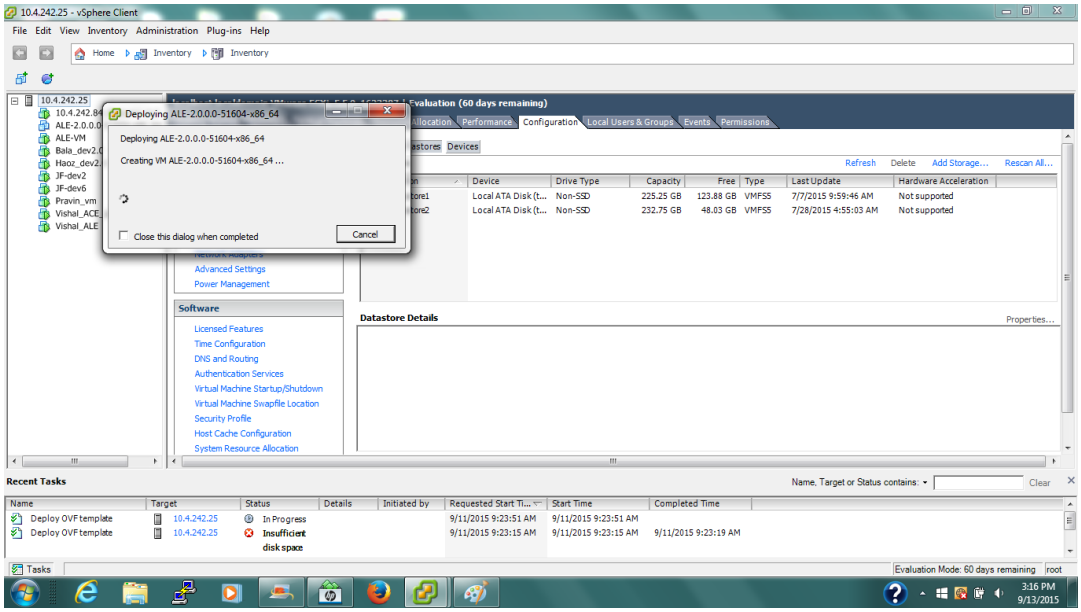
10. Verify all configuration details under the **Deployment Settings** summary page.
    - If any changes must be made, click **Back** to navigate back to any previous pages.
    - If all settings are correct, select **Power on after deployment**, and then click **Finish**.

Figure 27 *The Deployment Settings Summary Page*



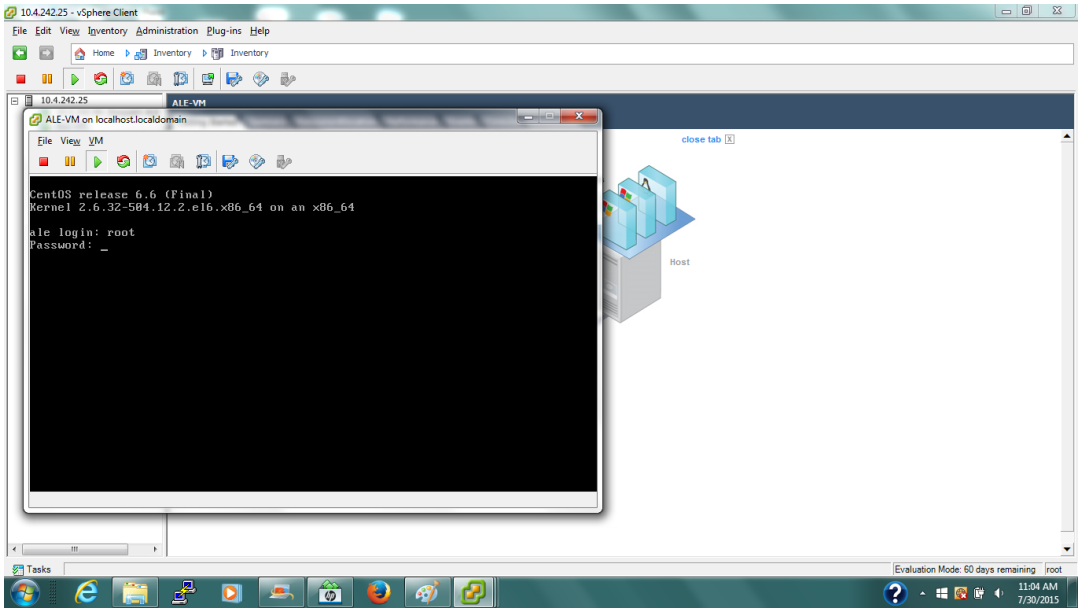
11. OVA installation begins immediately.

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:

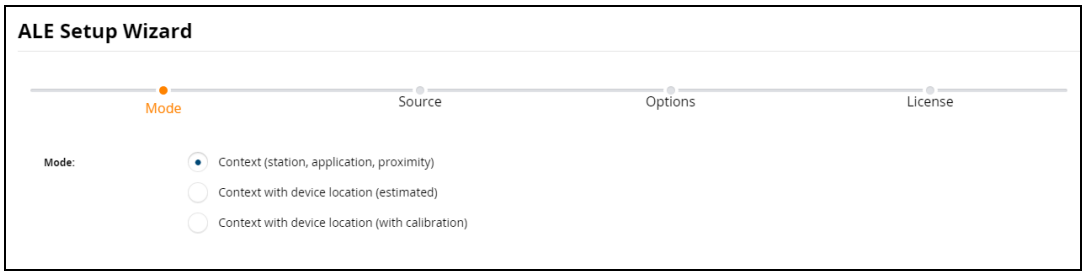
1. Access the ALE WebUI at **https://<ale-ip-address>**, and login with your username and password. Under the default factory settings, the login credentials are:
   - **Username**: admin
   - **Password**: welcome123

NOTE

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

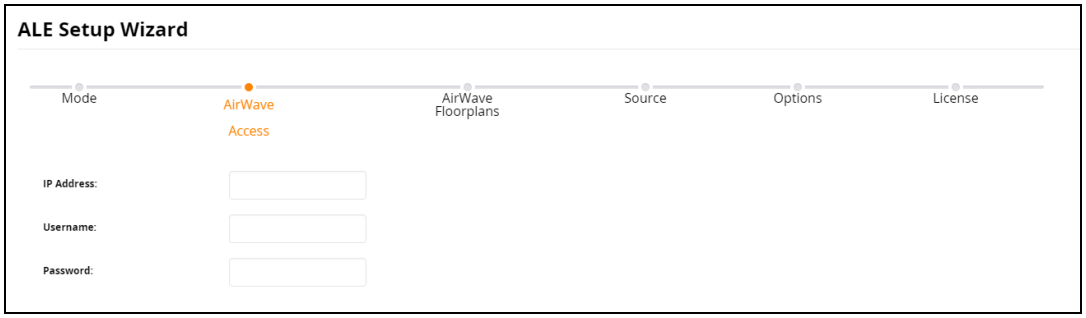
2. Upon entering the setup wizard, the first page that appears is **Mode**. Select the mode for your deployment:
   - **Context (station, application, proximity)**
   - **Context with device location (estimated)**
   - **Context with device location (with calibration)**

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

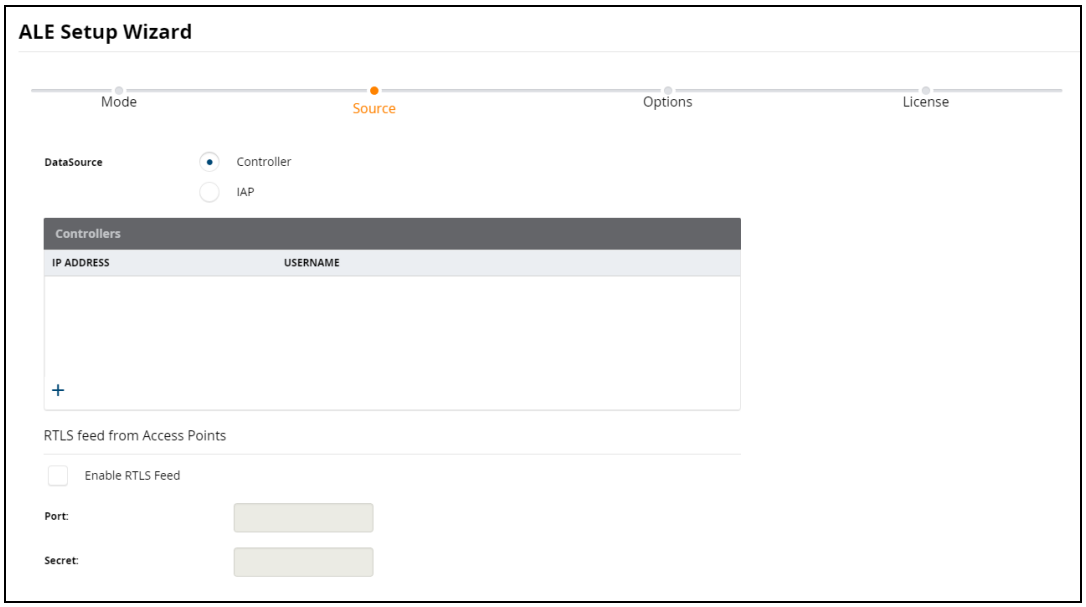
The RTLS secret code and port number must be the same as configured on the controller.

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.

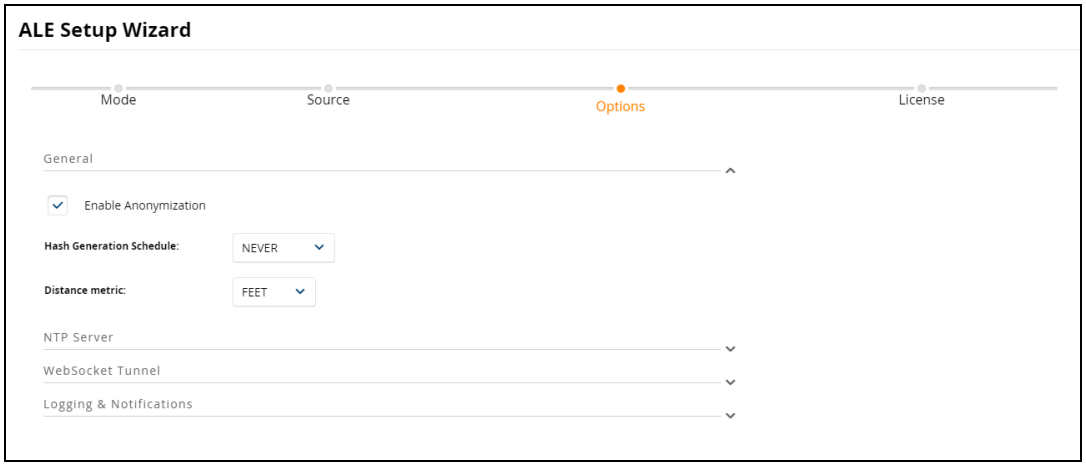
The port number must be the same as configured on the IAP to send ALE data.

  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*

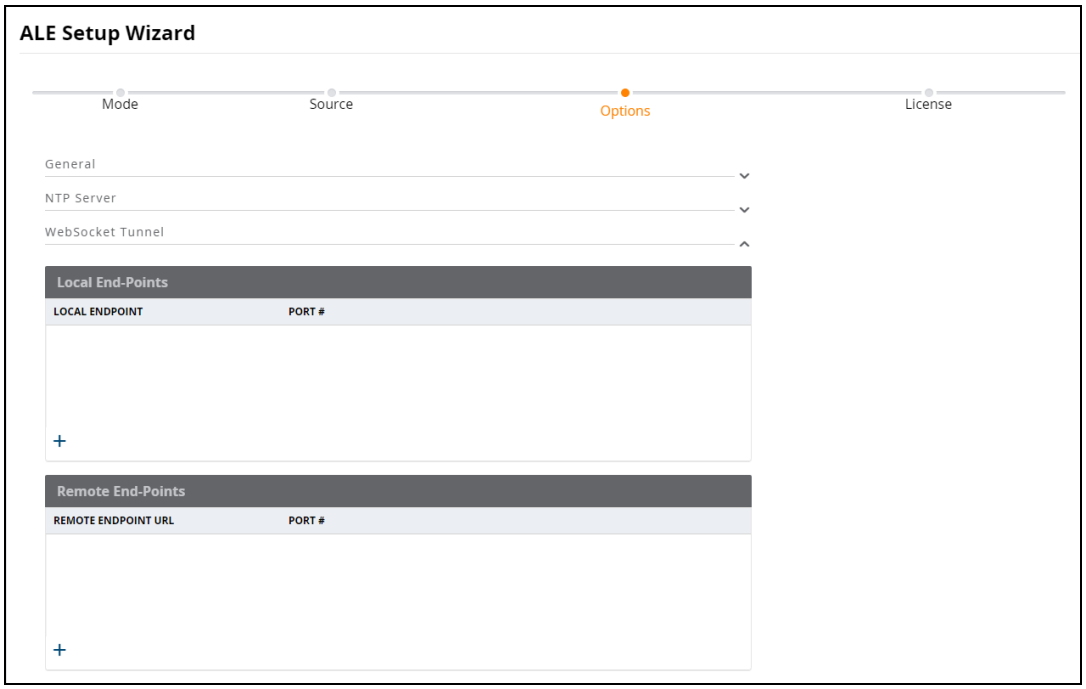


   - **NTP Server**: The **NTP Server** tab allows you to add an NTP server to synchronize all system clocks.
     a. To add an NTP server, enter the IP address of the server in the corresponding text box.
     b. If an NTP server is not added, ALE attempts to synchronize with default NTP servers.

Figure 35 *ALE Setup Wizard - NTP Server*

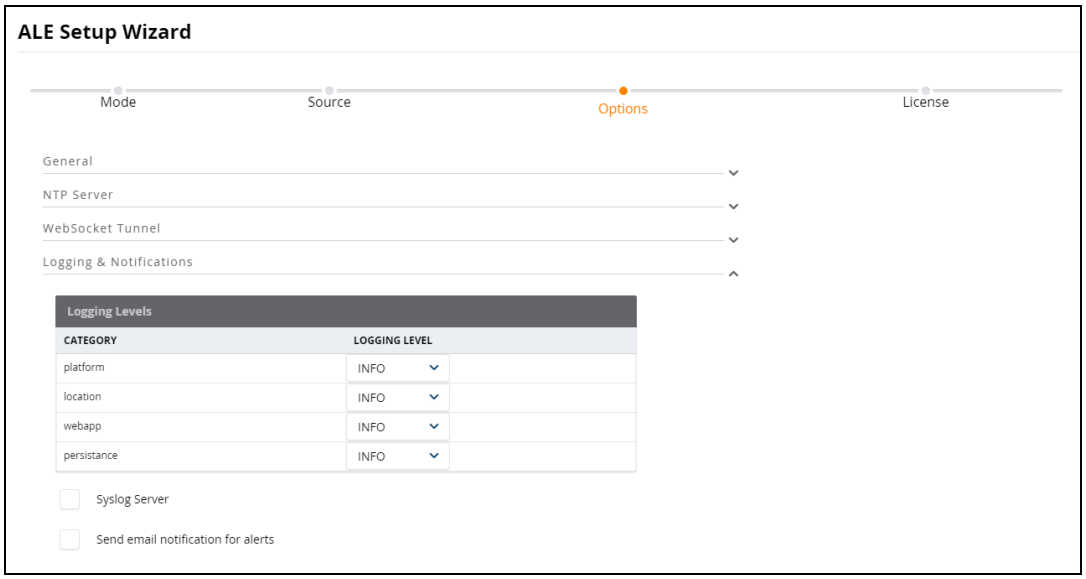
- **WebSocket Tunnel**: The **WebSocket Tunnel** tab allows you to add tunnels for secure communication between local end-points and remote end-points. For more details on WebSocket Tunnel, refer to Configuring WebSocket Tunnel.
  a. Select the desired end-points from the **Local End-Points** table and **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*



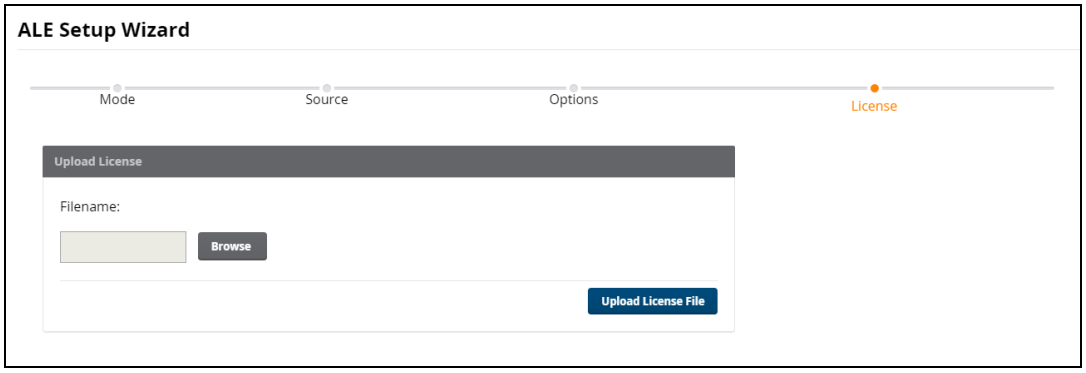
- **Logging & Notifications**: The **Logging & Notifications** tab allows you to set logging levels and notifications for ALE component log files.
  a. Designate the logging level type for each component (error, warning, info, debug, trace).
  b. If a syslog server is required, select the check box for **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*



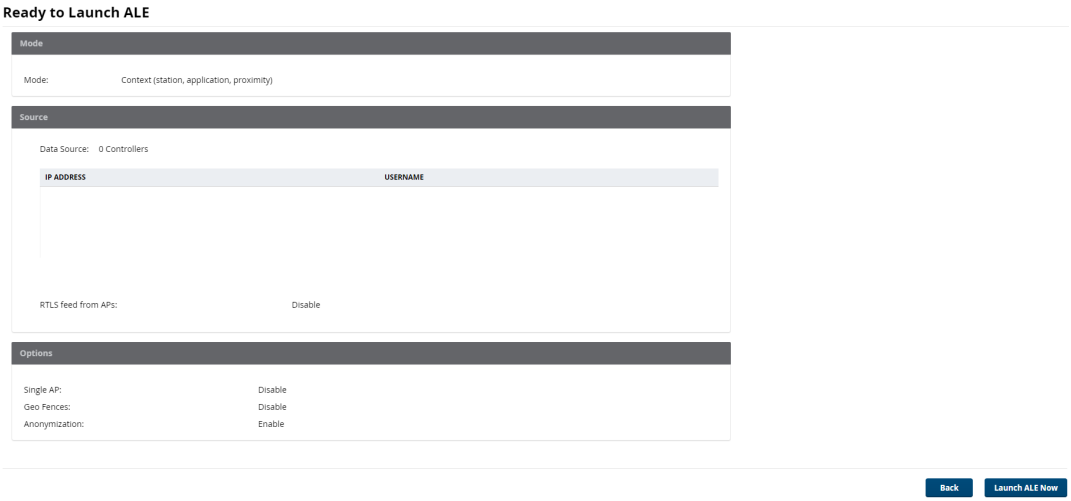
7. After all optional settings are configured, click **Next**.
8. The last page of the ALE Setup Wizard is **License**:
   a. Under **Upload License**, click **Browse** to search for your current ALE license. Windows Explorer opens. See ALE Licenses for more information on the different types of licenses 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).

