



DIMS Dashboard Documentation

Release 1.0.10

Linda Parsons

Aug 16, 2017

Contents

1	Using the Dashboard	3
1.1	Mitigation Scenario	3
1.2	System health	7
1.3	Live log streaming	7
1.4	Chat	12
1.5	User display and trust groups	12
2	Tickets	17
2.1	General ticket structure	17
2.2	Topics	17
3	Ticket API	19
3.1	HTTP Verbs	20
3.2	Responses	20
3.3	Retrieve a list of tickets	21
3.3.1	No parameters	21
3.3.2	With query parameters	22
3.4	Creating an activity	23
4	Contact	29
5	License	31

Contents:

CHAPTER 1

Using the Dashboard

This section will introduce basic usage of the DIMS Dashboard.

Currently this section contains a demo runthrough with commentary. The following sub-sections will go through the:

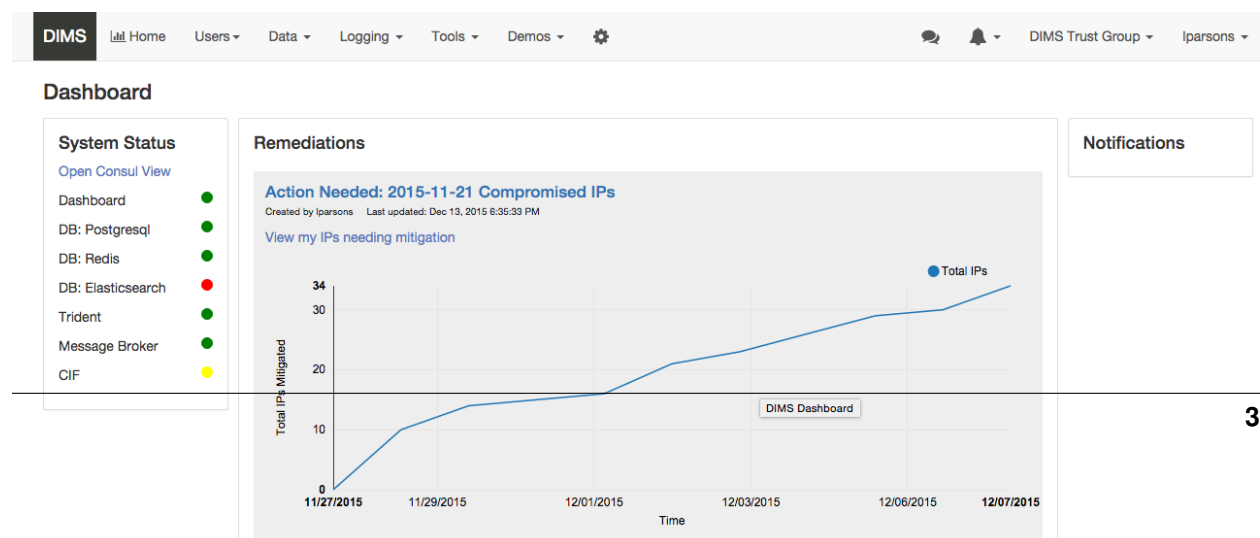
1. Mitigation Scenario
2. System health
3. Live log streaming
4. Chat
5. User display and trust group info - show users by trust group

The demo application is not currently using https, so you won't need to worry about certificates when logging in. Make sure you are logged out of the dashboard (if you are already logged in) and reload the login page if you are already on that page (to make sure your client has the latest version).

Mitigation Scenario

Go to `demo.prisem.washington.edu` and log in using your ops-trust username and password.

The main dashboard will display.

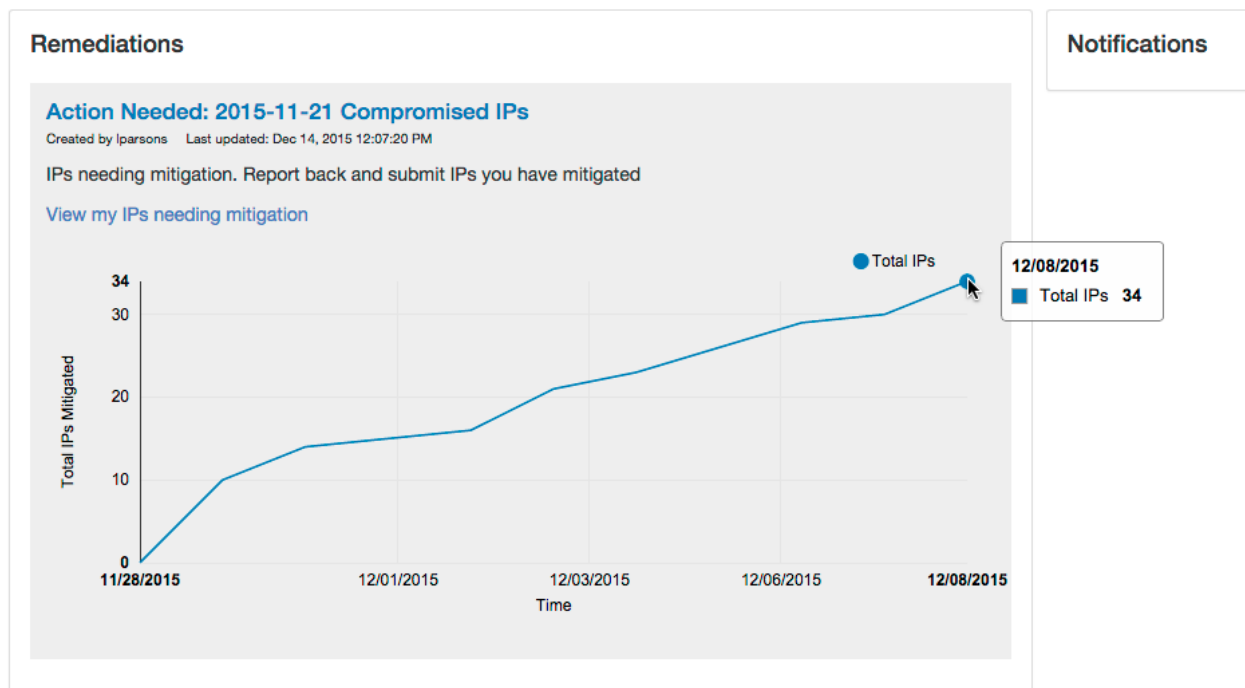


Remediations are mitigation activities

in
the
sys-
tem
where
the
logged
in
user
(lpar-
sons
in
the
screen-

shots) has IPs that are compromised and need to be remediated. Currently the system contains one of these “Mitigation scenario” activities which was bootstrapped programmatically.

