Chapter 1 - Overview

This document provides a complete reference architecture guide for infrastructure required for machine learning use cases. This document specifically covers open source machine learning software that includes Kubeflow using Charmed Kubernetes solution on Dell EMC hardware delivered by Canonical. This includes Dell EMC PowerEdge servers for workloads and storage based on Intel® Xeon® Scalable Processors and Dell EMC Networking.

This guide discusses the Dell EMC hardware specifications and the tools and services to set up both the hardware and software, including the foundation cluster and the Kubernetes cluster to run machine learning workloads. It also covers other tools used for the monitoring and management of the cluster in detail and how all these components work together in the system.

The guide also provides the deployment steps and references to configuration and automation scripts developed by Dell EMC and Canonical for the deployment process.

Revisions

Date	Description
March 2020	Initial release

Acknowledgements

This paper was produced by the following:

Author: Andrey Grebennikov

Support:

Other:

Table of contents

Executive summary

A Kubernetes cluster is now a common need by many organisations to execute cloud native workloads. Kubeflow is an open, community driven project to make it easy to deploy and manage a machine learning (ML) stack on Kubernetes. Dell EMC, Intel and Canonical have worked together to test and build a reference architecture that details software, hardware, and integration points of all solution components for deploying and operating Kubeflow. The architecture provides prescriptive guidance and recommendations for:

- Hardware design
- Infrastructure nodes
- Cloud nodes
- Storage nodes
- Network hardware and design
- Software layout
- System configurations

This architecture can be used as a starting point for infrastructure teams to setup and deploy Kubeflow for on-premises infrastructure. Machine learning is a powerful tool that is gaining wide acceptance across all industry segments to solve various problems and is helping achieve state of the art results thanks to the plethora of data available and access to high-performance compute infrastructure. Kubeflow is a good candidate for deploying and working with ML components as it enables a seamless transition using containers to deploy and scale ML code from developer systems to scale-out infrastructure without any code modifications. Kubernetes and Kubeflow together truly democratise ML for organisations and makes it possible for all organisations to accelerate their journey to becoming an AI enabled company.

.

1 Dell EMC Kubeflow solution overview

The Dell EMC rack level solution for Kubeflow is comprised of pools of compute, storage and networking resources which are managed through a single point of rack management. All nodes in the rack are Dell PowerEdge servers handling compute, control, and storage functions, as assigned by the Metal as a Service (MAAS) management nodes.

1.1 Kubernetes

This architecture guide is based on Charmed Kubernetes - upstream code of Kubernetes release 1.16 delivered by Canonical. Canonical commercially distributes and supports the pure upstream version of Kubernetes. Ubuntu is the reference operating system for Kubernetes deployments, making it an easy way to build Kubernetes clusters. In Ubuntu, Kubernetes is delivered in the form of snaps - the universal Linux app packaging format - which dramatically simplifies the installation and upgrades of components.

Canonical's Discoverer programme family of services provides the service to design, deploy, manage and support customer clouds in POC, development, pre-production and production environments.

Canonical reference architectures are delivered on a converged infrastructure approach, where any of the servers can accommodate more than one specific Kubernetes role or service simultaneously. This converged approach has many benefits, including simplicity of operation and management overhead. Canonical can also deploy Kubernetes in a more traditional manner, grouping servers per role - controllers, storage, and container pods.

1.2 Kubeflow

The architecture guide introduces the machine learning platform Kubeflow. Canonical is an active member of the Kubeflow community and strongly believes in its role in democratising AI/ML by making model training and deployment as frictionless as possible. Canonical delivers the upstream components of the Kubeflow ecosystem as the add-on to the Kubernetes, wrapping it with automation so that it can be easily deployed, managed and upgraded.

1.3 Core components

Component	Codename
Persistent Storage	Ceph RBD
Compute	Kubernetes Worker (Docker based)
Machine Learning Platform	Kubeflow
Networking	Flannel or Canal
Logging	Graylog
Monitoring	Prometheus

The standards-based APIs are the same between all Kubernetes deployments, and they enable customer and vendor ecosystems to operate across multiple clouds. The site specific infrastructure combines open and proprietary software, Dell EMC hardware, and operational processes to deliver cloud resources as a service.

The implementation choices for each cloud infrastructure are highly specific to the requirements of each site. Many of these choices can be standardised and automated using the tools in this reference architecture. Conforming to best practices helps reduce operational risk by leveraging the accumulated experience of Dell EMC and Canonical.

Canonical's Metal as a Service (MAAS) is used as a bare metal and virtual machine (VM) provisioning tool. The foundation cluster is composed of MAAS and other services (running in highly available (HA) mode) that are used to deploy, manage and update the Kubernetes cluster nodes.

1.3.1 MAAS (Metal as a Service) physical cloud

Metal as a Service (MAAS) is a complete automation of physical servers for data centre operation efficiency on premises, it is open source software supported by Canonical.

MAAS treats physical servers like virtual machines, or instances in the cloud. Rather than having to manage each server individually, MAAS turns bare metal into an elastic cloud-like resource.

MAAS provides management of a large number of physical machines by creating a single resource pool out of them. Participating machines can then be provisioned automatically and used as normal. When those machines are no longer required they are "released" back into the pool. MAAS integrates all the tools you require in one smooth experience. It includes:

- Web UI, optimised for mobile devices
- Ubuntu, CentOS, Windows, RHEL and SUSE installation support open source IP Address Management (IPAM)
- Full API/CLI support
- High availability
- IPv6 support
- Inventory of components
- DHCP and DNS for other devices on the network
- DHCP relay integration
- VLAN and fabric support
- NTP for the entire infrastructure
- Hardware testing
- Composable hardware support

MAAS works with any system configuration, and is recommended by the teams behind both Chef and Juju as a physical provisioning system.

1.3.2 Key MAAS Features:

Feature	Description
Automation	Automatic discovery and registration of every device on the network. BMC (IPMI, AMT and more) and PXE (IPv4and IPv6) automation.
Fast deployment	Zero-touch deployment of Ubuntu, CentOS, Windows, RHEL and SUSE. Deploys Linux distributions in less than 5 minutes.
Machine configuration	Configures the machine's network interfaces with bridges, VLANs, bonds and more. Creates advanced file system layouts with RAID, bcache, LVM and more.
DevOps integration	Integration with DevOps automation tools like conjure-up, Juju, Chef, Puppet, SALT, Ansible and more.
Pod management	Turns bare-metal servers into hypervisors, allowing automated creation of virtual machines, and presents them as new servers available for the deployment.
Network management	Observes and catalogues every IP address on the network (IPAM). Built-in highly available DHCP (active-passive) and DNS (active-active).
Service tracking	Monitors and tracks critical services to ensure proper operations.
General management	Comes with a REST API, Web UI and CLI.

1.3.3 Juju modelling tool

Juju is an open source application modelling tool that allows quick and efficient deployment, configuration, scaling, and operation of cloud infrastructures on public clouds such as AWS, GCE, and Azure; along with private clouds such as MAAS, OpenStack, and VMware VSphere.

The Juju store allows access to a wide range of best practice solutions which can be deployed with a single command. Juju can be used from the command line or through its powerful graphical representation of the model in the GUI.

1.3.4 Why use Juju?

Whether it involves machine learning, deep learning, container orchestration, real-time big data or stream processing, big software needs operations to be open source and automated.

Juju encapsulates all the ops knowledge required to automate the behaviour of the application.

