CloudVision Mastery Workshop
Lab Guide

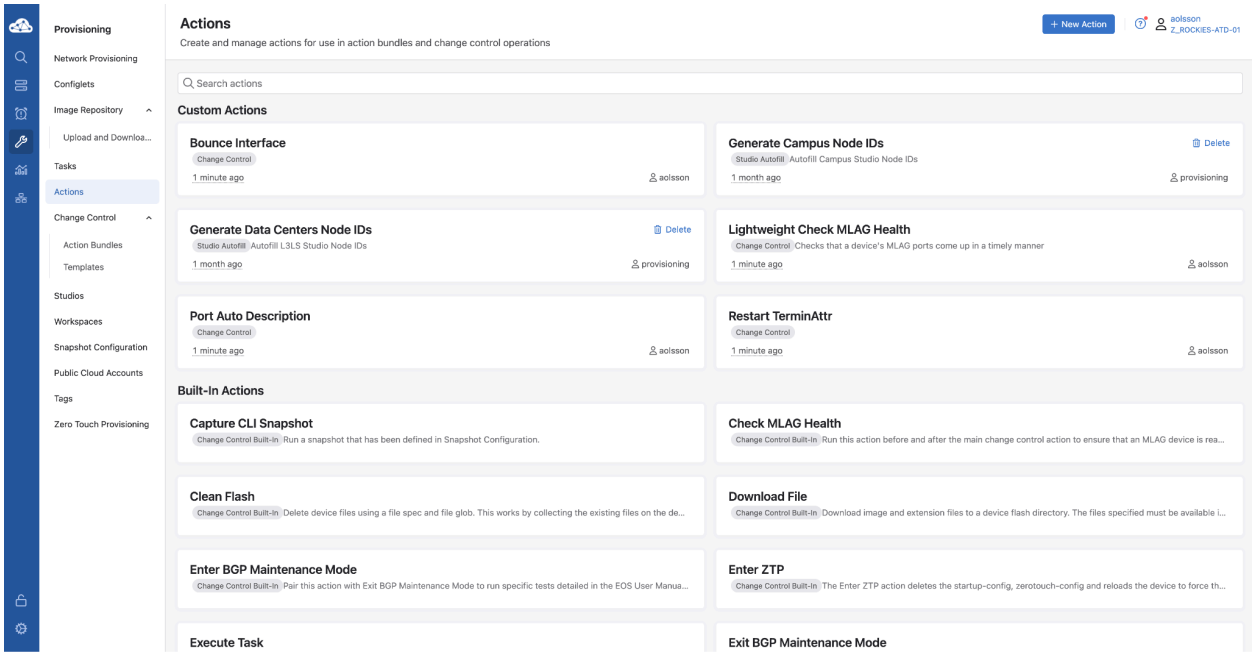
ACTIONS

Actions can be used within a change control to run a series of commands on devices. There are a number of built-in actions that can be selected when configuring a change control operation or creating a change control template. You can also create custom actions by creating arguments and writing a Python 3 script to manipulate the arguments.

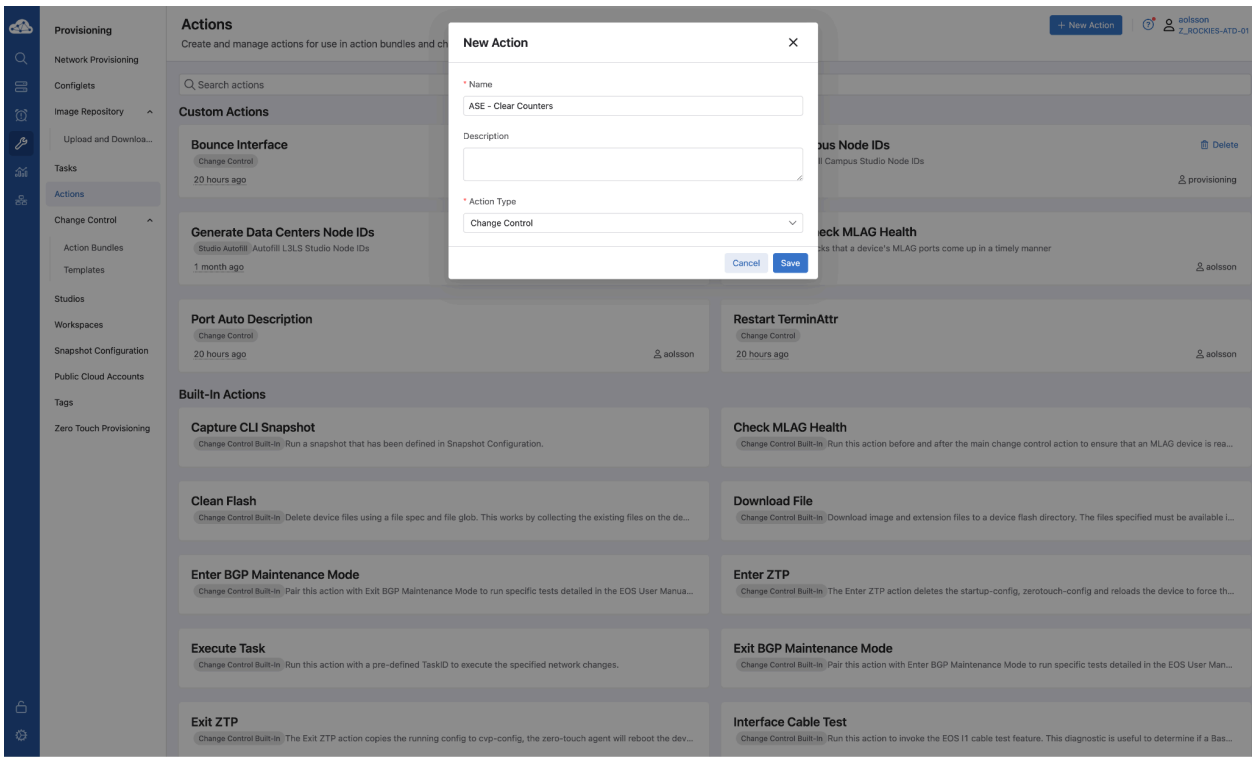
When configuring a change control, you'll select the dynamic values of an argument, like the device the action will run against and the image to install. When creating a custom action, you'll use dynamic arguments, like DeviceID, to enable users to select values when configuring a change control.