Hover over the graph to display data points. Note the number of mitigated IPs at the most recent data point (the dates may differ than that in the screenshot):



Click **View my IPs needing mitigation** link to display a modal window where you can submit IPs that have been mitigated. Right now the UI for this consists of the modal displaying all remaining IPs you need to address.

This mitigation activity has IPs that need to be remediated for the users dittrich, lparsons, mbogges, and swarner. So your IPs will look different than those in this figure.

Check off some IPs indicating that they have been mitigated and click *Submit*.

The modal window will close and the graph will be updated. Hover over the last data point to verify. For this user, the total IPs mitigated is now 39.

Note: Currently, to start a new mitigation activity, a user will do so via the Dashboard (UI not available yet), using a

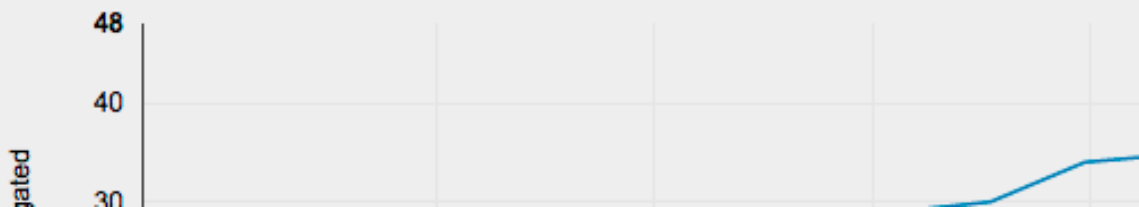
Remediations

Action Needed: 2015-11-21 Compromised IPs

Created by lparsons Last updated: Dec 14, 2015 12:07:20 PM

IPs needing mitigation. Report back and submit IPs you have mitigated

[View my IPs needing mitigation](#)



Users ▾ Data ▾ Log Monitor Tools ▾ Demos ▾ ⚙

IPs needing remediation

You are responsible for taking mitigating action against the following IP addresses. Check off the IPs you are submitting as remediated and click **Submit**

<input type="checkbox"/> 5.144.130.38	<input type="checkbox"/> 81.162.73.224	<input type="checkbox"/> 193.107.16.206
<input type="checkbox"/> 37.0.124.155	<input type="checkbox"/> 91.106.207.103	<input type="checkbox"/> 193.107.17.72
<input type="checkbox"/> 37.140.192.17	<input type="checkbox"/> 91.189.136.16	<input type="checkbox"/> 195.211.154.177
<input type="checkbox"/> 37.152.88.39	<input type="checkbox"/> 91.200.32.231	<input type="checkbox"/> 195.211.154.180
<input type="checkbox"/> 37.247.101.58	<input type="checkbox"/> 91.213.96.32	<input type="checkbox"/> 195.211.155.227
<input type="checkbox"/> 46.20.12.227	<input type="checkbox"/> 93.170.131.69	<input type="checkbox"/> 202.190.179.48
<input type="checkbox"/> 46.29.255.100	<input type="checkbox"/> 162.251.113.166	<input type="checkbox"/> 212.79.127.227
<input type="checkbox"/> 65.207.23.201	<input type="checkbox"/> 178.18.25.125	<input type="checkbox"/> 212.235.117.238
<input type="checkbox"/> 72.69.215.66	<input type="checkbox"/> 186.233.144.136	<input type="checkbox"/> 213.142.143.194
<input type="checkbox"/> 77.221.144.40	<input type="checkbox"/> 189.203.240.71	

ers Data Log Monitor Tools Demos

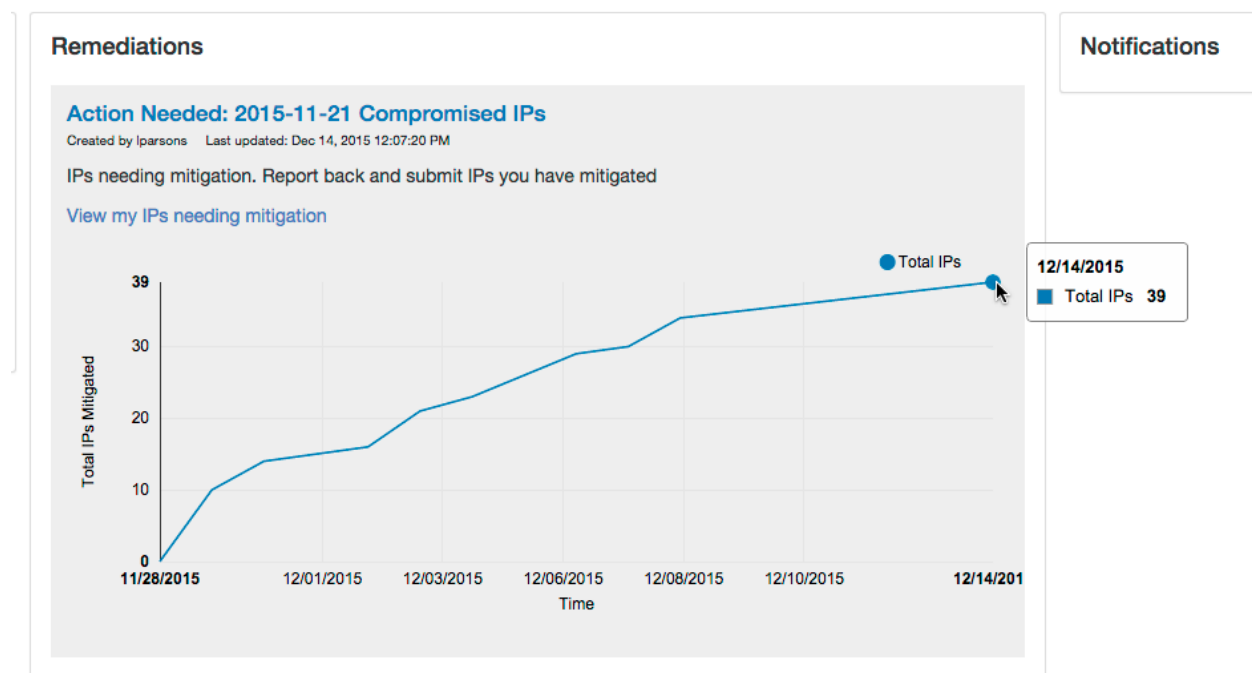
IPs needing remediation

You are responsible for taking mitigating action against the following IP addresses. Check off the IPs you are submitting as remediated and click **Submit**

