Cisco ACI App Center Developer Guide

First Published: 2017-01-18

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: http://www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

© 2017 Cisco Systems, Inc. All rights reserved.

CONTENTS

Preface

This preface includes the following sections:

Audience

This guide is intended primarily for data center administrators with responsibilities and expertise in one or more of the following:

- Virtual machine installation and administration
- Server administration
- Switch and network administration

Document Conventions

Command descriptions use the following conventions:

Convention	Description
bold	Bold text indicates the commands and keywords that you enter literally as shown.
Italic	Italic text indicates arguments for which the user supplies the values.
[x]	Square brackets enclose an optional element (keyword or argument).

Convention	Description
[x | y]	Square brackets enclosing keywords or arguments separated by a vertical bar indicate an optional choice.
{x | y}	Braces enclosing keywords or arguments separated by a vertical bar indicate a required choice.
[x {y | z}]	Nested set of square brackets or braces indicate optional or required choices within optional or required elements. Braces and a vertical bar within square brackets indicate a required choice within an optional element.
`variable`	Indicates a variable for which you supply values, in context where italics cannot be used.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Examples use the following conventions:

Convention	Description
`screen font`	Terminal sessions and information the switch displays are in screen font.
`boldface screen font`	Information you must enter is in boldface screen font.
italic screen font	Arguments for which you supply values are in italic screen font.
< >	Nonprinting characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

This document uses the following conventions:

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Warning IMPORTANT SAFETY INSTRUCTIONS

This warning symbol means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents. Use the statement number provided at the end of each warning to locate its translation in the translated safety warnings that accompanied this device.

SAVE THESE INSTRUCTIONS

Related Documentation

Cisco Application Centric Infrastructure (ACI) Documentation

The ACI documentation is available at the following URL: http://www.cisco.com/c/en/us/support/cloud-systems-management/application-policy-infrastructure-controller-apic/tsd-products-support-series-home.html.

Cisco Application Centric Infrastructure (ACI) Simulator Documentation

The Cisco ACI Simulator documentation is available at http://www.cisco.com/c/en/us/support/cloud-systems-management/application-centric-infrastructure-simulator/tsd-products-support-series-home.html.

Cisco Nexus 9000 Series Switches Documentation

The Cisco Nexus 9000 Series Switches documentation is available at http://www.cisco.com/c/en/us/support/switches/nexus-9000-series-switches/tsd-products-support-series-home.html.

Cisco Application Virtual Switch Documentation

The Cisco Application Virtual Switch (AVS) documentation is available at http://www.cisco.com/c/en/us/support/switches/application-virtual-switch/tsd-products-support-series-home.html.

Cisco Application Centric Infrastructure (ACI) Integration with OpenStack Documentation

Cisco ACI integration with OpenStack documentation is available at http://www.cisco.com/c/en/us/support/cloud-systems-management/application-policy-infrastructure-controller-apic/tsd-products-support-series-home.html.

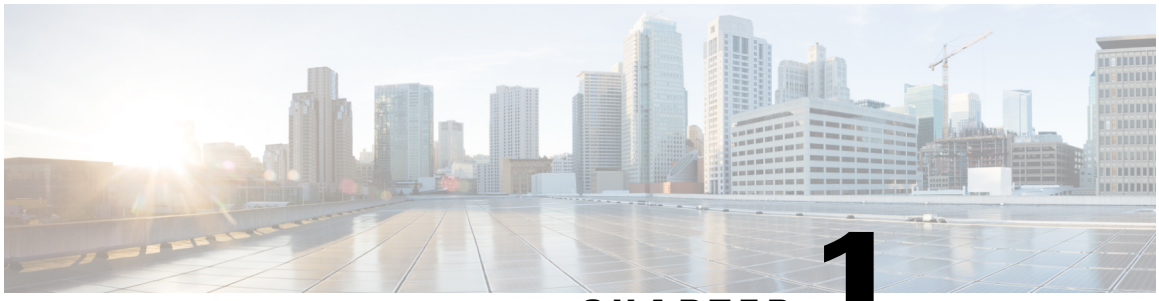
Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to apic-docfeedback@cisco.com. We appreciate your feedback.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, using the Cisco Bug Search Tool (BST), submitting a service request, and gathering additional information, see *What's New in Cisco Product Documentation* at: http://www.cisco.com/c/en/us/td/docs/general/whatsnew/whatsnew.html

Subscribe to *What's New in Cisco Product Documentation*, which lists all new and revised Cisco technical documentation as an RSS feed and delivers content directly to your desktop using a reader application. The RSS feeds are a free service.



CHAPTER 1

Overview

This chapter contains the following sections:

About Cisco ACI App Center

The Cisco ACI App Center allows you to fully enable the capabilities of the APIC by writing applications running on the controller. Using the Cisco ACI App Center, customers, developers, and partners will be able to build applications to simplify, enhance, and visualize their use cases. These applications are hosted and shared at the Cisco ACI App Center and installed in the APIC.

APIC supports two types of applications:

- Stateful
- Stateless

About Stateless Applications

A stateless application (app) is a simple HTML, CSS, or JavaScript based front-end that is run as part of the APIC UI. These apps are also referred to as front-end only applications. They can be launched from the **Apps** tab in the APIC UI or can be inserted as a separate tab in any part of the APIC UI.

Stateless apps are inserted in the APIC UI as an IFRAME. In this type of applications, app specific state is stored on the APIC for that app. The app queries APIC using its northbound REST APIs and retrieves information from the APIC. In stateless app, no state is maintained in the between two invocations of the app.

Figure 1: Stateless Application

Some of the common examples for stateless app include the following:

- Data visualization apps that gather data available from querying the APIC and that can present them in a visual format.
- L4-L7 vendor specific configuration apps.

Launching a Stateless Application

Depending on how the stateless application is designed, it can be launched in the following ways:

- If the app is inserted as part of the APIC's UI, the admin or user can navigate to the specific section in the APIC UI and access the app by clicking on the corresponding tab for the app.
- In the APIC UI, the admin can navigate to **Apps** > **Installed Apps** and then double-click the app to create a tab in the APIC UI that will contain the application.

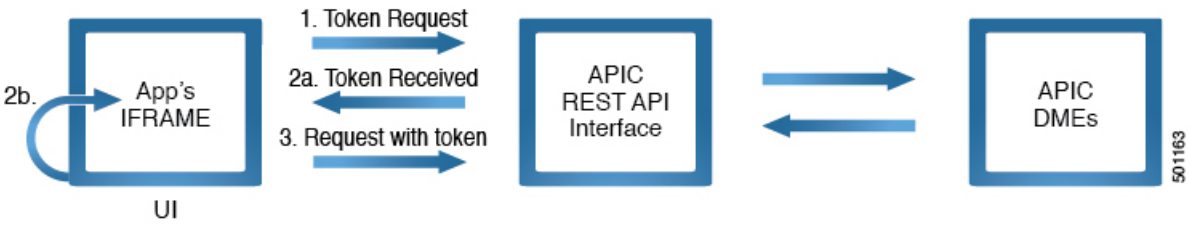
In both cases, an app stops functioning when admin or user navigates away from the **App** tab. An admin or tenant may invoke the app by navigating to the specific section of the APIC UI.

Single Sign On for Stateless Application

An admin or tenant does not require a new log in mechanism to launch a stateless app. Stateless apps use the same session as the admin or tenant that is currently logged in to the APIC UI.

When an admin installs an app, APIC creates an user and role for the app. This user and role has the privilege as described by the app's metadata in the `app.json` file.

Figure 2: SSO Sequence for Stateless Application



The SSO sequence for stateless apps consists of the following steps:

When an admin user clicks on the app to launch the app, the APIC UI launches the app in the IFRAME.

1. APIC UI makes a request to APIC server for a token that can be used by the app for SSO.
2. Upon receiving the token, the APIC UI passes the token to the app by using the token passing mechanism provided in the `app-start.html` file included in the app.
3. The app uses this token to make REST API calls to APIC.

About Stateful Applications

A stateful application (app) has a backend service that runs continuously on the APIC. Consequently, the app may store a state in this backend for specific functions. Stateful app's backend service is run on APIC in a sandboxed containerized environment; namely a docker container. The service makes queries to the APIC using the APIC's REST API interface. A stateful app may also have a front-end component in addition to

backend component. This front-end component is inserted in the APIC UI as an IFRAME, in the same way as a stateless app. If a stateful app is developed without a front-end, then it is installed using REST APIs.

Figure 3: Stateful Application with a Front-End

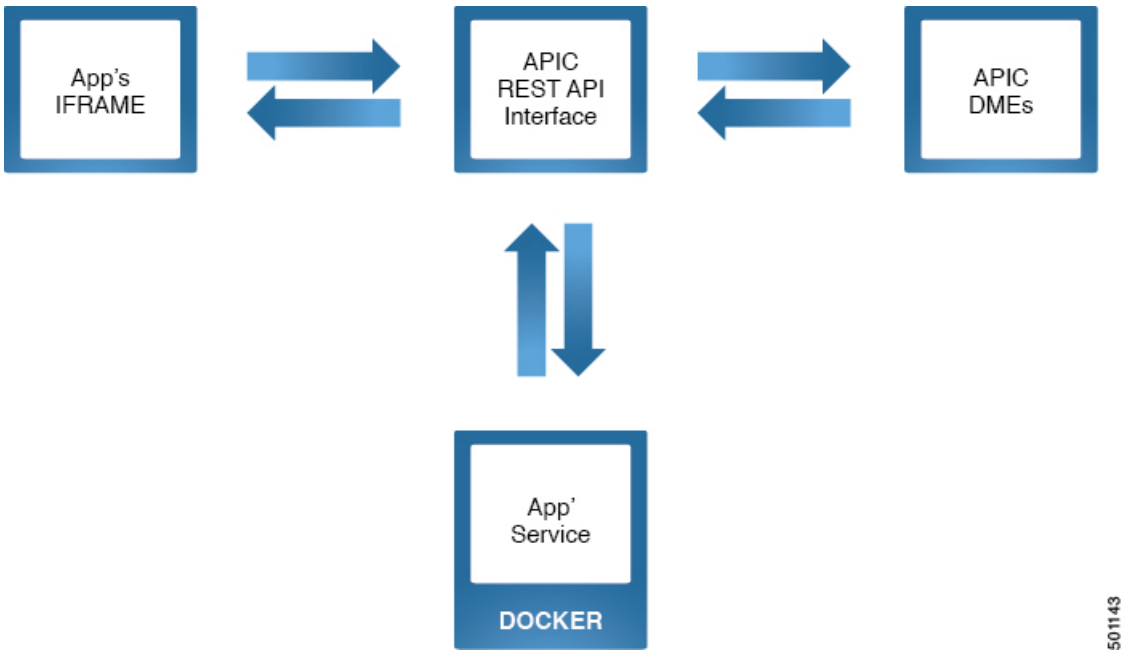
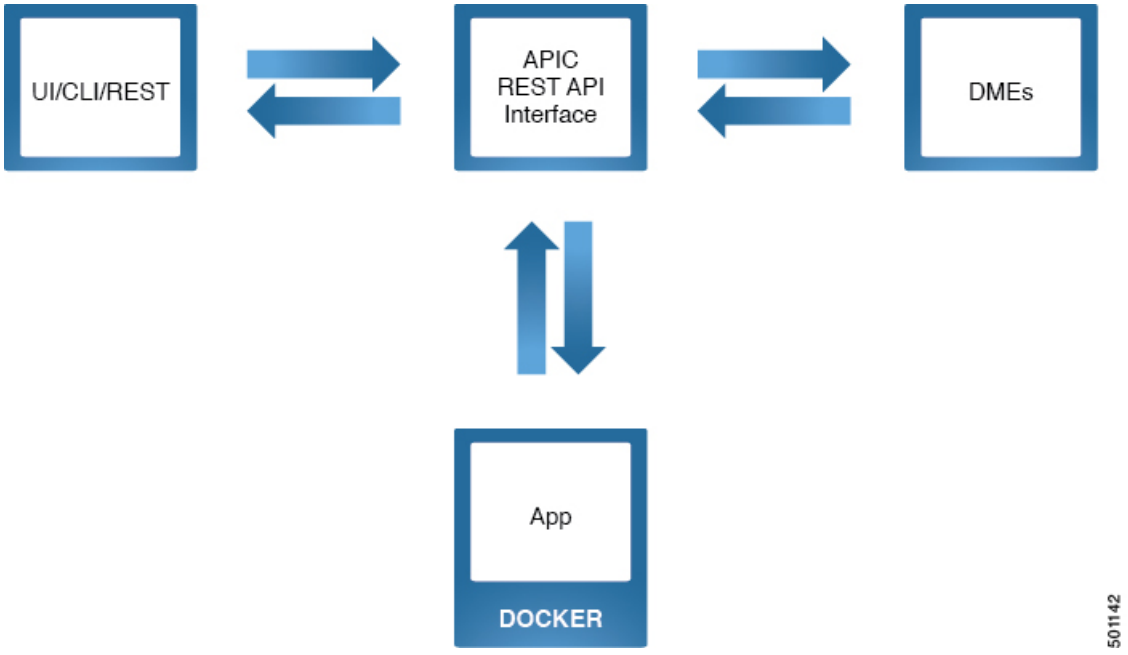


