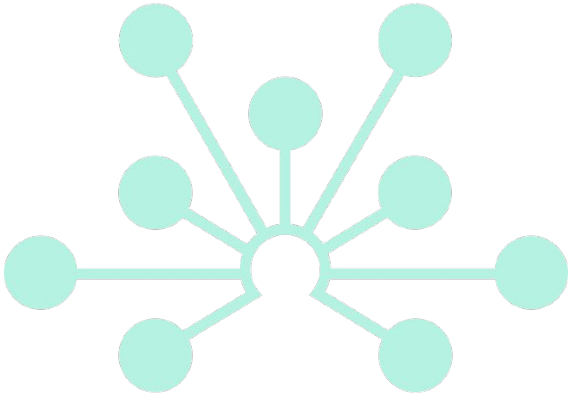


Move Fast and Don’t Break Things: How testing enabled a total rewrite of critical infrastructure

Ethan Holz
Research Software Engineer, Ecosystem Infrastructure

July 23rd, 2025

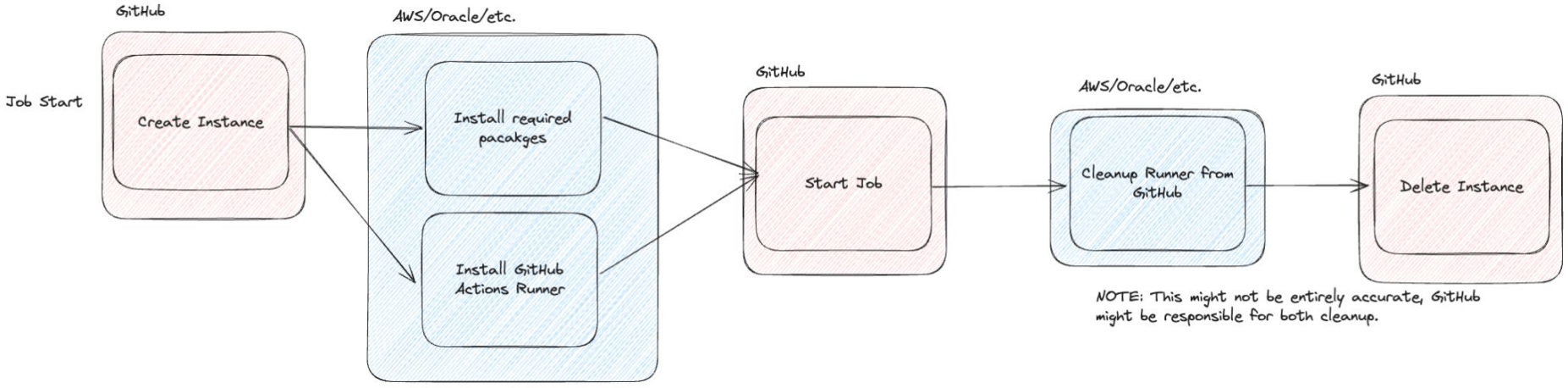
Agenda

- The gha-runner project
- Why we wanted to re-architect the underlying code
- How tests enabled us to make big changes
- Where we failed
- How can you do this in your own projects

