TECHNOLOGY GUIDE

Intel Corporation

Multi Cloud Services – High Performance Computation Modernization in Financial Services Industries

Authors

Petar Torre

Yury Kylulin

Bruno Domingues

Intel Corporation

Shay Naeh

Cloudify

1 Introduction

Financial Services Industry and other vertical customers running High Performance Computing (HPC) workloads with strict compute requirements and adopting Cloud Native principles, require orchestration for managing Multi-Cloud and Edge services that may run from edge to data center. This guide describes how the Monte Carlo simulation application benefits from Intel® Advanced Vector Instructions 512 (Intel® AVX-512), and how such containerized application can be used on Kubernetes* cluster automated across multiple clouds with Cloudify* orchestration platform.

The solution was developed as a public showcase demonstrating scalability, with robust automation. It is loosely coupled and fully modular, respecting the boundaries of orchestration, applications, software platform, and hardware platform layers. These attributes ease the application on-boarding and lifecycle management efforts, while allowing performance-optimized deployments. Figure 1 shows a high-level view of the solution, which is described in detail in later sections of this document.

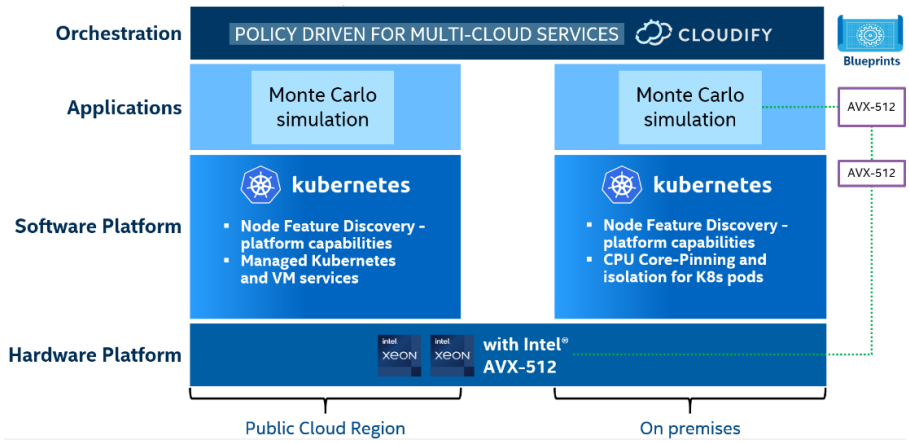

Figure 1. Multicloud Solution Layered Stack with Orchestration, Applications, Software Platform, and Hardware Platform

The presence of identical cloud infrastructure hardware instances across multiple locations minimizes the efforts of application migration, validation, and supporting of hybrid cloud environments, and makes operations easier. Intel offers an unmatched portfolio for the unique requirements of cloud and edge implementations and enables open source ecosystem like Kubernetes and developers to take advantage of such innovation.

This guide is intended for architects and engineers in Financial Services Industry and other verticals with strict compute requirements, interested in best practices for designing fully automated hybrid cloud environments based on Kubernetes-managed containers.

This document is part of the Network Transformation Experience Kit, which is available at https://networkbuilders.intel.com/network-technologies/network-transformation-exp-kits.

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	April 2021	Initial release.

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
AWS*	Amazon Web Services*
BMRA	Bare Metal Reference Architecture
EPA	Enhanced Platform Awareness
K8s*	Kubernetes*
NFD	Node Feature Discovery
NUMA	Non-Uniform Memory Access
TOSCA	Topology and Orchestration Specification for Cloud Applications
VPC	Virtual Private Cloud

1.2 Reference Documentation

Table 2. Reference Documents

REFERENCE	SOURCE
All sources for this white paper	https://github.com/intel/Financial-Services-Workload-Samples
Cloudify website	https://cloudify.co/
Intel® Network Builders website for Containers Experience Kits	https://networkbuilders.intel.com/network-technologies/container-experience-kits
Container Bare Metal for 2nd Generation Intel® Xeon® Scalable Processor Reference Architecture (installation guide)	https://builders.intel.com/docs/networkbuilders/container-bare-metal-for-2nd-generation-intel-xeon-scalable-processor.pdf
Node Feature Discovery Application Note	https://builders.intel.com/docs/networkbuilders/node-feature-discovery-application-note.pdf
Kubernetes CPU Manager – CPU Pinning and Isolation	https://kubernetes.io/blog/2018/07/24/feature-highlight-cpu-manager/, https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies/
Monte Carlo	https://en.wikipedia.org/wiki/Monte_carlo_simulation#Finance_and_business
Intel Advanced Vector Extensions 512	https://www.intel.com/content/www/us/en/architecture-and-technology/avx-512-overview.html

1.3 Motivation

Not all Kubernetes nodes are created equally — Workloads such as low-latency trading in finance or Network Functions Virtualization require fast processing of network traffic and special consideration for placement on physical servers where they can benefit from appropriate hardware acceleration on certain Kubernetes nodes.

Edge environments — Even the biggest cloud environments consist of smaller data centers, some of which can run on small edge or on-premises locations. The motivation for such design is usually a mix of bandwidth, latency, and privacy requirements. From a workload placement perspective, it is essential to orchestrate which workloads are placed on which edges.

This type of solution is matured over several years in Communications Service Provider vertical, where the Kubernetes enhancements (described in this guide) and many other features are being productized by major vendors like Red Hat as platform foundation for key carrier network workloads to build modern 5G Core and virtualized Radio Access Networks.

2 Monte Carlo Simulation and Intel AVX-512

Monte Carlo simulation is commonly used to evaluate the risk and uncertainty that would affect the outcome of different decision options. Monte Carlo methods are used in corporate finance and mathematical finance to value and analyze (complex) instruments, portfolios, and investments by simulating the various sources of uncertainty affecting their value, and then determining the distribution of their value over the range of resultant outcomes.

Monte Carlo algorithms are used to calculate the value of an option with multiple sources of uncertainties and random features, such as changing interest rates, stock prices or exchange rates, etc. to evaluate complex instruments, portfolios, and investments. Monte Carlo European options is a numerical method that uses statistical sampling techniques to approximate solutions to quantitative problems. This is a compute-bound, double precision workload and benefits from Intel® Turbo Boost Technology and Intel® Hyper-Threading Technology.

Such simulations are based on vector type random number generators. Intel Advanced Vector Extensions 512 (Intel AVX-512) is a set of instructions that can accelerate performance for workloads and usages such as scientific simulations, financial analytics, artificial intelligence (AI)/deep learning, 3D modeling and analysis, image and audio/video processing, cryptography and data compression.

Previous Intel® AVX2 contained most vector instructions of 256 bits. Intel AVX-512 doubles that width to 512 bits and can boost performance for such demanding workloads.

There can be many types of Monte Carlo simulations with different requirements such as desired time to complete, or sensitivity from other workloads etc. As a baseline for this guide, we assumed that simulation run needs to complete under certain time that can vary slightly. It is also possible to configure the systems for more sensitive workloads requiring more predictability or for more resilient (less sensitive) workloads that can run in shared cloud environments with impairments coming from the workload of other tenants.

The achieved results show that the elapsed time to complete simulation runs with Intel AVX-512 is about half of the time taken with Intel AVX2. This result correlates with doubling the width of vector instructions used.