Figure 4: Stateful Application without a Front-End



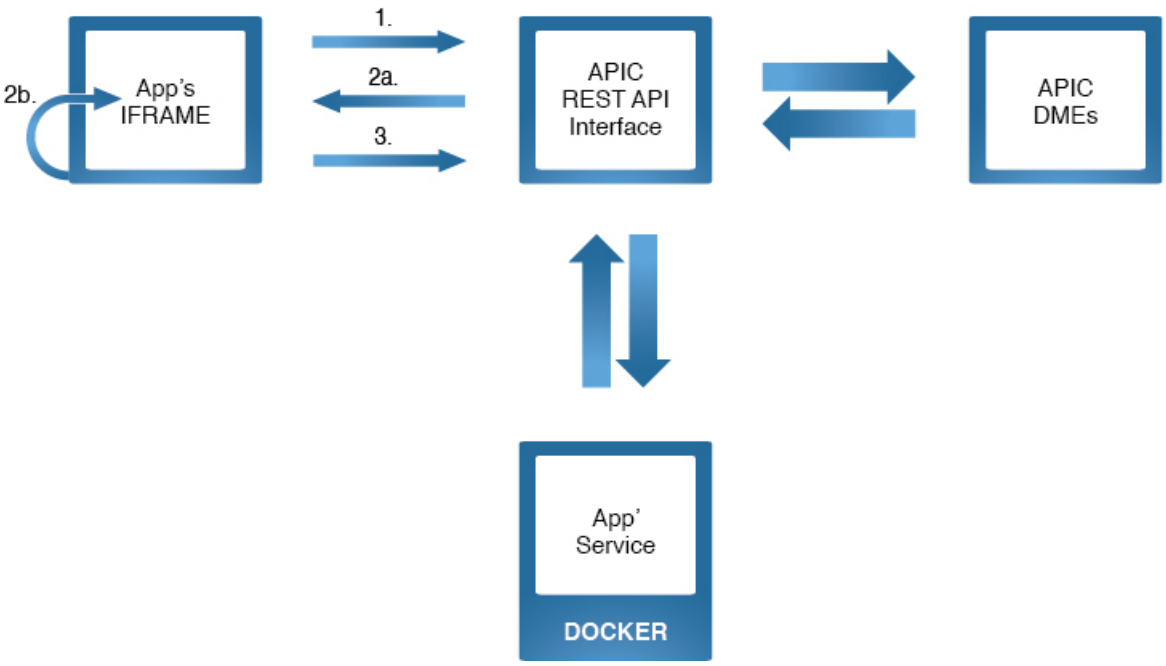
Some of the common examples for stateful app include the following:

- Visualization Apps that can plot graphs for the historical data for a specific time interval.
- Alerts apps that can send alerts based on certain events that are not supported natively in APIC.
- Monitoring apps that can track APIC's events, faults, and statistics and analyze it for detecting anomalies.
- Apps to sync the data between APIC and a third party vendor.

Single Sign On for Stateful Application

When an admin installs an app, APIC creates an user and role for the app. This user and role has the privilege as described by the app's metadata in the `app.json` file.

Figure 5: SSO Sequence for Stateful Application



The SSO sequence for stateful apps consists of the following steps:

When an admin user clicks on the app to launch the app, the APIC UI launches the app in the IFRAME.

SSO between the app UI and the APIC backend

1. APIC UI makes a request to APIC server for a token that can be used by the app for SSO.
2. Upon receiving the token, the APIC UI passes the token to the app by using the token passing mechanism provided in the `app-start.html` file included in the app.
3. The app uses this token to make REST API calls to APIC.

SSO between the app container and the APIC backend

1. APIC provides a private key to the app's container. In the container, the key is available in the directory `/home/app/credentials`.

2 The app uses the key to authenticate.

3 APIC authenticates based on key and replies accordingly.

Understanding Permissions for an Application

The APIC provides access according to a user's role through role-based access control (RBAC). A Cisco Application Centric Infrastructure (ACI) app user (aaaAppUser) is associated with the following:

- A set of roles called permissions
- Permission level: read-only or write (read and write)
- One or more security domain tags that identify the portions of the management information tree (MIT) that a user can access

The permissions and permission level for an application are defined in the `app.json` file. See Metadata Required for Developing an Application, on page 14. The security domains for an app are associated when it is installed. Before you install an app, you can review the permissions, permission level, and security domain for an app.

When an app is installed in APIC, an app user and role is created for the app using the meta data from the `app.json` file. Both the app user and the role have the same name, corresponding to the following format:

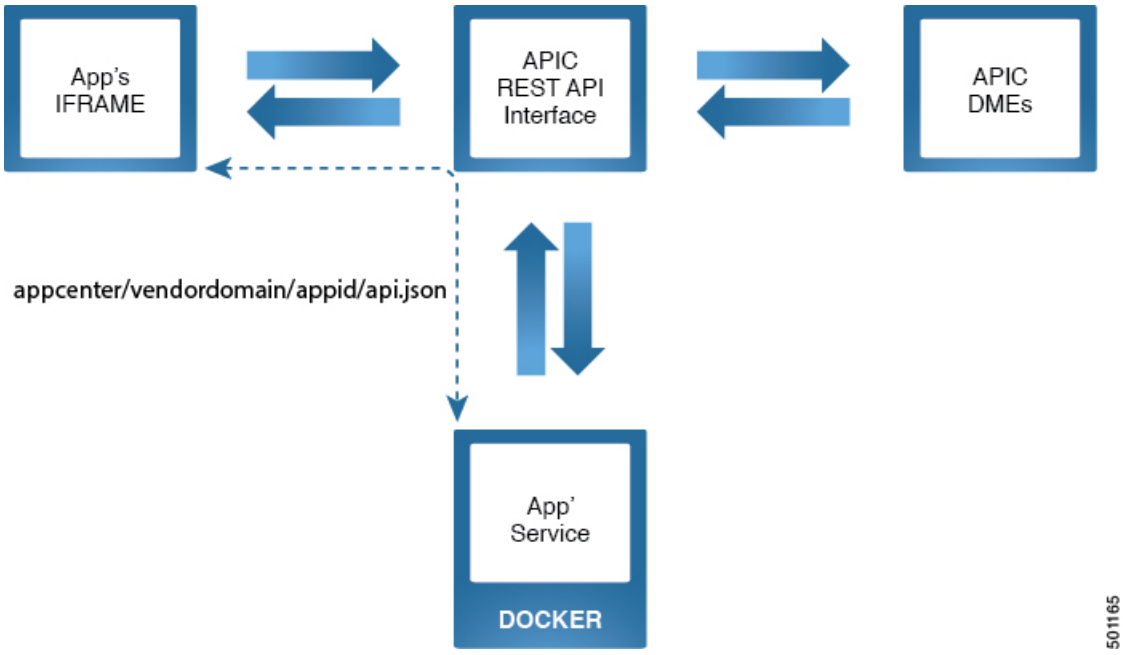
```
vendordomain_appid
```

All the queries made by the app is restricted to user, role, and the security domain created for this app. You can limit the managed objects an app can access using RBAC. It is recommended to assign the minimum set of permissions that is required for the app's functionality. See Permissions, on page 53.

See *Cisco ACI AAA RBAC Rules and Privileges* for more information about user roles, privileges, and security domains.

Understanding Application Communication

Figure 6: Application Communication



The front-end of an app communicates with the backend by issuing the following API call to the app backend running in docker, where `api.json or api.xml` in an API path provided by the application.

```
GET/POST https://APIC_IP/appcenter/vendordomain/appid/api.json
Or
GET/POST https://APIC_IP/appcenter/vendordomain/appid/api.xml
```

The `vendordomain` and `appid` are specific to the app and is defined in the `app.json` file.

Note The IP address of APIC IP is always 172.17.0.1 in relation to the docker. When the backend makes calls to the APIC, it uses the IP address 172.17.0.1.

You can retrieve the *APIC_IP* in JavaScript, using document.location.origin.

The request is then forwarded to the docker instance where the app is running. The app returns a response which is then forwarded back to the front-end. The API URLs must be declared in the `app.json` file and only authenticated users can make the API call.

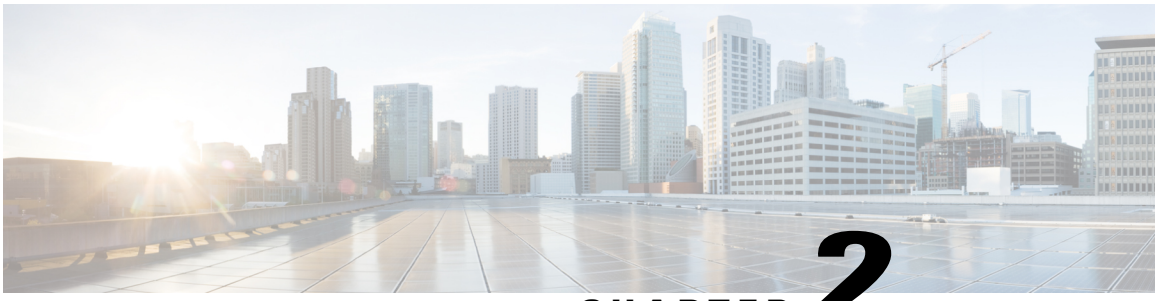
During the installation of an app, a user and role are created and then a certificate key pair is assigned to the user. When the app is installed, the private key is added to the docker image. The private key is located in the docker image in the directory `/home/apps/credentials`. Using the private key, the app then creates a session with NGINX for the user. Once the session is created, API calls can be made to retrieve the MIT information. Since the session was created for a user, the app is limited to access the information only available to the user and user is limited to the permission as defined in the `app.json` file.

Docker instances can be located on different APICs. It is not recommended to have communication between docker instances located on different APICs or the same APIC.

See Signing in to the APIC from the Application Using RBAC, on page 32.

Requirements for Developing an Application

- Stateless app can be developed using the APIC Simulator. To develop and test a stateless app, download the APIC Simulator OVA.
- To develop a stateful app, you must have access to the Cisco APIC image, release 2.2(1x) and later. You will also need access to a docker environment running a docker container.
- Stateless apps are web applications containing HTML, CSS, or JavaScript files.
- A stateful app can be developed using any language inside the docker container. Since APIC supports python bindings for its API, python may be preferred for developing a stateful app.



CHAPTER 2

Developing a Stateless Application

This chapter contains the following sections:

Components of Stateless Application

A Stateless application includes the following files:

- `app.json` — A JSON file containing the metadata required for developing an application. The metadata also informs the APIC on where to insert the app in the APIC UI. See Metadata Required for Developing an Application, on page 14.
- `app.html` — A HTML file that implements the UI or the front-end of the application.
- `app-start.html` — It contains information to receive the tokens from the APIC and coverts the data into a cookie.

Workflow for Developing a Stateless Application

Use this procedure to develop a stateless application.

Step 1 Setting up the directory structure and the files required for the application.
See Creating Directory Structure for a Stateless Application, on page 12.

Step 2 Creating the metadata for the application.
See Metadata Required for Developing an Application, on page 14.

Step 3 Packaging the application.
See Packaging an Application, on page 36.

Step 4 (Optional) Enabling signature validation for an application.
This step is applicable only if you are publishing the app to Cisco ACI App Center.See Enabling Signature Validation for an Application, on page 40

Step 5 Do one of the following:

- Publishing the application to Cisco ACI App Center for external distribution. See Publishing an Application, on page 39.
- Uploading the application to APIC for internal distribution. See Uploading an Application to APIC, on page 40.

Step 6 Download the app from Cisco ACI App Center.
This step is required only if you are publishing the app to Cisco ACI App Center. See Downloading Application From Cisco ACI App Center , on page 40.

Step 7 Installing and launching the application.
See Installing an Application, on page 41.