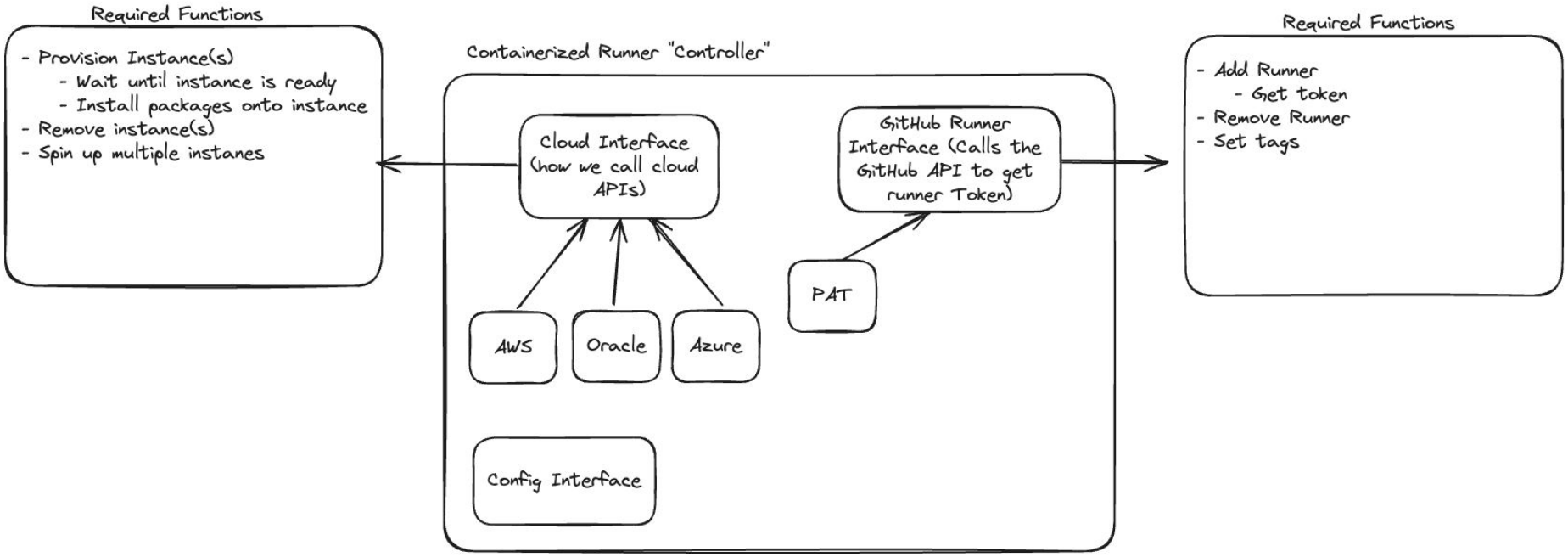
gha-runner Project

- A project to provide simple, ephemeral GitHub Actions Runners on multiple compute providers
- Focused on best practices and user experience
- 60% of the time was spent on testing and validation
- Used by at least of 6 public projects (and a few not public ones)
- Based on [machulav/ec2-github-runner]

High-level architecture



Interfaces


Required Functions
- Provision Instance(s)
- Wait until instance is ready
- Install packages onto instance
- Remove instance(s)
- Spin up multiple instanes
Containerized Runner "Controller"
Cloud Interface (how we call cloud APIs)
GitHub Runner Interface (Calls the GitHub API to get runner Token)
PAT
AWS
Oracle
Azure
Config Interface
Required Functions
- Add Runner
- Get token
- Remove Runner
- Set tags

The idea of how this should work (v0.3.0)

```yaml
name: Reusable OpenMM GPU Test
on:
  push: main

jobs:
  openmm-test:
    runs-on: ubuntu-latest
    defaults:
      run:
        shell: bash -leo pipefail {0}
    needs:
      - start-aws-runner
    steps:
      - uses: actions/checkout@v4
      - name: Print disk usage
        run: "df -h"
      - name: Print Docker details
        run: "docker version || true"
      - name: Check for nvidia-smi
        run: "nvidia-smi || true"
      - uses: mamba-org/setup-micromamba@main
        with:
          environment-name: openmm
          create-args: >-
            openmm
          condarc: |
            channels:
              - conda-forge
      - name: Test for GPU
        id: gpu_test
        run: python -m openmm.testInstallation
```

The idea of how this should work (v0.3.0)

```yaml
start-aws-runner:
  runs-on: ubuntu-latest
  outputs:
    mapping: ${{ steps.aws-start.outputs.mapping }}
    instances: ${{ steps.aws-start.outputs.instances }}
  steps:
    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        role-to-assume: ${{ secrets.AWS_ROLE }}
        aws-region: us-east-1
    - name: Create cloud runner
      id: aws-start
      uses: omsf-eco-infra/gha-runner@main
      with:
        provider: "aws"
        action: "start"
        aws_region_name: us-east-1
        aws_image_id: ami-0f7c4a792e3fb63c8
        aws_instance_type: ${{ inputs.instance_type }}
        aws_home_dir: /home/ubuntu
      env:
        GH_PAT: ${{ secrets.GH_PAT }}
```

The idea of how this should work (v0.3.0)

```yaml
stop-aws-runner:
  runs-on: ubuntu-latest
  needs:
    - start-aws-runner
    - openmm-test
  if: ${{ always() }}
  steps:
    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        role-to-assume: ${{ secrets.AWS_ROLE }}
        aws-region: us-east-1
    - name: Stop instances
      uses: omsf-eco-infra/gha-runner@main
      with:
        provider: "aws"
        action: "stop"
        aws_region_name: us-east-1
        instance_mapping: ${{ needs.start-aws-runner.outputs.mapping }}
      env:
        GH_PAT: ${{ secrets.GH_PAT }}
```

