Project Goal

This project is focused on making several improvements to the cloud based applicationOpenStudio-server that is used by building energy modelers.

Application background

OpenStudio is a cross-platform (Windows, Mac, and Linux) collection of software tools to support whole building energy modeling using EnergyPlus and advanced daylight analysis usingRadiance. OpenStudio is an open source project to facilitate community development, extension, and private sector adoption. OpenStudio includes graphical interfaces along with a Software Development Kit (SDK)

OpenStudio-server is an energy modeling framework that uses OpenStudio and EnergyPlus to run whole distributed building energy modeling simulations in the cloud. Currently, there are some challenges associated with the setup as discussed below. This project is focused on addresses these by optimizing and incorporating concepts and open-sourced tools learned from CSCI-7000 DataCenter and Cloud Computing course @ CU Boulder.

Challenges

1) The current framework runs in a containerized platform - Docker Swarm. The cluster setup is manually intensive as many SDN and IaaC steps are hard-wired into ruby client code to configure, provision, a deploy the cluster setup for Docker Swarm. Furthermore, the current Swarm arch only allows a fixed number of nodes making it difficult for users to determine the optimal cluster size to balance costs and processing time.

2) The current application back-end HTTP API suffers from TCP traffic bottlenecks from the worker node responses. Each worker node returns a large data set, and since many work nodes can send responses simultaneously, it can saturate the link and create very high latent response times and even induce hung processes.

3) Due to the issue discussed on #2, raw, high-resolution model output was never designed to be included as part of the output as this data is several times larger than the coarser model output and too intensive to be integrated for end-user post -processing. Although this would be highly beneficial to provide access to the high-resolution model output, up until now, there has been no way to address latency issues to make this as usable feature.

Potential Solutions

1) Upgrade the current dockerized deployment using Kubernetes orchestration framework. Since the application currently is using Ruby, use a Ruby Kubernetes client to handle the provisioning, deployment, and tear-down of the cluster. Kubernetes offers auto-scaling features that can be triggered by CPU utilization. This should be implemented at the worker nodes which can then auto-scale dynamically based on the model runs.

2) Several enhancements could be made to optimize the API. The goal is to evaluate using an overlay by-pass framework (Slim or FreeFlow) to determine the efficiency gains in terms of network throughput and latency. Simulate model runs on the existing Kubernetes framework that implements a container overlay

network vs FreeFlow that by-passes the overlay network stack.

Dataset or tools

Deploying FreeFlow would require RDMA enabled NICs, so having a cloud environment testbed such as CloudLab would be important to test and measure the performance gains. In addition to having a suitable cloud hardware, several container network overlay by-pass frameworks exist:

https://github.com/danyangz/Slim

https://github.com/microsoft/Freeflow