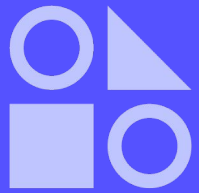


Covalent.

agnostiq

covalent.xyz

Santosh K. Radha, PhD

Head of Product

Ara Ghukasyan, PhD

Research Software Engineer

Rethinking for the Compute Era.

Covalent.

A simple, scalable, and performant platform designed for high-compute

Unify all hardware & clouds

Make all of your hardware, clouds, and on-premises resources accessible with a single line of Python code

Heterogeneous hardware & software

Covalent supports workflows that leverage a diverse range of hardware & software.

(Covalent Cloud)

Billing, user management & administration

Gain out-of-the-box cloud capabilities such as teams & organization-level management, billing & cost management, authentication, RBAC and more

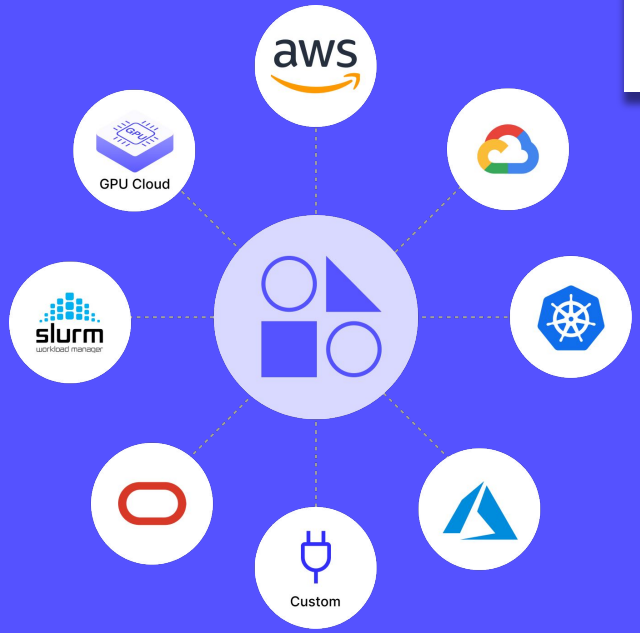
Multi-cloud mobility.

with a single line of code

documentation

Covalent.

```
Untitled-1

def train_svm(data, C, gamma):
    X, y = data
    clf = svm.SVC(C=C, gamma=gamma)
    clf.fit(X,y)
    return clf
```