<input checked="" type="checkbox"/> 5.144.130.38	<input type="checkbox"/> 81.162.73.224	<input type="checkbox"/> 193.107.16.206
<input checked="" type="checkbox"/> 37.0.124.155	<input type="checkbox"/> 91.106.207.103	<input type="checkbox"/> 193.107.17.72
<input checked="" type="checkbox"/> 37.140.192.17	<input type="checkbox"/> 91.189.136.16	<input type="checkbox"/> 195.211.154.177
<input checked="" type="checkbox"/> 37.152.88.39	<input type="checkbox"/> 91.200.32.231	<input type="checkbox"/> 195.211.154.180
<input checked="" type="checkbox"/> 37.247.101.58	<input type="checkbox"/> 91.213.96.32	<input type="checkbox"/> 195.211.155.227
<input type="checkbox"/> 46.20.12.227	<input type="checkbox"/> 93.170.131.69	<input type="checkbox"/> 202.190.179.48
<input type="checkbox"/> 46.29.255.100	<input type="checkbox"/> 162.251.113.166	<input type="checkbox"/> 212.79.127.227
<input type="checkbox"/> 65.207.23.201	<input type="checkbox"/> 178.18.25.125	<input type="checkbox"/> 212.235.117.238
<input type="checkbox"/> 72.69.215.66	<input type="checkbox"/> 186.233.144.136	<input type="checkbox"/> 213.142.143.194
<input type="checkbox"/> 77.221.144.40	<input type="checkbox"/> 189.203.240.71	

Submit

Close



form to submit the suspect IPs that the user probably received on a Trident email list. The system then automatically parses the list and bins the IPs according to attributes belonging to users, creating a new activity that will appear in the Remediations list for those users that are affected. There will also be some sort of notification. (In the future this creation would be automated by a service that can process emails that come into the system.)

The **Watching** section lists Activities that the user has subscribed to, either by subscribing to a public activity created by someone else or by creating a new activity.

Watching

Flow Analysis 9-2014 95

Created by dittrich Last updated: Dec 14, 2015 12:07:32 PM

Search for records associated with suspicious CIDR block

Flow Analysis 1-2014

Created by lparsons Last updated: Dec 14, 2015 12:07:32 PM

Search for records associated with APT1 intrusion set

Flow Analysis 1-2014

Created by swamer Last updated: Dec 14, 2015 12:07:32 PM

Activities are collections of data, queries, etc. They can be public or private. If a user subscribes to a public activity, the user receives a notification when new data is added to the activity. This is a first cut at the UI, and most of the UI display/functions (creating, sharing, subscribing) are currently in progress and not online (server side API and associated modules exist). The only thing you can see right now in the UI is the list of activities.

System health

The status area on the left is mostly static at present. However, a link to open the consul UI in a new tab exists.

Click **Open Consul view**:

and the Consul UI will open in a new tab with the **NODES** tab selected.

(You can demonstrate the Consul UI at this point if desired.)

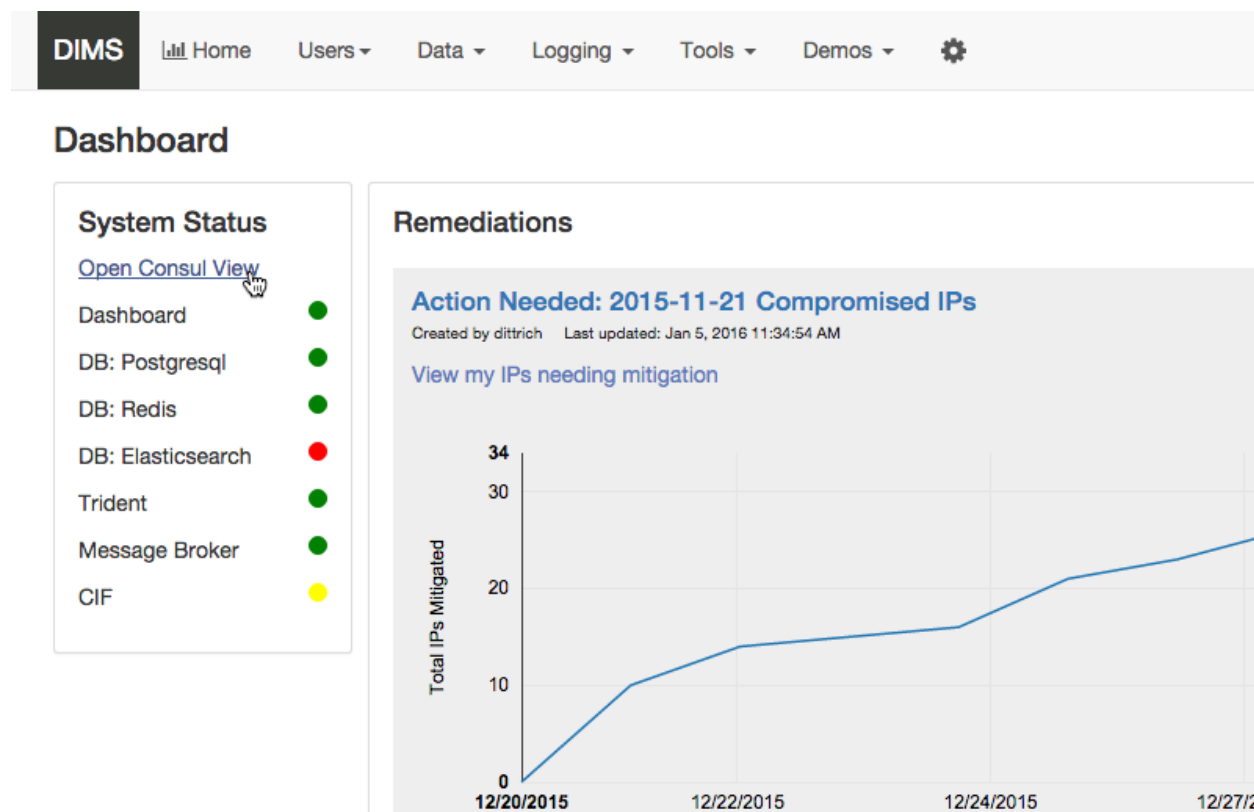
Live log streaming

The live log monitoring is now a popup panel so as to persist data across page views. That means the buffers won't be cleared if you go to a different section on the site (e.g. new page load).