How this works today

```yaml
jobs:
  start-aws-runner:
    runs-on: ubuntu-latest
    permissions:
      id-token: write
      contents: read
    outputs:
      mapping: ${{ steps.aws-start.outputs.mapping }}
    steps:
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_ROLE }}
          aws-region: us-east-1
      - name: Create cloud runner
        id: aws-start
        uses: omsf/start-aws-gha-runner@v1.0.0
        with:
          aws_image_id: ami-0d5079d9be06933e5
          aws_instance_type: g4dn.xlarge
          aws_home_dir: /home/ubuntu
        env:
          GH_PAT: ${{ secrets.GH_PAT }}
```

```yaml
self-hosted-test:
  runs-on: self-hosted
  needs:
    - start-aws-runner
  steps:
    - uses: actions/checkout@v4

    - name: Print disk usage
      run: "df -h"

    - name: Print Docker details
      run: "docker version || true"

    - name: Setup Conda Environment
      uses: mamba-org/setup-micromamba@v2
      with:
        environment-file: devtools/conda-envs/test_env.yaml

    - name: Install Package and test plugins
      run: python -m pip install . utilities/test_plugins/

    - name: Double-check local installation
      run: python -c "from openff.evaluator import __version__; print(__version__)"
    - name: Run integration tests
      run: |
        cd integration-tests/default-workflows/
        python run.py
```

```yaml
stop-aws-runner:
  runs-on: ubuntu-latest
  permissions:
      id-token: write
      contents: read
  needs:
    - start-aws-runner
    - self-hosted-test
  if: ${{ always() }}
  steps:
    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        role-to-assume: ${{ secrets.AWS_ROLE }}
        aws-region: us-east-1
    - name: Stop instances
      uses: omsf/stop-aws-gha-runner@v1.0.0
      with:
        instance_mapping: ${{ needs.start-aws-runner.outputs.mapping }}
      env:
        GH_PAT: ${{ secrets.GH_PAT }}
```

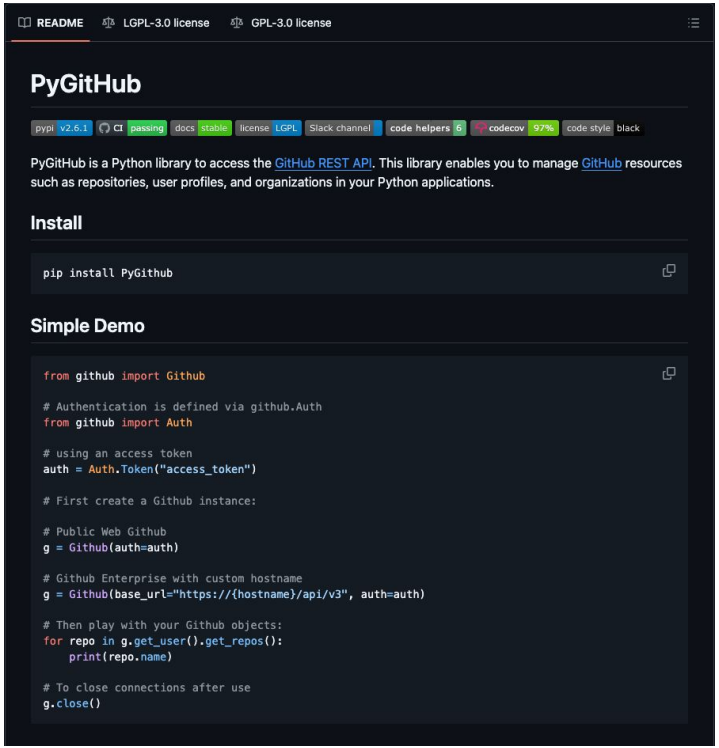
What testing looked like for us

- Well defined mocks for AWS using moto
- Well defined deployment flow
- Convoluted testing for GitHub and input parsing
- Integration tests that could be run on AWS

Moment #1: Let’s Remove PyGithub

Why?

- Bloated dependency
- The API we need isn’t in the library
- Difficult to mock



README LGPL-3.0 license GPL-3.0 license

PyGitHub

pypi v2.6.1 CI passing docs stable license LGPL Slack channel code helpers 6 codecov 97% code style black

PyGitHub is a Python library to access the GitHub REST API. This library enables you to manage GitHub resources such as repositories, user profiles, and organizations in your Python applications.

Install

```
pip install PyGithub
```

Simple Demo

```
from github import Github

# Authentication is defined via github.Auth
from github import Auth

# using an access token
auth = Auth.Token("access_token")

# First create a Github instance:

# Public Web Github
g = Github(auth=auth)

# Github Enterprise with custom hostname
g = Github(base_url="https://{hostname}/api/v3", auth=auth)

# Then play with your Github objects:
for repo in g.get_user().get_repos():
    print(repo.name)

# To close connections after use
g.close()
```