1.3.5 Software versions

The following versions of software are part of this reference architecture:

Component	Version
Ubuntu	18.04 LTS (kernel 4.15)
Kubernetes	1.16
MAAS	2.6.x
Juju	2.7.x
KubeFlow	0.6

2 Dell EMC PowerEdge hardware specifications

The reference architecture solution is based on the Dell EMC PowerEdge R640, R740xd and R740xd2. The reference architecture uses the following rack and server specifications.

2.1 Dell EMC PowerEdge rack specifications

Component type	Component description	Quantity
Rack	Standard data center rack (1) with enough capacity to hold 12x 1RU nodes, and 3x 1RU switches	1
Chasis	Dell PowerEdge R740xd (3 Infras3 Cloud nodes) Dell PowerEdge R740 (3 Cloud nodes) Dell PowerEdge R740xd2 (4 storage nodes)	6 3 4
Data switches	S5248F-ON (25G ToR, 48 ports)	2
iDRAC/Provisioning switch	S3048-ON	1

2.2 Dell EMC PowerEdge server specifications

Dell EMC PowerEdge R640 server specifications

Component type	Component description	Quantity
Processor	Intel Xeon Gold 6248 2.5G, 20C/40T, 10.4GT/s, 27.5M Cache, Turbo, HT (150W) DDR4-2933	2
Memory	32GB RDIMM 2666MT/s Dual Rank	12
Drive controller	PERC H740P Mini (Embedded)	1
Network Daughter card	Intel X710 Quad Port 10GbE SFP+, rNDC	1
Additional network card	Intel XXV710 Dual Port 25GbESFP28 PCIe Adapter, Low Profile	1
Boot system	DellEMC BOSS controller card Dell/Intel 240Gb M.2 SATA SSD (configured as RAID1)	1

Data drives	960GB SSD SAS Mix Use 12Gbps512n 2.5in Hot-plug Drive, PX05SV,3 DWPD,5256 TBW	6
OOB license	iDRAC9, Enterprise	1

Dell EMC PowerEdge R740xd server specifications

Component type	Component description	Quantity
Processor	Intel Xeon Gold 6248 2.5G, 20C/40T, 10.4GT/s, 27.5M Cache, Turbo, HT (150W) DDR4-2933Intel Xeon Gold6248 6126 2.6G, 12C/24T, 10.4GT/s , 19.25M Cache, Turbo, HT (125W) DDR4-2666	2
Memory	32GB RDIMM 2666MT/s Dual Rank	12
Drive controller	PERC H740P Mini (Embedded)	1
Network Daughter card	Intel X710 Quad Port 10GbE SFP+, rNDCIntel X520 DP 10Gb DA/SFP+, + I350 DP 1Gb Ethernet	1
Additional network card	Intel XXV710 Dual Port 25GbESFP28 PCIe Adapter, Low Profile	1
Boot system	DellEMC BOSS controller card Dell/Intel 240Gb M.2 SATA SSD SSDSCKJB240G7R (configured as RAID1)	1
Data drives	960GB SSD SAS Mix Use 12Gbps512n 2.5in Hot-plug Drive, PX05SV,3 DWPD,5256 TBW	6
OOB license	iDRAC9, Enterprise	1

Dell EMC PowerEdge R740xd2 server specifications

Component type	Component description	Quantity
Processor	Intel Xeon Gold 6248 2.5G, 20C/40T, 10.4GT/s, 27.5M Cache, Turbo, HT (150W) DDR4-2933Intel Xeon Gold6248 6126 2.6G, 12C/24T, 10.4GT/s ,	2

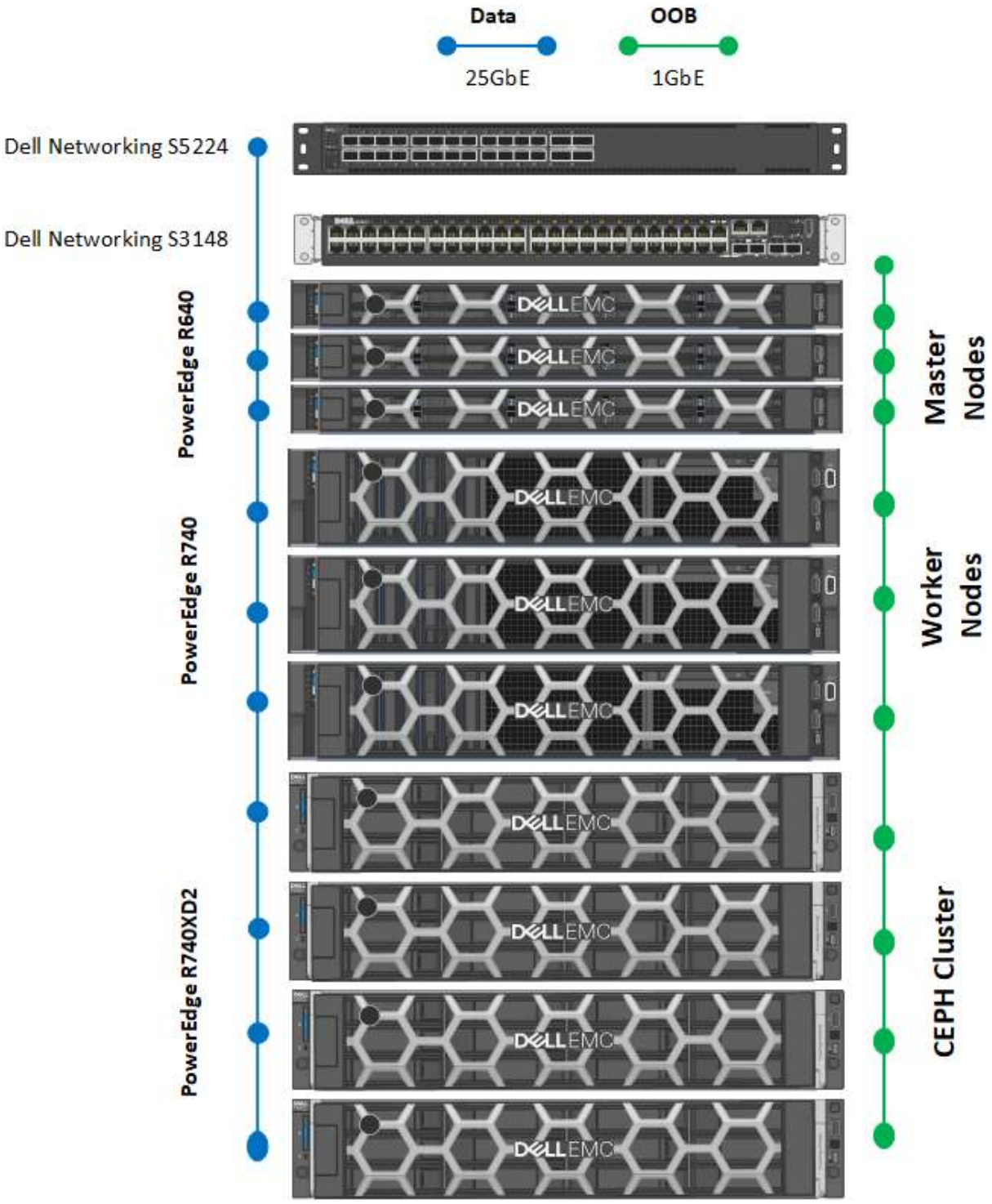
	19.25M Cache, Turbo, HT (125W) DDR4-2666	
Memory	32GB RDIMM 2666MT/s Dual Rank	12
Drive controller	PERC H740P Mini (Embedded)	1
Network Daughter card	Intel X710 Quad Port 10GbE SFP+, rNDCIntel X520 DP 10Gb DA/SFP+, + I350 DP 1Gb Ethernet	1
Additional network card	Intel XXV710 Dual Port 25GbESFP28 PCIe Adapter, Low Profile	1
Boot system	DellEMC BOSS controller card Dell/Intel 240Gb M.2 SATA SSD SSDSCKJB240G7R (configured as RAID1)	1
Data drives	960GB SSD SAS Mix Use 12Gbps512n 2.5in Hot-plug Drive, PX05SV,3 DWPD,5256 TBW	6
OOB license	iDRAC9, Enterprise	1