Create an Action

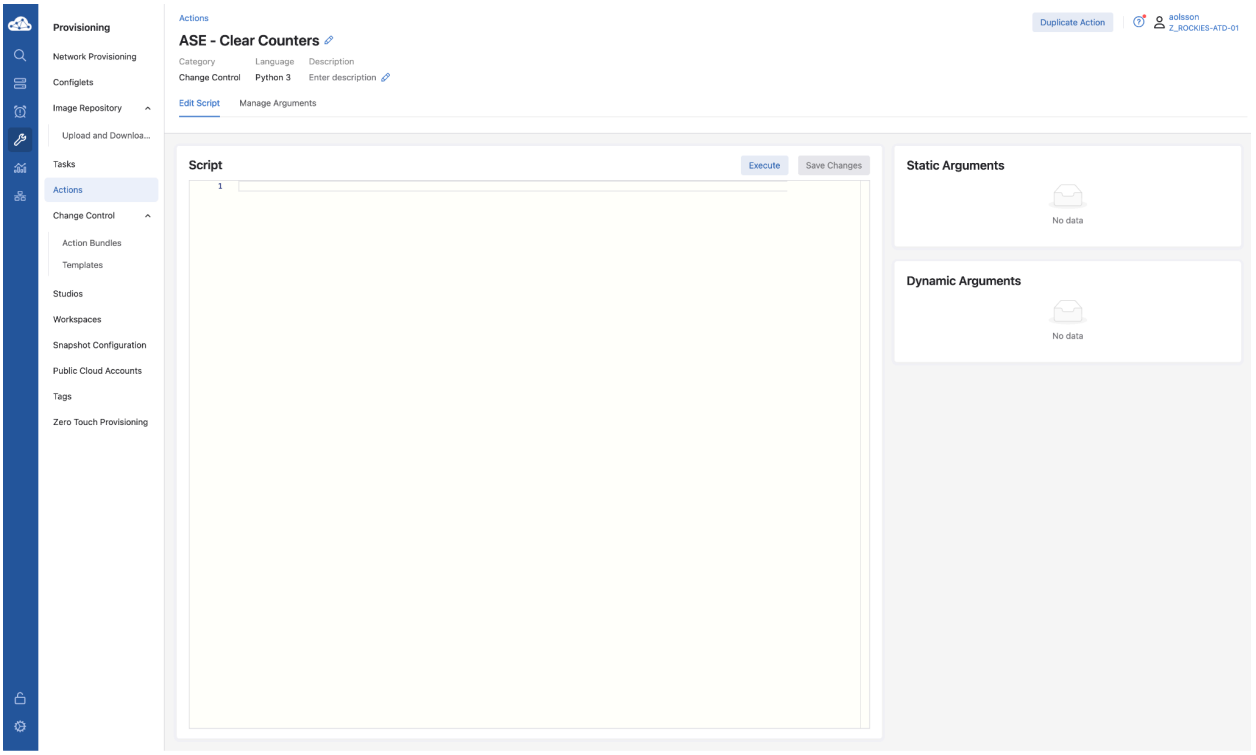
1. Navigate to **Provisioning** > **Actions.**
2. To create a new action, click on the "**+ New Action**" button at the top of the screen.