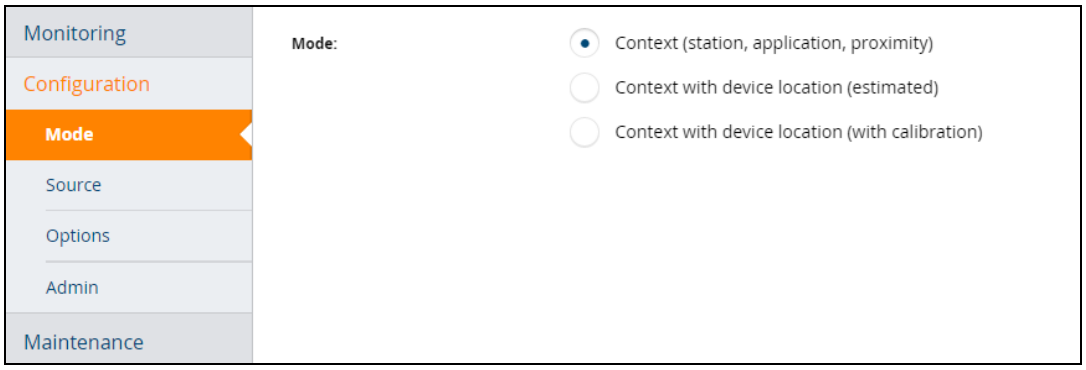
Context (without Maps or Locations)

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.

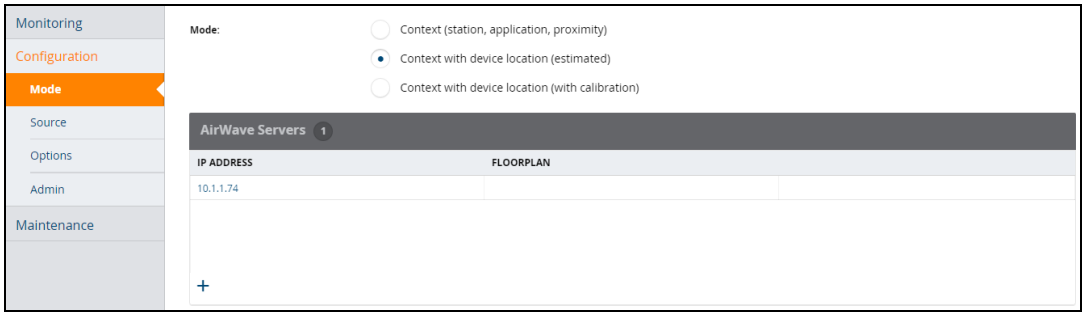
Context with Device Location (Estimated)

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:

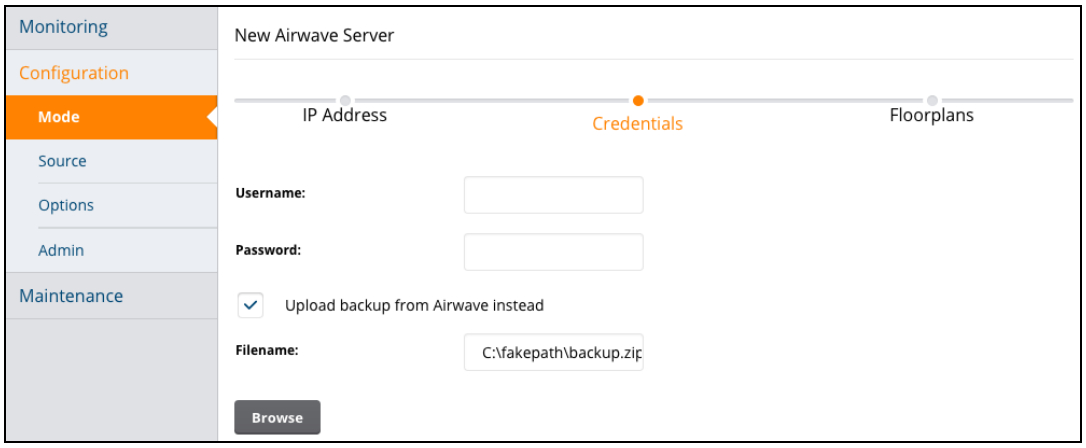
1. Navigate to **Configuration > Mode** in the WebUI.
2. Under **Mode**, select **Context with device location (estimated)**.

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.

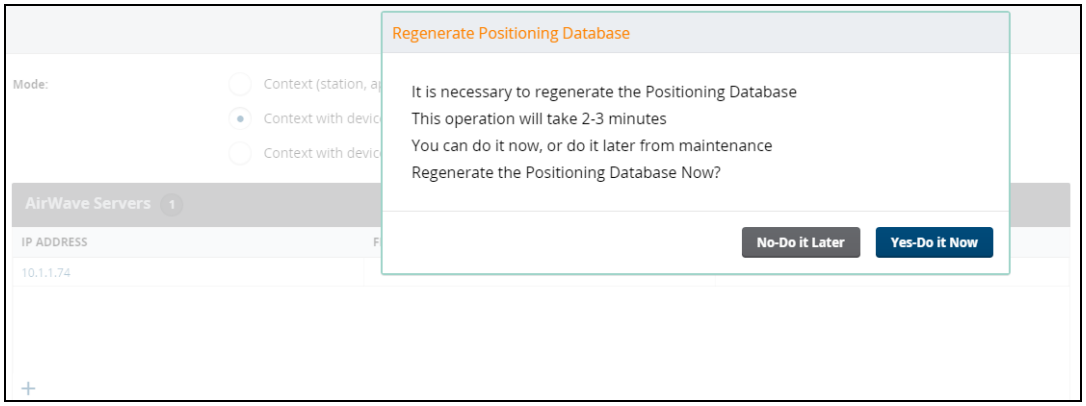
You may be prompted to regenerate the Positioning Database (PDB). Select **Yes-Do it Now** to regenerate the PDB immediately, or **No-Do it Later** to regenerate the PDB at a later time through the **Maintenance** tab.

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.

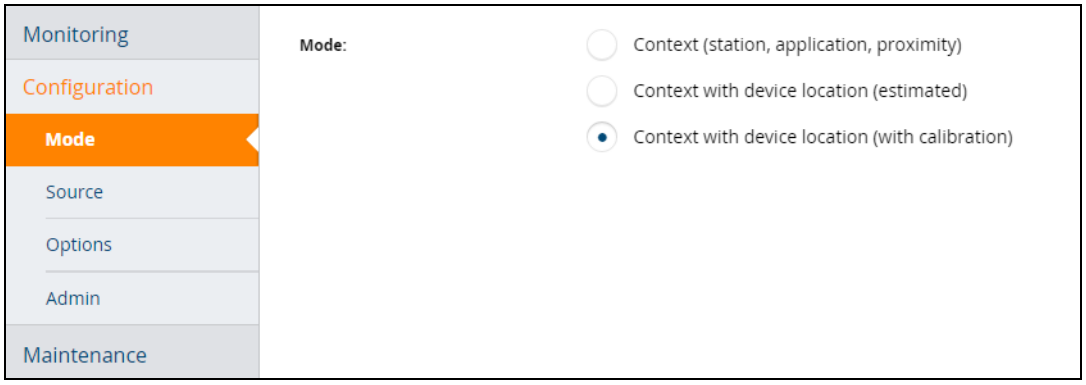
NOTE

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.

Controller deployments also support a PAPI security enhancement that regulates communication between controllers and ALE by authenticating PAPI messages using an MD5 digest/hash and secret key (salt value). This feature provides an increased level of security during ALE-controller communication.

NOTE

Up to 10 controllers can be configured for a deployment.

Before a controller entry can be configured on ALE, ALE must be added to the controller as a management server.

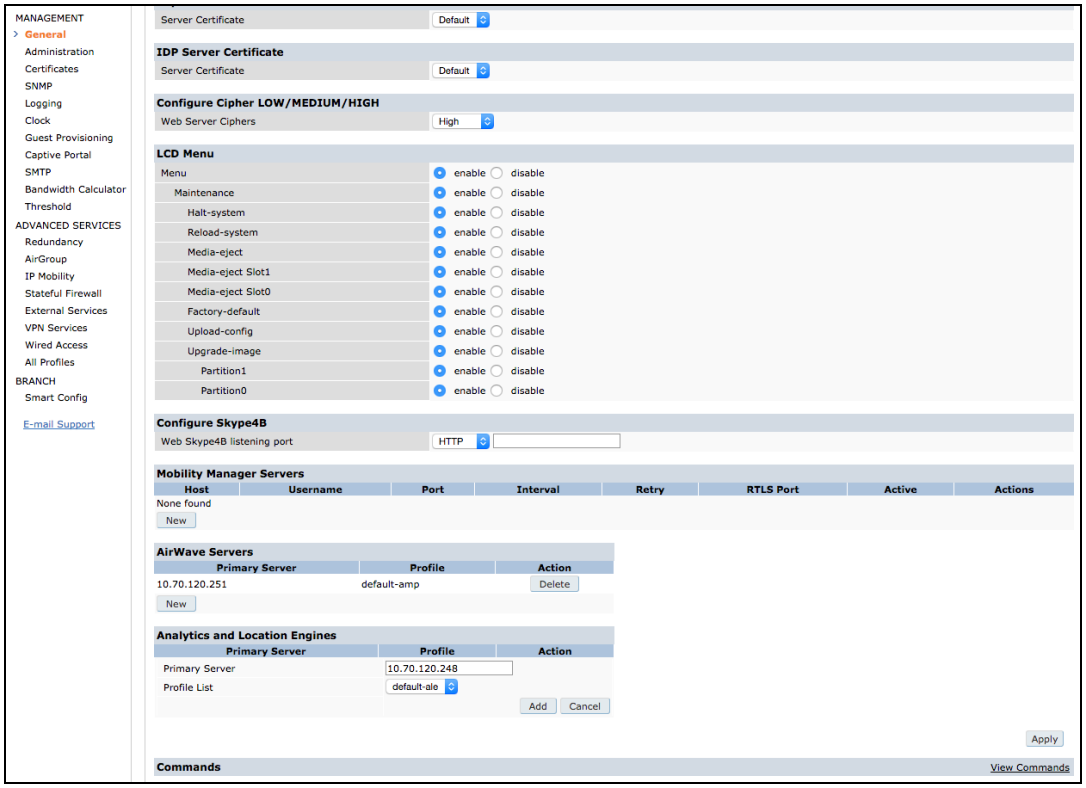
Configuring the Controller

Configure the following on the controller to begin an AMON feed from the controller to ALE. Complete steps 1-6 to forward all contextual information and RSSI data to ALE as part of the AMON feed. Complete step 7 if your deployment requires that RSSI data is sent to ALE through an RTLS feed. If RTLS is configured on the controller, RSSI data is sent to ALEdirectly from each individual AP, while all other contextual information is fowarded through the AMON feed.

To add ALE to a controller:

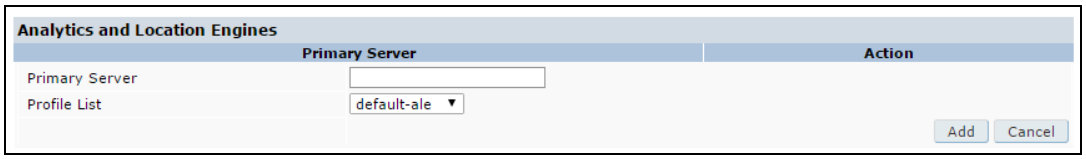
1. Access the controller WebUI.
2. Navigate to **Configuration > Management > General**.

Figure 45 *Adding ALE to a Controller*



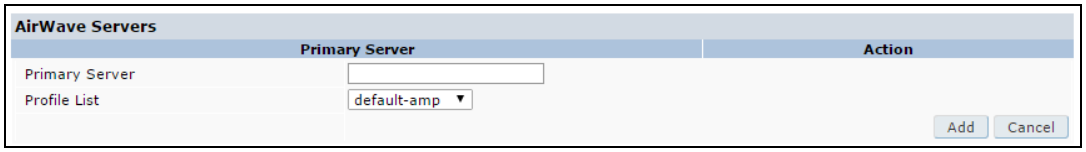
3. Under **Analytics and Location Engines**, click **New** to add an ALE server.
4. Enter the IP address of the ALE server under **Primary Server**.

Figure 46 *Adding a New ALE Server*



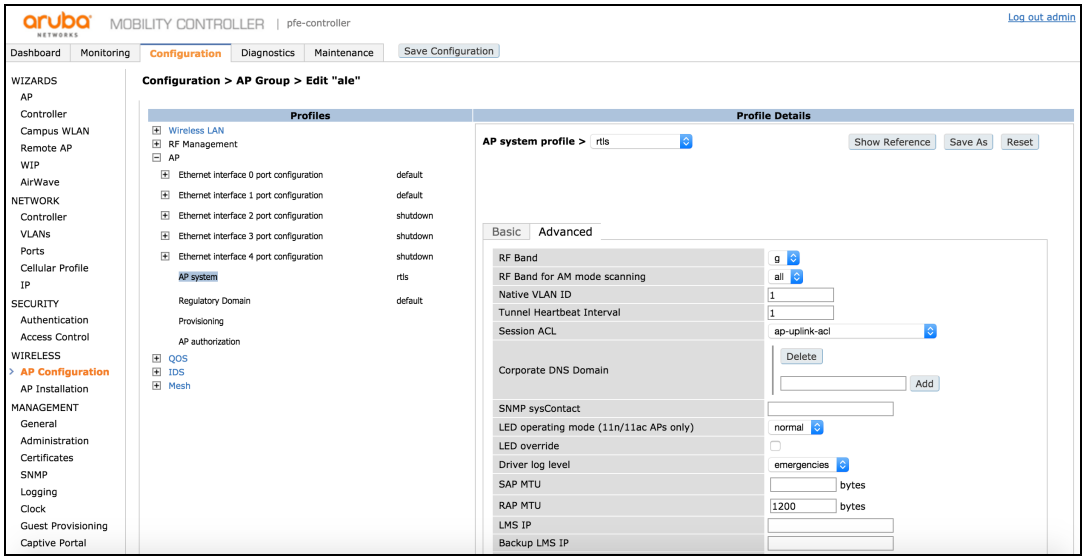
5. (Optional) If your ALE instance is deployed under context mode with device location (estimation), you must also add an AirWave server to the controller under **AirWave Servers** on the same page.
   a. Click **New** to add an AirWave server.
   b. Enter the IP address of the AirWave server under **Primary Server**.
   c. Click **Add**.

