10G Lab
Future Infrastructure Group

April 2022

CableLabs®

Best Practices for Deploying 5G into a Shared Environment

Prepared by
Carmela Stuart, Director / Future Infrastructure Group | c.stuart@cablelabs.com
Ryan Fuerst, Senior DevOps Engineer / Future Infrastructure Group | r.fuerst@cablelabs.com
Sanjhana Jayagopal, Network Engineer / Future Infrastructure Group | s.jayagopal@cablelabs.com
Ashwini Kadam, Senior Engineer / Future Infrastructure Group | a.kadam@cablelabs.com
Randy Levensalor, Principal Architect / Future Infrastructure Group | r.levensalor@cablelabs.com

Executive Summary

Maintaining and managing separate wireline and wireless access networks with overlapping capabilities is not only inefficient and costly, it limits the types of services that operators (both those operating mobile networks today and those seeking to build new mobile capability) will be able to provide in the future. Convergence merges these technologies into a singular, multi-modal communication platform that will help reduce network complexity, cut operational costs, improve quality, and create new business and service opportunities.

The business value gained from converged networks is multi-faceted. A unified framework for a seamless connectivity experience across home, office, and public areas is enabled through a system that allows for flexible and programmable deployment of network functions across multiple operator locations and a common management framework for

- zero-touch provisioning;
- monitoring;
- fault management;
- end-to-end quality of service (QoS); and
- security and analytics across wireline, Wi-Fi, and wireless networks.

For vendors, the decoupling of hardware from software in an operator's shared environment facilitates on-demand delivery of new features and bug fixes to its 5G applications by utilizing the operator's continuous integration, continuous delivery (CI/CD) pipeline. Furthermore, application development is simplified with a minimum viable hardware configuration and less customization per operator. The onus of resource tuning (memory, compute, storage, etc.) is on the operator, allowing operators to have more control over an application's performance and removing some of that burden from the vendor applications.

The first step in the evolution toward full convergence is platform convergence, where virtualized or containerized network stacks share hardware (compute storage, and networking), and these software-based systems are deployed and managed by a common management layer. This paper explores the early efforts of deploying 5G network stacks into a shared environment and catalogs the challenges and approaches taken to ensure that an optimized environment to host multiple network stacks is achieved. At the time of writing, the focus was on deployment and configuration.

Introduction

CableLabs launched a new "10G Lab" where the goal is to deploy multiple 5G cores and vRAN systems alongside a cable hybrid fiber-coax (HFC) network. Initially, the network stacks deployed will be containerized systems that are co-located on shared infrastructure but run independently of one another. Over time and as the convergence initiative and the specifications to support full convergence evolve, the lab will also evolve to bring a fully converged core and access network to fruition.

During the deployment of the first 5G core, several considerations were addressed for security, networking, resource allocation, performance, automation, and general ease of management through the separation of vendor applications. The technical details associated with decisions made to address these areas are highlighted in the sections that follow.

CableLabs' efforts to co-locate more than one network stack onto this shared infrastructure have raised some challenges, and addressing these challenges early may circumvent operators from facing the same challenges as they work with vendors to ensure their network stacks follow best practices that improve the security and manageability of their products in a shared environment.

Achieving the goal of co-location of network stacks on shared infrastructure utilizing a common management interface and full support of convergence in the future will lead to many benefits, such as

- reduced power consumption, space, and hardware costs;
- elimination of overprovisioned environments by matching resources to demand;
- automated versus manual deployment of services; and
- ability to create dynamic, customized network topologies at different geographical locations that improve throughput and reduce latency for subscribers.

Decoupling the network software from proprietary hardware increases the cadence of upgrades to a system, which reduces the time to market for new capabilities and optimizations.

1 Baseline Environment

The 10G Lab environment is based on Red Hat OpenShift Container Platform 4.8 (OCP) hosting multiple Kubernetes clusters on bare metal commercial off-the-shelf (COTS) x-86 based servers with both Intel and AMD CPUs, with Arista and Cisco switches. The examples provided in this paper are specific to this installation. However, much of the information provided could be applicable across any flavor of Kubernetes container orchestration platform running on any type of COTS servers and switches.

2 Hardware Requirements

The minimum required hosts for OCP running on bare metal are three controller plane nodes and two compute nodes. The CPU and memory sizing of these nodes is workload dependent and can be scaled up as needed by adding additional CPU or memory to the compute nodes or adding additional compute nodes.

OCP will work with the networking cards that the physical host supports, but the end user needs to consider expected bandwidth and features required for their workloads and plan accordingly. For the 10G Lab, there is a mix of 25GbE and 100GbE cards that support single root I/O virtualization (SR-IOV) in the environment.

If machine learning or AI workloads are planned, then graphics processing unit (GPU) resources may need to be included on the compute nodes where these workloads will be run.

For storage, it is highly recommended that local installation is run on SSD drives for the best performance. A minimum of 100 GB per node is required; however, the maximum amount of storage is workload dependent and needs to be planned accordingly. For the 10G Lab to meet workload requirements, CableLabs implemented an external shared storage cluster using Red Hat Ceph Storage 5. Ceph is an open-source, software-defined storage solution designed to address block, file, and object storage needs. This software package can be natively deployed to a specific OCP cluster or deployed standalone, as has been done in the 10G Lab to support multiple clusters. If deployed natively, the compute nodes will need additional storage beyond the minimum requirements depending on the amount of storage required by the workloads.