3 How Cloudify Orchestration Works

Cloudify, as a global orchestrator, provisions workloads to run on distributed Kubernetes clusters based on a set of requirements and available resources that match those requirements. Figure 2 shows the Cloudify console view.

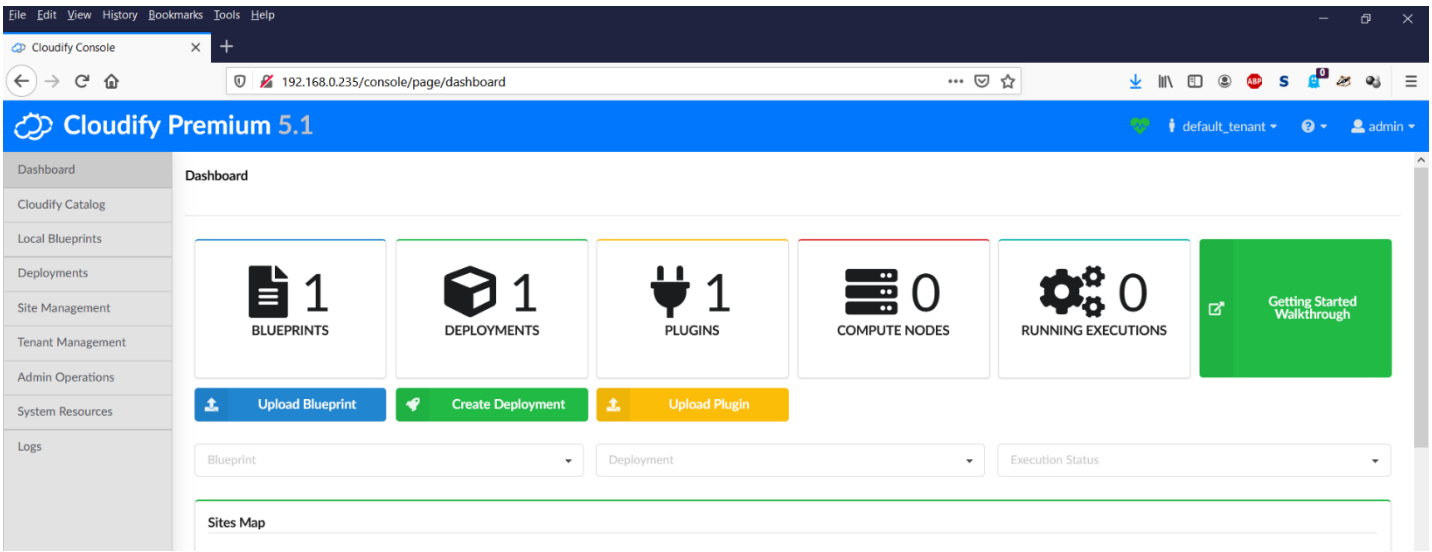

Figure 2. Cloudify Console View

Figure 3 describes a multi-cloud network setting, orchestrated by Cloudify. The diagram shows four Kubernetes-managed locations across multi-clouds. Each Kubernetes cluster supports a set of platform capabilities addressing different performance and operation needs. Based on criteria such as location, resource availability, and special resource requirements, Cloudify provisions a workload to the correct Kubernetes cluster. Yet this is only part of the work - each Kubernetes cluster is composed from multiple nodes, each having different hardware capabilities. Cloudify works with Intel-led Kubernetes enhancements required for optimized placement and performance of compute- or network-intensive workloads. This supports multiple capabilities like Node Selector for CPU finding instructions, CPU Manager for pinning and isolation, Device Plugins for mapping acceleration devices to pods, and Topology Manager for transparent assignment across NUMA nodes. In this demo of Monte Carlo compute-intensive application, Cloudify maps the workload to the right Kubernetes nodes by using *node labels* and *node selectors*, while all intelligence about the NUMA topology, CPU pinning, or assignment of specific hardware devices is performed within the software platform of Kubernetes, Kubelet, and Linux*.

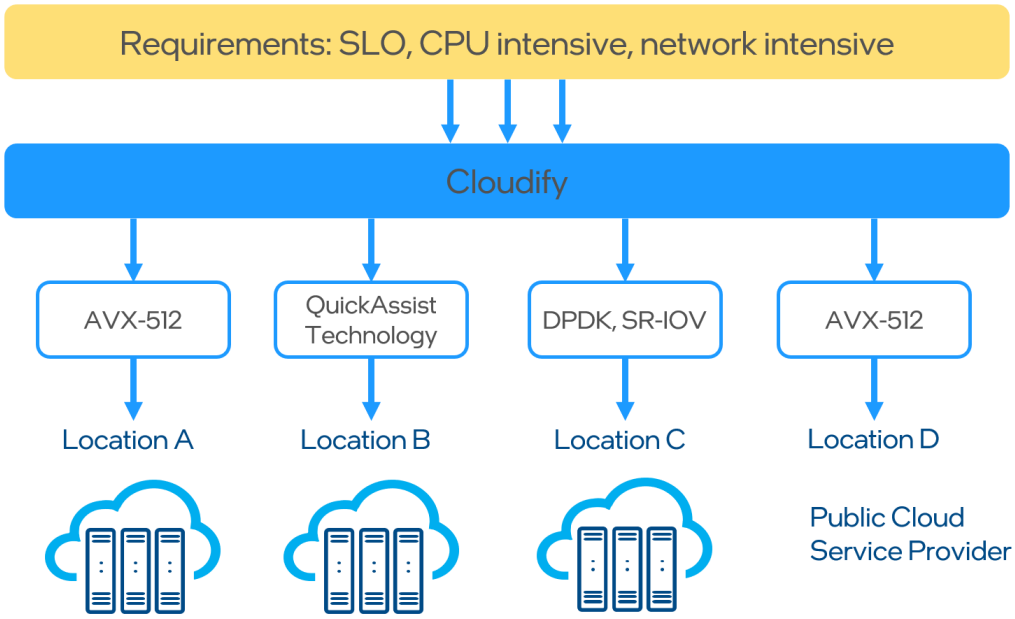

Figure 3. Requirements for Central Cloudify Orchestrated to Distributed Sites

3.1 How to Select Node in Kubernetes Cluster

Node Feature Discovery (NFD) is a Kubernetes add-on that detects and advertises hardware and software capabilities of a platform, which in turn can be used to facilitate intelligent scheduling of a workload. By utilizing NFD, each Kubernetes node is labeled with a list of the hardware capabilities it supports. Figure 4 shows an example that Single-Root Input/Output Virtualization (SR-IOV) and Non-Uniform Memory Access (NUMA) are supported only by node 1, giving node 1 an advantage for supporting performance sensitive workloads.