Figure 47 *Adding a New AirWave Server*



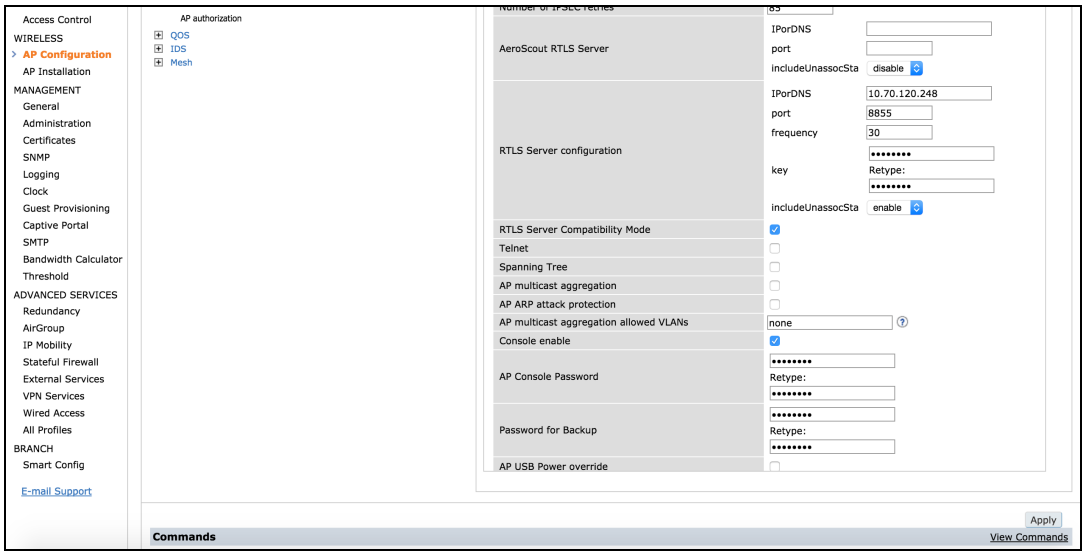
6. Click **Appy** and **Save Configuration** to save your changes.
7. (Optional) If your deployment requires an RTLS feed to forward RSSI data to ALE, navigate to **Configuration > Wireless > AP Configuration** to configure the RTLS settings.

Figure 48 *Configuring RTLS Settings on the Controller*



   a. Under **Profiles**, expand **AP** and select **AP system**.
   b. Select the **Advanced** tab under **Profile Details**.
   c. Enter the IP address of the RTLS server (ALE server), the destination port, and frequency of RTLS updates under **RTLS Server configuration**. Create an RTLS key and enable **includeUnassocSta**.
   d. Click **Apply** and **Save Configuration** to save your changes.

Figure 49 *Configuring an RTLS Server*

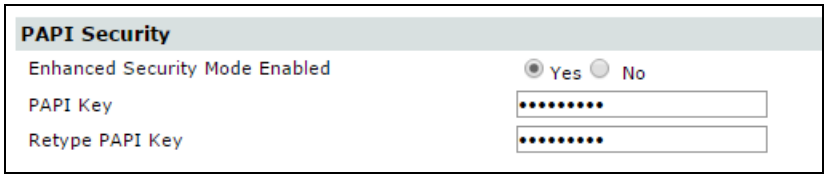


8. (Optional) If your deployment requires the PAPI security enhancement to secure communication between the controller and ALE, navigate to **Configuration > Network > Controller > System Settings** to enable PAPI security.
   a. Under **PAPI Security**, select **Yes** to enable the enhanced security mode.
   b. Enter a secret key under the **PAPI Key** text box to add a salt value to the MD5 digest/hash.
   c. Retype the PAPI key.
   d. Click **Apply** and **Save Configuration** to save your changes.

NOTE

If you do not configure a secret key, ALE populates a default key value.

Figure 50 *Enabling PAPI Security on the Controller*



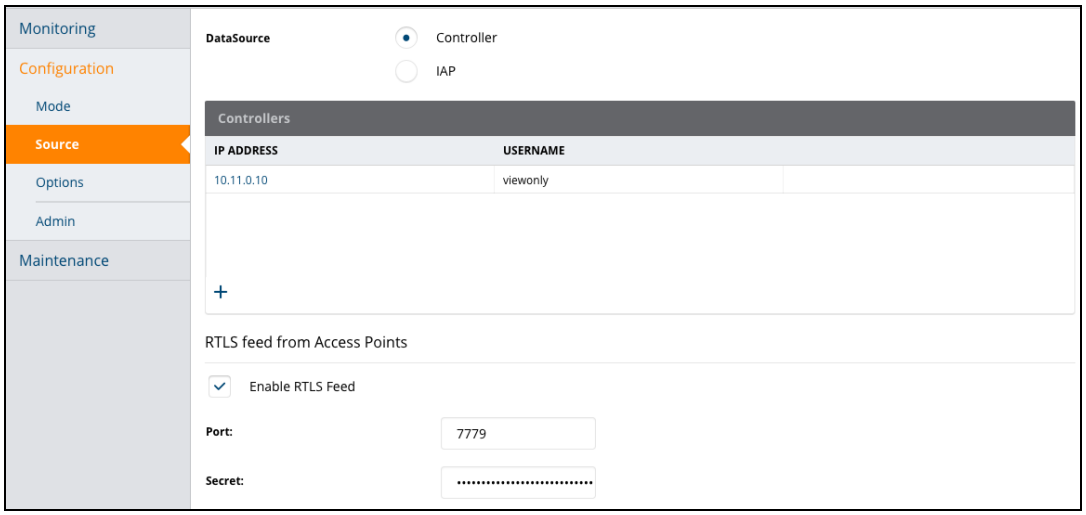
Configuring ALE

After the ALE server is configured on the controller, the controller entry can be added to ALE.

To configure a controller on ALE:

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

Figure 51 *Configuring a Controller*



a. To add a new controller, click the **+** button on the bottom of the **Controllers** table. The **New Controller** popup window appears.

b. Enter the IP address, username, and password of the controller in the corresponding text boxes.

c. (Optional) If your deployment requires the PAPI security enhancement, click the checkbox for **Enable PAPI Security**. Enter a secret key under the **Key** text box to add a salt value to the MD5 digest/hash.

NOTE

If you do not configure a secret key, ALE populates a default key value.

Figure 52 *Enabling PAPI Security*

d. Click **Add** to add the new controller.

NOTE

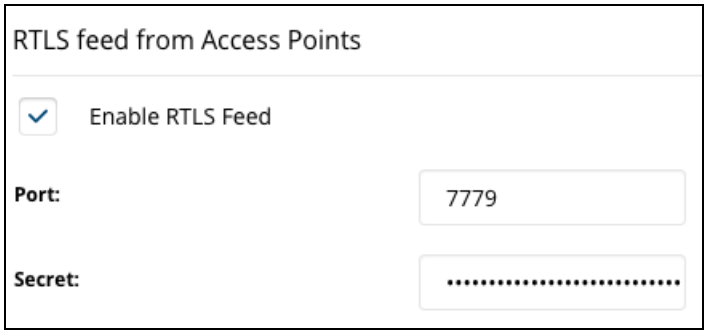
The source IP address for AMON traffic must be set to the source address of the interface on the controller in the same subnet as the next hop to reach the ALE.

4. (Optional) If your deployment requires an RTLS feed, select **Enable RTLS Feed** under **RTLS feed from Access Points**.
   a. 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 53 *Enabling RTLS*



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

NOTE

If PAPI security is disabled on an existing controller, the controller must be deleted from the **Controllers** table under **Configuration > Source > DataSource > Controller**, and then re-added as a new controller.

IAP Deployment

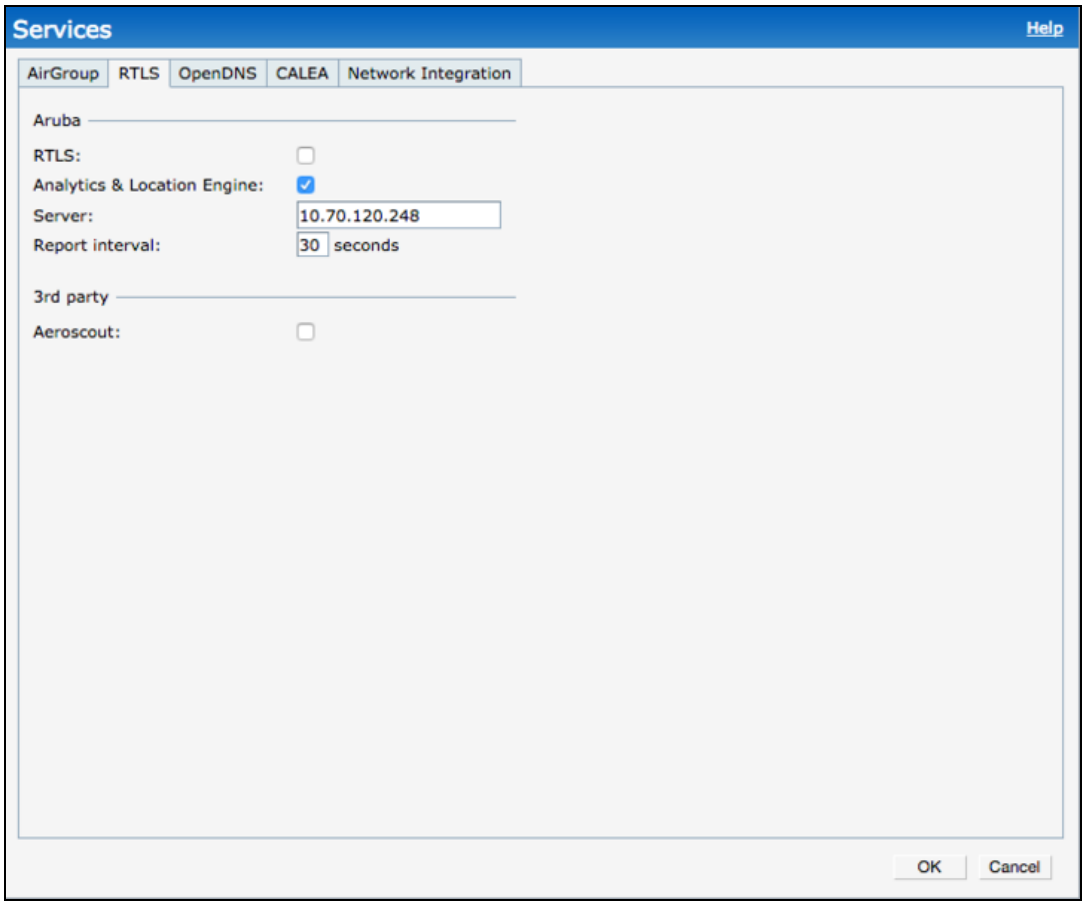
ALE supports integration with IAP. The IAP sends client information and other status information to the ALE server.

Configurating the Instant VC

To configure the IAP VC:

1. Navigate to **More > Services > RTLS** from the VC homepage.
2. Select the checkbox for **Analytics & Location Engine**.
3. Enter the IP address of the ALE server under **Server**.
4. Set the **Report interval** to 30 seconds.
5. Click **OK** to save your configuration.

Figure 54 *Adding an ALE Entry to the IAP*



NOTE

Do NOT select the checkbox for **RTLS** when **Analytics & Location Engine** is selected, as this setting is used for RTLS engines outside of ALE. Similarly, do not enable RTLS on ALE when IAP is the source of data for ALE.

Configuring ALE

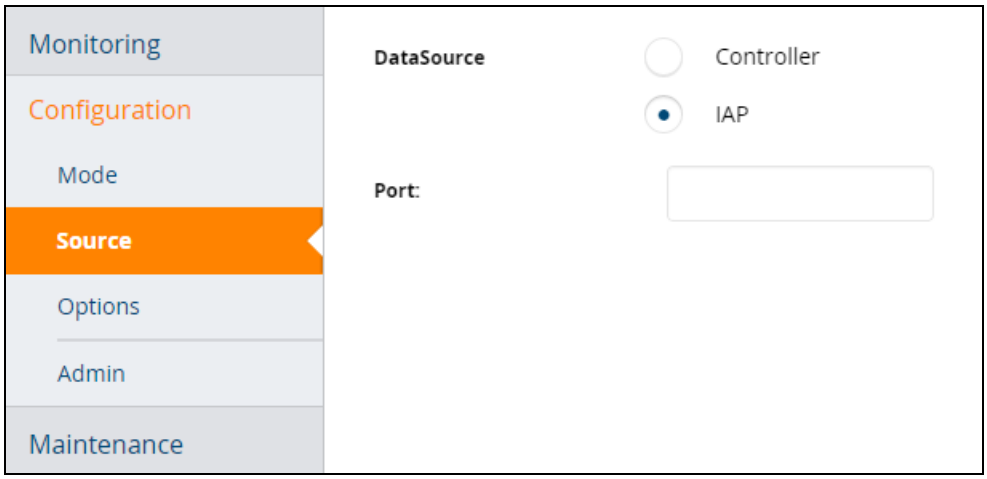
To configure an IAP deployment:

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

NOTE

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 55 *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.

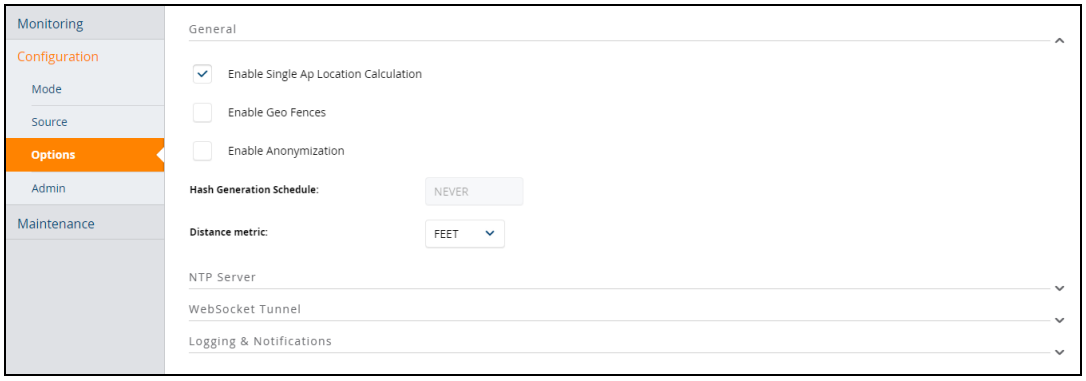
Single AP Location Calculation

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 56 *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 2.0.0.x 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:

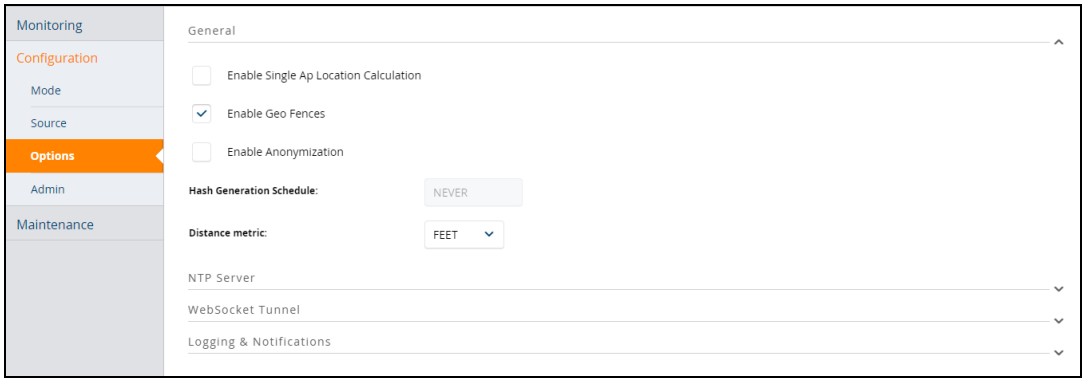
1. Login to your AirWave server.
2. Click on the **VisualRF** tab and select the campus.
3. Select the building and floor.
4. Once the floor map is displayed, click on the **Edit** tab.
5. Select the **Draw Region** tool to create the GeoFence region.
   a. Enter a name for the newly created region.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



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

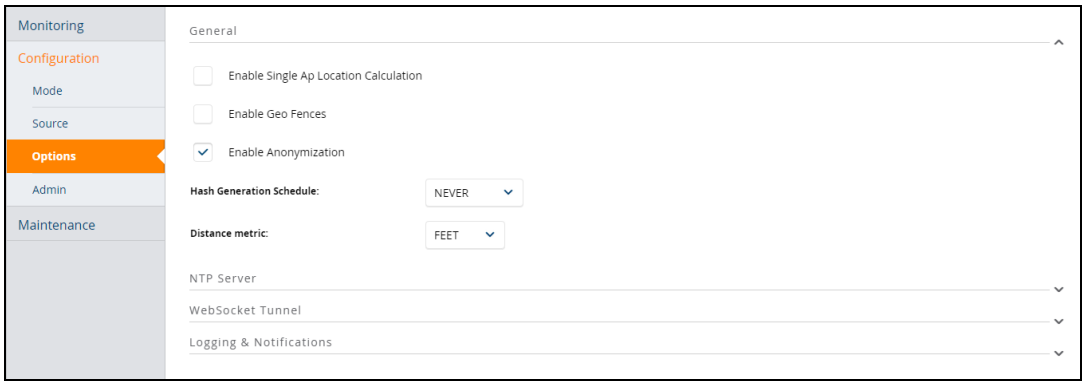
Anonymization

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 58 *Enabling Anonymization*