3 Container Platform Installation and Configuration

Installation and configuration of a container platform is a multi-step process. This includes configuration of the physical hardware and its subcomponents, setting up the proper networking at the switch, installing the container orchestration software, and integrating everything into a CI/CD pipeline. Each of these steps requires inputs that are prone to error when done manually. To solve this, CableLabs implements infrastructure as code (IaC) using Ansible as the automation framework. Ansible is an open-source project with a wide range of vendor-supported modules that makes working with non-homogeneous environments easier. The 10G Lab contains multiple hardware vendors, which makes an agnostic tool highly useful for this work. Upstream Ansible documentation can be found at https://www.ansible.com.

Physical hardware configuration includes setting up the bare-metal COTS servers used to host the container platform software. This setup involves configuring the server management software to remotely access the hardware, upgrading firmware, configuring the bios, and configuring any local storage installed in the server. The Ansible automation for this includes authentication access, network settings (dns, ntp), required bios settings, and local storage configuration for the physical hardware.

Networking switch configuration involves setting up the ports the server is physically plugged into to match the networking setup for the cluster. This includes any specific VLANs or subnets that are required. Ansible automation for this includes setting up VLANs for the platform and other port-specific settings.

Installation of the container orchestration software, a Kubernetes-based system, is done after the previous tasks have been finished. It involves loading the software onto the bare-metal COTS servers. The Ansible automation for this configures the physical server to boot to the installation software and provides installation-specific variables required for install. Upon completion, this results in a running cluster with no manual input outside of running the ansible playbooks.

In the 10G Lab, CableLabs hosts the CI/CD pipeline on the container platform once it has been deployed; therefore, this requires an additional step to integrate ongoing configuration and management of the platform into that pipeline. The Ansible automation for this installs CableLabs' CI/CD pipeline of choice, Argo CD, into the clusters. Once that is complete, ongoing cluster maintenance and management is handled through GitOps. Examples of Ansible scripts used in the 10G Lab can be found at https://github.com/cablelabs/gitops.

4 Cluster Capabilities

The Kubernetes environment was extended to include several additional baseline services installed as needed on each cluster. Some of these services are primarily for cluster management, whereas others were required specifically to support a 5G vendor's network stack. Capabilities fall into three different categories: general capabilities, capabilities required by 5G applications, and external services (see Figure 1).

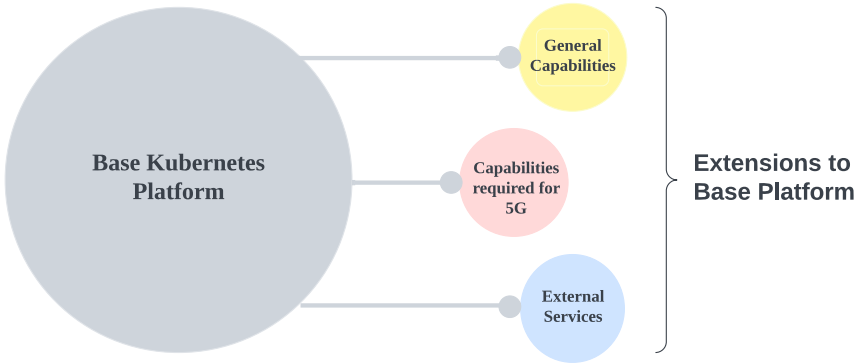

Figure 1: Platform Capabilities

4.1 General Management and Maintenance Capabilities

Kubernetes installers configure several core features by default. These enable scheduling workloads and managing clusters. There are several items that will need to be configured to improve the management and capabilities in the cluster. The capabilities listed in the section apply to running Kubernetes on bare metal along with integration into some existing CableLabs infrastructure.

- The Lightweight Directory Access Protocol (LDAP) is used to implement authentication against corporate LDAP server versus local accounts.
- Logging aggregates all the logs from the cluster to a default location, making it easier to analyze or visualize the data with Kibana. Link to upstream documentation.
- Garbage collection frees up cluster resources by removing terminated containers and images not referenced by any running pods.
- MetalLB provides load balancer type services to the cluster. Link to upstream documentation.
- Monitoring configures Prometheus to enable monitoring and alerting on a cluster.
- OpenShift container storage is used to provide persistent storage for the cluster. The 10G Lab has it configured to access the shared Ceph cluster in CableLabs' environment. Link to upstream documentation.
- A registry provides an internal, integrated container image registry for the cluster.
- The Performance Addon Operator is used to tune the running system to handle specific workloads, such as low latency, cloud-native network functions (CNFs), and Data Plane Development Kit (DPDK). Link to upstream documentation.
- KubeVirt enables management of virtual machines (VMs) within the platform. Link to upstream documentation.
- GitOps/Argo CD is the tool that will be used to manage the cluster after the initial deployment and can also manage the applications that use the cluster. Link to upstream documentation.

4.2 Capabilities Required by 5G Stacks

The capabilities required by 5G stacks are primarily to enable network protocols and performance that are needed when deploying network functions on Kubernetes. Low-latency and high-performance networking requirements currently cannot be met with a typical Kubernetes deployment. Some of the features, such as DPDK and SR-IOV, provide pods with direct access to the hardware.

- OCP service mesh (requires Jaeger, Kiali, and Elasticsearch)—The OCP service mesh is based upon Istio. Link to upstream documentation.
- Stream Control Transmission Protocol (SCTP)—A reliable message-based protocol that runs on top of an IP network.
- Single root I/O virtualization (SR-IOV)—An extension to the PCI Express (PCIe) specification, SR-IOV allows a device to separate access to its resources among various PCIe hardware functions. Link to upstream documentation.
- OVN-Kubernetes—Provides and overlay based networking implementation. Link to upstream documentation.