Prerequisites

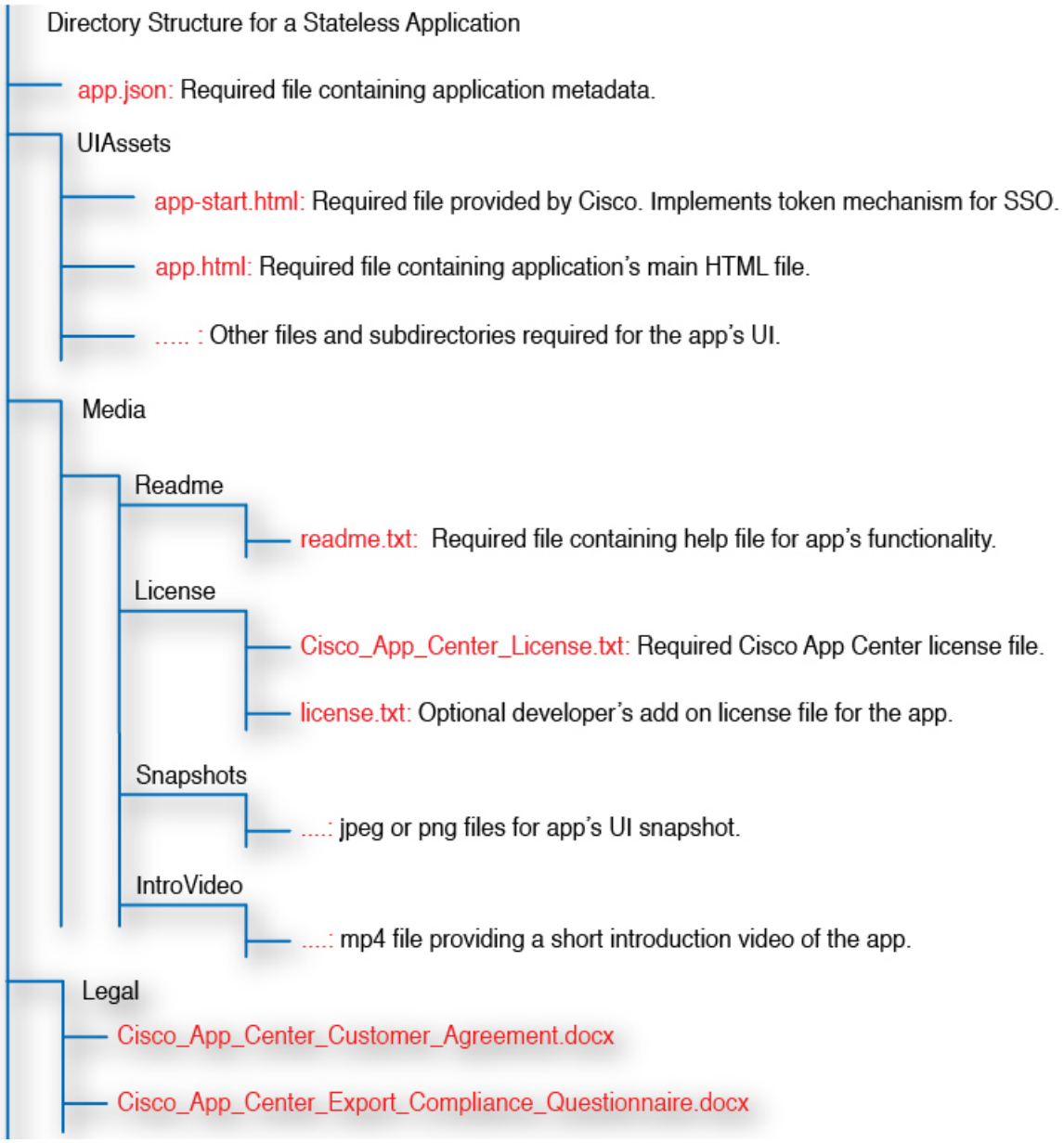
- You have obtained the `app-start.html` file from Cisco DevNet to be included in the application.
- You must have a developer account to access the Cisco ACI App Center.
- You have read the Cisco App Center Development Principles and Guidelines.

Guidelines and Limitations

- You must configure NTP policy to keep the time in all APICs in sync. This requirement is necessary since the X509 certificate could be generated on one APIC and validated on a different APIC.

Directory Structure for Stateless Application

Figure 7: Recommended Directory Structure for a Stateless Application



Creating Directory Structure for a Stateless Application

Use this procedure to create the directory structure and all the files required for developing a stateless application for the Cisco ACI App Center. See the *Appendix* for examples of the various files required to develop the application.

Step 1 Create a directory for the app you are developing in your workspace. All the folders and files required for developing the application must be added to this folder.

Step 2 Create the metadata for the app in the `app.json` file.
This file is required and has information required by the Cisco ACI App Center to recognize the app and validate it. See Metadata Required for Developing an Application, on page 14 for information regarding the metadata required for the `app.json` file.

Step 3 Create a `Media` folder and the files specified in this folder for your app.
This folder contains the following folders and files:

- `Readme` (Required) — The readme directory contains the `readme.txt` file and cannot be empty. When you publish the app to the Cisco ACI App Center, the `readme.txt` file is used to present the information about the app to the user on the app description page in the Cisco ACI App Center.
- `License` (Required) — The license folder contains the required `Cisco_App_Center_License.txt` file. It is the Cisco license file for the app and is added automatically when using the Cisco packager. Optionally, the developer can also add a separate app specific license file for the app in this location.
- `Snapshots` (Optional) — The snapshot folder contains files which provide a preview of the app before the user downloads the app from the Cisco ACI App Center. It is optional and provides information regarding the app in various modes.
- `IntroVideo` (Optional) — The IntroVideo folder is optional. It contains a video which introduces the app and give information on how the app works. The supported format for the video is mp4.

Step 4 Create a `Legal` folder and add the files containing the legal information required for your app.
The directory must include the following two required files. These files are automatically provided when using the Cisco packager to package an app.

- `Cisco_App_Center_Customer_Agreement.docx`
- `Cisco_App_Center_Export_Compliance_Questionnaire.docx`

Step 5 Create a `UIAssets` folder and the files specified in this folder for your app.
The `UIAssets` folder is the core folder which contains all the intelligence about the app. This folder contains the HTML, CSS, and JavaScript files for the app. This folder must at least include the following files:

- `app.html` (Required) — A required HTML file that implements the UI or the front-end of the application. The content of this file is specific to the app. This file contains the HTML page that will be embedded in APIC's UI. It can import various others files such as CSS or Javascript files provided within the UIAssets folder. This file must contain the function to use the tokens specified in `app-start.html`.

- `app-start.html` (Required) — A HTML file provided by Cisco and can be downloaded from Cisco DevNet. Every application must include this file for single-sign on to work. It is recommended that you do not modify this file.

  It contains the cookie information to implement the single sign-on in an application. It contains the cookie data and the mechanism to retrieve the data from APIC. This file must contain the data for the cookies, token and challenge. The value of the cookie is sent to APIC as headers as part of each request made from app's UI to avail single sign-on. This file also includes the loading sequence for an app. It contains a message which is displayed when the app is being loaded.

  It contains information to receive the tokens from the APIC and coverts the data into a cookie. You must then get the tokens used in a cookie and use it in further requests.

  APIC regularly sends a token to the application. The app must have the mechanism to receive and update its token accordingly. You can retrieve the token using `Ext.util.Cookies.get`, each time you make a request.

```
<script type="text/javascript">
    window.addEventListener('message', function (e) {
        if (e.source === window.parent) {
            var tokenObj = Ext.decode(e.data, true);
            if (tokenObj) {
// Setting the cookie with the tokens received by the APIC
                Ext.util.Cookies.set('app_' + tokenObj.appID + '_token', tokenO
bj.token);
                Ext.util.Cookies.set('app_' + tokenObj.appID + '_urlToken', tokenO
bj.urlToken);
            }
        }
    });
  </script>
```

  Another option for implementation, is to store the tokens from the cookie in variables. In this example, the application HelloAci uses `window.APIC_DEV_COOKIE` and `window.APIC_URL_TOKEN` when sending a request.

```
<script type="text/javascript">
    window.APIC_DEV_COOKIE = Ext.util.Cookies.get("app_Cisco_HelloAciStateful_token");
    window.APIC_URL_TOKEN =  Ext.util.Cookies.get("app_Cisco_HelloAciStateful_urlToken");

    window.addEventListener('message', function (e) {
        if (e.source === window.parent) {
            var tokenObj = Ext.decode(e.data, true);
            if (tokenObj) {
                window.APIC_DEV_COOKIE = tokenObj.token;
                window.APIC_URL_TOKEN = tokenObj.urlToken;
            }
} });
</script>
```

  App requests made to the APIC require a custom `DevCookie` and `APIC-Challenge` header to be set for proper authentication. After accessing the tokens, you must include them in the HTTP requests headers.

```
window.APIC_DEV_COOKIE = Ext.util.Cookies.get("app_Cisco_HelloAciStateful_token");
window.APIC_URL_TOKEN =  Ext.util.Cookies.get("app_Cisco_HelloAciStateful_urlToken");
```

```
HttpRequest.prototype.query = function (query, success_func)
{
    var Http = new XMLHttpRequest();
    var url1 = 'https://'+gSystem+'/'+query;

    Http.open("GET", url1, false);

    Http.setRequestHeader("DevCookie",window.APIC_DEV_COOKIE);
    Http.setRequestHeader("APIC-Challenge",window.APIC_URL_TOKEN);
    Http.setRequestHeader ("Accept", "application/json");

    Http.onreadystatechange = function(){
        if (Http.readyState==4){
            if (Http.status==200){
                success_func(Http.responseText);
            }
        }
    }

    Http.send();
};
```

- Other files and sub directories required for the app's UI.

Metadata Required for Developing an Application

The `app.json` file contains metadata for the app. The following table lists the metadata required for developing an app for the Cisco ACI App Center.

Table 1: Metadata to be Specified in the `app.json` File

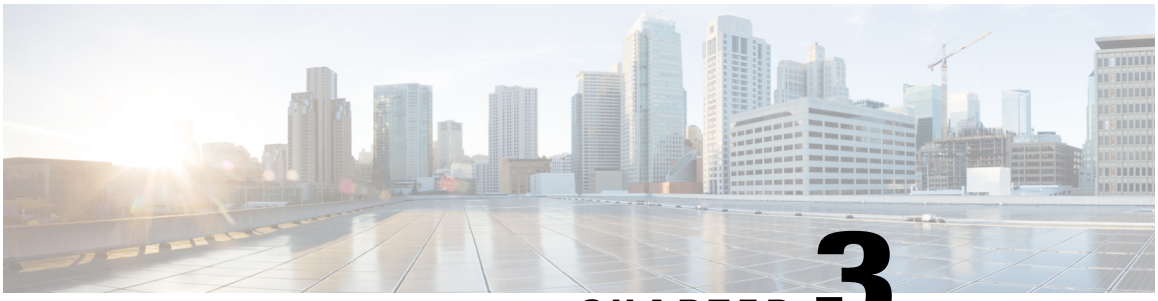
Parameter	Mandatory	Can Be Updated	Description	Restrictions
appid	Yes	No	The unique ID for the app which is used to identify the app in the Cisco ACI App Center and in the APIC.	The ID is a string and can be up to 32 alpha numeric characters only.
version	Yes	Yes	Version of the application specifying a major *M* version and a minor *m* version.	The version is a string *M.m* and *M* and *m* are positive integers. The values of *M* and *m* are in the range 0 to 9999.

Parameter	Mandatory	Can Be Updated	Description	Restrictions
iconfile	Yes	Yes	The path to the icon file in the UIAssets folder. The icon file contains the thumbnail for the app. The thumbnail is used to uniquely identify the app in the Cisco ACI App Center.	The path to the icon file is a string and the file name can be up to 256 characters. The supported file formats are jpeg or png.
name	Yes	No	The name of the application.	The name is a string and can be up to 256 characters.
shortdescr	Yes	Yes	The description of the app. This information is displayed when the app is listed in the Cisco ACI App Center.	The short description is a string and can be up to 1024 characters.
vendor	Yes	No	The name of the company.	The name is a string and can be up to 256 alpha numeric characters only.
vendordomain	Yes	No	The domain ID of the company.	The ID is a string and can be up to 32 alpha numeric characters only.
apicversion	Yes	Yes	The minimum APIC software version required for the app's functionality.	Must be unique and greater than last approved application. The format is *major.minor(mp), where* • m=maintenance • p=patch • major, minor, maintenance is 0 <= and <=999 • Patch is character [a-z]

Parameter	Mandatory	Can Be Updated	Description	Restrictions
signature	No	Yes	The signature required for the application files. The signature is issued by Cisco ACI App Center after an app is approved and is allowed to be distributed. **Note** Only apps with the signature are supported by Cisco. Apps without the signature are not supported.	None
price	No	Yes	Total cost to download the app. The default is $0.	The format for the value is float.
contact	No	Yes	The contact information for the app.	The format is JSON dictionary.
contact-phone	No	Yes	The contact phone number of the company.	The phone number is a string.
contact-email	No	Yes	The contact email of the company.	The email is a string.
contact-url	No	Yes	The contact URL of the company.	The URL is a string.
permissions	Yes	Yes	The permissions required by the app to access the various managed objects and utilities in the MIT. See *Cisco ACI AAA RBAC Rules and Privileges* for more information about user roles, privileges, and security domains. **Note** It is recommended to assign the minimum set of permissions that is required for the app's functionality.	The format is JSON array.