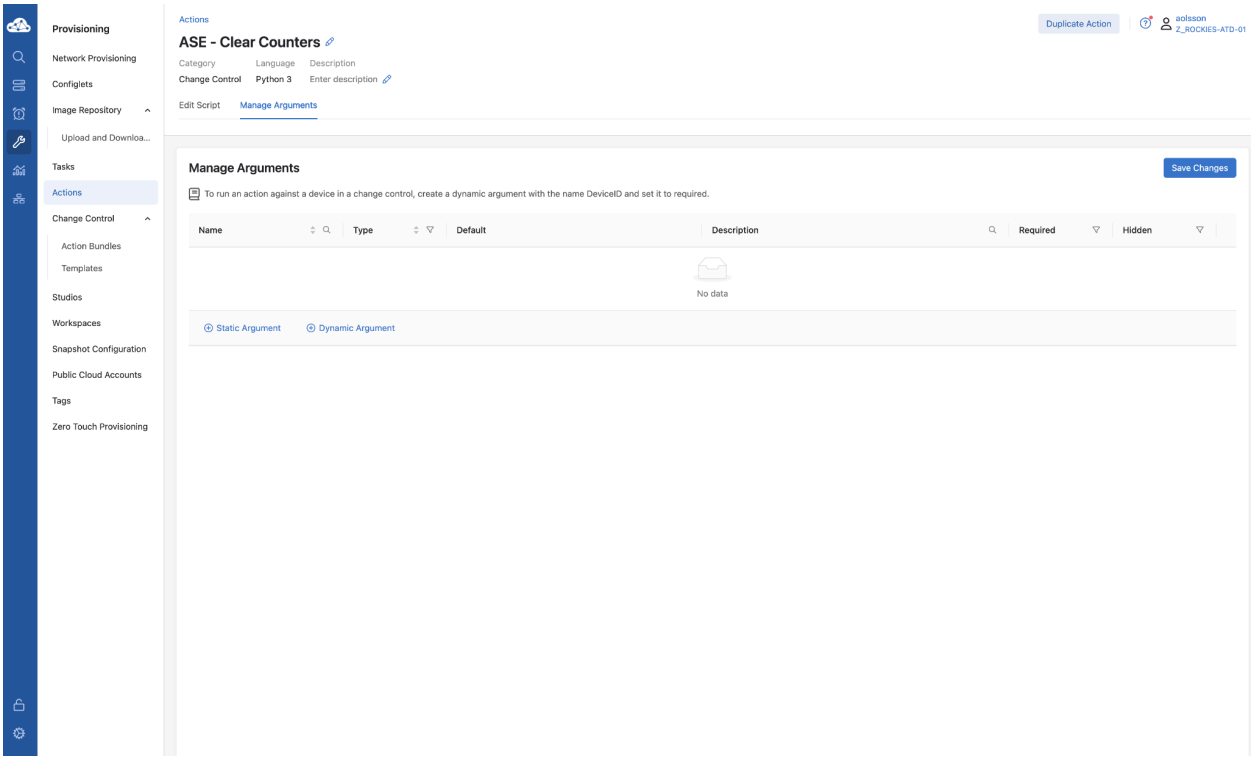
3. A “**New Action**” pop-up will appear. In the “**Name**” field, type your initials followed by “**- Clear Counters.**”
4. Click the blue “**Save**” button to save the new action.



5. You’ll then be redirected to the action configuration page for the new action.



6. We'll first want to create some arguments for the Device and Interface for the action to be run against. To do this, click the "**Manage Arguments**" button just below the action's name

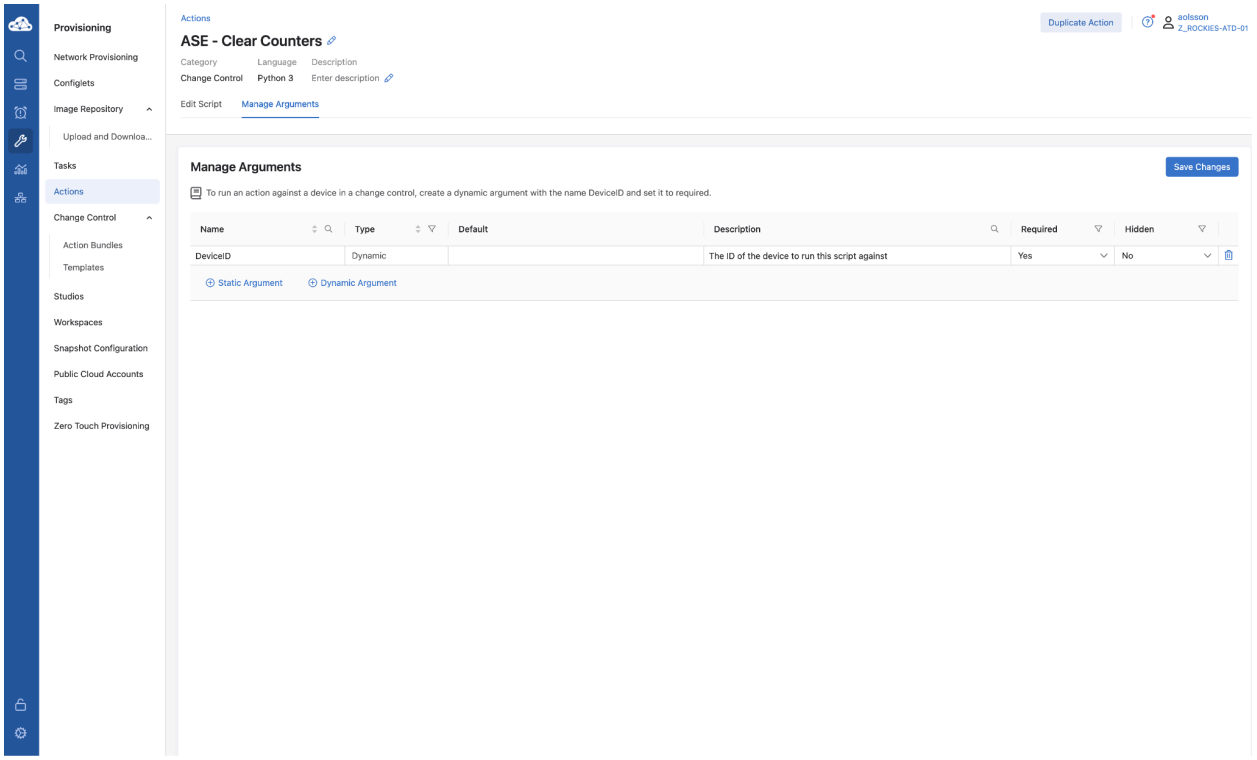


7. Create a new argument by selecting the "**+ Dynamic Argument**" at the bottom of the "**Manage Arguments**" section.

NOTE: Keep an eye on the Arguments and Variables as they are case sensitive.