4.3 Services External to Cluster

These services are shared between multiple clusters and are used for both installing and running Kubernetes.

- HAProxy—Open-source load balancing solution used to provide access to the ingress controller for the cluster.

- dnsmasq—Open-source software providing domain name system (DNS) for the 10G Lab.
- Ceph—Open-source distributed storage, which provides the back-end for the storage in the cluster.

5 Automated Processes with CI/CD

Continuous integration (CI) and continuous delivery (CD) is used to automate CableLabs' software delivery process. GitLab is being used for continuous integration. The CI pipelines build, test, and validate code. The CI pipeline also includes a linter, which helps analyze source code to flag programming errors, stylistic errors, bugs, and suspicious constructs.

OpenShift GitOps (Argo CD) is used for continuous deployment. Argo CD manages the cluster once it is up and running and deploys code changes and services in an automated way following infrastructure as code (IaC) practices. It does this by monitoring Git repositories and deploying any changes detected from the code branch it is monitoring. This process is for cluster maintenance, operators required for cluster management, or required cluster configurations for 5G stacks. No cluster-wide changes are made to the cluster outside of this process. Examples of this include namespace creation, configuring access control settings, service account creation, operator deployments, kernel modifications, and any other cluster settings. Upstream documentation for Argo CD can be found here.

Challenges with this approach include shifting from older models of configuration management, where some things are done manually, and others are done via automation to a single pipeline where everything is done as IaC. It can also involve a significant amount of work to convert tasks done via the GUI to code. In addition, troubleshooting issues can pose significant challenges to those who are unaccustomed to working with CI/CD pipelines and try to make manual configuration changes to fix a problem. However, the benefits of utilizing a tool like Argo CD can significantly reduce the time spent on manual changes made via a GUI. Figure 2 illustrates the CI/CD process for clusters.

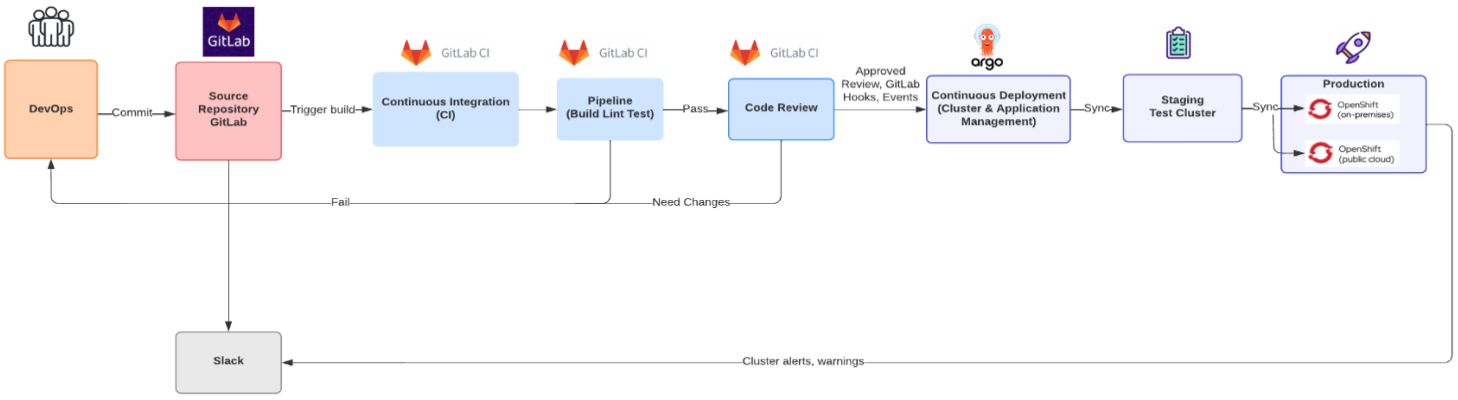

Figure 2: CI/CD Process for Clusters

Figure 3 shows a snippet of applications managed by Argo CD.