1. Click **Logging** in the Navigation bar and select **Live log streaming**.

The Live log streaming window anchored to the bottom of the browser window will display.

There are tabs for the log exchanges the server monitors. Each tab has a button to turn on/ turn off that particular log monitor. The user can clear the buffer using the "Clear" button. The user can hide the window by clicking



the minimize button (down arrow in title bar), and then maximize it by clicking the maximize button (up arrow). Clicking the close button (X) turns off all monitors and closes the window. The window can also be closed by clicking **Logging > Live log streaming** in the Nav bar. (This is a toggle - if the window is active, clicking it closes the window. If the window is closed, clicking the button opens the window.)

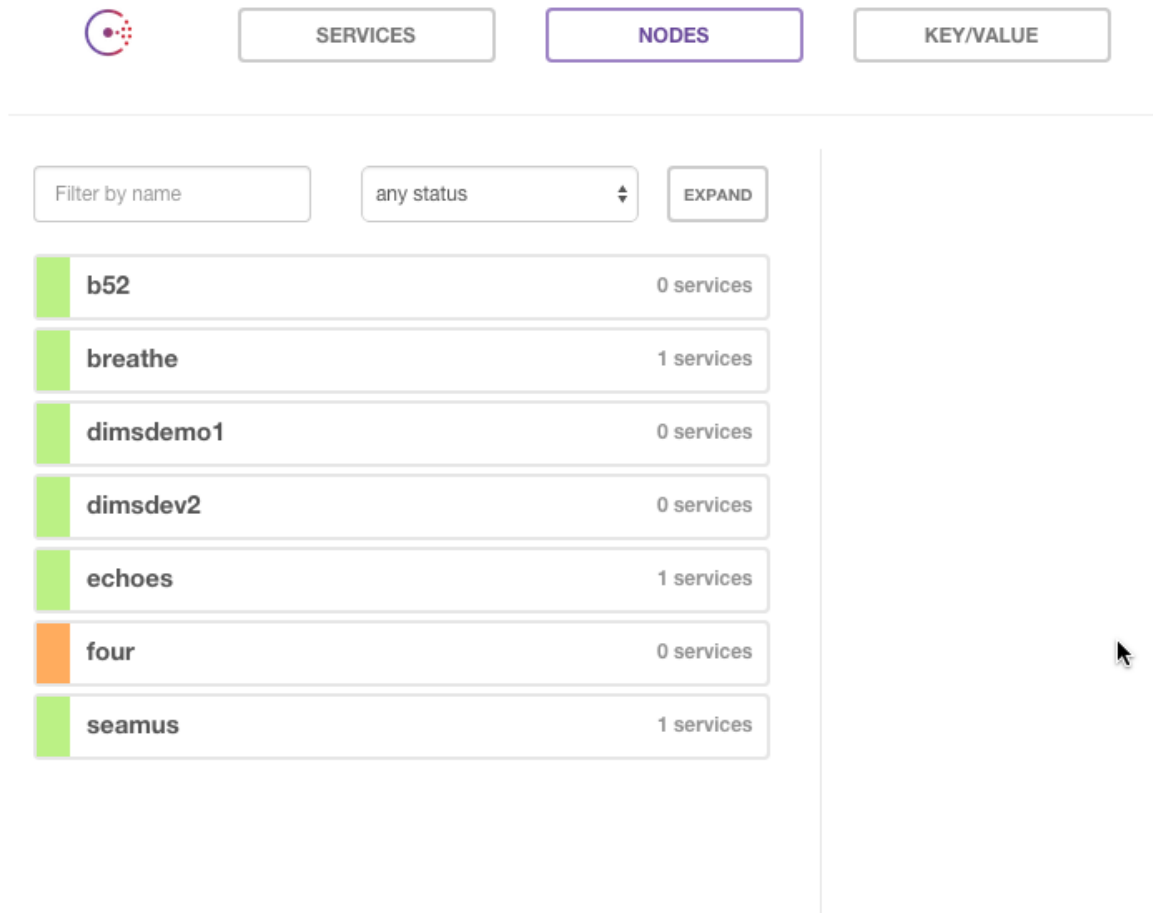
The *Live log streaming* window, like the *Chat* window, is independent of other page views. So it will remain active even if you go to a different view via a menu or navigation button.

2. Click on **Devops** tab and click button **Turn on Devops**
3. Do the same for Health - click on Health tab and click button **Turn on Health**
4. You could start an activity that reports to devops via another program, or wait a couple minutes and you'll probably get info on Health:
5. Click the minimize button:
and the logs will minimize to the bottom of the window.
6. Then click maximize to open it again.

The messages will still be there (maybe more).

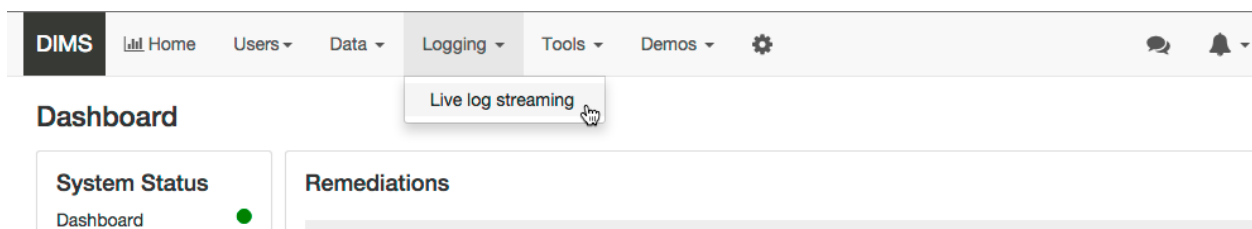
You can go to different locations in the app without clearing the log buffers. So go to **Users > Find DIMS users** to display users in your current trust group. The users will display behind the streaming window. Minimize the streaming log display to view the users:

7. You can clear the log buffers individually by clicking **Clear** in a log tab. To clear all the buffers and close the display, click the **Log Monitor** link in the nav bar or just click the **X** in the monitor window title bar.