Parameter	Mandatory	Can Be Updated	Description	Restrictions
permissionslevel	Yes	No	The permission level required for the application. The permission level defines if the user has read or write access to the app.	The permission level is a string and can be either read or write.
api **Note** The metadata is only applicable for a stateful app.	Yes	Yes	The API supported by the application. Each entry in the API list contains the API URL and the corresponding description. The API is used to query the backend. **Note** In the API, only POST and GET operations are supported.	The format is JSON dictionary. The key and value are strings.
author	Yes	Yes	The name of the app developer.	The author is a string and can be up to 256 characters.
insertionURL	No	Yes	The insertion URL of the app in the APIC UI. The URL informs the APIC on where to insert the app in the APIC UI. By default, the user will be able to run the app from the **Installed Apps** tab. See Integrating the App's UI in the APIC UI, on page 52.	The insertion URL is a string.

Parameter	Mandatory	Can Be Updated	Description	Restrictions
category	Yes	Yes	The categories of the app used by Cisco ACI App Center to filter the apps. The allowed categories allowed are: • Tools and Utilities • Visibility and Monitoring • Optimization • Security • Networking • Cisco Automation and Orchestration	The format is JSON array.



CHAPTER 3

Developing a Stateful Application

This chapter contains the following sections:

Components of Stateful Application

A Stateful application includes the following files:

- `app.json` — A JSON file containing the metadata required for developing an application. The metadata also informs the APIC on where to insert the app in the APIC UI. See Metadata Required for Developing an Application, on page 14.
- `app.html` — A HTML file that implements the UI or the front-end of the application.
- `app-start.html` — It contains information to receive the tokens from the APIC and coverts the data into a cookie.
- `.tgz` — A `.tgz` file such as `aci_appcenter_docker_image.tgz` containing the docker image. A docker image contains all the packages required by the app to implement the backend. The image can contain packages such as web server to open the API, OpenSSL for security, Cisco APIC Python SDK (cobra) for querying the APIC.
- `start.sh`— A script containing the initializations required by the application. This script is executed automatically after the docker image is installed.

Workflow for Developing a Stateful Application

Use this procedure to develop a stateful application.

Step 1 Setting up the directory structure and the files required for the application.
See Creating Directory Structure for a Stateless Application, on page 12.

Step 2 Creating the metadata for the application.
See Metadata Required for Developing an Application, on page 14.

Step 3 Signing on to the APIC from the app.
See Signing in to the APIC from the Application Using RBAC, on page 32

Step 4 Packaging the application.
See Packaging an Application, on page 36.

Step 5 (Optional) Enabling signature validation for an application.
This step is applicable only if you are publishing the app to Cisco ACI App Center.See Enabling Signature Validation for an Application, on page 40

Step 6 Do one of the following:

- Publishing the application to Cisco ACI App Center for external distribution. See Publishing an Application, on page 39.
- Uploading the application to APIC for internal distribution. See Uploading an Application to APIC, on page 40.

Step 7 Download the app from Cisco ACI App Center.
This step is required only if you are publishing the app to Cisco ACI App Center. See Downloading Application From Cisco ACI App Center , on page 40.

Step 8 Installing and launching the application.
See Installing an Application, on page 41.

Prerequisites

- You have obtained the `app-start.html` file from Cisco DevNet to be included in the application.
- You have obtained the reference docker image provided by Cisco from Cisco DevNet for developing a stateful application.
- You must have a developer account to access the Cisco ACI App Center.
- You have read the Cisco App Center Development Principles and Guidelines.

Guidelines and Limitations

- The size of the docker image should not exceed 1 GB.
- Every stateful application must have a separate docker image. Sharing of docker images is currently not supported.
- You must configure NTP policy to keep the time in all APICs in sync. This requirement is necessary since the X509 certificate could be generated on one APIC and validated on a different APIC.

Directory Structure for Stateful Application

Figure 8: Recommended Directory Structure for a Stateful Application

Creating Directory Structure for a Stateful Application

Use this procedure to create the directory structure and all the files required for developing a stateful application for the Cisco ACI App Center. See the *Appendix* for examples of the various files required to develop the application.

Step 1 Create a directory for the app you are developing in your workspace. All the folders and files required for developing the application must be added to this folder.

Step 2 Create the metadata for the app in the `app.json` file.
This file is required and has information required by the Cisco ACI App Center to recognize the app and validate it. See Metadata Required for Developing an Application, on page 14 for information regarding the metadata required for the `app.json` file.

Step 3 Create a `Media` folder and the files specified in this folder for your app.
This folder contains the following folders and files:

- `Readme` (Required) — The readme directory only contains the `readme.txt` file and cannot be empty. When you publish the app to the Cisco ACI App Center, the `readme.txt` file is used to present the information about the app to the user on the app description page in the Cisco ACI App Center.
- `License` (Required) — The license folder contains the `Cisco_App_Center_License.txt` file. It is the Cisco license file for the app and is added automatically when using the Cisco packager. Optionally, the developer can also add a separate app specific license file for the app in this location.
- `Snapshots` (Optional) — The snapshot folder contains files which provide a preview of the app before the user downloads the app from the Cisco ACI App Center. It is optional and provides information regarding the app in various modes.
- `IntroVideo` (Optional) — The IntroVideo folder is optional. It contains a video which introduces the app and give information on how the app works. The supported format for the video is mp4.

Step 4 Create a `Legal` folder and add the files containing the legal information required for your app.
The directory must include the following two files. These files are automatically provided when using the Cisco packager to package an app.

- `Cisco_App_Center_Customer_Agreement.docx`
- `Cisco_App_Center_Export_Compliance_Questionnaire.docx`

Step 5 Create a `UIAssets` folder and the files specified in this folder for your app.
The `UIAssets` folder is the core folder which contains all the intelligence about the app. This folder contains the HTML, CSS, and JavaScript files for the app. This folder must at least include the following files:

- `app.html` (Required) — A HTML file that implements the UI or the front-end of the application. The content of this file is specific to the app. This file contains the HTML page that will be embedded in APIC's UI. It can import various others files such as CSS or Javascript files provided within the UIAssets folder. This file must contain the function to use the tokens specified in `app-start.html`.

- `app-start.html` (Required) — A HTML file provided by Cisco and can be downloaded from Cisco DevNet. Every application must include this file for single-sign on to work. It is recommended that you do not modify this file.

  It contains the cookie information to implement the single sign-on in an application. It contains the cookie data and the mechanism to retrieve the data from APIC. This file must contain the data for the cookies, token and challenge. The value of the cookie is sent to APIC as headers as part of each request made from app's UI to avail single sign-on.

  This file also includes the loading sequence for an app. It contains a message which is displayed when the app is being loaded.

  It contains information to receive the tokens from the APIC and coverts the data into a cookie. You must then get the tokens used in a cookie and use it in further requests.

  APIC regularly sends a token to the application. The app must have the mechanism to receive and update its token accordingly. You can retrieve the token using `Ext.util.Cookies.get`, each time you make a request.

```
<script type="text/javascript">
    window.addEventListener('message', function (e) {
        if (e.source === window.parent) {
            var tokenObj = Ext.decode(e.data, true);
            if (tokenObj) {
// Setting the cookie with the tokens received by the APIC
                Ext.util.Cookies.set('app_' + tokenObj.appID + '_token', tokenO
bj.token);
                Ext.util.Cookies.set('app_' + tokenObj.appID + '_urlToken', tokenO
bj.urlToken);
            }
        }
    });
  </script>
```

  Another option for implementation, is to store the tokens from the cookie in variables. In this example, the application HelloAci uses `window.APIC_DEV_COOKIE` and `window.APIC_URL_TOKEN` when sending a request.

```
<script type="text/javascript">
    window.APIC_DEV_COOKIE = Ext.util.Cookies.get("app_Cisco_HelloAciStateful_token");
    window.APIC_URL_TOKEN =  Ext.util.Cookies.get("app_Cisco_HelloAciStateful_urlToken");

    window.addEventListener('message', function (e) {
        if (e.source === window.parent) {
            var tokenObj = Ext.decode(e.data, true);
            if (tokenObj) {
                window.APIC_DEV_COOKIE = tokenObj.token;
                window.APIC_URL_TOKEN = tokenObj.urlToken;
            }
} });
</script>
```

App requests made to the APIC require a custom `DevCookie` and `APIC-Challenge` header to be set for proper authentication. After accessing the tokens, you must include them in the HTTP requests headers.

```
window.APIC_DEV_COOKIE = Ext.util.Cookies.get("app_Cisco_HelloAciStateful_token");
window.APIC_URL_TOKEN =  Ext.util.Cookies.get("app_Cisco_HelloAciStateful_urlToken");

HttpRequest.prototype.query = function (query, success_func)
{
    var Http = new XMLHttpRequest();
    var url1 = 'https://'+gSystem+'/'+query;

    Http.open("GET", url1, false);

    Http.setRequestHeader("DevCookie",window.APIC_DEV_COOKIE);
    Http.setRequestHeader("APIC-Challenge",window.APIC_URL_TOKEN);
    Http.setRequestHeader ("Accept", "application/json");

    Http.onreadystatechange = function(){
        if (Http.readyState==4){
            if (Http.status==200){
                success_func(Http.responseText);
            }
        }
    }

    Http.send();
};
```

Step 6 Create a `Image` folder.
This folder contains the required docker image such as `aci_appcenter_docker_image.tgz` for the application. A docker image contains all the packages required by the app to implement the backend. The image can contain packages such as Web server to open the API, OpenSSL for security, Cisco APIC Python SDK (cobra) for querying the APIC. The execution environment for the app should be provided in this image. See Creating a Docker Image, on page 26 on how to create a docker image and add it to the image folder.

Cisco also provides reference docker images and this image can be downloaded from Cisco DevNet. Cisco provides the following docker images:

- Docker image containing Cobra SDK.
- Docker image containing SQLite database, Cobra SDK, and Acitoolkit.
- Docker image containing MySQL database, Cobra SDK, and Acitoolkit.

Note The size of the docker image should not exceed 1 GB. Every stateful application must have a separate docker image. Sharing of docker images is currently not supported.

If you bring your own docker image or update the Cisco's reference image, you must first unzip or untar the `docker.tgz` file, then remove the `manifest.json` file, and finally tar or zip the `docker.tgz` file.

When the docker image is mounted, it contains the following directories located in `/home/app`:

- `src` — Contains all the source files for the app.

- `credentials` — Contains the private key to query the APIC.
- `data` — Contains the data for the distributed file system in the APIC cluster.
- `logs` — Contains the logs for the app that is collected as part of tech support.

Step 7 Create a `Service` folder and the files specified in this folder for your app.
This folder contains the service files.

- `start.sh` (Required) — It contains the first script that is executed after the docker container is installed. It includes all the initializations required for the application. It also allows you to start any script specified in the docker image.
- Other files (Optional) — This folder could contain a `server.py` file that runs a Web server providing an API for the application. In this case, `start.sh` file must contain the line starting `server.py`. In this release, only python is supported as an execution environment.

Creating a Docker Image

Use this procedure to create the docker image for a stateful app and add it to the image directory. Creating a docker image is necessary, only if you want to deviate from the reference docker image provided by Cisco for the ACI App Center. The docker image must be created on a local machine or on a VM that has a docker engine installed.

Before You Begin

- You have obtained the reference docker image provided by Cisco.

Step 1 Log in to your local workspace.

Step 2 Enter the command **docker load -i** *path to base docker image* to upload the reference docker image.

Step 3 (Optional) To remove the Cobra SDK and Acitoolkit package from the reference image perform the following steps. Removing the packages, will decrease the size of the container.

a) Remove the Cobra SDK and Acitoolkit packages from the reference image.
b) Enter the commands **pip uninstall acicobra** and **pip uninstall acimodel** to remove the Cobra SDK package provided in the reference image.

c) Enter the command **pip uninstall acitoolkit** to remove the Acitoolkit package provided in the reference image.

Step 4 Enter the command **docker images** to retrieve the image ID.

Step 5 Enter the command **docker run -d** *Image_ID* **tail -f /dev/null** to run the docker container and mount the packages.

Step 6 Enter the command **docker ps** to retrieve the docker container ID.

Step 7 Enter the command **docker exec -it** *Container_ID* **/bin/bash** to connect to the docker container.

Step 8 Install the packages in the container.

Step 9 Enter the command **docker commit** *Container_ID Image_Name***:** *Tag* to commit the updates.

