
Walkoff Documentation

Release 1.0

United States Government

Nov 07, 2019

Contents:

1	What is WALKOFF?	3
2	Pre-requisites	5
3	Deploying WALKOFF	7
3.1	Interface Overview	8
3.2	Walkoff Development	12
3.3	Prepackaged Applications	22
3.4	Application Development	23
3.5	API Gateway	32
3.6	Changelog	42
	HTTP Routing Table	63

This documentation is intended as a reference for app and workflow developers as well as project contributors and operators. Here you will find walkthroughs, tutorials and other useful information about applications that are shipped with Walkoff, our changelog, and how to interact with Walkoff using its RESTful API.

CHAPTER 1

What is WALKOFF?

WALKOFF is a flexible, easy to use, automation framework allowing users to integrate their capabilities and devices to cut through the repetitive, tedious tasks slowing them down,

WHAT WE OFFER

- *Easy-to-use:* Drag-and-drop workflow editor. Sharable apps and workflows.
- *Flexibility:* Deployable on Windows or Linux.
- *Modular:* Plug and play integration of almost anything with easy-to-develop applications.
- *Visual Analytics:* Send workflow data to custom dashboards (and soon, Elasticsearch & Kibana!)

CHAPTER 2

Pre-requisites

Ensure that Docker, Docker Compose 3+, and git are installed!

- Docker CE: <https://docs.docker.com/install/#supported-platforms>
- Docker Compose: <https://docs.docker.com/compose/install/>
- Git: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

If you do not already have a Docker Swarm initialized or joined, run the following command to create one:

```
docker swarm init
```

Note: If you have multiple NICs you will need to use `--advertise-addr` to pick an address from which the swarm will be accessible.

Deploying WALKOFF

1. Open a terminal and clone WALKOFF:

```
git clone -b development https://github.com/nsacyber/WALKOFF.git
```

2. Move into the WALKOFF directory:

```
cd WALKOFF
```

3. Launch WALKOFF with the bootloader, building components as well. If you are on Windows, be sure to use PowerShell and not Command Prompt.

```
# Windows PowerShell
.\walkoff.ps1 up --build

# If verbose output is desired:
.\walkoff.ps1 up --build --debug
```

```
# Unix shell
./walkoff.sh up --build

# If verbose output is desired:
./walkoff.sh up --build --debug
```

4. Navigate to the default IP and port. The default port can be changed by altering the port NGINX is exposed on (the right-hand port) in *bootloader/walkoff-compose.yml*. Note that you should use HTTPS, and allow the self-signed certificate when prompted.

```
https://127.0.0.1:8080
```

5. The default username is “admin” and password is “admin”. These can and should be changed upon initial login.
6. To stop WALKOFF, use the bootloader:

```
# Windows PowerShell
.\walkoff.ps1 down

# If removing encryption key, stored images, verbose output, and postgres_
↪volume is desired:
.\walkoff.ps1 down --key --registry --debug --volume

# Unix shell
./walkoff.sh down

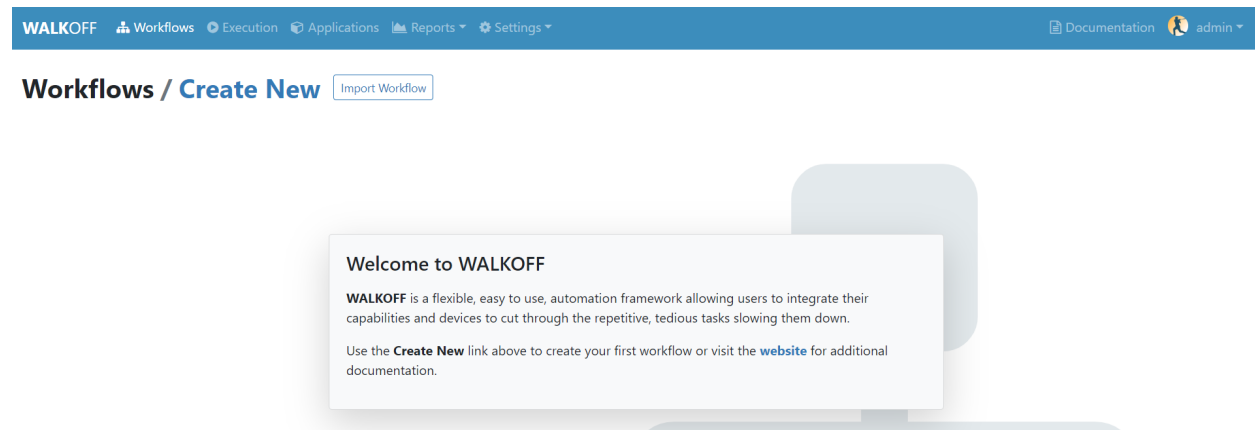
# If removing encryption key, stored images, verbose output, and postgres_
↪volume is desired:
./walkoff.sh down --key --registry --debug --volume
```

3.1 Interface Overview

In this tutorial, we will briefly cover each of the components of the WALKOFF UI.

3.1.1 Workflows

Here, you can create or import a WALKOFF workflow. To build your own workflow, click **Create New**. You can also import and export workflows from this page from/to your local filesystem.



3.1.2 Execution

Here, you can find the status of Workflows that have begun execution as well as finding the results from said Workflows. You can also choose to abort workflows from this page.

WALKOFF Workflows Execution Applications Reports Settings

Documentation admin

Execution

Select a Workflow Filter Results...

Name	ID	User	Started At	Completed At	Status	Current App	Current Node	Actions
No data to display								
0 total								

3.1.3 Applications

Here you can find all of the apps. It is home to the app editor, in which you can edit and update application images that Walkoff uses to run workflows. Make sure that once you edit and app, you click rebuild to see the changes persist to your local copy of the file.

WALKOFF Workflows Execution Applications Reports Settings

Documentation admin

Manage Applications

basics 1.0.0
 An example of a Walkoff App specification
[Edit Application](#) [Rebuild Image](#)

Builtin 1.0.0
 Walkoff built-in functions useful in workflow development.
[Edit Application](#) [Rebuild Image](#)

hive 1.0.0
 The Hive app allows for walkoff to generate or close cases in TheHive
[Edit Application](#) [Rebuild Image](#)

ip_addr_utils 1.0.0
 An IP address app that will allow users to specify ip addresses and will format them correctly
[Edit Application](#) [Rebuild Image](#)

nmap 1.0.0
 A simple app to interact with map
[Edit Application](#) [Rebuild Image](#)

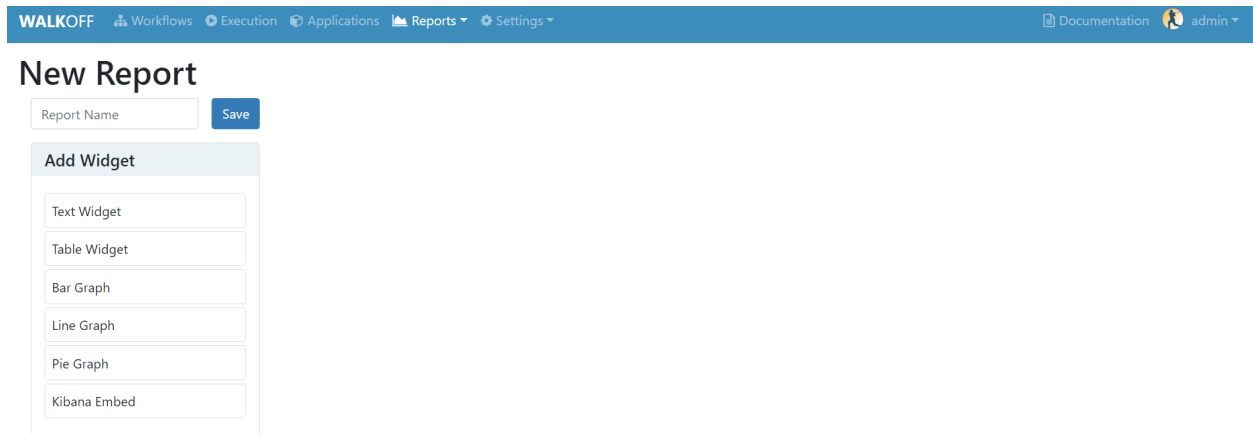
power_shell 1.0.0
 A power shell app that can run commands on a remote host.
[Edit Application](#) [Rebuild Image](#)

ssh 1.0.0
 Executes ssh shell commands via SSH
[Edit Application](#) [Rebuild Image](#)

walk_off 1.0.0
 A meta application that utilizes several endpoints of WALKOFF's API
[Edit Application](#) [Rebuild Image](#)

3.1.4 Reports

This is the home of custom reports for WALKOFF. These can be used to provide visualizations, telemetry, etc. using results directly from Workflow results. Currently, only CSV output is supported; results being piped to Elasticsearch are in the process of being implemented.

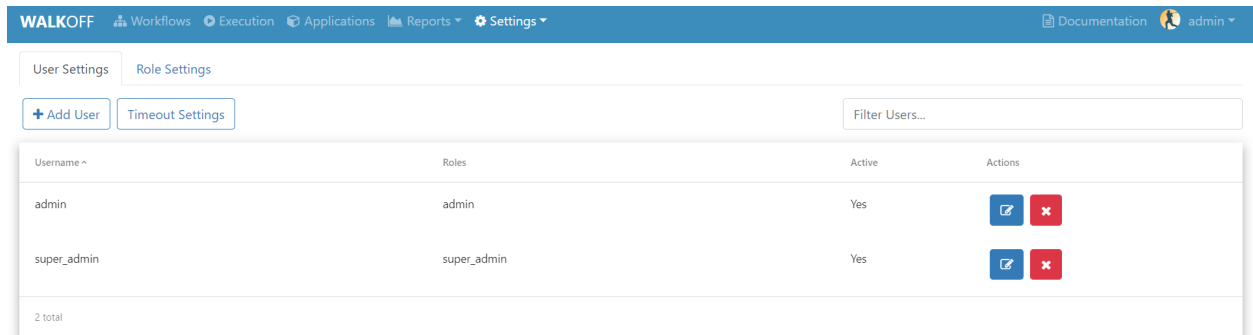


3.1.5 Settings

Here you can find options necessary to make Walkoff more configurable, such as adding Users or global variables. You can also schedule workflow execution through the scheduler or even manage your docker swarm using Portainer. Look below for a description of the features mentioned in this drop-down tab.

Users

Here you can add users or update role settings/permissions that can lock down access to certain tabs in Walkoff.



Globals

Global Variables can be used to store information such as credentials, configuration, etc. that can be referenced inside workflows. These are stored as encrypted values for security purposes.

WALKOFF
Workflows
Execution
Applications
Reports
Settings
Documentation
admin

Global Variables

+ Add Global
Filter Globals...

Name	ID	Value	Description	Actions
No data to display				
0 total				

Scheduler

Here, you can create and configure schedules for running workflows.

WALKOFF
Workflows
Execution
Applications
Reports
Settings
Documentation
admin


Scheduler

+ Add To Schedule
Scheduler Stopped
Filter Results...

Name ^	Workflow(s)	Status	Type	Rule	Actions
No data to display					
0 total					

Portainer

Portainer is a UI where you can manage your docker swarm allowing you to do things like scaling or viewing logs of specific containers. To navigate here, select portainer.io from the settings drop down menu. It will take you to this page



Please create the initial administrator user.

Username
admin

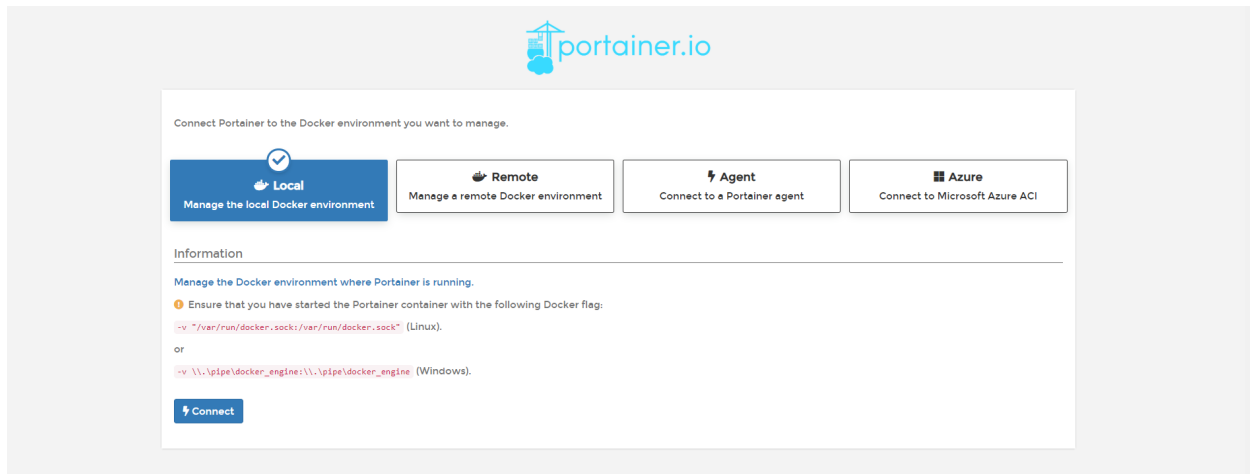
Password

Confirm password

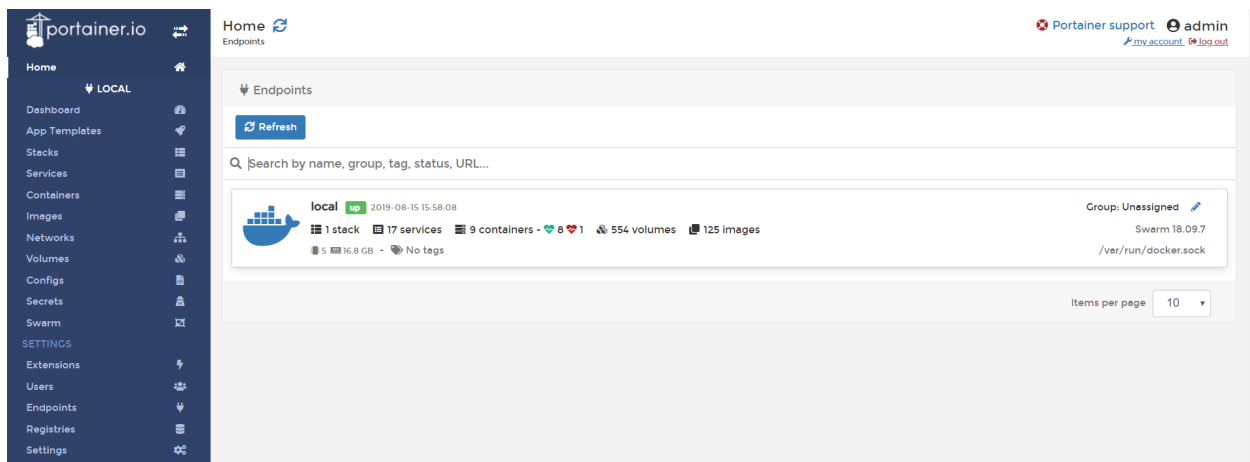
The password must be at least 8 characters long

Create user

Make sure to enter a password that is at least 8 characters long. Once you create your password, if prompted with four selection options for the configuration of Portainer, select “Local”, then click “Connect”



Once you click “Connect” it should take you to the portainer UI which is linked to our docker swarm. Feel free to look around and check out all of the cool things that portainer has to offer!



