**Project:** INFRA

**Issue Type:** Story

**Summary:** Implement Cloud Custodian policy & scheduler to auto-delete CFN stacks by end\_time tag

⸻

**Description:**

We need to automate cleanup of short-lived CloudFormation stacks by tagging each stack with an end\_time timestamp and having a scheduled Cloud Custodian Lambda delete any stack whose end\_time has passed. This will ensure unused stacks are torn down promptly without manual intervention.

⸻

**Acceptance Criteria:**

1. A new Cloud Custodian YAML policy (delete-by-end-time.yml) exists in our repo under /policies/ with:

• resource: cfn

• mode.type: periodic, schedule: "rate(5 minutes)"

• filters matching tag:end\_time with value\_type: age, op: ge, value: 0

• actions to disable termination protection and delete the stack

2. An IAM execution role CustodianLambdaRole is created with:

• Trust policy allowing Lambda service to assume it

• Permissions policy granting CloudFormation list/describe/delete and CloudWatch Logs write

3. A dedicated S3 bucket (e.g. my-custodian-bucket) is created for staging Custodian Lambda bundles.

4. A single custodian run --region <region> -s s3://<bucket> policies/delete-by-end-time.yml command successfully provisions:

• Lambda function custodian-cfn-delete-by-end-time

• EventBridge rule scheduled every 5 minutes invoking the Lambda

5. A dry-run execution shows correct stacks are targeted without actual deletion.

6. Monitoring dashboards or alarms are in place to surface any errors or invocation failures.

⸻

**Tasks:**

• Create policies/delete-by-end-time.yml using agreed filter and mode parameters

• Define and apply trust-policy.json and permissions-policy.json for CustodianLambdaRole

• Provision the S3 bucket for staging (aws s3 mb …)

• Run and validate dry-run (custodian run --dryrun …) and review .cache output

• Deploy policy (custodian run -s s3://…) and verify Lambda & EventBridge in AWS Console

• Test end-to-end: tag a stack with an expired end\_time and confirm it’s deleted within 5 minutes

• Document setup steps and update runbook

⸻

**Definition of Done:**

• All acceptance criteria met and reviewed by Infra team

• Implementation code and policies checked into Git

• Runbook updated and team notified of the new automated cleanup process

• Clean execution logs with zero errors for at least two consecutive runs

Here are **acceptance criteria** (in Given/When/Then format) for each task in the Custom Actions & Policies guide:

⸻

**1. Create a Python module**

**Task:** Create a Python module (e.g. custodian\_custom/actions.py).

• **Given** a project directory initialized for Python packaging

• **When** I add custodian\_custom/actions.py under the package root

• **Then** the file exists and is importable (import custodian\_custom.actions succeeds)

⸻

**2. Define an action class**

**Task:** Define an action class inheriting from c7n.actions.Action.

• **Given** I have a module custodian\_custom/actions.py

• **When** I write a class class MyAction(Action): ... that subclasses c7n.actions.Action

• **Then** Custodian recognizes MyAction when registered (no import errors)

⸻

**3. Implement validate()**

**Task:** Implement the validate() method for schema checks.

• **Given** the action class declares a JSON schema in its schema attribute

• **When** I run custodian validate against a policy using this action

• **Then** the policy either passes or fails validation strictly according to my validate() logic

⸻

**4. Implement process(resources, event=None)**

**Task:** Implement the core logic in process().

• **Given** a list of test resource dicts and a stubbed AWS session

• **When** I call MyAction().process(resources)

• **Then** the intended side-effects occur (e.g. AWS API calls to remove tags) on every resource

⸻

**5. Register the action via entry\_points**

**Task:** Add an entry point in setup.py:

entry\_points={

"custodian.actions": [

"my-action = custodian\_custom.actions:MyAction"

]

}

• **Given** a built package (e.g. via python setup.py sdist)

• **When** I install it into the Custodian environment (pip install .)

• **Then** custodian run --help lists my-action under available actions

⸻

**6. Install your package in the Custodian runtime**

**Task:** Deploy your custom action code so Custodian Lambda can import it.

• **Given** my custom action package is published to PyPI or otherwise made available

• **When** I include it in my Lambda deployment (e.g. via requirements.txt or –extra-requirements)

• **Then** the Lambda logs show no “module not found” errors, and my action runs successfully in AWS

⸻

**7. Use custom actions in policies**

**Task:** Reference the custom action in a YAML policy:

actions:

- type: cleanup-tag

tag: Environment

• **Given** the custom action is installed in the Lambda runtime

• **When** I run custodian run with a policy that uses type: cleanup-tag

• **Then** resources matching my filters trigger the CleanupTagAction logic

⸻

**8. Define policy defaults and variables**

**Task:** Use a defaults: block to DRY out common settings:

policies:

defaults:

mode:

type: periodic

schedule: rate(1 day)

- name: example-policy

resource: aws.s3

actions: [ delete ]

• **Given** multiple policies share identical mode or other settings

• **When** I validate or run policies using the defaults: block

• **Then** those shared settings apply automatically to all policies without repetition

⸻