The screenshot shows the DIMS Dashboard with the 'NODES' tab selected. At the top, there are three tabs: 'SERVICES', 'NODES' (highlighted with a purple border), and 'KEY/VALUE'. Below the tabs, there is a filter section with a text input 'Filter by name', a dropdown menu 'any status', and an 'EXPAND' button. The main content area displays a list of nodes, each with a colored square icon, a name, and a service count. The nodes are: 'b52' (green), 'breathe' (green), 'dimsdemo1' (green), 'dimsdev2' (green), 'echoes' (green), 'four' (orange), and 'seamus' (green). A mouse cursor is visible on the right side of the dashboard.

Node Name	Services
b52	0 services
breathe	1 services
dimsdemo1	0 services
dimsdev2	0 services
echoes	1 services
four	0 services
seamus	1 services



The screenshot shows the DIMS Dashboard footer. On the left, there is a 'DIMS' logo. To its right are navigation links: 'Home', 'Users', 'Data', 'Logging', 'Tools', 'Demos', and a settings gear icon. On the far right are communication icons: a speech bubble and a bell. Below the navigation links, the 'Logging' dropdown menu is open, showing 'Live log streaming' with a mouse cursor pointing at it. The main content area is divided into two sections: 'System Status' and 'Remediations'. The 'System Status' section shows 'Dashboard' with a green dot. The 'Remediations' section is empty.

DIMS | Home | Users | Data | Logging | Tools | Demos | Settings | Chat | Notifications

Logging dropdown menu:

- Live log streaming

Dashboard

System Status

Dashboard ●

Remediations

The image consists of three sequential screenshots of the DIMS Dashboard, illustrating the 'Live log streaming' feature.

Top Screenshot: The dashboard header shows the 'Logging' menu open with the 'Live log streaming' option highlighted. The main content area includes a 'System Status' sidebar with indicators for Dashboard (green), DB: Postgresql (green), DB: Redis (green), and DB: Elasticsearch (red). The 'Remediations' section displays an 'Action Needed: 2015-11-21 Compromised IPs' alert, created by 'Iparsons' and last updated on 'Dec 13, 2015 6:35:33 PM'. A table below the alert shows a count of '34'.

Middle Screenshot: The 'Live log streaming' panel is shown with the 'Devops' tab selected. The 'Turn on Devops' button is highlighted, indicating the feature is being activated.

Bottom Screenshot: The 'Live log streaming' panel is shown with the 'Devops' tab selected. The 'Turn off Devops' button is highlighted, indicating the feature is being deactivated. The log stream area shows the message '[*] Waiting for messages...'.

DB: Elasticsearch ● 34

Live log streaming ✕

Devops Health Logs Test Report CI Testing

Turn off Health Clear

```
[*] Waiting for messages...
2016-01-05T19:26:37.788Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:26:37.840Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
```

DB: Elasticsearch ● 34

Live log streaming ✕

Devops Health Logs Test Report CI Testing

Turn off Health Clear

```
[*] Waiting for messages...
2016-01-05T19:26:37.788Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:26:37.840Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
```

Created by iparsons Last updated: Dec 13, 2015 6:35:49 PM

Live log streaming ✕

Live log streaming ✕

Devops Health Logs Test Report CI Testing

Turn off Health Clear

```
[*] Waiting for messages...
2016-01-05T19:26:37.788Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:26:37.840Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
2016-01-05T19:27:37.809Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:27:37.869Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
2016-01-05T19:28:37.819Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:28:37.889Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
2016-01-05T19:28:40.360Z u12-dev-svr-1 ac017944-da2d-40e6-883f-af349c6dfdc9 dims-dashboar [utils/healthLogger.js] [12334] INFO redis healthy localhost:6379, database 0
```

Live log streaming ✕

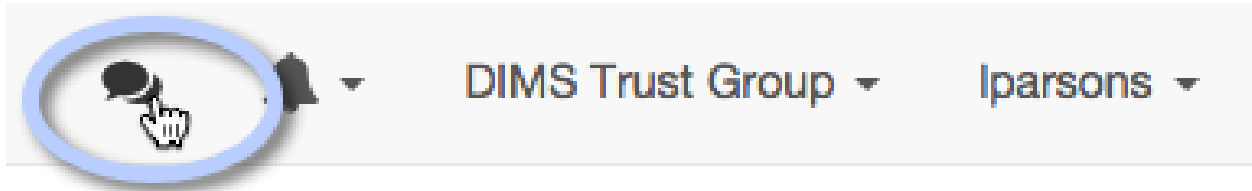
Devops Health Logs Test Report CI Testing

Turn off Health Clear

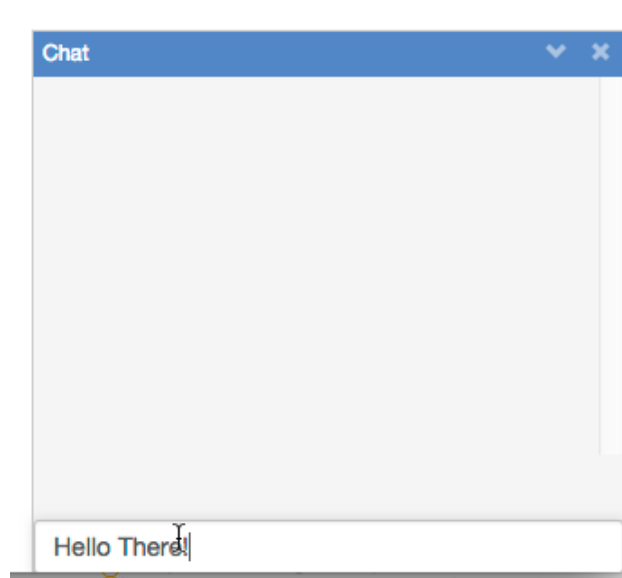
```
[*] Waiting for messages...
2016-01-05T19:26:37.788Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:26:37.840Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
2016-01-05T19:27:37.809Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:27:37.869Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
2016-01-05T19:28:37.819Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:28:37.889Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
2016-01-05T19:28:40.360Z u12-dev-svr-1 ac017944-da2d-40e6-883f-af349c6dfdc9 dims-dashboar [utils/healthLogger.js] [12334] INFO redis healthy localhost:6379, database 0
2016-01-05T19:29:37.821Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:29:37.902Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
2016-01-05T19:30:37.826Z u12-dev-svr-1 f8368f86-d9ef-4b62-9f5b-f70c841fb120 dims-dashboar [utils/healthLogger.js] [12334] INFO dashboard healthy
2016-01-05T19:30:37.878Z u12-dev-svr-1 4175bde1-30cb-4866-9d3c-b9bd9c5543e1 dims-dashboar [utils/healthLogger.js] [12334] INFO postgresql healthy at postgresql://localhost/ops-trust
```