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

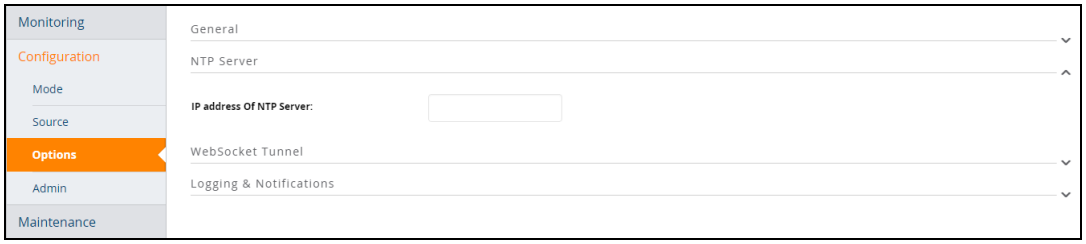
Configuring the NTP Server

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 59 *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

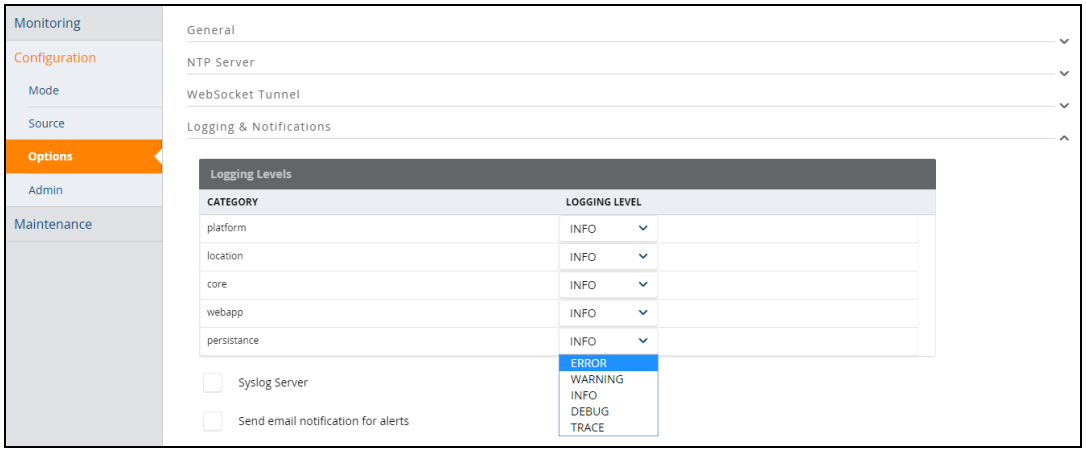
Log files are available for each of the major ALE components:

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

To set logging levels for each ALE component:

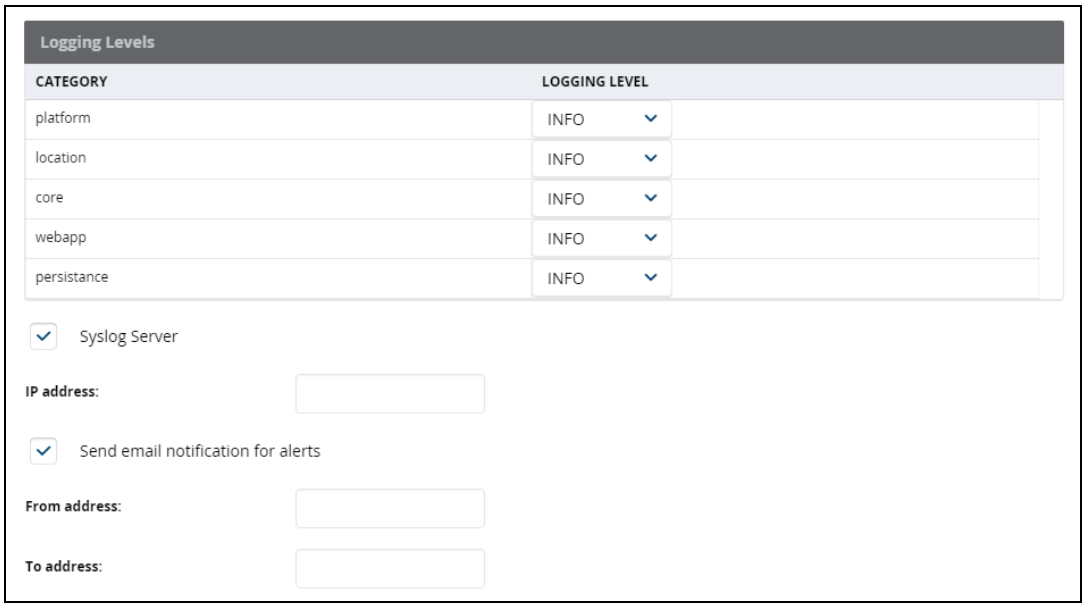
1. Navigate to **Configuration > Options** in the WebUI.
2. Open the **Logging & Notifications** tab.
3. Under **Logging Level** in the **Logging Levels** table, designate the logging level type for each component.

Figure 60 *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 61 *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 access to the ALE WebUI. Depending on the role type, each user is provided with a specific set of privileges to modify and/or view configurations. Similarly, users can be granted access to the REST API via HTTP authentication.

ALE supports the following user roles:

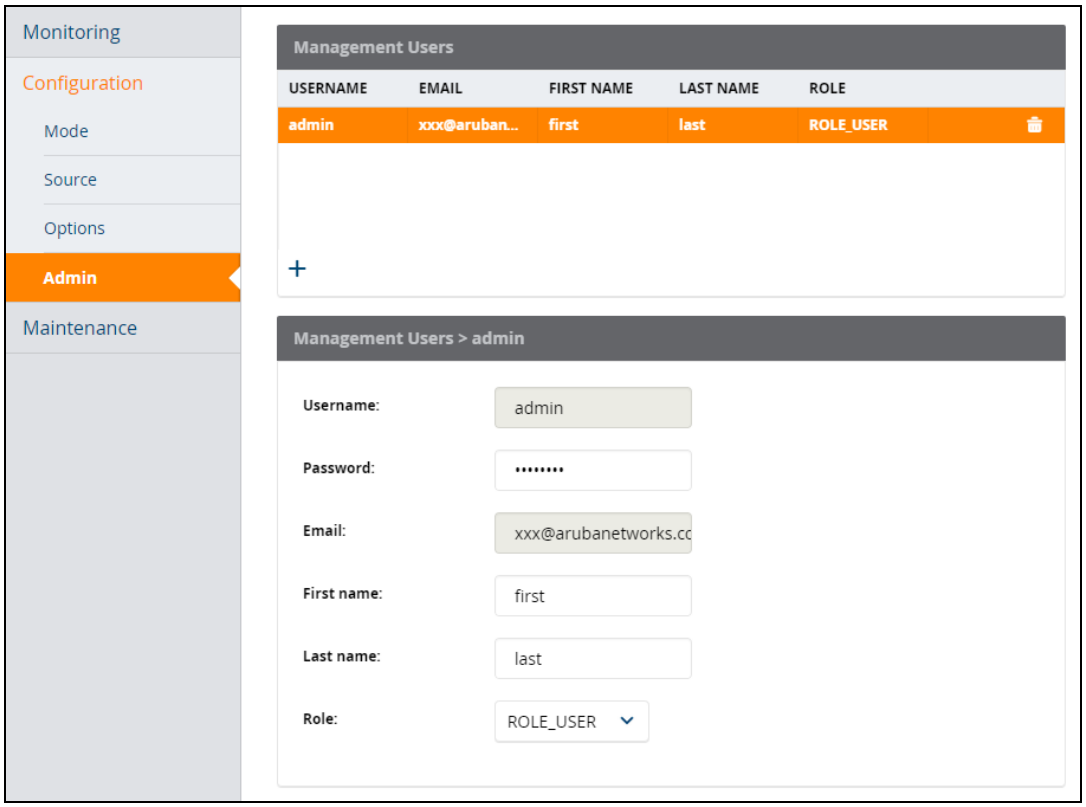
- **Admin User**: Admin users can modify, update, and view any configuration on ALE. They also have access to all REST APIs.
- **REST User**: REST users can only access REST APIs. They cannot access the ALE WebUI to view and/or modify any configurations.
- **View-Only User**: View-only users can access the ALE WebUI to view configurations, but they cannot modify any configurations.

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 62 *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.

Authentication through ClearPass Policy Manager

ALE also supports authentication through Aruba ClearPass Policy Manager (CPPM), which is a network policy management tool that allows users to configure all access security requirements through a single platform. When ALE is configured with CPPM, user authentication is performed externally through the CPPM server. Users are assigned specific privileges based on the ALE role assigned through CPPM.

To configure CPPM for ALE:

1. Create an XML file for your ALE deployment. See the *ClearPass Policy Manager Configuration API Guide* for more information on the format and contents of XML files.

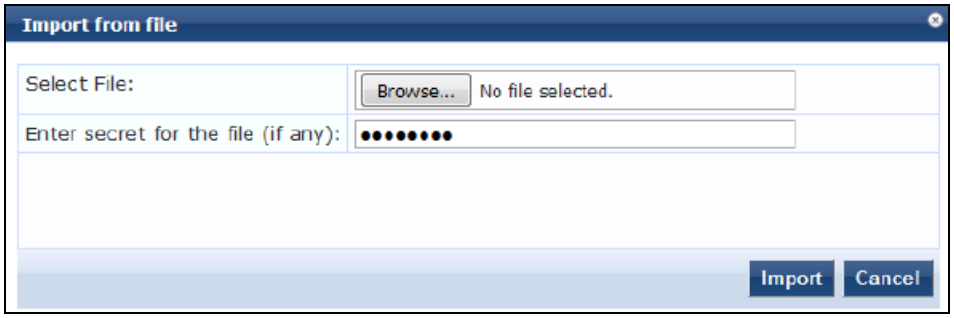
ApplDictionary > name must be **ALE**, and **ApplDictionaryAttributes > attrName** must be **Role**.