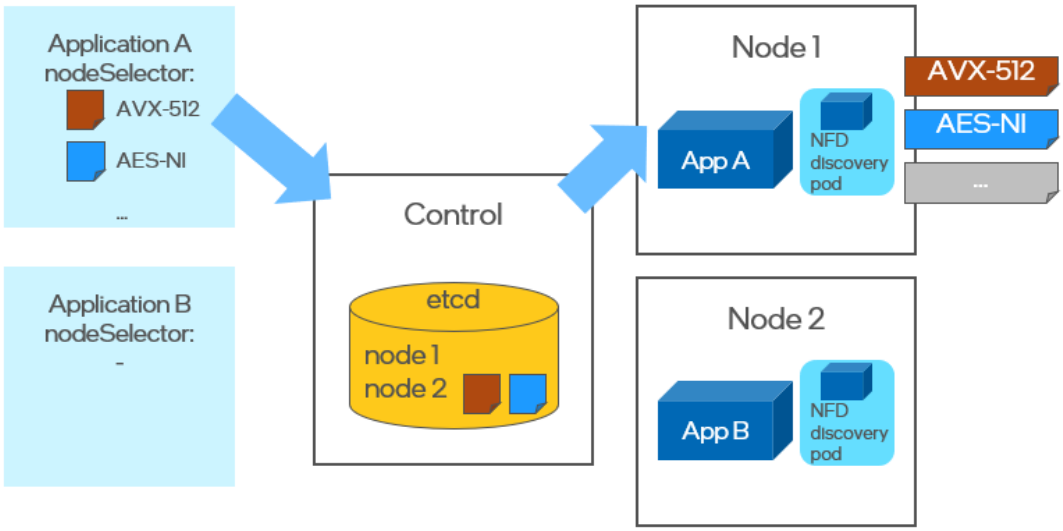

Figure 4. NFD in Kubernetes

Application node-selectors are defined in the YAML file as part of the pod deployment definition. These selectors are used to map applications (pods) to nodes that support the required capabilities. See Figure 5 as an example of a list of possible node-selector labels and Figure 6 as an example of a YAML file.

```
    node.alpha.kubernetes-incubator.io/nfd-cpuid-ADX: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AESNI: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX2: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512BW: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512CD: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512DQ: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512F: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512VL: "true"
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX: "true"
...
```

Figure 5. List of NFD Labels

```
apiVersion: v1
kind: Pod
metadata:
  name: montecarloavx512
  labels:
    name: montecarloavx512
spec:
  restartPolicy: Never
  nodeSelector:
    node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512BW: "true"
...
```

Figure 6. Monte Carlo Pod Placement with "nodeSelector"

In the case of the demonstration discussed in this guide we provisioned[1] (1) NodeJS pod on a generic Kubernetes node and (2) Monte Carlo pod on a Kubernetes node identified per NFD with Intel AVX-512 capability that supports CPU instructions for vector extensions. All the Kubernetes nodes supporting the Intel AVX-512 capability are grouped under a special group named AVX512. In Figure 7, the QAT group is marked with a light blue background. This allocation is done on an on-premises Kubernetes cluster.

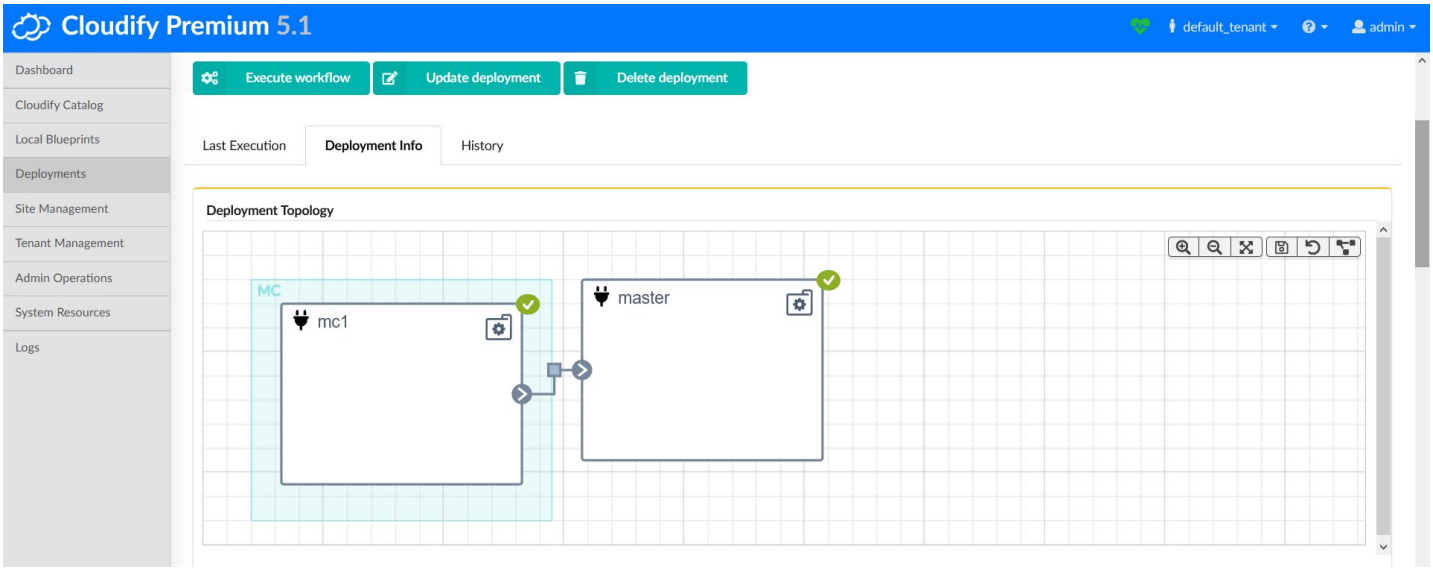

Figure 7. Cloudify Console with Composer View

3.2 Non-Kubernetes and Hybrid Environments

Cloudify can provision workloads on both Kubernetes and non-Kubernetes hybrid environments. As shown in Figure 8, workloads can be provisioned to Amazon Web Services* (AWS*). A Virtual Private Cloud (VPC) environment is instantiated on AWS and a VM is created in that VPC. By matching the workload requirements, Cloudify places the workload on the right node in AWS.

[1] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

A mixture of Kubernetes and non-Kubernetes environments can be maintained by the orchestrator. Moreover, these environments can be located on-premises or on any public cloud.

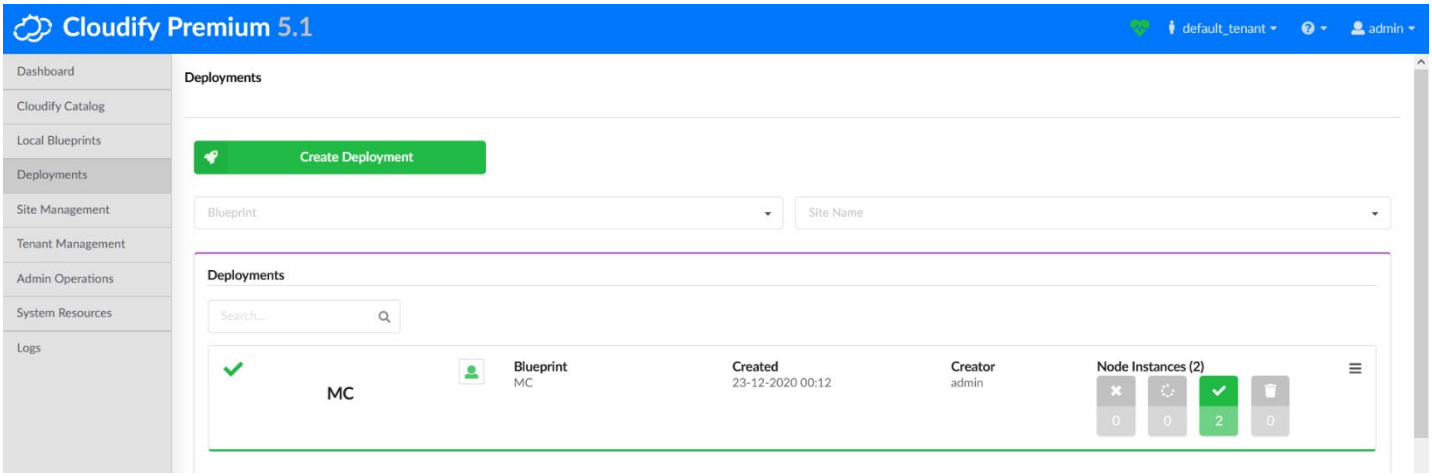

Figure 8. Cloudify Console with Deployments View

3.3 Intent-Based Decoupling of Layers

With the term "intent", we mean that we specify the 'what' and not the 'how'. The need for 'CPU intensive' hardware may differ based on the environment because each environment may hold different definitions and parameters. If we decouple tenant user from the environment, it makes the process of application placement to the platform simple and transparent. The user specifies the requirements they need, and Cloudify will match those requirements with the right compute nodes per network definitions.