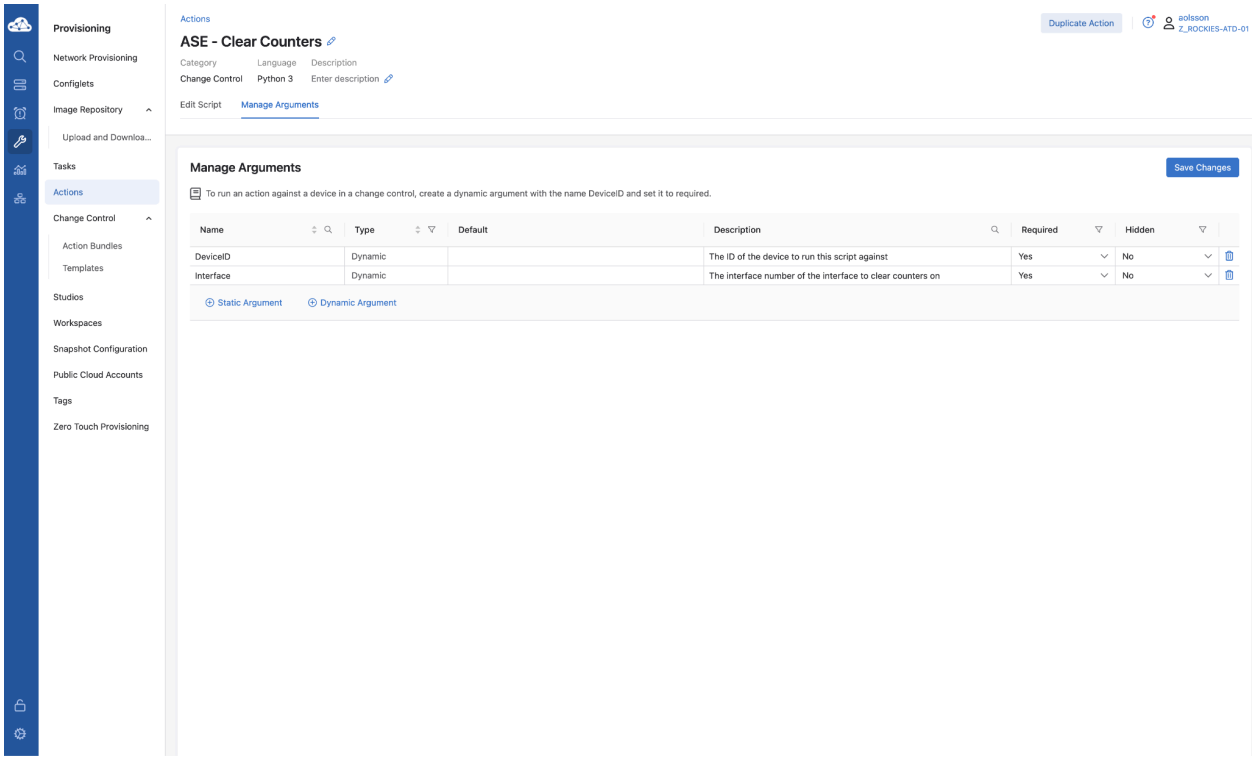
8. A new line for the new argument will appear in the "**Manage Arguments**" section. Enter the following into the new line:

```
Name : DeviceID
Type : Dynamic
Default : Leave Blank
Description : The ID of the device to run this script against
Required : Yes
Hidden : No
```

