

---

# **StateChecker Documentation**

**Hrishikesh Deshpande, Andy Gossett, Axel Bodart**

**Oct 03, 2018**



---

## Contents:

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Install</b>	<b>3</b>
2.1	App Mode . . . . .	3
2.2	Standalone Mode . . . . .	3
2.3	Building ACI Application . . . . .	5
<b>3</b>	<b>Usage</b>	<b>7</b>
3.1	Configuring Users . . . . .	7
3.2	Connecting to the Fabric . . . . .	8
3.3	Creating Snapshots . . . . .	8
3.4	Creating Comparisons . . . . .	9
3.5	Managing Definitions . . . . .	10
<b>4</b>	<b>API</b>	<b>11</b>
4.1	Authentication . . . . .	12
4.2	APIC API access to StateChecker App . . . . .	12
<b>5</b>	<b>Development</b>	<b>15</b>
<b>6</b>	<b>Indices and tables</b>	<b>17</b>



# CHAPTER 1

---

## Introduction

---

State Checker (also known as StateChangeChecker) is a Cisco ACI application that allows operators to snapshot a collection of managed objects (MO) in the fabric and perform snapshot comparisons. This allows operators to answer questions such as:

- What changed in my fabric ?
- Are my critical objects the same after maintenance ?
- Did any route change on any node ?
- Are all the local endpoint learns still present ?

Predefined templates are available for the most common checks such as verifying basic L1/L2 state along with routing and VMM hypervisor connectivity and inventory. Operators can also create their own templates controlling what objects are collected and how they are compared.

This application can be installed directly on the Cisco APIC or deployed in standalone mode.



StateChecker can be installed directly on the APIC as an ACI app or deployed in standalone mode.

Currently, the APIC imposes a **2G** memory limit and a **10G** disk quota on stateful applications. If an operator needs to create and maintain a large number of snapshots or to manage multiple fabrics within a single instance of the application, it may be beneficial to run the application in standalone mode.

## 2.1 App Mode

`app mode` is when the application is deployed and run on the APIC. The most recent public release can be downloaded from [ACI AppCenter](#). After downloading the app, follow the directions for uploading and installing the app on the APIC:

- [2.x Install Video Example](#)
- [2.x Install Instructions](#)
- [3.x Install Instructions](#)

Users can also build the app from source prior to installing on the APIC. See the [Building ACI Application](#) to build the app as an ACI application from source.

## 2.2 Standalone Mode

`standalone mode` allows the application to run on a dedicated host, VM, or container controlled by the operator and make remote connections to the APIC to collect the required data. There are a few options for deploying the app in `standalone mode` below. Once deployed, the default credentials for logging into the application are:

- `username: admin`
- `password: cisco`

### 2.2.1 Pre-build OVA

**Warning:** Details to come

### 2.2.2 Manual Install

The app has been developed to be deployed as an all-in-one container. This relaxes the requirements on the host machine as everything will be dynamically installed when the container is built. The only requirements on the host/VM is the following:

- git
- docker
- linux or macOS

---

**Note:** Deployment scripts are written in bash and have not been tested on Windows. Want to get the build script working in windows? Feel free to submit a pull request!

---

To begin, use git to clone the source repository:

```
$ git clone https://github.com/datacenter/statechecker.git
```

Next execute the `build_app.sh` script with the `-s` flag to deploy the application in standalone mode. By default the build script will deploy the application running on http port (p) 5000 and https port (P) 5001. You can customize these ports using the `-p` and `-P` flags, respectively.

```
$ ./statechecker/build/build_app.sh -h

Help documentation for ./statechecker/build/build_app.sh
-i [image] docker image to bundled into app (.tgz format)
-h display this help message
-k [file] private key uses for signing app
-P [https] https port when running in standalone mode
-p [http] http port when running in standalone mode
-r relax build checks (ensure tools are present but skip version check)
-s build and deploy container for standalone mode
-v [file] path to intro video (.mp4 format)
-x send local environment proxy settings to container during build

$ ./statechecker/build/build_app.sh -s
2018-09-07T19:23:16 check build tools: backend
2018-09-07T19:23:16 checking following dependencies: docker
2018-09-07T19:23:16 all build tool dependencies met
2018-09-07T19:23:16 deploying standalone container Cisco/StateChangeChecker:1.0
2018-09-07T19:23:16 building container
...
Successfully tagged aci/statechangechecker:1.0
2018-09-07T19:23:16 starting container
```