....w.covalent.xyz

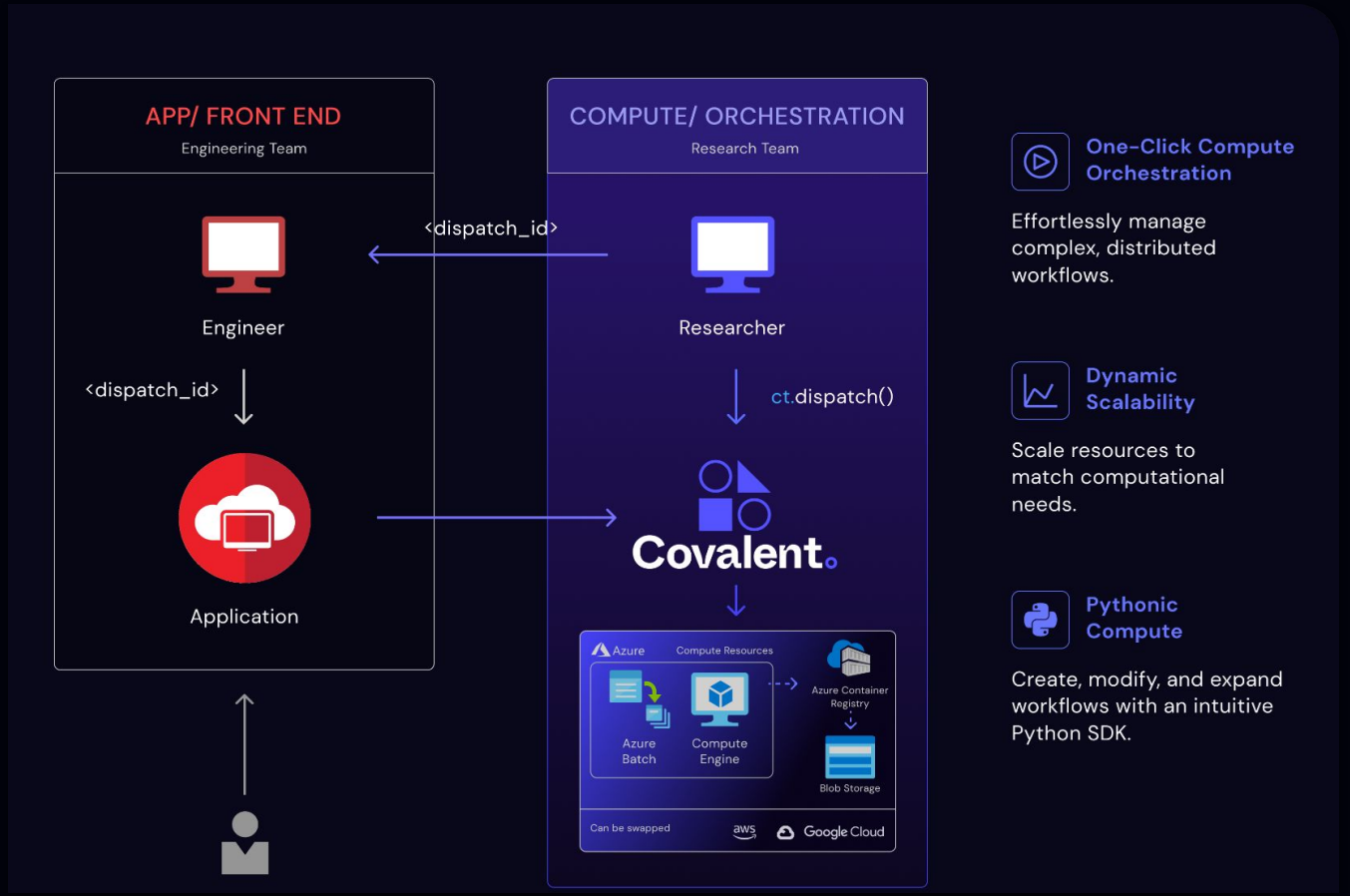
Customize *executors* to run tasks wherever

```
awsbatch_exec = AWSBatchExecutor(
    vcpu=64,
    num_gpus=4,
    memory=16,
    time_limit=7200,
)

@ct.electron(awsbatch_exec, deps_pip=["qiskit==0.43.1"])
def classical_compute(**data):
    # your code here
    ...
```

```
slurm_exec = SlurmExecutor(
    username = "user",
    address = "login.cluster.org",
    ssh_key_file = "/home/user/.ssh/id_rsa",
    options={
        "qos": "regular",
        "time": "01:30:00",
        "nodes": 1,
        "constraint": "gpu",
    },
    prerun_commands=[
        "module load package/1.2.3"
    ],
    postrun_commands=[
        "cp * ~/mount_dir/user_account -r"
    ]
)
```

Covalent as a backend for high-compute applications

- Orchestrate large scale computations
- Track, manage, and share computation results



One-Click Compute Orchestration

Effortlessly manage complex, distributed workflows.

Dynamic Scalability

Scale resources to match computational needs.

Pythonic Compute

Create, modify, and expand workflows with an intuitive Python SDK.

Why does Covalent exist?

Computational research

Time spent on...

