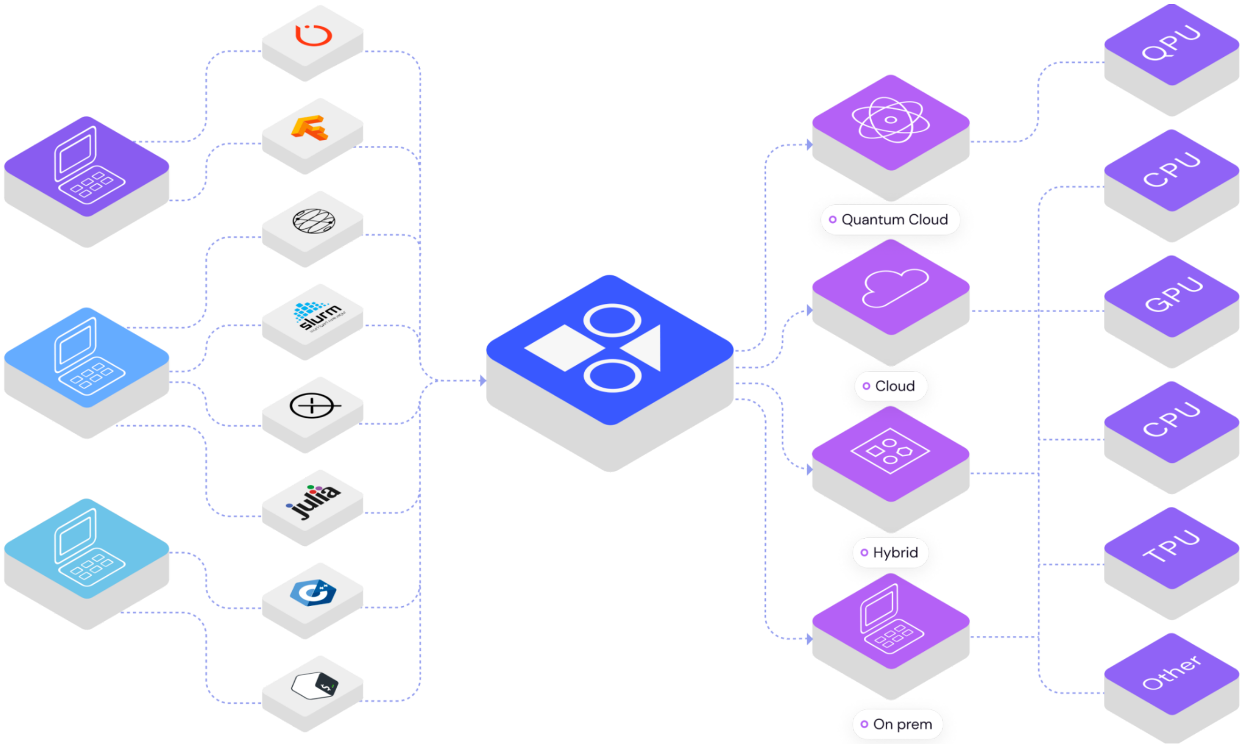
Covalent

# Navigating the Modern HPC Landscape

*The HPC landscape is larger, more complex, and more interconnected than ever before. Covalent provides a single interface for users to prototype and scale heterogeneous solutions on any combination of hardware – from a laptop to a supercomputer – and everything in between.*