Chat

1. Click the chat icon in the Nav bar to open the chat window:



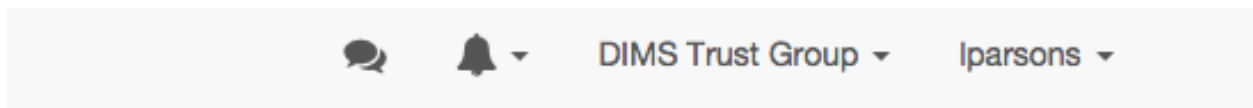
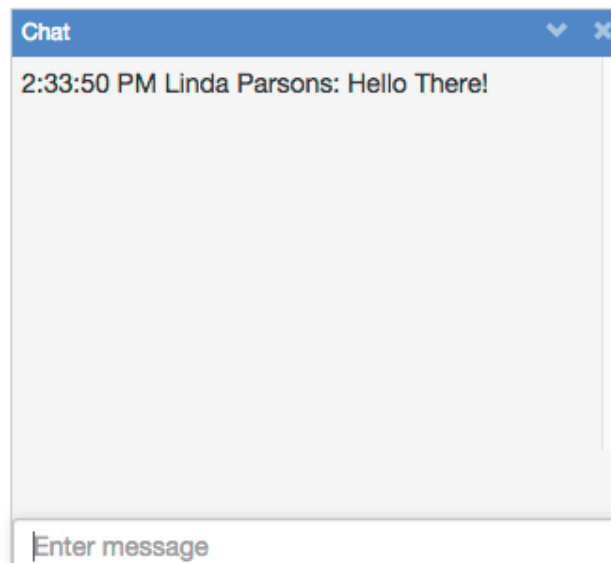
Unless you're chatting with someone else who is logged in, there isn't much to see (you can send messages to yourself however). Enter a message in the message area of the chat box and press **Enter** key.



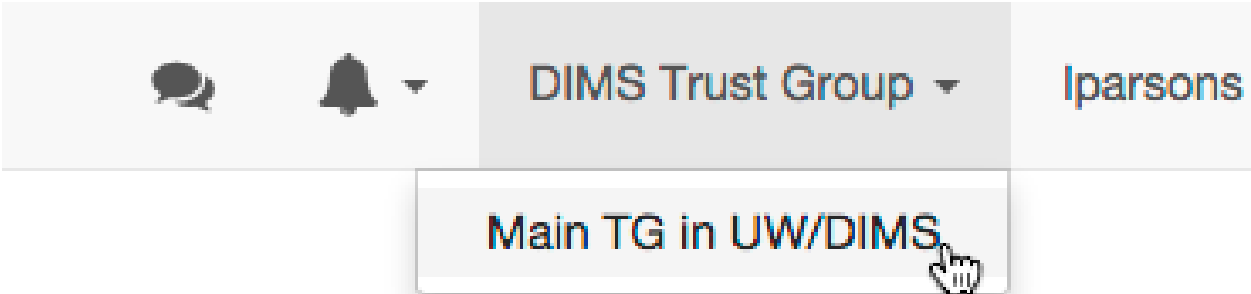
The message you sent will appear in your chat window:

User display and trust groups

1. Note that the name of the trust group you are logged into displays on the menu bar:
The system remembers your last selection. If you have never selected a trust group, it will choose the first one in your list of trust groups when you first log in.
2. Display your profile information by selecting `dittrich > Profile` in the nav bar. Note that the trust group info now displays in the profile.
3. Change your trust group by clicking on the current trust group in the nav bar and selecting an option that displays in the menu. (If you are only in one trust group then no options will display.)
4. Note that the trust group listed in your profile will change to the current trust group:

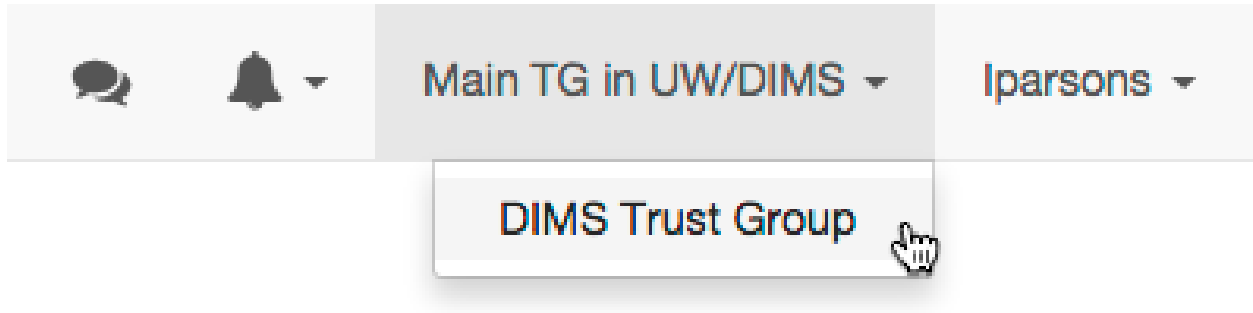


Admin	true
SysAdmin	true
Trust Group	DIMS Trust Group
Trust Group ID	dims
PGP Key Expiration	



Admin	true
SysAdmin	true
Trust Group	Main TG in UW/DIMS
Trust Group ID	main
PGP Key Expiration	

5. To see the users in your currently selected trust group, select **Users > Find DIMS Users** on the nav bar. The list of users in the current trust group will display.
6. Again, change the trust group via the trust group menu in the nav bar. The list of users will change to reflect the users in the new current trust group.



There are currently two types of tickets:

- Activity
- Mitigation

Activities are ad hoc ways for users to save information. An activity can have any number of topics associated with it, and users can create topics and add them to an activity.

Mitigation tickets are more structured in that a user can create one by making the appropriate api call and supplying a list of IPs, but the system automatically creates all associated topics, which have specific purposes in a mitigation activity.

General ticket structure

A *ticket* is described

- A Redis key/value pair where the value is a 1-level hash
- Zero or more associated “topi”

Topics

A topic is always associated with a parent ticket. The parent key can be derived from the topic key.

When a topic is added to a ticket, its key is added to the set of topic keys owned by the ticket (parent).