By utilizing Topology and Orchestration Specification for Cloud Applications (TOSCA), we can write an intent-based blueprint that decouples application need from a Kubernetes cluster implementation. In this scenario, the tenant only needs to specify the requirement for nodes with certain capabilities and Cloudify will match the right resources and provision the workloads correctly.

Intent-based definitions decouple the workload requirements from the underlying environment without changing anything at the higher level of the workload definition. Even when changing the environment where the workload runs and moving the workload to a new environment, Cloudify will look for the right resources and definitions on the new environment and will select them based on the workload requirements.

TOSCA also helps in the 'matching' process. TOSCA defines *Requirements* and *Capabilities* primitives, where a user specifies in the "Requirements" primitive what it needs, for example, CPU intensive or Network intensive and "Capabilities". TOSCA also holds a list of supported capabilities by a compute node. In Kubernetes, *Requirements* are defined by node selectors and *Capabilities* by node labels. Hence, TOSCA definitions cover the generic use cases and are not restricted to Kubernetes environments, pods and nodes.

To summarize, TOSCA requirements and capabilities provide the mechanism to define a generic case for workload requirements and map them to nodes that supports the capabilities to fulfill those requirements.

In Kubernetes specifically:

- TOSCA 'Capabilities' → Kubernetes node Label
- TOSCA 'Requirements' → Kubernetes node Selector

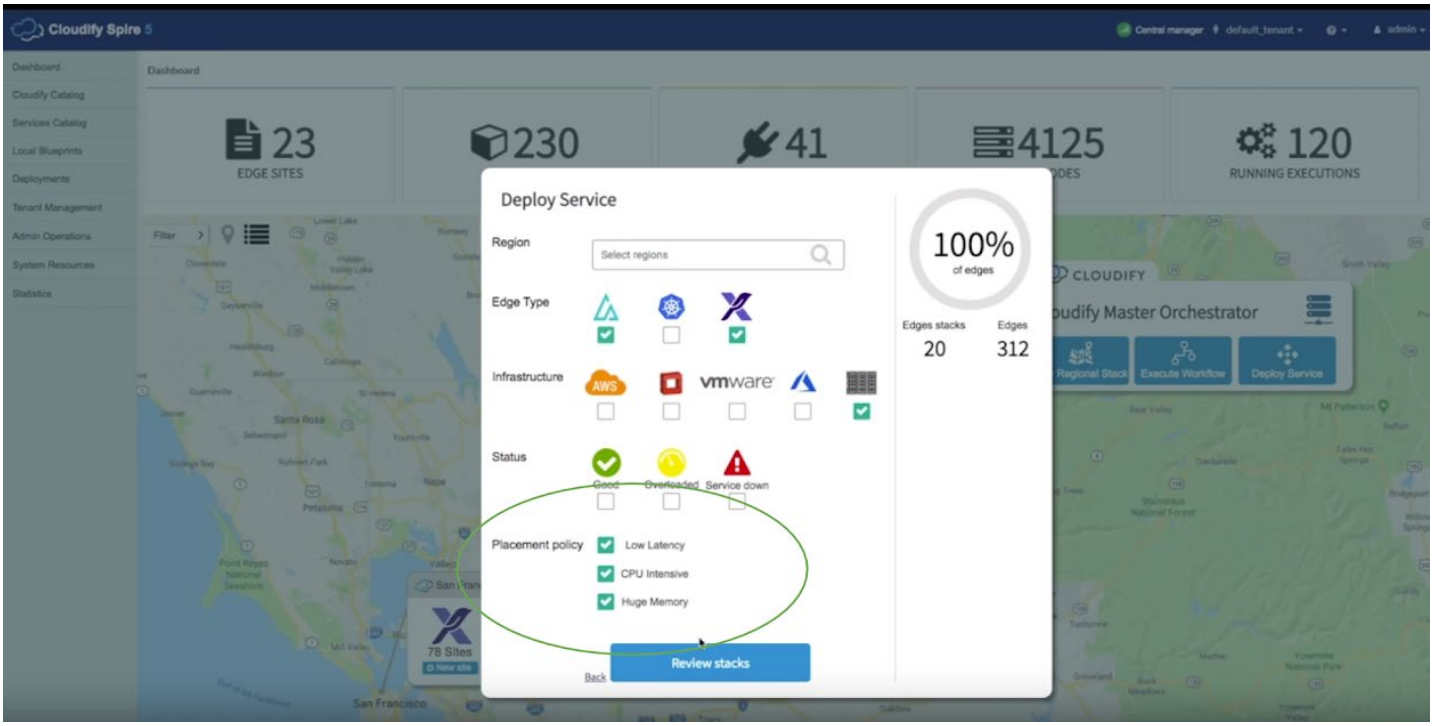

Figure 9. Intent-Based Placement

4 Software and Hardware Platform

4.1 Physical Topology

The physical topology[2] for the testing uses a Kubernetes cluster based on one control node and two worker nodes. Both server nodes are with Intel® Xeon® processor including Intel AVX-512. On a separate host, Ansible* runs in the 'Ansible VM' enabling Kubernetes cluster installation using Bare Metal Reference Architecture (BMRA) v2.0.