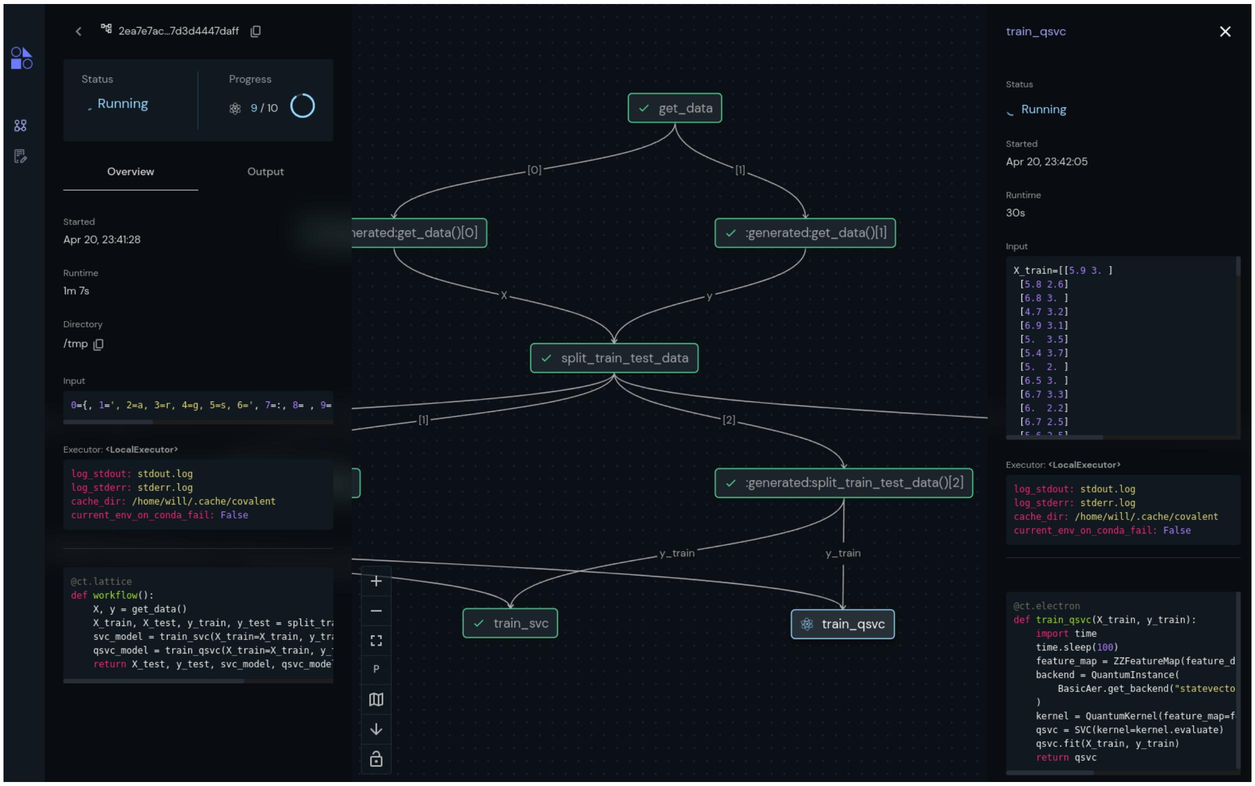


The introduction of quantum computers has provided a new and fresh perspective on what "High Performance Computing" really means.  While today's quantum computers can hardly be described as "high performance," their interactions with high-performance classical compute resources has revitalized discussions around the importance of well-orchestrated heterogeneous distributed computing. On the classical side, we've already seen the introduction of exascale systems such as Fugaku, while in the quantum space many end-users are already asking how novel quantum or quantum-inspired devices may augment the power of the best supercomputers. Moreover, since most users experience quantum computers on a cloud platform, such as AWS Braket, IBMQ, or Azure Quantum, it is increasingly common for users to begin to think about how the cloud plays a broader role in HPC. These trends – the emergence of quantum computing, the availability of HPC on the cloud, and the development of exascale supercomputers – together paint a very compelling picture for the future of HPC.

As exciting as this may seem, for users it can be information overload. Forgetting the challenges of using a single supercomputer for a moment, imagine the challenges one might face using a variety of devices. In one example application, a user performs some data validation using machine learning with a set of GPUs, then performs an optimization algorithm using a multi-core machine interacting with a quantum annealer, and then uses their local laptop to visualize results. In this somewhat simple example, the user already needs to manage a multi-stage workflow, resubmit jobs when hardware fails, and manage software environments and data transfers. Needless to say, the user will also want the flexibility to reproduce their results and swap out hardware when possible. If this user also has access to multiple supercomputers, or cloud compute resources, they may be interested in rerouting tasks when queue times increase. Ultimately, this user may end up asking themselves, “Is this really the most efficient use of my time and money?”

At Agnostiq, we faced these same challenges in our research on novel quantum machine learning applications. As a startup, we already knew that properly managing time and money in research endeavors can mean the difference between success and failure, which is why we started building a tool to make workflow management easy and understandable. Earlier this year we were excited to release it on GitHub as a free, open-source platform called Covalent. Covalent is a workflow orchestration tool which allows users to manage heterogeneous applications at scale. It serves as a single point-of-entry for the broader computing landscape, including supercomputers and quantum computers, as well as cloud-based HPC, on-prem infrastructure, and even low- and general-compute resources. Covalent allows users to flexibly send collections of tasks to *almost any* combination of compute backends without investing too heavily in any one option. In an era when computing technology is as fluid as it is now, users need tools which allow them to respond quickly to new devices and platforms.