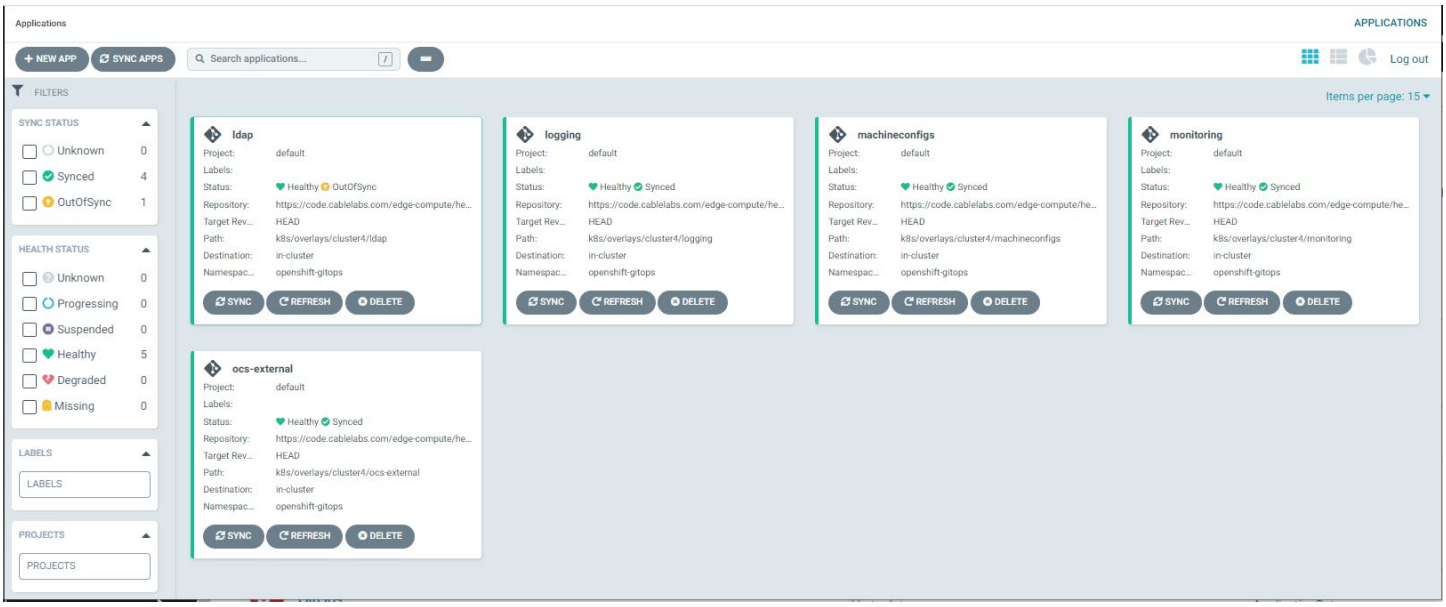

Figure 3: Argo CD Example Deployment

Figure 4 references the deployment of a specific application.

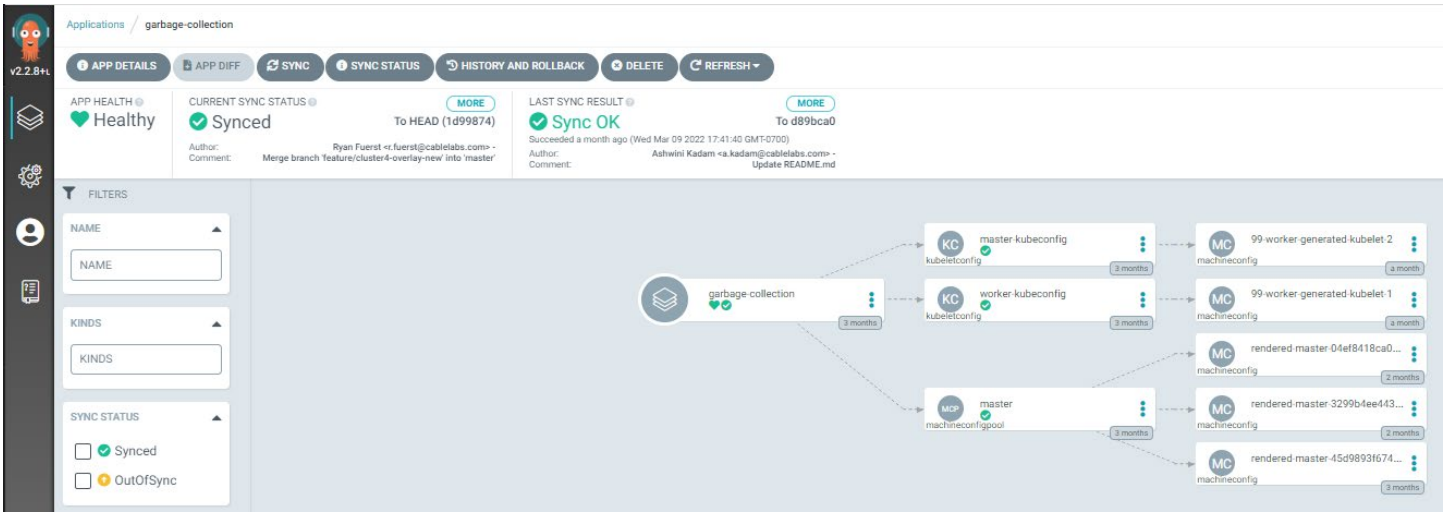

Figure 4: Argo CD Example Application

5.1 Declarative Management with Kustomize

In adopting a GitOps and the IaC approach to cluster configuration, maintenance, and application deployment, a decision was made to standardize on Kustomize to organize and manage the configuration files in GitLab. Kustomize is a tool that can be used to manage Kubernetes objects in a common industry-standard YAML format. Kustomize is used to configure namespaces, operator subscriptions, cluster settings, RBAC controls, etc. This does not preclude vendors or others from using their own deployment code using Helm or other tools. Upstream Kustomize documentation can be found at https://kustomize.io/. Examples of Kustomize scripts used in the 10G Lab can be found at https://github.com/cablelabs/gitops.

The 10G Lab also hosts third-party applications, such as vendor 5G deployments. Vendors can build their applications using Helm or any other standard tool and push their container images to a CableLabs-provided container image registry in GitLab. Vendors are provided access to a jump server to use for their deployments. Required permissions to deploy vendor applications are granted in their respective namespaces on the cluster. Vendor-specific third-party application deployments are not managed via Argo CD.

6 Namespaces

Namespaces provide a mechanism to scope resources in a cluster. To manage multi-vendor deployments, applications are deployed within a namespace that isolates the applications from those in other namespaces.

For vendor deployments, custom vendor prefixed namespaces have been created by CableLabs to isolate and manage applications. Vendors do not have permission to create their own namespaces.

Creating vendor prefixed namespaces for vendor deployments helps to maintain human readability and to distinguish multiple vendor deployments with ease for cluster administrators.