Writing
Debugging
Executing
Maintaining

Time spent on research

Where Covalent fits in the (Python) stack

Covalent supports a growing ecosystem of hardware and software

Classical Resources

- Slurm executor
- AWS Fargate executor
- AWS Batch executor
- Azure executor
- GCP executor
- Kubernetes executor

Software Packages

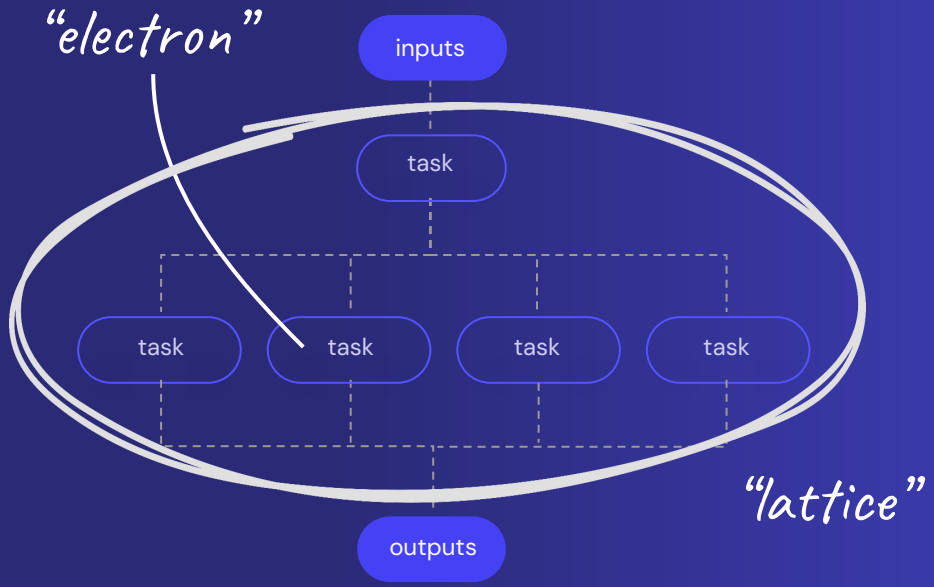
Any and all packages!

PyTorch
RAPIDS
DASK
TensorFlow
Hugging Face
Keras
pandas
scikit-learn

Languages

python™
BASH
julia

Concepts review



Electrons

Granular tasks that comprise a workflow

```python
@ct.electron
def task(x):
    ...
    # your code here
```

Lattice

workflow defined by task interactions

```python
@ct.lattice
def workflow(inputs):
    outputs = []
    for x in inputs:
        outputs.append(task(x))
    return outputs
```

Dispatch

Async deploy, track and reproduce workflows

```python
dispatch_id = ct.dispatch(workflow)(inputs)
```

Covalent

Tutorial Notebooks

🔗 **https://github.com/AgnostiqHQ/tutorials_covalent_pydata_2023.git**

Notebook 1: Covalent Quickstart

Covalent Quickstart

Using Covalent means interacting with four main elements.

Two decorators for *creating workflows*:

- `@ct.electron` - designates workflow tasks
- `@ct.lattice` - designates the main workflow function

and two functions for *running workflows*:

- `ct.dispatch()` - dispatches the workflow
- `ct.get_result()` - retrieve a workflow result

```
1  import covalent as ct
2
3  # Construct manageable tasks out of functions
4  # by adding the @ct.electron decorator
5  @ct.electron
6  def add(x, y):
7      return x + y
8
9  @ct.electron
10 def multiply(x, y):
11     return x*y
12
13 # Note that electrons can be shipped to variety of compute
14 # backends using executors, for example, "local" computer.
15 # See below for other common executors.
16 @ct.electron(executor="local")
17 def divide(x, y):
18     return x/y
19
20 # Construct the workflow by stitching together
21 # the electrons defined earlier in a function with
22 # the @ct.lattice decorator
23 @ct.lattice
24 def workflow(x, y):
25     r1 = add(x, y)
```

3. Using Covalent w/ a remote Dask Cluster

Notebook 2: Covalent + Dask + AWS

TUTORIAL: Automatic Image Captioning

+ Code + Markdown

Imports

```python
import os
from dataclasses import dataclass
from datetime import datetime
from typing import List

import covalent as ct
import torch
from PIL import Image
from transformers import BlipForConditionalGeneration, BlipProcessor

DASK_SCHEDULER_ADDRESS = "<paste-from-terraform-output>"
```

[1] Python

Constants

```python
ROW_BLANK = "{datetime} | {image_name:<30} | {description}\n"

TRIGGER_DIR = os.path.abspath("./images-triggered")
CATALOG = os.path.join(TRIGGER_DIR, "catalog.txt")

if not os.path.exists(CATALOG):
    with open(CATALOG, "w") as f:
        f.write('')
```