2.3 Rack layout

The reference deployment of Canonical Kubernetes on the Dell EMC PowerEdge utilises three nodes as infrastructure nodes, seven nodes as cloud nodes and an additional four for dedicated storage. The reference deployment uses the following purpose:

Node inventory

Node	**Purpose**
Rack1-MAAS1, Rack1-MAAS2, Rack1-MAAS3	Infra #1 (MAAS, Juju, LMA)
Rack1-cloud1, Rack1-cloud2, Rack1-cloud3, Rack1-cloud4, Rack1-cloud5, Rack1-cloud6,	Converged node handling Kubernetes components
Rack1-storage1, Rack1-storage2, Rack1-storage3, Rack1-storage4	Storage functions


Data
OOB
25GbE
1GbE
Dell Networking S5224
Dell Networking S3148
PowerEdge R640
PowerEdge R740
PowerEdge R740XD2
Master Nodes
Worker Nodes
CEPH Cluster

3 Network architecture

A Dell EMC KubeFlow solution is agnostic to the top of rack (ToR) switch a customer may choose for out-of-band (OOB) management. The reference implementation in this document uses the Dell EMC S3048-ON switch for this purpose. Two of the Dell EMC Networking S5248F-ON switches are used at the leaf-layer of the standard leaf-spine topology. The two switches are used to implement high availability on the data network. A pair of switches of similar or better capacity may be added at the spine-layer of the topology, if desired.

3.1 Infrastructure layout

The network consists of the following major network infrastructure layouts:

- Data network infrastructure: The server NICs and the leaf switch pair. The leaf switches are connected to the data center user networks and carry the main service traffic in / out of the reference architecture.
- Management network infrastructure: The BMC management network, which consists of iDRAC ports and the OOB management ports of the switches, are aggregated into a 1-rack unit (RU) Dell EMC PowerConnect S3148 switch. This 1-RU switch in turn can connect to the data center management network.
- MAAS Services: The MAAS Rack Controllers (see below) provide DHCP, IPMI, PXE, TFTP and other local services on the provisioning and iDRAC network. Ensure that the MAAS DCHP server is isolated from the data center DHCP server.

3.2 Network components

The following component blocks make up this network:

- Server nodes
- Leaf switches and networks
- VLANs
- Out-of-Band Management switch and network

3.3 Server nodes

To create a highly available solution, the network must be resilient to the loss of a single network switch, network interface card (NIC) or bad cable. To achieve this, the network configuration uses channel bonding across the servers and switches.

There are several types (or modes) of channel bonding, however only one is recommended and supported for this solution: 802.3ad or LACP (mode=4)

The endpoints for all nodes are terminated to switch ports that have been configured for LACP bonding mode, across two Dell EMC S5248F-ON’s configured with VLT across them. For details regarding network configuration on the servers, please contact your Dell EMC services and sales representative.

Supported channel bonding modes

Node type	Channel bonding type
Infrastructure nodes	802.3ad (LACP mode 4, channel fast)
Cloud nodes	802.3ad (LACP mode 4, channel fast)

3.4 Leaf switches

This reference implementation uses two Dell EMC Networking S5248F-ON switches. There is a redundant physical 2x40GbE connection between the two switches. The recommended architecture uses Dell EMC VLT between the switches in the leaf pair.

3.5 VLANs

This reference architecture implements at a minimum four separate networks through Layer-2 VLANs. Multiple networks below can be combined into single subnet based on end user requirements.

VLAN	Channel bonding type
OOB Management	Used for the iDRAC network
Internal	Used for cluster provisioning, monitoring and management
External	Used for communication between cluster components, as well as external access to the workloads, also for consuming persistent storage resources by the workloads
Storage (cluster)	Used for replicating persistent storage data between units of Ceph

3.6 Out-of-Band management network

The Management network of all the servers is aggregated into the Dell EMCPowerConnect S3148-ON in the reference architecture. One interface on the Out-of-Band (OOB) switch provides an uplink to a router/jumphost.

The OOB management network is used for several functions:

- The highly available software uses it to reboot and partition servers.
- When an uplink to a router is added and the iDRACs are configured to use it as a gateway, there are tools for monitoring the servers and gathering metrics on them. Discussion of this topic is beyond the scope of this document. Contact your Dell EMC sales representative for additional information.

4 Cluster Infrastructure components

The infrastructure nodes are composed of the following services and tools that are configured for high availability:

- MAAS
- Juju
- Monitoring
- Log aggregation

This section provides details about how each of these components work.

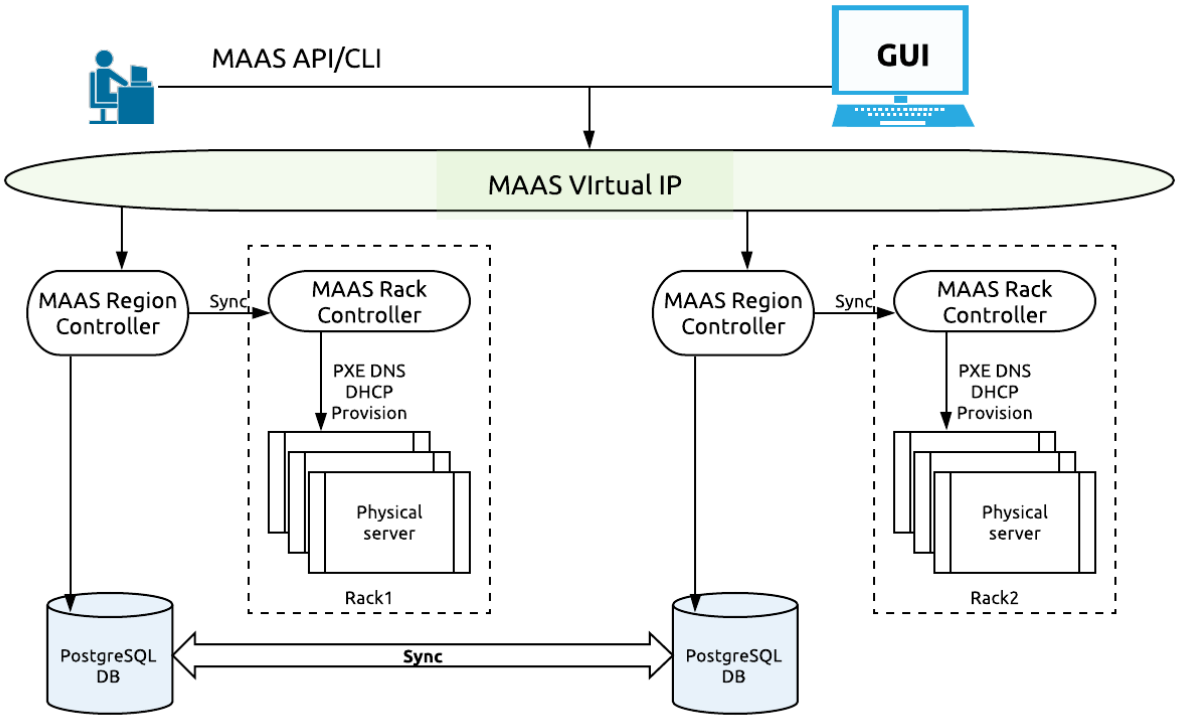
4.1 How MAAS works

MAAS has a tiered architecture with a central Postgres database backing a region controller (regiond) that deals with operator requests. Distributed rack controllers, or (rackd) provide high-bandwidth services to multiple racks. The controller itself is stateless and horizontally scalable, and only presents a REST API.