7 Security

7.1 Standard Access Controls

OpenShift clusters provide a default level of access control. Access controls include authentication, authorization, security context constraints (SCC), container security, and direct access to worker nodes. With the default level of standard access control, a user can run non-root processes in a user's containers and can have view permissions to all resources as an authenticated user, access to a restricted SCC, and non-root access to worker nodes, but they do not have access to MacVLAN settings.

For vendor deployments, standard access control is not enough to run deployments and requires modifications per the vendor's needs. Access controls are specific to each vendor and granted accordingly.

7.2 Service Accounts

A service account provides an identity for processes that run in a pod; those processes can be authenticated by the Kubernetes API server.

Service accounts are created to allow components to access the API server without the use of user credentials. Each project has an associated default service account that it uses for API server access. Default service accounts are not granted any RBAC privileges; that is, no roles are bound.

For vendor deployments, each project has an associated custom service account. Vendor-specific service accounts follow the principle of least privilege, granting them only the necessary permissions and privileges. Vendor-specific service accounts are created following the naming convention wherein service accounts are prefixed with vendor names, making it obvious to cluster administrators and easier to maintain. Detailed mappings of service accounts and namespaces are maintained for vendor deployments.

Initially, a custom service account was defined for each vendor. However, consideration should be given to standardizing service accounts around a set of privileges based upon the type of applications being deployed; for example, creation of a service account to support network-intensive applications that need to use DPDK on a network adapter.

7.3 Pod Security Administration

Pod security administration helps to restrict the behavior of pods in a clear, consistent fashion defining different isolation levels for pods. Pod security administration can be enforced in OpenShift via security context constraints (SCCs). Note that SCCs are a RedHat OpenShift-specific implementation of pod security administration. Currently, pod security administration is a beta feature in Kubernetes. More information on the Kubernetes pod security administration can be found here.

SCCs are used to control permissions of pods and define a set of conditions that a pod must meet to be accepted into the system. By default, the cluster has pre-defined SCCs varying from the most restrictive to privileged access.

On creating a new project in the cluster, a default service account is created and assigned the most restrictive SCC. This principle of least privilege is also followed in vendor deployments.

For vendor deployments, if the required permissions match one of the pre-defined SCCs, then those privileges are granted for the project. If an application's needs do not meet with pre-defined SCCs, custom vendor-specific SCCs tailored to applications needs are created and have associated local role bindings with the service account of the project. Detailed mapping of the SCCs and associated local roles and role bindings are maintained.

When creating vendor-specific custom SCCs, there are a few best practices to keep in mind.

- Always prefer restricted SCCs. If the application needs are not met with a restricted SCC, then define a custom SCC based on the restricted SCC only granting the necessary permissions.
- Do not grant privileged SCC access to any applications. This is the least restrictive SCC in the cluster that grants privileges equivalent to a cluster admin.
- While defining custom SCCs, always drop all capabilities exclusively and add only the necessary capabilities. Dropping all the default non-required capabilities and only appending allowedCapabilities and defaultAddCapabilities allows the container to be more secure.
- Minimize container privilege escalation in custom SCCs. The best way to prevent container privilege escalation is to not use privileged containers at all. However, if you are running an application that requires executing with the root user, there is a way to minimize the chances of malicious activity. This is done by user namespace remapping that remaps the user for that specific container to a less privileged user on the host. Essentially, the container views the user as the root, whereas the host does not. Remapping includes assigning a range of UIDs that function within the container (namespace) as normal UIDs but have no privileges on the host.
- Add access control settings in custom SCCs with careful consideration. Prefer setting them as MustRunAs to MustRunAsRange. This enforces a range of user ID values that a container can request. Avoid using RunAsAny. If using RunAsAny cannot be avoided, MustRunAsNonRoot is a more secure option. While specifying MustRunAs and MustRunAsRange, it is advisable to specify the permissible userID, groupID range.
- Configure fsGroup settings in custom SCC with careful consideration. Prefer using MustRunAs. MustRunAs requires at least one range to be specified if not using pre-allocated values.
- Configure supplemental groups in custom SCC with careful consideration. Prefer using MustRunAs. MustRunAs requires at least one range to be specified if not using pre-allocated values.
- Apply seccomp profiles. To add more restrictions, seccomp can be used. Seccomp, secure computing mode, is a Linux kernel feature that can be used to limit the process running in a container to only call a subset of the available system calls. These system calls can be configured by creating a profile that is applied to a container or pod. Seccomp profiles are stored as JSON files on the disk.

Analyze applications to find out the capabilities required. To find out which capabilities the application needs, Red Hat developed a SystemTap script (container_check.stp). Another tool is “capable,” which is part of the BPF Compiler Collection (BCC) tools. It can be installed on RHEL 8 with “dnf install bcc.”

7.4 Role-Based Access Control (RBAC)

Role-based access control is a mechanism to grant authenticated users the right permissions, thereby authorizing them to do what is requested. In short, RBAC is used as an authorization mechanism and is one of the layers of security within the cluster. Within the cluster, RBAC validation is managed by an admission controller that intercepts, monitors, and enforces security access control policies for API server requests made by users. RBAC validation is enforced using roles and role bindings.

By default, authenticated users are granted view permissions while accessing the cluster. A user with default view permissions cannot make modifications but can see most objects in a project. With an authenticated user, if no specific roles or role bindings exist, the authenticated user has basic view permissions on most resources.

7.4.1 Roles and Role Bindings

For RBAC validation, namespace-specific permissions are granted as opposed to cluster-wide permissions. This ensures users are only granted the necessary permissions. With multi-vendor deployments, this helps in managing and ensuring the validity of resources and preventing devastating effects on the cluster.