3.2 Walkoff Development

In this page we will discuss the workflow editor, how to create a workflow, and the tools available to you.

3.2.1 Workflow Creation

In this tutorial, we will create a basic workflow to demonstrate the general features of the workflow editor.

Create Global

Before we create a workflow, we will create a global variable that our actions will use.

In the top navigation bar, click on **Settings**, then **Globals**, then **Add Global**. Fill out the required fields for this example and set the value to any string of your choice. Here, you may also opt to restrict access to your variable using role-based permissions as described in [User and Role Creation](#). Then click **Save** in the dialog box. If nothing is populated in the globals table, then verify that you created your encryption key as described in [Pre-requisites](#).

Add Global Variable

Name
Example Global

Value
This is our first workflow!

Description (optional)

Restrict Access

By default anyone with access to WALKOFF will have full permissions on this resource. You can limit access to only users of specific roles by selecting the role and access level below.

Role	Permission
No Role Based Permissions	

Select Role... Select Access Level... Add New

Close Save

Create Workflow

In the top navigation bar, click **Workflows** to return to the main Workflow page. Then, near the top of the page, click the **Create New** button. Enter a name for a new workflow (and, optionally, tags and description). Here, you may also opt to restrict access to your variable using role-based permissions as described in [User and Role Creation](#). Finally, click **Continue**.

Create Workflow

Enter a Workflow Name
Hello World Example

Enter a Description (Optional)
This is an example Workflow

Add Tags (Optional)
Add Tags...

Restrict Access

By default anyone with access to WALKOFF will have full permissions on this resource. You can limit access to only users of specific roles by selecting the role and access level below.

Role	Permission
No Role Based Permissions	

Select Role... Select Access Level... Add New

Cancel Continue

Add Actions to Workspace

Let's begin by adding a `hello_world` action and a `echo_string` action from the HelloWorld app. Expand the `basics` app by clicking on the app name in the left pane. Then, double-click, or click and drag the desired actions into the workspace.

Ensure that the `hello_world` action is set as the starting node by clicking `Set as Start Action` in the Action Parameters pane.

The screenshot shows the Walkoff web interface. The top navigation bar includes 'WALKOFF', 'Workflows', 'Execution', 'Applications', 'Reports', and 'Settings'. The main header indicates the current workflow is 'Hello World*[Incomplete]'. On the left, a sidebar lists 'Builtin' and 'basics' categories. The 'basics' category is expanded, showing a list of actions: 'hello_world', 'random_number', 'string_to_json', and 'echo_string'. The 'hello_world' action is highlighted. In the center workspace, a workflow diagram shows a 'hello_world' action connected to an 'echo_string' action. On the right, the 'Action Settings' pane for 'hello_world' is displayed. It includes fields for 'ID' (2c455399-57b0-7461-8a49-58da66960882), 'Name' (hello_world), and 'Returns: (string)' (HELLO WORLD FROM host.name). A 'Set Start Action' button is visible in the top right of the settings pane.

Configure Options

Some actions will have inputs; some required, some optional. In this case, the `echo_string` action has a required parameter, but the `hello_world` action does not. Set the `echo_string` parameter's type to `global` from the drop down and select the global that you created in the previous examples. The action will reference this global to echo the string.

Finally, connect the actions together by clicking and dragging from the top of the `hello_world` action to the top of the pause action. You can also right-click and drag from one action to the other.

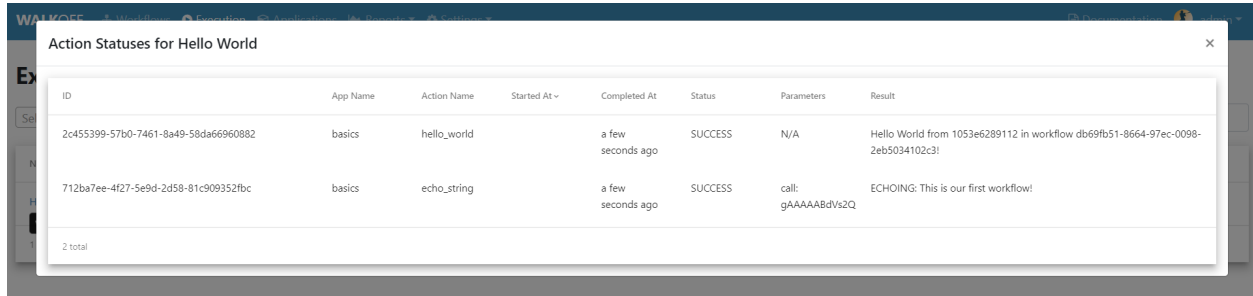
The screenshot shows the Walkoff web interface with the 'Hello World' workflow. The 'echo_string' action is now selected in the 'Action Settings' pane. The 'Arguments' section shows 'Call (string)' set to 'message to repeat' with a dropdown menu set to 'Global' and 'Example Global'. The 'Returns: (string)' section shows 'Example' set to 'REPEATING: Hello world'. The workflow diagram in the center workspace shows the 'hello_world' action connected to the 'echo_string' action by a line from the top of 'hello_world' to the top of 'echo_string'.

Save and Execute Workflow

Using the buttons in the toolbar, save, then execute the workflow. Workflows must be saved before execution in order for the execution to use the current state of the workflow.

Examine Results:

Check the results of your workflow under the `Execution` tab below your workflow. These results can also be viewed later under the `Execution` tab at the top of the screen. If everything was configured properly, you can expect to see results similar to what is shown below.



ID	App Name	Action Name	Started At	Completed At	Status	Parameters	Result
2c455399-57b0-7461-8a49-58da66960882	basics	hello_world		a few seconds ago	SUCCESS	N/A	Hello World from 1053e6289112 in workflow db69fb51-8664-97ec-0098-2eb5034102c3!
712ba7ee-4f27-5e9d-2d58-81c909352fbc	basics	echo_string		a few seconds ago	SUCCESS	call: gAAAAABdVs2Q	ECHOING: This is our first workflow!
2 total							

3.2.2 Workflow Editor

In this tutorial, we will explore the different components of the Workflow Editor interface.

Toolbar



From left to right, the buttons in the toolbar are:

Save Workflow	Saves current workflow under specified name
Undo	Reverts the most recent change in the editor
Redo	Reapplies the most recent undone action
Delete Selected Nodes	Deletes the selected Node or Branch
Copy	Copies the selected Node
Paste	Pastes the previously copied or cut Node
Execute Workflow	Schedules the Workflow for execution
Clear Execution Results	Clears highlighting and results of executed nodes
Edit Description	Edit the description of the current workflow
Create Variable	Create and edit workflow-scoped variables

Highlighting

Selected Nodes or Edges will be highlighted in blue. During execution, Nodes that have been scheduled for execution will turn yellow. If they execute successfully, they will turn green; if not, they will turn red.

Apps and Actions

Apps and Actions that you have installed in your Walkoff instance will appear in the left pane. Click the name of an App to reveal the Actions that the App provides (an App is simply a way for us to group associated Actions). To use an Action from the pane, double click on the name of the Action or click and drag the Action into the workspace.

Navigating Workspace

The workspace can be navigated using the buttons in the top left of the pane. From top to bottom, these buttons can be used to pan, zoom to fit, or zoom in and out. You can also click and drag on an empty area to pan, and use the scroll wheel on your mouse to zoom in and out.

Connecting Actions Together

When hovering over the top edge of a node, a dot will appear. To create an edge from one node to the next, click and drag from that dot to the next action in the sequence; an arrow will appear, linking the actions together and creating a branch. A node can point to more than one node; they will all execute unless not chosen by a condition.

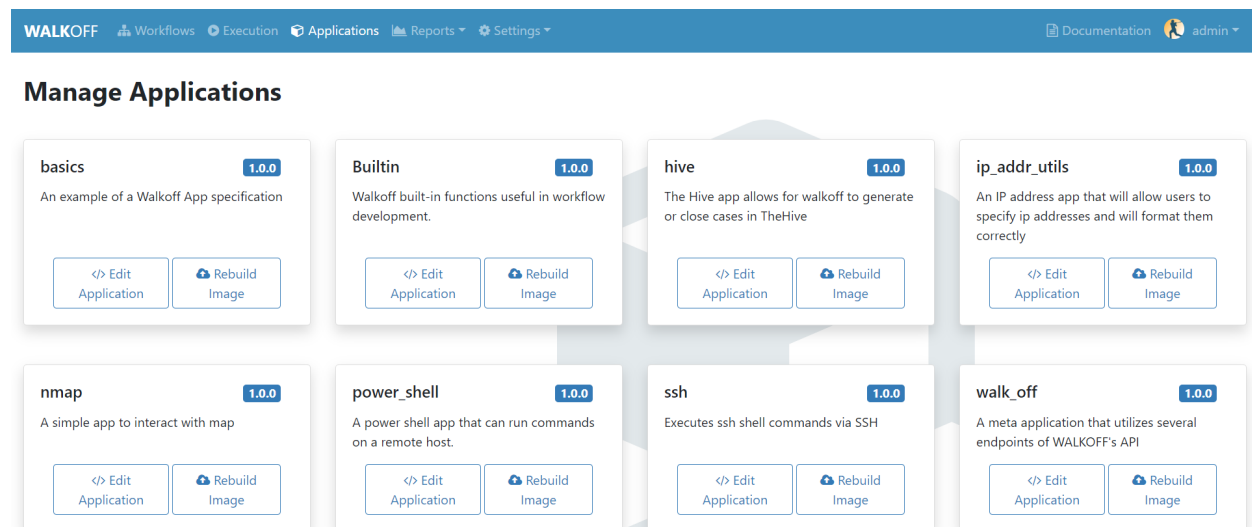
Branch Configuration

When an action is connected to more than one subsequent action, all of those actions will be scheduled. However as actions execute asynchronously, the order in which those actions execute is not guaranteed. If order is important, assign them sequentially.

Editing Actions

When an Action is selected, its properties will appear in the right pane. These include the App it came from, the Action it will perform, and the Name of the Action (separate from its unique ID). You can also set the starting Action of the Workflow in this pane.

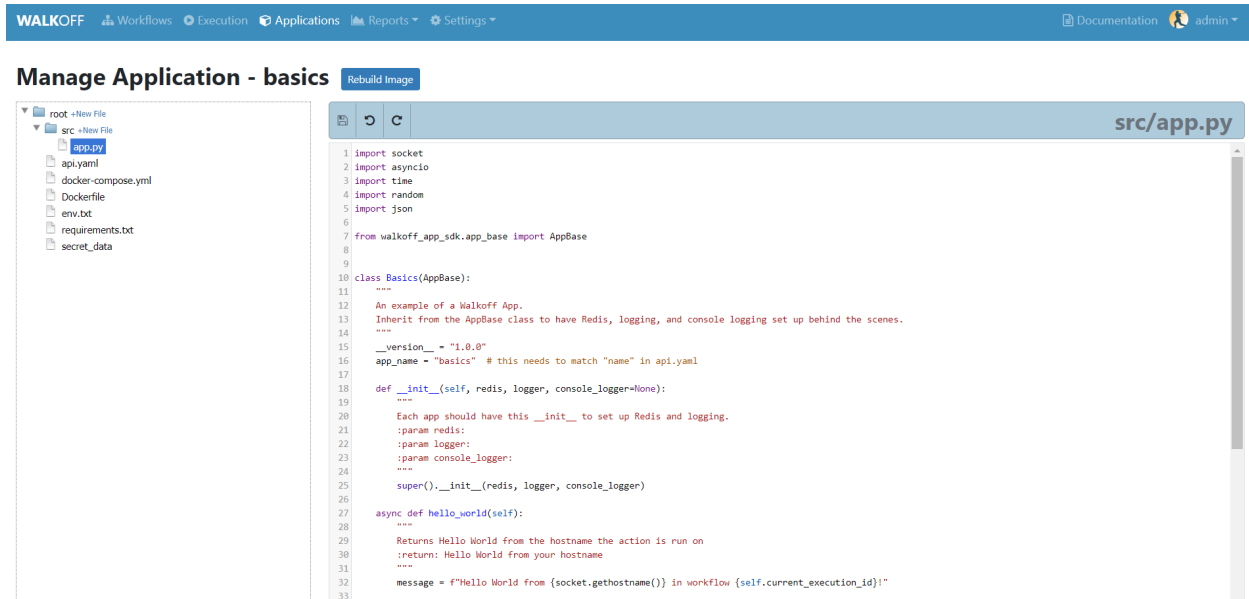
3.2.3 App Editor



The screenshot shows the 'Applications' tab in the Walkoff interface. The top navigation bar includes 'WALKOFF', 'Workflows', 'Execution', 'Applications' (active), 'Reports', and 'Settings'. On the right, there are links for 'Documentation' and a user profile 'admin'. The main content area is titled 'Manage Applications' and displays a grid of application cards. Each card has a title, a version number (1.0.0), a brief description, and two buttons: 'Edit Application' and 'Rebuild Image'.