How testing looked

```python
@pytest.fixture
def github_release_mock():
    # Create MagicMocks for components not directly patched
    mock_release = MagicMock()
    mock_asset = MagicMock()
    mock_runners = MagicMock()
    runner = MagicMock(spec=SelfHostedActionsRunner)
    runner2 = MagicMock(spec=SelfHostedActionsRunner)

    # Setup fixed attributes for mocks
    asset_name = "runner-linux-x64.tar.gz"
    asset_url = "https://github.com/testing/runner-linux-x64.tar.gz"
    mock_asset.name = asset_name
    mock_asset.browser_download_url = asset_url
    runner.labels.return_value = [{"name": "runner-linux-x64"}]
    runner2.labels.return_value = [{"name": "runner-linux-x64"}]
    mock_runners.__iter__.return_value = [runner, runner2]

    # Using patch as a context manager inside the fixture
    with patch("gha_runner.gh.Github") as mock_github:
        mock_repo = MagicMock()
        mock_github.return_value.get_repo.return_value = mock_repo
        mock_repo.get_latest_release.return_value = mock_release
        mock_repo.get_self_hosted_runners.return_value = mock_runners
        mock_release.get_assets.return_value = [mock_asset]
        mock_repo.remove_self_hosted_runner.return_value = True

        instance = GitHubInstance("test", "testing/testing")

        yield instance, mock_asset, mock_repo
```

How testing looked

```python
@pytest.mark.parametrize(
    "status_code, ok, content, error",
    [(200, True, None, None), (404, False, "Not Found", TokenRetrievalError)],
)
def test_create_runner_token(post_fixture, status_code, ok, content, error):
    instance, mock_response = post_fixture
    mock_response.status_code = status_code
    mock_response.ok = ok
    mock_response.content = content
    error_str = f"Error creating runner token: Error in API call for https://api.github.com/repos/testing/testing/actions/runners/registration-token: {mock_response.content}"
    if error:
        with pytest.raises(error, match=error_str):
            instance.create_runner_token()
    else:
        response = instance.create_runner_token()
        assert response == mock_response.json.return_value["token"]
        mock_response.json.assert_called_once()
```

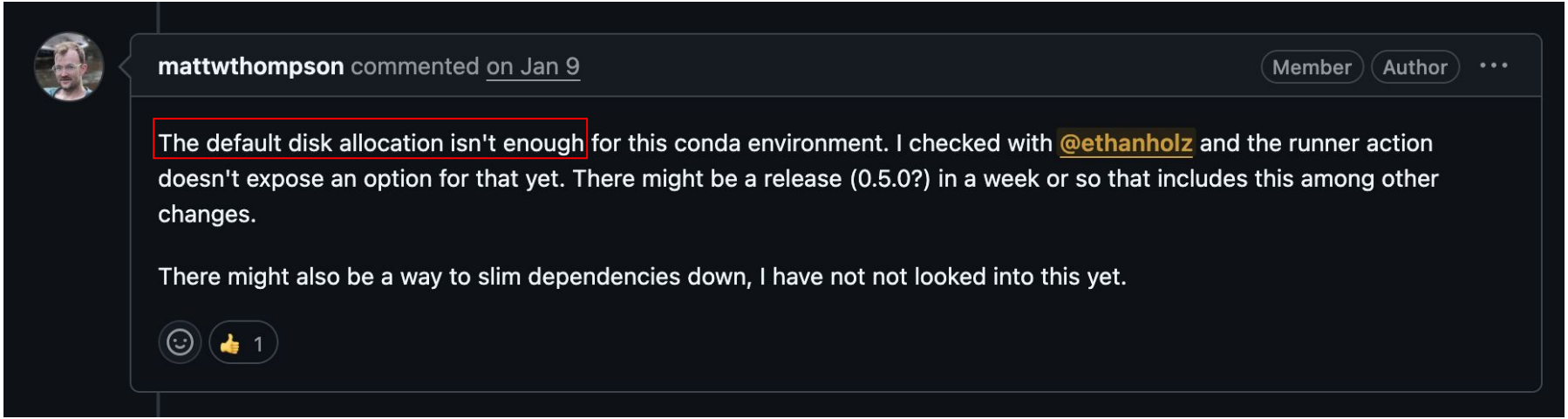
How we fixed this problem

1. Replace existing code with a generic requests approach side-by-side
2. Write tests that model the functionality of the initial tests using the responses library
3. Run tests to ensure that both pass
4. Remove the old tests

How testing looks now

```python
@pytest.fixture
def github_instance():
    return GitHubInstance(token="fake-token", repo="test/test")


@pytest.fixture
def mock_runner():
    return SelfHostedRunner(
        id=1, name="test-runner", os="linux", labels=["test-label"]
    )


def test_init(github_instance):
    assert github_instance.token == "fake-token"
    assert github_instance.repo == "test/test"
    assert github_instance.BASE_URL == "https://api.github.com"


def test_headers(github_instance):
    headers = github_instance._headers({})
    assert headers["Authorization"] == "Bearer fake-token"
    assert headers["X-Github-Api-Version"] == "2022-11-28"
    assert headers["Accept"] == "application/vnd.github+json"


@responses.activate
def test_create_runner_token(github_instance):
    responses.add(
        responses.POST,
        "https://api.github.com/repos/test/test/actions/runners/registration-token",
        json={"token": "test-token"},
        status=200,
    )
    token = github_instance.create_runner_token()
    assert token == "test-token"
```