Rackd provides DHCP, IPMI, PXE, TFTP and other local services. They cache large items like operating system install images at the rack level for performance but maintain no exclusive state other than credentials to talk to the controller.

MAAS is a mission critical service that provides infrastructure coordination upon which HPC and cloud infrastructures depend. High availability in the region controller is achieved at the database level. The region controller will automatically switch gateways to ensure high availability of services to network segments in the event of a rack failure.

4.2 MAAS logical design



4.3 Juju components

For an overview of Juju, refer to the Juju modelling tool. This section discusses the working of different components of Juju.

4.4 Juju controller - the heart of Juju

The Juju controller manages all the machines in the running models and responds to the events that are triggered throughout the system. It also manages scale-out, configuration, and placement of all models and applications.

Juju controller has to be located in the same physical segment of the network as OpenStack cluster, and be able to execute calls to MAAS API and connect to OpenStack Cluster nodes.

Juju controller is supposed to be created using a set of KVM Virtual machines mentioned in the previous steps.

4.5 Charms

Charms are a collection of scripts that contain all of the operations necessary to deploy, configure, scale, and maintain cloud applications with Juju. Charms encapsulate a single application and all the code and know-how that it takes to operate it, such us how to combine and work with other related applications, or how to upgrade it.

Charms also allow a hierarchy, with subordinate charms to complement a main service.

4.6 Bundles

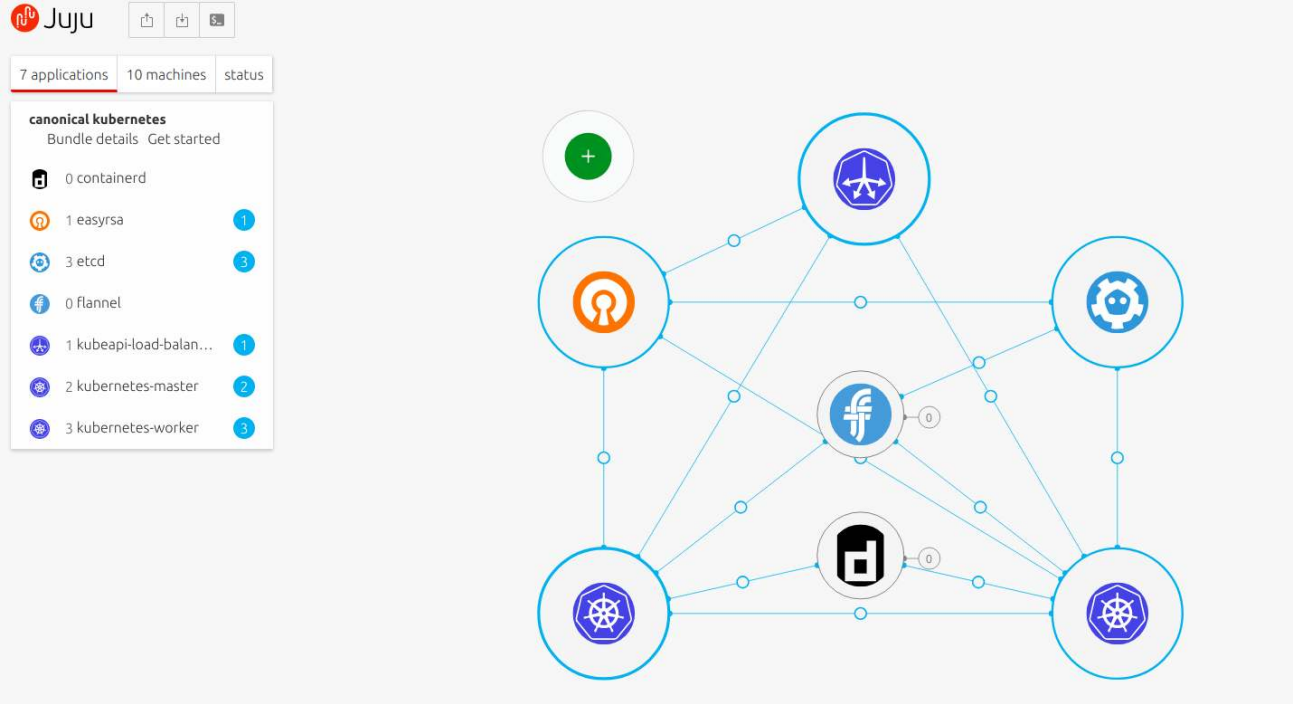
Bundles are ready-to-run collections of applications that are modelled to work together and can include particular configurations and relations between the software to be deployed.

Bundles may also be optimized for different deployment scenarios of the same software. For example, a scale-out, production-ready version like the Canonical Distribution of Kubernetes, or a development-friendly test version like Kubernetes Core.

Bundles perform the following functions:

- Install
- Configure
- Connect
- Upgrade and update
- Scale-out and scale-back
- Perform health checks
- Undertake operational actions
- Benchmark

Juju supports UI representation of deployed bundles. The Graphic User Interface allows dynamic manipulation with a cluster’s configuration options and layout prior to the bundle deployment, and also dynamic configuration during the lifetime.



4.7 Provision

Juju allows to specify desired number of machines to be added to the cluster. It is possible to request machines based on various constraints, alternatively Juju requests machines automatically.

4.7.1 Deploy

Juju allows to layout the application across the existing machines, dynamically scale application on existing infrastructure or onto new machines. Also it is possible to re-deploy entire application infrastructure to another cloud, with a few clicks of the mouse.

4.8 Monitor and manage

The Juju controller manages:

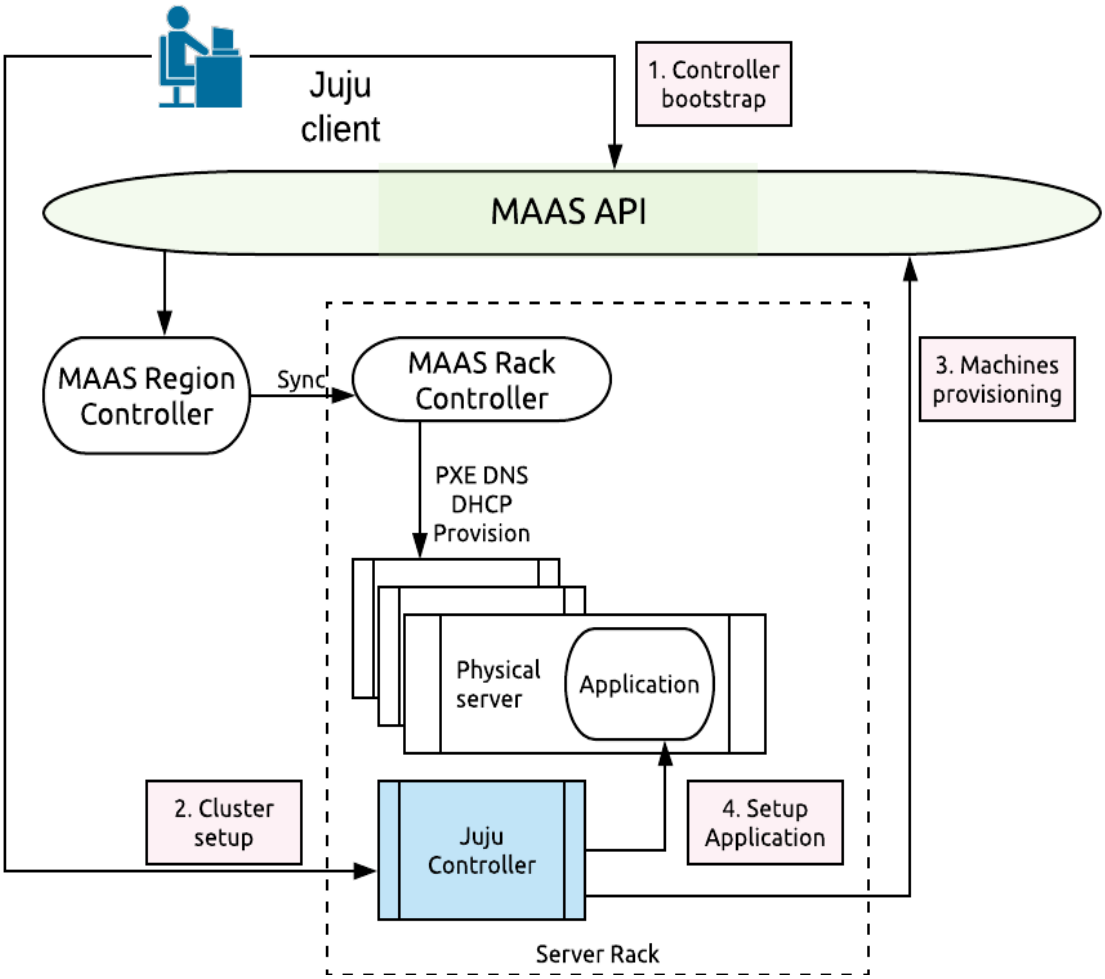
- Multiple models
- All machines (physical and virtual) in all running models
- Scale out, configure and placement
- User accounts and identification
- Sharing and access

4.9 Comparing Juju to any configuration management tool

Juju provides a higher level of abstraction, and supplies the tools needed to manage the full scope of operations beyond deployment and configuration management, regardless of the machine on which it runs.

One of the main advantages of Juju is the dynamic configuration ability, which enables users to:

- Reconfigure services on the fly.
- Add, remove, or change relationships between services.
- Scale in or out with ease, shares the operational knowledge and makes the most of the wider community.



4.10 Monitoring

Canonical Kubernetes solution includes a monitoring suite based on the best open source tools available.

From an architectural standpoint, monitoring suite consists of Telegraf (host metric collection), Prometheus (monitoring server) and Grafana (monitoring dashboard).

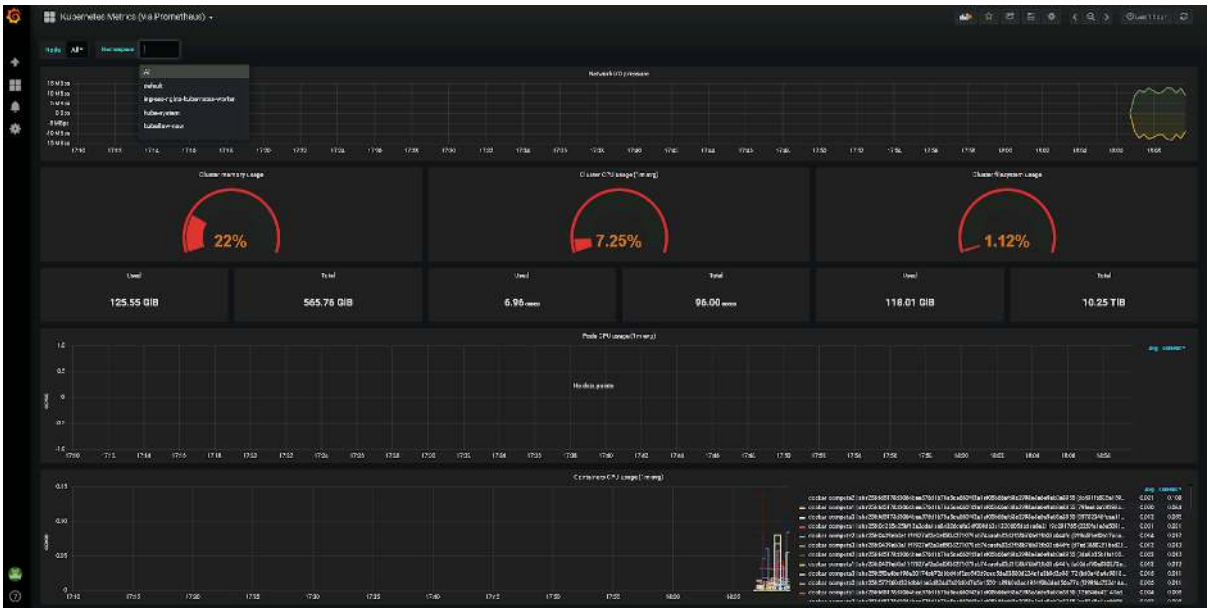
4.10.1 Prometheus

Prometheus is a systems and services monitoring system. It collects metrics from configured targets at given intervals, evaluates rule expressions, displays the results, and can trigger alerts if some condition is observed to be true.

4.10.2 Grafana

Grafana is the leading graph and dashboard builder for visualizing time series metrics.

Kubernetes-related dashboard



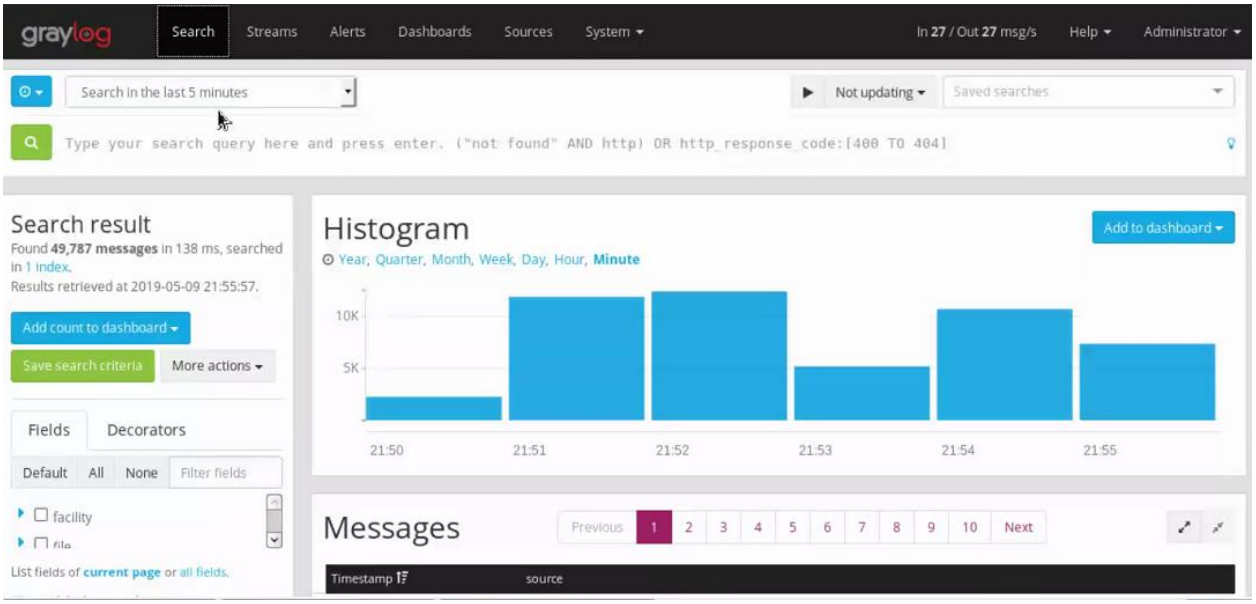
4.10.3 Telegraf

Telegraf is a client-side system that collects information about the status of the host services and makes them available for pulling by any monitoring solutions (in this architecture Prometheus).