Step 10 Enter the command **docker save** *Image_Name***:** *Tag* | **gzip -c >** *path to the output directory* to save the image as a `tgz` file.

Note In the **docker save** command, use *Image_Name* : *Tag* and do not use *Image_ID*.

Step 11 Add the `tgz` file to the image folder of the app.

Metadata Required for Developing an Application

The `app.json` file contains metadata for the app. The following table lists the metadata required for developing an app for the Cisco ACI App Center.

Table 2: Metadata to be Specified in the `app.json` File

Parameter	Mandatory	Can Be Updated	Description	Restrictions
appid	Yes	No	The unique ID for the app which is used to identify the app in the Cisco ACI App Center and in the APIC.	The ID is a string and can be up to 32 alpha numeric characters only.
version	Yes	Yes	Version of the application specifying a major *M* version and a minor *m* version.	The version is a string *M.m* and *M* and *m* are positive integers. The values of *M* and *m* are in the range 0 to 9999.
iconfile	Yes	Yes	The path to the icon file in the UIAssets folder. The icon file contains the thumbnail for the app. The thumbnail is used to uniquely identify the app in the Cisco ACI App Center.	The path to the icon file is a string and the file name can be up to 256 characters. The supported file formats are jpeg or png.

Parameter	Mandatory	Can Be Updated	Description	Restrictions
name	Yes	No	The name of the application.	The name is a string and can be up to 256 characters.
shortdescr	Yes	Yes	The description of the app. This information is displayed when the app is listed in the Cisco ACI App Center.	The short description is a string and can be up to 1024 characters.
vendor	Yes	No	The name of the company.	The name is a string and can be up to 256 alpha numeric characters only.
vendordomain	Yes	No	The domain ID of the company.	The ID is a string and can be up to 32 alpha numeric characters only.
apicversion	Yes	Yes	The minimum APIC software version required for the app's functionality.	Must be unique and greater than last approved application. The format is *major.minor(mp), where* • m=maintenance • p=patch • major, minor, maintenance is 0 <= and <=999 • Patch is character [a-z]

Parameter	Mandatory	Can Be Updated	Description	Restrictions
signature	No	Yes	The signature required for the application files. The signature is issued by Cisco ACI App Center after an app is approved and is allowed to be distributed. **Note** Only apps with the signature are supported by Cisco. Apps without the signature are not supported.	None
price	No	Yes	Total cost to download the app. The default is $0.	The format for the value is float.
contact	No	Yes	The contact information for the app.	The format is JSON dictionary.
contact-phone	No	Yes	The contact phone number of the company.	The phone number is a string.
contact-email	No	Yes	The contact email of the company.	The email is a string.
contact-url	No	Yes	The contact URL of the company.	The URL is a string.
permissions	Yes	Yes	The permissions required by the app to access the various managed objects and utilities in the MIT. See *Cisco ACI AAA RBAC Rules and Privileges* for more information about user roles, privileges, and security domains. **Note** It is recommended to assign the minimum set of permissions that is required for the app's functionality.	The format is JSON array.

Parameter	Mandatory	Can Be Updated	Description	Restrictions
permissionslevel	Yes	No	The permission level required for the application. The permission level defines if the user has read or write access to the app.	The permission level is a string and can be either read or write.
api **Note** The metadata is only applicable for a stateful app.	Yes	Yes	The API supported by the application. Each entry in the API list contains the API URL and the corresponding description. The API is used to query the backend. **Note** In the API, only POST and GET operations are supported.	The format is JSON dictionary. The key and value are strings.
author	Yes	Yes	The name of the app developer.	The author is a string and can be up to 256 characters.
insertionURL	No	Yes	The insertion URL of the app in the APIC UI. The URL informs the APIC on where to insert the app in the APIC UI. By default, the user will be able to run the app from the **Installed Apps** tab. See Integrating the App's UI in the APIC UI, on page 52.	The insertion URL is a string.

Parameter	Mandatory	Can Be Updated	Description	Restrictions
category	Yes	Yes	The categories of the app used by Cisco ACI App Center to filter the apps. The allowed categories allowed are: • Tools and Utilities • Visibility and Monitoring • Optimization • Security • Networking • Cisco Automation and Orchestration	The format is JSON array.

Data Types for a Stateful App

Important Notes for Persisting, Encrypting, Storing, and Logging Data

- An app can create, write, or read directories or files into the `/home/app/data` directory. The app's `/home/app/data` directory is persisted and available between two invocations of the app. This directory is persisted even during APIC switch over due to APIC failure or during upgrades.
- An app must write its data into the `/home/app/data` directory and tech support logs must be written into the `/home/app/log` directory.
- The app data is already protected in the APIC, but App can implement its own encryption mechanism to store the data and decryption mechanism to read the data.
- If an App needs faster queries to the stored data, it can store the data in a database as opposed to a file. Cisco provides the following two docker images that have database support. See Cisco DevNet.
    - Docker image providing SQLite database support
    - Docker image providing MySQL database support
- An app can log errors, debugs, and warnings into the `/home/app/log` directory. These logs are local to an APIC, unlike the App data directory. Logs can be collected using tech support. See Collecting Tech Support Logs for a Stateful App, on page 43.

Signing in to the APIC from the Application Using RBAC

Use the procedure to sign in to the APIC from the app using RBAC.

Step 1 Perform the following steps on the front-end files.

a) Link `app.js` to your `app.html` front-end form.

b) From the `app.js`, obtain the `APIC_DEV_COOKIE` and `APIC_URL_Token` based on the app vendor and app name.

Example:

```
window.APIC_DEV_COOKIE = Ext.util.Cookies.get("app_Infoblox_InfobloxAci_token");
window.APIC_URL_TOKEN =  Ext.util.Cookies.get("app_Infoblox_InfobloxAci_urlToken");
```

c) Target the API of the backend server. Ensure the API is defined in the `app.json` metadata file.

Example:

```
var infobloxUrl = document.location.origin + "/appcenter/Infoblox/InfobloxAci/run_infoblox.json";
```

d) Make an `AJAX` call to send the front-end form data to the server on the backend.

Example:

```
$(document).ready(function(){
  $('#submit_button').click(function(e){
    // e.preventDefault();
    console.log('sending form data...');
    $.ajax({
      type: 'post',
      url: infobloxUrl,
      headers: {
       "DevCookie": window.APIC_DEV_COOKIE,
       "APIC-challenge": window.APIC_URL_TOKEN
      },
      // data: $(this).serialize(),
      data: JSON.stringify({
        'ip_address': document.getElementById('inputIPAddress').value,
        'username': document.getElementById('inputUsername').value,
        'password': document.getElementById('inputPassword').value
      }),
      contentType: 'application/json;charset=UTF-8',
      dataType: 'json',
      success: function(){
        alert('success');
      }
    });
  });
});
```

Step 2 Perform the following steps on the backend files.

a) Use COBRA to convert `App-Username (Vendor_AppID)` to a `Cert-User`.

Example:

```
from cobra.model.pol import Uni as PolUni
from cobra.model.aaa import UserEp as AaaUserEp
from cobra.model.aaa import AppUser as AaaAppUser
from cobra.model.aaa import UserCert as AaaUserCert

certUser = 'Infoblox_InfobloxAci'
polUni = PolUni('')
```

```
aaaUserEp = AaaUserEp(polUni)
aaaAppUser = AaaAppUser(aaaUserEp, certUser)
aaaUserCert = AaaUserCert(aaaAppUser, certUser)
```

b) Unpack `pKey` from the `plugin.key` file on app. This file is automatically generated when the app is installed.

Example:

```
pKeyFile = '/home/app/credentials/plugin.key'
with open(pKeyFile, "r") as file:
    pKey = file.read()
```

c) On the server, receive the IP address, username, and password of the app from the front-end input form through the `AJAX` call. Forward this data along with the `aaaUserCert` and `pKey` to the service's starting script of the app.

Example:

```
@app.route('/run_infoblox.json', methods=['GET', 'POST'])
def run_infoblox():
    infoblox_url = request.json["ip_address"]
    infoblox_user = request.json["username"]
    infoblox_pw = request.json["password"]

 infoblox.StartScript(infoblox_url, infoblox_user, infoblox_pw, aaaUserCert, pKey)
```

Step 3 Perform the following steps on the service's starting script file.

a) Sign a `POST` request to `/api/requestAppToken.json` including the `pKey`.

Example:

```
from OpenSSL.crypto import FILETYPE_PEM, load_privatekey, sign

uri = "/api/requestAppToken.json"

app_token_payload={"aaaAppToken":{"attributes":{"appName": "Infoblox_InfobloxAci"}}}
data = json.dumps(app_token_payload)
payLoad= "POST" + uri + data

p_key = load_privatekey(FILETYPE_PEM, self.p_key_str)

signedDigest = sign(p_key, payLoad.encode(), 'sha256')
signature = base64.b64encode(signedDigest).decode()
```

b) Create a custom-signed token specific to the `/api/requestAppToken.json` request.

Example:

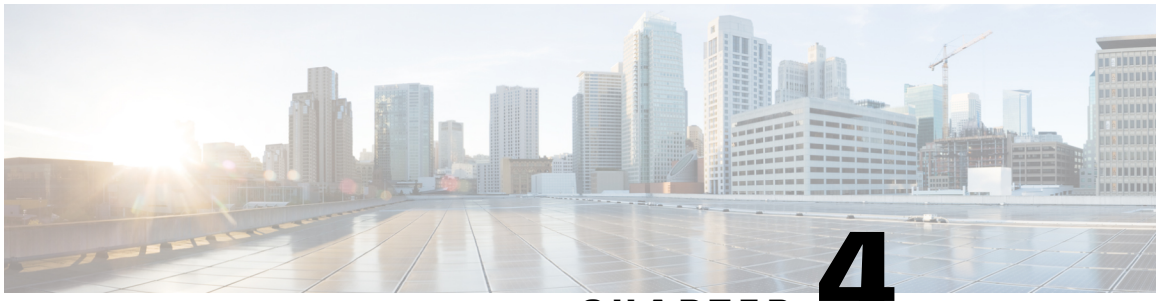
```
user_cert_str= str(self.user_cert.dn)
token = "APIC-Request-Signature=" + signature + ";"
token += "APIC-Certificate-Algorithm=v1.0;"
token += "APIC-Certificate-Fingerprint=fingerprint;"
token += "APIC-Certificate-DN=" + user_cert_str
```

c) Make a request to `/api/requestAppToken.json` using the payload specified before and validate with the token created. The IP address of APIC in relation to the app is 172.17.0.1.

Example:

```
session= requests.session()
r= session.post("http://172.17.0.1/api/requestAppToken.json", data=data, headers={'Cookie' : token
 })
auth = json.loads(r.text)
auth_token = auth['imdata'][0]['aaaLogin']['attributes']['token']
```

d) Use this `auth_token` to create more requests and to create a WebSocket connection. Refresh the `auth_token` every five minutes using the `/api/aaaRefresh.json` method. See *Cisco APIC REST API Configuration Guide* for more information.



CHAPTER 4

Packaging and Publishing an Application

This chapter contains the following sections:

About Packaging and Publishing an Application

After developing your application, you must validate and package the application. You can package an application and upload the packaged application directly to APIC or distribute the app through Cisco ACI App Center.

Prerequisites

- You have packager `cisco_aci_app_packager-1.0.tar.gz` file from Cisco DevNet. The packager file contains the packager `aci-app-packager.py`, validator `aci_app_validator.py`, and license `Cisco_App_Center_License.txt` files, required for packaging the app. The packager and validator files must be in the same directory. If the `Cisco_App_Center_License.txt` is not present, the app cannot be validated and will not be packaged.
- You have the Python version 2.7.x with the following modules:
  - Pycrypto, version 2.6.1
  - Python-magic, version 0.4.12
  - Validators, version 0.10.3
- For publishing the app to Cisco ACI App Center, you must have a developer account to access the Cisco ACI App Center.

You have generated the keys for the application, to publish the app to Cisco ACI App Center. See Generating Keys for an Application, on page 39.

Packaging an Application

Use this procedure to validate and package an application. You can package an application and upload the packaged application directly to APIC or distribute the app through Cisco ACI App Center.

Step 1 Log in to your workspace.

Step 2 To install the packager file, enter the command **pip install** *cisco_aci_app_packager-1.0.tar.gz*.

Step 3 To extract the package, enter the command **tar xvfz** *cisco_aci_app_packager-1.0.tar.gz*

Step 4 Do one of the following:

- To upload the package directly to APIC, enter the command **python aci-app-packager.py -f** *path to the folder location of the app to be packaged* to invoke the script.
- To distribute the app through Cisco ACI App Center, enter the command **python aci-app-packager.py -f** *path to the folder location of the app to be packaged* **-p** *path to the folder location of the private key downloaded from the Cisco ACI App Center* to invoke the script.