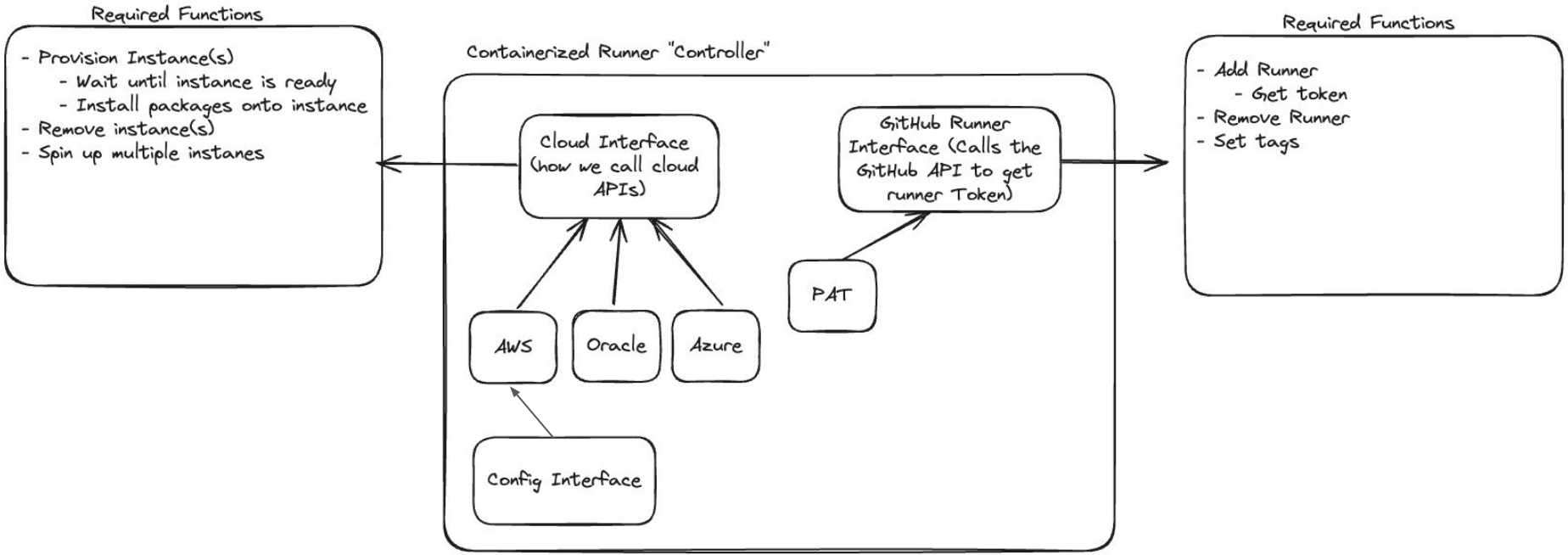
Moment #2: We need more disk!



Moment #2: We need more disk!

The problem

- Parsing new inputs
  - Non-trivial
  - Non-deterministic
- Much of the existing code relied on AWS convention
  - Convoluted for developers to add new cloud providers


Required Functions
- Provision Instance(s)
- Wait until instance is ready
- Install packages onto instance
- Remove instance(s)
- Spin up multiple instanes
Containerized Runner "Controller"
Cloud Interface (how we call cloud APIs)
GitHub Runner Interface (Calls the GitHub API to get runner Token)
PAT
AWS
Oracle
Azure
Config Interface
Required Functions
- Add Runner
- Get token
- Remove Runner
- Set tags

A brief interlude on parsing inputs

- We are using Docker to run this CI
- Inputs to Docker GitHub Actions are passed like the following: INPUT_<NAME>
- All inputs are passed as strings, even if no value is set (creating an empty string)

How the functionality looked

```python
def _env_parse_helper(
    params: dict, var: str, key: str, is_json: bool = False
) -> dict:
    val = os.environ.get(var)
    if val is not None and val != "":
        if is_json:
            params[key] = json.loads(val)
        else:
            params[key] = val
    return params


def parse_aws_params() -> dict:
    params = {}
    ami = os.environ.get("INPUT_AWS_IMAGE_ID")
    if ami is not None:
        params["image_id"] = ami
    instance_type = os.environ.get("INPUT_AWS_INSTANCE_TYPE")
    if instance_type is not None:
        params["instance_type"] = instance_type
    params = _env_parse_helper(params, "INPUT_AWS_SUBNET_ID", "subnet_id")
    params = _env_parse_helper(
        params, "INPUT_AWS_SECURITY_GROUP_ID", "security_group_id"
    )
    params = _env_parse_helper(params, "INPUT_AWS_IAM_ROLE", "iam_role")
    params = _env_parse_helper(params, "INPUT_AWS_TAGS", "tags", is_json=True)
    region_name = os.environ.get("INPUT_AWS_REGION_NAME")
    if region_name is not None:
        params["region_name"] = region_name
    home_dir = os.environ.get("INPUT_AWS_HOME_DIR")
    if home_dir is not None:
        params["home_dir"] = home_dir
    params = _env_parse_helper(params, "INPUT_EXTRA_GH_LABELS", "labels")
    return params
```