Topics are stored in redis as follows:

Table 2.1: Topic storage

Data	Key	Value
Metadata	Topic metadata key	(hash) JSON metadata

We currently restrict the metadata saved for a topic. We do not allow user-defined metadata. Should we change this behavior?

Metadata provided by calling method

```
{
  datatype: (required) 'set' or 'string'
  name: (required) name of topic
  description: (optional) description of topic (default - '')
}
```

```
{
  createTime: Unix epoch time when topic is created
  modifiedTime: Unix epoch time when topic modified
  num: Topic counter: (via topicCounterKey) - used to ensure uniqueness
}
```

CHAPTER 3

Ticket API

The dashboard server provides a REST API for working with *tickets*.

Note: One API design decision was how *open* to make the API - that is, would the API automatically restrict data sent back for certain requests. For example, we have the concept of *public* and *private* tickets. When a GET request is made, do we want the API to return all tickets or a subset such as all public and all private belonging to the current user?

The API is currently restrictive - a GET request to `/api/tickets` would return all public activity tickets and all private tickets owned by the calling user.

The API is intended to require authentication. The plan is that the client would have obtained the token for the calling user and included it with the API call. The server will then look up the attributes for the user referenced by the token. These would be:

- username - user name (in Ops-trust) of the calling user
- trustgroup - trust group the user is logged into
- admin - is the user an admin in the trust group

This has not been implemented yet as we need to determine how to integrate this with Trident and their login tokens, as well as have a Trident instance running.

In the interim, the server currently authenticates users with the Dashboard via their Ops-trust (Trident) usernames and passwords and establishes a persistent login session for the user. This can be used to protect the API endpoints when accessing via the Dashboard client, but would prevent other clients from accessing. So the API is not protected by the current authentication mechanism in order to allow other clients access until we get the token authentication implemented.

Until token authentication is implemented however, requests from clients other than the Dashboard will not be able to retrieve user private tickets.

HTTP Verbs

GET	/api/ticket	list
POST	/api/ticket	create
GET	/api/ticket/:id	show
PUT	/api/ticket/:id	update
DELETE	/api/ticket/:id	delete

Responses

The API returns JSON. JSON responses follow the unofficial JSEND spec. See <http://labs.omniti.com/labs/jsend/wiki> for more information.

Successful requests will return JSON with a status of `success` and a `data` property with the JSON result.

```
{
  "data": <json>
  "status": "success"
}
```

The `data` property will generally be in the following form for one ticket:

```
"data": {
  "ticket": {
    <json describing ticket>
  }
}
```

or for multiple tickets:

```
"data": {
  "tickets": [ <array of json where each one describes a ticket> ]
}
```

Since mitigations are a *special* form of ticket, we use the terms `mitigation` and `mitigations` as keys to their response:

```
"data": {
  "mitigation": {
    <json describing mitigation ticket>
  }
}
```

Requests that do not send back data (such as delete) will return with `data` set to `null`:

```
{
  "status": "success",
  "data": null
}
```

Unsuccessful requests will return JSON with an error message and a status of `error`:

```
{
  "message": "You do not have permission to access this ticket",
  "status": "error"
}
```



```
{
  "status": "error"
}
```

Requests that failed due to invalid data or parameters submitted may generate a `fail` response:

```
{
  "status": "fail",
  "data": <wrapper for reason request failed>
}
```

For example:

```
{
  "status": "fail",
  "data": {
    "name": "A name for the new ticket is required"
  }
}
```

Note: Currently most errors are reported as `error` rather than `fail`. We are working on refactoring so that errors that should be reported as `fail` are done so.

An HTTP status code is included in the response headers. For example, the following request returns with 400:

```
$ curl -k -I http://192.168.56.103/api/ticket
HTTP/1.1 400 Bad Request
Server: nginx/1.8.0
Date: Wed, 13 Jan 2016 16:44:48 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 96
Connection: keep-alive
X-Powered-By: Express
ETag: W/"60-kIP4LSNmFtWUQFvEw0Zo/g"
set-cookie: connect.sid=s%3Ah6h88KJrXfxT4Ycabeldk5aTFY26SRx8.
o7d2T9y03YrbbR8ssnmF0tFEpfV9VNI6F7l9oJQEgAg; Path=/; HttpOnly
```

Retrieve a list of tickets

Returns list of tickets

No parameters

When no parameters are provided, the system defaults to the following parameters:

```
type: 'activity',
```

This will return all public activities plus any private activities belonging to the calling user.

Invoked via GET http://dashboard_url/api/ticket/ Returns HTTP status code and string reply

Using curl: `curl -k https://dashboard_url/api/ticket/`

Sample response:

```
{ "data": [
  "ticket:1",
  "ticket:2",
  "ticket:3" ]
}
```

With query parameters

```
private: boolean
type: string
ownedBy: string
open: boolean
```

type can be mitigation or activity. For mitigation tickets, no other parameters are needed, and any extra provided are ignored.

```
$ curl -k http://192.168.56.103/api/ticket?type=mitigation | python -m json.tool
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100  1177  100  1177    0     0  34609      0 --:--:-- --:--:-- --:--:-- 35666
{
  "data": [
    {
      "data": [
        [
          1450004398760,
          0
        ],
        [
          1450068277926,
          6
        ],
        [
          1450085780836,
          13
        ],
        [
          1452404888722,
          329
        ],
        [
          1452478109289,
          343
        ]
      ],
      "ips": {
        "data": [],
        "user": null
      },
      "key": "dims:ticket:mitigation:1",
      "metadata": {
        "createdTime": 1452682798841,
        "creator": "lparsons",
        "description": "IPs needing mitigation. As you mitigate IPs, submit_
→them here.",
        "initialNum": 1408,
```

```

        "mitigatedNum": 343,
        "modifiedTime": 1452682798841,
        "name": "Action Needed: 12/12/2015 Compromised IPs",
        "num": 1,
        "open": true,
        "private": false,
        "type": "mitigation",
        "unknownNum": 864
    }
}
1,
"status": "success"
}