[3] Python

Utilities

```python
def get_model(type : object, id : str, label=None, **params):
```

4. LLMs with Covalent and AWS Batch

Notebook 3: Covalent w/ LLMs and Stable Diffusion

TUTORIAL: AI Image Generator for Wikipedia Articles

+ Code + Markdown

Imports

```python
import re
from collections import OrderedDict
from pathlib import Path
from typing import Any, List

import covalent as ct
import torch
from diffusers import StableDiffusionPipeline
from transformers import T5ForConditionalGeneration, T5Tokenizer
from wikipediaapi import Wikipedia
```

[1] Python

```python
# Declare some constants.
MAX_LENGTH_PROMPT = 77
INPUT_BLANK = "summarize: {text}"  # format prompt text into this
TOKENIZER_MODEL = "t5-small"
```

[2] Python

First, some useful helper code...

Post processor class that will render the outputted images in this Jupyter Notebook.

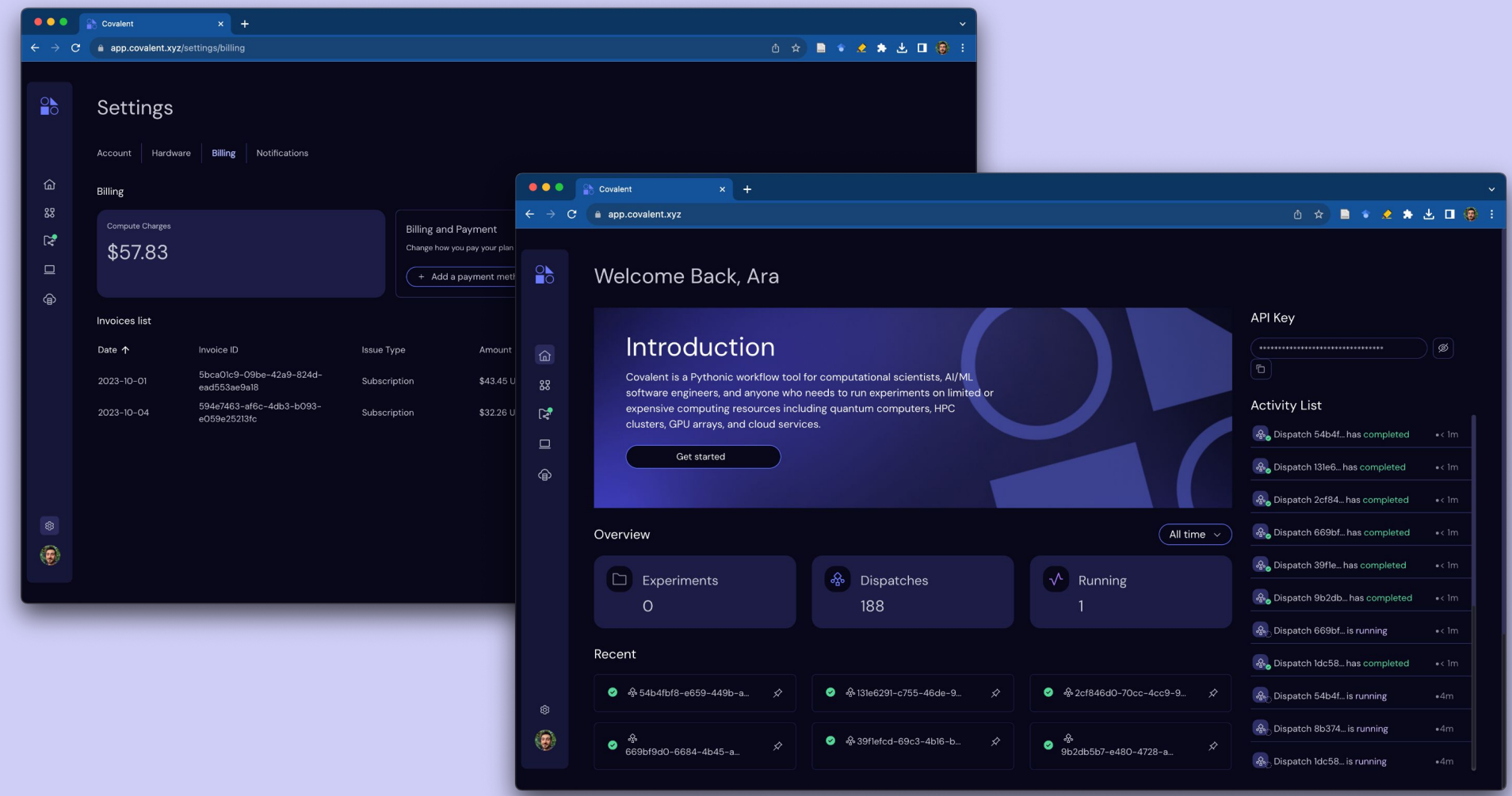
```python
class PostProcessor:
    """Post-process the generated images and summaries."""

    # Regex pattern for splitting sentences.
```