How hard it was to test

```python
@pytest.mark.parametrize(
    "env_vars, expected_output",
    [
        ({}, [{}]),
        (
            {"INPUT_AWS_IMAGE_ID": "ami-1234567890"},
            [{"image_id": "ami-1234567890"}],
        ),
        (
            {"INPUT_AWS_INSTANCE_TYPE": "t2.micro"},
            [{"instance_type": "t2.micro"}],
        ),
        (
            {"INPUT_AWS_SUBNET_ID": "subnet-1234567890"},
            [{"subnet_id": "subnet-1234567890"}],
        ),
        (
            {"INPUT_AWS_SECURITY_GROUP_ID": "sg-1234567890"},
            [{"security_group_id": "sg-1234567890"}],
        ),
        (
            {"INPUT_AWS_IAM_ROLE": "role-1234567890"},
            [{"iam_role": "role-1234567890"}],
        ),
        (
            {"INPUT_AWS_TAGS": '[{"Key": "Name", "Value": "test"}]'},
            [{"tags": [{"Key": "Name", "Value": "test"}]}],
        ),
        (
            {"INPUT_AWS_REGION_NAME": "us-east-1"},
            [{"region_name": "us-east-1"}],
        ),
        (
            {"INPUT_AWS_HOME_DIR": "/home/ec2-user"},
            [{"home_dir": "/home/ec2-user"}],
        ),
        ({"INPUT_EXTRA_GH_LABELS": "test"}, [{"labels": "test"}]),
        (
            {"INPUT_AWS_IMAGE_ID": "ami-1234567890"},
            [{"image_id": "ami-1234567890"}],
        ),
        (
            {
                "INPUT_AWS_IMAGE_ID": "ami-1234567890",
                "INPUT_AWS_INSTANCE_TYPE": "t2.micro",
            },
            [
                {"image_id": "ami-1234567890"},
                {"image_id": "ami-1234567890", "instance_type": "t2.micro"},
            ],
        ),
    ],
)
def test_parse_aws_params(env_vars, expected_output):
    idx = 0
    for key, value in env_vars.items():
        if key == "INPUT_AWS_TAGS":
            os.environ[key] = value
        else:
            os.environ[key] = str(value)
        assert parse_aws_params() == expected_output[idx]
        idx += 1
    for key in env_vars.keys():
        del os.environ[key]
```

How we fixed this problem

- Rethink the needs of input parsing
  - Need to be flexible, repeatable, and easily testable
  - Devs should be confident in their parsing they do
- Take this one step further and remove the need for AWS
  - The process of provisioning is repeatable, leverage it
  - Make the pieces that are always the same, testable