You can validate that the container is running on the configured ports via:



```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	NAMES
c81a139aed67	aci/statechangechecker:1.0	"/bin/sh -c \$BACKEND..."	17 minutes ago	statechangechecker_1.0

Up 17 minutes 0.0.0.0:5000->80/tcp, 0.0.0.0:5001->443/tcp

Once deployed you can access the application on the host and port number you configured. If running on a local machine with default options, you would access the site at <http://localhost:5000>.

**Note:** the build script will mount the source code as a read only directory within the container. Any development should be done on the host, not the container. Similarly, if the source code is removed from the host it will cause the application running in the container to fail.

## 2.3 Building ACI Application

If you are unable to download the app from the appstore or need to build from source to resolve a bug or enhancement, you can build the application from source. On the development environment, you'll need to have the following installed:

- git
- docker
- zip
- python2.7 + pip
- Node.js v9.8.0
- Npm v5.8.0

First, clone the source code repo. From within the source directory, use pip to install the packager dependencies. Finally, execute the `build_app.sh` script.

**Note:** If using a node or npm version higher than that listed in the requirements, used the `-r` flag during the build process.

```
$ git clone https://github.com/datacenter/statechecker.git
$ cd ./statechecker
$ pip install build/app_package/cisco_aci_app_tools-1.1_min.tar.gz
$ ./build/build_app.sh
```



# CHAPTER 3

---

## Usage

---

StateChecker allows the operator to collect snapshots of the fabric and perform comparisons to understand how the fabric has changed. To get started, perform the following steps:

- *Configuring Users*
- *Connecting to the Fabric*
- *Creating Snapshots*
- *Creating Comparisons*
- *Managing Definitions*

---

**Note:** There are a few differences when running StateChecker in `app mode` vs. `standalone`. In `app mode`, all authentication and authorization is handled by the APIC and not enabled on the app. Second, only the fabric in which the app is installed can be monitored. The app uses certificates installed by the APIC to access the MOs and therefore no fabric setup or configurations are required. All **fabric** and **user** options are hidden when running in `app mode`.

---

## 3.1 Configuring Users


---

**Note:** Configuring users is only available in `standalone mode`

---

At install a single user is created with username `admin` and default password `cisco`. Operators can configure multiple users for accessing StateChecker app. A user can have one of three roles:

- `FULL_ADMIN` role is capable of performing on all read and write operations within the app
- `USER` role is a read only role that can view snapshots and comparisons but cannot create or edit them
- `BLACKLIST` role is not allowed to access the application

Click the users icon  at the top right of the application to manage users. Users can be added, deleted, and updated as needed. Also, the user tab allows passwords to be changed. It is highly recommended to change the default password.



## 3.2 Connecting to the Fabric

---


**Note:** Configuring the fabric is only available in `standalone` mode


---

A fabric refers to a single APIC cluster and corresponding switches. StateChecker app can be configured to perform snapshots across multiple fabrics. Note, snapshot comparison must be between two snapshots collected from the same fabric.

To get started, click the fabric icon  Fabric Manage fabric entries. Click the add icon  to add a new fabric. Configure the following for connecting to the fabric:



- **Name** unique identifier for this fabric