The packager uses the validator to validate the directory structure of the app and the app metadata. If the validation is successful, the app is packaged. If the app cannot be validated, an error message is displayed. Ensure that the packager and validator files are present in the same directory.

The output of the packaged app is located in the app directory. The package name for the app is created using the meta data from the `app.json` file. This ensures that every package name is unique. The following format is used to create the package name:

```
vendordomain-appid-version.aci
```

Example:

Packing an app for uploading the package to APIC

```
python aci-app-packager.py -f /local/varbahara/danube/mgmt.git/appstore/apps/ContractViewer

Validation of mandatory files and directories successful
Retrieving app meta data successful
Validation of app meta data successful
App successfully packaged /local/varbahara/danube/mgmt.git/appstore/apps/Cisco-ContractViewer-1.0.aci
```

Packaging an app for distributing an app through the Cisco ACI App Center

```
python aci_app_packager.py -f /local/sunverma/ContractViewer -p /local/sunverma/my_private_key.pem
Validation of mandatory files and directories successful
Retrieving app meta data successful
Validation of app meta data successful
App successfully packaged - /local/sunverma/Cisco-ContractViewer-1.0.aci
```

Example for Packaging an Application

Example workflow for packaging an application

```
Install cisco_aci_app_packager
==============================

user@osboxes:~/app$ pip install cisco_aci_app_packager-1.0.tar.gz
Processing ./cisco_aci_app_packager-1.0.tar.gz
Collecting flask (from cisco-aci-app-packager===1.0)
  Downloading Flask-0.11.1-py2.py3-none-any.whl (80kB)
    100% |████████████████████████████████| 81kB 1.9MB/s
Collecting pycrypto (from cisco-aci-app-packager===1.0)
  Downloading pycrypto-2.6.1.tar.gz (446kB)
    100% |████████████████████████████████| 450kB 1.0MB/s
Collecting validators (from cisco-aci-app-packager===1.0)
  Downloading validators-0.11.1.tar.gz
Collecting python-magic (from cisco-aci-app-packager===1.0)
  Downloading python-magic-0.4.12.tar.gz
Collecting click>=2.0 (from flask->cisco-aci-app-packager===1.0)
  Downloading click-6.6-py2.py3-none-any.whl (71kB)
    100% |████████████████████████████████| 71kB 4.3MB/s
Collecting Werkzeug>=0.7 (from flask->cisco-aci-app-packager===1.0)
  Downloading Werkzeug-0.11.11-py2.py3-none-any.whl (306kB)
    100% |████████████████████████████████| 307kB 1.7MB/s
Collecting Jinja2>=2.4 (from flask->cisco-aci-app-packager===1.0)
  Downloading Jinja2-2.8-py2.py3-none-any.whl (263kB)
    100% |████████████████████████████████| 266kB 2.3MB/s
Collecting itsdangerous>=0.21 (from flask->cisco-aci-app-packager===1.0)
  Downloading itsdangerous-0.24.tar.gz (46kB)
    100% |████████████████████████████████| 51kB 4.4MB/s
Collecting six>=1.4.0 (from validators->cisco-aci-app-packager===1.0)
  Downloading six-1.10.0-py2.py3-none-any.whl
Collecting decorator>=3.4.0 (from validators->cisco-aci-app-packager===1.0)
  Downloading decorator-4.0.10-py2.py3-none-any.whl
Collecting MarkupSafe (from Jinja2>=2.4->flask->cisco-aci-app-packager===1.0)
  Downloading MarkupSafe-0.23.tar.gz
Building wheels for collected packages: cisco-aci-app-packager, pycrypto, validators,
python-magic, itsdangerous, MarkupSafe
  Running setup.py bdist_wheel for cisco-aci-app-packager ... done
  Stored in directory:
/home/user/.cache/pip/wheels/56/ed/5d/4be7c57b82475f9c99e96c3dddbe45796456cee0a2d4280b8b
  Running setup.py bdist_wheel for pycrypto ... done
  Stored in directory:
/home/user/.cache/pip/wheels/80/1f/94/f76e9746864f198eb0e304aeec319159fa41b082f61281ffce
  Running setup.py bdist_wheel for validators ... done
  Stored in directory:
/home/user/.cache/pip/wheels/9a/ca/46/13ef93cd6d834fc0c2a144060ea0964fc0558932ae82b7241c
  Running setup.py bdist_wheel for python-magic ... done
  Stored in directory:
/home/user/.cache/pip/wheels/f8/61/e5/dd1029e23a94a40b995eab1a9e06a1cf874c7745a383af5034
  Running setup.py bdist_wheel for itsdangerous ... done
  Stored in directory:
/home/user/.cache/pip/wheels/fc/a8/66/24d655233c757e178d45dea2de22a04c6d92766abfb741129a
  Running setup.py bdist_wheel for MarkupSafe ... done
  Stored in directory:
/home/user/.cache/pip/wheels/a3/fa/dc/0198eed9ad95489b8a4f45d14dd5d2aee3f8984e46862c5748
Successfully built cisco-aci-app-packager pycrypto validators python-magic itsdangerous
MarkupSafe
Installing collected packages: click, Werkzeug, MarkupSafe, Jinja2, itsdangerous, flask,
pycrypto, six, decorator, validators, python-magic, cisco-aci-app-packager
Successfully installed Jinja2 MarkupSafe Werkzeug cisco-aci-app-packager click decorator
flask itsdangerous pycrypto python-magic six validators
You are using pip version 8.1.2, however version 9.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.


Validate installation
=====================
```

```
user@osboxes:~/app$ python
Python 2.7.12+ (default, Sep 17 2016, 12:08:02)
[GCC 6.2.0 20160914] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import Crypto
>>> import magic
>>> import validators
>>> import flask
>>>
```

Extract the package

```
==================

user@osboxes:~/app$ tar xvfz cisco_aci_app_packager-1.0.tar.gz
cisco_aci_app_packager-1.0/
cisco_aci_app_packager-1.0/setup.py
cisco_aci_app_packager-1.0/packager/
cisco_aci_app_packager-1.0/packager/__init__.py
cisco_aci_app_packager-1.0/packager/aci_app_packager.py
cisco_aci_app_packager-1.0/packager/aci_app_validator.py
cisco_aci_app_packager-1.0/packager/Cisco_App_Center_Customer_Agreement.docx
cisco_aci_app_packager-1.0/packager/Cisco_App_Center_License.txt
cisco_aci_app_packager-1.0/packager/Cisco_App_Center_Export_Compliance_Questionnaire.docx
cisco_aci_app_packager-1.0/setup.cfg
cisco_aci_app_packager-1.0/PKG-INFO
cisco_aci_app_packager-1.0/cisco_aci_app_packager.egg-info/
cisco_aci_app_packager-1.0/cisco_aci_app_packager.egg-info/top_level.txt
cisco_aci_app_packager-1.0/cisco_aci_app_packager.egg-info/requires.txt
cisco_aci_app_packager-1.0/cisco_aci_app_packager.egg-info/PKG-INFO
cisco_aci_app_packager-1.0/cisco_aci_app_packager.egg-info/dependency_links.txt
cisco_aci_app_packager-1.0/cisco_aci_app_packager.egg-info/not-zip-safe
cisco_aci_app_packager-1.0/cisco_aci_app_packager.egg-info/SOURCES.txt
cisco_aci_app_packager-1.0/README.rst

Go to packager directory
=======================

user@osboxes:~/app$ cd cisco_aci_app_packager-1.0/packager/
```

Use package command to package your app

```
=======================================
```

Without developer signature:

```
user@osboxes:~/app/cisco_aci_app_packager-1.0/packager$ python aci_app_packager.py -f
/home/user/app/VisuDash
Validation of mandatory files and directories successful
Retrieving app meta data successful
Validation of app meta data successful
App successfully packaged - /home/user/app/Cisco-VisuDash-1.0.aci
```

With developer signature:

```
user@osboxes:~/app/cisco_aci_app_packager-1.0/packager$ python aci_app_packager.py -f
/home/user/app/VisuDash -p /home/user/app/aci_app_qa_private_key.pem
Validation of mandatory files and directories successful
Retrieving app meta data successful
Validation of app meta data successful
App successfully packaged - /home/user/app/Cisco-VisuDash-1.0.aci
```

Cisco ACI App Center

Generating Keys for an Application

Use this procedure to generate developer signature or private keys for the application. The signature is required for packaging the app before publishing the app to the Cisco ACI App Center.

Before You Begin

- You have a developer account to access the Cisco ACI App Center.

Step 1 Log in to Cisco ACI App Center.

Step 2 Choose **My Account** > **Developer Signature**.

Step 3 Click **Request New Key** to generate the keys.
Once you generate the keys you can use the keys to package the app and then publish the app to Cisco ACI App Center.

Publishing an Application

Use this procedure to upload and publish the application to Cisco ACI App Center. After you publish the app to the Cisco ACI App Center, the app is validated, approved, and certified by Cisco ACI App Center. Once the app is certified, users can download and install the app to APIC.

Before You Begin

- You have a developer account to access the Cisco ACI App Center.
- You have signed and packaged the application. See Packaging an Application, on page 36.

Step 1 Log in to Cisco ACI App Center.

Step 2 Choose **Developer Dashboard**.
The Developer Dashboard is displayed

Step 3 Click **Publish a new app**.

Step 4 Click **Upload** to upload the app signed and packaged by the developer to the Cisco ACI App Center.
The uploaded app is then displayed in the Dashboard. The app is then validated and sent for approval. Once the app is approved, it is certified by Cisco ACI App Center and available for downloading.

Downloading Application From Cisco ACI App Center

Use this procedure to download an approved application from the Cisco ACI App Center.

Step 1 Log in to Cisco ACI App Center as an end user.

Step 2 Click **Browse Apps**.
The list of apps available for download are displayed. You can click the list icon to display the available apps in the list view or you can click the grid icon to display the available apps in a grid view.

Step 3 Select an app to view the details.

Step 4 Click **Download**. Review the license agreement and click **Agree and Download**.
The app is downloaded to your local machine.

Cisco APIC

Enabling Signature Validation for an Application

All apps published to Cisco ACI App Center are signed by Cisco. An admin user can choose to enable signature validation for all the apps on the APIC. Once you enable signature validation, only the apps signed by Cisco can be installed on the APIC. By default, signature validation for an app is disabled.

Enable signature validation for an app, using the following REST API POST:

Example:

```
https://<APIC IP>/api/plgnhandler/mo/.xml:
<apPluginPolContr>
    <apPluginPol verifySignature="enable"/>
</apPluginPolContr>
```

Uploading an Application to APIC

Use this procedure to upload your packaged application to APIC. Only an APIC admin user can upload and install an application.

Before You Begin

- You have developed your application.

- You have packaged your application.

Step 1 Log in to the Cisco APIC.

Step 2 On the menu bar, choose **Apps** > **All Apps**.

Step 3 Click the + icon to add an app.

Step 4 Click **Browse** to upload the app.

Step 5 Click **Submit** to upload the app.
After the app is uploaded, the thumbnail of the uploaded app is displayed under the **All Apps** tab.

Installing an Application

Use this procedure to install your application to APIC. Only an APIC admin user can upload and install an application.

Before You Begin

- You have developed the application.
- You have packaged the application.
- You have uploaded the application to APIC.

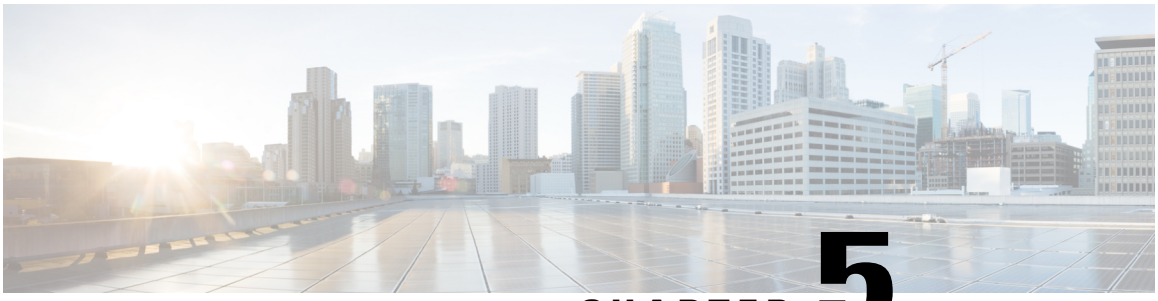
Step 1 Log in to Cisco APIC.

Step 2 On the menu bar, choose **Apps** > **All Apps**.

Step 3 Select the app and verify the User Name, User Role, Permissions for the app. Specify the **Security Domain** for the app. See Understanding Permissions for an Application, on page 6 for more information.

Step 4 Click **Install** to install the app. You can also select **Install** from the **Actions** drop-down list to install an app.
Once the app is installed, it is displayed on the **Installed Apps** tab.