For vendor-specific deployments, local roles with local role bindings or cluster roles with local role bindings are created according to the application needs in an associated namespace. Vendor-specific local roles and role bindings grant access to various resources required for the application deployment in a specified namespace. A detailed mapping of roles and role bindings is maintained for each vendor.

Vendors may run into errors while accessing resources due to restrictive roles and role bindings. Vendor permission errors can be resolved by granting them access to requested API resources but with a restricted set of verbs to be able to get past the issue with the necessary permissions only.

Using cluster roles and cluster role bindings for vendor deployments is discouraged. Each vendor has access and visibility only to the required namespaces. Visibility beyond the necessary namespaces is not granted. This helps in isolating vendors in multi-vendor deployments.

7.5 Certificates

The cluster setup uses default transport layer security (TLS) certificates to encrypt traffic, authenticate infrastructure components, and authenticate certain users. An internally signed default certificate authority (CA) is used for creating, signing, and configuring cluster certificates. Besides the default CA, a custom CableLabs CA is configured to be used individually by user-facing components. The cert manager is used as an add-on to automate the management and issuance of TLS certificates from the custom CableLabs CA.

7.6 Image Pull Secrets, Image Registry, and Deployment Tokens

CableLabs provides an image registry for transferring a vendor’s containers into the cluster. The image registry is hosted on the CableLabs instance of GitLab. Access tokens were generated for the vendor’s team to use for this purpose. This allows for multiple individuals from the vendor’s organization to use the same deployment token inside automation for uploading or downloading container images for their applications. Best practices would limit these tokens to the least required privileges, including a read token to be used for pulling images and a separate write token when updating or writing images to the registry.

7.7 Identity Provider Integration

To control access to the cluster and configure user authentication, LDAP is used as an identity provider. The cluster's control plane includes a built-in OAuth 2.0 server that determines the user's identity from the configured identity provider and creates an access token. By using LDAP as the identity provider, various LDAP groups can be used in managing users and configuring their respective permissions. LDAP user groups are synced and pruned with cluster records, enabling a seamless sync and management of user groups in one place. Daily cronjobs run to ensure user authentication is up to date. User authentication and permissions to access the cluster vary and are customized per cluster as per needs.

7.8 Limited Access to Cluster Information

Within the cluster, vendor deployments only have visibility and access to resources in their respective namespaces with necessary RBAC privileges enforced for each namespace. Vendors do not have cluster-wide access and cannot retrieve any information or run commands to access cluster information, such as listing node labels. It is discouraged for vendor deployments to have any application implementations tightly coupled to cluster-specific information, such as node labels, since these labels can change over time and are not visible to vendors.

8 Networking

8.1 MacVLAN

The CableLabs clusters support MacVLAN interfaces, allowing pods on a node to communicate with other pods through a physical network interface. The MacVLAN sub-interface is provided with a unique MAC address and can be exposed in the underlay network for external connectivity.

The vendor deployments will be required to provide information about the namespace that will attach to the MacVLAN network, preference for traffic visibility (bridge, passthru, etc.), and IP address configuration for the network attachment.

It is important to ensure that the sub-interface is present on the cluster nodes before configuring an additional network. VLAN interfaces can be created using the NMstate operator. However, installing the container-native virtualization (CNV) operator with an existing NMstate operator blocks the NodeNetworkConfigurationPolicy (NNCP) from being applied to the cluster because of a bug in OpenShift 4.8. This is the case if KubeVirt is installed in the cluster that brings in a conflicting NMstate operator. An alternative method would be to use an nmconnection key file to define the VLAN and push it out to the /etc/NetworkManager/system-connections directory on the nodes using an OCP machine config.

The MacVLAN network can be added as an additional network definition by editing the cluster network operator (CNO) configuration. Since the CNO also manages the network components for the entire cluster, this method poses a higher risk of bringing down the cluster network if a wrong configuration is applied. To avoid disruption to the cluster, a NetworkAttachmentDefinition object can be used to create the MacVLAN network. It is essential to deploy the network attachment in the namespace that will host workloads requiring access to the MacVLAN domain. The network attachment object will also define the properties of the additional network, such as the parent VLAN sub-interface, mode (private/VEPA/bridge/passthru) and the IP address management (IPAM) information.

Upstream documentation for MacVLAN can be found here.

8.2 SR-IOV

The SR-IOV network operator is used to manage the SR-IOV enabled interfaces and network attachments in the cluster. Pre-requisites for SR-IOV deployments are enabling SR-IOV in the global bios of the physical worker nodes assigned for these workloads, as well as enabling SR-IOV at the bios network interface controller (NIC) level and configuring how many virtual function (VF) interfaces will be allowed. Unless all worker nodes in a cluster will be hosting SR-IOV applications, the cluster admin will need to create a node-specific label to attach to worker nodes to distinguish these nodes from normal worker nodes. At the system kernel level, the input-output memory management unit (IOMMU) driver is required, as well as any modules required by the specific NIC hardware vendor present for SR-IOV support.

Vendors will need to provide how many VFs will be required for their applications and if they require DPDK to be enabled. Cluster administrators will take that information and create a SriovNetworkNodePolicy to associate the node interface to the resource granted to the vendor. These policies include any NIC hardware vendor-specific settings. Cluster administrators will take those settings and create a SriovNetwork and assign that network to a vendor namespace for vendor usage.

The operator supports a list of SR-IOV capable devices, and this can be found in the supported-nic-ids config map. For the operator to be able to recognize unsupported NICs, an additional config map is created describing the vendor ID, PF device ID, and the VF device ID of the NIC.