5. LLMs with Covalent Cloud

Covalent Cloud

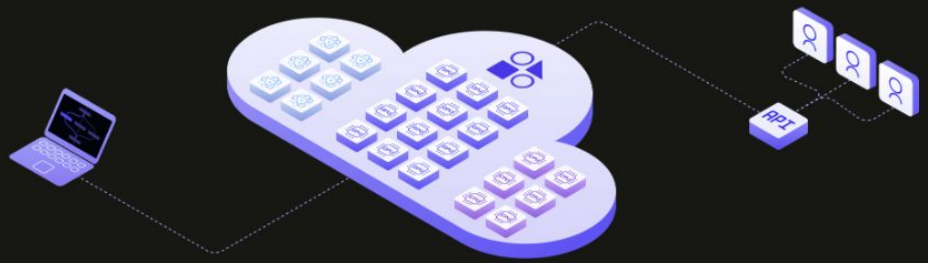
agnostiq

A fully-managed platform for AI & HPC


Covalent
app.covalent.xyz/settings/billing
Settings
Account
Hardware
Billing
Notifications
Billing
Compute Charges
$57.83
Billing and Payment
Change how you pay your plan
+ Add a payment meth
Invoices list
Date
Invoice ID
Issue Type
Amount
2023-10-01
5bca01c9-09be-42a9-824d-ead553ae9a18
Subscription
$43.45 U
2023-10-04
594e7463-af6c-4db3-b093-e059e25213fc
Subscription
$32.26 U
Covalent
app.covalent.xyz
Welcome Back, Ara
Introduction
Covalent is a Pythonic workflow tool for computational scientists, AI/ML software engineers, and anyone who needs to run experiments on limited or expensive computing resources including quantum computers, HPC clusters, GPU arrays, and cloud services.
Get started
API Key
Activity List
Dispatch 54b4f... has completed
Dispatch 131e6... has completed
Dispatch 2cf84... has completed
Dispatch 669bf... has completed
Dispatch 39f1e... has completed
Dispatch 9b2db... has completed
Dispatch 669bf... is running
Dispatch 1dc58... has completed
Dispatch 54b4f... is running
Dispatch 8b374... is running
Dispatch 1dc58... is running
Overview
All time
Experiments
0
Dispatches
188
Running
1
Recent
54b4fbf8-e659-449b-a...
131e6291-c755-46de-9...
2cf846d0-70cc-4cc9-9...
669bf9d0-6684-4b45-a...
39f1efcd-69c3-4b16-b...
9b2db5b7-e480-4728-a...

Notebook 3: Let's try it on Covalent Cloud

Fully Managed workflow orchestration with *Covalent Cloud*.



Usage

Covalent Cloud provides a very similar interface to open-source Covalent.

👉 Add one or more @ct.electron decorators to designate workflow tasks (i.e. *electrons*).