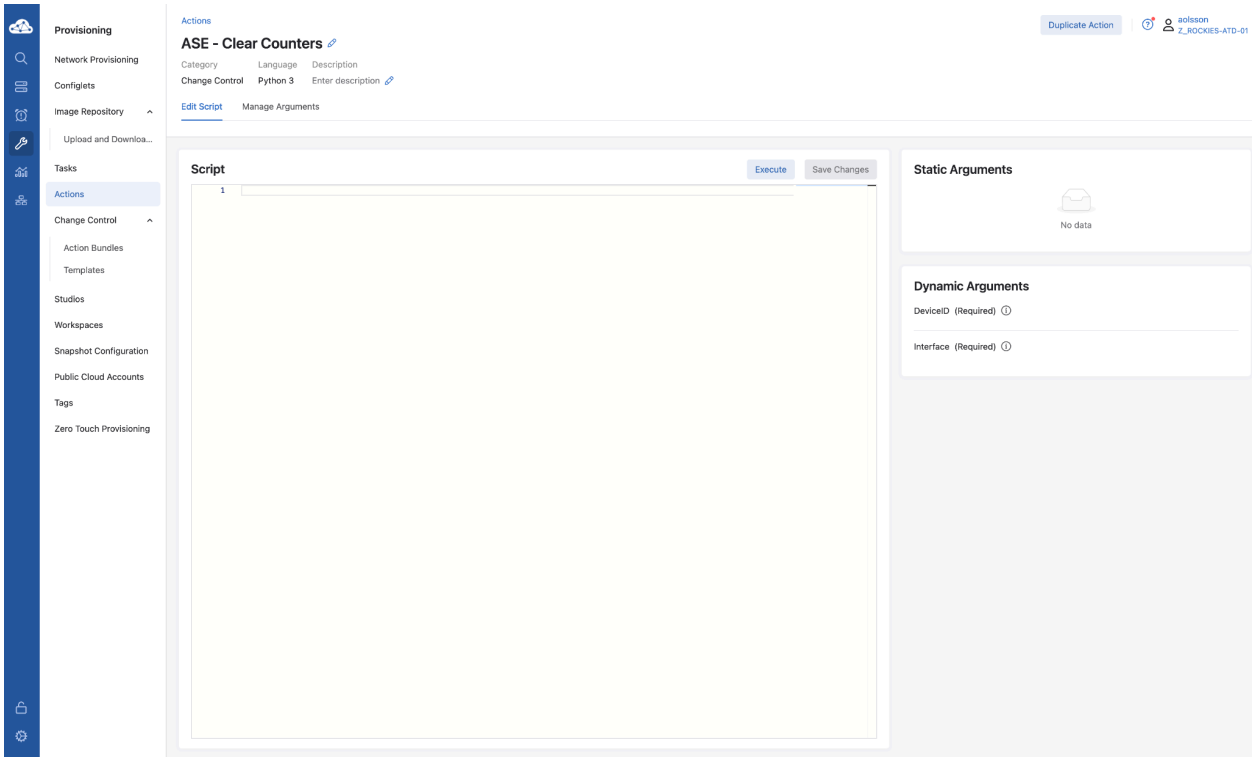


9. Repeat steps 7 & 8 to create the interface argument using the following values:

```
Name : Interface
Type : Dynamic
Default : Leave Blank
Description : The interface number of the interface to clear counters on
Required : Yes
Hidden : No
```



10. Click the blue “**Save Changes**” button on the right side of the screen.
11. Now that the Arguments are created, we can head to the Script section to begin writing the action. Click on the “**Edit Script**” button below the Action name.



12. On line 1 of the script we'll want to identify who wrote the script. Go ahead and enter the following on line 1, replacing "your-name-here" with your name:

```
# Written by your-name-here
```

13. The next thing we need to do is create a variable in the script that references the "**Interface**" argument that was created. To do that, we skip a line an then add the following on lines 3 & 4.

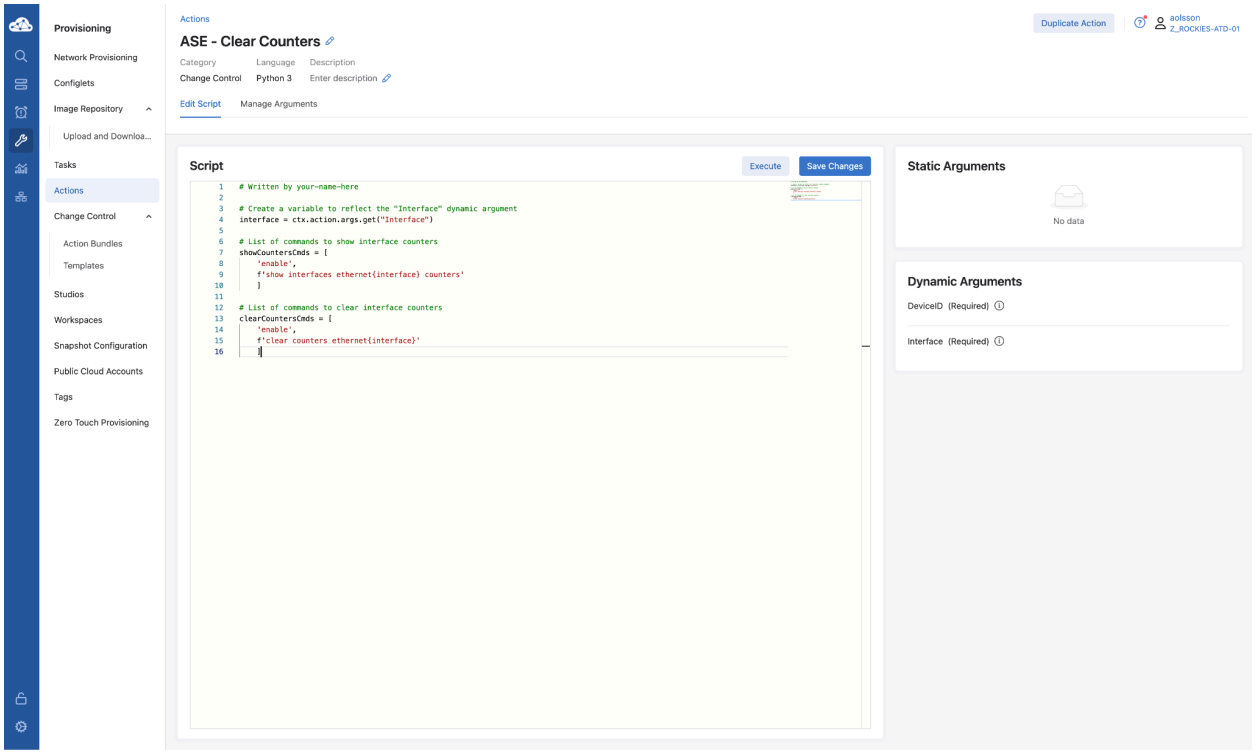
```
# Create a variable to reflect the "Interface" dynamic argument
interface = ctx.action.args.get("Interface")
```