Upstream documentation for the SR-IOV operator can be found here.

8.3 NetworkAttachmentDefinition

NetworkAttachmentDefinition is a custom resource description (CRD) schema specified by the Network Plumbing Working Group to express the intent for attaching pods to one or more logical or physical networks. More information is available at https://github.com/k8snetworkplumbingwg/multi-net-spec.

These definitions are created for the purpose of attaching pods or virtual machines to one or more logical or physical networks. These can include MacVLAN or SR-IOV style networks. The NetworkAttachmentDefinitions requires cluster-admin permissions to create the definitions because this is related to the underlining physical hardware and network settings. As a result, the cluster-admin must work with the end user to define the definition. The cluster-admin will provide the physical layout, while the end user may provide the networking details, such as subnet details.

Deployment of these definitions follows the CableLabs standard GitOps process. They are managed in GitLab, and Argo CD pushes out modifications or changes. Any changes that need to be made follow the standard GitOps approach. These definitions can also have a dependency on machine configs that are configuring the underlying physical hardware.

8.4 OVN-Kubernetes

The default network provider that is being used for cluster networking in the CableLabs environment is OVN-Kubernetes. It uses open virtual network (OVN) to manage network traffic flows, implements Kubernetes network policy for ingress and egress rules, and uses the Generic Network Virtualization Encapsulation (GENEVE) Protocol rather than VXLAN to create an overlay network between nodes. One important feature of OVN-Kubernetes is that it supports IPv6 on bare metal servers.

OVN-Kubernetes was preferred as the default container network interface (CNI) over OpenShift SDN by the vendors. It supports the version of Kubernetes deployed in CableLabs' clusters and meets the needs of most of the applications, such as hardware offloading for DPDK.

8.5 External Access to Cluster Services

Within Kubernetes, there are multiple ways to expose services outside of the cluster. NodePort is a primitive way to get external traffic directly to a service by opening a specific port on all the nodes, and any traffic that is sent to this port is forwarded to the service. This method has some limitations, including the need to track which nodes have pods with exposed ports, it only exposes one service per port, it does not allow the use of standard service ports like port 80, port 443, etc. (only the port range 30,000 to 32,767 is allowed), and it requires the opening of firewall rules to allow access to assignable ports. Because of these limitations, reserving ports on nodes is strongly discouraged.

To make the cluster more flexible and to reduce complexity, the use of NodePort is not allowed in the CableLabs cluster. This removes the dependency on hard-coded ports being assigned to services and eliminates the possibility of ports assigned to one vendor's services from conflicting with another vendor's services.

For moving traffic from outside the cluster to a running application instance, CableLabs is utilizing the MetalLB load balancer. MetalLB will manage the pool of IP addresses it can distribute and is well-suited to support Border Gateway Protocol (BGP) for more complex networking scenarios, particularly when multiple Kubernetes clusters are involved in the environment.

In addition, CableLabs uses HAProxy as a load balancer to access the ingress controller hosted on Kubernetes. The ingress controller manages and exposes the external HTTP/HTTPS routes to the cluster services. An ingress controller acts as an API gateway and manages ingress traffic in Kubernetes through an ingress resource that defines the connectivity rules.

8.6 Precision Time Protocol (PTP) Sync

Precision Time Protocol provides a method to deliver time synchronization across devices in a Local Area Network (LAN) with sub-microsecond accuracy. PTP messages can be carried over existing data network connections rather than having a dedicated connection for the purpose of device synchronization. They are untagged when enabled on a trunk port; therefore, it is required to have a native VLAN configured on the switch ports to carry the traffic. PTP is propagated over a multicast network from a primary/grandmaster clock to all the client clocks. PTP messages can be transported over either IPv4, IPv6, or Ethernet networks, and the boundary clocks can be configured to use a specific transport type for PTP.

Time, frequency, and phase synchronization need to be accurately distributed across the 5G radio unit (RU) and distributed unit (DU) for a smooth radio access network (RAN) operation without any impacts to performance (throughput, handover success rate, etc.). To meet this requirement, vendor 5G deployments used the Viavi MTS-5800 as the single source to deliver PTP time/phase sync using a G.8275.1 profile, Synchronous Ethernet (SyncE), Time of Day (ToD)/1PPS, and external sync at 10 MHz through a Fibrolan switch. The initial setup involving Arista 7170 as the boundary clock had to be removed because this switch does not support PTP profiles other than IEEE 1588.

Port mirroring is a useful tool to capture and inspect timestamps and flags on the PTP packets at different points in the network to verify the sync operations.

9 Service Mesh

Service meshes play a critical role in managing many microservices through several key capabilities, including load balancing, access control, and version management.

Load balancing enables microservices to scale based upon demand and route around failures.

Access control in the service mesh provides a layer of security to determine which namespaces and applications are permitted to access any given microservice.

Version management enables multiple versions of the same microservice, enabling a new version of the microservice to be rolled out without impacting applications that rely on the current version of the service.

Service meshes are not a core component to Kubernetes—there are multiple versions and flavors available to install. Service meshes can also be combined with other applications to provide a complete solution. For instance, Istio was the predominate service mesh when many applications were written. Even with multiple vendors supporting the same version of Istio, they are not always compatible with tools used for implementing access control. There can only be a single Istio instance per cluster, which means that all applications must share the same install.