Start with how we want to parse

```python
@pytest.mark.parametrize(
    "env_vars, expected_output",
    [
        ({}, [{}]),
        (
            {"INPUT_AWS_IMAGE_ID": "ami-1234567890"},
            [{"image_id": "ami-1234567890"}],
        ),
        (
            {"INPUT_AWS_INSTANCE_TYPE": "t2.micro"},
            [{"instance_type": "t2.micro"}],
        ),
        (
            {"INPUT_AWS_SUBNET_ID": "subnet-1234567890"},
            [{"subnet_id": "subnet-1234567890"}],
        ),
        (
            {"INPUT_AWS_SECURITY_GROUP_ID": "sg-1234567890"},
            [{"security_group_id": "sg-1234567890"}],
        ),
        (
            {"INPUT_AWS_IAM_ROLE": "role-1234567890"},
            [{"iam_role": "role-1234567890"}],
        ),
        (
            {"INPUT_AWS_TAGS": '[{"Key": "Name", "Value": "test"}]'},
            [{"tags": [{"Key": "Name", "Value": "test"}]}],
        ),
        (
            {"INPUT_AWS_REGION_NAME": "us-east-1"},
            [{"region_name": "us-east-1"}],
        ),
        (
            {"INPUT_AWS_HOME_DIR": "/home/ec2-user"},
            [{"home_dir": "/home/ec2-user"}],
        ),
        ({"INPUT_EXTRA_GH_LABELS": "test"}, [{"labels": "test"}]),
        (
            {"INPUT_AWS_IMAGE_ID": "ami-1234567890"},
            [{"image_id": "ami-1234567890"}],
        ),
        (
            {
                "INPUT_AWS_IMAGE_ID": "ami-1234567890",
                "INPUT_AWS_INSTANCE_TYPE": "t2.micro",
            },
            [
                {"image_id": "ami-1234567890"},
                {"image_id": "ami-1234567890", "instance_type": "t2.micro"},
            ],
        ),
    ],
)
def test_parse_aws_params(env_vars, expected_output):
    idx = 0
    for key, value in env_vars.items():
        if key == "INPUT_AWS_TAGS":
            os.environ[key] = value
        else:
            os.environ[key] = str(value)
        assert parse_aws_params() == expected_output[idx]
        idx += 1
    for key in env_vars.keys():
        del os.environ[key]
```

```python
def test_env_builder():
    env = {}
    env["INPUT_AWS_IMAGE_ID"] = "ami-1234567890"
    env["INPUT_AWS_INSTANCE_TYPE"] = "t2.micro"
    env["INPUT_GH_REPO"] = "owner/test"
    env["GITHUB_REPOSITORY"] = "owner/test_other"
    env["INPUT_INSTANCE_COUNT"] = "1"
    env["INPUT_AWS_TAGS"] = '{"Key": "Name", "Value": "test"}'
    builder = (
        EnvVarBuilder(env)
        .update_state("INPUT_AWS_IMAGE_ID", "image_id")
        .update_state("INPUT_AWS_INSTANCE_TYPE", "instance_type")
        .update_state("GITHUB_REPOSITORY", "repo")
        .update_state("INPUT_GH_REPO", "repo")
        .update_state("INPUT_INSTANCE_COUNT", "instance_count", type_hint=int)
        .update_state("INPUT_AWS_TAGS", "tags", is_json=True)
    )
    config = builder.params
    assert config["image_id"] == "ami-1234567890"
    assert config["instance_type"] == "t2.micro"
    assert config["repo"] == "owner/test"
    assert config["instance_count"] == 1
    assert isinstance(config["instance_count"], int)
    assert config["tags"] == {"Key": "Name", "Value": "test"}
    assert isinstance(config["tags"], dict)
```

Implementation change

```python
def _env_parse_helper(
    params: dict, var: str, key: str, is_json: bool = False
) -> dict:
    val = os.environ.get(var)
    if val is not None and val != "":
        if is_json:
            params[key] = json.loads(val)
        else:
            params[key] = val
    return params


def parse_aws_params() -> dict:
    params = {}
    ami = os.environ.get("INPUT_AWS_IMAGE_ID")
    if ami is not None:
        params["image_id"] = ami
    instance_type = os.environ.get("INPUT_AWS_INSTANCE_TYPE")
    if instance_type is not None:
        params["instance_type"] = instance_type
    params = _env_parse_helper(params, "INPUT_AWS_SUBNET_ID", "subnet_id")
    params = _env_parse_helper(
        params, "INPUT_AWS_SECURITY_GROUP_ID", "security_group_id"
    )
    params = _env_parse_helper(params, "INPUT_AWS_IAM_ROLE", "iam_role")
    params = _env_parse_helper(params, "INPUT_AWS_TAGS", "tags", is_json=True)
    region_name = os.environ.get("INPUT_AWS_REGION_NAME")
    if region_name is not None:
        params["region_name"] = region_name
    home_dir = os.environ.get("INPUT_AWS_HOME_DIR")
    if home_dir is not None:
        params["home_dir"] = home_dir
    params = _env_parse_helper(params, "INPUT_EXTRA_GH_LABELS", "labels")
    return params
```



```python
    def _parse_single_param(self, config: ParamConfig):
        """Parse a single parameter from the environment variables."""
        value = self.env.get(config.env_var)
        if value is not None and (config.allow_empty or value.strip()):
            parsed_value = self._parse_value(
                value, config.is_json, config.type_val
            )
            self._update_params(config.key, parsed_value)

    def _update_params(self, key: str, value: Any):
        self._params = deepcopy(self._params)
        self._params[key] = deepcopy(value)

    def update_state(
        self,
        var_name: str,
        key: str,
        is_json: bool = False,
        allow_empty: bool = False,
        type_hint: Type = str,
    ) -> "EnvVarBuilder":
        """Update the state of the builder with a single parameter.

        Returns
        -------
        EnvVarBuilder
            Returns self for method chaining

        Raises
        ------
        ValueError
            If any configured environment variable parsing fails

        Notes
        -----
        - Empty strings are ignored by default unless allow_empty is True
        - JSON parsing is performed before type conversion if is_json is True

        """
        config = ParamConfig(var_name, key, is_json, allow_empty, type_hint)

        self._parse_single_param(config)
        return self

    @property
    def params(self) -> dict:
        """Returns a copyt of the dictionary of parsed parameters."""
        return deepcopy(self._params)
```