4.11 Log Aggregation

The solution also implements the Graylog suite for log aggregation, which makes it easy for customers to have visibility on the different logs from their cloud services without accessing them directly.

Sample of log aggregation dashboard.



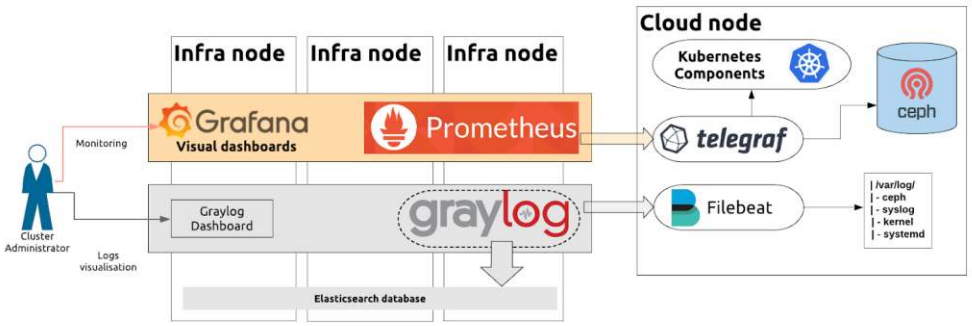
4.11.1 Elasticsearch

Elasticsearch is a distributed database used for storing indexed logs and acts as the backend for Graylog.

4.11.2 Filebeat

As a log forwarder, Filebeat tails various log files on the client side and quickly sends this information to Graylog for further parsing and enrichment, or to Elasticsearch for centralized storage and analysis.

Overall the composition of monitoring and logging tools and their relation to Kubernetes and storage components is represented below.



5 Charmed Kubernetes components

This chapter presents detailed information about the Kubernetes components included as charms in Charmed Kubernetes.

5.1 Storage charms

Ceph is a distributed software-defined storage solution designed to provide excellent performance, reliability, and scalability. Canonical uses Ceph by default for storage; however, this can be replaced by, or complemented with, another storage solution.

Ceph cluster consists of the following logical components, each deployed with its own charm:

- Ceph monitor (cluster logic and the access endpoint)
- Ceph OSD (cluster capacity)
- Rados Gateway (S3 and OpenStack Swift functionality)

5.2 Kubernetes charms

5.2.1 EasyRSA

EasyRSA is the PKI used by Charmed Kubernetes to build and deploy x509 certificates used to secure the communication between the various layers of Charmed Kubernetes: etcd cluster, Kubernetes API and others.

It is a simple CLI tool that Juju leverages via SSH implemented over the charm relation protocol. Charms consuming certificates can query the relation to get access to the CA cert, or ask for the creation of separate server or client certificates.

5.2.2 Kubernetes-master

The Kubernetes Master is in charge of managing the Kubernetes cluster. It is made of 3 underlying components:

- The API Server is in charge of being the interface between the administrators and users of the cluster and the cluster itself, but also provides services within the cluster
- The Scheduler is in charge of allocating pods to their nodes
- The Controller Manager is in charge of maintaining the cluster in a desired state

All of these applications are stateless and can be scaled in and out very easily. The state of the cluster is saved in the etcd database.

5.2.3 Kubernetes-worker

The Worker is the component that runs the compute tasks in Kubernetes. It is based on two core elements:

- The kubelet is the Kubernetes Agent that executes the plan and drives Docker and manipulates the core objects of the Kubernetes API
- The kube-proxy is a service that drives iptables to make sure that the translation between services and pods is efficiently rendered.

5.2.4 Etcd

Etcd is the database holding the state of the cluster at any point in time. It is a critical element in Kubernetes, since damaging the DB (or worse, losing it) will irreparably impact the state of the cluster. Also, anyone with write access on etcd can potentially modify the state of the cluster.

5.2.5 Flannel

Flannel is a virtual network that gives a subnet to each host for use with container runtimes.

This charm will deploy flannel as a background service, and configure CNI for use with flannel, on any principal charm that implements the kubernetes-cni interface.

As Flannel is usually great for the Proof-of-Concept deployments, it is highly recommended to use other networking solutions such as Canal (Calico/Flannel) for production deployments.

5.2.6 API Load Balancer

The API load balancer is a simple nginx service that exposes a proxied endpoint to the Kubernetes API. It converts the API default port 6443 to run on the more classic 443.

6 Kubeflow as machine learning platform

Kubeflow is a composition of open source tools for developing, orchestrating, deploying, and running scalable and portable machine learning workloads on top of Kubernetes.

At the high level Kubeflow is comprised of the following components and functionalities:

- Jupyter notebooks - an open-source application that allows users to blend code, equation-style notation, free text and dynamic visualisations to give data scientists a single point of access to their experiment setup and notes.
- Katib (hyper-parameter tuning) - Hyper-parameters are set before the machine learning process takes place. These parameters (e.g. topology or number of layers in a neural network) can be tuned with Katib. Katib supports various ML tools such as TensorFlow, PyTorch and MXNet, making it easy to reuse previous experiments results.
- Pipelines - facilitate end-to-end orchestration of ML workflows, management of multiple experiments and approaches, as well as easier re-use of previously successful solutions into a new workflow. This helps developers and data scientists save time and effort.
- Serving - Kubeflow makes two service systems available: TensorFlowServing and Seldon Core. These allow multi-framework model serving and the choice should be made based on the needs of each project.
- Training - model training is possible with various frameworks, in particular TensorFlow, Chainer, MPI, MXNet and PyTorch. These cover the most popular frameworks in the data science and AI/ML space, ensuring that developers are able to use Kubeflow's MLOps features with their favourite tools.

Utilising Kubernetes as the infrastructure for Kubeflow installation unlocks the possibility for dynamic scaling of platform components, as well as addition of new or upgrading existing components at any time.

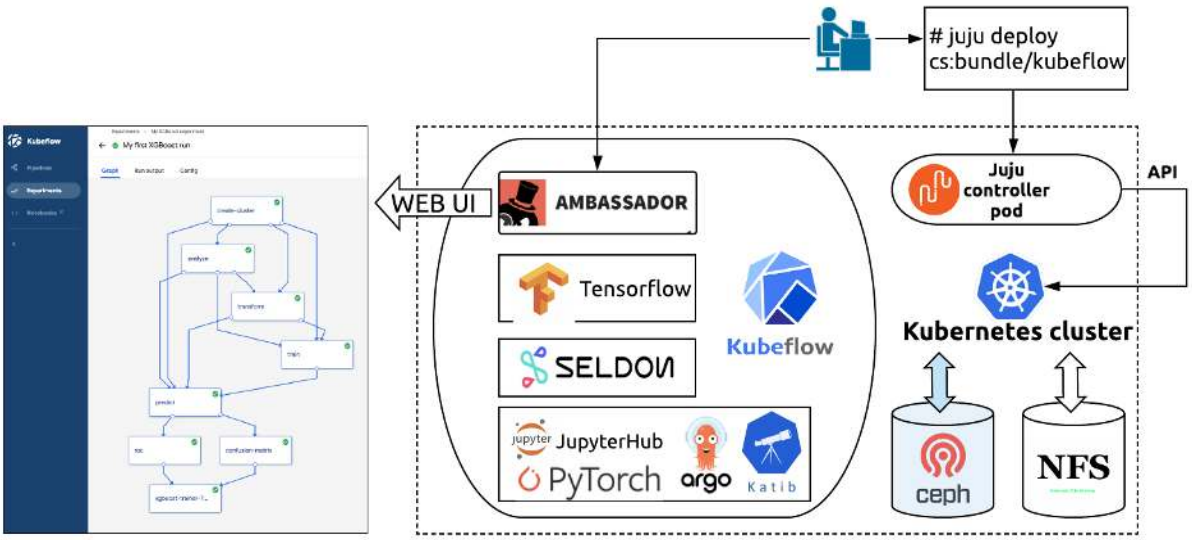
6.1.1 Kubeflow and charms

Canonical delivers Kubeflow components in an automated fashion, using the same approach and toolset as for deploying the infrastructure and Kubernetes cluster - with help of Juju.