14. Now, we need to add two different lists of commands that will need to be run on the switch. The first list will show the interface's existing counters, and the second will clear the interface counters. Skip a line and then paste the following into lines 6 - 16:

```
# List of commands to show interface counters
showCountersCmds = [
   'enable',
   f'show interfaces ethernet{interface} counters'
   ]

# List of commands to clear interface counters
clearCountersCmds = [
   'enable',
   f'clear counters ethernet{interface}'
   ]
```

15. Validate that the script section matches the following screenshot.



16. Let's continue editing the script by adding some logging stating that we're getting the values of the current interface counters. Skip another line and then add the following to lines 18 & 19:

```
# Create log entry
ctx.alog(f'Getting counters for Ethernet{interface}')
```

17. Now, we can proceed with running the commands listed in the "showCountersCmds" list on the switch. To do that, add another blank line and the following to lines 21 and 22.

```
# Run the list of commands to get counters
cmdResponse = ctx.runDeviceCmds(showCountersCmds)
```

18. The commands have now been run on the switch and the response has been stored in the variable "**cmdResponse.**" Let's create a log entry that outputs the response. On lines 24 - 25 add the following:

```
# Create a log entry of the counters
ctx.alog(f'The currect interface counters are : {cmdResponse[1]["response"]}')
```

NOTE: The cmdResponse value returned above is similar to the following:

```
[{'error': '', 'response': {}},
{'error': '', 'response': {
   'interfaces': {
      'Ethernet1': {
         'inBroadcastPkts': 1,
         'inDiscards': 0,
         'inMulticastPkts': 38,
         'inOctets': 7529,
         'inUcastPkts': 0,
         'lastUpdateTimestamp': 1717697811.8986592,
         'outBroadcastPkts': 836, 'outDiscards': 0,
         'outMulticastPkts': 37,
         'outOctets': 45227, 'outUcastPkts': 0}
      }
   }
}]
```

The cmdResponse list contains a dictionary for each command run on the device. Since we're interested in the "show interfaces ethernet counters" command response value, we need to specify "**[1]**" to indicate the second item in the list (since list values start at the 0th item in Python) and "**["response"]**" for the response data.

19. Now that we have logged what the counters were, let's proceed with clearing the counters. Let's create a blank line, then create another log entry stating that we will clear the counters. On lines 27 - 28 enter the following:

```
# Create log entry before clearing the counters
ctx.alog(f'Clearing counters for Ethernet{interface}')
```

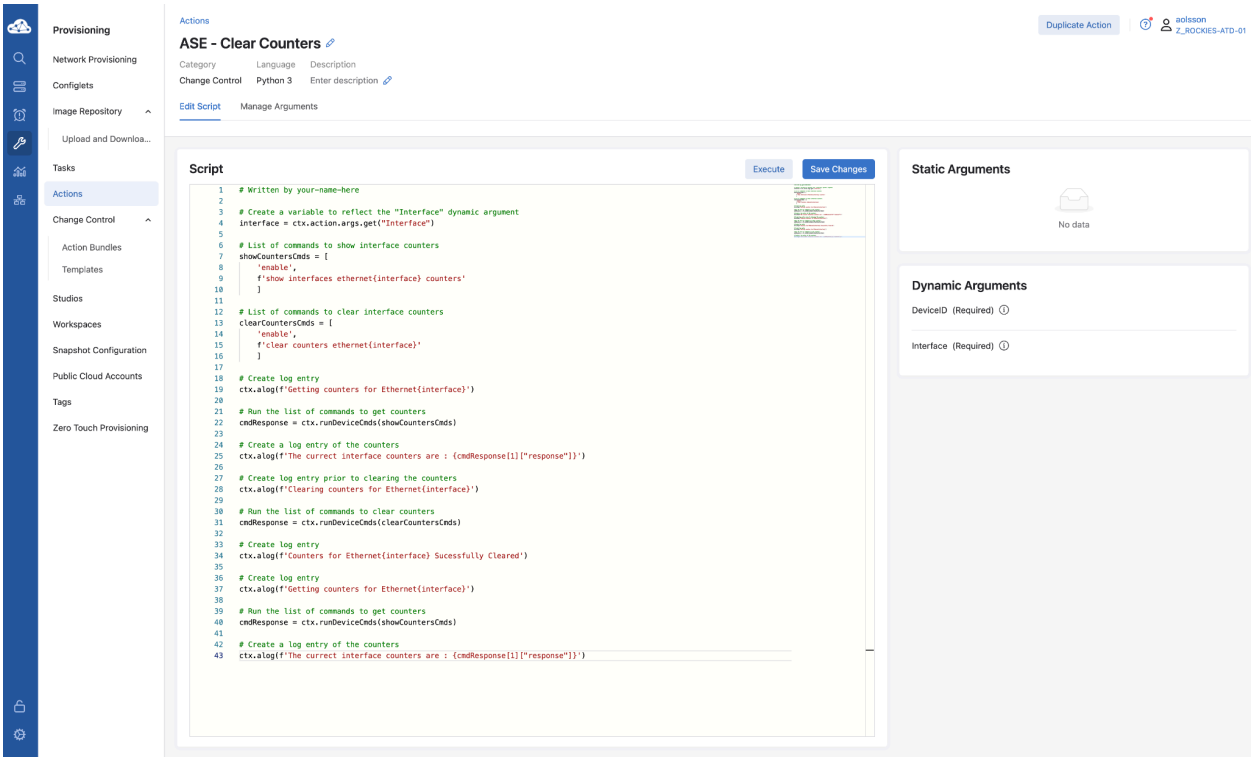
20. Let's actually clear the counters now. Let's skip another line and then enter the following on lines 30 - 31:

```
# Run the list of commands to clear counters
cmdResponse = ctx.runDeviceCmds(clearCountersCmds)
```