Reduce implementation work

```
builder = (
    EnvVarBuilder(env)
    .update_state("INPUT_AWS_IMAGE_ID", "image_id")
    .update_state("INPUT_AWS_INSTANCE_TYPE", "instance_type")
    .update_state("INPUT_AWS_SUBNET_ID", "subnet_id")
    .update_state("INPUT_AWS_SECURITY_GROUP_ID", "security_group_id")
    .update_state("INPUT_AWS_IAM_ROLE", "iam_role")
    .update_state("INPUT_AWS_TAGS", "tags", is_json=True)
    .update_state("INPUT_EXTRA_GH_LABELS", "labels")
    .update_state("INPUT_AWS_HOME_DIR", "home_dir")
    .update_state("INPUT_INSTANCE_COUNT", "instance_count", type_hint=int)
    .update_state(
        "INPUT_AWS_ROOT_DEVICE_SIZE", "root_device_size", type_hint=int
    )
    .update_state("INPUT_ARCHITECTURE", "arch")
    # This is the default case
    .update_state("AWS_REGION", "region_name")
    # This is the input case
    .update_state("INPUT_AWS_REGION_NAME", "region_name")
    # This is the default case
    .update_state("GITHUB_REPOSITORY", "repo")
    # This is the input case
    .update_state("INPUT_GH_REPO", "repo")
)
```

```
def _modify_root_disk_size(self, client, params: dict) -> dict:
    """ Modify the root disk size of the instance.

    Parameters
    ----------
    client
        The EC2 client object.
    params : dict
        The parameters for the instance.

    Returns
    -------
    dict
        The modified parameters

    Raises
    ------
    botocore.exceptions.ClientError
       If the user does not have permissions to describe images.
    """
    try:
        client.describe_images(ImageIds=[self.image_id], DryRun=True)
    except ClientError as e:
        # This is the case where we DO have access
        if "DryRunOperation" in str(e):
            image_options = client.describe_images(ImageIds=[self.image_id])
            root_device_name = image_options["Images"][0]["RootDeviceName"]
            block_devices = deepcopy(image_options["Images"][0]["BlockDeviceMappings"])
            for idx, block_device in enumerate(block_devices):
                if block_device["DeviceName"] == root_device_name:
                    if self.root_device_size > 0:
                        block_devices[idx]["Ebs"]["VolumeSize"] = self.root_device_size
                        params["BlockDeviceMappings"] = block_devices
                    break
        else:
            raise e
    return params
```

Where we failed

- We did not have sufficient separation of concerns to make behavior repeatable
- Our integration tests were not kept up to date and did not cover the functionality we claimed to support
  - We claim to support arm64 but did not test this between migrations
  - Recently fixed this change

What we achieved

- Split code functionality into common functionality and actions
  - Two GitHub Actions
  - A “standard” library for working with GitHub
  - Remove dependency on PyGitHub to make mocking simpler
- Systematic removal of complexity in testing
  - Does this test our code or does it spend most of its time mocking functionality?
- Simpler and more extensible code that is easily tested
  - Testable code is extensible code

Doing this in your own project

- Stick to user-defined norms when you can
- Reduce duplication
- Rely on your testing to do the heavy lifting
- Write the tests based on how your users are using your code
- Limit mocking by extracting the functionality you need
    - Mock infrastructure, not libraries

```yaml
start-aws-runner:
  runs-on: ubuntu-latest
  outputs:
    mapping: ${{ steps.aws-start.outputs.mapping }}
    instances: ${{ steps.aws-start.outputs.instances }}
  steps:
    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v4
      with:
        role-to-assume: ${{ secrets.AWS_ROLE }}
        aws-region: us-east-1
    - name: Create cloud runner
      id: aws-start
      uses: omsf-eco-infra/gha-runner@main
      with:
        provider: "aws"
        action: "start"
        aws_region_name: us-east-1
        aws_image_id: ami-0f7c4a792e3fb63c8
        aws_instance_type: ${{ inputs.instance_type }}
        aws_home_dir: /home/ubuntu
      env:
        GH_PAT: ${{ secrets.GH_PAT }}
```

```yaml
jobs:
  start-aws-runner:
    runs-on: ubuntu-latest
    permissions:
      id-token: write
      contents: read
    outputs:
      mapping: ${{ steps.aws-start.outputs.mapping }}
    steps:
      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: ${{ secrets.AWS_ROLE }}
          aws-region: us-east-1
      - name: Create cloud runner
        id: aws-start
        uses: omsf/start-aws-gha-runner@v1.0.0
        with:
          aws_image_id: ami-0d5079d9be06933e5
          aws_instance_type: g4dn.xlarge
          aws_home_dir: /home/ubuntu
        env:
          GH_PAT: ${{ secrets.GH_PAT }}
```

Questions?

Interested in learning more about cloud?

I am hosting a cloud office hours session tomorrow at 12 PM Mountain about cloud compute for comp chem/comp bio researchers but will also be suitable for RSEs! This is aimed to be high level to get people familiar with cloud compute terminology! Check the US-RSE #events channel to register!