Figure 63 *Example XML File*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TipsContents xmlns="http://www.avendasys.com/tipsapiDefs/1.0">
  <TipsHeader exportTime="Wed Jan 27 14:16:47 PST 2016"version="6.5"/>
  <ApplDictionaries>
    <ApplDictionary description="ALE Application Attributes"name="ALE">
      <ApplDictionaryAttributes attrType="String"attrName="Role"/>
    </ApplDictionary>
  </ApplDictionaries>
</TipsContents>
```

2. After the file is created, login to the CPPM server using your ClearPass credentials.
3. Navigate to **Administration > Dictionaries > Applications**.
4. Click the **Import** button (Import) at the top-right corner of the page. The **Import from file** window appears.

Figure 64 *Import From File Window*

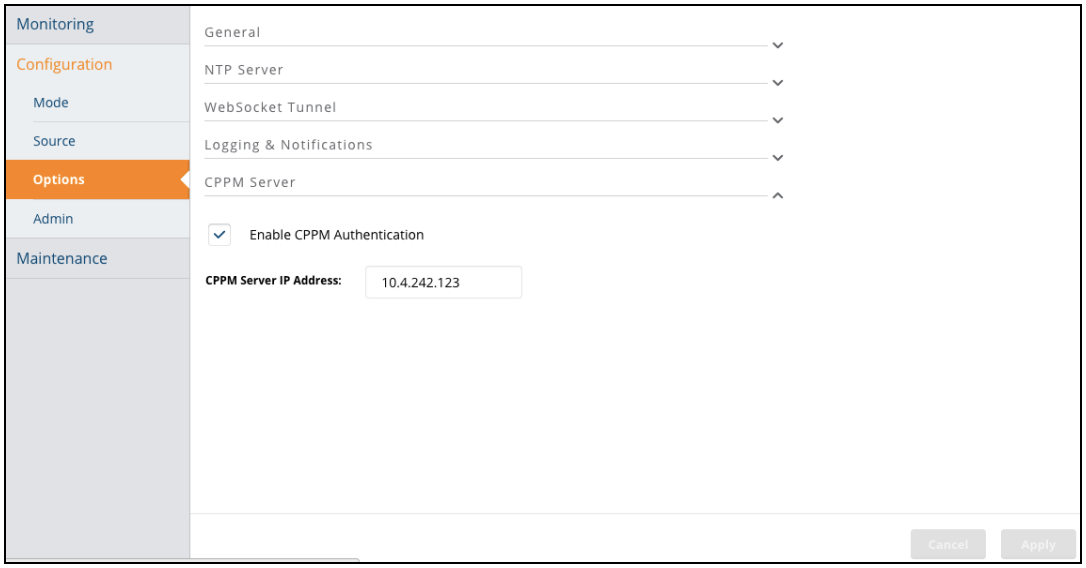


5. Click **Browse...** to select the ALE XML file from your local file explorer.
6. If the file is password protected, enter the password in the **Enter secret for the file (if any)** field.
7. Click **Import**.
8. Configure the necessary enforcement policies under **Configuration > Enforcement > Enforcement Policies**. See the *ClearPass Policy Manager User Guide* for more details on configuring enforcement policies.

After all ALE enforcement policies have been configured through CPPM, access your ALE server to complete CPPM authentication setup:

1. Navigate to **Configuration > Options > CPPM Server** in the WebUI.
2. Select **Enable CPPM Authentication**, and then enter the IP address of the CPPM server.

Figure 65 *Enabling CPPM Authentication*



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

After your CPPM server has been enabled on ALE, you can begin authenticating users through CPPM. For more information on how CPPM handles user authentication and authorization, refer to *ClearPass Policy Manager User Guide > Authentication and Authorization*.

Custom Certificates for HTTPS

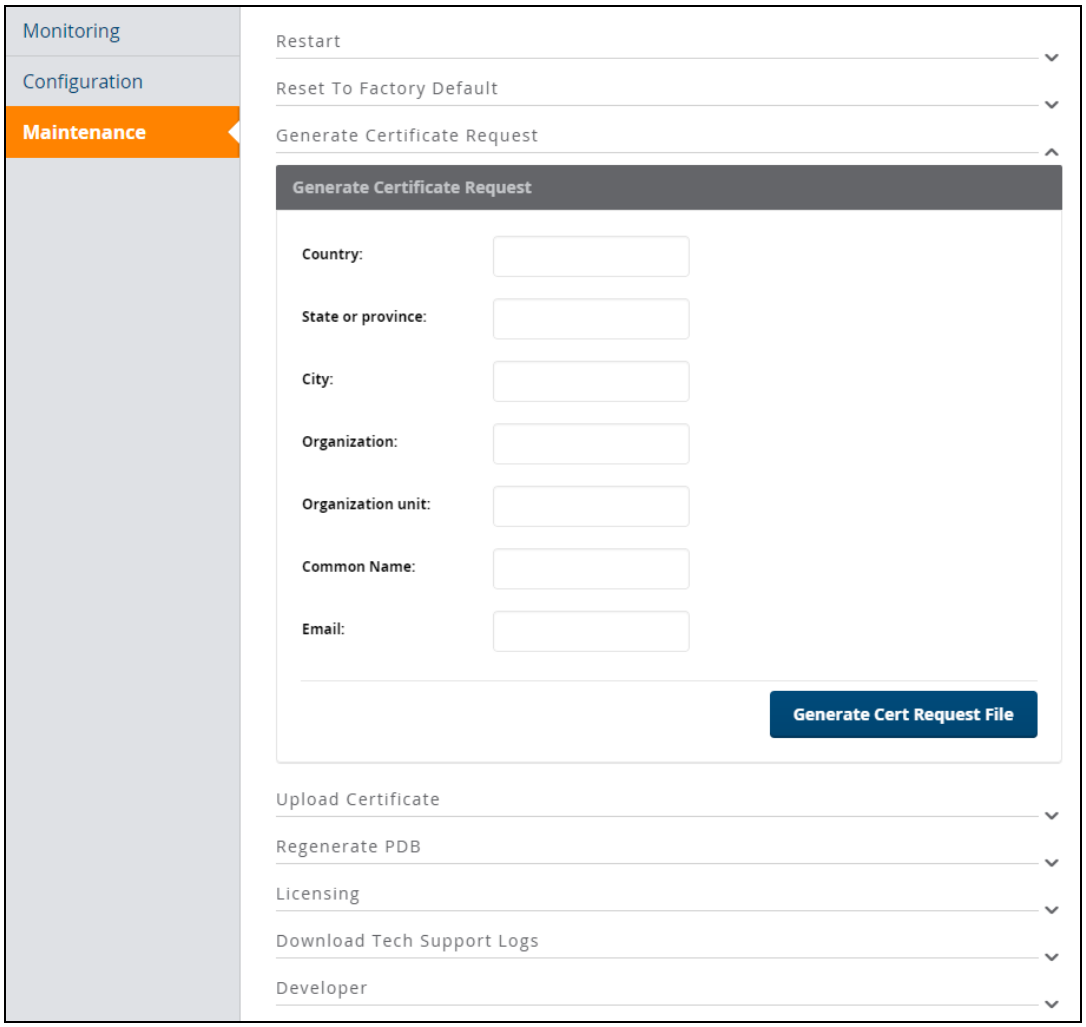
When ALE is installed, self-signed certificates are automatically created and used for HTTPS in the user interface and REST API. To install your own server certificates, generate and upload your certificates manually through the **Maintenance** tab of the WebUI.

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 66 *Generating a Certificate Request*

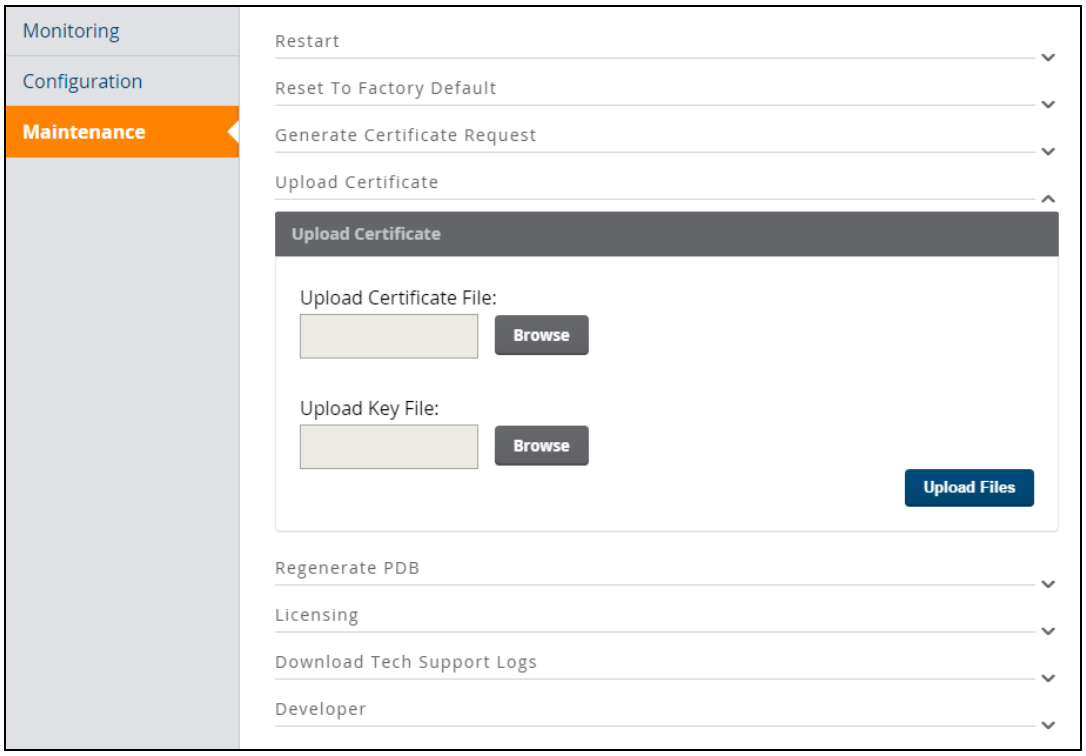


To upload a certificate:

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 67 *Uploading a Certificate*



ALE Licenses

ALE 2.0 operates under three licensing schemes:

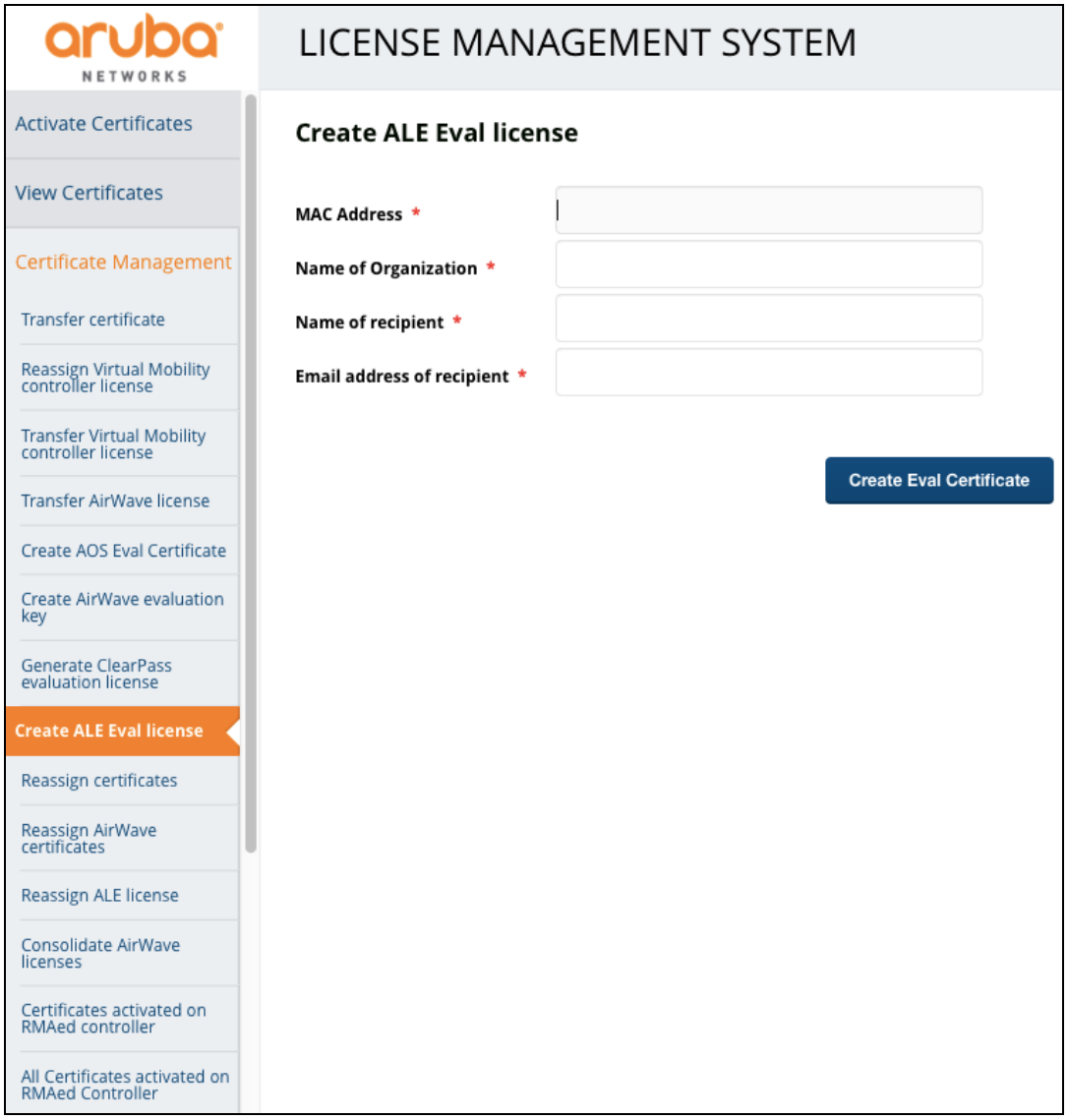
- **Unlicensed**: If ALE is unlicensed, it does not publish any ZMQ messages.
- **Evaluation License**: An evaluation license contains an AP count and expiry period. Once the license expires, ALE switches to the unlicensed state and stops publishing ZMQ messages.
- **Permanent License**: A permanent 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.

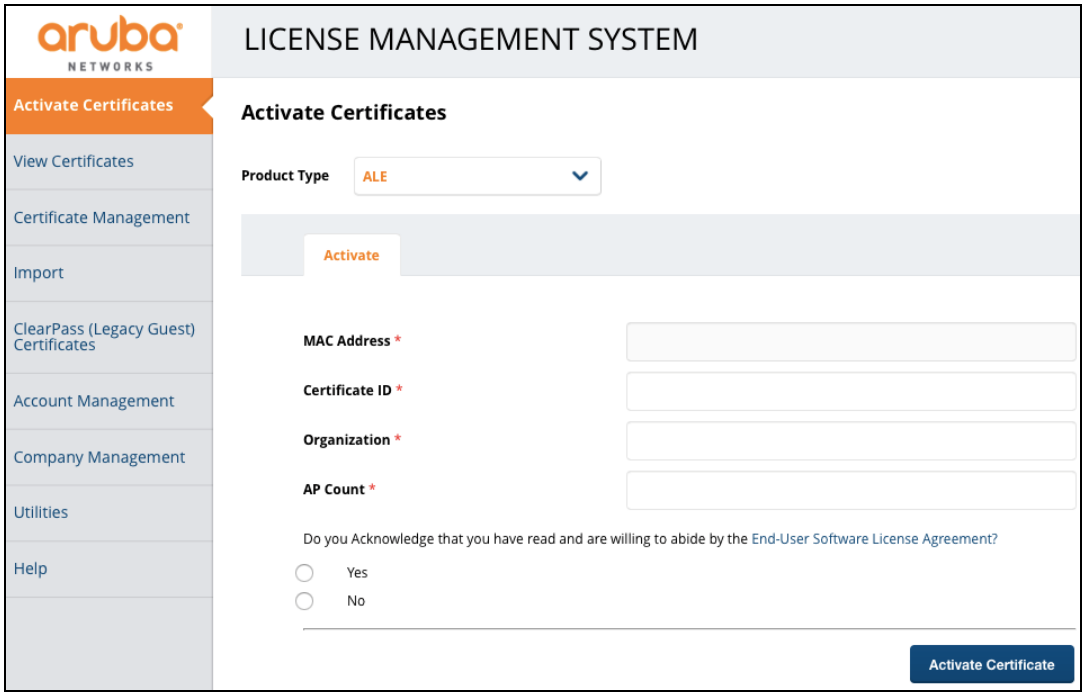
Figure 68 *Generating an Evaluation License*



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 69 *Generating a Permanent License*

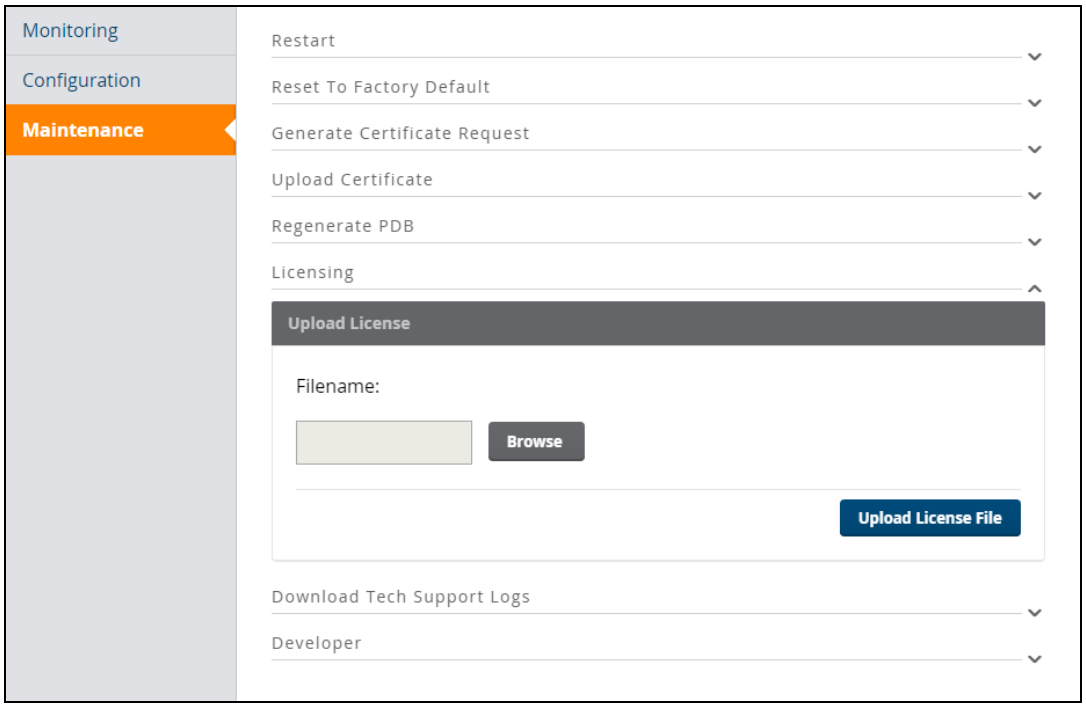


Uploading a License

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 70 *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>/calibration**.
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:

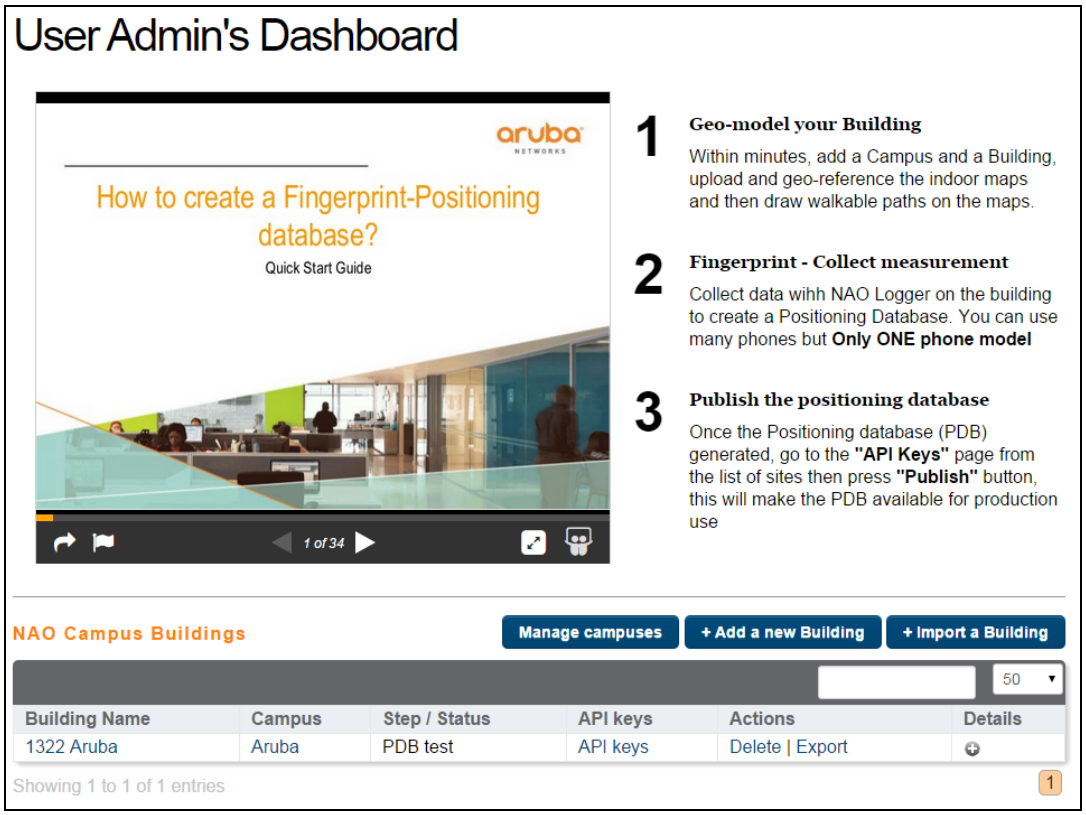
1. Access the Aruba Nao Campus dashboard.
2. Click **Manage campuses** under **NAO Campus Buildings**.
3. Click **New Campus**.
4. Enter a name and description for the campus, and then click **Save**.

Creating a Building

To add a new Aruba Nao Campus building:

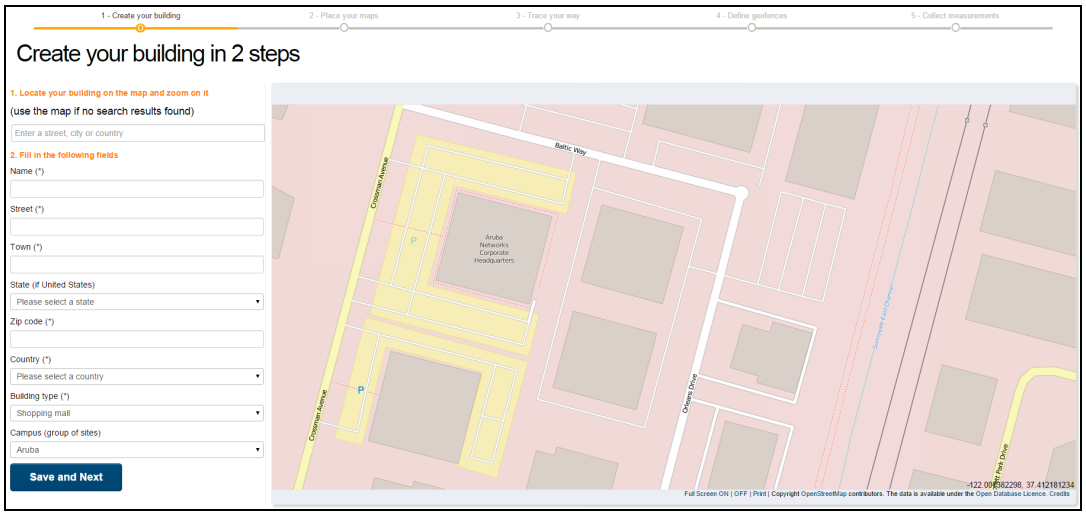
1. Access the Aruba Nao Campus dashboard.

Figure 71 *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 72 *Creating an Aruba Nao Campus Building*

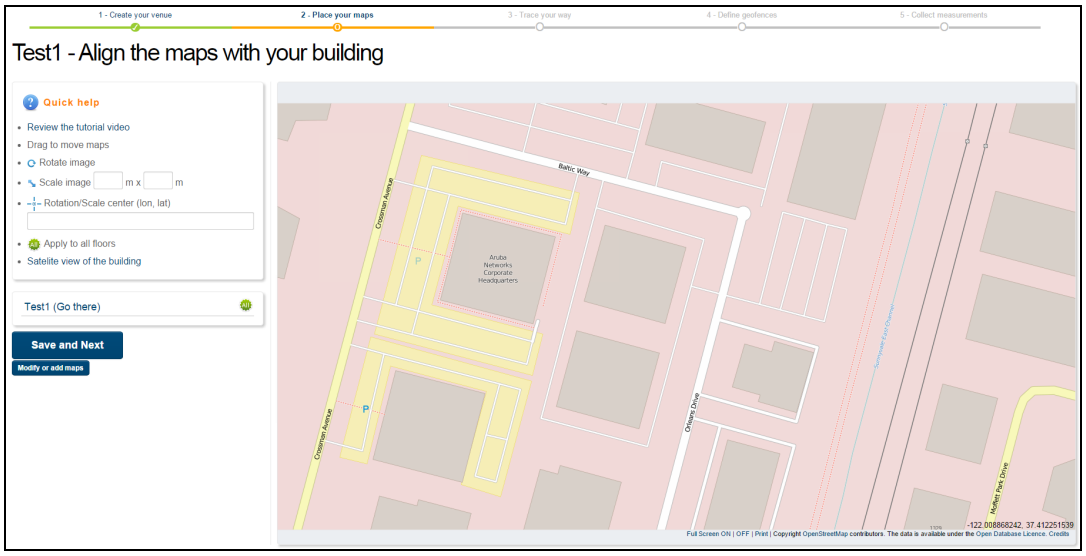


5. Once your building is selected, click **Save and Next**.
6. Under the **Place your maps** page, upload floor plans by dragging and dropping map files (JPG, JPEG, and PNG) to the **Drop files or click here...** box. You can also click the box to select floor plans from your file manager.

Figure 73 *Uploading Maps to Aruba Nao Campus*

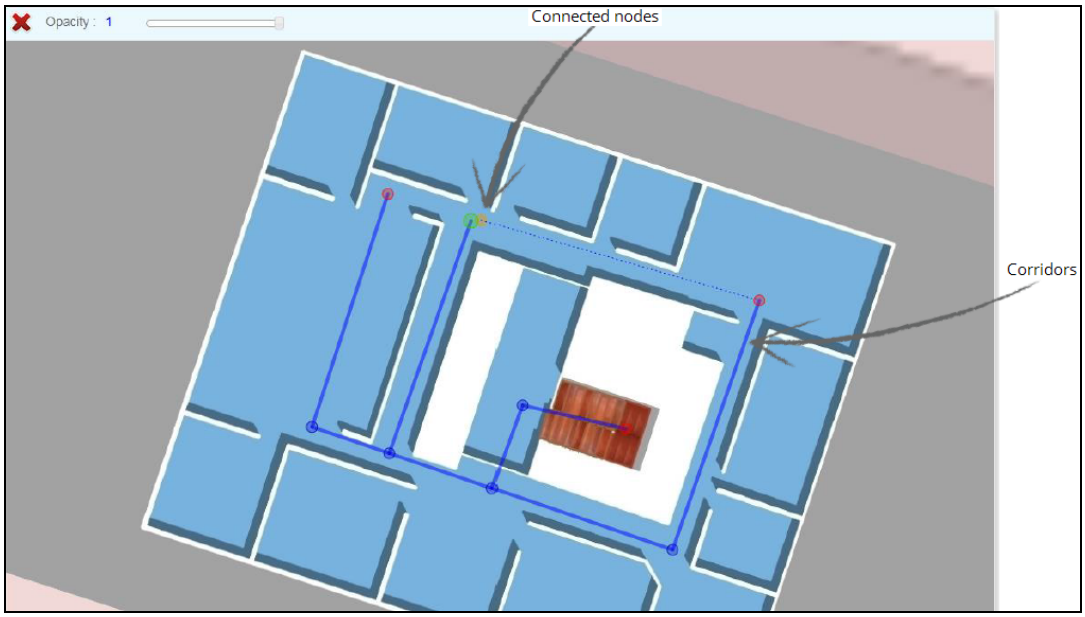
7. Create a name and indicate the floor level number for each floorplan.
8. Click **Save and Next**.
9. Align each floorplan with the building by dragging, rotating, and scaling the image to the correct size and position. The **All** button applies the same alignment to each floorplan.

Figure 74 *Aligning Maps with a Building*



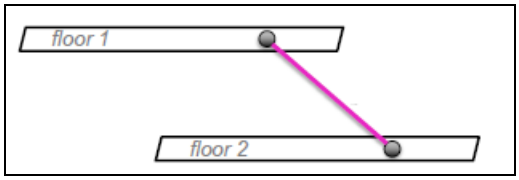
10. Click **Save and Next**.
11. Under **Trace your way**, create a path on the map for collecting fingerprint data. This path appears as a blue line and is later used on the Aruba Nao Logger Android application to generate the Positioning Database (PDB). This is also known as a graph and defines the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

Figure 75 *Creating a Pathway for Fingerprinting*



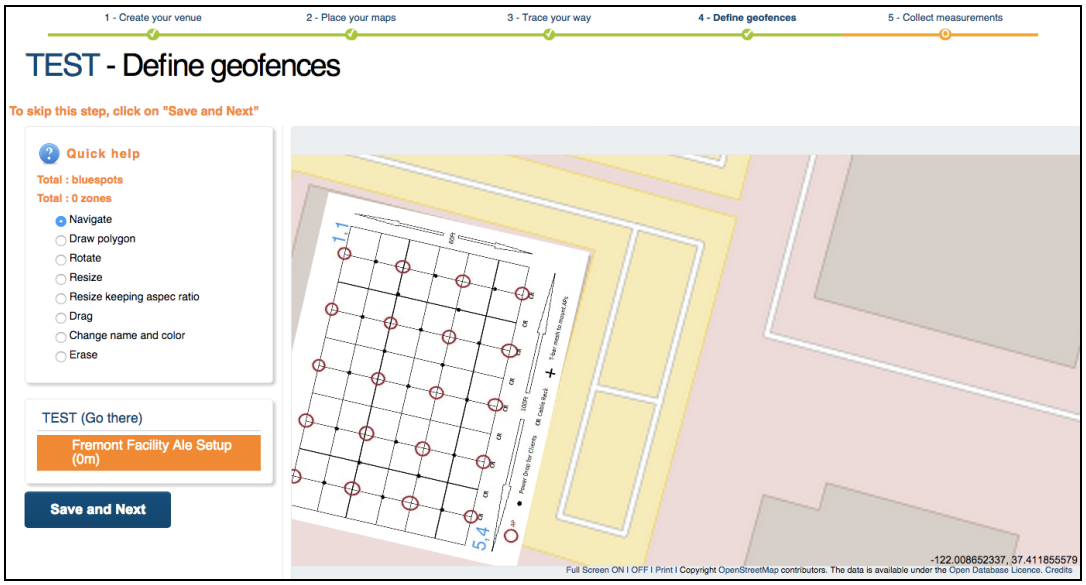
12. If your building contains escalators, stairs, or elevators:
    a. Draw a node on one floor.
    b. Change the floor using the floor selector.
    c. Draw a second node on the new floor. The two nodes connect into a stairway, escalator, or elevator.

Figure 76 *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.

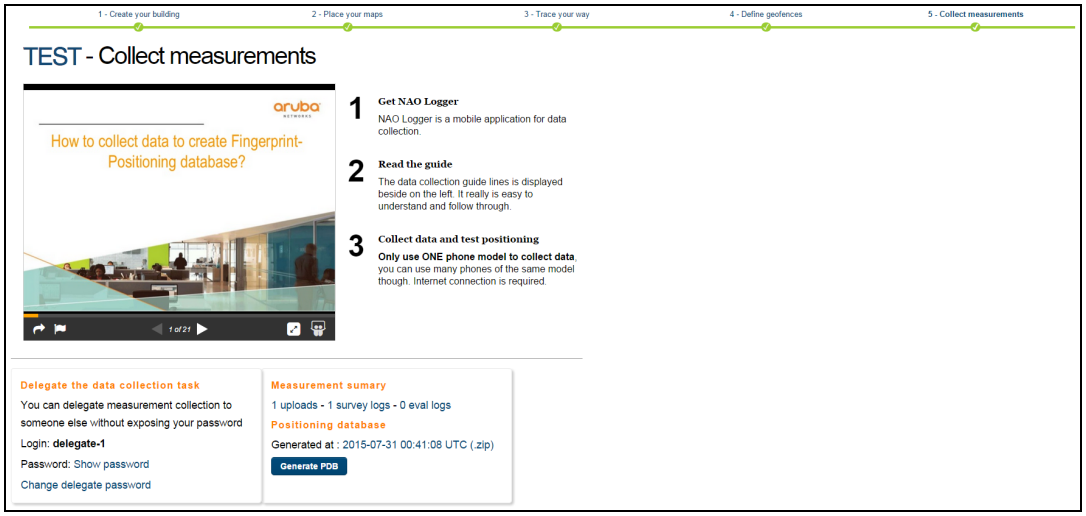
Figure 77 *Defining GeoFences*



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 78 *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 79 *Logging in on an Android tablet*

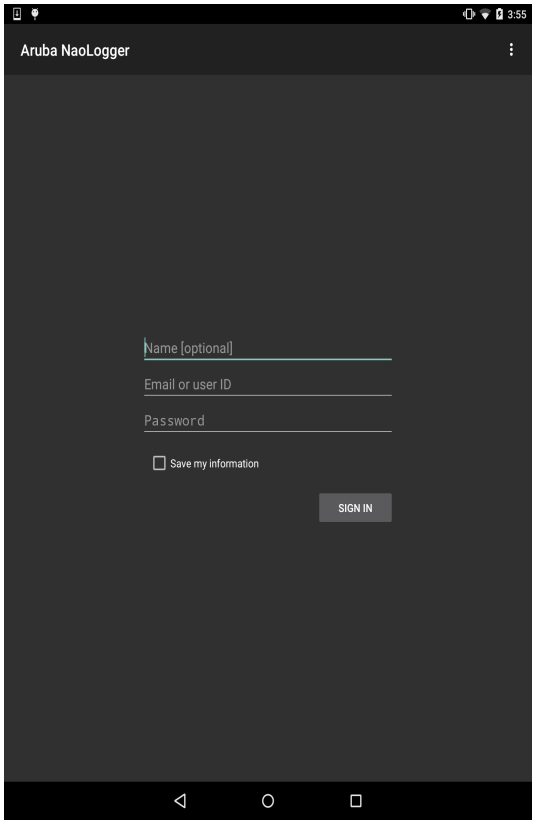
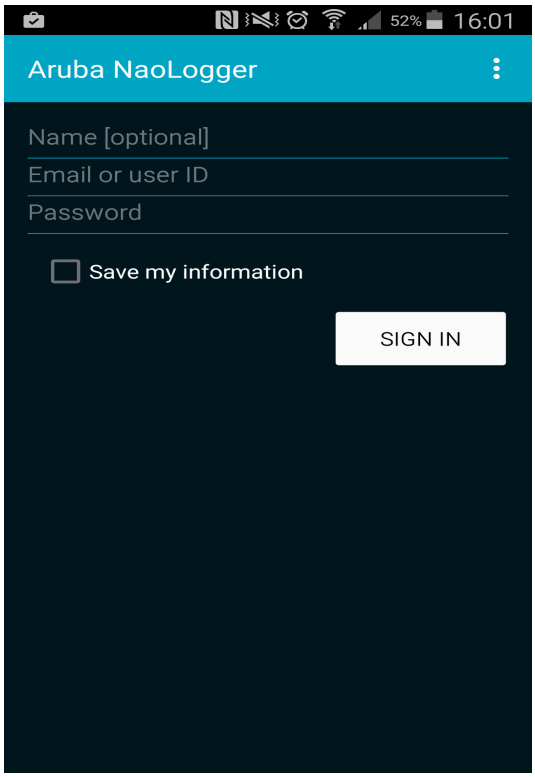


Figure 80 *Logging in on an Android phone*



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

6. The map opens and displays the following:

Table 10: *Aruba Nao Logger Map Functions*

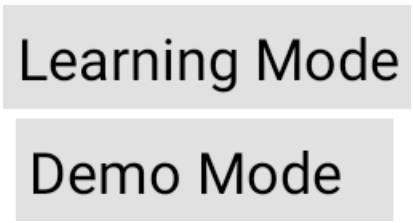
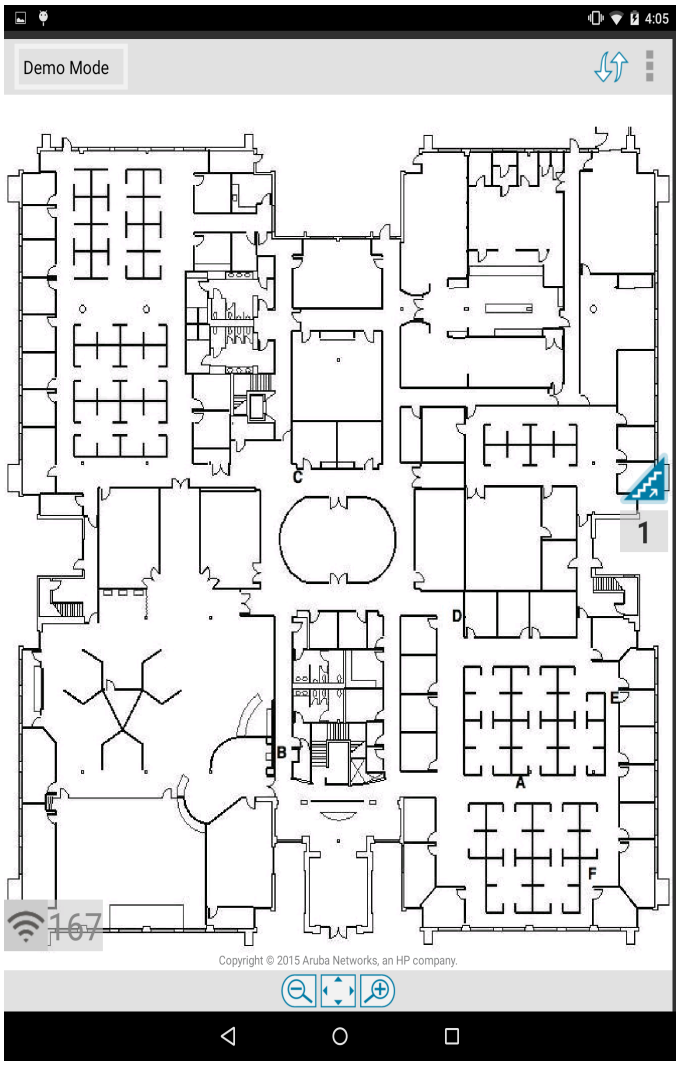
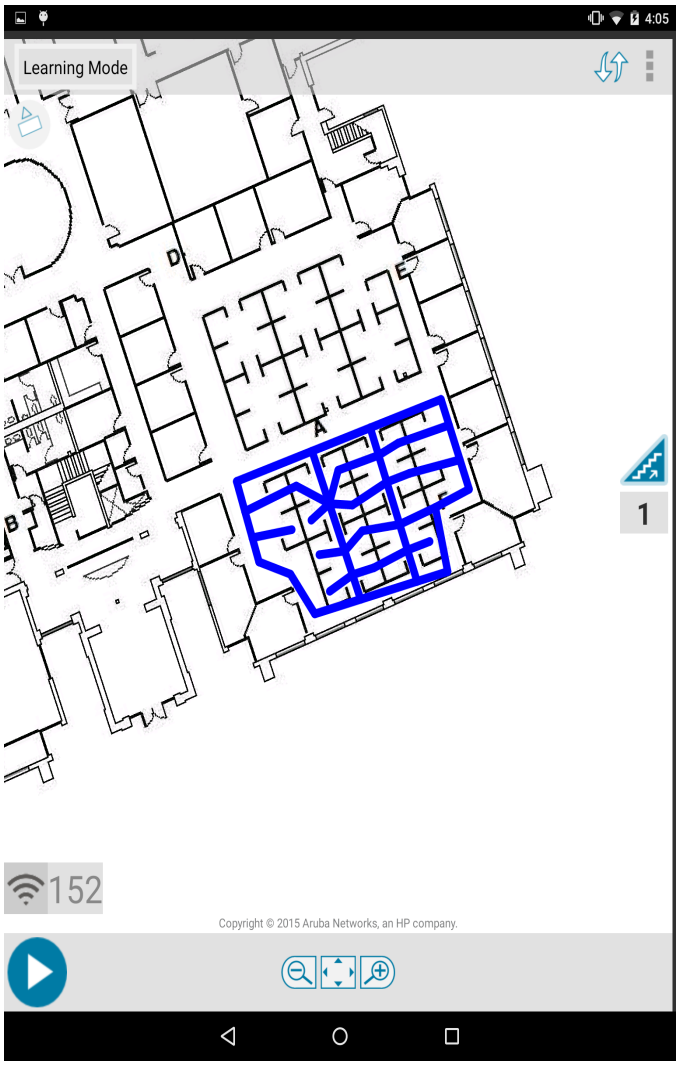
Function	Icon	Description
Mode		**Learning Mode** is used for data collection. **Demo Mode** is used for positioning testing, which uses the location algorithm running inside the Android app (based on sensors on the phone) to validate fingerprint quality. Whenever a fingerprinting operator switches to 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		**Floor Level** indicates the floor number of the current floorplan. For example, a floor level of 2 indicates that the current floorplan is on the second floor of the building.
Wi-Fi Beacons		**Wi-Fi Beacons** displays the number of Wi-Fi beacons available on a floor.