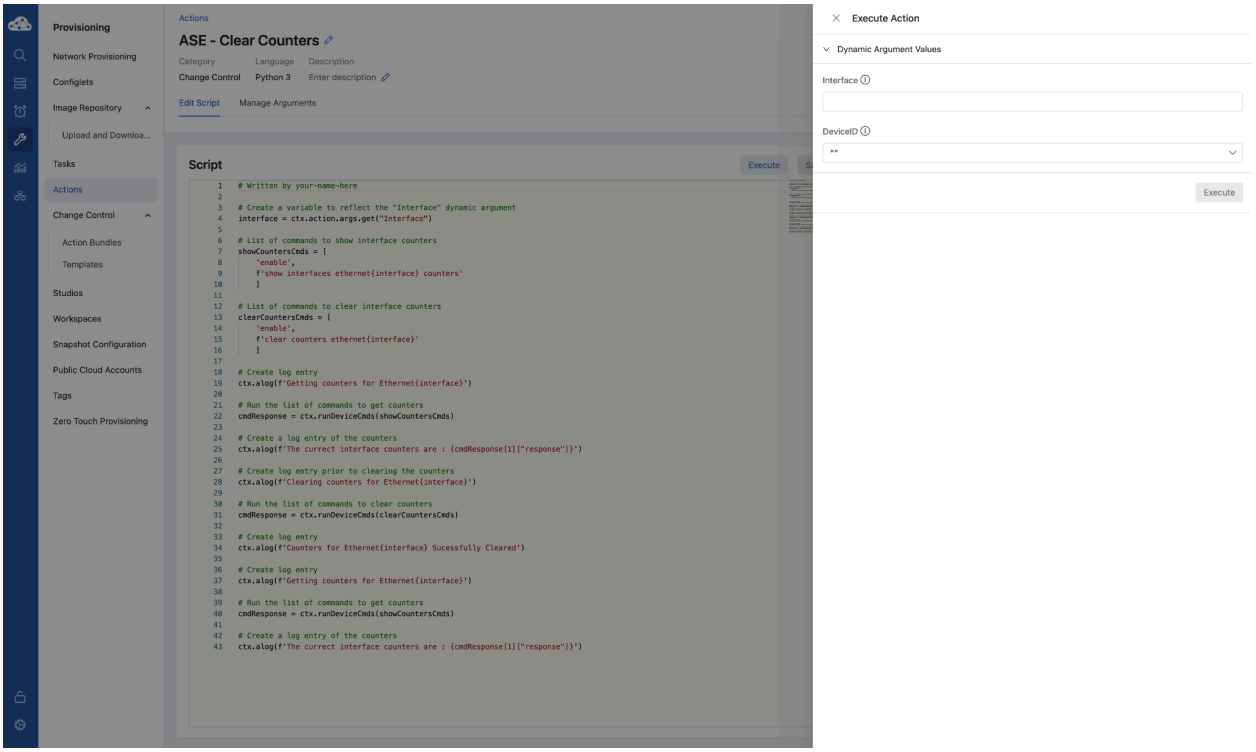
21. Now that the counters are cleared let's create another log entry stating that the counters have been successfully cleared. To do that, let's skip another line and add the following content to lines 33 - 34:

```
# Create log entry
ctx.alog(f'Counters for Ethernet{interface} Sucessfully Cleared')
```

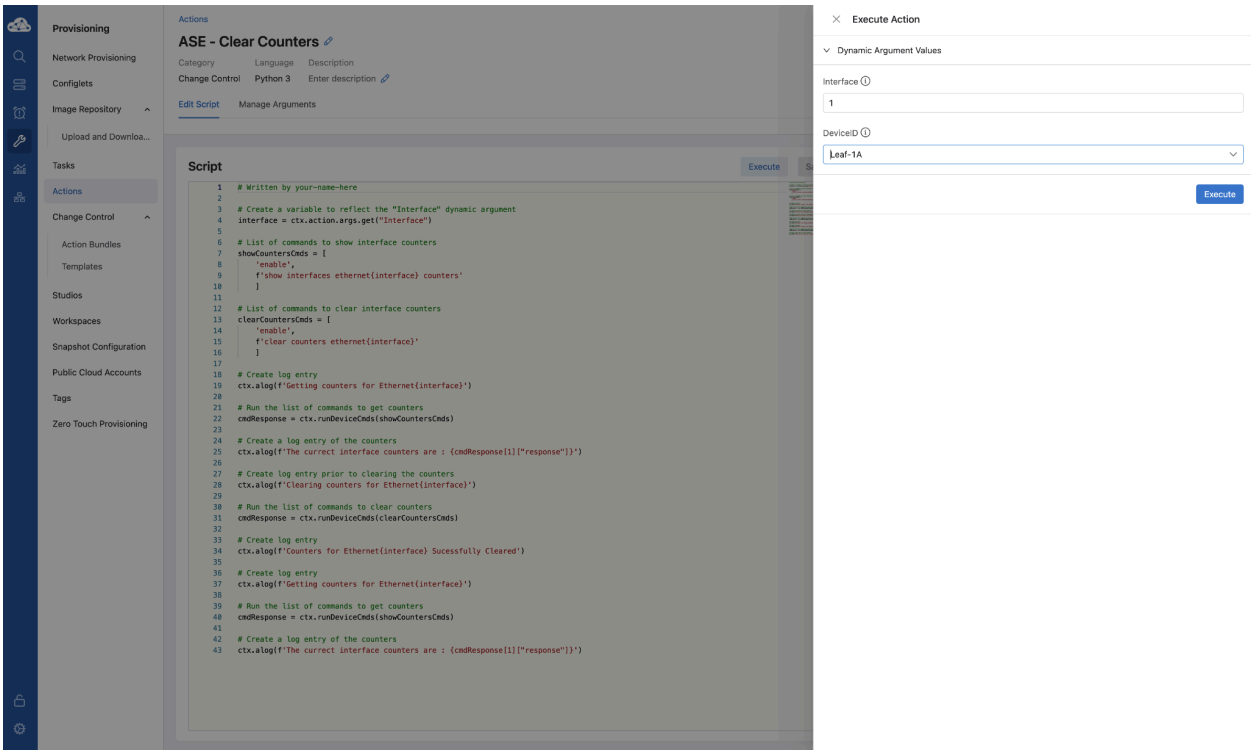
22. For the final part of the script, let's repeat steps 16 - 18 to get the interface's current counter values and log them.
23. Validate that the script matches the screenshot below, then click the blue "**Save Changes**" button to save the action.