App Name	Version	Description
basics	1.0.0	An example of a Walkoff App specification
Builtin	1.0.0	Walkoff built-in functions useful in workflow development
hive	1.0.0	The Hive app allows for walkoff to generate or close cases in TheHive
ip_addr_utils	1.0.0	An IP address app that will allow users to specify ip addresses and will format them correctly
nmap	1.0.0	A simple app to interact with map
power_shell	1.0.0	A power shell app that can run commands on a remote host
ssh	1.0.0	Executes ssh shell commands via SSH
walk_off	1.0.0	A meta application that utilizes several endpoints of WALKOFF's API

If you navigate to the Applications tab at the top toolbar of Walkoff you will be directed to a page in which you can manage applications. If you click on the button named Edit Application on the desired app you wish to edit, it will take you to a new page that will contain the file tree on the left side of the page. At this point, you can click app.py and the file contents will be displayed on the right as seen below.



If you want to make edits to any file, you can do that in the file editor that is displayed above. If you would like, you can make changes to multiple files, just make sure to hit the save icon before navigating away from the page. This will save the current file to minio. At this point, minio is prepared to build using the newly saved files. Once you click the “Rebuild Image” button, WALKOFF will pull what is in Minio and rebuild the images based off of those files. Once a successful image is built, Walkoff will copy the files from Minio into your working directory such that those file changes are visible locally.

NOTE: You can undo and redo changes made in the app editor using the symbols next to the save button

3.2.4 User and Role Creation

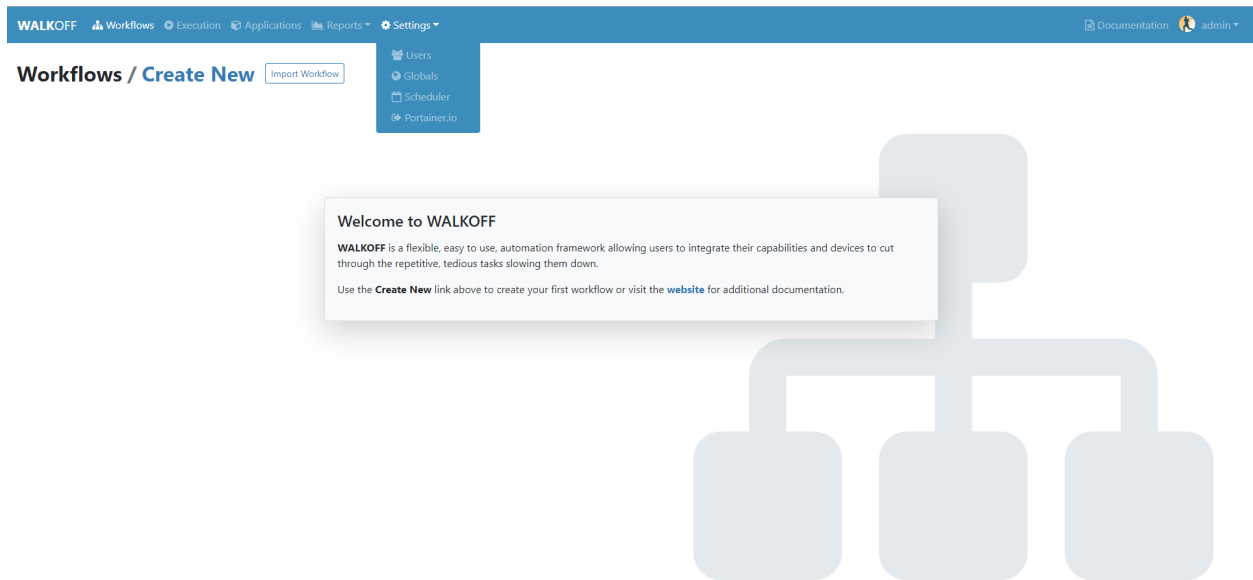
WALKOFF allows for role-based user creation. By default, WALKOFF has four pre-built roles: `super_admin`, `admin`, `work`, and `workflow_developer`.

- The `super_admin` role is a singular immutable account. This role has the ability to delete/create new users and is given full access to all resources created by any user on WALKOFF. Newly created users cannot be given this role; this account cannot be deleted and its role cannot be changed.
- The `admin` role by default is given the same control over WALKOFF as the `super_admin` role. However, newly created users can be given this role, and this role may be edited. Furthermore, other users with differing roles may hide workflows and global variables from this role tier.
- The `workflow_developer` role is given the ability to create, update, and delete workflows, global variables, workflow variables, dashboards, and schedules. This role does not have the ability to change WALKOFF settings, add new users or roles.
- The `workflow_operator` role by default is only given the ability to read all WALKOFF resources aside from workflow variables, which they may update and utilize. They also have the ability to execute workflows that they have access to.

Creating a User

In order to create a new user, you must be logged in as a user with user creation permission (by default, this permission is given to the `super_admin` role).

- First, navigate to the `settings` tab on the WALKOFF toolbar and click on `users`.



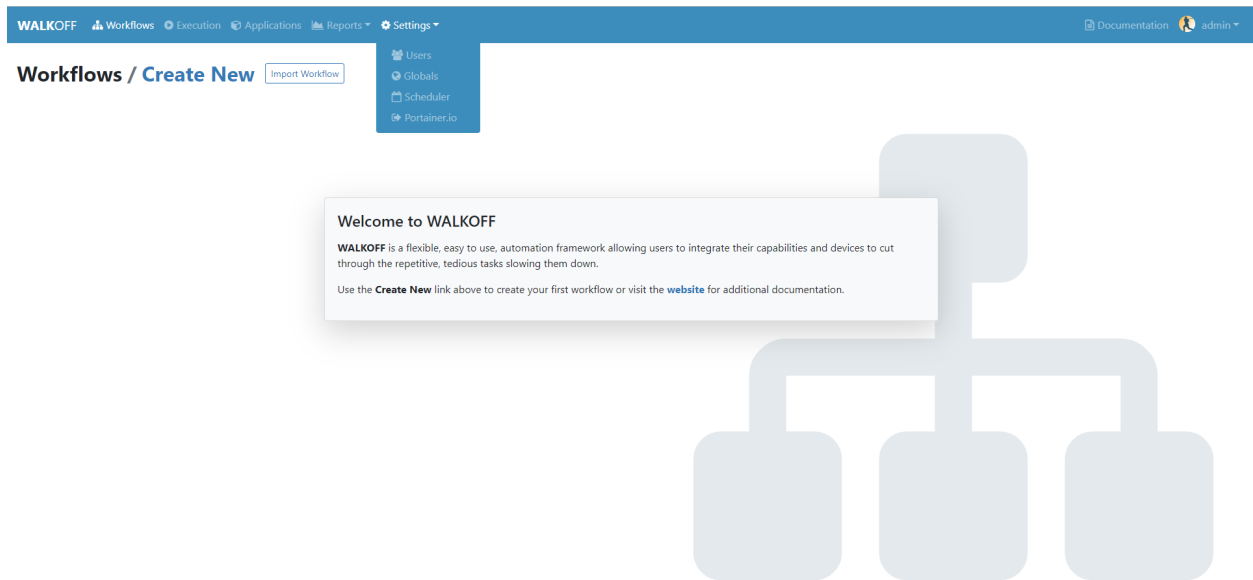
- From this page, click on the `create user` button. Fill in the desired username, password, and role tier for this account and click `Add User` to complete your user creation.

The image shows the 'Add New User' form in the WALKOFF application. The form has a white background with a dark blue border. It contains several input fields: 'Username' with a red asterisk and a value of 'workflow_developer'; 'New Password' with a red asterisk and a masked input field; 'Roles' with a red asterisk and a dropdown menu showing 'workflow_developer'; and 'Confirm New Password' with a red asterisk and a masked input field. There is also an 'Active' toggle switch and two buttons at the bottom: 'Undo Changes and Close' and 'Add User'.

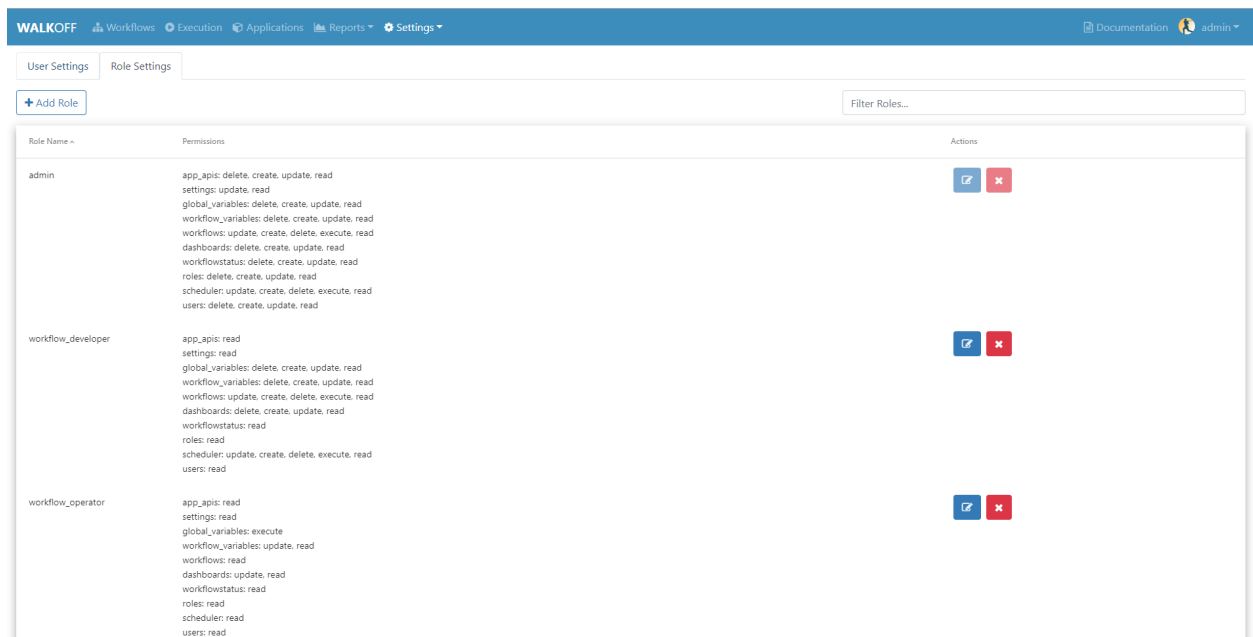
Creating a Role

In order to create a new user, you must be logged in as a user with role creation permission (by default, this permission is given

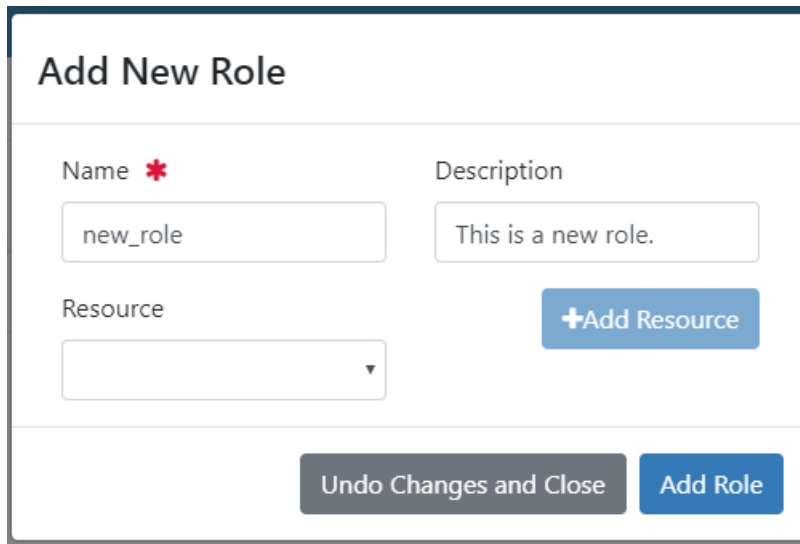
- First, navigate to the `settings` tab on the WALKOFF toolbar and click on `users`.



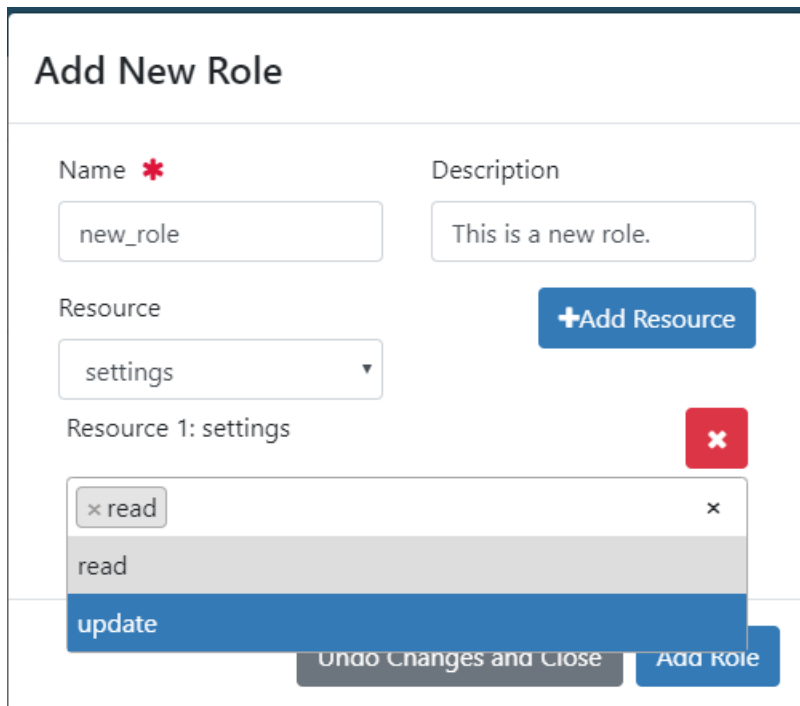
- From this page, click on the `role settings` tab.