[2] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

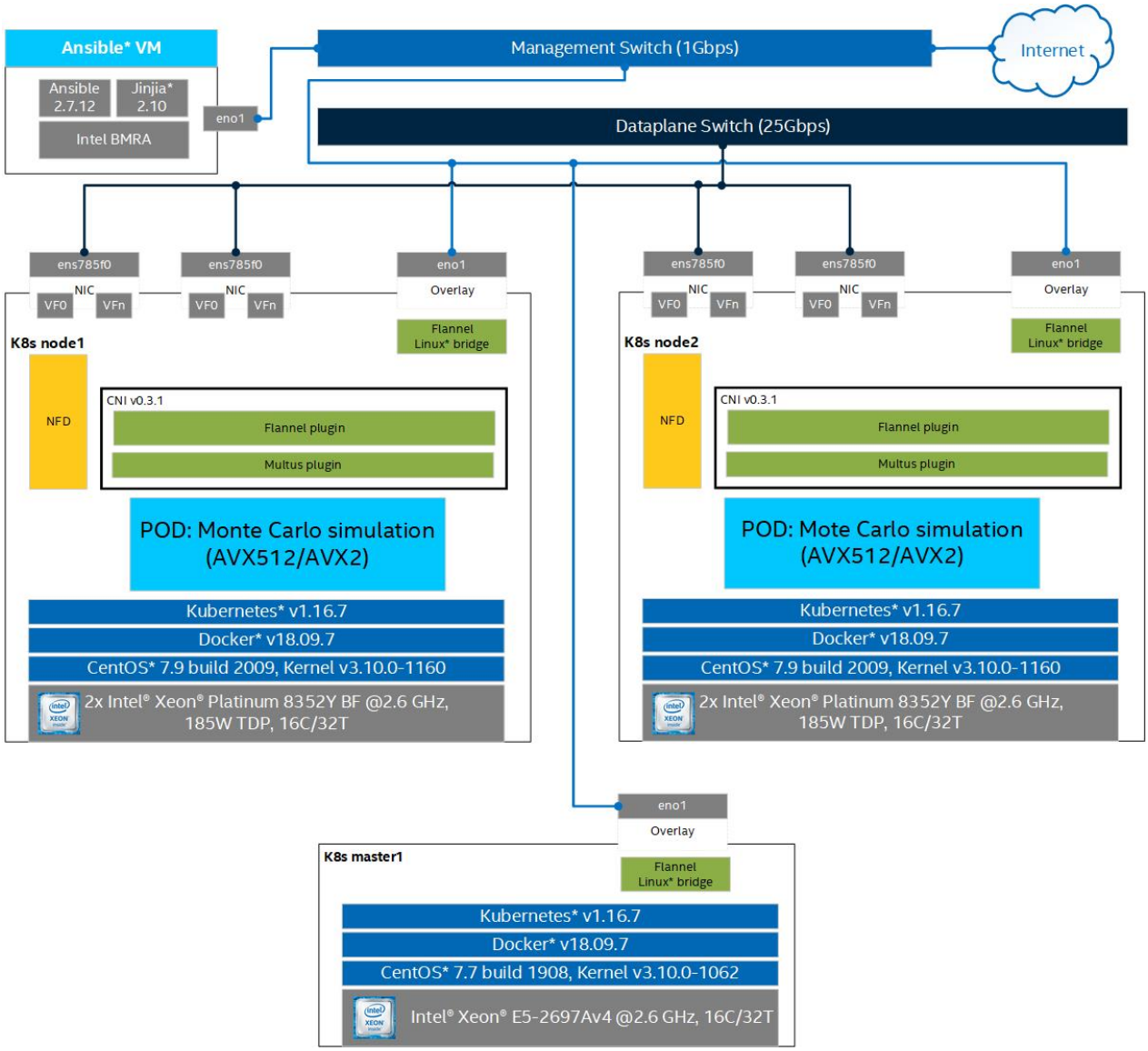

Figure 10. Physical Topology

4.2 Software Topology

For the Kubernetes cluster and plugins setup, we used the Container Bare Metal Reference Architecture Ansible Playbook (available as part of Container Bare Metal for 2nd and 3rd Generation Intel® Xeon® Scalable Processor). In addition, the Cloudify Manager was installed to work with the cluster through RESTful APIs.

In this setup, we used the Node Feature Discovery capability of Kubernetes to demonstrate the role of intelligent workload placement.

4.3 Hardware Specifications

This section lists the hardware components and systems that were utilized in this test setup[3]. The 3rd Generation Intel® Xeon® Scalable processors feature a scalable, open architecture designed for the convergence of key workloads such as applications and services, control plane processing, high-performance packet processing, and signal processing.

[3] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

Table 3. Hardware Specifications

ITEM	DESCRIPTION	NOTES
Platform	Intel® Xeon® Processor Scalable Family	Intel® Xeon® processor-based dual-processor server board 2 x 25 GbE LAN ports
Processors	2x Intel® Xeon® Platinum 8352Y Processor	Base configuration: 32 cores, 64 threads, 2.2 GHz, 205 W, 48 MB L3 total cache per processor, 3 UPI Links, DDR4-3200, 8 memory channels Intel SpeedStep Performance Profile: 16 cores, 32 threads, 2.6 GHz, 185 W, 48 MB L3 total cache per processor, 3 UPI Links, DDR4-3200, 8 memory channels
	2x Intel® Xeon® E5-2697v4 Processor	16 cores, 32 threads, 2.6 GHz, 145 W, 40 MB L3 total cache per processor, 2 QPI Links, DDR4-2400, 4 memory channels
Memory	1TB (16 x 64GB 3200MHz DDR RDIMM) or minimum all 8 memory channels populated (1 DPC)	256 GB to 1 TB
Networking	2 x NICs - Required Each NIC NUMA aligned	2 x Dual Port 25 GbE Intel® Ethernet Network Adapter XXV710 SFP28+
		2 x Dual Port 10 GbE Intel® Ethernet Converged Network Adapter X722
Local Storage	Intel SSD DC S3500	
BIOS	Intel Corporation SE5C6200.86B.0020.P18.2102081146	Intel® Hyper-Threading Technology (Intel® HT Technology) enabled Intel® Virtualization Technology (Intel® VT-x) enabled Intel® Virtualization Technology for Directed I/O (Intel® VT-d) enabled Intel® SpeedStep Technology (Intel® SST) enabled Intel® SpeedStep Performance Profile (Intel® SST-PP) enabled
Switches	Huawei* S5700-52X-LI-AC Huawei* CE8860-4C-EI with CE88-D24S2CQ module	Management 1 GbE Switch Dataplane 25 GbE Switch

4.4 Software Specifications

Table 4. Software Versions

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Cloudify	Cloudify Premium 5.1	https://cloudify.co/
Monte Carlo Simulation		https://github.com/intel/Financial-Services-Workload-Samples/tree/main/MonteCarloEuropeanOptions
Host OS	CentOS* 7.9 build 2009 Kernel version: 3.10.0-1160.15.2.el7.x86_64	https://www.centos.org/
Ansible	Ansible v2.7.1	https://www.ansible.com/
BMRA 2.0 Ansible Playbook	Master Playbook v1.0	https://github.com/intel/container-experience-kits

SOFTWARE FUNCTION	SOFTWARE COMPONENT	LOCATION
Python*	Python 2.7	https://www.python.org/
Kubespray*	Kubespray v2.8.0-31-g3c44ffc	https://github.com/kubernetes-sigs/kubespray
Docker*	Docker* 18.09.7-ce, build 2d0083d	https://www.docker.com/
Container Orchestration Engine	Kubernetes v1.16.7	https://github.com/kubernetes/kubernetes
Kubernetes CPU Manager	Included in Kubernetes	https://github.com/kubernetes/examples/blob/master/staging/cpu-manager/README.md
Node Feature Discovery	NFD v0.3.0	https://github.com/kubernetes-sigs/node-feature-discovery
Intel Ethernet Drivers		https://sourceforge.net/projects/e1000/files/ixgbe%20stable/5.2.1 https://sourceforge.net/projects/e1000/files/ixgbevf%20stable/4.2.1 https://sourceforge.net/projects/e1000/files/i40e%20stable/2.0.30 https://sourceforge.net/projects/e1000/files/i40evf%20stable/2.0.30