Step 5 To launch an app, select the app from the **Installed Apps** tab. In the `app.json` file, if an Insertion URL is specified, then the app is also inserted in the APIC UI in the location as specified in the insertion URL. You can then launch the app from location where the app is inserted in the APIC UI.



CHAPTER 5

Troubleshooting

This chapter contains the following sections:

Troubleshooting an Application

Troubleshooting the Front-End of the App

- Open the JavaScript console of the app in any browser, to troubleshoot the front-end of an app.
- Monitor the API requests called when using the application.

Collecting Tech Support Logs for a Stateful App

1. Log in to the tech support UI.
2. Create an **On Demand** tech support policy.
3. For an **On Demand** tech support policy, select **For Apps** option.
4. Select the name of the app from the **App** drop-down list.
5. The logs will be located in the `/data2/logs/app-name` directory. The app data will be located in the `/gluster/gv0/app-name` directory.

Troubleshooting the Backend of a Stateful App

You can modify the source files running in the backend on the service directory on the APIC. You must then uninstall and reinstall the app on the APIC for the changes to take effect.

Troubleshooting an Installed Application

To report any issues, regarding an installed application you can send an email to the app developer. Select the app to view the details such as contact information.

Troubleshooting RBAC

After installing an app, ensure that the Managed Objects (MO) for the app user, user role, and security domain are created.

1. Log in to Visore to access the MIT for the app. See the *Cisco APIC REST API Configuration Guide* for information about using Visore.
2. Run a query to verify that the aaaRole, appAppUser MOs are created for the app.
3. Run a query to verify that the MO **apPlugin** is created. Verify that the fields **configInfo** is empty and configSt is populated as none.

Verifying Creation of Managed Objects for an Application

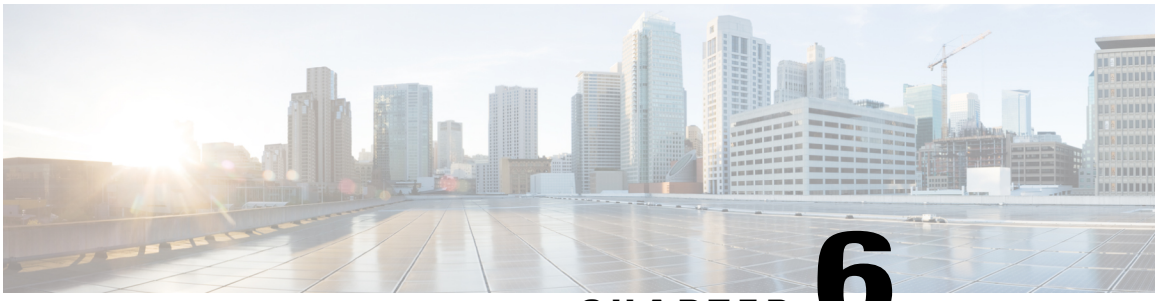
After you upload an app to APIC, ensure that the Managed Objects (MOs) **firmwareFirmware** and **apPlugin** are created.

1. Log into Visore. See the *Cisco APIC REST API Configuration Guide* for information about using Visore.
2. Run a query to verify that the MOs **firmwareFirmware** and **apPlugin** are created.
3. For the **apPlugin** MO, verify that the fields **permissions**, **permissionsLevel**, **pluginState**, **pluginType**, and **securityDomain** are defined.
4. For a stateful app, verify that the **apPluginAppliance** MO is created for all the APICs in a cluster. The leader APIC will have the fields **cntrInstID**, and **cntrInstIP** populated with the container ID and IP address.

Troubleshooting Scenarios

The following table summarizes common troubleshooting scenarios for developing an app for the Cisco ACI App Center.

Problem	Solution
The status of the docker is displayed as Restarting and you are unable to ssh to the docker container. This issue is encountered if there is an issue with the starting script or *start.sh*.	Access the `app.log` located at `/data2/logs/<Vendor>_<AppID>/logs` in the APIC to troubleshoot the issue.



CHAPTER 6

Appendix

This chapter contains the following sections:

Example of Files Used in a Stateless Application

This section contains examples of files used in Contract Viewer application.

- See the *Cisco APIC REST API Configuration Guide* for information about APIC REST APIs.
- To view source code of Contract Viewer application, see Cisco DevNet.

The Contract Viewer app provides information regarding the flow of traffic between endpoint groups on a contract basis. It allows the user to visualize traffic flow in the Consumer, Provider, Filter and Contract mode. A DVR is also provided to enable playback of traffic. The app provides a visual and effective way to troubleshoot and detect unexpected traffic behavior.

Example of `app.json` file for Contract Viewer application

```
{
    "apicversion":"2.2(1a)",
    "appid":"ContractViewer",
    "author":"UVX",
    "category":[
        "Visibility and Monitoring"
    ],
    "contact":{
        "contact-email":"aciappcenter-support@cisco.com",
        "contact-url":"http://www.cisco.com/go/aci"
    },
    "iconfile":"TrafficMap.jpg",
    "insertionURL":"fv:infoTenant:center",
    "name":"Contract Viewer",
    "permissions":[
        "tenant-epg",
        "tenant-network-profile",
```

```
        "tenant-security"
    ],
    "permissionslevel":"read",
    "shortdescr":"Contract Viewer for the stats data between EPGs",
    "vendor":"Cisco",
    "vendordomain":"Cisco",
    "version":"1.0"
}
```

Example of `app.html` file for Contract Viewer application

```
<!DOCTYPE html>
<html>
<meta charset="utf-8">
<body>
<link rel="stylesheet" type="text/css" href="bipartitelayout.css">
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
<script src="scripts/d3/d3.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/async/1.5.2/async.js"></script>
<script type="text/javascript" src="/extjs/ext-all-debug.js"></script>
<script src="scripts/jquery/jquery.2.1.4.js"></script>
<script src="scripts/jquery/jquery.cookie.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.8.3/underscore.js"></script>
<script src="scripts/jquery/jquery.leanModal.min.js"></script>
<script src="scripts/biPartite.js"></script>
<script src="scripts/CreateHttpRequest.js"></script>
<script src="scripts/drawFuncAndUtils.js"></script>
<script src="scripts/zeroTrafficCase.js"></script>
<script src="scripts/helpers.js"></script>
<script src="scripts/FilterQueries.js"></script>
<script src="scripts/handleTenantRequest.js"></script>
<script src="scripts/appDeploySlider.js"></script>
<link rel="stylesheet" href="font-awesome.min.css" />
<link href="https://gitcdn.github.io/bootstrap-toggle/2.2.0/css/bootstrap-toggle.min.css"
rel="stylesheet">
<script
src="https://gitcdn.github.io/bootstrap-toggle/2.2.0/js/bootstrap-toggle.min.js"></script>
<link rel="stylesheet" type="text/css" href="d3.slider.css" media="screen" />
<script>

    $(function(){
        if (httpLoginRequest()) {
                  console.log("SUCCESS!");
        } else {
            console.log("FAIL!");
        }
    });

</script>
<div id="graph-container"></div>
<div class="container">
    <div id="player" class="hidden">
        <button title="Rewind" type="button" id="button_rev" class="btn"
onclick='buttonRevPress()'>
            <i class="fa fa-backward"></i>
        </button>

        <button title="Pause" type="button" id="button_pause" class="btn"
onclick='buttonPausePress()'>
            <i class="fa fa-pause"></i>
        </button>

        <button title="Play" type="button" id="button_play" class="btn"
onclick='buttonPlayPress()'>
            <i class="fa fa-play"></i>
        </button>

        <button title="Switch to Compact View" type="button" id="overallView" class="btn"
onclick='showOverallView()'>
            <i class="glyphicon glyphicon-eye-open"></i>
```

```
        </button>
        <button title="Refresh" type="button" id="refreshView" class="btn"
onclick='refreshView()'>
            <i class="glyphicon glyphicon-refresh"></i>
        </button>
        <input type="checkbox" id="isFilterSelected"  checked data-toggle="toggle"
data-on="Contract View" data-off="Filter View" data-onstyle="success" data-offstyle="info"
 data-width="120">
    </div>
    <div id="filterToggleOnly" class="hidden">
        <input type="checkbox" id="isFilterSelectedZeroTrafficCase"  checked
data-toggle="toggle" data-on="Contract View" data-off="Filter View" data-onstyle="success"
 data-offstyle="info" data-width="120">
    </div>
</div>
<div id="slider-container">
    <div id="slider"></div>
</div>
<div id="traffic-slider"></div>
<div id="traffic-container"></div>
</body>
</html>
```

Example of Files Used in a Stateful Application

This section contains examples of files used in Hello ACI application.

- See the *Cisco APIC REST API Configuration Guide* for information about APIC REST APIs.
- To view source code of Hello ACI application, see Cisco DevNet.

The Hello ACI application is a stateful application that displays a list of tenants.

Example of `app.json` file for Hello ACI application

```
{
    "api":{
        "getTenant.json":"Get tenant information"
    },
    "apicversion":"2.2(1a)",
    "appid":"HelloAciStateful",
    "author":"ABC EFG",
    "category":[
        "Visibility and Monitoring"
    ],
    "contact":{
        "contact-email":"aciappcenter-support@cisco.com",
        "contact-url":"http://www.cisco.com/go/aci"
    },
    "iconfile":"HelloAci.png",
    "name":"HelloAciStateful",
    "permissions":[
        "tenant-qos",
        "tenant-security",
        "tenant-epg",
        "tenant-connectivity-l3"
    ],
    "permissionslevel":"write",
    "shortdescr":"This app demonstrates an example of an stateful app.",
    "vendor":"Cisco",
    "vendordomain":"Cisco",
    "version":"1.0"
}
```

Example of `readme.txt` file for Hello ACI application

```
HELLO ACI
=====================


# Introduction

This application aims at providing a toy example as a stateful app.


# Components

This application shows the different tenants in the fabric. It is composed of:
- a frontend: web UI, see the UIAssets/ directory,
- a backend: docker container, see the Service/ directory.


# Workflow

The workflow for this application is the following:

1. The user opens the application. This triggers a request from the frontend towards the
backend:

        APIC IP/appcenter/Cisco/HelloAciStateful/getTenant.json

2. At the backend side, upon reception of this request, the web server queries the APIC for
 the tenants ('fvTenant').

3. Upon reception of the reply from the APIC, the web server forges a response containing
those tenants.

4. The frontend receives this response, reads it and generates a graph showing the different
 tenants.
```

Example of `app.html` file for Hello ACI application

```
<!DOCTYPE html>
<meta charset="utf-8">

<!-- Style -->
<link rel="stylesheet" type="text/css" href="style.css">

<!-- Import of Javascript files -->
<!-- ext framework is available from the APIC -->
<script type="text/javascript" src="/extjs/ext-all-debug.js"></script>

<script type="text/javascript" src="jquery/jquery.2.1.4.js"></script>
<script type="text/javascript" src="misc.js"></script>
<script type="text/javascript" src="d3/d3.min.js"></script>


<!-- Token management-->

<!--
  Get the tokens at first launch.

  When the app is first launched, the tokens are contained is a cookies, named
  app_VENDORDOMAIN_APPID_token and app_VENDORDOMAIN_APPID_urlToken.
  In this case, we're using Ext to retrieve those tokens and put them into
  window.APIC_DEV_COOKIE and window.APIC_URL_TOKEN.
-->
<script type="text/javascript">
  window.APIC_DEV_COOKIE = Ext.util.Cookies.get("app_Cisco_HelloAciStateful_token");
  window.APIC_URL_TOKEN =  Ext.util.Cookies.get("app_Cisco_HelloAciStateful_urlToken");
</script>

<!--
  Handle the refresh of the tokens.

  This will automatically update window.APIC_DEV_COOKIE and window.APIC_URL_TOKEN
```

```
  when the APIC sends new tokens to the application.
-->
<script type="text/javascript">
  window.addEventListener('message', function (e) {
    if (e.source === window.parent) {
        var tokenObj = Ext.decode(e.data, true);
        if (tokenObj) {
            window.APIC_DEV_COOKIE = tokenObj.token;
            window.APIC_URL_TOKEN = tokenObj.urlToken;
        }
    }
  });
</script>

<!-- Main logic / visualization -->
<script type="text/javascript" src="app.js"></script>


<body>
    <div id="tree-container"></div>
</body>
</html>
```