- Fill in the desired role name and optional description.



- Select the resource(s) to which you would like to give this role access. After selecting, click `Add Resource`. Then, select which specific permissions you would like to add.



- After resource selection, click `Add Role` to complete your role creation.

3.2.5 Parameter Passing and Node Types (Builtin)

Actions may have parameters, whose properties are defined by the App's `api.yaml`. By default, a parameter's value is a static value that you specify in the workflow editor. To use the output from an Action as the parameter value for another Action, select `Action Output` for the parameter, then select the action you wish to receive the output from. The source action must be a predecessor at some level to the destination action.

Parameter Passing

There are four different types of parameters you can pass to an action: Static Value, Action Output, Global Variable, and Workflow Variable.

Static Value

Static Value parameters are those that are set at workflow creation time. In this case, simply supply the value you would like to use.

Action Output

Using Action Output allows you to use the output from a previous node as an input parameter to a later node. If an action relies on the output of a previous action, this is the appropriate method to use.

Global Variable

As the name implies, a Global Variable can be used by any workflow in Walkoff. These are set at create-global and are encrypted by default for storage and are decrypted at action execution time. These can be useful for storing values like credentials or API tokens for use across multiple workflows.

Workflow Variable

Similar to Global Variables, but scoped only to the current workflow. Create a workflow variable using the Create Variable button found in the [Toolbar](#). These are not encrypted, and are stored within the workflow itself, which means they persist when the workflow is exported.

Node Types

Trigger

Workflow execution can be paused by Triggers. To use one, select the `Builtin` app in the app bar. Double click or drag the `Trigger` action to the workflow space. Connect the Trigger as you would any other action. When execution reaches a trigger, it will pause. To resume execution, send a PATCH to `/api/workflowqueue/<execution_id>` following the format given in the [API Gateway](#) documentation.

Condition

Conditions are used to select one (or no) branch from many to execute based on the output of its parent node. Conditions are written using simple python:

```
if previous_node.result.get('value') == 0:
    selected_node = blue_node
elif previous_node.result.get('value') == 1:
    selected_node = red_node
```

Refer to nodes by their label, replacing spaces with underscores. Access their results by using `<node_label>.result`. You must then assign `selected_node` to the label of one of the nodes directly following the condition.

A graphical editor for this will be added at a later date.

Parallel Action

A special case for Actions, which allows you to run many copies of it in parallel by splitting one of its parameters. For instance, running one copy of an action for each IP in a list of IPs, instead of one action that will iterate over that list. These are denoted by an option called `Run in Parallel` located in the parameters panel at the bottom, which is used to select the parameter to parallelize on. Depending on the implementation of the action, this could result in improvements in execution time.

3.3 Prepackaged Applications

3.3.1 Adversary_Hunting

As the name describes, this app is a group of pre-made actions that run Kansa scripts behind the scenes.

3.3.2 Basics

The Basics app includes simple actions used for testing and as an example/template for app development. There are also some useful actions like `echo_string`/`echo_json` that can be useful for kicking off workflows by overriding parameters

3.3.3 Hive

This is a way for you to interact with the Hive ticketing system. It allows you to create, update, and close cases in the hive as well as other actions you can perform on your instance of the Hive.

3.3.4 IP_Addr_Utils

This is a simple app with an action that converts CIDR notation to a list of IPs. This can be useful for generating lists of IPs to parallelize actions on.

3.3.5 Mitre_Attack

Similar to the Adversary Hunting app, this app contains actions that perform one of scripts from the Mitre Attack framework.

3.3.6 Nmap

This app wraps the Nmap executable. It can run scans and take in the same parameters (hosts and flags) as the regular executable. There are helper functions that can be used for parsing hosts from a host discovery scan, parse scan results for OS fingerprinting, etc.

3.3.7 Power_Shell

This app uses PowerShell Remoting Protocol (PSRP) to execute PowerShell scripts on remote Windows hosts. Code can be provided either as .ps1 scripts in the Docker host or as text input on the workflow. Remote hosts must be configured for PowerShell remoting and the corresponding configuration options must be selected on the app.

3.3.8 SSH

This app creates an SSH session to a remote host for executing commands or transferring files over SFTP. As above, bash or other shell scripts can be placed on the Docker host, or commands can be entered in the workflow. The app can also execute commands locally on the Docker container the app is running in.

3.3.9 Walk_Off

The Walk_Off app provides a demonstration of how to interact with Walkoff's REST API, as well as a more complex example/template for app development.

3.4 Application Development

The minimal directory structure for a WALKOFF application is as follows:

```
WALKOFF
+-- apps
    +-- app_name
        +-- version_number
            |-- Dockerfile
            |-- docker-compose.yml
            |-- api.yml
        +-- src
            +-- your_code.{c, cpp, py, ..., etc.}
            +-- any other files you wish to be accessible in the app container
```

3.4.1 Full Development Instructions

If you would like to follow along by adding a VirusTotal app to your Walkoff instance, follow the **EXAMPLE** bullets at the end of most steps.

Watch this space for an update to the App Editor, which will allow you to create new apps fully from the UI. For now, you can only modify existing apps with it.

1. Write Python Functions in a Standalone Script

- Start by developing your app and its functions in a standalone script outside of WALKOFF – this way you can get basic functionality down before dealing with WALKOFF.
- **Note:** all functions that you expect to turn into actions must be written for asyncio (i.e. `async def function_name()`)
- **EXAMPLE:** Below is example code that can be used to interact with VirusTotal's Api as a standalone script

```
@staticmethod
def _pretty_print(some_dict):
    pretty = json.dumps(some_dict, sort_keys=False, indent=4)
    return pretty

async def ip_lookup(self, ip, apikey):
    url = 'https://www.virustotal.com/vtapi/v2/ip-address/report'
    parameters = {'ip': ip, 'apikey': apikey}
    response = requests.get(url, params=parameters)
    response_dict = response.json()
    pretty = self._pretty_print(response_dict)
    await self.logger.info(pretty)
    return pretty
```

2. Copy the hello_world application folder from the WALKOFF/apps directory

- Rename the copied package to the name of your desired application
- **Note:** The package name must be in `snake_case` and should have the same name as the app you want to create.
- **EXAMPLE:** Make sure you are in the WALKOFF/apps directory before continuing with example below.

```
cp -r hello_world virus_total
```

3. Copy your developed python functions into the app.py file in the 1.0.0/src directory

- Ensure that your new functions are included *under* the HelloWorld class declaration.
- **Note:** Only files under `src` will be copied into the application's Docker container.
- **EXAMPLE:** Delete everything after `def __init__()` but before `if __name__ == "__main__"`, then paste your standalone script into the gap that has been cleared for your code. One line above the pretty print script, add `@staticmethod` to the same column start as `def _pretty_print`. This is because `_pretty_print` is a *helper function* and we won't define it in the `api.yaml` later on.

4. Change the HelloWorld class name in app.py to match the class name of your new app

- Ensure that this class name matches the `asyncio __main__` call at the bottom of `app.py`
- Likewise, also change the `app_name` value to reflect your new application name
- **EXAMPLE:** For this step we will change the name of the HelloWorld class to `VirusTotal`, then below that, change the `"app_name"` value to be `"virus_total"` instead of `"hello_world"`. Finally at the end of the file change `HelloWorld.run()` to be `VirusTotal.run()`. By the end of all of these actions, your `app.py` file should look like this:

```

1  import asyncio
2  import json
3  import requests
4
5  from walkoff_app_sdk.app_base import AppBase
6
7
8  class VirusTotal(AppBase):
9      version = "1.0.0"
10     app_name = "virus_total"
11
12     def __init__(self, redis, logger, console_logger=None):
13         super().__init__(redis, logger, console_logger)
14
15     @staticmethod
16     def _pretty_print(some_dict):
17         pretty = json.dumps(some_dict, sort_keys=False, indent=4)
18         print(pretty)
19         return pretty
20
21     async def ip_lookup(self, ip, apikey):
22         url = 'https://www.virustotal.com/vtapi/v2/ip-address/report'
23         parameters = {'ip': ip, 'apikey': apikey}
24         response = requests.get(url, params=parameters)
25         response_dict = response.json()
26         pretty = self._pretty_print(response_dict)
27         await self.console_logger.info(pretty)
28         return pretty
29
30
31 if __name__ == "__main__":
32     asyncio.run(VirusTotal.run())
33
34

```

5. Change the `api.yaml` metadata file to describe your app and its actions

- For WALKOFF to recognize a function as an action, it must have a corresponding entry in the app's `api.yaml` file
- The action names in this file must exactly match your function names in code.
- You must include at least name, app_version, and actions in this file.
- **EXAMPLE:**

```

walkoff_version: 1.0.0
app_version: 1.0.0
name: virus_total
description: Send api call to Virus Total for various actions.
contact_info:
  name: Walkoff Team
  url: https://github.com/nsacyber/walkoff
  email: walkoff@nsa.gov
license_info:
  name: Creative Commons
  url: https://github.com/nsacyber/WALKOFF/blob/master/LICENSE.md
actions:
  - name: ip_lookup
    description: Look up an IP in VT database
    parameters:
      - name: apikey
        schema:
          type: string
          required: true
          description: enter api key
      - name: ip
        schema:
          type: string
          required: true
          description: enter ip address

```

(continues on next page)

(continued from previous page)

```
returns:
  schema:
    type: string
```

6. Change the `requirements.txt` to match your applications needs

- This file should include any Python package dependencies your app contains
- The Dockerfile will use this to pip install dependencies
- **EXAMPLE:**

```
requests
```

7. Edit `docker-stack-windows.yml` (Windows only)

- At the current time, the WALKOFF “Bootloader” for Windows is rather basic due to time constraints. It will be brought up to the same functionality as the Linux version once time allows.
- **Copy an existing app service definition and change the service name to match your app’s directory name.**
 - **Note:** If you want directories on your host to be available in the container, you can add volume mounts here.
- **EXAMPLE:**

```
services:
  <other service definitions>
  app_virus_total:
    build:
      context: apps/walk_off/1.0.0
      dockerfile: Dockerfile
    configs:
      - common_env.yml
    deploy:
      mode: replicated
      replicas: 0
      restart_policy:
        condition: none
    image: 127.0.0.1:5000/walkoff_app_virus_total:1.0.0
```

Optional: Dockerfile Customization

- This will control how your app will be built.
- See `hello_world’s` Dockerfile for a detailed, step-by-step example on how to create your own Dockerfile
- If your application’s Python dependencies require any OS libraries to build, or if your application requires any OS packages to run, include them in this file.
- You can test building your app with the Dockerfile before running it in WALKOFF:

```
docker build -f apps/app_name/1.0.0/Dockerfile apps/app_name/1.0.0
```

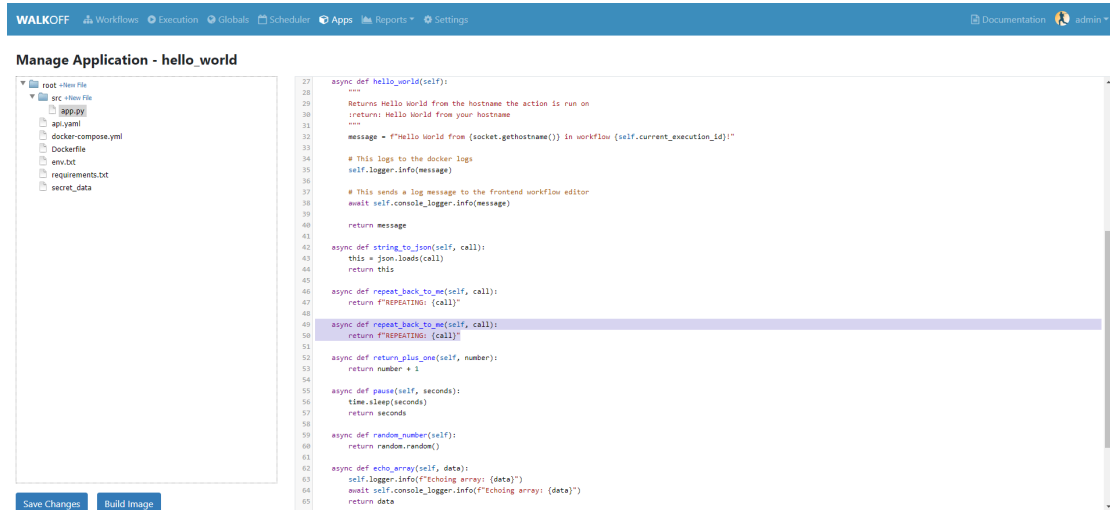
- **EXAMPLE:** We won’t be doing anything here.

Updating Your Application

To edit an existing application, navigate to the App Editor in WALKOFF. Using this UI, you can edit existing apps, add new files, and build app images.