Let’s dig a little deeper into what Covalent offers. After modularizing their Python code, users can wrap their functions using simple one-line Python decorators. These decorators tell the Covalent server how and where to execute the task. When functions are stitched together as a workflow, they form a dependency graph which describes how information flows among the tasks. Then, workflows can themselves become tasks in a larger workflow, thereby enabling users to create ever larger and more complex workflows using standardized and trusted algorithmic building blocks. Users can then view their workflows as dependency graphs in a rich browser-based user interface. Within each graph, users can study execution metrics such as execution time and software environments, view cached inputs and outputs, trace error messages to their origins, and even inspect source code. Together, this information provides a powerful, holistic view of a distributed workflow in a digestible format.



Status
Running

Progress
9 / 10

Started
Apr 20, 23:41:28

Runtime
1m 7s

Directory
/tmp

Input

```
0={, 1=', 2=a, 3=r, 4=g, 5=s, 6=', 7=:, 8= , 9=
```

Executor: <LocalExecutor>

```
log_stdout: stdout.log
log_stderr: stderr.log
cache_dir: /home/will/.cache/covalent
current_env_on_conda_fail: False
```

```
@ct.lattice
def workflow():
    X, y = get_data()
    X_train, X_test, y_train, y_test = split_tr
    svc_model = train_svc(X_train=X_train, y_tr
    qsvc_model = train_qsvc(X_train=X_train, y_
    return X_test, y_test, svc_model, qsvc_mode
```

train_qsvc

Status
Running

Started
Apr 20, 23:42:05

Runtime
30s

Input

```
X_train=[[5.9 3. ]
 [5.8 2.6]
 [6.8 3. ]
 [4.7 3.2]
 [6.9 3.1]
 [5.  3.5]
 [5.4 3.7]
 [5.  2. ]
 [6.5 3. ]
 [6.7 3.3]
 [6.  2.2]
 [6.7 2.5]
```

Executor: <LocalExecutor>

```
log_stdout: stdout.log
log_stderr: stderr.log
cache_dir: /home/will/.cache/covalent
current_env_on_conda_fail: False
```

```
@ct.electron
def train_qsvc(X_train, y_train):
    import time
    time.sleep(100)
    feature_map = ZZFeatureMap(feature_d
    backend = QuantumInstance(
        BasicAer.get_backend("statevecto
    )
    kernel = QuantumKernel(feature_map=f
    qsvc = SVC(kernel=kernel.evaluate)
    qsvc.fit(X_train, y_train)
    return qsvc
```

*Covalent's graphical user interface*

Covalent can also be used in combination with a variety of other software tools in the distributed computing stack. Users may already be familiar with other Pythonic tools lower in the stack used to accelerate computations. Joblib, Numba, and PyCUDA are examples of Python tools which can be leveraged to parallelize applications over multiple cores. Likewise, packages such as Dask, Rapids, and PySpark can be leveraged to distribute parallel applications over multiple devices. Users in HPC may be familiar with the "OpenMP + MPI" experience, which achieves the same effect at a more granular level. At one layer above in the distributed computing stack, we find enterprise-grade workflow orchestration tools, such as Airflow, and Luigi. These tools are used in large-scale enterprise machine learning and data analytics applications, where certain tasks must run on a time-based schedule without fail. Covalent sits at the layer in between, which is why we describe it as a "distributed workflow" tool. In terms of practical differences in software design, *workflow instances* are the primary objects rather than *workflow definitions*. This core design principle enables the type of rapid iteration needed for pre-production research workflows, while remaining compatible with tools at the other layers in the stack.

Covalent is freely available and open-source on GitHub and we strongly encourage contributions from HPC practitioners, software engineers, and enthusiasts alike. Get started today by trying it out for yourself, checking out the documentation, or visiting our website at www.covalent.xyz. If you are an enterprise partner looking for a more bespoke implementation of Covalent, please reach out to sales@agnostiq.ai.