The current version of Juju is able to register an existing Kubernetes cluster as the target and deploy software on top of that using operator approach with help of Juju Kubernetes charms.

It either utilises the existing Juju controller that is used for deploying the infrastructure, or bootstraps new controller on top of Kubernetes as the pod. In this reference architecture, the latter approach is used.

Once deployed, Juju registers the model of the Kubeflow cluster that describes the desired components layout, configuration options if necessary, optionally the details of images and scaling options.



The end user experience when deploying and operating the deployment on top of bare metal infrastructure and Kubernetes is identical, however the same charms can not be used for both deployment substrates.

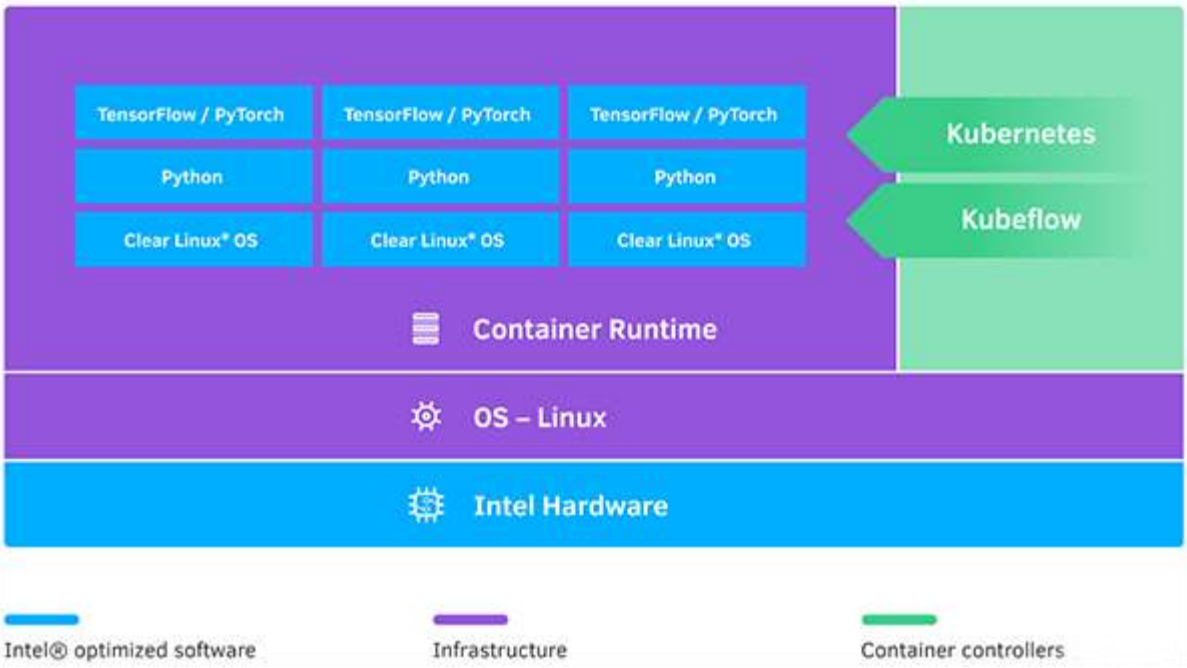
Canonical has developed and tested the set of charms automating the delivery of the most demanded components of Kubeflow, providing the reference bundle that can be re-used when deploying to a Kubernetes cluster.

6.1.2 Intel software and system stacks

Kubeflow components responsible for training and serving the models are able to be deployed on the commodity hardware, however it is recommended to choose the proper combination of software and hardware to achieve the best performance.

Intel provides optimized frameworks, tools, libraries and SDKs to quickly develop, train and deploy solutions, including Intel® Distribution for Python, Intel® Data Analytics Acceleration Library (Intel® DAAL) library, Intel-optimized XGBoost and Intel-optimized Tensorflow for Intel® Xeon processors. Intel® Data Analytics Acceleration Library (Intel® DAAL) library is an easy-to-use library that boosts big data analytics and machine learning performance.

Intel has also developed a Deep learning reference stack - a set of container images based on optimised libraries that utilise additional features of Second Generation Intel Xeon Scalable processors.



Using these images with Kubeflow allows the developer to dramatically increase the speed of models training (TensorFlow and PyTorch training jobs) and model serving (SeldonCore). The stack reference includes various images allowing the users to choose the set of optimised libraries based on their unique use case, while they all benefit from the hardware acceleration.

To learn more, go to https://software.intel.com/en-us/

6.2 How to use Intel-optimized AI stack?

As mentioned earlier, Intel develops deep learning stack along with distributing individual optimized containers. Additionally, For optimized distributed containerized experience on Tensorflow, Intel is working with Google to provide a Dockerfile with Intel-optimized Tensorflow + Horovod for an easier user experience on Intel® Xeon processors. This section will provide the steps on how to use Intel-optimized Tensorflow on MNIST and Intel brain imaging deep learning use case on Canonical Kubernetes (K8s) stack

1. Clone the Tensorflow github repository. The dockerfile is compatible with 1.x Tensorflow versions

```
git clone  https://github.com/tensorflow/tensorflow
git checkout r1.15
```

2. Build the docker image with Intel-optimized Tensorflow + Horovod + OpenMPI DockerFile

```
cd tensorflow/tensorflow/tools/dockerfiles/dockerfiles/mkl_horovod/

docker build --build-arg TF_PACKAGE=intel-tensorflow --build-arg
TF_PACKAGE_VERSION=1.15 -t intel-mkl-tf-horovod:1.0
```

3. Start the container and do a quick sanity check if Intel® MKLDNN is enabled

```
Docker run -it intel-mkl-tf-horovod:latest bash

python -c "import tensorflow; print(tensorflow.pywrap_tensorflow.IsMklEnabled())"
```

This should print “True”. If the sanity check prints “False”, the common cause would be building the image from a wrong source or missing “mkl” configuration flags.