- **Hostname** DNS hostname or IP address of a single APIC in the cluster. Ideally this would be APIC-1 but can be any APIC in the cluster. The other APIC controllers within the cluster are discovered dynamically and their out-of-band IPv4 addresses are cached and used in the absence of the configure hostname
- **Username** for API access to the fabric. Note, the user must have `admin` read access to the MOs in the fabric for proper operation of StateChecker app.
- **Password** for configured username



The app will test the configured credentials at the time the fabric is created or edited. Users can additionally test access for a configured fabric by clicking the verify icon .

Any changes to the configured fabrics should automatically be updated within the GUI. Users can also click the refresh icon  to manually refresh the state of all configured fabrics.

## 3.3 Creating Snapshots

A snapshot is the collection of managed objects (MOs) from the fabric. The specific MOs collected are dependent on the definition chosen. By default, the `Full` definition is selected which collects all MOs within the app. See [Managing Definitions](#) for more details on definitions.

To collect a snapshot, click the snapshot icon  Snapshot Create or delete snapshots. Next, click the add icon  and provide a **description**, **definition**, and a **fabric**. Note, when running in `app` mode the fabric option is not available.

For backup purposes, a snapshot can be downloaded. Once a snapshot has completed, click the download icon  which will create a `.tgz` file with all snapshot data. This can later be imported back into the app via the upload icon . There are a few restrictions to be aware of when using the upload operation:

- The fabric name for the uploaded snapshot must exist within the app
- The definition name for the uploaded snapshot must exist within the app
- When running in `app_mode`, there is a **20MB** limitation on file uploads. The snapshot must be less than **20MB** to import.

## 3.4 Creating Comparisons

Operators can compare snapshots to determine what has changed within the fabric. To perform a comparison, click

the comparison icon




and then add comparison

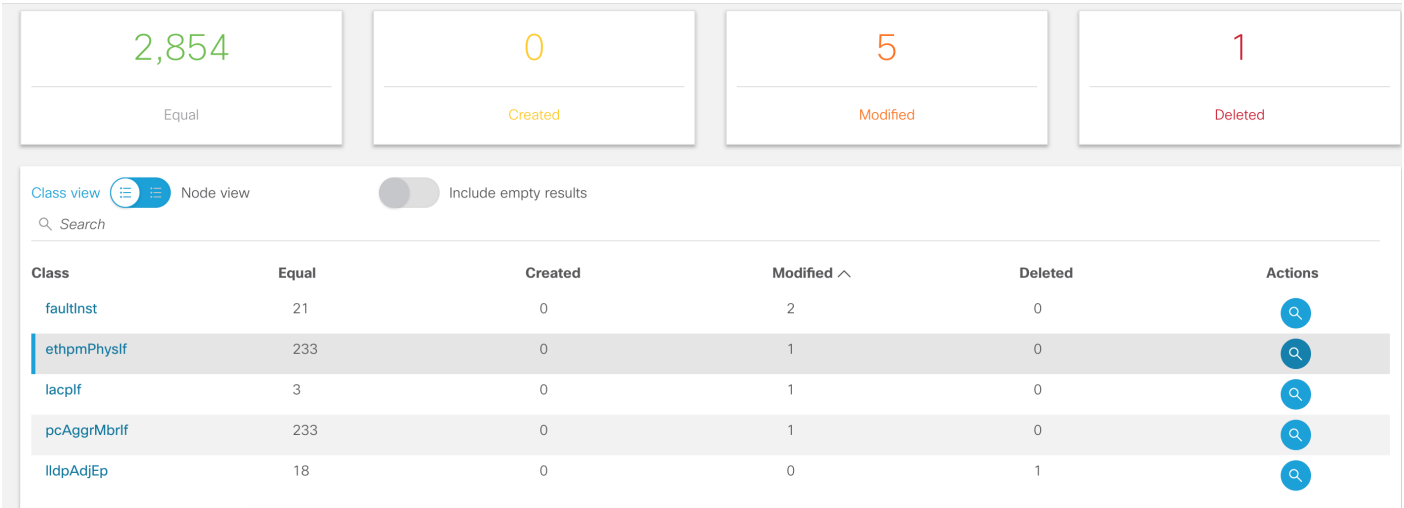


. There are several options for a


comparison:

- **Snapshot 1** The first snapshot to be compared
- **Snapshot 2** The second snapshot to be compared. Note, if snapshot2 is before snapshot1 based on the collection time within the snapshot, the app will swap snapshot1 and snapshot2 pointers
- **Definition** The definition to use for the comparison. The definition is a list of one or more MOs. Here, the definition is used as a filter to control which objects are compared. See [Managing Definitions](#) for more details.
- **Nodes** Operators can filter the comparison to a selection of one or more nodes. This is useful when a maintenance window has been performed and there are several known changes that have occurred but operators expect specific state to be the same on a subset of leafs. For example, the addition of new service leafs should not have affected `Access` MOs on the border leafs. By default this option is empty which implies that the comparison should be performed on all nodes within the snapshots.
- **Compare Options** There are a few knobs to control how the comparison is performed.
  - **Dynamic** Some MOs along with MOs attributes are marked as **dynamic**. Dynamic objects and attributes are those which are expected to change between snapshots. Examples includes hardware indexes and file descriptor. The dynamic option is **disabled** by default.