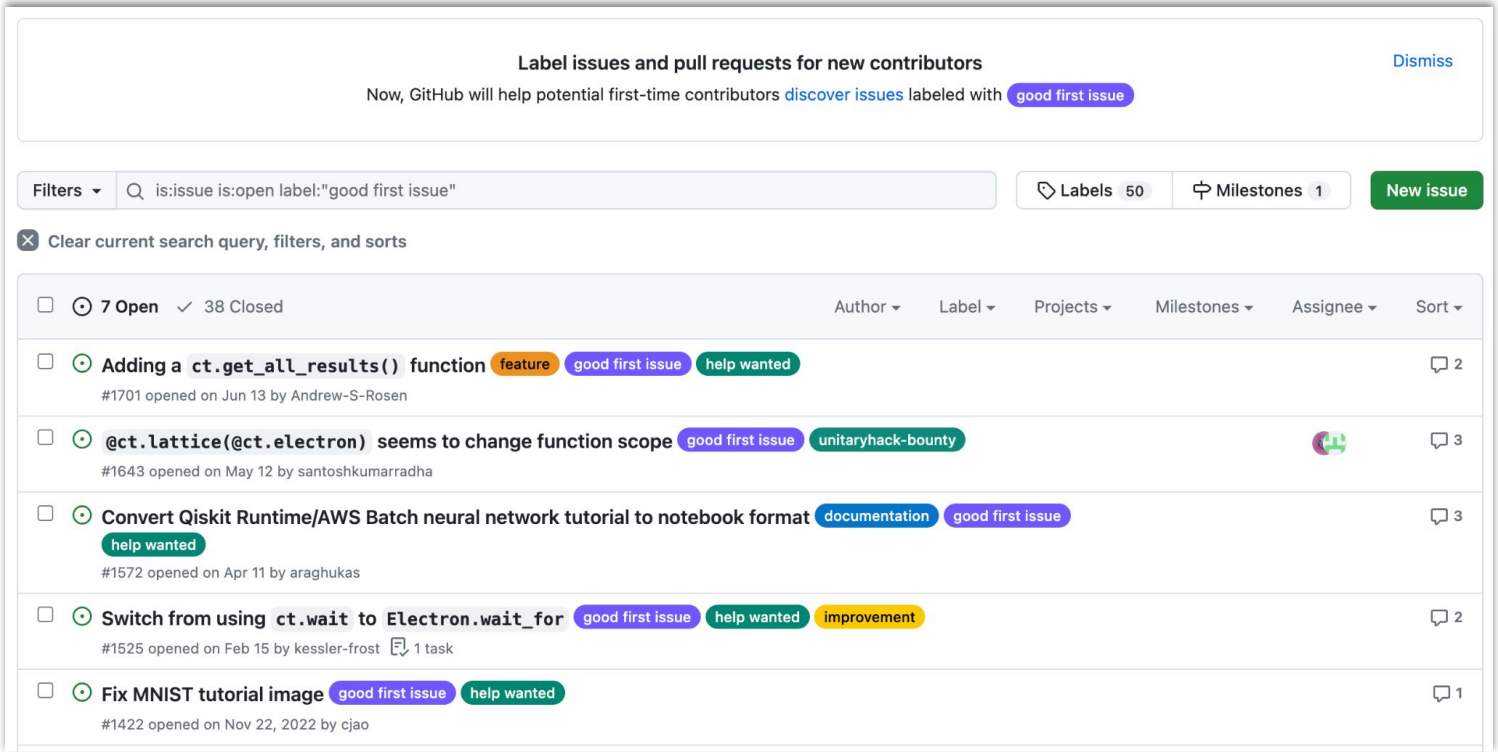
👉 Specify executors to choose electron *compute resources*, without worrying about provider-specific details.

Covalent Cloud Highlights

- ☁️ Complete cloud abstraction. No infrastructure setup, no provider credentials required.
- 🤝 Support for multiple users and enterprises.
- 🧾 Unified billing and usage tracking.

You can contribute too!

🔗 github.com/AgnostiqHQ/covalent/issues



Covalent is the simplest way to orchestrate high-compute workflows across heterogeneous computing environments

agnostiqHQ/covalent 🌟

covalent.xyz

@covalentxyz