Project Goal

This project is focused on making several improvements to the cloud based application [OpenStudio-server](https://github.com/NREL/OpenStudio-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](https://github.com/NREL/EnergyPlus) and advanced daylight analysis using [Radiance](https://github.com/NREL/Radiance/). 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

Timeline

Now that the OpenStudio-server deployment configs have been updated to run within a k8s environment, I can begin testing out RDMA and Slim in attempt to improve network I/O. Within the semester ends I should have enough time to get one of the two options (Freeflow and/or Slim) to evaluate performance improvements it relates to network latency, throughput and resource (cpu) consumption.

Concerns

The main concern is finding a cloud provider that offers RDMA enabled NICs to run implement FreeFlow. Slim requires access to the underlining VM, which I believe can be done using k8s provisioned VMs and

then making modifications. If that doesn't work I might have to use custom VMs and install k8s manually to implement Slim.

CheckPoint 1

Added sections Timeline and Concerns to README.md as these were missing. Sections Challenges was already in the project proposal.
https://github.com/CU-CSCI7000-Fall2019/final-project-csci7000-container-network-op/commit/e9ffb14600743e288635b9d217217a84ffa3670b

Added Checklist as a GitHub issue to track progress.
https://github.com/CU-CSCI7000-Fall2019/final-project-csci7000-container-network-op/issues/1

Concerns

Timeline should stay on course. Now that the OpenStudio-server deployment configs have been updated to run within a k8s environment, I can begin testing out RDMA and Slim in various cloud envs to attempt to improve network I/O. By semester ends I should have enough time to get one of the two options (Freeflow and/or Slim) to evaluate performance improvements as it relates to network latency, throughput and resource (cpu) consumption.

Additional Concerns

My initial look at cloud offerings on AWS and Google I did not see VM instances that have RDMA enabled NICs. I also looked through the Cloud hardware on CloudLab and didn't see RDMA support either. I will look into Microsoft Azure, but if no Cloud Providers offer RDMA NIC then testing FreeFlow may not be possible. Slim should be okay given that it modifies the VM itself which should be doable.

Evaluate and Validate project

Evaluation will be done by measuring network performance improvements as it pertains to running energy modeling simulations on both the standard k8s setup (no optimizations) vs running on FreeFlow and/or Slim and measuring the performance gain as it pertains to network throughput, latency and resource utilization.

Checkpoint 2

Changes - The purpose of this project is to test and evaluate high-performance network techniques as it relates to container-based workflows. Thus far, the work has been around getting Microsoft freeflow (https://github.com/microsoft/Freeflow) to work in a cloud environment, and as stated earlier, Freeflow works by using RDMA enabled NICs to offer higher throughput, reduced latency and reduced compute resources. As of now, no change has been made to the project as Microsoft Azure offers VM instances that have RDMA enabled NICs. But it is also worth noting, and possible investigating even after this project is over, that Amazon offers SR-IOV capable instances and these allow for containers to communicate directly to the hardware NIC. SR-IOV could offer similar advantages as RDMA and potential easier to deploy as not all node instances would require RDMA. This could actually reduce costs as only select nodes in a Kubernetes cluster pool would need this type of instances (e.g. NFS, HTTP API) that consume a large portion of the application traffic. I will mark this as my fall back option to explore in the event RDMA is not successful.

TimeLine

I have an account on Microsoft Azure that offers instances of RDMA NICs (see instances types below). Unfortunately, Microsoft's free student credits do not allow me to access these types of instances, so I needed to purchase compute time to complete this project on Azure. They also limit you to 8 total cores under the student account which wouldn't work anyway so I have to upgrade to test out OpenStudio-server regardless.

The tests to be conducted will be the following instance type on Azure.

RDMA Enabled NICs: ResourceType: ComputeCores SKU: HC Series (H16r,) x2

Non-RDMA Enabled NICs: ResourceType: ComputeCores SKU: HC Series (H16,) x2

⊕	H16r	16	112 GiB	2,000 GiB	$1.75/hour	$1.1834/hour (~32%)	$0.7767/hour (~56%)	$0.35/hour (~80%)
⊕	H16	16	112 GiB	2,000 GiB	$1.591/hour	$1.0757/hour (~32%)	$0.7061/hour (~56%)	$0.318/hour (~80%)

The only difference between H16 vs H16r is the 'r' has the RDMA enabled NIC so the experiment will evaluate the performance difference between these two instance types. This should be a fair comparison as the other hardware is exactly the same.

The Kubernetes cluster runs in several containers / components (below). Since this will be ran in a multiple VM node cluster, it'll be important to tag containers to nodes as otherwise these will be randomly distributed based on compute available by the kubernetes engine. You could end up with different results, say for instance, the NFS is scheduled on the same node as the Web back-end API vs scheduling them on different nodes.

- Mongo-db
- Web (OpenStudio-Server) (nginx front-end)
- Web-Background (OpenStudio-Server) (rails back-end)
- NFS server for persistent shared storage
- Redis
- Worker (runs batch EnergyPlus simulations)
- Rserver (runs algorithmic simulations)

I had to modify the existing Kubernetes deployment configs as there are differences in how each cloud provider manages persistent volumes. These changes have been checked in to the repo.
https://github.com/CU-CSCI7000-Fall2019/final-project-csci7000-container-network-op/commit/ff376b71393dc76134a834a4a9012ede58ed03c6

A k8s deployment will be deployed on a small scale setup on 2 cluster setups. The first cluster "baseline" will use the non-RDMA enabled NICs (H16). Then the same tests will be conducted on the RDMA instances (H16r) to determine any performance gains.

Baseline: 2x H16 instances. 32 vCPU, 224 GiB, 4TB,
RDMA Enabled: 2x H16r instances. 32 vCPU, 224 GiB, 4TB,

I will be using a baseline Energy Model run that has notorious been slow to run and even crashes the API due to flooding the network with too much data results being returned from the worker nodes. The first observation will be qualitative to determine if RDMA alleviates some of the pressures due to higher throughput. Then, I will go into more detail and attempt to capture the metrics in terms of compute resources, network throughput (time and total traffic) and disk I/O. Some of these metrics can captured using Kubernetes logging tools (e.g. Prometheus) but work will need to be done to track the network outside of the k8s cluster. FreeFlow is using a special container to flow the traffic to the RDMA NIC and this is where the perforce gain should be expected. I can use the OpenStudio-workflow logs to determine the processing time differences as an initial measurement, but I may need to use other networking tools installed directly on the VM hosts (e.g. iperf3) to fully understand the potential RDMA improvement.

The above steps can be repeated using the AWS SR-IOV instances as well as this setup should also come with a network performance gain.