```

Creating an activity

```

/**
Returns a ticket: ticket key, ticket metadata, list of associated topic keys
@return HTTP Status code and string reply.

@example
Example response:
{"data": {
  "ticket": {
    "num": "1",
    "creator": "testuser",
    "type": "data",
    "createdTime": "1418060768120",
    "open": "true",
    "key": "ticket:1",
    "topics": ["ticket:1:data:cif:results:result1.txt",
               "ticket:1:data:cif:results:result2.txt"]
  }
}

@example How to invoke
GET https://dashboard_url/api/ticket/ticket:1

Using curl:
curl -k https://dashboard_url/api/ticket/ticket:1

@param {string} id Ticket key in format ticket:<num>
/

/**
Creates a new ticket
@method create
@return HTTP Status code and string reply.
{"data": {
  "ticket": {
    "num": "2",
    "creator": "testuser",
    "type": "data",
    "createdTime": "1418060768120",
    "open": "true",
    "key": "ticket:2"
  }
}

```

```

    }
  }
}
@example

POST https://dashboard_url/api/ticket/
  body:
  {
    "type": "data",
    "creator": "testuser"
  }

Using curl:
  curl --data "type=data&creator=testuser" -k https://dashboard_url/api/ticket

@param {string} type Type of ticket being created
@param {string} creator Username of user creating ticket (optional if user logged in,
                    ignored if user logged in)
/
/**
Adds a topic (metadata) to a ticket and saves the data (content)
@method addTopic
@return HTTP Status code and string reply.
@example
Sample json response:

  { "data": {
    "topic": {
      "parent": {
        "num": "12", "creator": "testUser", "type": "analysis", "createdTime":
↪ "1418131797522", "open": "true"
      },
      "type": "analysis",
      "name": "namesearch:result2",
      "dataType": "hash"
    },
    "content": { "firstname": "bob", "lastname": "johnson" },
    "key": "ticket:12:analysis:namesearch:result2"
  }
}

@example
Example URI
  POST https://dashboard_url/api/ticket/ticket:27/topic
  body:
  {
    "name": "cif:results:1418060768120",
    "dataType": "string",
    "content": <string content>
  }

Note that content in this example could be JSON that is stringified. Content could
↪ also be content of a
file, base64'd, as in
  POST https://dashboard_URL/api/ticket/ticket:28/topic
  body:
  {
    "name": "mal4s:result:result1.png",
    "dataType": "string",

```

```

    "content": <base64 content of a .png file>
  }

Using curl with hash content (content is uri encoded):
    curl --data "name=name&search:results&data&type=hash&content=%7B%22firstname%22%3A%22bob%22%2C%22lastname%22%3A%22johnson%22%7D" -k https://dashboard_url/api/ticket/
    ticket:12/topic

A successful response from the curl command might look like the following (line
    feeds added for clarity - reponse is just a string):
    {
      "data": {
        "topic": {
          "parent": {
            "num": "12", "creator": "testUser", "type": "analysis", "createdTime":
            "1418131797522", "open": "true",
            "type": "analysis", "name": "name&search:result2", "data&type": "hash",
            "content": {
              "firstname": "bob", "lastname": "johnson"
            }, "key":
            "ticket:12:analysis:name&search:result2"
          }
        }
      }
    }

@param {string} id Ticket key in format ticket:<num>
@param {string} name Name of the topic - this represents the last part of the topic
    key after
    ticket:<num>:<ticket_type>:
@param {string} dataType Redis data structure to store the contents in - can be
    string or hash
@param {string} content Content to be stored

Note that content is optional if type is string. If no content is specified, then an
    empty string
    is stored at the topic key. You would use this if you want to use the contents of a
    file as the
    data to be stored. First create the topic with a type of string and no content. Then
    you use the
    returned topic key and do an update (PUT) of the topic with the uploaded file.

You cannot overwrite an existing topic with the same key. An error is returned if
    the topic already
    exists
/

/**
Retrieves a ticket topic's metadata and content. Invoked via GET

<pre>Sample response:

{
  "data": {
    "topic": {
      "parent": {
        "num": "12", "creator": "testUser", "type": "analysis", "createdTime":
        "1418131797522", "open": "true"
      },
      "type": "analysis",
      "name": "name&search:result2",
      "data&type": "hash"
    },
    "content": {
      "firstname": "bob", "lastname": "johnson"
    },
    "key": "ticket:12:analysis:name&search:result2"
  }
}
</pre>

```

```

@method showTopic

@example

    GET https://dashboard_url/api/ticket/topic/ticket:27:analysis:namesearch:result2

    Using curl:

        curl -k https://dashboard_url/api/ticket/topic/
↪ticket:27:analysis:namesearch:result2

@param {string} id Ticket topic key in format ticket:<num>:<type>:<topic_name>
@return HTTP Status code and string reply.
/

/**
Updates a ticket topic. You can only update content.
@method updateTopic
@return HTTP Status code and string reply.
    {"data":{"
        "topic":{"
            "parent":{"
                "num":"12","creator":"testUser","type":"analysis","createdTime":
↪"1418131797522","open":"true"
            },
            "type":"analysis",
            "name":"namesearch:result2",
            "dataType":"hash"
        },
        "content":{"firstname":"john","lastname":"johnson"},
        "key":"ticket:12:analysis:namesearch:result2"
    }}
}
@example

    PUT https://dashboard_url/api/ticket/ticket:27/topic
    body:
    {
        "content": <string content>
    }
    Note that content in this example could be JSON that is stringified. Content could
↪also be content of a
    file, base64'd, as in
    PUT https://dashboard_URL/api/ticket/ticket:28/topic
    body:
    {
        "content": <base64 content of a .png file>
    }

    Using curl with hash content (content is uri encoded):
        curl --data "content=%7B%22firstname%22:%22john%22,%22lastname%22:%22johnson%22
↪%7D" -k https://dashboard_url/api/ticket/ticket:12/topic

    A successful response from the curl command might look like the following (line
↪feeds added for clarity - reponse is just a string):
    {"data":{"
        "topic":{"

```

```
    "parent":{"num":"12","creator":"testUser","type":"analysis","createdTime":
↪ "1418131797522","open":"true"},
    "type":"analysis","name":"namesearch:result2","dataType":"hash"},
    "content":{"firstname":"john","lastname":"johnson"},"key":
↪ "ticket:12:analysis:namesearch:result2"}}

@param {string} id Topic key in format ticket:<num>:<type>:<topic_name>
@param {string} content Content to be stored

/
```


CHAPTER 4

Contact

Section author: Dave Dittrich (@davedittrich) <dittrich @ u.washington.edu>

CHAPTER 5

License

Copyright © 2014,2015 University of Washington. All rights reserved.

```
Berkeley Three Clause License
=====
```

```
Copyright (c) 2014, 2015 University of Washington. All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
```

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```