Example of `app-start.html` file for Hello ACI application

```
<html>
<head>
    <title>Loading</title>
 <meta content="text/html"/>

 <link rel="stylesheet" type="text/css"
href="/insieme/stromboli/resources/css/insieme-ext-theme.css" />
 <script type="text/javascript" src="/extjs/ext-all-debug.js"></script>

 <script type="text/javascript">
  function onBodyLoad() {
   var arr, url, newUrl = "app.html", myMask = new Ext.LoadMask(Ext.getBody(), {msg:"Please
 wait..."});
   myMask.show();
   window.addEventListener('message', function (e) {
    myMask.hide();
    myMask.destroy();
    if (e.source === window.parent) {
     var tokenObj = Ext.decode(e.data, true);
     if (tokenObj) {
      Ext.util.Cookies.set('app_' + tokenObj.appId + '_token', tokenObj.token);
      Ext.util.Cookies.set('app_' + tokenObj.appId + '_urlToken', tokenObj.urlToken);
      url = window.location.href;
      arr = url.split("?");
      if (arr.length >= 2 && !Ext.isEmpty(arr[1])) {
       newUrl += "?" + arr[1];
      }
      window.location.href = newUrl;
     } else {
      Ext.Msg.alert("Error", "Can not load token from backend.");
     }
    }
   });
  }
 </script>

</head>
<body onLoad="onBodyLoad()"></body>
</html>
```

Example of `app.js` file for Hello ACI application

```
function formatAPICResp(response) {
    var config = {
        name: "",
```

```
        children: []
    };
    config["name"] = "APIC Tenant Config";
    for (var i = 0; i < response.imdata.length; i++) {
        config.children.push({
            "name" : response.imdata[i].fvTenant.attributes.dn,
            "children" : [],
        });
    }
    return config;
}

// Send a request to the backend (docker container) to retrieve the tenants.
var queryUrl = document.location.origin + "/appcenter/Cisco/HelloAciStateful/getTenant.json";
d3.json(queryUrl)
.header("DevCookie", window.APIC_DEV_COOKIE)
.header("APIC-challenge", window.APIC_URL_TOKEN)
.get(function(error, flare) {
    // Callback to build the graph after the reply of the backend

    treeData = formatAPICResp(flare);
```

Example of `start.sh` file for Hello ACI application

```
#!/bin/sh
/usr/sbin/sshd

# Run the server
python /home/app/src/Service/plugin_server.py
```

Example of `server.py` file for Hello ACI application

```
from flask import Flask
from cobra.mit.access import MoDirectory
from cobra.mit.session import CertSession
from cobra.mit.session import LoginSession
from cobra.model.pol import Uni as PolUni
from cobra.model.aaa import UserEp as AaaUserEp
from cobra.model.aaa import AppUser as AaaAppUser
from cobra.model.aaa import UserCert as AaaUserCert
from cobra.internal.codec.jsoncodec import toJSONStr, fromJSONStr
from cobra.internal.codec.xmlcodec import _toXMLStr, fromXMLStr

import json
import logging

app = Flask(__name__)

def createCertSession():
    ''' Creates a session with the APIC.
    Returns a CertSession (Cobra SDK) that can be used to query the APIC.
    '''

    certUser = 'Cisco_HelloAciStateful' # Format: <Vendordomain>_<AppId>
    pKeyFile = '/home/app/credentials/plugin.key' # Fixed for every app

    polUni = PolUni('')
    aaaUserEp = AaaUserEp(polUni)
    aaaAppUser = AaaAppUser(aaaUserEp, certUser)

    aaaUserCert = AaaUserCert(aaaAppUser, certUser)

    with open(pKeyFile, "r") as file:
        pKey = file.read()

    apicUrl = 'https://172.17.0.1/' # Fixed, APIC's gateway for the app

    session = CertSession(apicUrl, aaaUserCert.dn, pKey, secure=False)
    return session
```

```
def respFormatJsonMos(mos, totalCount):
    ''' Format a JSON reply from MOs.
    Inputs:
        - mos: array of MOs
        - totalCount: number of MOs

    Output:
        - JSON dictionary, following this format
        {
           "imdata":[{<MO>}, {<MO>},...],
           "totalCount": ...
        }

    Example:
        {
           "imdata":[
              {
                 "fvTenant":{
                    "attributes":{
                       "dn":"uni/tn-common",
                       ...
                    }
                 }
              },
              {
                 "fvTenant":{
                    "attributes":{
                       "dn":"uni/tn-infra",
                       ...
                    }
                 }
              }
           ],
           "totalCount":"3"
        }
    '''
    jsonStr = '{"totalCount": "%s", "imdata": [' % totalCount
    first = True
    for mo in mos:
        if not first:
            jsonStr += ','
        else:
            first = False
        jsonStr += toJSONStr(mo, includeAllProps=True)
    jsonStr += ']}'
    jsonDict = json.loads(jsonStr)

    return json.dumps(jsonDict)

@app.route('/')
def hello_world():
    ''' Test the connectivity.
    '''
    logging.info('Received API Request from Client - /')
    return 'Cisco HelloACI PlugIn Version 1.0.'

@app.route('/getTenant.json')
def get_tenant():
    ''' Queries the APIC for tenants and replies with those tenants,
    in a JSON format.
    '''
    logging.info('Received API request from client, api: /getTenant.json')

    # Create session
    loginSession = createCertSession()

    # Create object to go through the MIT
    moDir = MoDirectory(loginSession)

    moDir.login()
    # Query for the tenants
    tenantMo = moDir.lookupByClass('fvTenant');
```

```
    moDir.logout()

    logging.info('Sending response')
    return respFormatJsonMos(tenantMo, tenantMo.totalCount)

if __name__ == '__main__':
    # Setup logging
    fStr='%(asctime)s %(levelname)5s %(message)s'
    logging.basicConfig(filename='/home/app/log/helloaci.log', format=fStr,
level=logging.DEBUG)

    # Run app flask server
    app.run(host='0.0.0.0', port=80)
```

Example of `app.js` file for Hello ACI application

```
function getUrlVars() {
    var vars = {};
    var parts = window.location.href.replace(/[?&]+([^=&]+)=([^&]*)/gi,
    function(m,key,value) {
      vars[key] = value;
    });
    return vars;
}

window.APIC_DEV_COOKIE = Ext.util.Cookies.get("app_Cisco_HelloAciStateful_token");
window.APIC_URL_TOKEN =  Ext.util.Cookies.get("app_Cisco_HelloAciStateful_urlToken");
```

Integrating the App's UI in the APIC UI

Use this procedure to integrate your app's UI into the APIC's UI using the **insertionURL** field in the `app.json`.

Step 1 Log in to APIC.

Step 2 Choose **admin** > **Show Debug Info** > **.**

Step 3 Navigate to the page where the app will be inserted in the APIC. The following information will be listed in the bottom of the page.

"`Current Screen:x.y.z [INSERTION_URL ] | ...`"

For example, if the app is to be integrated in the tenant's UI, the INSERTION_URL is "`[fv:infoTenant:center:a]`". Where "`:a`" part corresponds to the **Dashboard** tab in the tenant's UI.

To integrate the app's UI as one of the tabs of the tenants, you must add the line "`insertionURL":"fv:infoTenant:center`" in the `app.json` file.

Note

- Use the following function to retrieve the URL variables.

```
function getUrlVars() {
      var vars = {};
      var parts = window.location.href.replace(/[?&]+([^=&]+)=([^&]*)/gi,
          function(m,key,value) {
              vars[key] = value;
          });
      return vars;
```

- The function returns a dictionary where the keys corresponds to the names of the variables and the value correspond to the value of the variable.

```
Object
            dn: "uni/tn-common"
            _proto_:Object
```

Permissions

The following table provides information on some of the Cisco ACI permissions required for developing an ACI app.

Permission	Description
admin	Complete access to everything (combine ALL roles)
aaa	Used for configuring authentication, authorization, accounting, and import or export policies.
vmm-connectivity	Used to read all the objects in APIC's VMM inventory required for VM connectivity.
vmm-security	Used for contract related configurations for a tenant.
vmm-policy	Used for managing policies for VM networking.
vmm-ep	Used to read VM and Hypervisor endpoints in the APIC's VMM inventory.
vmm-protocol-ops	Not used by VMM policies.
tenant-qos	Only used as Write access for firmware policies.
tenant-security	Used for contract related configurations for a tenant.

Permission	Description
tenant-network-profile	Used for managing tenant configurations, such as deleting and creating network profiles, and deleting and creating endpoint groups.
tenant-epg	Used for managing tenant configurations such as deleting or creating endpoint groups, VRFs, and bridge domains.
tenant-connectivity-l1	Used for Layer 1 connectivity changes, including bridge domains and subnets.
tenant-connectivity-l2	Used for Layer 2 connectivity changes, including bridge domains and subnets.
tenant-connectivity-l3	Used for Layer 3 connectivity changes, including VRFs.
tenant-connectivity-mgmt	Used for tenant in-band and out-of-band management connectivity configurations and for debugging or monitoring policies such as atomic counters and health score.
tenant-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
tenant-protocol-l1	Used for managing configurations for Layer 1 protocols under a tenant.
tenant-protocol-l2	Used for managing configurations for Layer 2 protocols under a tenant.
tenant-protocol-l3	Used for managing configurations for Layer 3 protocols under a tenant.
tenant-protocol-mgmt	Only used as write access for firmware policies.
tenant-protocol-util	Used for debugging, monitoring, observer policies such as traceroute, ping, oam, and eptrk.
tenant-protocol-ops	Used for tenant traceroute policies.
tenant-ex-connectivity-l1	Used for write access firmware policies.
tenant-ex-connectivity-l2	Used for managing tenant L2Out configurations.
tenant-ex-connectivity-l3	Used for managing tenant L3Out configurations.

Permission	Description
tenant-ex-connectivity-mgmt	Used as write access for firmware policies.
tenant-ex-connectivity-util	Used for debugging, monitoring, observer policies such as traceroute, ping, oam, and eptrk.
tenant-ext-protocol-l1	Used for managing tenant external Layer 1 protocols. Generally only used for write access for firmware policies.
tenant-ext-protocol-l2	Used for managing tenant external Layer 2 protocols. Generally only used for write access for firmware policies.
tenant-ext-protocol-l3	Used for managing tenant external Layer 3 protocols such as BGP, OSPF, PIM, and IGMP.
tenant-ext-protocol-mgmt	Used as write access for firmware policies.
tenant-ext-protocol-util	Used for debugging, monitoring, observer policies such as traceroute, ping, oam, and eptrk.
fabric-connectivity-l1	Used for Layer 1 configuration under the fabric. Example: selectors and port Layer 1 policy and vPC protection.
fabric-connectivity-l2	Used in firmware and deployment policies for raising warnings for estimating policy deployment impact.
fabric-connectivity-l3	Used for Layer 3 configuration under the fabric. Example: Fabric IPv4, IPv6, and MAC protection groups.
fabric-connectivity-mgmt	Used for atomic counter and diagnostic policies on leaf switches and spine switches.
fabric-connectivity-util	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
fabric-protocol-l1	Used for Layer 1 protocol configurations under the fabric.
fabric-protocol-l2	Used for Layer 2 protocol configurations under the fabric.
fabric-protocol-l3	Used for Layer 3 protocol configurations under the fabric.

Permission	Description
fabric-protocol-mgmt	Used for fabric-wide policies for NTP, SNMP, DNS, and image management.
fabric-protocol-util	Used for firmware management traceroute and endpoint tracking policies.
fabric-protocol-ops	Used for ERSPAN and health score policies.
fabric-equipment	Used for atomic counter, diagnostic, and image management policies on leaf switches and spine switches.
access-connectivity-l1	Used for Layer 1 configuration under infra. Example: selectors and port Layer 1 policy configurations.
access-connectivity-l2	Used for Layer 2 configuration under infra. Example: Encap configurations on selectors, and attachable entity.
access-connectivity-l3	Used for Layer 3 configuration under infra and static route configurations under a tenant's L3Out.
access-connectivity-mgmt	Used for management infra policies.
access-connectivity-util	Used for tenant ERSPAN policies.
access-protocol-l1	Used for Layer 1 protocol configurations under infra.
access-protocol-l2	Used for Layer 2 protocol configurations under infra.
access-protocol-l3	Used for Layer 3 protocol configurations under infra.
access-protocol-mgmt	Used for fabric-wide policies for NTP, SNMP, DNS, and image management.
access-protocol-util	Used for tenant ERSPAN policies.
access-protocol-ops	Used for operations-related access policies such as cluster policy and firmware policies.
access-equipment	Used for access port configuration.
access-qos	Used for changing CoPP and QoS-related policies.
nw-svc-params	Used for managing Layer 4 to Layer 7 service policies.

Permission	Description
ops	Used for operational policies including monitoring and troubleshooting policies such as atomic counter, SPAN, TSW, tech support, traceroute, analytics, and core policies.
nw-svc-devshare	Used for managing shared Layer 4 to Layer 7 service devices.
nw-svc-policy	Used for managing Layer 4 to Layer 7 network service orchestration.
nw-svc-device	Used for managing Layer 4 to Layer 7 service devices.