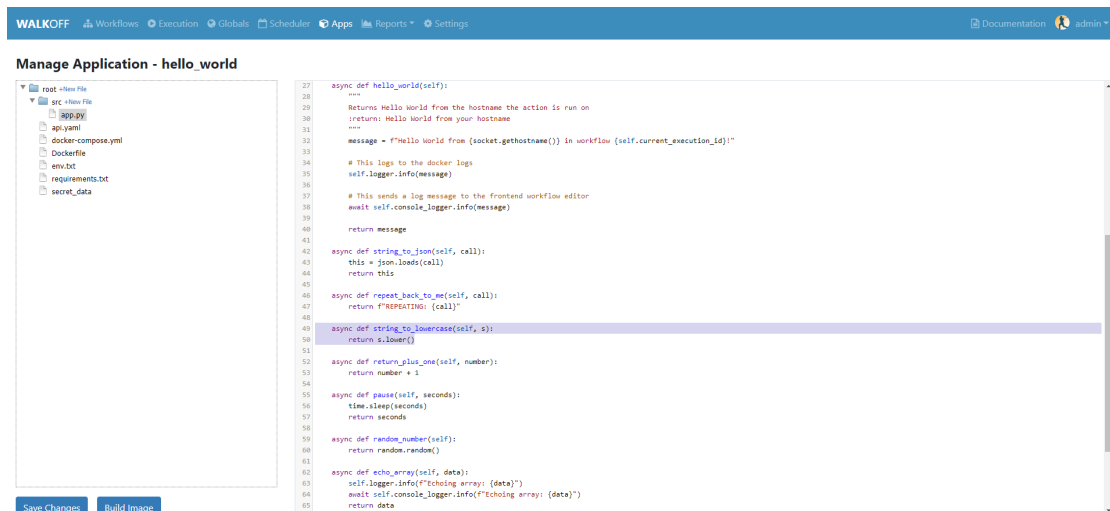
Let's add a new action to the `hello_world` app.

1. Open `src/app.py` and start by copying an existing action

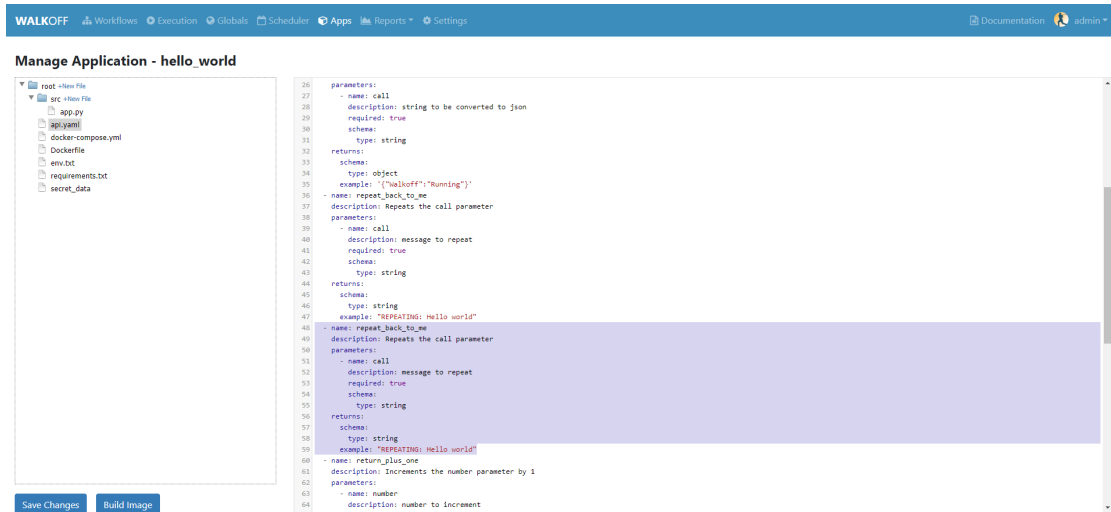


2. Edit the action as desired and save changes

If you need to add binary dependencies, add it to the Dockerfile. If you need to import new Python modules, be sure to add them to `requirements.txt`. If you need to read files, be sure to place them inside `src` to make them usable inside the app container.

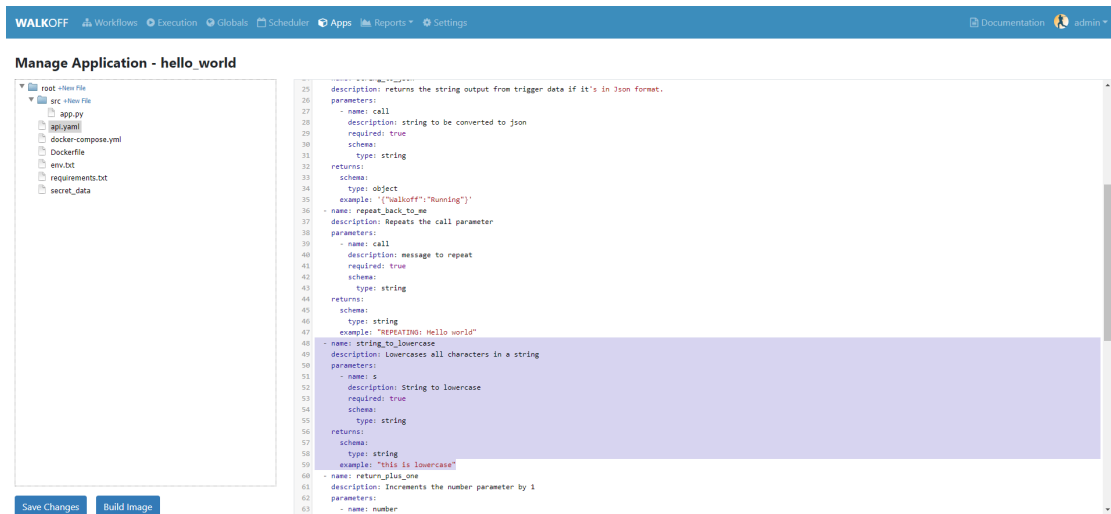


3. Open `api.yaml` and start by copying an existing action



4. Edit the action as desired and save changes

Any time you see the `schema` key, you can use JSON Schema to specify how the parameter or return value should look like. A full schema for what is permissible can be found [here](#).



5. Build the image and watch the build logs

You can watch the Umpire logs to view build status: `docker service logs -f walkoff_core_umpire``

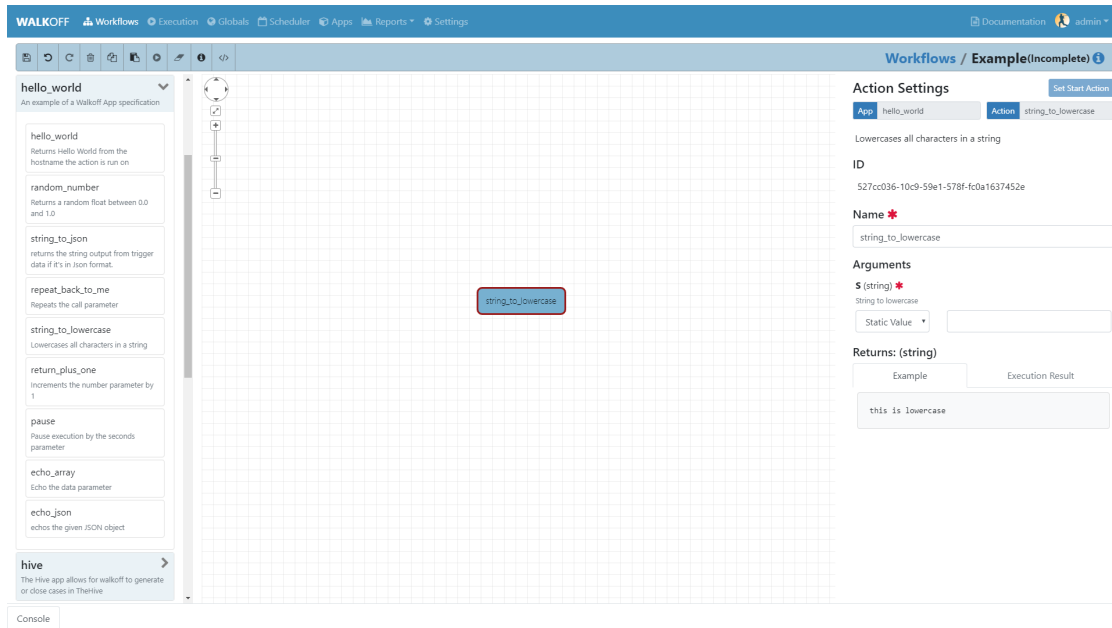
The output will look something like this:


```

Umpire - INFO:Sending image to be built
Umpire - INFO:Docker image building
UMPIRE - DEBUG:Step 1/11 : FROM 127.0.0.1:5000/walkoff_app_sdk as base
UMPIRE - DEBUG:Pulling from walkoff_app_sdk
UMPIRE - DEBUG:Digest:
↳sha256:76c9c7c3d16697d0edd4a3dfffb9591c69cff0c6fdf1ca87e092a0f7cbeee34ab
UMPIRE - DEBUG>Status: Image is up to date for 127.0.0.1:5000/walkoff_app_
↳sdk:latest
UMPIRE - DEBUG:---> 279ce0973000
UMPIRE - DEBUG:Step 2/11 : FROM base as builder
UMPIRE - DEBUG:---> 279ce0973000
UMPIRE - DEBUG:Step 3/11 : RUN mkdir /install
UMPIRE - DEBUG:---> Using cache
UMPIRE - DEBUG:---> dd8c3a031946
UMPIRE - DEBUG:Step 4/11 : WORKDIR /install
UMPIRE - DEBUG:---> Using cache
UMPIRE - DEBUG:---> fe17dbbc4e04
UMPIRE - DEBUG:Step 5/11 : COPY requirements.txt /requirements.txt
UMPIRE - DEBUG:---> Using cache
UMPIRE - DEBUG:---> 31e89590b9d4
UMPIRE - DEBUG:Step 6/11 : RUN pip install --no-warn-script-location --
↳prefix="/install" -r /requirements.txt
UMPIRE - DEBUG:---> Using cache
UMPIRE - DEBUG:---> ca768328609e
UMPIRE - DEBUG:Step 7/11 : FROM base
UMPIRE - DEBUG:---> 279ce0973000
UMPIRE - DEBUG:Step 8/11 : COPY --from=builder /install /usr/local
UMPIRE - DEBUG:---> Using cache
UMPIRE - DEBUG:---> 30fd3018eff0
UMPIRE - DEBUG:Step 9/11 : COPY src /app
UMPIRE - DEBUG:---> cb20500dbc0b
UMPIRE - DEBUG:Step 10/11 : WORKDIR /app
UMPIRE - DEBUG:---> Running in 98e4322863b3
UMPIRE - DEBUG:Removing intermediate container 98e4322863b3
UMPIRE - DEBUG:---> b50497e91a92
UMPIRE - DEBUG:Step 11/11 : CMD python app.py --log-level DEBUG
UMPIRE - DEBUG:---> Running in 1f97d270877f
UMPIRE - DEBUG:Removing intermediate container 1f97d270877f
UMPIRE - DEBUG:---> ae238196c4e6
UMPIRE - DEBUG:Successfully built ae238196c4e6
UMPIRE - DEBUG:Successfully tagged 127.0.0.1:5000/walkoff_app_hello_world:1.
↳0.0
Umpire - INFO:Docker image Built
Umpire - INFO:Pushing image 127.0.0.1:5000/walkoff_app_hello_world:1.0.0.
Umpire - INFO:Pushed image 127.0.0.1:5000/walkoff_app_hello_world:1.0.0.

```

6. Try out your new action in the workflow editor.



Naming and String Requirements:

- App name must be **snake_case** and match in all the following locations:
 1. app directory
 2. app_name in app.py
 3. app_name in api.yaml
 4. service name in docker-compose.yml
- Your action names in api.yaml must match the function names they correspond to in app.py
- If your script is not named app.py, the new name must match the command at the end of your Dockerfile

3.4.2 Troubleshooting

There are several key places to look to debug an application, navigate to Portainer located in the settings tab and go through the setup process if you haven't done so already. This will help you with the following steps for debugging:

1. **Umpire** Following the umpire's logs (`docker service logs -f walkoff_umpire` or clicking on the containers tab in portainer and then clicking on the logs icon next to walkoff_core_umpire) can give you an indication of whether build issues are happening within the stack. Building an app for the very first time can take a long time for example if it contains C dependencies that need to be compiled.
2. **Docker Services** Watching docker services (`watch -n 0.5 docker service ls` or select the services tab on the left panel) can give you an indication of whether your app is running or crashing. At idle with no work, apps and workers scale to 0/N replicas. If you see something repeatedly scaling up and back down to 0, it may be crashing.
3. **Worker Service Logs** Checking the worker service log after the service becomes available for the first time (`docker service logs -f worker` or clicking on the containers tab in portainer and then clicking on the logs icon next to walkoff_core_worker) will allow you to view the worker logs. Generally apps will not cause problems here, but there may be edge cases missing in scheduling apps.

4. **App Service Logs** Checking the app service log after the service becomes available for the first time (`docker service logs -f walkoff_app_app_name` or clicking on the services tab then the associated app followed by `service logs`) will allow you to view the stdout of your app, as well as any exceptions it might be raising.
5. **Console Logging** If you are more familiar with print debugging, you can add information to the console logger by following the code below. This will display the console output in the workflow editor page under the tab Console.

```
message = "This is to be printed to the console logger"
await self.logger.info(message)
```

6. App Containers

- Obtain `app_container_name` from `docker ps`.
- You can `docker exec -it app_container_name /bin/sh` into your app container while it is running to check things like network connectivity, the filesystem, or to run your app manually inside it. (If it is crashing on startup, you will need to fix that first or override its starting command with a sleep instead)

You can also run the app manually outside of docker entirely. Keep in mind while running your app this way, you will have access to your host's filesystem in a way that is not normally accessible to app containers.