Synchronize		The **Synchronize** button sends all collected data to the server to generate the PDB.
Start/Stop		The **Start** and **Stop** button begins and ends the data-collection process.

Figure 81 *Aruba Nao Logger Overview*



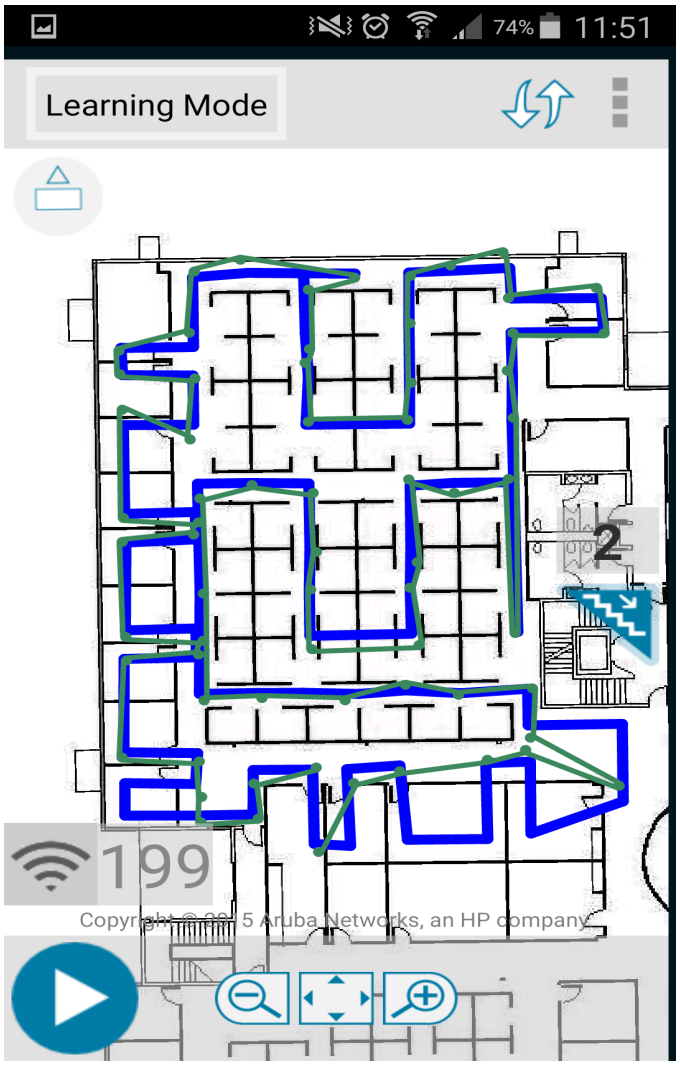
7. Set the map to **Learning Mode**, and then press the **Start** button to begin fingerprinting. The green crosshair indicates your current location.

Figure 82 *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 83 *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

All fingerprinting data associated with a building is deleted and must be regathered in a new data-collection session if any floorplan within the building is modified, a walkable path is changed, or a floor is added/deleted.

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).

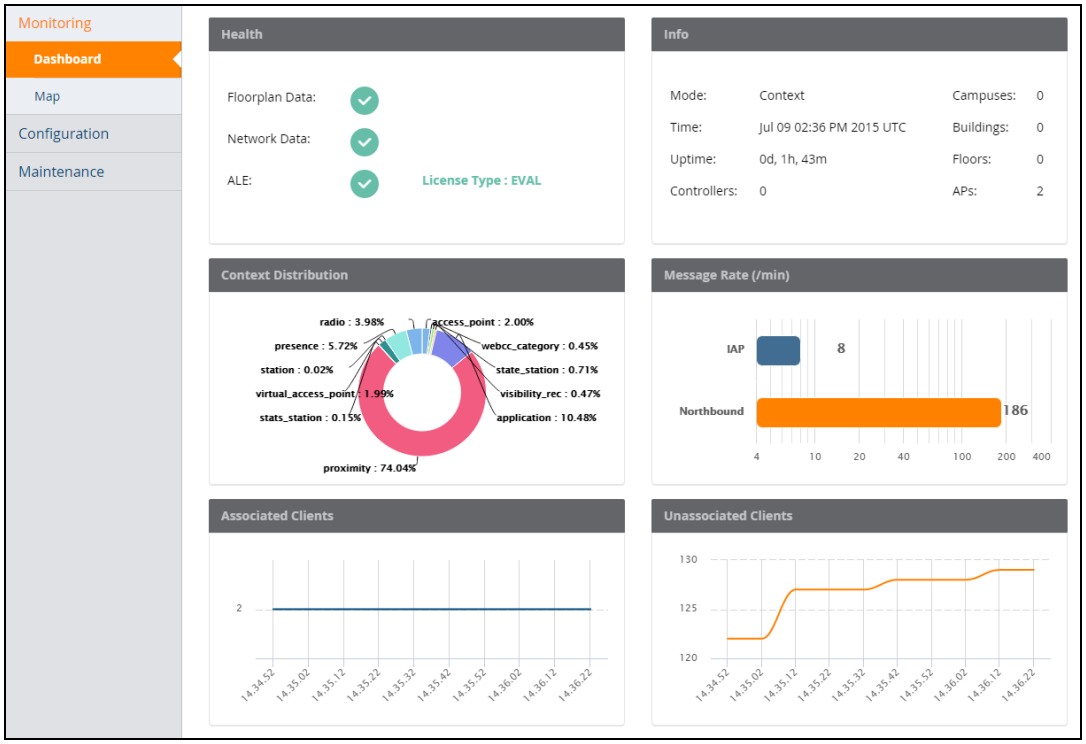
To download the .proto file:

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 84 *The ALE 2.0 Dashboard Overview*



Health

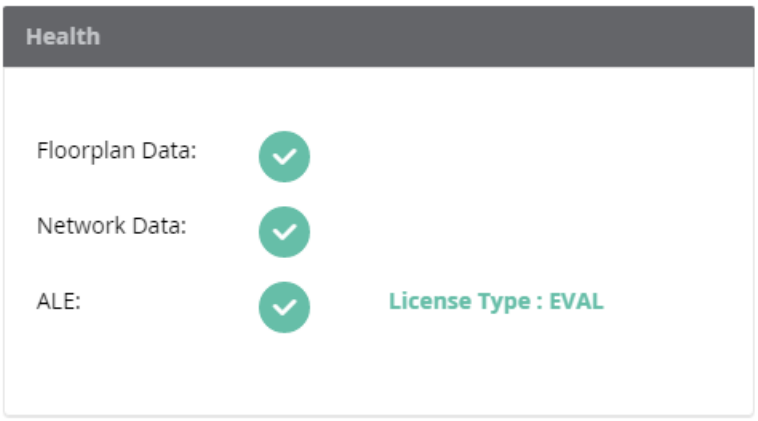
The **Health** section of the ALE Dashboard displays an overview of data and system health.

Table 11: *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 85 *ALE Dashboard - Health*



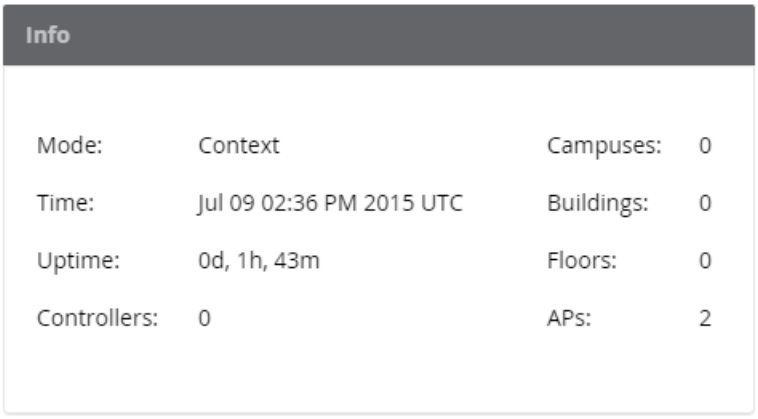
Info

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

Figure 86 *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

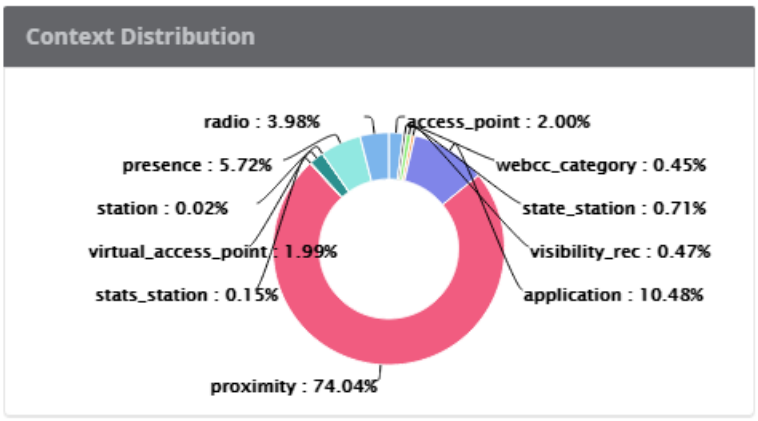
Figure 87 *ALE Dashboard - Info*



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.

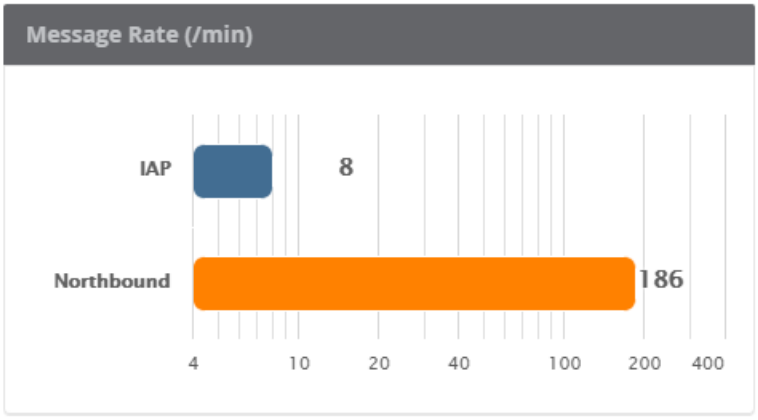
Figure 88 *ALE Dashboard - Context Distribution*



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").

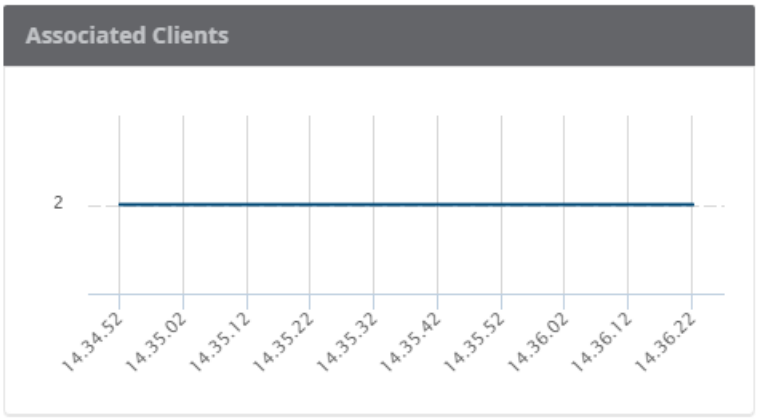
Figure 89 *ALE Dashboard - Message Rate (/min)*



Associated Clients

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

Figure 90 *ALE Dashboard - Associated Clients*



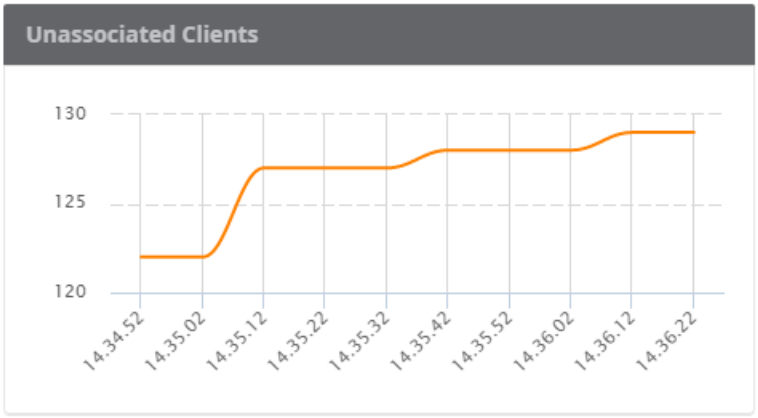
Unassociated Clients

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 91 *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 92 *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 12.

Table 12: *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 13 can be executed in the launch script.

Table 13: *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

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

Tunnel Clients

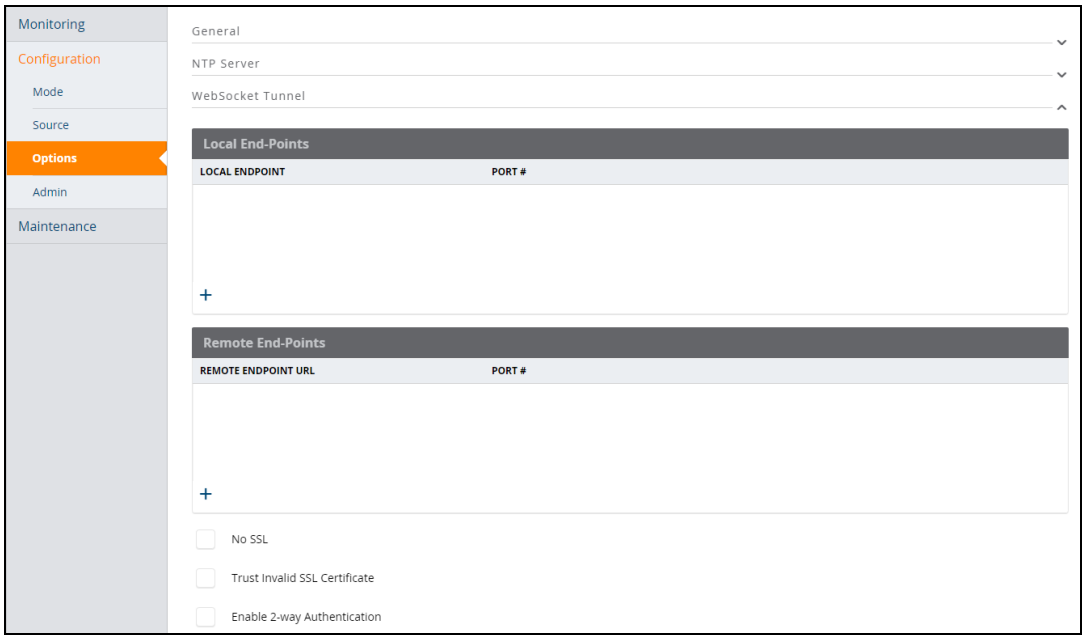
The tunnel client runs on the ALE server.

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 93 *Configuring Tunnel Clients*



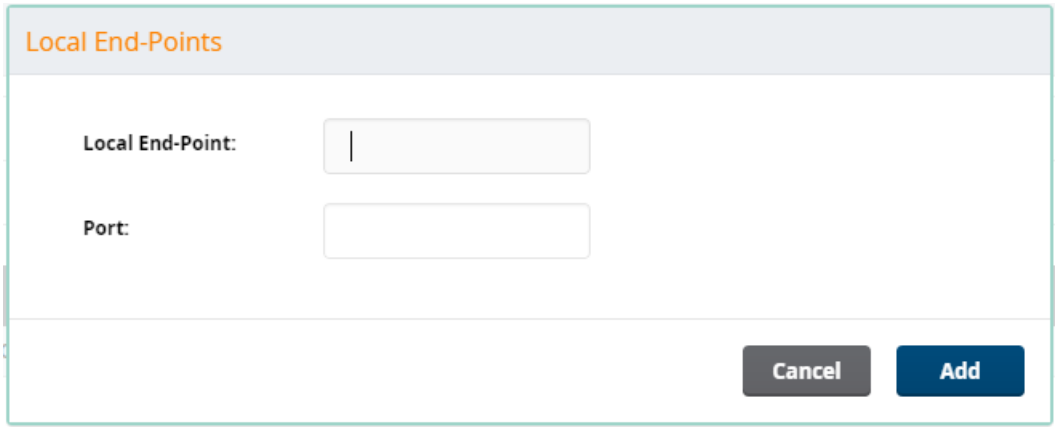
a. To add a new end-point, click the **+** button on the bottom left corner of the **Local End-Points** or **Remote End-Points** table. A pop-up window to add a new end-point opens.

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.

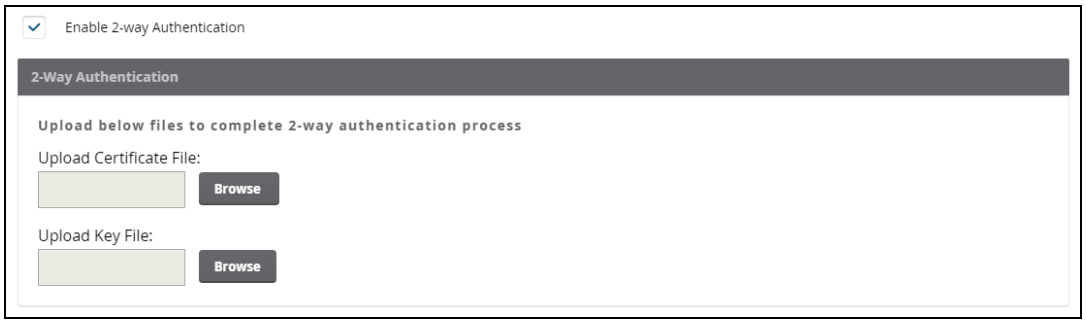
   b. To specify a remote end-point, enter the hostname or IP address of the server running the WebSocket server. The port must be set to the same port configured in the **nbapi.tunnel.server.port** field of the WebSocket server configuration.

Figure 94 *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 95 *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 96 *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
```