4.5 Platform BIOS settings[4]

Table 5. Platform BIOS Setting

MENU (ADVANCED)	PATH TO BIOS SETTING	BIOS SETTING	SETTINGS FOR DETERMINISTIC PERFORMANCE	SETTINGS FOR MAX PERFORMANCE WITH TURBO MODE ENABLED	REQUIRED OR RECOMMENDED
Power Configuration	CPU P State Control	EIST PSD Function	HW_ALL	SW_ALL	*Recommended*
Boot Performance Mode	Max. Performance	Max. Performance	*Required*		
Energy Efficient Turbo	Disable	Disable	*Recommended*		
Turbo Mode	Disable	Enable	*Recommended*		
Intel® SpeedStep® (Pstates) Technology	Disable	Enable	*Recommended*		
Hardware PM State Control	Hardware P-States	Disable	Disable	*Recommended*	
CPU C State Control	Autonomous Core C-State	Disable	Enable	*Recommended*	
CPU C6 Report	Disable	Disable	*Recommended*		
Enhanced Halt State (C1E)	Disable	Enable	*Recommended*		

[4] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

MENU (ADVANCED)	PATH TO BIOS SETTING	BIOS SETTING	SETTINGS FOR DETERMINISTIC PERFORMANCE	SETTINGS FOR MAX PERFORMANCE WITH TURBO MODE ENABLED	REQUIRED OR RECOMMENDED
Energy Perf Bias	Power Performance Tuning	BIOS Controls EPB	BIOS Controls EPB	*Recommended*	
ENERGY_PERF_BIAS_CFG Mode	Perf	Perf	*Recommended*		
Package C State Control	Package C State	C0/C1 State	C6	*Recommended*	
Intel® Ultra Path Interconnect (Intel® UPI) Configuration	Intel® UPI General Configuration	LINK L0P ENABLE	Disable	Disable	*Recommended*
LINK L1 ENABLE	Disable	Disable	*Recommended*		
SNC	Disable	Disable	*Recommended*		
Memory Configuration	Enforce POR	Disable	Disable	*Recommended*	
IMC Interleaving	2-Way Interleave	2-Way Interleave	*Recommended*		
Volatile Memory Mode	2 LM mode	2 LM mode	*Required*		
Force 1-Ch Way in FM	Disabled	Disabled	*Required*		
Platform Configuration	Miscellaneous Configuration	Serial Debug Message Level	Minimum	Minimum	*Recommended*
PCI Express* Configuration	PCIe* ASPM Support	Per Port	Per Port	*Recommended*	
Uncore	Uncore Frequency Scaling	Disable	Disable	*Required*	

Note: To gather performance data[5] required for conformance, use either column with deterministic performance or turbo mode enabled in this table. Some solutions may not provide the BIOS options that are documented in this table. For Intel® Select Solution, the BIOS should be set to the "Max Performance" profile with virtualization.

4.6 Prepare Software Platform

Kubernetes cluster must be ready before Cloudify can use Kubernetes plugin to orchestrate applications on it. Here we show how this can be performed with Intel Bare Metal Reference Architecture on premises and describe the same in public cloud instances.

4.6.1 Ansible Host, Control and Worker Node Software Prerequisites

1. As root enter the following commands in Ansible Host:

```
# yum install -y epel-release
# wget https://releases.ansible.com/ansible/rpm/release/epel-7-x86_64/ansible-2.7.12-
1.el7.ans.noarch.rpm
# yum install -y ./ansible-2.7.12-1.el7.ans.noarch.rpm
# easy_install pip
# pip2 install jinja2 --upgrade
# yum install -y python36 python2-jmespath
```

[5] See backup for workloads and configurations or visit www.Intel.com/PerformanceIndex. Results may vary.

2. Enable password-less login between all nodes in the cluster.
   Step 1: Create authentication SSH-keygen keys on Ansible Host:
   ```
   # ssh-keygen
   ```
   Step 2: Upload generated public keys to all the nodes from Ansible Host:
   ```
   # ssh-copy-id root@node-ip-address
   ```

4.6.2 Deploy Intel Bare Metal Reference Architecture Using Ansible Playbook

1. Get Ansible playbook:
   ```
   # git clone https://github.com/intel/container-experience-kits.git
   # cd container-experience-kits/playbooks
   ```
2. Copy example inventory file to the playbook home location:
   ```
   # cp examples/inventory.ini .
   ```
3. Edit the inventory.ini to reflect the requirement. Here is the sample file.
   ```
   [all]
   k8s-control1 ansible_host=192.168.0.235 ip=192.168.0.235 ansible_user=root
   k8s-node1 ansible_host=192.168.0.236 ip=192.168.0.236 ansible_user=root
   k8s-node2 ansible_host=192.168.0.237 ip=192.168.0.237 ansible_user=root

   [kube-master]
   k8s-control1

   [etcd]
   k8s-control1

   [kube-node]
   k8s-node1
   k8s-node2

   [k8s-cluster:children]
   kube-master
   kube-node

   [calico-rr]
   ```
4. Copy group_vars and host_vars directories to the playbook home location:
   ```
   # cp -r examples/group_vars examples/host_vars .
   ```
5. Update group_vars to match the desired configuration.
   ```
   # vim group_vars/all.yml

   ---
   ## BMRA master playbook variables ##

   # Node Feature Discovery
   nfd_enabled: true
   nfd_build_image_locally: true
   nfd_namespace: kube-system
   nfd_sleep_interval: 30s

   # Intel CPU Manager for Kubernetes
   cmk_enabled: false
   cmk_namespace: kube-system
   cmk_use_all_hosts: false # 'true' will deploy CMK on the master nodes too
   #cmk_hosts_list: node1,node2 # allows to control where CMK nodes will run, leave this option
   commented out to deploy on all K8s nodes
   cmk_shared_num_cores: 12 # number of CPU cores to be assigned to the "shared" pool on each of
   the nodes
   cmk_exclusive_num_cores: 20 # number of CPU cores to be assigned to the "exclusive" pool on
   each of the nodes
   cmk_shared_mode: spread # choose between: packed, spread, default: packed
   cmk_exclusive_mode: spread # choose between: packed, spread, default: packed

   ## Proxy configuration ##
   proxy_env:
     http_proxy: ""
     https_proxy: ""
     no_proxy: ""
   ```

```
## Kubespray variables ##
```

If you use older Linux kernel versions, interaction between kubelet and Completely Fair Scheduler (CFS) can lead to unnecessary throttling for the pods fully consuming Hyper-Threaded cores. To avoid that in `/etc/kubernetes/kubelet.env` configure `--cpu-cfs-quota=false` and restart kubelet service with the following command:

```
systemctl restart kubelet
```

More on that on https://github.com/kubernetes/kubernetes/issues/67577 .

4.7 Intel Bare Metal Reference Architecture

The Container Bare Metal for 2nd and 3rd Generation Intel® Xeon® Scalable Processor Reference Architecture document provides guidelines for setting a Kubernetes performant platform for data plane and other performance-sensitive workloads, independent of vendor implementations. Based on the platform set, various tests can be performed even before such platforms are productized. Companies that plan to develop their own Kubernetes-based platform can refer to this document for additional details.

4.8 Similar Setup in Public Cloud

Cloud Service Providers (CloudSP) offer managed Kubernetes services with worker nodes based on their instances/VM services. Such clusters can typically be setup by implementing Infrastructure as a Code concept like Terraform configurations, with proprietary tools from CloudSP, or manually over CloudSP GUI consoles.