1. Install the WALKOFF App SDK (assuming you're starting from WALKOFF's directory)

```
cd app_sdk
pip install -e .
```

2. Add debug flags to the umpire's service definition in `docker-compose.yml`

```
umpire:
  command: python -m umpire.umpire --log-level=debug --disable-app-
↩autoheal --disable-app-autoscale
  image: localhost:5000/umpire:latest
  build:
    context: ./
    dockerfile: umpire/Dockerfile
  networks:
    - walkoff_default
<...>
```

3. Run the rest of WALKOFF via docker-compose as described in the main Readme

```
cd ..
docker stack deploy --compose-file=docker-compose.yml walkoff
```

4. Export environment variables that the app would normally expect inside its container, but change service names to localhost

```
export REDIS_URI=redis://localhost
export REDIS_ACTION_RESULT_CH=action-results
export REDIS_ACTION_RESULTS_GROUP=action-results-group
export APP_NAME=hello_world
export HOSTNAME=$(hostname)
export PYTHONPATH="${PYTHONPATH}:${pwd}"
```

5. Navigate to and run your `app.py`. The app will exit if no work is found, so ensure you run your app just after executing the workflow.

```
python apps/hello_world/1.0.0/src/app.py
```

3.5 API Gateway

3.5.1 API Endpoints

apps

GET /apps

Gets all apps

Status Codes

- 200 OK – Success

GET /apps/apis

Get all app apis

Query Parameters

- **page** (*integer*) – page of data to get

Status Codes

- 200 OK – Success

POST /apps/apis

Create app api

Status Codes

- 200 OK – Success

GET /apps/apis/{app}

Get and app's api

Parameters

- **app** (*string*) – Name OR ID of the app to get

Status Codes

- 200 OK – Success
- 404 Not Found – App does not exist

PUT /apps/apis/{app}

Replace app api

Parameters

- **app** (*string*) – Name OR ID of the app to get

Status Codes

- 200 OK – Success
- 400 Bad Request – AppApi already exists.
- 404 Not Found – AppApi does not exist.

DELETE /apps/apis/{app}

Delete app api

Parameters

- **app** (*string*) – Name OR ID of the app to get

Status Codes

- 204 No Content – Success
- 404 Not Found – AppApi does not exist.

authorization**POST /auth****Login and get access and refresh tokens****Status Codes**

- 201 Created – Success
- 401 Unauthorized – Unauthorized

POST /auth/refresh**Get a fresh access token****Status Codes**

- 200 OK – Success
- 401 Unauthorized – Unauthorized

POST /auth/logout**Logout of walkoff****Status Codes**

- 204 No Content – Success
- 400 Bad Request – Invalid refresh token

configuration**GET /configuration****Reads the configuration****Status Codes**

- 200 OK – Success
- 401 Unauthorized – Unauthorized access

PUT /configuration**Updates the configuration****Status Codes**

- 200 OK – Success
- 401 Unauthorized – Unauthorized access
- 515 – Could not write configuration to file

PATCH /configuration**Updates the configuration****Status Codes**

- 200 OK – Success
- 401 Unauthorized – Unauthorized access
- 515 – Could not write configuration to file

globals

GET /globals

Get all globals

Query Parameters

- **page** (*integer*) – page of data to get
- **to_decrypt** (*string*) – Determine whether or not to decrypt global variable

Status Codes

- 200 OK – Success

POST /globals

Add a global

Status Codes

- 201 Created – Object created
- 400 Bad Request – GlobalVariable already exists

GET /globals/{global_var}

Read a global

Parameters

- **global_var** (*string*) – ID of the global to be fetched

Query Parameters

- **to_decrypt** (*string*) – Determine whether or not to decrypt global variable

Status Codes

- 200 OK – Success
- 404 Not Found – GlobalVariable does not exist.

PUT /globals/{global_var}

Update a global

Parameters

- **global_var** (*string*) – ID of the global to be fetched

Status Codes

- 200 OK – Success
- 404 Not Found – GlobalVariable does not exist

DELETE /globals/{global_var}

Remove a global

Parameters

- **global_var** (*string*) – ID of the global to be fetched

Status Codes

- 204 No Content – Success
- 404 Not Found – GlobalVariable does not exist

GET /globals/templates

Get all global templates

Query Parameters

- **page** (*integer*) – page of data to get

Status Codes

- 200 OK – Success

POST /globals/templates

Add a global

Status Codes

- 201 Created – Object created
- 400 Bad Request – GlobalVariable already exists

GET /globals/templates/{global_template}

Read a global template

Parameters

- **global_template** (*string*) – ID of the global template to be fetched

Status Codes

- 200 OK – Success
- 404 Not Found – GlobalVariableTemplate does not exist.

PUT /globals/templates/{global_template}

Update a global template

Parameters

- **global_template** (*string*) – ID of the global template to be fetched

Status Codes

- 200 OK – Success
- 404 Not Found – GlobalVariableTemplate does not exist

DELETE /globals/templates/{global_template}

Remove a global

Parameters

- **global_template** (*string*) – ID of the global template to be fetched

Status Codes

- 204 No Content – Success
- 404 Not Found – GlobalVariableTemplate does not exist

roles

GET /roles

Read all roles

Status Codes

- 200 OK – Success

POST /roles

Create a role

Status Codes

- 201 Created – Object created.
- 400 Bad Request – Object exists.

GET /roles/{role_id}

Read a role

Parameters

- **role_id**(*string*) – The name that needs to be fetched.

Status Codes

- 200 OK – Success
- 404 Not Found – Object does not exist.

PUT /roles/{role_id}

Update a role

Parameters

- **role_id**(*string*) – The name that needs to be fetched.

Status Codes

- 200 OK – Success
- 404 Not Found – Object does not exist.

DELETE /roles/{role_id}

Delete a role

Parameters

- **role_id**(*string*) – The name that needs to be fetched.

Status Codes

- 204 No Content – Success
- 404 Not Found – Object does not exist.

GET /availableresourceactions

Read all available resource actions

Status Codes

- 200 OK – Success

scheduler

GET /scheduler

Get the current scheduler status.

Status Codes

- 200 OK – Success

PUT /scheduler

Update the status of the scheduler

Status Codes

- 200 OK – Success

GET /scheduledtasks

Get all the scheduled tasks

Status Codes

- 200 OK – Success

POST /scheduledtasks

Create a new Scheduled Task

Status Codes

- 201 Created – Success
- 400 Bad Request – Scheduled task already exists

GET /scheduledtasks/{scheduled_task_id}

Get the scheduled task

Parameters

- **scheduled_task_id** (*string*) – The ID of the scheduled task.

Status Codes

- 200 OK – Success
- 404 Not Found – Scheduled task does not exist

PUT /scheduledtasks/{scheduled_task_id}

Update a new Scheduled Task

Parameters

- **scheduled_task_id** (*string*) – The ID of the scheduled task.

Status Codes

- 200 OK – Success
- 400 Bad Request – Scheduled task name already exists
- 404 Not Found – Scheduled task does not exist

DELETE /scheduledtasks/{scheduled_task_id}

Delete the scheduled task

Parameters

- **scheduled_task_id** (*string*) – The ID of the scheduled task.

Status Codes

- 204 No Content – Success
- 404 Not Found – Scheduled task does not exist

users

GET /users
Read all users

Status Codes

- 200 OK – Success

POST /users
Create a user

Status Codes

- 201 Created – User created.
- 400 Bad Request – Could not create user <username>. User already exists.

GET /users/{user_id}
Get a user

Parameters

- **user_id** (*integer*) – The id of the user to be fetched

Status Codes

- 200 OK – Success
- 404 Not Found – Could not display user <username>. User does not exist.

PUT /users/{user_id}
Update a user

Parameters

- **user_id** (*integer*) – The id of the user to be fetched

Status Codes

- 200 OK – Success
- 400 Bad Request – Invalid password
- 404 Not Found – Could not edit user <username>. User does not exist.

DELETE /users/{user_id}
Delete a user

Parameters

- **user_id** (*integer*) – The id of the user to be fetched

Status Codes

- 204 No Content – Success
- 401 Unauthorized – Could not delete user <username>. User is current user.
- 404 Not Found – Could not delete user <username>. User does not exist.

workflows

GET /workflows

Read all workflows in playbook

Query Parameters

- **page** (*integer*) – page of data to get

Status Codes

- **200 OK** – Success
- **404 Not Found** – No workflows exist.

POST /workflows

Create a workflow

Query Parameters

- **source** (*string*) – The ID of the workflow to clone

Status Codes

- **201 Created** – Workflow created.
- **400 Bad Request** – Workflow already exists.

GET /workflows/{workflow}

Read a workflow

Parameters

- **workflow** (*string*) – The name or ID of the workflow to get.

Query Parameters

- **mode** (*string*) – Set to export to send as file.

Status Codes

- **200 OK** – Success
- **404 Not Found** – Workflow does not exist.

PUT /workflows/{workflow}

Update a workflow

Parameters

- **workflow** (*string*) – The name or ID of the workflow to get.

Status Codes

- **200 OK** – Success
- **400 Bad Request** – Workflow already exists.
- **404 Not Found** – Workflow does not exist.

DELETE /workflows/{workflow}

Delete a workflow

Parameters

- **workflow** (*string*) – The name or ID of the workflow to get.

Status Codes

- 204 No Content – Success
- 404 Not Found – Workflow does not exist.

workflowqueue

GET /workflowqueue

Get status information on the workflows currently executing

Query Parameters

- **page** (*integer*) – page of data to get

Status Codes

- 200 OK – Success

POST /workflowqueue

Execute a workflow

Status Codes

- 202 Accepted – Success asynchronous.
- 400 Bad Request – Invalid input error.
- 404 Not Found – Workflow does not exist.

GET /workflowqueue/{execution}

Get status information on a currently executing workflow

Parameters

- **execution** (*string*) – The ID of the execution to get.

Status Codes

- 200 OK – Success
- 404 Not Found – Object does not exist.

PATCH /workflowqueue/{execution}

Abort or trigger a workflow

Parameters

- **execution** (*string*) – The ID of the execution to get.

Status Codes

- 204 No Content – Success.
- 400 Bad Request – Invalid input error.
- 404 Not Found – Workflow does not exist.

DELETE /workflowqueue/cleardb

Removes workflow statuses from the execution database. It will delete all of them or ones older than a certain number of days

Query Parameters

- **all** (*boolean*) – Whether or not to delete all workflow statuses, defaults to false
- **days** (*integer*) – The number of days of workflow statuses to keep

Status Codes

- 204 No Content – Success

dashboards

GET /dashboards

Read all dashboards

Retrieves all dashboards currently stored in the database.

Query Parameters

- **page** (*integer*) – page of data to get

Status Codes

- 200 OK – Success
- 404 Not Found – No dashboards exist.

POST /dashboards

Create a dashboard

Creates a dashboard from the JSON in request body

Status Codes

- 201 Created – Workflow created.
- 400 Bad Request – Workflow already exists.

PUT /dashboards

Update a dashboard

Updates a whole dashboard using the JSON request body

Status Codes

- 200 OK – Success
- 404 Not Found – Dashboard does not exist.

GET /dashboards/{dashboard}

Get a dashboard by id

Retrieve a single dashboard from database by ID.

Parameters

- **dashboard** (*string*) – ID of the global to be fetched

Status Codes

- 200 OK – Success
- 404 Not Found – No dashboard with that ID exist.

DELETE /dashboards/{dashboard}

Delete a dashboard

Deletes a dashboard by ID

Parameters

- **dashboard** (*string*) – ID of the global to be fetched

Status Codes

- 201 Created – Workflow updated.

- 404 Not Found – Dashboard does not exist.

3.5.2 JSON Formulation

3.6 Changelog

3.6.1 [1.0.0-beta.1]

This update introduces a number of new features, including an App Editor in the UI, more granular role-based permissions, and a “bootloader” for automating deployment of WALKOFF.

Added

- App Editor for editing app files and building Docker images from said apps. You can use this to change apps while WALKOFF is running, without restarting the whole stack.
- More granular RBAC where permissions on individual workflows and global variables can be restricted to specific roles.
- Execution results display on individual nodes to aid in identifying results.
- Bootloader container to automate deployment and teardown of the WALKOFF stack.
- Autogenerated walkoff_client Python package for interfacing with WALKOFF API. Work in progress.
- JSON editor GUI for editing arrays and objects in action parameters.
- JSON editor GUI for editing Workflow and Global Variables

Changed

- Dashboards renamed to Reports. Lots of work still to do here.
- Condition and Transform exceptions now get passed up into workflow results for easier debugging.
- Common config location to minimize number of locations where ports, service names, etc. need to change when configuring WALKOFF.
- WALKOFF now runs only on Docker Swarm, no longer with plain Docker Compose.
- All services now follow a walkoff_core, walkoff_app, or walkoff_resource naming scheme to disambiguate services from other stacks.
- Exceptions in Apps are now propagated correctly to action results.
- All WALKOFF services that need to be exposed now route through NGINX.

Security

- WALKOFF now uses HTTPS behind NGINX using a self-signed certificate.

Fixed

- Worker's workflow_types are now tagged with a _walkoff_type to prevent ambiguity with user provided data
- SSH app updated to allow more conventional use of wildcards, relative/absolute paths, etc.
- Conditions no longer cause entire subtrees to be cancelled, only the immediate successors.
- walkoff_default Docker network is attachable by default for external services to attach to us.
- Endpoints for PUT now correctly use resource IDs in path parameters.
- Importing a workflow with the same name as an existing one should no longer overwrite the original.
- Umpire scaling heartbeat slowed down to reduce race conditions - will be replaced with on-demand, resource aware scaling in future.
- Queued actions are correctly cleaned up when aborting a workflow.
- Enforced startup order of all the services to avoid busywaiting when services aren't up.
- Reduced intensity of "Server not responding" pop-up.
- Database commit issues relating to Workflow errors resolved.
- Parameters are correctly passed through to node status messages on the frontend.
- Actions now enforce results being JSON serializable, preventing issues with serializing Python objects.
- Action console logger reconnected to frontend.