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 86).

```
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 Location Engine supports two types of APIs: a polling-based REST API, and a publish/subscribe API based on Google Protobuf and ZeroMQ. Refer to the *Analytics and Location Engine 2.0.0.x API Guide* for more details on 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.
- 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.

When the Analytics and Location 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.

The following section provides sample code for the ZeroMQ application:

```
#include <stdio.h>
#include <string>
#include <iostream>
#include <iomanip>
#include <typeinfo>
#include <netinet/in.h>
#include <netdb.h>
#include <zmq.h>
#include "objects/schema.pb.h"

namespace gpb = google::protobuf;

namespace

{

   const char* g_appName = NULL;
   const char* const DEFAULT_ZMQ_ENDPOINT = "tcp://localhost:7779";
   const char* const DEFAULT_ZMQ_SUB_FILTER = "";

   void usage()
   {
      printf("\n");
      printf("Usage: %s [options]\n", g_appName);
      printf("Options:\n");
      printf("  -e <endpoint>  ZMQ endpoint to connect/bind to. Default: %s\n", DEFAULT_ZMQ_
      ENDPOINT);
      printf("  -f <filter>    Message filter to apply on a ZMQ_SUB socket. Default: %s\n",
      strlen(DEFAULT_ZMQ_SUB_FILTER) ? DEFAULT_ZMQ_SUB_FILTER : "<empty>");
      printf("  -b             Listen for ZMQ endpoint on port 7779. Default: connect\n");
      printf("\n");
      exit(1);
   }

   std::ostream& printIndent(std::ostream& os, int indent)
   {
      for (int i = 0; i < indent; ++i)
         os << "    ";
      return os;
   }

   std::ostream& printIpAddress(std::ostream& os, const ce::nbapi::ip_address& msg, int
   indent)
   {
      static const size_t IPV6_SIZE = (sizeof(uint8_t) * 16);
      static const size_t IPV4_SIZE = sizeof(uint32_t);

      if (msg.has_af())
      {
         ce::nbapi::ip_address::addr_family af = msg.af();
         printIndent(os, indent); os << "af: " << msg.addr_family_Name(af) << "\n";

         if (msg.has_addr())
         {
            const std::string& addrTmp = msg.addr();
            struct sockaddr_in sin;
```

```
            struct sockaddr_in6 sin6;
            socklen_t salen;
            int error = -1;
            char nameInfo[80];

            if (af == ce::nbapi::ip_address::ADDR_FAMILY_INET6)
            {
               memset(&sin6, 0, sizeof(struct sockaddr_in6));
               sin6.sin6_family = AF_INET6;
               memcpy(sin6.sin6_addr.s6_addr, addrTmp.data(), IPV6_SIZE);
               salen = sizeof(struct sockaddr_in6);
               error = getnameinfo((struct sockaddr *)&sin6, salen, nameInfo, sizeof(nameInfo),
               NULL, 0, NI_NUMERICHOST);
            }
            else
            {
               memset(&sin, 0, sizeof (struct sockaddr_in));
               sin.sin_family =  AF_INET;
               memcpy(&sin.sin_addr.s_addr, addrTmp.data(), IPV4_SIZE);
               salen = sizeof(struct sockaddr_in);
               error = getnameinfo((struct sockaddr *)&sin, salen, nameInfo, sizeof(nameInfo),
               NULL, 0, NI_NUMERICHOST);
            }

            if (!error)
               printIndent(os, indent); os << "addr: " << nameInfo << "\n";
         }
      }
      return os;
   }

   std::ostream& printMacAddress(std::ostream& os, const ce::nbapi::mac_address& msg, int
   indent)
   {
      if (msg.has_addr())
      {
         const std::string& addrTmp = msg.addr();
         if (addrTmp.size() == 6)
         {
            const unsigned char* addrData = (const unsigned char*)addrTmp.data();
            char macAddStr[18];
            sprintf(macAddStr, "%02x:%02x:%02x:%02x:%02x:%02x",
               addrData[0], addrData[1], addrData[2],
               addrData[3], addrData[4], addrData[5]);
            printIndent(os, indent); os << "addr: " << macAddStr << "\n";
         }
      }
      return os;
   }

   std::ostream& printMessage(std::ostream& os, const gpb::Message& msg, int indent)
   {
      const gpb::Descriptor* desc = msg.GetDescriptor();

      try
      {
         if (desc == ce::nbapi::ip_address::descriptor())
            return printIpAddress(os, dynamic_cast<const ce::nbapi::ip_address&>(msg), indent);
         else if (desc == ce::nbapi::mac_address::descriptor())
            return printMacAddress(os, dynamic_cast<const ce::nbapi::mac_address&>(msg),
            indent);
      }
```

```
catch (const std::bad_cast& e)
{
return os;
}

const gpb::Reflection* refl = msg.GetReflection();
std::vector<const gpb::FieldDescriptor*> fieldDescList;

refl->ListFields(msg, &fieldDescList);
for (size_t i = 0; i < fieldDescList.size(); ++i)
{
   const gpb::FieldDescriptor* fieldDesc = fieldDescList[i];
   int fieldSize = fieldDesc->is_repeated() ? refl->FieldSize(msg, fieldDesc) : -1;
   int k;

   switch (fieldDesc->cpp_type())
   {
      case gpb::FieldDescriptor::CPPTYPE_MESSAGE:
      {
         if (fieldDesc->is_repeated())
         {
            for (k = 0; k < fieldSize; ++k)
            {
               printIndent(os, indent); os << fieldDesc->name();
               os << " {\n";
               printMessage(os, refl->GetRepeatedMessage(msg, fieldDesc, k), indent+1);
               printIndent(os, indent); os << "}\n";
            }
         }
         else
         {
            printIndent(os, indent); os << fieldDesc->name();
            os << " {\n";
            printMessage(os, refl->GetMessage(msg, fieldDesc), indent+1);
            printIndent(os, indent); os << "}\n";
         }
      }
      break;
      case gpb::FieldDescriptor::CPPTYPE_INT32:
      {
         int32_t tmpInt32;
         if (fieldDesc->is_repeated())
         {
            for (k = 0; k < fieldSize; ++k)
            {
               tmpInt32 = refl->GetRepeatedInt32(msg, fieldDesc, k);
               printIndent(os, indent); os << fieldDesc->name();
               os << ": " << tmpInt32 << "\n";
            }
         }
         else
         {
            tmpInt32 = refl->GetInt32(msg, fieldDesc);
            printIndent(os, indent); os << fieldDesc->name();
            os << ": " << tmpInt32 << "\n";
         }
      }
      break;
      case gpb::FieldDescriptor::CPPTYPE_INT64:
      {
         int64_t tmpInt64;
```

```
   if (fieldDesc->is_repeated())
   {
      for (k = 0; k < fieldSize; ++k)
      {
         tmpInt64 = refl->GetRepeatedInt64(msg, fieldDesc, k);
         printIndent(os, indent); os << fieldDesc->name();
         os << ": " << tmpInt64 << "\n";
      }
   }
   else
   {
      tmpInt64 = refl->GetInt64(msg, fieldDesc);
      printIndent(os, indent); os << fieldDesc->name();
      os << ": " << tmpInt64 << "\n";
   }
}
break;
case gpb::FieldDescriptor::CPPTYPE_UINT32:
{
   uint32_t tmpUInt32;
   if (fieldDesc->is_repeated())
   {
      for (k = 0; k < fieldSize; ++k)
      {
         tmpUInt32 = refl->GetRepeatedUInt32(msg, fieldDesc, k);
         printIndent(os, indent); os << fieldDesc->name();
         os << ": " << tmpUInt32 << "\n";
      }
   }
   else
   {
      tmpUInt32 = refl->GetUInt32(msg, fieldDesc);
      printIndent(os, indent); os << fieldDesc->name();
      os << ": " << tmpUInt32 << "\n";
   }
}
break;
case gpb::FieldDescriptor::CPPTYPE_UINT64:
{
   uint64_t tmpUInt64;
   if (fieldDesc->is_repeated())
   {
      for (k = 0; k < fieldSize; ++k)
      {
         tmpUInt64 = refl->GetRepeatedUInt64(msg, fieldDesc, k);
         printIndent(os, indent); os << fieldDesc->name();
         os << ": " << tmpUInt64 << "\n";
      }
   }
   else
   {
      tmpUInt64 = refl->GetUInt64(msg, fieldDesc);
      printIndent(os, indent); os << fieldDesc->name();
      os << ": " << tmpUInt64 << "\n";
   }
}
break;
case gpb::FieldDescriptor::CPPTYPE_DOUBLE:
{
   double tmpDouble;

   if (fieldDesc->is_repeated())
```

```
      {
         for (k = 0; k < fieldSize; ++k)
         {
            tmpDouble = refl->GetRepeatedDouble(msg, fieldDesc, k);
            printIndent(os, indent); os << fieldDesc->name();
            os << ": " << tmpDouble << "\n";
         }
      }
      else
      {
         tmpDouble = refl->GetDouble(msg, fieldDesc);
         printIndent(os, indent); os << fieldDesc->name();
         os << ": " << tmpDouble << "\n";
      }
   }
   break;
   case gpb::FieldDescriptor::CPPTYPE_FLOAT:
   {
      float tmpFloat;
      if (fieldDesc->is_repeated())
      {
         for (k = 0; k < fieldSize; ++k)
         {
            tmpFloat = refl->GetRepeatedFloat(msg, fieldDesc, k);
            printIndent(os, indent); os << fieldDesc->name();
            os << ": " << tmpFloat << "\n";
         }
      }
      else
      {
         tmpFloat = refl->GetFloat(msg, fieldDesc);
         printIndent(os, indent); os << fieldDesc->name();
         os << ": " << tmpFloat << "\n";
      }
   }
   break;
   case gpb::FieldDescriptor::CPPTYPE_BOOL:
   {
      bool tmpBool;

      if (fieldDesc->is_repeated())
      {
         for (k = 0; k < fieldSize; ++k)
         {
            tmpBool = refl->GetRepeatedBool(msg, fieldDesc, k);
            printIndent(os, indent); os << fieldDesc->name();
            os << ": " << (tmpBool ? "true" : "false")  << "\n";
         }
      }
      else
      {
         tmpBool = refl->GetBool(msg, fieldDesc);
         printIndent(os, indent); os << fieldDesc->name();
         os << ": " << (tmpBool ? "true" : "false")  << "\n";
      }
   }
   break;
   case gpb::FieldDescriptor::CPPTYPE_ENUM:
   {
      const gpb::EnumValueDescriptor* enumDesc;

      if (fieldDesc->is_repeated())
```

```
      {
         for (k = 0; k < fieldSize; ++k)
         {
            enumDesc = refl->GetRepeatedEnum(msg, fieldDesc, k);
            printIndent(os, indent); os << fieldDesc->name();
            os << ": " << enumDesc->name() << "\n";
         }
      }
      else
      {
         enumDesc = refl->GetEnum(msg, fieldDesc);
         printIndent(os, indent); os << fieldDesc->name();
         os << ": " << enumDesc->name() << "\n";
      }
   }
   break;
   case gpb::FieldDescriptor::CPPTYPE_STRING:
   {
      std::string tmpString;

      if (fieldDesc->is_repeated())
      {
         for (k = 0; k < fieldSize; ++k)
         {
            tmpString = refl->GetRepeatedStringReference(msg, fieldDesc, k,
            &tmpString);

            printIndent(os, indent); os << fieldDesc->name();

            if (fieldDesc->type() == gpb::FieldDescriptor::TYPE_STRING)
               os << ": " << tmpString << "\n";
            else
            {
               char* tmpSz = new char[(tmpString.size()*2)+1];
               const unsigned char* data = (const unsigned char*)tmpString.data();
               for (size_t j=0; j < tmpString.size(); ++j)
               sprintf(&tmpSz[j*2], "%02X", data[j]);
               os << ": " << tmpSz << "\n";
               delete[] tmpSz;
            }
         }
      }
      else
      {
         tmpString = refl->GetStringReference(msg, fieldDesc, &tmpString);

         printIndent(os, indent); os << fieldDesc->name();

         if (fieldDesc->type() == gpb::FieldDescriptor::TYPE_STRING)
            os << ": " << tmpString << "\n";
         else
         {
            char* tmpSz = new char[(tmpString.size()*2)+1];
            const unsigned char* data = (const unsigned char*)tmpString.data();
            for (size_t j=0; j < tmpString.size(); ++j)
            sprintf(&tmpSz[j*2], "%02X", data[j]);
            os << ": " << tmpSz << "\n";
            delete[] tmpSz;
         }
      }
   }
   break;
```

```
            default:
               printIndent(os, indent); os << ": ???\n";
               break;
         }
      }
      return os;
   }

   std::ostream& operator<<(std::ostream& os, const ce::nbapi::nb_event& ev)
   {
      return printMessage(os, ev, 0);
   }
}

int main(int argc, char* argv[])
{
   g_appName = argv[0];
   std::string endpoint(DEFAULT_ZMQ_ENDPOINT);
   std::string filter(DEFAULT_ZMQ_SUB_FILTER);
   int c;
   bool doBind = false;

   while((c = getopt(argc, argv, "hf:e:b")) != -1)
   {
      switch (c)
      {
         case 'f':
            if (optarg && optarg[0])
               filter.assign(optarg);
            break;
         case 'e':
            if (optarg && optarg[0])
               endpoint.assign(optarg);
            break;
         case 'b':
            doBind = true;
            break;
         default:
            usage();
            break;
         };
      }

      void* ctx = zmq_ctx_new();
      if (!ctx)
         perror("zmq_ctx_new");
      assert(ctx);

      void* sub = zmq_socket(ctx, ZMQ_SUB);
      if (!sub)
         perror("zmq_socket");
      assert(sub);

      if (doBind)
      {
         endpoint.assign("tcp://*:7779");
         printf("Attempting to 'bind' to endpoint: %s\n", endpoint.c_str());
         if (zmq_bind(sub, endpoint.c_str()) != 0)
         {
            perror("zmq_bind");
            assert(0);
         }
```

```
    }
    else
    {
       printf("Attempting to 'connect' to endpoint: %s\n", endpoint.c_str());
       if (zmq_connect(sub, endpoint.c_str()) != 0)
       {
          perror("zmq_connect");
          assert(0);
       }
    }
    printf("Connected to endpoint: %s\n", endpoint.c_str());

    if (zmq_setsockopt(sub, ZMQ_SUBSCRIBE, filter.c_str(), filter.size()) != 0)
    {
       perror("zmq_setsockopt");
       assert(0);
    }
    printf("Subscribed to topic: \"%s\"\n", filter.c_str());

    zmq_msg_t zmsg;
    size_t cnt = 1;
    std::string strTopic;
    ce::nbapi::nb_event ev;
    int rc;

    while (true)
    {
       bool more = true;
       size_t partNum = 0;

       while (more)
       {
          zmq_msg_init(&zmsg);
          rc = zmq_msg_recv(&zmsg, sub, 0);
          more = zmq_msg_more(&zmsg) != 0;

          if (rc < 0)
          {
             perror("zmq_msg_recv");
             more = false;
          }
          else
          {
             if (partNum == 0)
             {
                strTopic.assign(static_cast<const char*>(zmq_msg_data(&zmsg)), zmq_msg_size
                (&zmsg));
                printf("[%zu] Recv event with topic \"%s\"\n", cnt++, strTopic.c_str());
             }
             else
             {
                ev.Clear();
                if (ev.ParseFromArray(zmq_msg_data(&zmsg), zmq_msg_size(&zmsg)))
                   std::cout << ev << std::endl;
                else
                   printf("Protobuf failed to parse event\n");
             }
          }
          zmq_msg_close(&zmsg);
          ++partNum;
       }
    }
```

```
    std::cout << "Cleaning..." << std::endl;

    if (zmq_close(sub) != 0)
       perror("zmq_close");

    if (zmq_ctx_destroy(ctx) != 0)
       perror("zmq_close");

    return 0;
}
```