4. Train the MNIST dataset keras sample available using the wget command at: https://github.com/horovod/horovod/blob/master/examples/keras_mnist_advanced.py

For quick debugging

```
mpirun -np 2 python keras_mnist_advanced.py --epochs 2
```

5. Once successful, quit the container (Ctrl + D) and build the app docker image with Intel-optimized Tensorflow + Horovod + OpenMPI + MNIST sample and push it to the docker hub

- Create a DockerFile with the following instructions to build the app

```
FROM intel-mkl-horovod-tf:1.0
RUN apt-get update && apt-get install -y git
RUN wget
https://raw.githubusercontent.com/horovod/horovod/master/examples/keras_mnist_adv
anced.py
RUN  pip install keras h5py psutil
```

- Build the docker image

```
docker build -t intel-mkl-tf-horovod-mnist:1.0
```

- Push the intel-mkl-tf-horovod-mnist:1.0 image to the docker hub

https://docs.docker.com/engine/reference/commandline/push

6. Login to the Canonical K8s to run the MNIST sample. Create a yaml file “intel-mnist.yaml” to run the app as a mpijob. A sample yaml file we used is below. Note that all the workers are running in the same node. Change the “image” to the app image name you pushed into the docker hub

```
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: intel-mnist
spec:
  cleanPodPolicy: Running
  mpiReplicaSpecs:
     Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - command: ["/bin/sh"]
            args: ['-c', 'mpirun -np 2 python keras_mnist_advanced.py --epochs
2']
            image: <your docker hub>/intel-mkl-horovod-mnist:1.0
            name: intel-mnist
            resources:
               limits:
                cpu: 1
          nodeSelector:
               env: "mpi-test"
     Worker:
      replicas: 2
      template:
        spec:
          containers:
          - image: <your docker hub>/intel-mkl-horovod-mnist:1.0
            name: intel-mnist
            resources:
              limits:
                cpu: 1
            restartPolicy: OnFailure
          nodeSelector:
                  env: "mpi-test"
  slotsPerWorker: 1
```

7. Assuming the mpi-operator is installed on your K8s, submit a kubernetes job to train the mnist model

```
kubectl apply -f intel-mnist.yaml
```

8. Watch the launcher and 2 workers train the mnist model.

```
kubectl get pods | grep intel-mnist
```

```
intel-mnist-launcher-xn77m                  1/1     Running                  0          8m59s
intel-mnist-worker-0                        1/1     Running                  0          8m59s
intel-mnist-worker-1                        1/1     Running                  0          8m59s
```

9. Train(repeat steps 7 and 8) with varying mpi process and workers. Update the yaml file “args” from mpi process 2 to 4, 8 and the “replicas” from 2 to 4, 8 and notice the training time for each epoch drop significantly while maintaining the accuracy

    - With 4 processes

```
Epoch 1/2
  3/117 [..............................] - ETA: 13:36 - loss: 2.2748 - accuracy:
0.1719  3/117 [..............................] - ETA: 13:39 - loss: 2.2681 -
accuracy: 0.1901  3  5/117 [>.............................] - ETA: 13:08 - loss:
2.2269 - accuracy: 0.1953  5/117 [>.............................] - ETA: 13:08 -
loss: 2.2189 - accuracy: 0.2047  5  7/117 [>.............................] - ETA:
12:46 - loss: 2.1615 - accuracy: 0.2422  7/117 [>.............................] -
ETA: 12:46 - loss: 2.1777 - accuracy: 0.22
……
Test loss: 0.049240677943918856
Test accuracy: 0.9836999773979187
Test loss: 0.049240677943918856
Test accuracy: 0.9836999773979187
Test loss: 0.049240677943918856
Test accuracy: 0.9836999773979187
Test loss: 0.049240677943918856
Test accuracy: 0.9836999773979187
```

    - With 8 processes

```
Epoch 1/2
2/58 [>.............................] - ETA: 7:33 - loss: 2.2766 - accuracy:
0.1523 2/58 [>.............................] - ETA: 7:36 - loss: 2.2647 -
accuracy: 0.1836 2/58 [>. 3/58 [>.............................] - ETA: 6:52 -
loss: 2.2329 - accuracy: 0.2083 3/58 [>.............................] - ETA: 6:52
- loss: 2.2373 - accuracy: 0.2083 3/58 [>. 4/58 [=>.............................]
- ETA: 6:32 - loss: 2.1646 - accuracy: 0.2539 4/58
[=>.............................] - ETA: 6:31 - loss: 2.1897 - accuracy: 0.2285
4/58 [=> 5/58 [=>.............................] - ETA: 6:12 - loss: 2.1557 -
accuracy: 0.2250 5/58 [=>.............................] - ETA: 6:11 - loss: 2.1438
- accuracy: 0.2547 5/58
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
```

```
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
Test loss: 0.06327981026750058
Test accuracy: 0.980400025844574
```

Intel Unet Brain Imaging use case

Try this procedure on the brain imaging Unet sample. The instruction to run the Unet model is available at https://github.com/IntelAI/unet/tree/master/3D. Clone the repository and follow steps 1 through 5 from the Readme and run the train_horovod.py script

```
mpirun -np 2  --map-by slot --rank-by core --oversubscribe --report-bindings
python train_horovod.py  --bz 2 --data_path <dataset_dirr>Task01_BrainTumour --
epoch 2
```

1. Build the Unet app docker image with Intel-optimized Tensorflow + Horovod + OpenMPI + Unet brain imaging sample and push it to docker hub

- Create a DockerFile with the following instructions to build the app

```
FROM intel-mkl-tf-horovod:1.0
RUN  git clone https://github.com/IntelAI/unet.git \
    pip install keras tqdm h5py psutil nibabel
```

- Build the docker image

```
docker build -t intel-mkl-tf-horovod-brain-imaging:1.0
```

- Push the intel-mkl-tf-horovod-brain-imaging:latest image to the docker hub

https://docs.docker.com/engine/reference/commandline/push

10. Login to the Canonical K8s to run the brain imaging app. Create a yaml file “intel-brain-imaging.yaml” and a PVC volume to mount the containers with brain imaging dataset. A sample yaml file to run a mpijob we used is below.

```
apiVersion: kubeflow.org/v1alpha2
kind: MPIJob
metadata:
  name: intel-brain-imaging-4
spec:
  cleanPodPolicy: Running
  mpiReplicaSpecs:
     Launcher:
      replicas: 1
```

```
      template:
        spec:
          containers:
          - command: ["/bin/sh","-c"]
            args: ['-c', 'cd unet/3D; mpirun -np 1 --map-by slot --rank-by core -
-oversubscribe --report-bindings python train_horovod.py --bz 2 --data_path /data
--epoch
2']
            image: <your docker hub>/intel-mkl-horovod-tf-brain-imaging:1.0
            name: intel-brain-imaging-4
            volumeMounts:
            - mountPath: "/data"
              name: intel-brain-volume
              readOnly: true
            resources:
               limits:
                cpu: 1
          volumes:
            - name: intel-brain-volume
              persistentVolumeClaim:
                 claimName: nfs1
          nodeSelector:
         env: "mpi-test"
     Worker:
      replicas: 1
      template:
        spec:
          containers:
          - image: <your docker hub>/intel-mkl-horovod-tf-brain-
imaging:1.0
            name: intel-brain-imaging-4
            volumeMounts:
            - mountPath: "/data"
              name: intel-brain-volume
              readOnly: true
            resources:
              limits:
                cpu: 1
            restartPolicy: OnFailure
          volumes:
            - name: intel-brain-volume
              persistentVolumeClaim:
                 claimName: nfs1
     nodeSelector:
           env: "mpi-test"
  slotsPerWorker: 1
```

11. Assuming the mpi-operator is installed on your k8s, submit a kubernetes job to train the unet model

```
kubectl apply -f intel-brain-imaging.yaml
```

12. Watch the launcher and workers train the Unet model. Also you can verify that MKLDNN is enabled from the launcher jobs

```
Kubectl get pods
kubectl logs <launcher job id>
```

Log snippet

```
….
Data Format = channels_last
Tensorflow version: 1.15.0
Intel MKL-DNN is enabled = True
Keras API version: 2.3.1
….
….
```

A github repo with detailed instructions on both these use cases will be released soon. The article will be updated with the repo details for users to seamlessly reproduce

7 Conclusion

DellEMC, Canonical and Intel deliver this joint solution that allows organizations to start exploring the capabilities of the open source machine learning platform Kubeflow, providing the services of AI enablement.