3.6.2 [1.0.0-alpha.2]

This update includes numerous bugfixes and a number of reintroduced features.

Added

- Trigger nodes allow you to pause workflow execution until webhook for the trigger is hit with data
- Basic Condition nodes allow you to perform branching execution in a more flowchart-like manner
- Basic Transform nodes allow you to write code snippets to transform/remap/select action results on the fly (UI support pending)
- Parallel Action node types in the workflow editor allow you to parallelize actions on a specified parameter
- Display UUIDs for workflow and workflow nodes in UI
- Portainer container creates UI for docker management

Changed

- WALKOFF now runs utilizing stack deploy, allowing for the use of external Docker secrets
- App version no longer required in app_name in api.yaml
- CRUD endpoints now accept resource names as keys when applicable
- Globals can now be arbitrary JSON (UI support pending)
- Builtins build location moved to Umpire and is only built once on startup

Removed

Security

- Implemented AES-256 encryption/decryption for Global Variables. Exclusive-access decryption based on account level standing still needs to be implemented in the future. Currently, any GET request to the API gateway will return a decrypted Global Variable, regardless of account.

Fixed

- Workflow import/export
- Workflow validation (still needs work); workflows can be saved in an incomplete state again
- Validate workflow name uniqueness when creating workflows
- Testing suite (still needs expansion)
- Uniqueness constraints on CRUD operations
- Dereferencing Global and Workflow variables in workflows
- Ability to override starting parameters in a workflow execution
- Ability to update/delete encrypted Global Variables
- Hide global values by default on Globals tab
- Default boolean parameters to false
- Copying and pasting of nodes in workflow editor
- Accessing action results before conditionals in parameters that follow it

3.6.3 [1.0.0-alpha.1]

This update includes a near-complete rewrite of the workflow execution logic, and a considerable refactor of the server in preparation for a future move to an asynchronous framework. The following changes are not exhaustive.

Added

- “Umpire” added, which handles building and replication of Worker and App containers.

Changed

- Docker Compose is required. Python 3.7 is required if running components locally (primarily for development).
- Execution logic completely rewritten to support containerized architecture from the ground up.
- Apps now live in their own containers, separate from workers.
- Apps should now be (internally) stateless
- Kubernetes support has been removed in favor of using Docker Swarm API.
- Playbooks removed, Workflows can now be grouped by tags instead.
- unittest has been replaced with pytest
- Redis is now the primary communication channel between components (removing ZMQ and Kafka).

- SQLite database should no longer be used if running locally for development.
- Workflow Execution page has been overhauled aesthetically.

Removed

- Triggers have been temporarily removed, but are targeted for a near-future 1.1 release.

3.6.4 [0.9.4]

Added

- Added ability to view WALKOFF Server API locally via `/api/docs` with the server running.

Fixed

- Execution DB now gets properly closed when WALKOFF exits. Fixes issues with docker-compose stop/start.
- Triggers on unbound actions (apps without devices) fixed.
- Add Docker image and compose file based on development branch.
- Upgraded WALKOFF Server API from swagger2 to openapi3, which includes improved security, and better request validation.
- Upgraded Python marshmallow library version, which includes stricter validation.
- Please note: because some dependency library versions were changed in the `requirements.txt` file, users must run the command `pip install --upgrade -r requirements.txt` to make sure all dependencies are met. This is also good practice to do after every new release.

3.6.5 [0.9.3]

This is a minor release to fix missing front-end resources. A number of documentation changes have also been made, particularly regarding installing WALKOFF on Windows, as running WALKOFF directly on Windows has no longer supported since 0.9.0.

Fixed

- References to running WALKOFF directly on Windows now emphasize lack of support.
- Front-end dependencies have been added to the repository.

3.6.6 [0.9.2]

This is a minor release primarily to ease installation of WALKOFF.

Added

- README.md contains further documentation on running WALKOFF locally, in Docker, or in Kubernetes
- NodeJS and NPM are no longer required, as the front-end components are now prepackaged in the main repository.

3.6.7 [0.9.1]

Added

- README.md now contains more detailed instructions on using WALKOFF with Docker, as well as a docker-compose file
- All databases will now be stamped with the most up-to-date alembic version, so WALKOFF will not run if you are using an out-of-date database (see Fixed section for more details)

Fixed

- When using Redis as an external accumulator, results are now pickled to preserve typing. This fixes the issue where everything (list, int, etc.) was incorrectly being returned as strings
- Fixed walkoffctl update script to correctly update databases – run `python -m walkoff local update` to update
- ActionResult objects are now pretty-printed correctly in the console and log files
- Python Redis library is now pinned in requirements.txt due to breaking changes
- Fixed certificate generation for Kubernetes certificates

Removed

- Update.py script was removed and replaced with walkoffctl update (see Fixed section for more details)

3.6.8 [0.9.0]

Please Note: From version 0.9.0 forwards, WALKOFF requires a Redis cache to operate. You can run Redis natively on most Linux distributions (see the Redis quickstart guide: <https://redis.io/topics/quickstart> or search for a package in your OS's package manager). On Windows, you will need to use Docker to run Redis in a container or expose Redis from a VM.

Added

- Support for running WALKOFF in a Kubernetes cluster using docker images and helm
- Command line interface (walkoffctl) for managing WALKOFF installations, both locally and on Kubernetes
- More comprehensive logging with Prometheus, fluentd, and flask
- Ability to run apps in separate containers for better scalability (still in-progress)
 - Support for templating Docker files for apps and runtimes
- Began introducing PyTest as a more maintainable alternative to unittest
- Support for tracking which User executed which Workflow
- Now have the option of choosing ZMQ sockets or Kafka message queues to communicate with executing Workflows

Changed

- Moved logic of executing a Workflow into Workflow execution context objects, which allows for more flexibility
- Upgraded to bootstrap4 and Angular 6.1.7

Removed

- Removed support for DiskCache – a Redis cache must be installed to run WALKOFF
- The Case database, as it was unnecessary on top of the comprehensive logging done by WALKOFF

Fixed

- Fixed the foreign key constraints in both databases (they were not being enforced previously)
- Connexion library version updated in requirements.txt to fix access_token and 404 error
- Minor aesthetic improvements to front-end

3.6.9 [0.8.5]

Fixed

- Fixed a bug caused by a new version of the connexion library which made the OpenAPI specification invalid

3.6.10 [0.8.4]

Added

- Workflows now support environment variables. These are top-level arguments to a workflow. These are then exposed on the execution page allowing users to modify the most important variables in a workflow without modifying the workflow itself.
- Added a health check endpoint at the /health endpoint

Changed

- The Metrics page now defaults to showing workflow metrics instead of app metrics

Fixed

- Action results and workflow results stream now filter for the currently-executing workflow. This eliminates many issues experienced by multiple users executing workflows concurrently from the workflow editor
- Fixed an error which caused the Scheduler to not execute workflows
- Fixed another bug in the scheduler in which the scheduled workflows would not persist across server restarts
- A bug where messages couldn't be sent
- A bug where modifying more than one device at a time on the playbook editor would cause the workflow to be invalidated

- Some database configuration bugs when used with non-SQLite databases
- Fixed a bug which wouldn't allow a user to abort a workflow if it was pending execution.

3.6.11 [0.8.3]

Added

- CSV to Array action in the Utilities app

Changed

- The action results SSE stream truncates the result using the `MAX_STREAM_RESULTS_SIZE_KB` config option

Fixed

- Bytes conversion bug in the RedisCacheAdapter
- Bug in playbook editor using users and roles as arguments
- Bug where some callbacks weren't getting registered
- Column width bug in playbook editor, execution, and metrics pages
- OpenAPI validation bug with newest version of the swagger validator

3.6.12 [0.8.2]

Added

- Arguments can now reference branches. This will resolve to the number of times that branch has been executed.
- Log messages are more comprehensive and useful.
- More error checking on the worker processes to harden them.

Fixed

- Bug where databases couldn't be used with a password.
- Bug where app instances would receive an Argument rather than the necessary integer ID.
- Compatibility issue with pip 10 and the `install_dependencies.py` script.
- Bug in validation of execution elements where, once an error was found it wouldn't be removed.
- Fixed bug where exporting playbooks with Python 3 would cause an error.
- Bug where argument ids were not stripped on exporting of playbooks, causing errors when importing them onto a different instance of Walkoff.

3.6.13 [0.8.1]

Fixed

- Bug where Workflows with unbounded Actions were unable to be executed

3.6.14 [0.8.0]

Added

- Multiple tools have been added to help develop workflows
 - Playbooks can be saved even if they are invalid. However, playbooks cannot be executed if they are invalid.
 - The playbook editor displays the errors on a workflow which must be solved before the workflow can be executed
 - You can now use Python's builtin `logging` module in an app, and the log messages will be displayed in the playbook editor
- The metrics page has been introduced in the UI which displays simple metrics related to the execution of workflows and actions.
- The devices used in the actions in workflows are now objects, enabling dynamic selection of the device used for the action. To further support this, an action in the Utilities app named `get devices by fields` allows you to query the devices database.
- The ability to use a key-value storage has been created. This is now the mechanism used to push workflows and backs the SSE streams. Currently two options are available for key-value store, `DiskCache`, a SQLite-backed key-value storage, and `Redis`. By default Walkoff will use `DiskCache`, but it is recommended that users configure and use `Redis`.
- The SSEs now use dedicated `SseStream` objects which are backed by the cache. These objects make constructing and using streams much easier. `walkoff.sse.InterfaceSseStream` and `walkoff.sse.FilteredInterfaceSseStream` objects have been made available to use in custom interfaces.
- A `CaseLogger` object which makes it much easier to log events to the case database has been created.

Changed

- The `interfaces.AppBlueprint` used to construct interfaces has been modified to extend from `walkoff.sse.StreamableBlueprint` which in turn extends Flask's `Blueprint`. This makes the interface cleaner and more flexible.
- Changes to the REST API
 - In the configuration resource:
 - * `workflow_path`, `logging_config_file`, and `zmq_requests` have been removed from the API
 - * The ability to edit the cache configuration has been added
 - In the playbook resources:
 - * All execution elements have a read only list of human-readable errors
 - * A workflow has a read only Boolean field `"is_valid"` which indicates if any of its execution elements have errors

- All changes to the configuration will only be applied on server restart
- Refactorings have been done to minimize the amount of global state used throughout Walkoff. Work will continue on this effort.
- Metrics are now stored in the execution database
- Changes to styling on the playbook editor

Deprecated

- `walkoff.helpers.create_sse_event` has been deprecated and will be removed in version 0.10.0. Use `walkoff.sse.SseEvent` or the streams in `walkoff.sse` instead. ### Fixed
- Bug where branches where all branches weren't being evaluated in a workflow
- Bug where object arguments could not be converted from strings

Contributor

- Testing the backend now requires the additional the dependencies in `requirements-test.txt`
- The minimum accepted unit test coverage for the Python backend is now 88%

3.6.15 [0.7.4]

Fixed

- Bug where some device fields were being deleted on update

3.6.16 [0.7.3]

Fixed

- Bug where NO_CONTENT return codes were failing on Werkzeug WSGI 0.14

Changed

- All node modules are now bundled into webpack

3.6.17 [0.7.2]

Fixed

- An unintentional backward-breaking change was made to the format of the dictionary used in the interface dispatcher which sometimes resulted in a dict with a "data" field inside a "data" field. This has been fixed.

3.6.18 [0.7.1]

Changed

- Improved deserialization in the user interface
- Empty arrays are omitted from returned execution element JSON structure in the REST API.

Fixed

- `PATCH /api/devices` now doesn't validate that all the fields of the device are provided.
- Fixed dependency bug on `GoogleProtocolBuffer` version

3.6.19 [0.7.0]

Added

- An execution control page is now available on the user interface. This page allows you to start, pause, resume, and abort workflows as well as displays the status of all running and pending workflows.
 - With this feature is a new resource named `workflowqueue` which is available through the `/api/workflowqueue` endpoints.
- You now have the ability to use a full set of Boolean logic on conditions. This means that on branches and triggers you can specify a list of conditions which must all be true (AND operator), or a list of conditions of which any must be true (OR operator), or a list of conditions of which exactly one must be true (XOR operator). You can also negate conditions or have child conditions. This new conditional structure is called a `ConditionalExpression` and wraps the old `Condition` objects.
- Playbooks can be exported to and imported from a JSON text file using the new `GET /api/playbooks?mode=export` and the `POST /api/playbooks` using a `multipart/form-data` body respectively.

Changed

- Significant changes to the REST API
 - We have changed the HTTP verbs used for the REST API to reflect their more widely-accepted RESTful usage. Specifically, the `POST` and `PUT` verbs have been swapped for most of the endpoints.
 - Workflows are now accessed through the new `/api/workflows` endpoints rather than the `/api/playbooks` endpoints
 - The `/api/playbooks` and the `/api/workflows` endpoints now use the `UUID` instead of the name.
 - The `/api/playbook/{id}/copy` and the `/api/playbooks/{id}/workflows/{id}/copy` endpoints are now accessed through `POST /api/playbooks?source={id_to_copy}` and the `POST /api/workflows?source={id_to_copy}` endpoints respectively.
 - Server-Sent Event streams are now located in the `/api/streams` endpoints
 - Errors are now returned using the RFC 7807 Problem Details standard
- Playbooks, workflows, and their associated execution elements are now stored in the database which formerly only held the devices. The both greatly increased scalability as well as simplified the interactions between the server and the worker processes as well as increased scalability.