  - **Remap** ACI abstracts logical resources from concrete values. For example, a user creates a BD and EPG and deploys it to a leaf. This logical model translates to a concrete vlan with a vlan identifier that is arbitrarily allocated. Removing and readding the EPG or reloading the switch may result in a different vlan id for a particular EPG. To perform comparison between snapshots, the StateChecker application will map the vlan identifier to a consistent logical value. Other objects that are remapped include include port-channels, sub-interfaces, tunnels, and loopback interfaces. Also, all MOs that reference this objects will also be remapped. For example, a route next hop may contain a vlan id that needs to be remapped before snapshot comparison. The remap option is **enabled** by default
  - **Statistic** Some object and attributes are statistics (counters) that are expected to increment at a regular interval. Operators can choose to include or exclude statistics during comparison via this option. The dynamic option is **disabled** by default.
  - **Timestamp** Timestamp MOs and attributes generally reference the time in which the value was created or modified. Operators can choose to include or exclude timestamps during comparison via this option. The dynamic option is **disabled** by default.
  - **Serialize** (development only) Snapshot comparisons can involve 10's to 100's of thousands of objects. For efficiency this is performed in parallel by multiple processes. For debugging, users can force comparison to operate serially. The dynamic option is **disabled** by default.


Once a comparison is completed, click the zoom icon  to see the details. The comparison page includes the overall totals of equal, created, modified, and deleted objects. Operators can further drill down into specific nodes and specific classes to understand what changes have occurred.



### 3.5 Managing Definitions

A definition is a collection of one or more managed objects (MOs). Operators can add and modify definitions by clicking the definition icon  at the top right of the page. There are four predefined definition templates that cannot be modified:

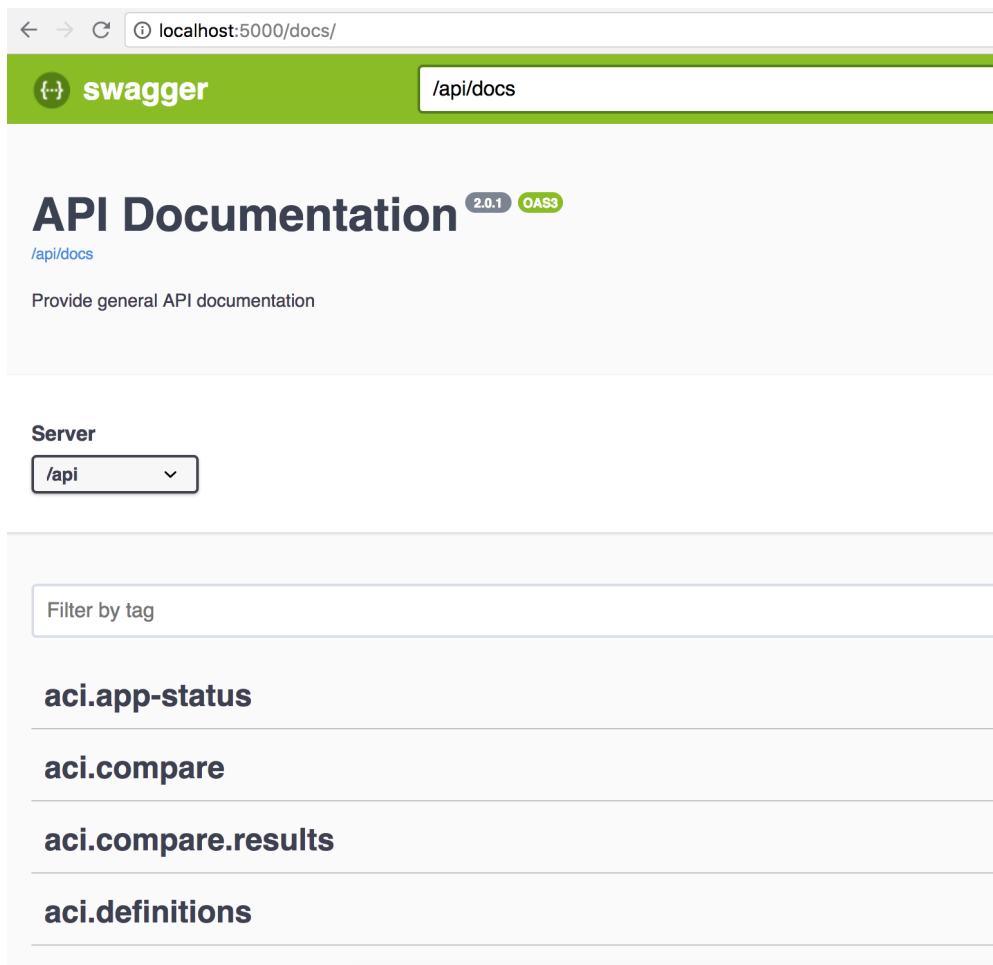
- **Full** default definition that includes all MOs defined within the app
- **Access** This definition collects access information including basic interface status, switch inventory information, along with L1/L2 state.
- **Routing** This definition collects basic information about L2/L3 local endpoint and route reachability state. This includes protocol status, adjacency information, and routing information base (RIB) state.
- **VMM** This definition collects virtual machine manager (VMM) information focusing on hypervisor connectivity, inventory, and topology state.

Operators can click the add icon  to create a new definition. A definition includes a name, description, and list of MOs (classes). This helpful for operators who have a specific set of objects that need to be compared outside of the predefined templates.

## CHAPTER 4

### API

Statechecker implements a restful `api` with auto-created swagger documentation for each endpoint. The swagger documentation is available at `/docs`.



The `api` implements basic CRUD (create, read, update, delete) operations for objects that support them. In general, create requests are issued via an HTTP POST, read requests via an HTTP GET, update requests via HTTP PATCH, and delete via HTTP DELETE. The url endpoint, parameters, and responses are all provided in the swagger docs.

---

**Note:** the swagger documentation is only viewable in standalone mode.

---

## 4.1 Authentication

Authentication can be enabled or disabled by the `LOGIN_ENABLED` option. This can be set via an environmental variable in the container, updates to `config.py` or `instance/config.py` values. It is enabled by default in standalone mode and disabled in app mode as APIC handles all authentication and authorization.

Session management is handled by login and logout APIs under the `users` class. For login, a cookie named 'session' is returned and should be provided in all subsequent API calls.