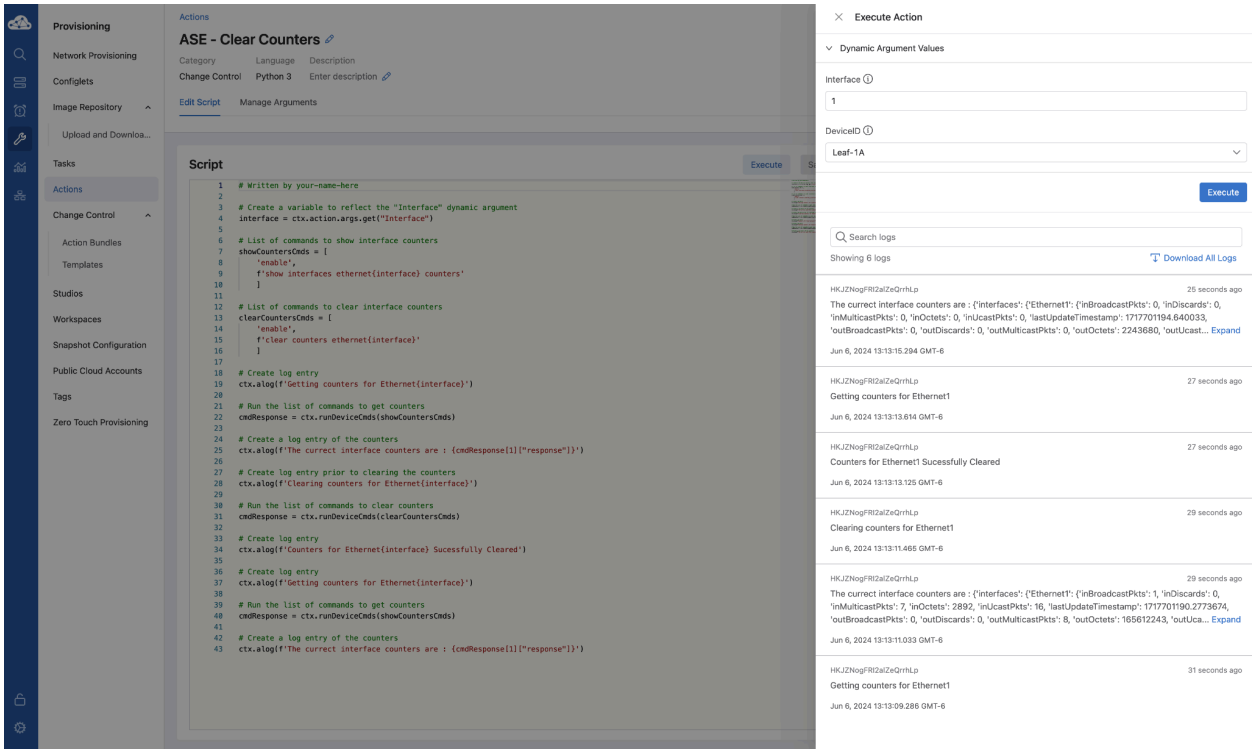
24. Now, we can proceed with testing our Action. To do so, click on the light blue "**Execute**" button.
25. An "**Execute Action**" dialog box will appear on the right side of the screen. Click "**Dynamic Argument Values**" to expand the arguments section.



26. Let's use Ethernet1 for this test, so specify "**1**" for the "**Interface**."
27. In the DeviceID dropdown, select "**Leaf-1A**"
28. Next, click the blue "**Execute**" button.



29. As the action executes, it will provide the logging output in the pane below the "**Execute**" button. The oldest log entry will be at the bottom, so you should read the logs from the bottom up. You'll be able to see that the current interface counters were displayed, they were then cleared, and the new counters were displayed.



30. Now, your custom action is available to be used within a change control.

NOTE: Check out this document for more information on custom actions. https://www.arista.com/en/support/toi/cvp-2021-3-0/14901-ui-for-custom-action-scripts Arista also has a repo with some custom actions you might find useful. That repo can be accessed here: https://github.com/aristanetworks/cloudvision-python-actions