- Paused workflows and workflows awaiting trigger data are now pickled (serialized to binary) and stored in a database table. Before, a conditional wait -was used to pause the execution of a workflow. By storing the state to the database, all threads on all worker processes are free to execute workflows.
- Information about the workflow which sent events are now available in both the Google Protocol Buffer messages as well as the arguments to callbacks using the interface event dispatcher.
- All times are stored in UTC time and represented in RFC 3339 format
- The marshmallow object serialization library is now used to serialize and deserialize execution elements instead of our old homemade solution

Deprecated

- The “sender_uids” argument in the interface dispatcher `on_xyz_event` decorators is now an alias for “sender_ids”. **This will be removed in version 0.9.0**

Removed

- The `/api/playbooks/{name}/workflows/{name}/save` endpoint has been removed.
- The `/api/playbooks/{name}/workflows/{name}/{execute/pause/resume}` endpoints have been removed. Use the `/api/workflowqueue` resource instead
- Removed `workflow_version` from the playbooks. This may be added later to provide backwards-compatible import functionality to the workflows.
- `/api/devices/import` and `/api/devices/export` endpoints have been removed. Use the new POST `/api/devices` with `multipart/form-data` and GET `/api/devices?mode=export` endpoints respectively.

Contributor

- The minimum accepted unit test coverage for the Python backend is now 86%

3.6.20 [0.6.7]

Fixed

- Fixed bug in `create_sse_event` where data field of the SSE would not be populated if no data was not specified, causing the SSE event to be invalid

3.6.21 [0.6.6]

Changed

- Omitting `sender_uids` or names on `dispatcher.on_xyz_event` decorators in interfaces now registers the decorated function for all senders. This is consistent with the previously inaccurate code examples in the tutorials.

3.6.22 [0.6.5]

Added

- Webpack is now used to increase UI performance

Changed

- Default return codes for the Walkoff app

Contributor

- Some UI tests are now run on Travis-CI

3.6.23 [0.6.4]

Changed

- The accept/decline method returns status codes indicating if the action was accepted or declined instead of true/false

Fixed

- Fixed a bug where roles weren't being deleted from the database
- Fixed issue preventing permissions to be removed on editing roles
- Fixed issue with messages not properly being marked as responded

3.6.24 [0.6.3]

Added

- Added a simple action in the Utilities app named "request user approval" which sends a message with some text to a user and has an accept/decline component.

Changed

- Refactoring of AppCache to use multiple objects. We had been storing it as a large dict which was becoming difficult to reason about. This is the first step of a larger planned refactoring of how apps are cached and validated

Fixed

- Bug on UI when arguments using an array type without item types specified
- Fixed issue with workflow migration caused to erroneously deleting a script

3.6.25 [0.6.2]

Multithreaded workers for increased asynchronous workflow execution

Added

- Multiple workflows can be executed on each worker process
- Decorator factory to simplify endpoint logic
- Endpoint to get system stats

Fixed

- Bug where roles couldn't be assigned to a user on creation

Contributor

- Added AppVeyor to test Walkoff on Windows

3.6.26 [0.6.1]

Added

- Multiple workflows can be executed on each worker process

Changed

- Bumped dependency of `flask-jwt-extended` to version 3.4.0

Fixed

- Default logging config issue
- Removed `walkoff/client/build` which was accidentally version controlled
- CodeClimate misconfiguration
- Bug fixes to messaging caused by messaging callback not being registered in the server

3.6.27 [0.6.0]

Introducing roles, messages, and notifications

Added

- Administrators can now create custom roles and assign users to those roles. Each resource of the server endpoint is protected by a permission, and roles can be created which combine resource permissions.
- Messages and notifications
 - Actions can now send messages to users
 - Messages can be used to convey information to users or to pause a workflow and wait for a user to approve its continued execution
 - When a user receives a message, a notification will appear
- Easy updates
 - An update script is provided to update to the most recent version if one is available. This script includes custom workflow migration scripts and database migration scripts generated by SQLAlchemy-Alembic. These are a work in progress.
 - * *Note 1: Database migrations only work for default database locations and using SQLite. This can be changed in the “alembic.ini” file*
 - * *Note 2: Now that databases and workflows can be updated easily, minor version updates will not occur on backward-breaking changes to the database schema or the playbook schema.*
 - This script also includes utility functions for backing up the WALKOFF directory, cleaning pycache, setting up WALKOFF after an update, etc.
- Explicit failure return codes for actions
 - Return codes which indicate a failure of the action can be marked with `failure: true`. This will cause an `ActionExecutionError` event to be sent
- Explicit success default return codes for actions
 - The default return code for an action can be specified with `default_return: YourReturnHere`
- Internal ZeroMQ addresses can be configured through the UI
- Added this change log

Changed

- Significant repository restructure
 - This repository restructure combined the `core` and `server` packages into a single `walkoff` package and moved modules such as `appcache` and `devicedb` out of the `apps` package
 - Top-level scripts with the exception of `walkoff.py` are now located in the `scripts` directory
 - These changes make the Walkoff project follow a more canonical repository structure, and are one step towards being able to install walkoff using `pip`, our eventual goal.
- Classes have been moved out of the `server.context.Context` class. They were located there to remove circular dependencies, but they have been moved into their own submodule.
- The `interface.__init__` module has been split into multiple modules
- The Sphinx Python documentation has been relocated to the `docs` directory and can be generated using `make html`. Additionally, they now use the ReadTheDocs theme.
- Google Protocol Buffer message structure has been significantly altered.
- Tags used for action, condition, and transform decorators have been encapsulated in a `WalkoffTag` enum

- `setup_walkoff.py` no longer explicitly calls Gulp

Security

- JWT structure changes
 - JWTs' identity is now the user ID, not the username
 - JWT claims are now the username and a list of role IDs this user possesses. These claims are populated on login, and require reauthentication to be updated.

3.6.28 [0.5.2]

Fixed

- Fixed a bug where the config host and port were not initialized before the server started.

3.6.29 [0.5.1]

Fixed

- A bug fix for case management due to a typo in the TS.

3.6.30 [0.5.0]

Introducing a more user-friendly playbook editor and custom event-driven interfaces

Added

- New user-friendly playbook editor
- Host and port can now be specified on the command line
- App-specific conditions and transforms
 - Conditions and transforms are now located in apps rather than in core, so they can be more easily created
- Branches now contain a “priority” field which can be used to determine the order in which the branches of a given action are evaluated
- Arguments to actions, conditions, and transforms which use references can select which component of the referenced action's output to use.
- Migration scripts to help ease a variety of backward-breaking changes – `migrate_workflows.py` and `migrate_api.py`
- Scripts to create Sphinx documentation have been added to the repository

Changed

- Custom interfaces with event handling
 - Interfaces are no longer attached to apps; they are now their own plugins and are contained in the `interfaces` directory
 - Interfaces can use new decorator functions to listen and respond to all events in Walkoff as they occur
- Better triggers
 - Triggers are no longer specified in the database. Instead, each individual action in a workflow can have its own set of conditions which can act as breakpoints in a workflow. You can send data to them through the server and have that data validated against a set of conditions before the action can resume.
 - You can still start a workflow from the beginning through the server
- Renamed workflow components for clarity
 - “steps” have been renamed “actions”
 - “next steps” have been renamed “branches”
 - “flags” have been renamed “conditions”
 - “filters” have been renamed “transforms”
- Script used to start the server has been renamed `walkoff.py`
- ZeroMQ keys are contained in the `.certificates` directory
- Playbook file format changes
 - Branches are now contained outside of actions, creating two top-level fields.
 - Branches have a `source_uid` and a `destination_uid` instead of just a `name` field
 - The start step on a workflow is indicated with the start step’s UID instead of its name
 - The `app` and `action` fields of actions, conditions, and transforms have been renamed `app_name` and `action_name` respectively.
 - Conditions and transforms contain an `app_name` field instead of just an `action` field
 - We have removed the `widgets` field and the `risk` field from actions
 - Devices for actions are specified by `id` rather than by `name`
 - Actions’ `inputs` field, as well as conditions’ and transforms’ `args` field has been renamed `arguments` and is now a complete JSON object
 - Playbooks now contain a `walkoff_version` field which will be used to indicate which version of WALKOFF created them. This will be helpful in the future to migrate workflows to new formats
- Minor changes to `api.yaml` schema
 - `dataIn` has been renamed `data_in`
 - `termsOfService` has been renamed `terms_of_service`
 - `externalDocs` has been renamed `external_docs` and is always an array
- Performance of worker processes has been improved by removing `gevent` from child processes and reducing polling
- The blinker Signals used to trigger events have been wrapped in a `WalkoffEvent` enum
- Internal sockets used for ZeroMQ communication have been moved to `core.config.config`

- Actions which are defined inside of a class must supply a device, or the workflow will fail on initialization
- The REST API to get the APIs of apps has been enhanced significantly and returns all of the API

Removed

- Unfortunately, event-driven actions have been broken for some time now. We have removed this functionality, but are working on an even better replacement for them in the meantime
- We have removed accumulated risk from workflows and risk from steps. This feature will be re-added at a future date
- We have removed widgets from the backend. This feature will be reimplemented later.
- Backend support for adding roles to users has been removed. All users are administrators as they have been in previous releases. There was never a UI component for this feature, and it was breaking some other components for editing users. Roles will be re-added in the next release.

Security

- HTTPS is enabled by default if certificates are placed in the `.certificates` directory.

Contributor

- Coverage.py is used to generate test coverage report. Travis-CI will fail if the code coverage is below 81% This percentage will rise over time

3.6.31 [0.4.2]

Fixed

- Bug fixes to Playbook editor
- Bug in global action execution

3.6.32 [0.4.1]

Fixed

- Bug fixes to playbook editor

3.6.33 [0.4.0]

Introducing custom devices and global app actions

Added

- Custom devices
 - Apps define their own fields needed in their devices
- Global app actions
 - Actions no longer need to be defined in a class

Fixed

- Performance improvements and bug fixes

3.6.34 [0.3.1]**Fixed**

- Bug Fixes

3.6.35 [0.3.0]

Introducing a new Angular-driven UI, schedulers, cases, and concurrency

Added

- Brand new UI
- Better concurrent execution of CPU-bound workflows
- Multiple workflows can be executed on the same scheduler
- New Scheduler UI page
- New Case Management UI

Changed

- Improved REST API
- Workflows are now stored as JSON

Fixed

- Bugs and performance improvements

Security

- Enhanced security using JSON Web Tokens
- Workflows are stored as JSON

3.6.36 [0.2.1]

Added

- Event-driven app actions
- Multiple return codes for actions and error handling

Changed

- Apps are now located in Walkoff-Apps repo
- Workflow results are stored in case database

Fixed

- Bug fixes and performance improvements

3.6.37 [0.2.0]

Introducing app action validation and improved data flow

Added

- New app specification using YAML metadata files
 - Better input validation using JSON schema
 - Arguments can be string, integer, number, arrays, or JSON objects
- Workflow animation during execution in the workflow editor
- Results from previously executed actions can be used later in workflows
- Better workflow monitoring during execution
- New apps
 - NMap
 - Splunk
 - TP-Link 100 Smart Outlet

Fixed

- UI styling and bug fixes
- Bug fixes and performance improvements

3.6.38 [0.1.2]

Added

- New Lixf playbook

Fixed

- Bug fixes to UI and apps

3.6.39 [0.1.1]

Added

- OpenAPI Specification for server endpoints and connexion Flask app
- New Apps
 - AR.Drone
 - Ethereum Blockchain
 - Facebook User Post
 - Webcam
 - Watson Visual Recognition
 - Tesla
 - Lix
- Better error handling in server endpoints
- Bug fixes
- Swagger UI documentation

Changed

- UI improvements

3.6.40 [0.1.0]

Initial Release

HTTP Routing Table

/apps

GET /apps, 32
GET /apps/apis, 32
GET /apps/apis/{app}, 32
POST /apps/apis, 32
PUT /apps/apis/{app}, 32
DELETE /apps/apis/{app}, 32

/auth

POST /auth, 33
POST /auth/logout, 33
POST /auth/refresh, 33

/availableresourceactions

GET /availableresourceactions, 36

/configuration

GET /configuration, 33
PUT /configuration, 33
PATCH /configuration, 33

/dashboards

GET /dashboards, 41
GET /dashboards/{dashboard}, 41
POST /dashboards, 41
PUT /dashboards, 41
DELETE /dashboards/{dashboard}, 41

/globals

GET /globals, 34
GET /globals/templates, 35
GET /globals/templates/{global_template}, 35
GET /globals/{global_var}, 34
POST /globals, 34
POST /globals/templates, 35
PUT /globals/templates/{global_template}, 35
PUT /globals/{global_var}, 34

DELETE /globals/templates/{global_template}, 35
DELETE /globals/{global_var}, 34

/roles

GET /roles, 36
GET /roles/{role_id}, 36
POST /roles, 36
PUT /roles/{role_id}, 36
DELETE /roles/{role_id}, 36

/scheduledtasks

GET /scheduledtasks, 37
GET /scheduledtasks/{scheduled_task_id}, 37
POST /scheduledtasks, 37
PUT /scheduledtasks/{scheduled_task_id}, 37
DELETE /scheduledtasks/{scheduled_task_id}, 37

/scheduler

GET /scheduler, 37
PUT /scheduler, 37

/users

GET /users, 38
GET /users/{user_id}, 38
POST /users, 38
PUT /users/{user_id}, 38
DELETE /users/{user_id}, 38

/workflowqueue

GET /workflowqueue, 40
GET /workflowqueue/{execution}, 40
POST /workflowqueue, 40
DELETE /workflowqueue/cleardb, 40
PATCH /workflowqueue/{execution}, 40

/workflows

GET /workflows, [39](#)

GET /workflows/{workflow}, [39](#)

POST /workflows, [39](#)

PUT /workflows/{workflow}, [39](#)

DELETE /workflows/{workflow}, [39](#)