For sensitive compute-intensive workloads, process isolation can be achieved by running pods on nodes of dedicated instances, and within those pods assigning available cores. This method is suggested because such managed Kubernetes offerings do not offer support for Kubernetes CPU Manager.

Figure 11 shows an example of simple Terraform configuration that can be applied to create such cluster with AWS.

```
...
module "eks" {
  source       = "git::https://github.com/terraform-aws-modules/terraform-
aws-eks.git?ref=v12.1.0"
  cluster_name = local.cluster_name
  vpc_id       = module.vpc.aws_vpc_id
  subnets      = module.vpc.aws_subnet_private_prod_ids
  node_groups = {
    eks_nodes = {
      desired_capacity = 1
      min_capacity     = 2
      max_capacity     = 2
      instance_type = "c5.2xlarge"
    }
  }
  manage_aws_auth = false
}
...
```

Figure 11. Terraform Configuration

To setup such cluster do:

```
cd $WORK_DIR/Financial-Services-Workload-Samples/MonteCarloEuropeanOptions/cloud/terraform-eks
terraform init
terraform plan
terraform apply
aws eks update-kubeconfig --region us-east-2 --name my-eks-cluster  # or other region
cd ../docker_packaging
# before running next commands - check for newer versions of NDF
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/node-feature-
discovery/v0.7.0/nfd-master.yaml.template
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/node-feature-
discovery/v0.7.0/nfd-worker-daemonset.yaml.template
```

Check it with:

```
kubectl get pods -A -o wide | grep node-feature-discovery
```

which should list NFD running on all nodes.

5 Prepare Tenant Software

For cloud tenant we prepare and package Monte Carlo simulation application. We also configure reporting system and orchestration.

This section shows how to prepare and use containerized docker images in the following steps:

- Prepare Linux* environment.
- Build containerized image for Monte Carlo simulation from custom Dockerfile.
- Start procedure with Pushgateway, Prometheus*, and Grafana* using prebuilt images.
- Create Kubernetes pod yaml files.
- Create Cloudify deployments.

5.1 Linux Environment

This guide assumes that you are running CentOS* 7. If you run Ubuntu* or another distribution, use equivalent commands like with `apt-get` or other package manager.

By default, Kubernetes CPU Manager policy is set to "none" which means that there will be no affinity beyond what the OS scheduler does automatically. To exclusively use cores on the node for the running pods, change the policy to "static", and if needed use `taskset` and accordingly modify `TASKSET` variable in .yaml deployment descriptors. This allows containers in the pods running in the Guaranteed QoS class use some sort of cpu pinning and isolation. In `/etc/kubernetes/kubelet.env` configure `--cpu-manager-policy=static` and restart kubelet service with the following command:

```
systemctl restart kubelet
```

For more information, refer to the article https://kubernetes.io/docs/tasks/administer-cluster/cpu-management-policies.

Do the compilations and integration on one of the nodes, as root, and have working directory here described as WORK_DIR, like /root/w.

In `bash` use the following environment variables, or at the end of `/etc/bashrc` or `~/.bashrc` add:

```
export WORK_DIR=/root/work  # or where you like to have it
export PUSHGWIP=<YOUR_IPV4_ADDR_WHERE_RUN_PUSHGATEWAY_PROMETEUS_AND_GRAFANA>
```

Restart bash if using `.bashrc` or similar files.

For start do:

```
mkdir $WORK_DIR
yum update
```

5.2 Monte Carlo Simulation Binaries with Intel AVX-512

Intel® System Studio 2020 among many other tools includes Intel® C++ Compiler, Intel® Threading Building Blocks (TBB) and Intel® Math Kernel Library (MKL). On system where recent versions of compiler, TBB and MKL are installed and configured (environment variables PATH, TBBROOT and MKLROOT pointing to right folders like those in /opt/intel/system_studio_2020), compile Monte Carlo simulation binaries with Intel AVX-512 and also previous Intel AVX2 to have it for comparison:

```
cd $WORK_DIR
git clone https://github.com/intel/Financial-Services-Workload-Samples.git
git checkout
cd Financial-Services-Workload-Samples/MonteCarloEuropeanOptions
export PATH=$PATH:/opt/intel/system_studio_2020/bin  # or where is icpc
export TBBROOT=/opt/intel/system_studio_2020/compilers_and_libraries_2020.4.304/linux/tbb  # or
where to find TBB
export MKLROOT=/opt/intel/system_studio_2020/compilers_and_libraries_2020.4.304/linux/mkl  # or
where to find MKL
make
cp
/opt/intel/system_studio_2020/compilers_and_libraries_2020.4.304/linux/compiler/lib/intel64_lin
/libiomp5.so .  # or where to find libiomp5.so
cp
/opt/intel/system_studio_2020/compilers_and_libraries_2020.4.304/linux/tbb/lib/intel64_lin/gcc4
.8/libtbbmalloc.so.2 .  # or where to find libtbbmalloc.so.2
```

Now in the folder you should have binaries MonteCarloInsideBlockingDP.avx2 and MonteCarloInsideBlockingDP.avx512. To test the binary, try it with:

```
export LD_LIBRARY_PATH=.
```

```
./MonteCarloInsideBlockingDP.avx2 1 8192 262144 8k | grep Elapsed
```

which should give results like:

```
Time Elapsed = 7.222699
```

and:

```
./MonteCarloInsideBlockingDP.avx512 1 8192 262144 8k | grep Elapsed
```

which will on systems with AVX-512 give shorter elapsed time or error if the system doesn't have AVX-512:

```
Please verify that both the operating system and the processor support Intel(R) AVX512F,
AVX512DQ, AVX512CD, AVX512BW and AVX512VL instructions.
```

5.3 Build Monte Carlo Container Image

CPU pinning and isolation is desired for many cases and workload types including but not limited to latency sensitive workloads. To solve this problem, a CPU manager can be used like the native Kubernetes CPU Manager. CPU Manager for Kubernetes allocates CPU resources as fully "isolated" cores by isolating all hyper-thread siblings. It works with isolcpus parameters for the best isolation from the system processes but can also work without them. Scheduled pod while starting needs to find assigned vCPUs and use them.

Enter the following commands on the node that can access public Docker Hub with username $DOCKERHUBUSER (or modify accordingly for local image repository over usual `localhost:5000/imagename`):

```
cd $WORK_DIR/Financial-Services-Workload-
Samples/MonteCarloEuropeanOptions/cloud/docker_packaging
export DOCKERHUBUSER=your_docker_hub_username
```

Build the docker image, enter docker hub credentials when asked, and load it to docker hub with:

```
./build
```

Configure pod description YAML files with

```
./configure
```

5.4 Configure Pushgateway, Prometheus, and Grafana Under Docker

On node used for reporting, where permanent IP address is `PUSHGWIP`, perform the following one-time setup procedure:

```
cd $WORK_DIR/Financial-Services-Workload-Samples/MonteCarloEuropeanOptions/reporting
./run_once
```

After the first time setup is completed, you only need to use the command:

```
./start_all
```

Verify if Grafana, Prometheus and Pushgateway are running with the command:

```
docker ps | grep -e grafana -e prometheus -e pushgateway
```

Later when restarting pods with Monte Carlo containers, old metrics can be cleaned with:

```
./stop_all && ./start_all
```

Using some modern browser, go to the Grafana page `http://$PUSHGWIP:3000`, login with admin/password, change password.