For the CableLabs environment, the OCP service mesh, which is based upon Istio, was leveraged. This presented a challenge when attempting to deploy multiple 5G core vendor stacks into a single shared cluster. Some of the 5G core vendors required different versions of Istio. Because it is not possible to install two distinct versions of Istio into the same cluster where the network stacks are co-located, a single version of the OCP service mesh was declared to be supported in the cluster, and vendors would have to support that version. In this case, ServiceMesh 2.x was installed, and anything ServiceMesh 2.x maintains API compatibility even when the underlying Istio changes. However, some vendors were asking for their own version of Istio, which would be problematic in a shared cluster. As a temporary solution, the vendor that could not support the OCP service mesh was allowed to be deployed in a separate cluster that was not running any other 5G workloads.

10 Hardware Dependencies

Hardware dependency issues were encountered when trying to design a shared cluster for 5G deployments. These workloads were only validated against Intel chipsets. Since vendor applications were not agnostic to either Intel or AMD CPUs, worker nodes with Intel CPUs were exclusively used for 5G deployments. Another dependency that was encountered involved network interface cards (NICs). The 10G Lab utilized Mellanox NIC cards in the clusters. Vendors may not have tested their deployments against this brand of NIC; therefore, their SR-IOV-enabled workloads required validation against this NIC. As a prerequisite to deploying 5G workloads into a new Kubernetes environment, it is recommended that the vendors test against the type of NICs being utilized to ensure proper operation of their workloads.

11 Kernel Modifications

Some workloads require specific kernel modules to be loaded or kernel tunes to be applied that are not present by default. The specific parameters that were tuned include

- HugePages—support memory pages greater than 4 KB that can be required by some applications;
- Process ID (PID) Limits—Linux Process IDs;
- Input-output memory management unit—required to expose virtual functions of SR-IOV;
- Stream Control Transmission Protocol (SCTP)—required by some vendors to run their applications; and
- Nvidia driver container—required for GPU workloads such as Machine learning or AI.

CableLabs used the performance-add-on operator to enable IOMMU, which is required by SR-IOV and to configure HugePages. OCP MachineConfigs were used to enable SR-IOV, SCTP, and MacVLAN on the underlining hardware. A driver-container toolkit was deployed to enable GPU workloads using the Nvidia operator.

11.1 HugePages

Configuring HugePages on nodes makes it possible for the operating system to support memory pages greater than the default of 4 KB, which can improve system performance by reducing the amount of system resources required to access page table entries.

5G core vendors require an increase to the HugePages configuration on the order of 1 G, particularly for the operation of the 5G user plane function (UPF) component.

The default setting for nodes in the cluster running an SR-IOV-enabled workload is 1 G. SR-IOV with DPDK will only work with a sufficient HugePages configuration allocated to the pod.

11.2 PID Limits

Kubernetes allows the user to limit the number of process IDs that a pod can use. OpenShift has a default limit of 1024. The default in a typical Linux install is 32,768. A value of 2048 was configured as the default after consulting with 5G vendors on what would be best for their workloads. By establishing a PID limit per pod, this will help prevent one pod from overwhelming the resources on a node by spawning many processes and potentially negatively impacting other pods on the node.

11.3 IOMMU

IOMMU is required for SR-IOV-enabled workloads. IOMMU translates between I/O virtual addresses (IOVA) and physical memory addresses. It is not enabled by default in the Linux kernel.

11.4 SCTP

SCTP is a reliable message-based protocol that runs on top of an IP network. It may be required by some 5G vendor applications. Enabling it requires changes to the underlying host system.

11.5 GPU Driver Container

Some workloads are best run on GPU hardware versus CPU hardware. To enable these workloads, CableLabs installed a driver-container toolkit that includes the required kernel modules that need to be run on the underlying host system that have GPUs installed.

Upstream documentation for Nvidia GPU can be found here.

12 Onboarding Questionnaire

When onboarding a new vendor to the platform, the following questions were asked to gather the required information to create their project/namespaces.

1. Name of the project/namespaces
    a. Multiple namespaces may be required; the following are examples of a production namespace and a staging namespace.
        i. foo-prod
        ii. foo-staging
2. Access
    a. User access
        i. List of required users. These should be added to a backend auth system group; e.g, LDAP versus a local account. These users should be split across different groups based on required access levels.

   b. Service accounts

      i. Any additional service accounts outside the defaults they would like created with appropriate permissions

   c. Additional capabilities required for application deployment beyond the defaults provided. Some deployments require specific access levels not enabled by default. Permissions will need to be updated to include any special requests.

3. Resource needs

   a. Storage, CPU, memory, and network capacity requirements to ensure the cluster can support the vendor's application

4. Network specific features

   a. SR-IOV or MacVLAN require cluster administrator permissions to set up. Cluster admins will need to know if these types of services are required so they can enable them on the cluster.

5. Additional operators beyond those already installed

If a deployment needs a software operator not already installed, the vendor will need to notify the cluster administrators of those requirements.

Conclusion

Working with 5G network stack vendors to deploy their systems into shared infrastructure has revealed many technical considerations that need to be addressed. Focusing on enabling the shared infrastructure to support low latency to meet the network performance requirements of 5G is critical. Ensuring that security measures are taken to logically separate 5G network stack workloads from one another, as well as from other workloads running on the shared infrastructure, requires some planning to establish proper access controls and resource utilization limitations. Enabling automation to build or rebuild new clusters and maintain the services and workloads running on those clusters is important for manageability and reliability.

The topics discussed in this paper lay the foundation for platform convergence where multiple network stacks can coincide on the same infrastructure, which then paves the way for full convergence to take place. The paper focuses primarily on the deployment and configuration of 5G network stacks on shared infrastructure. It is to be expected that when the network stacks become operational, other considerations for managing their co-location will likely surface, and additional lessons will be learned.