```
# login
curl -X POST "http://localhost:5000/api/users/login" -H "Content-Type: application/
↪json" \
  -d '{"password":"cisco","username":"admin"}'

# logout
curl -X POST "http://localhost:5000/api/users/logout"
```

## 4.2 APIC API access to StateChecker App

When running in app mode and deployed on the Cisco ACI APIC, the app is accessible via the **`http://<apic-ip>/appcenter/Cisco/StateChangeChecker/proxy.json`**. Note, the APIC restricts apps to GET and POST methods only. Additionally, only static url endpoints are allowed. Since this app uses GET, POST, PATCH, and DELETE as well as dynamic url endpoints, a `proxy` option was implemented to allow it to run on the APIC. For the proxy, all requests are sent as POST to `proxy.json` endpoint with the following required in the data field:

- **method** The original method intended for the app (GET, POST, PATCH, or DELETE)
- **url** The original url that is intended to be proxied. As an example, to get a list of snapshots, the url would be `/api/aci/snapshots` as seen in the swagger docs.
- **data** The original data sent via POST, PATCH requests.

The user must also have admin read access on the APIC and use the APIC `aaaLogin` api to acquire a `token` and `urlToken` for accessing the app API. The `token` must be included in all requests as a cookie named **APIC-Cookie** or an HTTP header named **DevCookie**. The `urlToken` must be included in all requests as a url parameter named **url-token** or HTTP header named **APIC-Challenge**

For example:

```
# login
curl -kX POST "https://<apic-ip>/api/aaaLogin.json?gui-token-request=yes" \
  -H "Content-Type: application/json" \
  -d '{"aaaUser":{"attributes":{"name":"<username>","pwd":"<password>}}}' \
  | python -m json.tool

{"imdata": [
```

(continues on next page)



(continued from previous page)

```

{"aaaLogin": {
  "attributes": {
    "buildTime": "Thu Apr 19 17:17:39 PDT 2018",
    "changePassword": "no",
    "creationTime": "1536530859",
    "firstLoginTime": "1536530859",
    "firstName": "",
    "guiIdleTimeoutSeconds": "65525",
    "lastName": "",
    "maximumLifetimeSeconds": "86400",
    "node": "topology/pod-1/node-1",
    "refreshTimeoutSeconds": "9600",
    "remoteUser": "false",
    "restTimeoutSeconds": "90",
    "sessionId": "weTl7S7hSx6CQ4KkkLZ21g==",
    "siteFingerprint": "B4NoUf+/99ZZsL5K",
    "token": "OxAAAAAAAAAAAAAAAAAAHQGVL9YKcIy2...", <---- token
    "unixUserId": "15374",
    "urlToken": "2f70d689cb5fd9aa3ff63046...", <----- urlToken
    "userName": "admin",
    "version": "3.1(2o)"
  },
},
<snip>

# access app api
curl --header "DevCookie: AAAAAAAAAAAAAAAAAAAHQGVL9YKcIy2..." \
  --header "APIC-Challenge: 2f70d689cb5fd9aa3ff63046..." \
  -kX POST "https://<apic-ip>/appcenter/Cisco/StateChangeChecker/proxy.json" \
  -d '{"url": "/api/aci/snapshots", "method": "GET"}'
```

More information on using the Cisco API is available on [Cisco APIC REST API Configuration Guide](#)



## CHAPTER 5

---

Development

---

TODO



## CHAPTER 6

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`