Add data source Prometheus with URL `http://$PUSHGWIP:9090` at desired Scrape interval, Save & Test.

Create Dashboard using Add Query Prometheus with Metric time_elapsed and desired refresh rate.

5.5 Cloudify Setup, Configuration and Importing Orchestration Blueprint

Install and configure Cloudify Manager image as per Cloudify web documentation, use plug-in for Kubernetes.

Go to directory `$WORK_DIR/Financial-Services-Workload-Samples/MonteCarloEuropeanOptions/cloud/cloudify` and there in `mc2.yaml` and `inputs.yaml` modify Kubernetes master endpoint and security credentials (`api_key: { get_secret: kubernetes_token }`). Into Cloudify Manager as deployment "MC" import `mc2.yaml`, `inputs.yaml` and two existing YAML files describing pods.

6 Orchestrating and Result

6.1 Orchestrating with Cloudify

Now in Cloudify Manager you can install the deployment "MC" by executing workflow Install. This will run with preference to nodes with Intel AVX-512 (Figure 4) and return result as shown in Figure 12.

```
> kubectl describe pod montecarloavx512
Name:         montecarloavx512
Namespace:    default
Priority:     0
Node:         k8s-node1/10.10.1.21
Start Time:   Wed, 23 Dec 2020 02:12:51 +0300
Labels:       name=montecarloavx512
Annotations:  k8s.v1.cni.cncf.io/networks-status:
                [{
                    "name": "default-cni-network",
                    "interface": "eth0",
                    "ips": [
                        "10.244.1.36"
                    ],
                    "mac": "0a:58:0a:f4:01:24",
                    "default": true,
                    "dns": {}
                }]
Status:       Running
IP:           10.244.1.36
IPs:
  IP:  10.244.1.36
Containers:
  montecarlo:
    Container ID:
docker://4561476b40763fbec6897331e47386de6637e65aa4c710025bfb72284d1a19ed
    Image:          >> your_docker_hub_username << /montecarlo
    Image ID:       docker-pullable:// >> your_docker_hub_username <<
/montecarlo@sha256:73c86af1c8c511d48ad777d52a26b5e69c3cc38239a625205bf767f212ab58c4
    Port:           <none>
    Host Port:      <none>
    Command:
      /app/start
    State:          Running
      Started:      >> timestamp <<
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     8
      memory:  500Mi
    Requests:
      cpu:     8
      memory:  500Mi
    Environment:
      PR:          8
      USE_AVX512:  1
      PUSHGWIP:    >> $PUSHGWIP <<
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-d9dtl (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-d9dtl:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-d9dtl
    Optional:    false
QoS Class:       Guaranteed
Node-Selectors:  node.alpha.kubernetes-incubator.io/nfd-cpuid-AVX512BW=true
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:          <none>
```

Figure 12. Describe Pod Correctly Assigned

6.2 Result in Grafana

Now running pods will send results into PushGateway, Prometheus will scrape it from there, and Grafana will show it as graphs similar to Figure 13.

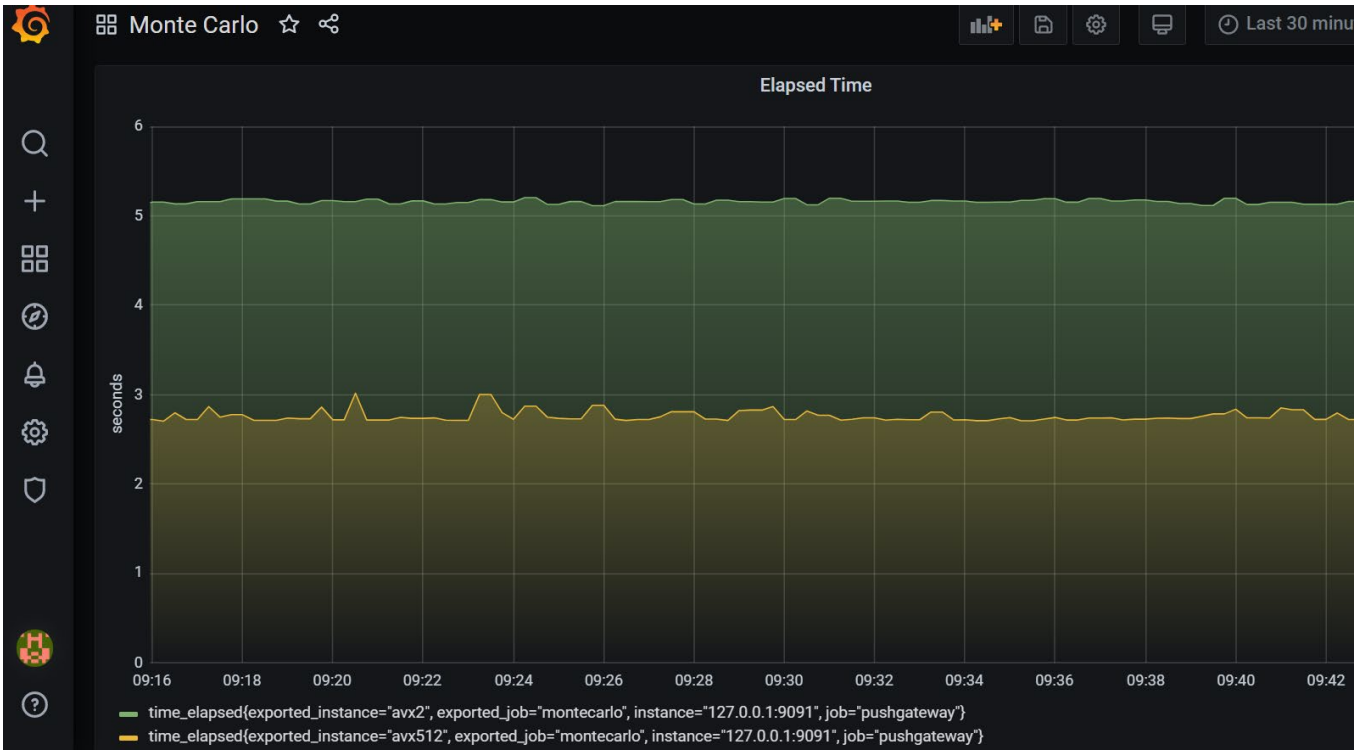

Figure 13. Grafana Dashboard with Metrics

Kubernetes CPU Manager is isolating pods on assigned vCPUs from other running workloads competing for those CPU cycles. The elapsed time can still slightly fluctuate if both (in this case, or more of them in generic case) Monte Carlo pods are scheduled at the same time on the same physical CPU socket when they still share resources like L3 cache, or in public cloud depending on the selected instance type.

7 Summary

This technology guide describes how to configure optimized Kubernetes in hybrid cloud, and shows how to build, package and orchestrate applications that will benefit from hardware acceleration and be properly decoupled from infrastructure. Same methodology can be applied to orchestrate other workloads to benefit from the same or other CPU instructions.

The Multi-Cloud Services on Kubernetes with Cloudify Orchestration and F5 Networks Functions technology guide describes the benefits of fixed function acceleration devices with Intel® QuickAssist Technology used to accelerate NGINX HTTPS.

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Your costs and results may vary.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

0421/DN/WIPRO/PDF 637679-001US